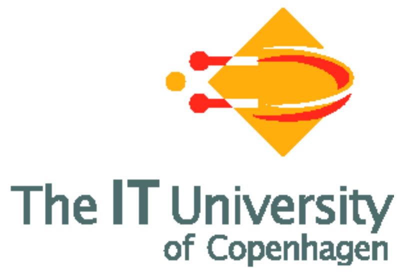



# Bigraphical Languages and their Simulation

Espen Højsgaard

A PhD Dissertation
Presented to the Faculty of the IT University of Copenhagen
in Partial Fulfillment of the Requirements of the PhD Degree

IT University of Copenhagen December 2011

**Abstract**

We study bigraphs as a foundation for practical formal languages and the problem of simulating such bigraphical languages. The theory of bigraphs is a foundational, graphical model of concurrent systems focusing on mobility and connectivity. It is a meta-model in the sense that it is parametrized over a *signature* and a set of *reaction rules* which determine the syntax and dynamic semantics, respectively. This allows for rather direct models and, together with a natural yet formal graphical notation and an elegant theory of behavioral equivalence, this makes bigraphs an enticing foundation for practical formal languages. However, the theory of bigraphs is still young. While direct models of many process calculi have been constructed, it is unclear how suitable bigraphs are for more practical formal languages. Also, the generality of bigraphs comes at a price of complexity in the theory and simulation of bigraphical models is non-trivial. A key problem is that of *matching*: deciding if and how a reaction rule applies to a bigraph. In this dissertation, we study bigraphs and their simulation for two types of practical formal languages: programming languages and languages for cell biology.

First, we study programming languages and *binding* bigraphs, a variant of bigraphs with a facility for modeling the binders found in most programming languages. Building on an existing term language and inductive characterization of matching we construct a provably correct matching algorithm. We implement the term language and matching algorithm resulting in the BPL Tool, a first tool for binding bigraphs, which provides facilities for modeling, simulation, and visualization. We then employ binding bigraphs and the BPL Tool to formalize a subset of WS-BPEL, a commercial programming language for implementing business processes. We also propose and formalize an extension to WS-BPEL which supports mobile processes and process management. While demonstrating the feasibility of using bigraphs as a foundation for programming languages, our work reveals an inconvenience in the formulation of binding bigraphs, exposes the need for higher-order reaction rules, data types and (practical) sortings, and demonstrates that the BPL Tool is too inefficient for simulating such a language. Finally, as an aside, we identify a core subset of WS-BPEL and construct an idempotent transformation from WS-BPEL into the core subset, thereby showing that a formalization need only cover the core subset to be complete.

Next, we study languages for cell biology and *stochastic* bigraphs, an extension to bigraphs that enables modeling and analysis of stochastic behavior which is useful in cell biology. We generalize an efficient and scalable stochastic simulation algorithm for the $\kappa$-calculus to bigraphs. For this purpose, we develop a number of theories for (stochastic) bigraphs: (i) a formulation of the theory that is amenable to implementation, (ii) embeddings, an alternative formulation of matches suitable for implementation, (iii) edit scripts, an alternative to reaction rules with a natural and fine-grained notion of modification, (iv) anchored matching, a localized matching algorithm, and (v) a notion and analysis of causality at the level of rules. Parts (i)-(iii) have been developed in full while parts (iv)-(v) are outlined in detail. Parts (i)-(iv) have been implemented in a prototype. Finally, we develop a bigraphical language for protein-protein interaction with dynamic compartments. Our approach differs from similar previous works in a number of respects. First, we elide the bigraphical underpinnings to obtain a simpler and more accessible presentation in the style of process calculi. In particular, the development is incremental, adding only the complexity necessary for each feature. Second, we give a graphical notation which corresponds to a subset of bigraphs but is more suitable for the domain. Third, our approach includes a novel mechanism for handling connected components, which is necessary to model diffusion of e.g., protein-complexes. Our work suggests that two refinements of stochastic bigraphs would be convenient: connected components should be easily identifiable, and matching should be restricted to certain local contexts.

# Acknowledgements

It would not have been possible to write this dissertation if not for all the kind and generous people who, in each their own way, have helped me.

I wish to thank my supervisor Thomas Hildebrandt and my co-supervisors Arne John Glenstrup and Henning Niss for introducing me to the wonderful world of research and for their guidance through my quest. Indeed, all the members of the Programming, Logics, and Semantics group at ITU have made the last 4 years a pleasure and adventure – the following PLS'ers deserve special mention (in order of acquaintance): Troels Damgaard, Søren Debois, Mikkel Bundgaard, Jacob Thamsborg, Rasmus Møgelberg, Hugo Lopez, Tim Hallwyl, Alexandre Buisse, and Gian Perrone.

As part of my PhD I visited the School of Informatics at the University of Edinburgh for about 4 months at the end of 2009. I wish to thank Stephen Gilmore for being the perfect host. I also wish to thank Michael D. Pedersen who, through his friendship and surfing lessons, contributed immensely to the great experience I had in Edinburgh. I also visited Laboratoire PPS at Université Paris Diderot for 4 months in the beginning of 2010. Jean Krivine, who hosted me, went far above and beyond his duties, offering friendship, guidance, inspiration, entertainment and help throughout my stay. For this I am eternally grateful.

My deep-felt thanks also go to my parents, my brother, and the rest of my family who have always supported me through life. My son Alfred in particular has been my tethering to the real world and have pulled me down to earth when the world of bigraphs was luring me away, threatening to engulf me.

Last but not least – quite the contrary actually – I wish to thank my wife Lykke, who always supports me and suffers my absentmindedness and other flaws of character without complaint. You are my ray of sunshine!

*Espen Højsgaard*
*Copenhagen, December 2011*

# Contents

*Abstract* *page* i

*Acknowledgements* iii

*Contents* v

## Part I: Overview 1

**1 Introduction** **3**
1.1 Papers . . . 4
1.2 Background . . . 5

**2 Summary** **19**
2.1 A Tool for Bigraphical Programming Languages (Part II) . . . 20
2.2 Bigraphical Semantics for Business Processes (Part III) . . . 24
2.3 Scalable Simulation of Stochastic Bigraphs (Part IV) . . . 31
2.4 A Bigraphical Language for Cell Biology (Part V) . . . 38
2.5 Conclusion . . . 42
2.6 Future Work . . . 43

**Bibliography** **45**

## Part II: A Tool for Bigraphical Programming Languages 53

**3 An Implementation of Bigraph Matching** **55**
*Arne J. Glenstrup, Troels C. Damgaard, Lars Birkedal, and Espen Højsgaard*
3.1 Introduction . . . 55
3.2 Bigraphs and Reactive Systems . . . 56
3.3 Inferring Matches Using a Graph Representation . . . 63
3.4 From Graph Matching to Term Matching . . . 65
3.5 Normal Inferences . . . 65
3.6 Bigraph Matching Algorithm . . . 69
3.7 Nondeterminism . . . 71
3.8 Tool Implementation and Example Modelling . . . 72
3.9 Conclusion and Future Work . . . 74
3.10 Bibliography . . . 77
3.A Auxiliary Technologies Details . . . 79

**4 The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems 85**
*Espen Højsgaard and Arne J. Glenstrup*
4.1 Introduction . . . 85
4.2 Installation . . . 87
4.3 Example: Polyadic $\pi$ and Mobile Phones . . . 88
4.4 Reference . . . 89
4.5 Conclusions and Future Work . . . 103
4.6 Bibliography . . . 104

## Part III: Bigraphical Semantics for Business Processes 107

**5 Formalizing WS-BPEL and Higher Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool 109**
*Mikkel Bundgaard, Arne J. Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss*
5.1 Introduction . . . 110
5.2 Binding Bigraphs and BPL Tool . . . 113
5.3 Formalizing WS-BPEL in the BPL Tool . . . 116
5.4 Motivating HomeBPEL . . . 128
5.5 Formalizing HomeBPEL . . . 130
5.6 Conclusion and Future Work . . . 140
5.7 Bibliography . . . 141

**6 Core BPEL: Semantic Clarification of WS-BPEL 2.0 through Syntactic Simplification using XSL Transformations 145**
*Tim Hallwyl and Espen Højsgaard*
6.1 Introduction . . . 146
6.2 Transformation Considerations . . . 148
6.3 Default Values and Elements . . . 150
6.4 Standard Attributes and Elements . . . 155
6.5 Desugaring Constructs . . . 157
6.6 Extensions . . . 178
6.7 Combining the Transformations . . . 180
6.8 Conclusions . . . 180
6.9 Bibliography . . . 182
6.A WS-BPEL vs. Core BPEL Syntax Summaries . . . 184
6.B XML Schema for Core BPEL . . . 193
6.C Transformation Example . . . 200
6.D XSLT Transformations . . . 203

## Part IV: Scalable Simulation of Stochastic Bigraphs 229

**7 Towards Scalable Simulation of Stochastic Bigraphs 231**
*Espen Højsgaard and Jean Krivine*
7.1 Introduction . . . 232
7.2 Background . . . 235
7.3 The Simulation Algorithm . . . 248
7.4 Stochastic Parametric Reactive Systems . . . 251
7.5 Bigraph Embeddings . . . 264

7.6 Bigraph Edit Scripts . . . 276
7.7 Rule Activation and Inhibition . . . 287
7.8 Anchored Matching . . . 295
7.9 Conclusions and Future Work . . . 298
7.10 Bibliography . . . 299
7.A Proofs . . . 301

**Part V: A Bigraphical Language for Cell Biology 355**

**8 Formal Cellular Machinery 357**
*Troels C. Damgaard, Espen Højsgaard, and Jean Krivine*

8.1 Introduction . . . 357
8.2 $\mathcal{C}_0$: forming molecules . . . 358
8.3 $\mathcal{C}_1$: naming molecules . . . 361
8.4 $\mathcal{C}_2$: placing molecules . . . 362
8.5 $\mathcal{C}_3$: moving molecules . . . 366
8.6 Conclusion . . . 369
8.7 Bibliography . . . 369
8.A Retrieving the $\kappa$-calculus. . . . 370
8.B Proof of the soundness Theorem . . . 372
8.C Proof of the completeness Theorem . . . 376

# Part I

# Overview

# Chapter 1

# Introduction

This dissertation comprises six papers with the common theme of testing the thesis that bigraphs may be used as an executable foundation for realistic formal languages.

The theory of bigraphs and bigraphical reactive systems is a foundational, graphical model of concurrent systems focusing on mobility and connectivity, developed by Milner and collaborators [Mil09]. It arose as a generalization of process calculi and provides a unifying framework for modeling systems of mobile and communicating agents. In particular, the $\pi$-calculus [MPW92] and mobile ambients [CG00] have been represented as instances of bigraphical reactive systems [JM04].

It was quickly realized that bigraphs had the potential to serve as more than a unifying framework for such abstract models of concurrency. The theory of bigraphs is a meta-model in the sense that it is parametrized over a *signature* and a set of *reaction rules* which determine the syntax and dynamic semantics, respectively. This provides a flexibility which promises rather direct models of many types of formal languages. Furthermore, bigraphs have a natural yet formal graphical notation and an elegant theory of behavioral equivalence. Altogether, this makes bigraphs an enticing foundation for formal languages.

This lead to the *bigraphical programming languages (BPL)* project in the PLS group at the IT University of Copenhagen (ITU) which researched the design and implementation of programming languages based on the theory of bigraphs, and the effort was continued in the *computer supported mobile adaptive business processes (CosmoBiz)* project. Along the way, interest also turned towards formal languages for cell biology for which bigraphs also seem (surprisingly) suitable.

In this dissertation, which is a product of the BPL and CosmoBiz projects, we research bigraphs as an executable foundation for realistic formal languages. For two different kinds of languages, programming languages and languages for cell biology, we investigate how appropriate variants of bigraphs may be implemented and we develop representative, realistic bigraphical languages. Through this approach, we both advance and evaluate the theory of bigraphs: advance because we have to expand the theory in order to implement bigraphs, and evaluate because we apply the theory to realistic languages.

### Contributions in this Dissertation

We first address the challenge of implementing *binding* bigraphs, a variant of bigraphs with a facility for modeling the binders found in most programming languages. The key challenge is the *matching problem*, the problem of deciding if (and where) a reaction rule applies. Based on the sound and complete inductive characterization of matching by Birkedal et al. [BDGM07] and the term language and normal form theorem by Damgaard and Birkedal [DB06], both for binding bigraphs, we have developed, and proven correct, a term-based matching algorithm. We have

implemented the matching algorithm in a first tool for binding bigraphs, the BPL Tool, which provides facilities for modeling, simulation, and visualization.

We have employed binding bigraphs and the BPL Tool to give a very direct formalization of a subset of WS-BPEL, a commercial programming language for implementing business processes. Furthermore, we have proposed and formalized an extension to the WS-BPEL language, where we added mobile, embedded processes and manipulation of these. As a step towards formalizing the full WS-BPEL language, we have identified a core subset the language and constructed an idempotent transformation from WS-BPEL into the core subset.

Next, we address the challenge of implementing *stochastic bigraphs*, a variant of bigraphs where reaction rules are equipped with a stochastic behavior which is essential in the modeling of cellular biology. As such models are often large, scalability is of the essence. Therefore, we have taken an efficient and scalable stochastic simulation algorithm for the $\kappa$-calculus as our starting point and generalized it to bigraphs. This required the development of a number of theories for (stochastic) bigraphs:

(i) stochastic parametric reactive systems, a formulation of the dynamic theory that is amenable to implementation,

(ii) bigraph embeddings, an alternative formulation of matches more suitable for implementation,

(iii) bigraph edit scripts, a fine-grained alternative to reaction rules with a natural notion of modification,

(iv) anchored matching, a localized matching algorithm, and

(v) a notion and analysis of causality at the level of rules.

Parts (i)–(iii) have been developed in full while parts (iv)–(v) are outlined in detail but some results have yet to be proven. Parts (i)–(iv) have been implemented in a prototype.

Finally, we have developed a bigraphical language for cell biology: a language for protein-protein interaction with dynamic compartments. This has been done before, but our approach differs in a number of respects. First, we have elided the bigraphical underpinnings to obtain a simpler and more accessible presentation in the style of process calculi. This allowed us to develop the language incrementally, only adding the complexity necessary for each feature. Second, we have given a graphical notation that corresponds to a subset of bigraphs but is more suitable for the domain. Third, our approach includes a novel mechanism for handling connected components, which is necessary to model diffusion of e.g., protein-complexes.

#### Structure of the Dissertation

This dissertation is organized as follows. Parts II–V contain six papers which are the written products of four lines of research we have pursued during my PhD. Chapter 2 summarizes and discusses each of these works individually and then concludes with a discussion of the works as a whole. The remainder of this chapter lists the papers and technical reports I have co-authored as part of my PhD, and provides the background necessary to appreciate Chapter 2.

## 1.1 Papers

In order of appearance, the following papers are included in this dissertation:

[GDBH10] Arne J. Glenstrup, Troels C. Damgaard, Lars Birkedal, and Espen Højsgaard. *An Implementation of Bigraph Matching.* Technical Report TR-2010-135, IT University of Copenhagen. December 2010.

[HG11] Espen Højsgaard and Arne J. Glenstrup. *The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems.* Technical Report TR-2011-145, IT University of Copenhagen. October 2011.

[BGH+08b] Mikkel Bundgaard, Arne J. Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. *Formalizing WS-BPEL and Higher-Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool.* Technical Report TR-2008-103, IT University of Copenhagen. May 2008.

[HH11] Tim Hallwyl and Espen Højsgaard. *Core BPEL: Semantic Clarification of WS-BPEL 2.0 through Syntactic Simplification using XSL Transformations.* Technical Report TR-2011-138, IT University of Copenhagen. March 2011.

[HK11] Espen Højsgaard and Jean Krivine. *Towards Scalable Simulation of Stochastic Bigraphs.* Technical Report TR-2011-148, IT University of Copenhagen. December 2011.

[DHK11] Troels C. Damgaard, Espen Højsgaard, and Jean Krivine. *Formal Cellular Machinery.* Proceedings of SASB 2011, the Second International Workshop on Static Analysis and Systems Biology. September 2011. Keynote talk. (to appear)

The papers appear in their original form, except for minor typographical changes and adaptation of their layout to fit into this dissertation.

The following papers have been omitted from this dissertation:

[BGH+08a] Mikkel Bundgaard, Arne J. Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. *Formalizing Higher-Order Mobile Embedded Business Processes with Binding Bigraphs.* Proceedings of the 10th International Conference on Coordination Models and Languages (COORDINATION'08). June 2008.

[BHH08] Mikkel Bundgaard, Thomas Hildebrandt, and Espen Højsgaard. *Seamlessly Distributed & Mobile Workflow - or: The right processes at the right places* Proceedings of the 1st Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES'08). June 2008.

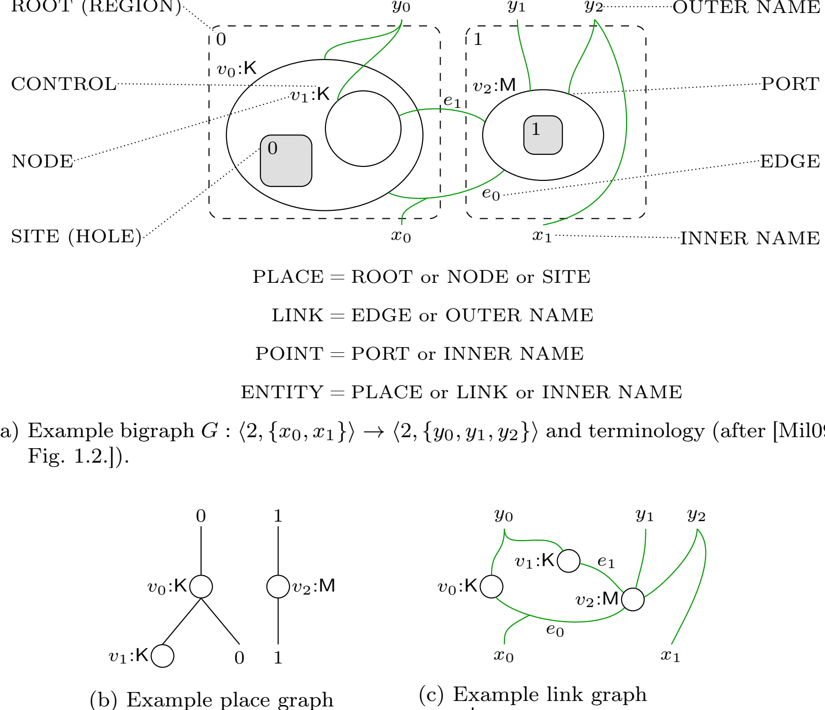
[HH12] Espen Højsgaard and Tim Hallwyl. *Core BPEL: Syntactic Simplification of WS-BPEL 2.0.* Proceedings of the 27th ACM Symposium on Applied Computing (SAC'12). March 2012. (to appear)

The paper [BGH+08a] is the conference version of the second half of the technical report [BGH+08b] and similarly the paper [HH12] is the conference version of the technical report [HH11]. [BHH08] is a position paper.

## 1.2 Background

In this section, we introduce the theory of bigraphs in sufficient detail to allow a reader who is familiar with process calculi to appreciate the summary and discussion of our works in Chapter 2. For a more thorough and formal treatment, please consult Milner's recent book: *The Space and Motion of Communicating Agents* [Mil09]. The bigraph-savvy reader should skim this section to get acquainted with the presentation style used in this dissertation and, more importantly, because we place our works in the context of some concurrent developments on the theory of bigraphs.

The theory of *bigraphs* and *bigraphical reactive systems* (BRSs) is a recent theory developed by Milner and collaborators as a graphical model of mobile and ubiquitous computing [JM04]. It focuses on two key aspects of such systems, connectivity and locality, and it admits encodings of




(a) Example bigraph $G : \langle 2, \{x_0, x_1\}\rangle \to \langle 2, \{y_0, y_1, y_2\}\rangle$ and terminology (after [Mil09, Fig. 1.2.]).

(b) Example place graph $G^{\mathsf{P}} : 2 \to 2$.

(c) Example link graph $G^{\mathsf{L}} : \{x_0, x_1\} \to \{y_0, y_1, y_2\}$.

Figure 1.1: Example bigraph and it constituent place and link graphs.

the archetypal process calculi for each of these two aspects, the $\pi$-calculus [MPW92] and mobile ambients [CG00] [JM04]. However, whereas process calculi usually have fixed syntax and semantics, bigraphs and BRSs are a meta-model that can be instantiated with a particular syntax, specified by a *signature*, and semantics, specified by a set of *reaction rules*; we shall elaborate on this below. For the sake of brevity, we call it the *theory of bigraphs* or simply *bigraphs*.

Our presentation of the theory of bigraphs is organized as follows. First, we define bigraphs, that are static structures in themselves. Next, we define BRSs, which provide the dynamic aspect of the theory. Finally, we discuss some developments of the theory which are particularly relevant to the presented works.

### 1.2.1 Bigraphs: the Static Structure

A bigraph consists of two orthogonal structures over the same set of nodes: the *place graph* which models locality, and the *link graph* which models connectivity. The place graph is an unordered forest and the link graph is a hypergraph, and both are finite. Figure 1.1a depicts an example bigraph $G$ and Figures 1.1b and 1.1c depict the corresponding place and link graphs, respectively. In the following, we exemplify concepts by listing the corresponding elements of this example in parentheses. Ovals represent nodes ($v_0, v_1, v_2$), and nodes are assigned *controls* ($\mathsf{K}, \mathsf{M}$) which are taken from a *signature*: a set of controls $\mathcal{K}$ and a map $ar : \mathcal{K} \to \mathbb{N}$ that assigns an *arity* to each control ($ar(K) = 2, ar(M) = 4$). The arity of a control specifies the number of *ports* (connection points in the link graph) a node with that signature has. We often omit the arity map, writing simply $\mathcal{K}$ for the signature, and when defining a signature, we give the arities inline; e.g., in the example the signature is $\mathcal{K} = \{K : 2, M : 4\}$.

The dashed rectangles are called *regions* or *roots* $(0, 1)$: they are the roots of the place graph

forest and they are identified by consecutive numbers starting from 0. We shall usually let the position of the root determine its identity, increasing from left to right. Roots model unspecified, possibly distinct, locations. The number of roots in a bigraph is called its *width*.

The gray rectangles are the dual of roots: called *sites* or *holes* $(0, 1)$, they model places where other bigraphs may be inserted, as we shall see when we define composition of bigraphs. As with roots, sites are identified by consecutive numbers starting from 0, but we always include the site identifiers if there is more than one. Collectively we call nodes, roots, and sites *places*.

The *outer names* $(y_0, y_1, y_2)$, written at the top of the bigraph, model connection points that a context must provide. Conversely, the *inner names* $(x_0, x_1)$, written at the bottom of the bigraph, are connection points that the bigraph provides when used as a context. Together with outer names, *edges* $(e_0, e_1)$ are *links* and they model connectivity. Links may connect multiple ports and/or inner names, collectively called *points*.

Collectively, we call places, links, and inner names *entities*. Note that ports are not individual entities, as they are a part of a node.

Bigraphs are assigned *interfaces*: an outer interface, or *outer face*, which summarizes the number of roots $n$ and the set of outer names $Y$, written $\langle n, Y\rangle$, and an inner interface, or *inner face*, which summarizes the number of sites $m$ and inner names $X$. We use $I, J, \ldots$ to range over interfaces and we write $G : I \to J$ when a bigraph $G$ has inner face $I$ and outer face $J$. The interfaces of the bigraph in Figure 1.1a are shown in the caption.

### Notation and terminology

For the sake of conciseness, we shall often make use of the following terminology and notation:

**discrete:**
A bigraph is discrete iff the link graph is a bijection between outer names and points, i.e., it is *open*, no link is *idle*, and no two points are siblings.

**idle:**
A link is *idle* iff no points are connected to it. A place is *idle* iff it has no children.

**interfaces:**
Interfaces are abbreviated as follows:

$$\begin{aligned} n &\stackrel{\text{def}}{=} \langle n, \emptyset\rangle \\ Y &\stackrel{\text{def}}{=} \langle 0, Y\rangle \\ \epsilon &\stackrel{\text{def}}{=} \langle 0, \emptyset\rangle \\ \langle Y\rangle &\stackrel{\text{def}}{=} \langle 1, Y\rangle \ . \end{aligned}$$

**ground, agent:**
A bigraph $G : \epsilon \to J$ is an *agent* or *ground*. We often omit the inner face of ground bigraphs, writing simply $G : J$.

**lean:**
A bigraph is *lean* iff no edges are idle.

**open, closed:**
A link is *open* iff it is an outer name. Conversely, a link is *closed* iff it is an edge. A bigraph is *open* iff all its links are open.

**prime:**
A bigraph $G : m \to \langle Y\rangle$ is prime.

**wide:**
A bigraph $G : I \to \langle n, Y\rangle$ is *wide* iff $n > 1$.

### Concrete vs Abstract Bigraphs

What we have discussed so far, where nodes and edges have identities, are usually called *concrete bigraphs*. We call the set of identifiers in a concrete bigraph its *support*. For many purposes support provides crucial structure, e.g., to derive labeled transition systems or for stochastic semantics, as we shall see later.

However, for other purposes identities are uninteresting, e.g., when modeling process calculi, so we often abstract away from them, thereby obtaining *abstract bigraphs*. More formally, abstract bigraphs are usually defined as equivalence classes of concrete bigraphs: Let $\simeq$ denote *support*

*equivalence* of bigraphs, i.e., $F \doteq G$ iff $F$ and $G$ differ only by a bijection on their support, and let $\dot{\approx}$ denote *lean support equivalence*, the extension of $\doteq$ that disregards idle edges. Abstract bigraphs are then defined as the equivalence classes of $\dot{\approx}$.

### Bigraph Operations

Concrete bigraphs admit a composition operation, $G \circ F$, whenever the supports are disjoint and the interfaces are compatible, i.e., $F : I \to J$ and $G : J \to K$. Figure 1.2 illustrates the composition of two example bigraphs. Intuitively, composition plugs the roots of $F$ into the holes of $G$ and splices the outer names of $F$ with the links of $G$ that have matching inner names. Composition lifts to abstract bigraphs by simply composing two concrete representatives with disjoint support and then abstracting the result. It should be clear that we can construct an identity bigraph for any interface $I$, $\mathsf{id}_I$, satisfying $\mathsf{id}_I \circ F = F$ and $G \circ \mathsf{id}_I$ for all bigraphs $F : J \to I$ and $G : I \to K$. Concrete bigraphs also admit a tensor operation, $F \otimes G$, which places the roots of $G$ next to the roots of $F$. Figure 1.4 illustrates the tensor product of two example bigraphs. Tensor operation is sometimes called horizontal composition, and, in contrast, $\circ$ is referred to as vertical composition. The tensor product is defined if the supports are disjoint and the outer, respectively inner, names are disjoint. Tensor also lifts to abstract bigraphs by simply tensoring two concrete representatives with disjoint support and then abstracting the result.

Vertical and horizontal composition are usually taken as primitive operations and other operations are then derived from these. Three oft-used such derived operations are

$F.G$**:** Called *nesting* or *dotting*, this operation is identical to composition, except $F$ is not allowed to have inner names and the outer names of $G$ become outer names of $F.G$, joining the links of any shared outer names of $F$ and $G$. Figure 1.3 illustrates the nesting of two example bigraphs.

$F \parallel G$**:** Called *parallel product*, this operation is identical to the tensor product, except that it dispenses with the requirement that outer names must be disjoint, and instead takes shared names to mean that the links should be shared. Figure 1.5 illustrates the parallel product of two example bigraphs.

$F \mid G$**:** Called *prime parallel product*, this operation is as the parallel product but in addition it merges the roots of $F$ and $G$. Figure 1.6 illustrates the prime parallel product of two example bigraphs.

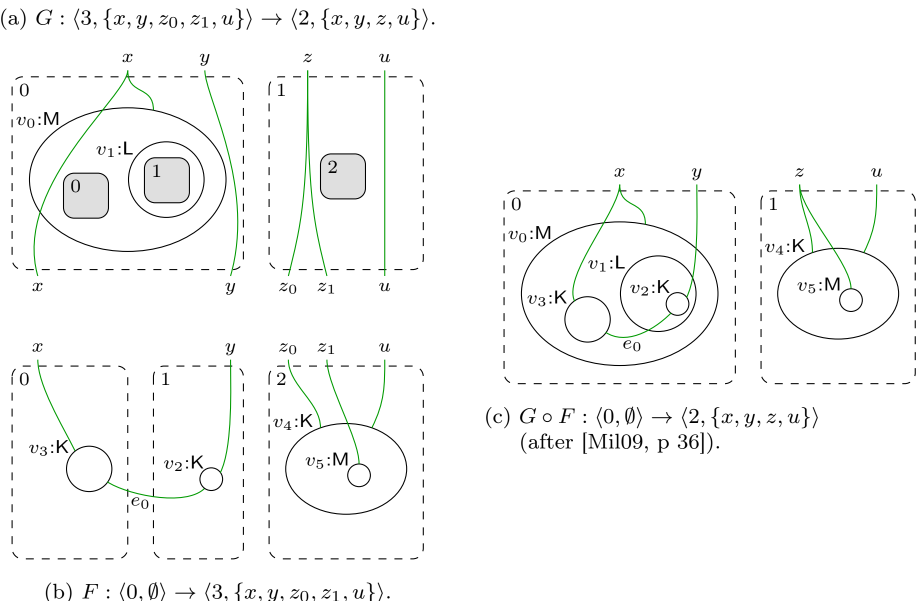
We shall use the convention that . and $\circ$ bind tightest while $\otimes$ and $\parallel$ bind weakest.

### Algebra of Bigraphs

There is a number of basic abstract bigraphs which, together with composition and tensor product, allow us build any bigraph, i.e., they provide a complete term language for abstract bigraphs. The basic abstract bigraphs are listed in Table 1.1, where we also give examples and define the names and variables we shall use to refer to each kind of basic abstract bigraph. We shall often write 1 for the barren root $merge_0$ and $\Box$ for the single site $merge_1$. As an example, the following term represents the abstraction of the bigraph of Figure 1.1a:

$$(/\{e_0, e_1\} \otimes \mathsf{id}_{\langle 2, \{y_0, y_1, y_2\}\rangle}) \circ (\mathsf{K}_{[y_0,e_0]}.(\Box \mid \mathsf{K}_{[y_0,e_1]}.1) \parallel \mathsf{M}_{[e_0,e_1,y_1,y_2]} \parallel [e_0, y_2]/[x_0, x_1]).$$

Jensen and Milner have axiomatized equivalence of abstract bigraphs in terms of basic bigraphs and shown that there are two normal forms, the so-called *discrete normal form* (DNF) and *connected normal form* (CNF) [JM04]. We shall not discuss these further here, as the details are irrelevant to our later discussions; what matters is that the term language for abstract bigraphs is sound, complete, and equipped with a structural congruence that coincides with graph isomorphism.

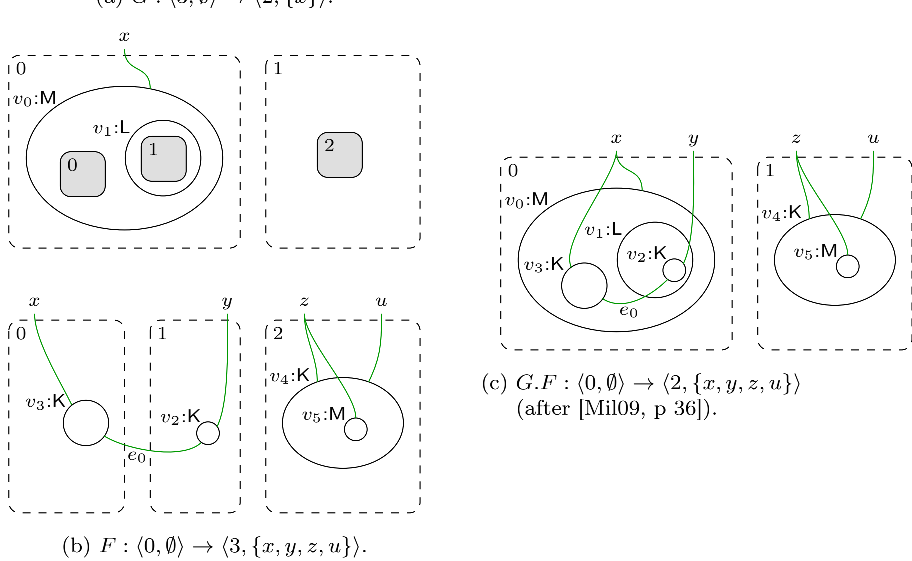



(a) $G : \langle 3, \{x, y, z_0, z_1, u\}\rangle \to \langle 2, \{x, y, z, u\}\rangle$.

(b) $F : \langle 0, \emptyset\rangle \to \langle 3, \{x, y, z_0, z_1, u\}\rangle$.

(c) $G \circ F : \langle 0, \emptyset\rangle \to \langle 2, \{x, y, z, u\}\rangle$ (after [Mil09, p 36]).

Figure 1.2: Example composition of bigraphs.

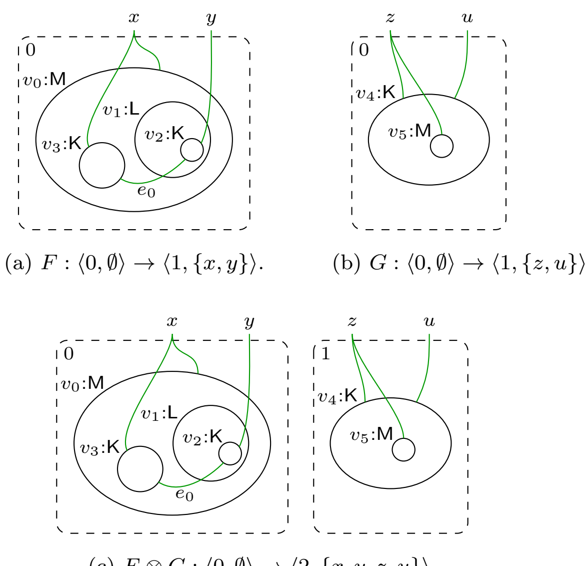



(a) $G : \langle 3, \emptyset\rangle \to \langle 2, \{x\}\rangle$.

(b) $F : \langle 0, \emptyset\rangle \to \langle 3, \{x, y, z, u\}\rangle$.

(c) $G.F : \langle 0, \emptyset\rangle \to \langle 2, \{x, y, z, u\}\rangle$ (after [Mil09, p 36]).

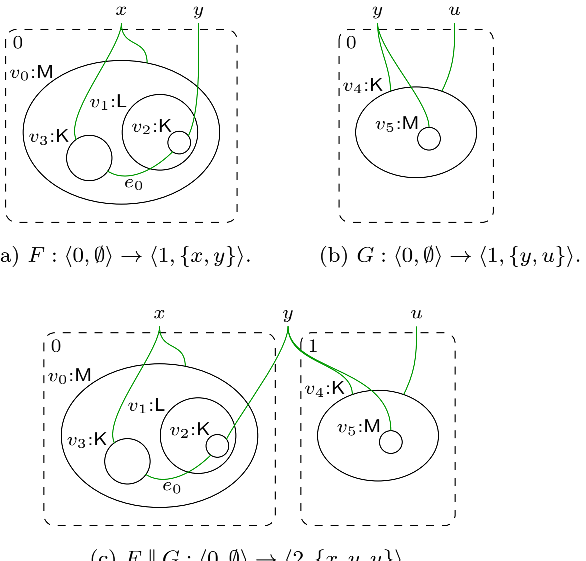
Figure 1.3: Example nesting of bigraphs.




(a) $F : \langle 0, \emptyset \rangle \to \langle 1, \{x, y\} \rangle$. (b) $G : \langle 0, \emptyset \rangle \to \langle 1, \{z, u\} \rangle$.

(c) $F \otimes G : \langle 0, \emptyset \rangle \to \langle 2, \{x, y, z, u\} \rangle$.

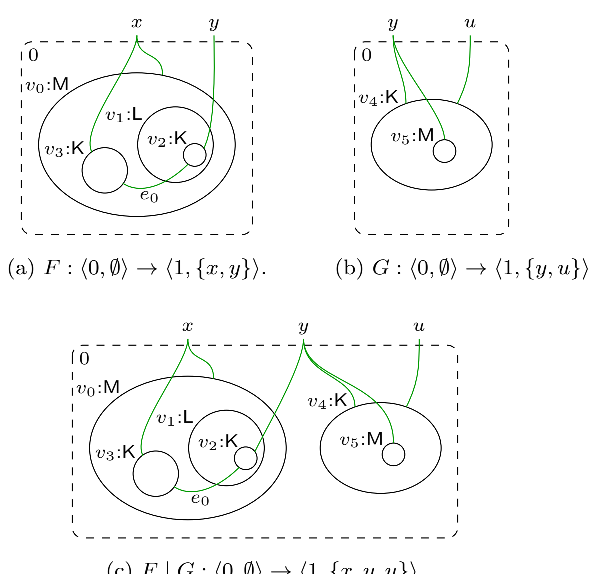
Figure 1.4: Example tensor product of bigraphs (after [Mil09, p 36]).




(a) $F : \langle 0, \emptyset \rangle \to \langle 1, \{x, y\} \rangle$. (b) $G : \langle 0, \emptyset \rangle \to \langle 1, \{y, u\} \rangle$.

(c) $F \parallel G : \langle 0, \emptyset \rangle \to \langle 2, \{x, y, u\} \rangle$.

Figure 1.5: Example parallel product of bigraphs (after [Mil09, p 36]).




(a) $F : \langle 0, \emptyset \rangle \to \langle 1, \{x, y\} \rangle$.

(b) $G : \langle 0, \emptyset \rangle \to \langle 1, \{y, u\} \rangle$.

(c) $F \mid G : \langle 0, \emptyset \rangle \to \langle 1, \{x, y, u\} \rangle$.

Figure 1.6: Example prime parallel product of bigraphs (after [Mil09, p 36]).

| | *Notation* | *Example* |
|---|---|---|
| Merge | $merge_n : n \to 1$ | $merge_3 =$ |
| Substitution $\sigma$ | $\vec{y}/\vec{X} : X \to Y$ | $[y_1, y_2, y_3]/[\{x_1, x_2\}, \{\}, \{x_3\}] =$ |
| Renaming $\alpha, \beta$ | $\vec{y}/\vec{x} : X \to Y$ | $[y_1, y_2, y_3]/[x_1, x_2, x_3] =$ |
| Closure | $/X : X \to \{\}$ | $/\{x_1, x_2, x_3\} =$ |
| Wiring $\omega$ | $(\mathsf{id} \otimes /Z)\sigma : X \to Y$ | $(\mathsf{id}_{\{y_1, y_2\}} \otimes /\{z_1, z_2\})$ $[y_1, z_1, y_2, z_2]/$ $[\{\}, \{x_1, x_2\}, \{x_3, x_4\}, \{x_5\}] =$ |
| Ion | $K_{\vec{y}} : 1 \to \langle 1, \{\vec{y}\} \rangle$ | $\mathsf{K}_{[y_1, y_2]} =$ |
| Permutation $\pi$ | $\{i \mapsto j, \ldots\} : m \to m$ | $\{0 \mapsto 2, 1 \mapsto 0, 2 \mapsto 1\} =$ |

Table 1.1: Basic abstract bigraphs and variables ranging over abstract bigraphs (after [BDGM07, Table 1]).

### 1.2.2 Bigraphical Reactive Systems: the Dynamic Structure

Bigraphs are equipped with dynamic semantics through *reaction rules*, similar to the rewrite rules of graph rewriting [Roz97, EEKR99, EKMR99], which generate *reactions* by replacing a *redex* with a *reactum* in a bigraph. However, bigraphical reaction rules differ from the usual rewrite rules in that they are *parametric*, i.e., part of the context becomes a parameter which the rule may discard or duplicate. Also, reactions are generated in an algebraic fashion, in contrast to the single- or double-pushout approaches of graph rewriting. Thus, while there are many similarities between bigraphical reactive systems and graph rewriting, the approaches are quite different; see [Mil05, Ehr02] for more thorough discussion of the connections.

There are several approaches for defining BRSs, but here we shall essentially follow Milner's book [Mil09]. BRSs are an instance of the more general *reactive systems*, but here we shall give the definitions in terms of bigraphs. A concrete ground reaction rule $(r : J, r' : J)$ consists of a redex $r$ and a reactum $r'$, both concrete and ground. If we can find an occurrence of $r$ in a concrete ground bigraph $a : I$, i.e., $a \simeq C \circ r$, then the rule generates the reaction $a \rightarrowtriangle a'$ where $a' \simeq C \circ r'$. We call $C$ the *context* and $\rightarrowtriangle$ the reaction relation. Reactions are limited to *active* contexts: every control is assigned a status, *active* or *passive*, and a context is active iff all its sites only have active ancestors in the place graph.

A concrete BRS consists of a signature and a set of ground reaction rules, closed under support equivalence, over a signature and the reaction relation generated by the rules. An abstract BRS is obtained by quotienting the agents, rules, and reaction relation of a concrete BRS by lean support equivalence.

Parametric rules, i.e., rules where the redex and reactum may have sites, are added to this framework by viewing them as generators of ground reaction rules as follows: A concrete parametric reaction rule $(R : m \to J, R' : m' \to J, \eta : m' \to m)$ consists of concrete redex $R$, concrete reactum $R'$, and *instantiation map* $\eta$ that maps sites of the reactum to sites of the redex. For a given site of the reactum, the instantiation map describes which parameter from the redex should be plugged into the site. For any concrete discrete bigraph $d = d_0 \otimes \cdots \otimes d_{m-1} : \langle m, Y \rangle$, with each $d_i$ prime, the parametric rule generates all the ground rules $(r, r')$ where $r \simeq R.d$, $r' \simeq R'.(d'_0 \parallel \cdots \parallel d'_{m'-1})$, and $d'_j \simeq d_{\eta(j)}$. We usually write $\bar{\eta}(d)$ for the instantiated parameters $d'_0 \parallel \cdots \parallel d'_{m'-1}$.

Note that we require $d$ to be discrete, i.e., it has no edges and its link map is a bijection between the ports and outer names. This means that all copies of a parameter will share their links, hence the use of the parallel product in the generated reactum. If we view edges as name restriction, this intuitively means that reaction cannot generate fresh names for copied parameters, as edges are never copied. Later, we shall discuss extensions to the theory that allow generation of fresh names.

#### Labeled Transition Systems

Though it plays no direct role in the work presented in this thesis, it is important to note that bigraphs have a theory of behavioral equivalence: labeled transition systems (LTSs) can be derived in a uniform manner from the reaction rules, and these LTSs enjoy the property that bisimulation is a congruence. We mention this here as it plays a prominent role in the bigraph literature: it has been a goal in the design of bigraphs that labels should be derivable rather than constructed ad-hoc for each calculus, and much work has gone into ensuring that extensions to the theory of bigraphs preserve this property. We refer the reader to Milner's book for an exhaustive treatment of this subject [Mil09]; for our purposes it suffices to note that the derivation of labels relies on the presence of support, i.e., concrete bigraphs.

### 1.2.3 Bigraph Developments

In the previous sections, we defined the foundations of the theory of bigraphs. In the following sections, we give an overview of some developments on the theory that are particularly relevant to the presented works.

#### Binding Bigraphs: Fresh and Binding Names

As discussed above, the usual theory of bigraphs does not allow edges to be copied nor inner names in redexes, which prevents us from creating new names and from modeling name-binders such as the input-prefix of the $\pi$-calculus. In this section we shall discuss extensions to the theory which addresses these issues by introducing *binders*, which can be thought of as edges that have locations. We shall use the term *pure bigraphs* to refer to the theory of bigraphs without binders, as defined in the previous sections.

The notion of binders in bigraphs have been evolving since Jensen and Milner's original formulation in [JM04] upon which our works in Parts II and III are based. In order to appreciate the discussions of binding in the next chapter, we shall discuss here the various stages of this evolution.

**Version 1: binding ports and singly located names** Originally, in what we shall simply call *binding bigraphs*, binders were added to bigraphs by designating some ports of a control as *binding* [JM04]. The binding ports of a node are treated as a localized link, a *bound link*, that must satisfy the *scope condition*: all the points of a binding port must be located inside the node. In order to ensure that composition preserves this property, interfaces are extended to allow names to be located at a single root or site: a *binding interface* $\langle m, \vec{X}, X\rangle$ corresponds to the pure interface $\langle m, X\rangle$ but in addition the vector of name sets $\vec{X} = [X_0, \ldots, X_{m-1}]$ specifies that the names of $X_i \subseteq X$ are located at $i$, subject to the condition that the $X_i$ are pairwise disjoint. We say that names in $\vec{X}$ are *local*, while the remaining names of $X$ are *global*. The scope condition is then extended to require that the points of an outer name located at root $i$ are all located inside that root.

Composition $F \circ G$ is still only defined if the interfaces match, and thus if an outer name is located at a root of $G$ it must also be located at the corresponding site of $F$. This ensures that composition preserves the scope condition. The parallel product $F \parallel G$ and prime parallel product $F|G$ are restricted to only allow sharing of global names, while nesting is $F.G$ is extended to allow local names in the interface between $F$ and $G$.

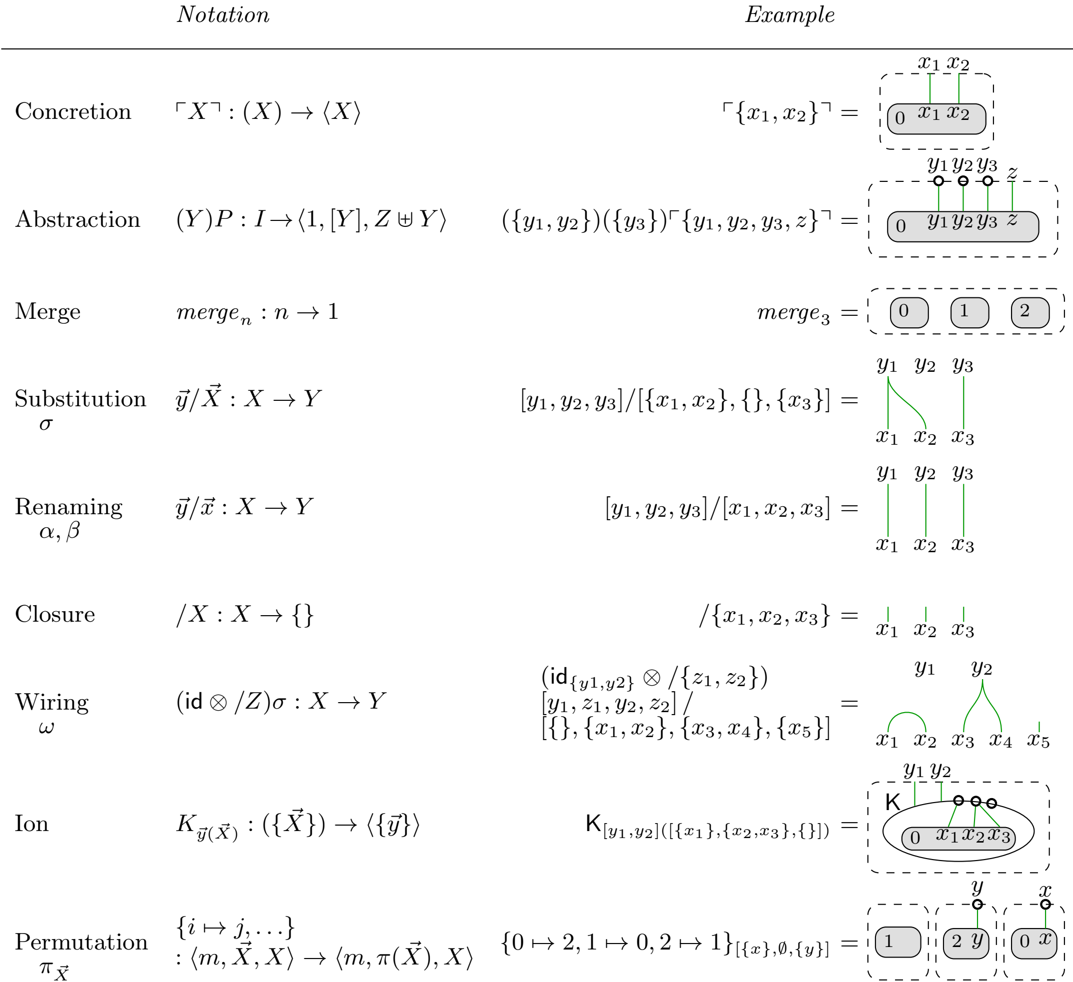
Binding requires some adjustments to the handling of parametric reaction rules:

- Recall from the definition of parametric reaction rules above that they generate ground reaction rules from discrete parameters, i.e., bigraphs where the link graph is a bijection between ports and outer names. In order to cater for parameters with bound links, we must refine the discreteness condition to allow bound links.

- We allow parametric reaction rules to have local inner names, in order to model binding prefixes. Again, this requires the discreteness condition to allow bound links, such that the redex may capture the bound linking. Furthermore, we must extend the instantiation map to record how to map the local names of each site of the reactum to the names of the corresponding site of the redex.

With these adjustments, the generation of ground reaction rules and reactions proceed as before, except that the instantiation map also renames the local outer names of the parameters.

Binding bigraphs is a conservative extension: any pure bigraph is also a binding bigraph, and pure bigraphs and pure parametric reaction rules generate the same ground reaction rules and reactions in both theories.



| | *Notation* | *Example* |
|---|---|---|
| Concretion | $\ulcorner X \urcorner : (X) \to \langle X \rangle$ | $\ulcorner\{x_1, x_2\}\urcorner =$ |
| Abstraction | $(Y)P : I \to \langle 1, [Y], Z \uplus Y \rangle$ | $(\{y_1, y_2\})(\{y_3\})\ulcorner\{y_1, y_2, y_3, z\}\urcorner =$ |
| Merge | $merge_n : n \to 1$ | $merge_3 =$ |
| Substitution $\sigma$ | $\vec{y}/\vec{X} : X \to Y$ | $[y_1, y_2, y_3]/[\{x_1, x_2\}, \{\}, \{x_3\}] =$ |
| Renaming $\alpha, \beta$ | $\vec{y}/\vec{x} : X \to Y$ | $[y_1, y_2, y_3]/[x_1, x_2, x_3] =$ |
| Closure | $/X : X \to \{\}$ | $/\{x_1, x_2, x_3\} =$ |
| Wiring $\omega$ | $(\mathsf{id} \otimes /Z)\sigma : X \to Y$ | $(\mathsf{id}_{\{y_1, y_2\}} \otimes /\{z_1, z_2\})$ $[y_1, z_1, y_2, z_2]\,/$ $[\{\}, \{x_1, x_2\}, \{x_3, x_4\}, \{x_5\}] =$ |
| Ion | $K_{\vec{y}(\vec{X})} : (\{\vec{X}\}) \to \langle\{\vec{y}\}\rangle$ | $\mathsf{K}_{[y_1, y_2]([\{x_1\}, \{x_2, x_3\}, \{\}])} =$ |
| Permutation $\pi_{\vec{X}}$ | $\{i \mapsto j, \ldots\}$ $: \langle m, \vec{X}, X \rangle \to \langle m, \pi(\vec{X}), X \rangle$ | $\{0 \mapsto 2, 1 \mapsto 0, 2 \mapsto 1\}_{[\{x\}, \emptyset, \{y\}]} =$ |

Table 1.2: Basic abstract binding bigraphs, the abstraction operation, and variables ranging over abstract bigraphs (after [BDGM07, Table 1]).

Damgaard and Birkedal have extended Jensen and Milner's axiomatization of pure bigraphs to binding bigraphs and shown, exploiting the adapted notion of discreteness, that binding bigraphs also have a discrete normal form, called the *binding discrete normal form* (BDNF) [DB06]. The corresponding extension of the basic abstract bigraphs of Table 1.1 to basic abstract binding bigraphs are shown in Table 1.2. Ions and permutations are extended with local names, we add a *concretion* which, when composed with a prime, makes local names global, and we introduce an *abstraction* operation which is inverse to concretion. The abstraction operation is only defined for prime bigraphs to ensure that the result obeys the scope condition (recall that prime bigraphs have no global inner names). Note that we write $\{\vec{X}\}$ for the union of the $X_i$ and similarly we write $\{\vec{y}\}$ for the set of names of a vector $\vec{y}$ of names. Also, we use a shorthand for prime interfaces where all names are local: $(X) \stackrel{\text{def}}{=} \langle 1, [X], X \rangle$.

**Version 2: binding ports and multiply located names** Milner realized that the restriction of names to at most one location in an interface was unnecessarily strict. For an interface $\langle m, \vec{X}, X \rangle$,

we may safely discard the condition that the $X_i$ must be pairwise disjoint and instead generalize the scope condition to state that any point of a local outer name must be located inside one of the roots where the outer name is located.

This generalization turns out to provide important expressivity: without it, we cannot express wide reaction rules where more than one root refers to the same bound link. The consequences of this limitation will become clear in Section 2.2.2, where we discuss our work on formalizing a commercial programming language using binding bigraphs without this generalization.

Milner also realized that with this generalization we can dispense with global names altogether: since rules are not allowed global inner names, we can safely encode a global outer name as an outer name that is located at all roots. Milner terms this variant *local bigraphs*.

**Version 3: located edges and multiply located names** Recently, Milner generalized binding further [Mil09], by dispensing with the notion of binding ports and instead turning binders into edges with a location[1]. Milner argues that this is a conservative extension, since we may use a sorting discipline (discussed in Section 1.2.3) to assign a number of binders to any control [Mil09, pp 134].

To our knowledge, this generalized treatment of binding has yet to be applied in a bigraphical model, and it is therefore hard to assess the importance of the added expressivity.

### Matching: the Algorithmic Challenge

The key challenge when implementing bigraphs is determining whether a redex occurs in an agent; we call this *matching*. More formally, given an agent $a$ and a parametric redex $R$ we want to determine whether there is a context $C$ and parameter $d$ such that $a \simeq C \circ (R \otimes \mathsf{id}_Z) \circ d$; we call this a *match*. From the generation of ground reaction rules and reactions it is clear that such a match induces the reaction $a \rightarrowtriangle a'$ where $a' \simeq C \circ (R' \otimes \mathsf{id}_Z) \circ \overline{\eta}(d)$.

Birkedal et al. have analyzed matching in binding bigraphs and given a sound and complete inductive characterization of matches [BDGM07]. This characterization provides valuable insight into the matching problem and may be used to derive a provably correct and complete matching algorithm, as we shall demonstrate in Part II.

Due to the similarity with the NP-complete subgraph isomorphism problem [GJ90], it was long expected that matching would be NP-hard [BDGM07]. Recently, Bacci et al. has shown that matching in the place graph is indeed NP-complete [BMR10], and since place graph matching is a proper sub-problem of bigraph matching, this means that bigraph matching is NP-hard. Bacci et al. actually proves a stronger result: place graph matching is fixed-parameter tractable [DF95], since the intractability (unless P = NP) of place graph matching is due to the width of the redex. This is perhaps not surprising, since there are polynomial time algorithms for the subtree isomorphism problem [ST99].

### Sortings

The basic notion of a signature presented above only allows us to specify the available set of controls and their arity, but not any other structural constraints. This is insufficient for many models, e.g., models of calculi or programming languages where terms have a certain structure which must be respected in the bigraphical model. Therefore signatures are often extended with a *sorting* which specifies such structural constraints.

As was the case for binding, sortings have evolved over time and may be presented in various ways, see for example [Mil09] and [Deb08]. The basic idea, however, is to enrich interfaces with a *sort* that ensures that composition is only defined when the result will be well-formed.

[1] As a technicality, Milner's presentation introduces binders as nodes with a particular kind of control, without ports, to which points may be connected.

Most work on sortings has gone into identifying sortings that are *safe*, i.e., sortings that do not ruin the derivation of labels and thus the behavioral equivalence theory. Milner gave a number of specific kinds of sortings that are safe [Mil09], while Debois [Deb08] has developed a general framework for constructing safe sortings from certain predicates on bigraphs.

Altogether, even though sortings are ubiquitous in bigraphical models, it is not quite an off-the-shelf theory for BRSs modelers, but still a somewhat ad-hoc approach to imposing structural constraints on bigraphs.

### Bigraphical Refinement

Given that the theory of bigraphs is a meta-model, a natural question to ask is whether we can formally relate different bigraphical models in a generic manner? For example, the derivation of an LTS gives us an equivalence of agents within a BRS, but what about agents of different BRSs?

Recently, Perrone et al. have proposed a notion of (vertical) *refinement* for BRSs, i.e., a way to relate BRSs at different levels of abstraction [PDH11]. Given that we want to show that one BRS, called the *concrete* BRS, is a refinement of another BRS, called the *abstract* BRS, the approach is to define an *abstraction functor* from the concrete BRS to the abstract BRS, which relates agents in such a way that their reactions in the concrete BRS correspond to the reactions of their abstractions.

More technically, an abstraction is *safe* if a concrete reaction either corresponds to an abstract reaction or to a *no-operation*, i.e., the abstraction of the agent is the same before and after reaction. An abstraction is *live* if concrete agents always exhibit some of the behavior of their abstractions. The authors give some sufficient conditions for abstraction functors to be safe, that seem manageable in practice: the functor must preserve and respect tensor, preserve active contexts, and weakly preserve reaction rules, i.e., either the abstraction of a concrete rule must yield either an abstract rule or a trivial rule where redex and reactum are equal.

This theory of bigraphical refinement has yet to be employed, but we expect that it will prove very useful. For example, as we shall discuss in Section 2.2.2, one could imagine using it to formally relate well-studied calculi, such as the $\pi$-calculus, to full-blown programming languages, thereby enabling transfer of techniques and results.

### Bigraphs as XML

Hildebrandt and Winther noticed that there is a rather immediate relation between pure bigraphs and XML [HW05]: there is an approximate correspondence between (i) bigraph nodes and XML elements, (ii) ports and attributes, and (iii) outer names and attribute values. However, the correspondence is not exact: XML models ordered trees whereas bigraphs are unordered, and there are no obvious constructs in XML that correspond to bigraph edges. They use this relation to give an XML representation of pure open bigraphs and an XML based implementation of pure open BRSs, called *Reactive XML* [HW05]. Independently, Conforti et al. noticed and exploited the correspondence to model XML as bigraphs [CMS05].

As we see in Section 2.2.2, this relation between bigraphs and XML can be exploited to give rather direct bigraphical semantics to XML-based programming languages.

### Higher-Order Bigraphs

During work on formalizing a subset of the WS-BPEL language [WS-07] in Reactive XML, Hildebrandt et al. discovered that there are some bigraphical models where it is convenient to use rules with non-ground parameters, what they call *higher-order* reaction rules [HNO06b]. In particular, they developed, employed, and implemented what could be called second-order reaction rules to require that the parameters of a reaction rule must contain specific ground bigraphs. The idea was

developed further by Birkedal et al. to a notion of higher-order bigraphs [BBD$^+$09], which allow one to express rules where parameters are higher-order contexts.

As stated above, and as we shall discuss in Section 2.2.2, there are models where second-order reaction rules are very convenient. It is less clear, however, whether higher-order bigraphs in general add useful expressivity, as they, to our knowledge, have not yet been employed in any bigraphical models.

### Calculational BRSs

When modeling some formal languages, such as programming languages, we need to model data and computation on data, i.e., classical computation. Recently, Debois has shown how classical computation can be captured as a class of BRSs called *free calculi*, and shown that these are Turing complete [Deb11]. Furthermore, Debois gives sufficient conditions to ensure that a free calculus is safely embedded in a larger BRS, which he calls a *calculational BRS*. By "safe", we mean that computational reaction cannot prevent other reactions, only enable them.

Though expressed non-constructively in loc. cit., the definition of calculational BRSs imply a rather modular approach to embedding classical computation in bigraphical models: we may construct a free calculus which models calculation on a particular type of data and then embed it in a larger BRS where that data type is needed. This holds great promise from the perspective of bigraphical languages as we shall see in the next chapter.

### Stochastic Bigraphs

For some models it is relevant to assign quantitative information to reactions. For example, we might wish to model how long a reaction takes, such that we may analyze the behavior of a system over time. In stochastic process algebra, e.g., [GHR92, Pri95, Hil96], this is achieved by enriching actions (prefixes) with probability distributions which stochastically characterize the quantitative behavior of actions. As reactions are generated by the synchronization of actions, these enriched actions allow us to derive stochastic characterizations of reactions. Usually, exponential distributions are used because they are memoryless: the probability of a reaction firing is independent of the time since the last reaction fired, which eases analysis. Exponential distributions are characterized by a single constant, their *rate*.

Recently, Krivine et al. have defined *stochastic bigraphs*, a stochastic semantics for bigraphs similar to those for stochastic process algebra [KMT08]. Rules are assigned a rate constant which is the parameter of an exponential distribution characterizing the stochastic behavior of reactions generated by the rule. To calculate the rate of a given reaction, again determining an exponential distribution, one counts all the distinct instances of each rule that generates that reaction and add the product of this count and the rate constant to the rate, which starts at zero. In order to properly count instances of rules one needs support, i.e., concrete bigraphs. There a many potential application of stochastic bigraphs and, as we shall discuss in Section 2.3, one such is the modeling of biological systems.

# Chapter 2

# Summary

Parts II–V of this dissertation contain six chapters each consisting of a single paper. They are the written products of the following four lines of research:

**Part II: A Tool for Bigraphical Programming Languages**

To assist development of, and experiments with, bigraphical programming languages, we developed an implementation of binding BRSs called the BPL Tool. The crux was the development of a provably correct matching algorithm for binding bigraph terms, described in the technical report in Chapter 3. The tool itself is presented in the technical report in Chapter 4.

**Part III: Bigraphical Semantics for Business Processes**

As part of a cross-disciplinary project aiming to develop formalized business process languages and implementations to support the needs of mobile workers, we used binding bigraphs and the BPL Tool to develop a formalization of a subset of the commercial business process language WS-BPEL and extended both language and formalization with higher-order, mobile, embedded processes. The extension and formalizations are presented in the technical report in Chapter 5. As a step towards formalizing all of WS-BPEL, we identified a core subset of the language and constructed an idempotent transformation from WS-BPEL into the core subset. This work is presented in the technical report in Chapter 6·

**Part IV: Scalable Simulation of Stochastic Bigraphs**

To assist development of, and experiments with, bigraphical languages for cell biology, we developed an implementation of stochastic bigraphs called SBAM. We recast a scalable stochastic simulation algorithm for the $\kappa$-algorithm to bigraphs, which required a number of developments for the theory of (stochastic) bigraphs. This work is presented in the technical report in Chapter 7.

**Part V: A Bigraphical Language for Cell Biology**

As part of an effort to extend the $\kappa$-calculus, a language for modeling protein-protein interaction, with dynamic compartments, we developed a bigraphical language for cell biology with a simple presentation in the style of process calculi. The language is presented in Chapter 8.

The following sections introduce each of these lines of work in more detail, and summarize and discuss each of the papers. At the end of the chapter, we conclude on the insights these works have provided with respect to the thesis that bigraphs may be used as an executable foundation for realistic formal languages, and outline some directions for future work.

## 2.1 A Tool for Bigraphical Programming Languages (Part II)

The BPL project researched the design of programming languages based on the theory of bigraphs and this work was continued in the CosmoBiz project. As part of this research, we undertook to implement the BPL Tool, a tool for abstract binding BRSs which could be used to model and experiment with such languages.

The goals for the BPL Tool were (i) to implement the complete theory, as it was unclear at the time which restrictions to the theory would be reasonable, and (ii) to have a close correspondence between the theory and its implementation to achieve trust in the correctness of the implementation. To achieve these goals, we needed a detailed formulation of the theory that could be implemented directly, and an interface language. Damgaard and Birkedal developed a term language and normal forms for abstract binding bigraphs [DB06] which both provided a convenient interface language and a formal representation of abstract bigraphs that could be directly and faithfully implemented. What remained was a matching algorithm for binding bigraph terms. As a first step towards a such, Birkedal et al. gave an inductive characterization of matching in binding bigraphs [BDGM07]. Based on this, we developed, proved correct, and implemented a matching algorithm as described in the technical report in Chapter 3: *An Implementation of Bigraph Matching*. This work is the core of the BPL Tool, which is presented in Chapter 4: *The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems*. The following sections summarize and discuss these two papers.

### 2.1.1 An Implementation of Bigraph Matching

#### Our Approach

We derived a matching algorithm for terms from the inference rules of Birkedal et al.'s inductive characterization of matching [BDGM07] using the following approach:

- The inductive characterization of matching was recast in bigraph terms, the crux being the addition of an inference rule allowing structural congruence to be applied to terms.
- By exploiting the binding discrete normal form (BDNF), we showed that it is only necessary to consider a subset of the possible derivations, the so-called *normal inferences*. These generate all matches for terms on BDNF. Moreover, normal inferences limit where and how structural congruence must be considered. We were therefore able to incorporate structural congruence into other rules, thus eliminating the need for a separate rule.
- Finally, we showed how to interpret the inference rules as a backtracking algorithm.

This approach required the development of some auxiliary theory for dealing with terms:

**normalization:**
We developed a normalization algorithm, as the developed matching algorithm requires terms to be on normal form.

**renaming:**
Structural congruence for bigraph terms includes a kind of $\alpha$-equivalence, which the normalization algorithm in principle must take into account in order to avoid name clashes. However, by preceding normalization with a phase where internal names are suitably renamed, $\alpha$-equivalence may be safely ignored. This is, however, non-trivial due to use of identity link graphs $\mathsf{id}_Z$ in the normal form, which prevents us from simply requiring every internal name to be unique. Instead, we developed a more refined notion of uniqueness, *horizontal* uniqueness, and a corresponding renaming algorithm.

**regularization:**
Reaction rules can always be expressed using a regular redex, i.e., a redex which can be expressed as a term without permutations. Such redexes are simpler to match, and the developed matching algorithm therefore requires redexes to be regular. As regular bigraphs may be represented using permutations, we developed a regularization algorithm that removes permutations.

These three algorithms were all expressed as inference systems with an obvious algorithmic reading.

Our approach allowed a relatively straightforward implementation in Standard ML, using an algebraic datatype to represent terms and a single function for each inference rule. The gap between code and theory is therefore very small, increasing trust in the correctness of the code.

The technical report briefly introduces the BPL Tool through an example: a model of the polyadic $\pi$-calculus. This introduction has been improved, expanded, and updated in the technical report about the BPL Tool which is discussed in the following section.

### Discussion and Future Work

The theory developed in the report formed the core of the first implementation of binding BRSs, an important step towards the practical development of bigraphical programming languages. The approach results in a high level of trust in the correctness and completeness of the matching algorithm and its implementation. As we shall discuss in more detail in the next section, the implementation has been successfully used to model and experiment with a number of binding BRSs.

However, the matching algorithm has turned out to be somewhat inefficient: it works well for smaller examples, such as the polyadic $\pi$-calculus, but it is unusable for more realistic models, such as the WS-BPEL model to be discussed in Section 2.2.2. Since matching is NP-hard in the general case and the algorithm is complete, it is to be expected that matching will be inefficient in some bigraphical models. But in addition we believe that, to a large extend, the inefficiency comes from the following aspects of the algorithm, which may be improved:

**structural congruence:** Though the search space is reduced by looking only for the so-called normal inferences, the algorithm will often find the same match more than once. Not because of identical siblings leading to isomorphic matches, but because the algorithm does not handle structural congruence optimally: essentially, the rules that handle structural congruence sometimes negate each other, meaning that the same terms are matched several times.

Close inspection of the algorithmic interpretation of normal inferences reveal that they almost correspond to depth-first traversals of the place graph of the agent. However, this is not reflected in the way the structural congruence for terms is incorporated into the inference rules: structural congruence is handled in four separate inference rules which together generates all possible structurally congruent terms for both agent and redex, even though this often leads to repeated matching of identical sub terms.

We therefore suggest that, instead of directly using the inference rules of the inductive characterization, one constructs inference rules which correspond exactly to the steps of a depth-first traversal, thereby ensuring that structural congruence is only applied when necessary.

By showing that these inference rules are derivable in the inference system of the inductive characterization, we preserve soundness of the algorithm. Completeness will have to be shown anew, but it will follow the old proof closely and thus should not be too difficult. The main challenge will be to prove that each match is found at most once, which combined with completeness means that they are found exactly once.

**link graph matching:** The term language, and in particular the BDNF, follows the structure of the place graph closely. This is reflected in the matching algorithm, where matching is only

guided by the place graph structure, which has some unfortunate effects on the matching of the link graph:

- it is mainly deferred to the axioms (leaves) of the matching inferences, i.e., information about the link graph is not used to prune the search space as early as it could,
- it is specified declaratively in the form of equations, without making clear how these equations should be solved, and
- it does not exploit the (partial) orthogonality of the link and place graphs: it ties the matching of the two graphs together in such a way that, if the matching algorithm needs to backtrack for one of the graphs, matching of both would must be redone.

If one could interleave matching of the link graph with the matching of the place graph, we believe that the search space could be pruned significantly. Perhaps one could use a constraint formulation along the following lines:

- while traversing the terms, i.e., place graph, gather information about the link graph in the form of constraints,
- if the set of constraints ever become inconsistent, we can prune the search space, and
- when a place graph match is found, generate the corresponding bigraph matches by solving the link graph constraints, without having to repeat the place graph matching.

The hope would be that one could express the constraints in a form that is supported by off-the-shelf constraint solvers.

These issues reflect that the algorithm works on a term representation of abstract bigraphs. Other representations might allow more efficient implementations. Indeed, Sevegnani et al. are working on an implementation of matching based on a SAT formulation of concrete bigraphs, which allows them to use one of the many highly optimized SAT solvers to solve the matching problem [SUC10]. As we shall discuss in Section 2.3, we have also proposed a matching algorithm based on a direct representation of concrete bigraphs and a notion of bigraph embeddings, which we believe to be more efficient than the one presented above. Overall, we are skeptical about the efficiency of term-based matching for bigraphs.

### 2.1.2 The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems

The BPL Tool is the first implementation of binding BRSs, and it provides facilities for manipulation, simulation, and visualization of binding BRSs. It can be used either through the included web and command line user interfaces or as a programming library. The BPL Tool is available for download from [BPL07] where documentation and an online demo of the tool may also be found.

The technical report in Chapter 4 presents the command line interface (CLI) of the BPL Tool. It provides instructions for installation, a hands-on example, and a functionality reference.

#### Our Approach

The central component of the BPL Tool CLI is an ASCII version of the term language for binding bigraphs, extended with syntactic sugar for oft-used constructions such as prime parallel product. Together with syntax for specifying bigraph signatures, reaction rules, and a set of functions, providing facilities such as matching, this forms the BPL Tool language called BPL Language (BPLL).

BPLL is embedded into Standard ML (SML) whereby any SML interpreter may be used as a CLI for the BPL Tool. The Standard ML of New Jersey interpreter [AM91] is recommended as it

allows the BPL Tool to provide a more refined interface. The embedding of BPLL into SML also means that it is easy for users to extend the BPL Tool by writing additional SML functions, and to integrate the BPL Tool as a programming library into SML programs.

In overview, the BPL Tool provides the following facilities:

**Signatures**
Signatures are defined one control at a time, by specifying a name, status, and arity. For modeling convenience, the tool supports naming ports.

**Bigraphs**
Bigraphs are constructed using an ASCII syntax for the binding bigraph term language extended with various forms of syntactic sugar such as the parallel product $\|$.

**Reaction rules**
A reaction rules is constructed from a pair of bigraph terms, an optional instantiation map, and an optional rule name for convenience. For modeling convenience, the tool will attempt to infer missing or partially specified instantiation maps in a well-specified manner.

**Matching**
All matches of a redex in an agent can be found. Matches are computed lazily (the set of matches are represented as a lazy list) such that one only incurs the cost of matching for the matches that are actually needed.

**Simulation**
Reactions may be generated from an initial agent and a set of reaction rules. The tool allows the specification of reaction *tactics* which are strategies for simulation, e.g., the application of rules in a certain order.

**Equality testing**
There is a bigraph equality operator, i.e., an operator for deciding structural congruence of bigraph terms.

**Term normalization, regularization, and simplification**
Terms may be normalized, regularized, and simplified. Simplification is a heuristic application of various structural congruence rules that often yield simpler bigraph terms. In particular, it often yields terms on a form that will allow the pretty printer to use more syntactic sugar.

**Pretty printing**
Values, such as matches or agents, may be pretty printed in the BPL Language. Through a number of options, the user may control various aspects of pretty printing, e.g., what kinds of syntactic sugar should be used.

**Visualization**
Scalable Vector Graphics and TikZ diagrams can be generated for bigraph terms. An external viewer is required to view the results.

Most of these features are demonstrated through an example: a BRS model of Milner's polyadic $\pi$-calculus model of a mobile phone system from [Mil99].

### Discussion and Future Work

The BPL Tool is a quite mature tool and it has successfully been used by a number of people in the modeling of various BRSs (all listed in the report) including a subset of the WS-BPEL language as we shall discuss in the next section. It provides many of the features one would want of a tool for specification and simulation of bigraphical languages.

New features and improvements that should eventually be incorporated into the tool include:

**more efficient matching**

As discussed in the Section 2.1, the matching algorithm is not efficient enough for realistic examples. In that section, we suggested a number of improvements to the algorithm that we hope will improve the algorithm significantly. Alternatively, one could imagine integrating another matching engine such as the SAT based one being developed by Sevegnani et al. [SUC10] or our own so-called anchored matching algorithm which will be discussed in Section 2.3.

**GUI**

Bigraphs being a graphical formalism, it seems natural to provide a graphical user interface for the BPL Tool. As a first step towards a such, we have implemented support for exporting bigraphs as BPLL models in the promising Big Red graphical bigraph editor, developed by Alexander Faithfull [Fai10] under the guidance of Stephen Gilmore and myself.

We should also like to see the BPL Tool integrated into BigWB, a workbench for bigraphs currently being developed by Miculan and coworkers, which aims to provide a unifying interface for the various tools for bigraphs. Recently, we have started coordinating our efforts; in particular, we are working on integrating Big Red and BigWB by designing a common infrastructure including a data model and file formats.

**dedicated UI**

While the embedding of BPLL into SML means easy extensibility and integration with SML programs, it also puts some restrictions on the choice of syntax resulting in a slightly inconvenient notation. By developing a dedicated CLI or GUI we would be free to choose a more convenient syntax.

**sortings**

Support for sortings would enable the tool to assist users in ensuring that bigraphs are well-formed and checking that reaction rules preserve well-formedness.

**datatypes**

Built-in support for datatypes and manipulation of data would make it easier to express models containing computations. We suggest that such that an extension should be founded on a solid formal foundation, such as the calculational BRSs of Debois [Deb11].

**generalized binding**

As discussed in Section 1.2.3, the notion of binding has been generalized, and as we shall see in Section 2.2.2, these generalizations increase the expressivity of reaction rules. We conjecture that extending the theory and implementation to allow for multiply located names will pose no significant challenges.

## 2.2 Bigraphical Semantics for Business Processes (Part III)

The CosmoBiz project was a cross-disciplinary research project, joining researchers in computer supported cooperative work, researchers in formal semantics and types for programming languages, and an industrial partner (Microsoft) in an effort to

> "provide formalizations and implementations of business process languages supporting mobile and adaptive business processes which support the needs of mobile workers and impact the future commercial business process management systems." [Hp07]

The studies presented in Part III are a strand of this effort. In the technical report in Chapter 5 we first develop an extensible formalization of a non-trivial subset of the commercial Web Services Business Process Execution Language (WS-BPEL) [WS-07] in binding bigraphs and the BPL Tool.

We then extend the language and formalization with locality and mobility features, resulting in the HomeBPEL language. The technical report in Chapter 6 is a first step towards expanding our formalization to all of WS-BPEL: we show that it is sufficient to formalize a subset of WS-BPEL, called Core BPEL, by giving an idempotent transformation from the full WS-BPEL language into the Core BPEL subset.

This work forms the foundation of future studies of business process languages in a mobile and adaptive setting. By expanding our formalizations to include all of WS-BPEL, a language which is used extensively in business process management systems, one would have a language with demonstrated ability to support business processes in practice, and, in addition, facilities for supporting business process mobility and adaptability. Furthermore, if the BPL Tool was made more efficient, it could serve as the computational core of a business process engine for this language which would provide a high level of trust in the correctness of the execution.

In order to present our work, we must first briefly introduce WS-BPEL.

### 2.2.1 A brief introduction to WS-BPEL

WS-BPEL is an imperative XML language for describing and implementing business processes based on web services: WS-BPEL processes communicate by invoking web services and are themselves exposed as web services, the interfaces of which are described using the Web Services Description Language (WSDL) [CCMW01]. In order to be executed, a WS-BPEL process description is deployed to an execution engine. When the corresponding web service is invoked, a new instance of the process is created and executed. It is interesting to note that while the WS-BPEL standard specifies the syntax for process descriptions, it makes no requirements as to how process instances should be represented – which is slightly surprising as business processes are often persisted for backup or migration as they are often long-running.

A WS-BPEL process description is composed of so-called *activities*, which are usually categorized as either *basic* or *structured*. The basic activities are the primitive operations, for example sending and receiving messages in a $\pi$-calculus-like manner, or assigning a value to a variable. Structured activities contain other activities and their semantics define the control flow of a process, such as branching or parallel execution.

WS-BPEL has many such activities, and we shall not discuss them all here, only point out two that are significant to our presentation:

`<exit>`: The `<exit>` activity is used to immediately end the execution of the process instance in which it is contained.

`<scope>`: The `<scope>` activity is used to define a new scope with local state such as variables. Scopes may be nested inside iterative and parallel constructs, with the consequence that several instances of the same scope may be executing at the same time.

### 2.2.2 Formalizing WS-BPEL and Higher Order Mobile Embedded Business Processes in the Bigraphical Programing Languages (BPL) Tool

As we saw above, WS-BPEL is an imperative XML language with $\pi$-calculus-like messaging constructs. And as discussed in Section 1.2.3 bigraphs and XML are closely related, so together with the fact that the $\pi$-calculus may be encoded straightforwardly in bigraphs, this suggests that a rather direct formalization of WS-BPEL as a BRS should be possible. Indeed, this was demonstrated in previous work by Hildebrandt et al. [HNO06b] where a subset of WS-BPEL was formalized in Reactive XML, an XML-centric model of computation based on a variant of pure, open bigraphs with second-order reaction rules [HW05, HNO06a] (cf. Section 1.2.3).

Furthermore, the ease with which a BRS can be extended, by adding controls and reaction rules, suggests that a bigraphical formalization of WS-BPEL could be extended to accommodate new language features; in particular, one would expect business process mobility to be easily formalizable as the modeling of mobility is a key idea behind BRSs.

To understand the contributions of our work, we must first summarize the Reactive XML formalization. Afterwards, we shall first discuss our WS-BPEL formalization, and then our proposal for extending WS-BPEL with higher-order processes.

#### The Reactive XML Approach to Formalizing WS-BPEL

Reactive XML is an XML representation of pure, open BRSs extended with second-order reaction rules, where reactions are obtained by rewriting the XML representation as specified by XML reaction rules [HW05, HNO06a]. The framework is flexible enough to admit a subset of WS-BPEL as a Reactive XML instance, allowing for an almost one-to-one correspondence between the language and its formalization. Also, as Reactive XML reactions rewrite XML, the approach has the advantage of also providing a runtime representation of process instances that is almost identical to process descriptions, the main difference being that instances have an active, instead of passive, control, whereby reaction is allowed to occur inside process instances but not process descriptions.

The Reactive XML formalization demonstrated the feasibility of giving rather direct bigraphical semantics to WS-BPEL. However, as Reactive XML is based on pure bigraphs and therefore cannot generate fresh names (cf. Sections 1.2.2 and 1.2.3), the formalism cannot model dynamic scopes as needed to model the `<scope>`-construct of WS-BPEL, which was therefore omitted from the formalization. It was also necessary to add second-order reaction rules (cf. Section 1.2.3) in order to correctly model variable scope resolution: a second-order rule can at the same time capture a `<scope>`-construct – including its variable declarations – and an activity that is nested at an arbitrary depth inside that `<scope>`, e.g., nested inside other scopes.

#### Our Approach to Formalizing WS-BPEL

Our formalization is similar to the Reactive XML formalization, except that we use binding bigraphs instead of Reactive XML, whereby we are able to model dynamic scopes and variable scope resolution without extensions to the usual theory of binding bigraphs. In turn, this allows us to employ standard theory and tools, such as the BPL Tool.

The formalization consists of two parts: a mapping from a subset of the WS-BPEL language to bigraphs, and a set of reaction rules modeling the dynamic semantics of WS-BPEL.

**Mapping WS-BPEL to bigraphs** The mapping is very direct: it preserves nesting, maps WS-BPEL elements to nodes with controls of the same name, e.g., `<process>` maps to `Process`, and maps attributes to links of the same name. The mapping only introduces one control that has no corresponding construct in WS-BPEL, namely the `Next` control which is used to model sequences; this is necessary since bigraphs, in contrast to XML, are unordered. The key technical point of the mapping is that it resolves scopes statically and encodes them as bound links: a scope has a binding port to which all variables and partner links of that scope, as well as references to these, are connected. This explicit encoding of scope resolution through bound links is what allows us to model dynamic scopes, as copies of a scope will have separate scope links. Similarly, the activities of a process are linked to a bound port of the process, which enables us to express reaction rules for activities, such as `<exit>`, that manipulate the state of the process without having to resort to higher-order extensions.

**WS-BPEL reaction rules** The reaction rules are mostly what one would expect of a rewriting semantics: there are one or two rules for each activity, which concisely capture the semantics of that activity.

The dynamic semantics introduces a number of auxiliary controls that capture runtime state. These auxiliary controls can be divided into two groups: (i) controls that capture dynamic WS-BPEL concepts such as process instances and links between process instances signifying dynamic connectivity, and (ii) bookkeeping controls necessary to capture the semantics of some activities whose semantics require multiple reactions, namely the `<scope>` and `<exit>` activities. The latter group is non-trivial and gives insight into the expressive convenience of binding bigraphs, so we shall discuss them in some detail:

`<exit>`: When executed, the `<exit>` activity must immediately end the execution of the process instance in which it is contained. An `<exit>` activity may be nested arbitrarily deep inside a process instance, e.g., nested inside `<scope>` activities, and such arbitrary nesting cannot be expressed using ordinary reaction rules. Instead, we introduce a level of indirection: process instances are assigned a status, `Running` or `Stopped`, in the form of a node located just inside the instance node. Using a wide reaction rule, we can then express that the `<exit>` activity changes the status of the instance to `Stopped`, and another rule may then discard the stopped process instance. All other rules will have to specify that the process instance status must be `Running`, to ensure that execution of an instance stops immediately after an `<exit>` activity has been executed.

If we had second-order reaction rules at our disposal, the indirection, and thus the instance status, could have been avoided.

`<scope>`: As discussed above, variable scopes are resolved when mapping from WS-BPEL to bigraphs and encoded in the form of bound links, which ensures that each instance of a scope is distinct from its counterparts. An activity that references a variable may be nested arbitrarily deep inside the scope of that variable, and we thus need wide reaction rules to model such activities, where each root refers to the bound scope link. Alas, only one root may refer to such a bound link in binding bigraphs. We must therefore unbind the scope link of a scope before the contained activities can execute: there is a rule that *activates* a scope by changing its control, from `Scope` to `ActiveScope`, and changes the bound scope link to an edge.

If we had a more general notion of binding where names may be located at more than one root (cf. Section 1.2.3), we could avoid the scope activation step.

### Our Approach to Higher Order Mobile Embedded Business Processes

HomeBPEL is a proposal for an extension to WS-BPEL where processes are first-class values that can be stored in variables, passed as messages, and activated as embedded sub-instances. A sub-instance is similar to a WS-BPEL scope, except that it can be dynamically *frozen* (using the new `<freeze>` activity) and stored as a process in a variable, and then subsequently be *thawed* (using the new `<thaw>` activity) when reactivated as a sub-instance. This extension allows one to model mobile business processes and business process management: a frozen sub-instance stored in a variable can be passed to a partner or inspected and manipulated as data and then be reactivated. In addition, we add activities for communication between sub-instances and their hosts, such that sub-instances may interact with their current environment.

As a motivating example, we model a pervasive health care scenario where treatment of patients follows predefined guidelines that are implemented as HomeBPEL processes. Every new treatment of a patient causes an instance of a guideline to be created, which is then passed to the assigned doctor's workflow process. While executing there, it will assist the doctor in following protocol and record the necessary data. A guideline may contain self-treatment guidelines in the form of a process which can be delegated to the patient's workflow process.

The challenge in extending our WS-BPEL formalization to HomeBPEL lies in the `<freeze>` activity. As discussed above, it is necessary to unbind the links of scopes before we can execute their contained activities, due to the somewhat limited variant of binding used in the BPL Tool. Freezing requires the reverse operation, i.e., we must deactivate all the scopes contained in the instance being frozen and rebind the corresponding scope links. As there can be any number of active scopes, freezing becomes an operation that requires a number of reactions, which we handle as we did for the `<exit>` activity: we add an additional status for instances, `Freezing`, which ensures that only the rules relating to freezing can be applied. However, since `<freeze>` and `<exit>` may now affect a number of processes, is is insufficient to change the status of just the instance that is directly affected. We therefore encapsulate the outermost instances in a container, `TopInstance`, and give it its own status, either `TopRunning` or `SubTransition`, which affects all nested instances.

The technicalities clutter up the formalization somewhat. In particular, the reaction rules of the WS-BPEL formalization all need to be extended to take the top-level status into account, and the arity of controls relating to the runtime representation, e.g., scopes and instances, are extended with ports that allow us to keep track of activated scopes. As suggested above, generalized binding and second-order reaction rules would probably allow us to avoid the clutter altogether. Alternatively, if reaction rules could be given priorities, we could avoid the `TopInstance` container and its status, as we could assign higher priority to the rules of activities that require multiple reactions. In the BPL Tool, this could be captured in the form of reaction tactics.

### Discussion and Future Work

Our formalizations of a subset of WS-BPEL and HomeBPEL demonstrate that binding bigraphs allow rather direct formalizations of XML-based programming languages with mobility features. However, they also expose the need for a less restrictive notion of binding than the one implemented in the BPL Tool. Also, they make it very clear that second-order reaction rules provide essential expressivity in order to give concise models of such languages. The formalizations also indicate that reaction tactics may allow simpler semantics since one may then avoid encoding reaction rule priorities, but studies are needed in order to determine the impact of reaction tactics on the behavioral theory of bigraphs.

We conjecture that our formalization can be extended to cover all of WS-BPEL, though the more complex features of WS-BPEL, such as link semantics, will probably require some bookkeeping in the runtime representation. However, this is not surprising, as the WS-BPEL standard describes the semantics of these features in terms of runtime concepts, such as *link state* [WS-07]. We expect that WS-BPEL's data model, a collection of XML models, and operations on data, XPath [XPa99] and XSLT [XSL99], could be formalized in terms of Debois' free calculi [Deb11]. We envision that bigraph tools will include libraries of such free calculi, modeling for instance XML data and XPath expressions over such data, and provide convenient syntax for integrating these seamlessly into other BRSs. In order to achieve efficient simulation, the provided free calculi could be implemented in code instead of being simulated.

In the current formalization, frozen processes are stored in variables as any other data, where they may be manipulated freely, with no guarantees that the result is a valid process description. Perhaps one could provide a set of safe primitive operations on process descriptions, such as sub-process reflection and general manipulation, e.g., editing or joining of frozen sub-processes. This relates to the work on Higher-Order (Petri) Nets and applications to workflow studied in [HM03].

Future work should also include studies of type systems, e.g., relations to the work on formalizations of WSDL types, contracts and session types [LPT06, BZ07, CHY07]. The addition of mobile embedded sub-instances also opens for a study of type systems that can guarantee safe process mobility and manipulation. In particular, it would be interesting to study the approaches used for Boxed Ambients [GCDC06] and for the higher-order $\pi$-calculus [MY07] on the safe integration of higher-order mobility and sessions.

Another relevant direction of future work is a detailed and complete study of the expressiveness of HomeBPEL in relation to workflow patterns (e.g., [RtHvdAM06]). One could also study the language primitives and expressiveness in relation to process calculi for mobility such as Ambients, Seal and Homer. In particular, we expect to examine a notion of subjective mobility as in Safe Ambients [LS03] by introducing a co-freeze activity to be carried out by the sub-instance, allowing it to decide whether (and when) it can be frozen.

Along the same lines, it would also be interesting to investigate if and how WS-BPEL can be seen as a refinement of the $\pi$-calculus (or other calculi) in terms of Perrone et al.'s recent notion of BRS refinement [PDH11]. Such a refinement relation would allow transfer of analyses and techniques from these calculi. Similarly, it would be interesting to investigate whether HomeBPEL is a refinement of higher-order calculi such as higher-order $\pi$ [San93] or Homer [Bun07].

### 2.2.3 Core BPEL

WS-BPEL is a full-fledged programming language with plenty of syntactic sugar to make the language more convenient [WS-07]. Unfortunately this results in a rather bloated language specification with much redundancy, which in turn leads to complex formalizations. Therefore, we propose to identify and formalize a simpler language core, and then define the semantics for the full language by giving a transformation into the core subset. This idea was pioneered by Landin, who coined the term *syntactic sugar* [Lan64] and demonstrated that the semantics of a programming language can be defined by giving a mapping to another language, by mapping parts of ALGOL 60 to the $\lambda$-calculus [Lan65a, Lan65b]. Later, it was demonstrated that the two languages need not be different and that to define the semantics for a language, one need only define it for a core subset and then give a mapping into this subset, cf. e.g., the definition of Standard ML [MTH90].

#### Our Approach

The WS-BPEL standard recognizes that the language is not minimal and hints at relations between different syntactic constructs but leaves those relations informal [WS-07]. Furthermore, it allows the omission of certain values and constructs, by specifying a number of defaults for these. Finally, the WS-BPEL standard specifies that some parts of the language may be ignored by an implementation, and defines how to safely do so. Our approach to identifying a core subset of WS-BPEL was to carefully analyze the WS-BPEL standard and the accompanying XML Schema, looking for any of these three kinds of opportunity for reducing the complexity of the language.

The indicators we looked for in our analysis of the XML Schema and standard were the following:

**XML Schema:**

- Attributes and elements marked as optional. These may have default values.
- Types that are referred to more than once. These may indicate redundancy.

**Standard:**

- Definitions of attributes or elements, where the semantics are explained by reference to other constructs. This may indicate redundancy.
- Attributes and elements which may be ignored by an engine. This may indicate that they are superfluous.

For each identified simplification candidate, we then examined whether the standard admits one of the following kinds of transformations:

- Does the standard specify a default value/element, or can we construct such a default, which can be made explicit?

- Can we decompose the element into a composition of smaller constructs?
- Can we remove the attribute/element without affecting the semantics?

The analysis resulted in a large number of transformations, so for the sake of brevity we only summarize a few interesting cases:

`<scope>`: Scopes in WS-BPEL allow variable declarations which may be initialized as part of the declaration. The WS-BPEL standard states that such inline initializations correspond to implicit assignment activities, but also states that variable initialization is all-or-nothing: if the initialization of a variable fails, a specific exception must be raised.

These variable initialization semantics can be made explicit by using three scopes:

1. An outer scope in which the variables are declared.
2. A scope that initializes the variables. It contains an exception handler that catches any exception and instead throws the *variable initialization* exception.
3. A scope that contains the activity of the original scope.

`<receive>`: WS-BPEL has two activities for receiving messages: `<receive>` which receives a message over a particular channel, and `<pick>` which will receive a single message from any of a number of channels, i.e., similar to input-guarded choice of the $\pi$-calculus.

Unsurprisingly, `<receive>` can be seen as syntactic sugar for `<pick>` that will only receive a message from a single channel.

As these examples indicate, the transformations are conceptually simple, but the devil is in the details. For example, WS-BPEL is extensible and, in particular, the language allows programmers to employ arbitrary languages to express conditions, and it only *recommends* that the evaluation of conditions result in a boolean value! This prevents us from transforming such conditions in a generic manner. For example, in order to transform a repeat-until loop into a while loop, the textbook transformation involves negating the condition. Since this cannot be done at the expression level, we are forced to use a more complicated transformation: the original condition is evaluated as part of an if-statement which assigns the appropriate boolean value to a variable.

Also, there are some constructs that one would perhaps expect to be syntactic sugar but which in fact are not. For example, WS-BPEL has two activities for sending messages, `<invoke>` and `<reply>`, and it seems that one of them should be redundant. However, WS-BPEL assigns incompatible WSDL types to these activities: `<invoke>` may only be used to initiate communication whereas `<reply>` may only be used to respond to a request. So while the two operations are operationally similar, the type system distinguishes them.

Overall, our analysis resulted in a set of independent transformations, which together form a transformation from WS-BPEL to a core subset which we call Core BPEL. The transformations are given in the form of a set of XSLT 1.0 transformations [XSL99], making them easily adoptable by other researchers and WS-BPEL implementers. As each template performs an independent transformation, users are free to use just a subset of the transformations. The syntax of the Core BPEL is defined by an XML Schema.

### Discussion and Future Work

Core BPEL is significantly simpler than WS-BPEL as witnessed by the sizes of their XML Schemas: the Core BPEL XML Schema has 73% the number of lines of its WS-BPEL counterpart. More importantly, since it disregards XML Schema overhead, the Core BPEL versions of the WS-BPEL standard's so-called *syntax summaries* for activities on average have 65% the number of lines, cf. Table 2.1.

| | | | | | |
|---|---|---|---|---|---|
| `<process>` | 45% | `<forEach>` | 90% | `<rethrow>` | 33% |
| `<assign>` | 87% | `<if>` | 53% | `<scope>` | 84% |
| `<compensate>` | 33% | `<invoke>` | 30% | `<sequence>` | 0% |
| `<compensateScope>` | 25% | `<pick>` | 77% | `<throw>` | 40% |
| `<empty>` | 33% | `<receive>` | 0% | `<validate>` | 25% |
| `<exit>` | 33% | `<repeatUntil>` | 0% | `<wait>` | 92% |
| `<extensionActivity>` | 50% | `<reply>` | 58% | `<while>` | 86% |
| `<flow>` | 300% | | | total | 65% |

Table 2.1: Number of lines in the Core BPEL syntax summaries for `<process>` and activities compared to their WS-BPEL counterparts. The Core BPEL XML Schema has 73% the number of lines of its WS-BPEL counterpart.

We have supported the validity of our transformations with quotes from the standard. Since the standard lacks official formal semantics, this is the best argument we can give that the transformations preserve semantics. Of course, one may examine our transformations in the context of one of the many WS-BPEL implementations or formalizations [Loh07, LVO+07, Sta05, WDW07, FGV06, FR05, Fah05, FGV04], but these are also just interpretations of the WS-BPEL standard. For similar reasons, it is unclear how one can show minimality of Core BPEL; all we can say is that we are not aware of any further or alternative transformations which yield a smaller language. However, there is one formal aspect that it is straightforward to verify for our transformations: transformed processes adhere to the same WSDL types.

The natural next step is of course to extend our own partial formalization of WS-BPEL to the Core BPEL subset.

## 2.3 Scalable Simulation of Stochastic Bigraphs (Part IV)

Some biologists argue that, in order to understand and make use of the vast amounts of data produced by experiments, it is necessary to take a leaf out of the engineer's book: formal languages should be used to build biological models with a precise meaning to facilitate communication and formal analysis [Laz02]. One would hope that the knowledge and techniques developed for formal languages in the field of computer science could be applied to answer the call from the field of biology. Indeed, the computer science community has embraced this challenge. In particular, the inherent concurrency and compositionality of biological systems has fostered an interest in the process calculus community. For example, stochastic process calculi, such as the stochastic $\pi$-calculus [Pri95], the $\kappa$-calculus [DL04], BioAmbients [RPS+04], and Bio-PEPA [CH09], have already been successfully employed to model, analyze, and simulate protein-protein interaction and, to a lesser extent, molecular localization and compartmentalization.

Recently it has been proposed that bigraphs could be used to model cellular biology, in particular protein-protein interaction combined with dynamic compartmentalization [DDK08, BGM09a]. As bigraphs can represent the $\pi$-calculus, the $\kappa$-calculus, and Ambients, it is unsurprising that bigraphs can be used to model such systems. However, there are three aspects of bigraphs that seem to make them especially well suited for the task:

1. As in the $\kappa$-calculus, the modeler is free to choose the signature and reaction rules that are suitable for a particular model. This is in contrast to for example $\pi$-calculus models, where one is restricted to a certain syntax and a single fixed reaction rule.

2. Molecular localization and dynamic compartmentalization is modeled straightforwardly in bigraphs using nesting and parametric reaction rules. BioAmbients have a rather flexible

notion of locations, while the other mentioned calculi are more limited in this respect. In particular, the $\kappa$ calculus does not have locations at all.

3. As in the $\kappa$-calculus, bigraphs provide a formal graphical notation for both the static and dynamic aspects of a model. This is not the case for the other mentioned calculi. In particular, Regev et al. acknowledge that this a significant shortcoming of their BioAmbients and point to bigraphs as a possible solution [RPS$^+$04].

Indeed, the bigraphical models of cellular biology in [DDK08, BGM09a] demonstrate that the bigraphical notation provides a natural way to visually and formally model protein interactions and cellular compartments. Combined with the stochastic semantics by Krivine et al. [KMT08], bigraphs provide a formal foundation for natural models of cellular biology that can enable analysis and simulation. However, analysis and simulation of stochastic bigraphs have yet to be developed.

The study presented in Part IV is part of an effort to develop a simulator for stochastic bigraphs suitable for cellular models. Such models are usually large both in the number of reaction rules and in the size of the bigraphical agent. Thus scalability in those two aspects is essential. The BPL Tool is ill-suited for this purpose for two reasons: it was not designed with the counting of matches in mind, and it is too inefficient as discussed in Section 2.1. We therefore set out to build a simulator from scratch, taking inspiration from an efficient and scalable simulation algorithm for the $\kappa$-calculus [DFFK07]. The $\kappa$-calculus is a stochastic process calculus with a graphical notation that is designed to model protein-protein interaction and which can be represented in bigraphs. Therefore it seems plausible that the $\kappa$-calculus simulation algorithm can be generalized to stochastic bigraphs. We report on our progress on this endeavor in the technical report in Chapter 7.

In order to present our work, we must first introduce the simulation algorithm for the $\kappa$-calculus by Danos et al. which we call KaSim [DFFK07].

## 2.3.1 The $\kappa$ Simulation Algorithm

For the sake of simplicity, we shall refrain from introducing the $\kappa$-calculus and instead present the KaSim algorithm using bigraph terminology. Also, we shall not concern ourselves with the physical underpinnings of the stochastic semantics of either the $\kappa$-calculus nor stochastic bigraphs, but simply assume they are well-founded. As we shall see, our studies can be adequately explained regardless of the interpretation of the stochastic semantics for bigraphs, and, in fact, most of our developments are independent of these.

KaSim is a generalization of what is known as Gillespie's algorithm, an algorithm for stochastic simulation of coupled chemical reactions [Gil76, Gil77]. It is based on the idea of assigning probabilities to reaction rules which are proportional to the number of instances of each rule in the current state of the system, and letting the frequency of reaction be proportional to the total number of rule instances.

From the perspective of this summary, the relevant parts of KaSim are its representation of the system state and its simulation loop.

#### The System State

The KaSim algorithm takes as input (1) a simulation duration, (2) an agent (the starting state), and (3) a set of reaction rules, where each rule is assigned a positive *rate constant*, which is essentially the speed with which that reaction takes place. In addition, it initially computes the following structures:

**matches:**
For each reaction rule, it finds all matches of the redex in the agent and stores them in the form of *graph embeddings.* The so-called *activity* of a rule is its rate constant times the number of matches of its redex.

**system activity:**

The *system activity* is the sum of the activities of all the reaction rules. Thus, if the system activity is 0 no reaction is possible.

**rule activation map:**

Through static analysis, it is determined whether applying one rule $\mathsf{R}_0$ can ever cause a new instance, i.e., match, of another rule $\mathsf{R}_1$. This is written $\mathsf{R}_0 \prec \mathsf{R}_1$ and $\prec$ is called the *rule activation map.*

**rule inhibition map:**

Dually, it is statically determined whether applying one rule $\mathsf{R}_0$ can ever prevent an instance of another rule $\mathsf{R}_1$, i.e., invalidate a match. This is written $\mathsf{R}_0 \# \mathsf{R}_1$ and $\#$ is called the *rule inhibition map.*

Instead of redex-reactum pairs, KaSim employs an alternative representation of reaction rules to make the static analyses for the rule activation and inhibition maps more fine-grained: rules consist of a redex and a set of actions describing how reaction modifies the redex. This allows the analyses to determine whether a rule actually modifies something in an agent that another rule may depend on.

#### Simulation Loop

The simulation loop consists of two steps: the Monte Carlo step and the update step. In the Monte Carlo step, two random choices are made, governed by the activity of the system and the rules respectively[1]: a time advance is generated and a reaction is chosen, i.e., a reaction rule $\mathsf{R}$ and an instance (i.e., match) of that rule. The update step is divided into three phases:

**1. negative update:**

The matches that will be invalidated by the chosen reaction are removed from the set of matches. The set of matches that may be invalidated can be narrowed down by exploiting two facts: (i) only a rule $\mathsf{R}'$ with $\mathsf{R} \# \mathsf{R}'$ may have matches that can be invalidated, and (ii) only matches that rely on a part of the agent that will be modified can be invalidated.

**2. rewrite:**

Rewrite the agent according to the chosen rule and match.

**3. positive update:**

Find new matches of redexes. Again, two facts enable us to reduce the search space: (i) only a rule $\mathsf{R}'$ with $\mathsf{R} \prec \mathsf{R}'$ may have new matches, and (ii) new matches must depend on a part of the agent that was just modified.

The key to the efficiency and scalability of the KaSim algorithm is the incremental and localized nature of these steps: the notion of modification allows us to only consider the modified part of the agent, the size of which depends only on the size of the redex and is thus independent of the size of the agent. The rule activation and inhibition maps reduce the dependency of the per-cycle simulation cost on the size of the set of rules compared to the naive approach where all rules must be considered in both the negative and positive update phases (though in the worst case all rules are related by $\#$ and $\prec$).

### 2.3.2 Towards Scalable Simulation of Stochastic Bigraphs

#### Our Approach

While the KaSim algorithm is conceptually simple, it relies on a number of concepts that have not previously been developed for bigraphs: embeddings that are equivalent to matches, localized

[1] The details of these random choices are irrelevant for this presentation; the full details are available in Section 7.3.

matching, actions, and causality analysis at the level of rules. Furthermore, the dynamic theory of stochastic bigraphs is not easily implementable for two reasons:

- it requires support, i.e., concrete bigraphs, for counting matches but is otherwise indifferent to support in the sense that all other definitions close under support equivalence, and

- parametric reaction rules are treated as generators of infinite families of ground (non-parametric) reaction rules, which clearly cannot be represented directly in an implementation.

In the technical report in Chapter 7 we address these topics in order to generalize KaSim to bigraphs. Our work on the causality analyses for rules is not quite complete, but we give a detailed outline of the approach and pinpoint the required results. The completed parts have been implemented in a prototype called the *stochastic bigraphical abstract machine (SBAM)* which currently allows stochastic simulation of certain BRSs: all controls must be active, reaction rules must be linear, and redexes must be solid (more on this below) and consist of a single connected component.

The following paragraphs summarize our work on the various parts of KaSim for bigraphs:

**Stochastic Parametric Reactive Systems** It shines through in Milner's presentation of the dynamic theory for bigraphs [Mil09] that the true aim was abstract bigraphs and that support was only added to the theory to enable the derivation of labels (cf. Section 1.2.2). All the definitions of the reaction semantics do their best to disregard support: either closure under support translation is explicitly assumed (the set of reaction rules), implicitly assumed (instantiation of parameters is only defined up to support equivalence), or explicitly ensured (the reaction relation and the generation of ground reaction rules from parametric rules). While this is fine in a setting where support is truly just a mathematical tool, it becomes problematic when implementing stochastic bigraphs where support must be handled explicitly in order to count matches. For that purpose, it must be clear where and how support translation should be applied. In their definition of stochastic bigraphs, Krivine et al. sidestep this problem by defining the stochastic semantics independently of the reaction semantics. However, this leads to a conceptual gap: there is no explicit connection between a redex occurrence leading to reaction (which is defined in terms of abstract bigraphs) and a redex occurrence in the stochastic semantics (defined in terms of concrete bigraphs).

Similarly, the treatment of parametric reaction rules is handled in a mathematically elegant and sound manner which is intractable in practice. As mentioned in Section 1.2.2, the dynamic theory of bigraphs is defined as an instance of the more general *reactive systems* which do not include a notion of parametric reaction rules. Instead, parametric bigraphical reaction rules are added to the theory by treating them as generators of infinite sets of ground reaction rules, cf. Section 1.2.2.

To obtain a solid formal foundation for our implementation, we therefore develop an alternative but equivalent presentation of the dynamic theory of bigraphs that is more amenable to direct implementation and unifies the reaction and stochastic semantics. More precisely, we develop what we call *stochastic parametric reactive systems (SPRSs)*, a generalization of reactive systems where support is handled explicitly, there are no unnecessary support translations, parametric reaction rules are first-class citizens, and the stochastic semantics are definitionally tied to the reaction semantics. We prove that when support is abstracted away, SPRSs have the same reactions as reactive systems. Thus, an abstract reactive system with a finite set of reaction rules can be directly, finitely, and faithfully represented and simulated as a concrete SPRS.

The stochastic semantics of SPRSs generalizes Krivine et al.'s stochastic semantics for bigraphs in three respects: (1) it is defined for parametric reaction rules, not just ground rules, (2) it allows non-linear reaction rules, and (3) it is defined at the more general level of reactive systems and is thus not bigraph specific.

**Bigraph Embeddings** Previous works have defined matches as decompositions of the agent into context, redex, and parameter (cf. e.g., Section 2.1.1) which is impractical to represent directly in a stochastic simulator. Instead, the KaSim algorithm relies on a representation of matches as graph embeddings: a structure preserving map from the entities of the redex to entities of the agent. These are more tractable as we need not construct the context and parameter explicitly.

We have defined embeddings for general link graphs, place graphs, and bigraphs and proven, for place graphs and bigraphs, that they are isomorphic to certain decompositions. In particular, an embedding $\phi : R \hookrightarrow a$ of a redex $R$ into an agent $a$ is isomorphic to a match. Furthermore, for an interesting class of bigraphs, those that are *solid*, we have shown that embeddings are determined by support translations of nodes. Solid bigraphs are interesting because many bigraphical models in the literature have solid redexes. For example, all BRSs in [Mil09, JM04, KMT08] have solid redexes.

Besides rather straightforward injectivity and structure preservation conditions, the key to ensuring correspondence between embeddings and decompositions is (a) to require that the embedding of an edge covers all its points and, similarly, that the embedding of a node covers all of its children, and (b) to require that a root is not mapped to a descendant of the embedding of a site. The latter requirement reflects the fact that there are no bigraph operations that can make one root of a bigraph a descendant of one of its other roots and thus such an embedding would have no corresponding decomposition. In other words, the condition expresses the fact that roots do not just model possibly disjoint locations, but subtrees that are disjoint.

**Bigraph Edit Scripts** The KaSim algorithm presumes a refined and precise causality analysis at the level of rules based on a notion of *modification*. In both $\kappa$ and bigraphs, the usual approach is to define reaction rules as redex-reactum pairs and to define reaction as the replacement of a redex by the corresponding reactum. In KaSim, however, it is assumed that a rule consists of a redex and a set of actions which describe how reaction modifies the redex. Reaction is achieved by letting an embedding of a redex mediate the corresponding actions to an agent.

We have taken a slightly different approach than the one employed for $\kappa$: instead of a *set* of actions, where each entity of the redex can be modified at most once, we use a *sequence* of actions. This is more in keeping with the tradition of graph theory from which we borrow the term *edit scripts* for sequences of actions which are also known as *edits* [Bil05]. We write $\Delta$ for edit scripts, $R' = \Delta(R)$ for the application of $\Delta$ to a redex $R$ resulting in reactum $R'$, and $(H', \phi') = \Delta(H, \phi)$ for the application of $\Delta$ to the bigraph $H$ mediated by the embedding $\phi : R \hookrightarrow H$ resulting in a bigraph $H'$ and an embedding $\phi' : R' \hookrightarrow H'$. We often abuse notation and write $\Delta(H, \phi)$ for $H'$.

We have defined a set of minimal edits for bigraphs and shown that edit scripts are sound and complete with respect to the reaction rules and reactions of the redex-reactum approach. In more detail, we have defined an alternative to SPRSs called *reconfiguration systems (RCSs)*. An RCS is equipped with so-called *reconfiguration rules* $(R, \Delta)$ consisting of a redex $R$ and an edit script $\Delta$. Reaction $a \twoheadrightarrow a'$ is defined as the application of an edit script mediated by an embedding $\phi : R \hookrightarrow a$ of the corresponding redex, i.e., $a' = \Delta(a, \phi)$. We have given constructions of reconfiguration rules from reaction rules and vice versa, and these induce constructions of RCSs from SPRSs and vice versa. Finally, we have proven that, when support is abstracted away, RCS and SPRSs have the same reactions.

Thus, reconfiguration systems provide an alternative but equivalent dynamic theory for abstract bigraphs with a notion of modification.

**Rule Activation and Inhibition** As discussed above, KaSim presumes the existence of static analyses of rules to decide whether one rule $\mathsf{R}_0$ may prevent or cause another rule $\mathsf{R}_1$ to be applied, resulting in the co-called *inhibition map* ($\mathsf{R}_0 \# \mathsf{R}_1$) and *activation map* ($\mathsf{R}_0 \prec \mathsf{R}_1$), repectively.

The KaSim paper does not actually define the inhibition and activation maps in terms of reactions, but in terms of embeddings and modifications. Assuming two linear reconfiguration

rules $\mathsf{R}_i = (R_i, \Delta_i)$ $(i = 0, 1)$ the two maps are defined as follows:

**inhibition map:** $\mathsf{R}_0 \# \mathsf{R}_1$ iff there is some agent $a$ and embeddings $\phi_i : R_i \hookrightarrow a$ such that $\mathsf{cod}(\phi_0) \cap \mathsf{cod}(\phi_1)$ contains at least one entity modified by $\mathsf{R}_0$.

**activation map:** $\mathsf{R}_0 \prec \mathsf{R}_1$ iff there is some agent $a$ and embeddings $\phi_0 : \Delta_0(R_0) \hookrightarrow a$, $\phi_1 : R_1 \hookrightarrow a$ such that $\mathsf{cod}(\phi_0) \cap \mathsf{cod}(\phi_1)$ contains at least one entity modified by $\mathsf{R}_0$.

It is implicitly assumed that these definitions imply that reactions generated by the related rules may be in conflict or causally related, respectively. While this is perhaps reasonable for $\kappa$, it requires a leap of faith and that leap only becomes larger when transferred to bigraphs which are more complex than $\kappa$. In fact, we are quite sure that these definitions are not quite right for bigraphs and bigraph embeddings. For example, given a bigraph embedding $\phi_1 : R_1 \hookrightarrow a$ it is possible to modify an entity in $\mathsf{cod}(\phi_1)$, resulting in agent $a'$, and still have the embedding $\phi_1 : R_1 \hookrightarrow a'$. Furthermore, the KaSim paper does not specify how to construct the inhibition and activation maps for $\kappa$.

In our work, we therefore take a different approach: we define the inhibition and activation maps in terms of reactions to ensure that they indeed characterize conflict and causality between reactions at the level of rules. Only then do we relate these definitions to embeddings and edit scripts, which we then use as a basis for a practical construction of the inhibition and activation maps based on pullback-pushout (PP) diagrams in the category of bigraph embeddings.

PP diagrams can be understood as characterizations of overlaps between redexes:

**pullbacks are overlaps:** Intuitively, the pullback of two embeddings $\phi_i : R_i \hookrightarrow a$ $(i = 0, 1)$ of redexes $R_0, R_1$ into the same bigraph $a$ is the sub-bigraph identified by the overlap of the two embeddings. Concretely, this sub-bigraph is given in the form of two embeddings $p_i : I \hookrightarrow R_i$ of a bigraph $I$ into each of the two redexes $R_0, R_1$ such that $\phi_0 \circ p_0 = \phi_1 \circ p_1$. When the embeddings do not overlap $I$ is the empty bigraph.

**pushouts are minimal examples:** Intuitively, the pushout of such a pullback is a minimal bigraph where the two redexes overlap as described by the pullback. Concretely, the pushout is a pair of embeddings $o_i : R_i \hookrightarrow H$ of the redexes into a minimal bigraph $H$ such that $o_0 \circ p_0 = o_1 \circ p_1$.

Together, we call a pullback and its pushout a PP diagram. We conjecture that for any two redexes there are a finite number of PP diagrams (up to iso on $I$ and $H$) and that these can be constructed. The foundation for this conjecture is that bigraphs are finite and it therefore seems very likely that two bigraphs can only overlap in a finite number of ways that may be enumerated.

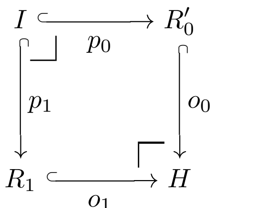
The approach requires that there are pullbacks and pushouts of pullbacks in the category of bigraph embeddings, i.e., the category where bigraphs are objects and embeddings are arrows. We argue in the paper that this is probably not the case if the category only contains embeddings that correspond to decompositions. However, by relaxing the conditions on embeddings slightly the category should have PP diagrams. In fact, we have a construction of PP diagrams in the category of link graph embeddings, but have yet to prove it correct and, due to time constraints, the construction is omitted from the paper.

Assuming we can construct them, PP diagrams should allow us to construct the inhibition and activation maps as follows. In the case of inhibition, assume two rules $\mathsf{R}_0, \mathsf{R}_1$ and a PP diagram for their redexes as defined above. Though it is not ground, the pushout $H$ can be thought of as a minimal agent with a given overlap of two embeddings. Thus, if the reactions of $H$, created by the two rules and embeddings are conflicting, then we have proof that $\mathsf{R}_0 \# \mathsf{R}_1$. A similar argument applies to activation analysis.

**Anchored Matching** A pillar in the scalability of the KaSim algorithm is that, after reaction, new matches are only searched for in the parts of the agent that have been modified. In other words, KaSim requires a localized matching algorithm that only searches a subset of the agent. Such an algorithm has not yet been presented in the bigraph literature. Previously published matching algorithms, including our own in Chapter 3, find matches anywhere in the agent. These algorithms are useful for the initialization phase of KaSim, where all matches must be found, but it is unclear how to specialize them to local matching.

The approach to localized matching for $\kappa$ in [DFFK07] is to construct all partial embeddings of a redex into modified parts of the agent and then see if they can be extended to total embeddings; we call this *anchored matching*. A similar approach is feasible for bigraphs, but we propose a refinement which exploits the construction of the activation map as discussed in the previous paragraph, by exploiting the following observations:

- Edit scripts are defined such that whenever a rule $\mathsf{R}_0 = (R_0, \Delta_0)$ is applied to an agent, we obtain both a new agent $a'$ and an embedding $\phi_0 : R'_0 \hookrightarrow a'$ of its reactum $R'_0 = \Delta_0(R_0)$ into that agent.
- For each rule $\mathsf{R}_1$ that is activated by $\mathsf{R}_0$, i.e., $\mathsf{R}_0 \prec \mathsf{R}_1$, we have a set of PP diagrams on the form




- Given such a PP diagram, if the map $\phi_0 \circ o_0^{-1}$ is a partial embedding of $H$ into $a'$, we can apply anchoring matching to find all total embeddings of $H$, $\phi_H : H \hookrightarrow a'$, satisfying $\phi_H \restriction_{\mathsf{rng}(o_0)} = \phi_0 \circ o_0^{-1}$.
- By composing with $o_1$, any embedding $\phi' : H \hookrightarrow a'$ of $H$ becomes an embedding of $R_1$, $\phi' \circ o_1 : R_1 \hookrightarrow a$, and thus we have found a match of $R_1$.

We believe that this approach will improve matching efficiency considerably in many cases, since it does not simply generate partial embeddings ad-hoc but precomputes the relevant ones.

### Discussion and Future Work

Overall, our work indicates that it is indeed possible to generalize the KaSim algorithm to stochastic bigraphs, at least for linear reaction rules. In fact, if we disregard the inhibition and activation maps, which are not strictly necessary, our results give a solid formal simulation algorithm for stochastic bigraphs with localized and incremental updates at each simulation step. The efficiency of the algorithm, however, has yet to be determined, both formally and empirically.

The individual developments in our work seem general enough that they should find applications outside our stochastic simulator. For example, we expect that our work on bigraph embeddings can help bridge the gap between bigraphs and graph rewriting, which Milner and Ehrig have already explored to some extent [Ehr02, Mil05]. They in particular focused on the fact that in the traditional graph rewriting approach, graphs are objects in a category whereas they are morphisms in the usual bigraphical approach. Following ideas by Cattani [CLM00] and Sobocinsky [Sob02] they use the cospan construction to turn objects into morphisms, and the coslice construction to turn morphisms into objects, thereby enabling transfer of results. Our category of bigraph embeddings, which substantially generalizes and corrects Milner's category of ground link graph

embeddings [Mil05], is a more direct solution to creating a bigraphical category where bigraphs are objects.

Our SPRSs are related to, and their formulation inspired by, the *parametric reactive systems* of Debois, where parametric reaction rules are also first-class citizens [Deb11]. However, contrary to our formulation, Debois does not make explicit that context and parameter may be connected without the involvement of the redex. This has the consequence that bigraphical reaction rules become generators of infinite families of rules. Furthermore, we go further than Debois, by formally showing that our formulation is equivalent to the usual (non-parametric) reactive systems.

Our set of edits are, while sound and complete, not necessarily optimal for all (or any) purposes. They were chosen because they have simple definitions, but it would be interesting to explore other options and their implications. For instance, we only allow nodes to be deleted if they have no children whereas in the context of the tree edit distance problem one usually allows this; the children are then simply moved to the parent [Bil05].

Our construction of edit scripts from reaction rules is rather naive: it essentially removes everything in the redex and then builds up the reactum, which clearly eliminates the benefits of edit scripts. It could be interesting to explore whether one can automatically derive better edit scripts. For example, the $\kappa$-calculus implementation uses a convention where the common prefix of redex and reactum are assumed to be unchanged by reaction. Alternatively, one could perhaps assign costs to the edits and apply optimization techniques to derive optimal edit scripts as in the tree edit distance problem [Bil05].

In his book [Mil09], Milner outlined an alternative way of specifying the relation between entities of redex and reactum that we achieve through edit scripts: he proposed to add a *tracking map* to each reactum rule, which specifies the relation between identities in the redex and reactum. While our approach is more complicated, it yields an elegant way to construct the inhibition and activation maps and is more suitable for an efficient implementation in the sense that an edit script constitutes a small-step recipe for reconfiguring the agent, whereby one avoids replacing the entire redex with the reactum as in Milner's approach.

Our work leaves some questions open:

- Does the category of bigraph embeddings have PP diagrams?
- Is there a finite set of PP diagrams for any pair of bigraphs?
- Is it sufficient to consider PP diagrams to construct the inhibition and activation maps?
- Is the anchored matching algorithm sound and complete?

We argue in the report that the answer to all of these questions is probably *yes*, but this will have to be shown formally.

If our PP diagram approach to characterizing causality and conflict at the level of rules is correct, it seems that it is a rather general and powerful tool. Besides enabling more efficient simulation, it will for example also provide modelers with valuable insight into the potential interaction of their rules. We also expect that the approach can be used for the dual purpose: characterizing parallel and sequential independence [Roz97, EKMR99].

In parallel with our work, Perrone et al. [PDH12] have given an algorithm for approximating the activation map (they call it *causation*) for reaction rules where redex and reactum are prime and contain at least one node. This allows them to significantly reduce the search space in their model checker for bigraphs [PDH12]. Our approach should give more precise results and thus enable more efficient model checking.

## 2.4 A Bigraphical Language for Cell Biology (Part V)

As discussed in the previous section, our motivation for building an efficient and scalable simulator for stochastic bigraphs is to enable simulation of models of cellular biology. Works by Damgaard

et al. [DDK08] and Bacci et al. [BGM09a] have already demonstrated that languages for such models can be based on bigraphs. Their languages essentially extend the $\kappa$-calculus [DL04] with (dynamic) compartments as in BioAmbients [RPS$^+$04] by exploiting the notions of locality and parametric reaction rules found in bigraphs. However, their languages do not require the full generality of bigraphs, though it is unclear from their developments how much of the complexity of the bigraphical framework is actually necessary to extend the $\kappa$-calculus with (dynamic) compartments. Furthermore, using bigraphs as the framework for developing these languages cause their definitions and presentations to become quite complex, especially when compared to their $\kappa$-calculus [DL04] and BioAmbients [RPS$^+$04] counterparts which are defined in the usual style of process calculi. In our experience, this has the consequence that researchers, who are unfamiliar with bigraphs, are alienated from the otherwise good ideas these bigraphical languages contain. The study in Part V is an attempt at bridging this gap.

### 2.4.1 Formal Cellular Machinery

In the paper in Chapter 8, we present a process calculus approach to defining a bigraphical language for cell biology. The developed language allows a natural representation of the $\kappa$-calculus and in addition allows modeling of dynamic compartments and diffusion of molecules.

Our language unifies ideas from a number of calculi:

1. The cellular medium can be described as a graph, where nodes represent molecules and edges represent physical connections between the molecules [DL03, FBH05, AK07, JLNV11].
2. Languages with a natural notion of location of reaction can be used to represent cellular compartments [RPS$^+$04, PRC08, BMSMT06, KMT08, BGM09b].
3. Interactions between compartments and proteins or vesicle transformations can be described using local patches of membranes, without committing to any particular global curvature [DP04, Car08].
4. Laws governing interactions of molecular components can be engendered by a small set of generators [Car04].

#### Our Approach

We elide the bigraphical underpinnings and instead define a series of process calculi $\mathcal{C}_0$,$\mathcal{C}_1$,$\mathcal{C}_2$,$\mathcal{C}_3$ each of which adds a feature to its predecessor. This approach pinpoints the complexity that comes from each feature, such as dynamic compartments. The calculi have formal graphical notations which, through a number of conventions, refine and specialize that of bigraphs to make the notation more lightweight and convenient for the domain of cell biology.

The following list summarizes the characteristics of the four calculi. As a help to the reader of this dissertation and to ease the following discussion, we relate the various aspects of the calculi to bigraphs; the paper mostly leaves this connection implicit.

**$\mathcal{C}_0$:** an "untyped" calculus aimed at modeling protein-protein interactions. Proteins are modeled as a collection of *domains*, each representing a substring of the amino acids of the protein, tied together by a *backbone*. Each domain has a number of *(interaction) sites* which represent parts of the domain that may physically interact with other proteins.

The dynamics of protein interactions is presented as a small set of *generator* rules which express biologically sensible reactions. Modelers can refine and compose the generators but not change them. The generators allow sites to connect or disconnect, proteins to detach or attach a domain, and domains to degrade or be synthesized.

In terms of bigraphs, a protein is an edge (the backbone) connecting a collection of nodes (its domains) with ports (their sites[2]).

**$\mathcal{C}_1$:** a "typed" version of the calculus that allows modelers to attach information to domains. In $\mathcal{C}_0$, one cannot distinguish domains (except for the number of sites) which is insufficient from a biological perspective since domains have very different properties depending on the amino acid strings they represent. Instead of capturing this information in a signature or type system for the calculus, we opted to embed this information in the terms themselves in the form of *info-nodes* (for lack of a better word). This provides flexibility for modelers and makes it easy to specify partial knowledge. A domain can have any number of info-nodes attached, which can record information such as its name and its current 3D folding in space. This enables us to write rules that only apply to certain types of domains/proteins which are in a particular state.

The extension only adds two simple rule generators: the ability to add or remove an info-node from a domain.

Bigraphically, an info-node is a node where the control represents the information. Each domain and its info-nodes are connected to a shared edge.

$\mathcal{C}_1$ allows a straightforward representation of the $\kappa$-calculus which we illustrate through an example in the paper.

**$\mathcal{C}_2$:** an extension of the calculus with (dynamic) compartments. The extension has two aspects: (i) the addition of compartments to the cellular structure we can describe, and (ii) an extension to rules and transition semantics to handle the interaction of compartments.

(i) A compartment is simply a unary constructor that wraps a term, and has its own set of info-nodes.

(ii) In order to allow rules to describe compartment interaction, we extend redexes and reactums with two features:

- Named parameters that allow a rule to capture unspecified contents of compartments.
- A projective view of membranes: a redex consists of a *wide term* which is a sequence of terms, where adjacent terms must be separated by exactly one membrane in any context where the redex matches. In fact, the distance between two terms in the sequence (assuming distance 1 between adjacent terms) is the number of membranes that must separate the terms in a match of the redex.
  We capture this *projective distance* constraint by giving an inference system for constructing contexts that satisfies it, and show this to be system sound and complete. This inference system can be thought of as a matching algorithm, similarly to the matching algorithm discussed in Section 2.1.1.

We add rule generators that allow handling of trans-membrane proteins, membrane budding, compartment interaction, and diffusion between compartments. In the style of [DDK08], we represent connected compartments using so-called *channels*: two info-nodes, one in each of the compartments, connected by an edge. Diffusion is linear, i.e., we cannot discard or duplicate parameters.

Bigraphically, compartments are nodes, parameters are sites, and wide terms are wide bigraphs. The projective distance constraint for matches corresponds to a restriction to bigraphical contexts where the parents of sites in the place graph are disjoint and form a subtree where the top-most parent has at most two children, the others at most one (intuitively an inverted 'V').

[2]Interaction sites should not be confused with the sites of bigraphs.

$\mathcal{C}_3$**:** an extension with diffusion of molecules. In $\mathcal{C}_2$, it is possible for molecules to only partially diffuse, which makes little biological sense. We therefore extend rules with the means to essentially specify that a parameter consists of a single connected component, i.e., a molecule, which can then be moved as a whole. This is achieved by extending structural congruence such that it allows a connected component to be ad-hoc encapsulated by a *species* constructor. A rule may then capture connected components by wrapping parameters in a species constructor, whereby legal parameters are restricted to connected components. The same mechanism also allows modeling of reactions that occur between domains that are part of the same connected component such as the formation of intramolecular connections.

This extension replaces the unsafe diffusion rule generator of $\mathcal{C}_2$ with a safe alternative, adds a generator to handle diffusion for trans-membrane proteins, and a generator for creating intramolecular connections.

Bigraphically, this corresponds to a non-standard constraint on parameters: only parameters consisting of exactly one connected component are allowed.

We illustrate the expressivity of $\mathcal{C}_3$ by showing how it allows a simple and natural model of *receptor internalization*: a patch of a cell membrane with a trans-membrane protein (a receptor) buds inwards, capturing both receptor and any attached protein complex in a vesicle inside the cell.

### Discussion and Future Work

Our $\mathcal{C}$-calculi are closely related to the $\mathcal{C}$ language of Damgaard et al. [DDK08], and somewhat related to the biobigraphs of Bacci et al. [BGM09a]. To a large extent, Damgaard et al. limited the complexity of their presentation by basing it on a family of bigraphical calculi for biological systems they had developed for the occasion [DK08]. Though a vast improvement over a presentation based directly on bigraphs, even these calculi come with a cost of complexity due to their generality, as well as limitations as to how ideas can be expressed. In particular, these calculi relied on a very general extension to the bigraphical theory, namely arbitrary predicates on parameters, in order to address the safe diffusion issue we mentioned above.

By allowing ourselves the freedom of constructing our $\mathcal{C}$-calculi from scratch, instead of bringing the bigraphical framework into play from the beginning, we obtain a simpler and more modular theory, where it is clear which aspects of the language are responsible for which aspects of the complexity of its definition. On the other hand, we will later have to formally relate our calculi to the bigraphical framework in order to be able to apply its results and, in particular, the SBAM simulator to $\mathcal{C}_3$. However, we think of the bigraphical foundations of the $\mathcal{C}$-calculi as a technicality that modelers should not need to deal with; the generality and complexity of bigraphs seem to alienate many.

From the summary above, it should be clear that the standard formulation of bigraphs is insufficient to give a direct model of $\mathcal{C}_2$ and $\mathcal{C}_3$: we need a formulation that will allow us (a) to restrict reaction to certain contexts, and (b) allow some parameters to be required to be connected components. Let us discuss these aspects in some detail:

(a) The usual theory of bigraphs already contains a mechanism for restricting reaction to certain contexts, namely those that are *active*. This is a non-local condition, since it requires that all ancestral nodes of every site have active controls. The restriction we require is strictly local: only the parents of sites must satisfy a condition, as we discussed for $\mathcal{C}_2$ above.

In fact, this restriction eases the matching problem considerably. As Bacci et al. observed [BMR10], the intractability of place graph matching stems from the width of reaction rules. Since the rules of the $\mathcal{C}$-calculi are not truly wide, we expect that the corresponding place graph matching will be tractable. Of course, this restriction has no (direct) effect on link graph matching. However, the link graph structure of $\mathcal{C}$-calculi is quite restricted, so we

are hopeful that matching for $\mathcal{C}$-calculi is easier than in the general case and perhaps even tractable.

(b) Recall from our discussion of binding in the background section (cf. Section 1.2.3) that, by default, parameters are discrete, i.e., contain no edges, and if edges should be captured they must be designated as binding.

While this approach seems reasonable for the models of calculi that motivated the development of bigraphs, it is not very well suited for the $\mathcal{C}$-calculi where links represent physical connections. In this setting, it is more reasonable to require that parameters can only capture complete molecules. We believe that this aspect can be solved by simply preventing parameters from being connected to the context.

In addition, $\mathcal{C}_3$ has rules that require a parameter to consist of exactly one molecule. A direct approach to supporting this in the bigraphical framework would be to add an annotation to sites, indicating the requirements on the number of connected components allowed/required in parameters.

These aspects require further investigation. In particular, the formulations of matches/embeddings and stochastics will have to be adjusted.

On a related note, we believe that our approach to handling connected components in $\mathcal{C}_3$ by using structural congruence is novel and elegant. It has allowed us to separate the connected component requirement from the matching formulation, keeping both relatively simple.

Besides these rather technical aspects, our language should of course also be evaluated. We plan to model and simulate a number of well-understood cellular mechanisms, to asses the modeling convenience of the language as well as its accuracy.

## 2.5 Conclusion

In the previous sections, we have summarized and discussed six individual works, each testing some aspect of the thesis that bigraphs may be used as an executable foundation for realistic formal languages. In this section we draw some overall conclusions from these works with respect to that thesis.

**Bigraphs for Formal Languages**

Our works have demonstrated that it is possible to use bigraphs as a foundation for rather direct formalizations of some kinds of realistic formal languages. In particular, languages with natural notions of locality and connectivity as well as dynamic semantics based on interaction. The flexibility and modularity of bigraphs, which the freedom in choice of signature and reaction rules offers, allows natural, domain specific models, a trait which is also aided by the formal graphical notation.

At the same time, our works also point out some areas where the theory of bigraphs needs to be expanded in order to fully realize the potential as a foundation for realistic formal languages. Some of these have already been addressed in the literature, but still lack constructive treatments that will make them practical, in particular with respect to implementation. These areas include generalized binding, higher-order reaction rules, data and calculation, sortings, and restrictions on parameters/contexts.

Furthermore, our bigraphical models (as well as many of those in the literature) suffer from a high degree of complexity for the reader that is not an expert in bigraphs. We see a need for simpler and more modular means to describe bigraphical models which, supplemented by good computer tools, can lessen the steep learning curve.

### Simulation of Bigraphs

The problem of simulating bigraphs has turned out to be rather challenging. While we started implementing simulators for bigraphs five years ago, and others later followed suit, none of these simulators are yet sufficiently efficient to satisfactorily simulate realistic models. Partly, this is due to a lack of development resources, but mainly it is due to the NP-hardness of matching in bigraphs. Our term-based matching algorithm, though provable correct and implementable, was too naive to be efficient in practice.

It is therefore necessary to investigate how matching can be implemented such that it is efficient in practice. Other fields have algorithms that are efficient for many large instances of NP-hard problems, so perhaps it would be worthwhile to explore connections to such fields. An example is Sevegnani et al.'s work on using a SAT-solver for the matching problem [SUC10].

At the same time, our work indicates that some bigraphical models do not need the full generality of bigraphs, but adhere to certain restrictions on redexes and matches that could potentially make matching easier. Of course, this means a trade-off between efficiency and generality of a tool, but we deem that such a trade-off is necessary.

Finally, our work shows that it is possible to incrementally and locally update the set of matches during simulation, which we expect will significantly improve the efficiency of bigraph simulators. Our work on rule inhibition/activation analysis also indicate that one can reduce the simulation cost for models with many rules.

## 2.6 Future Work

In the summary above, and in the works themselves, we discuss numerous lines of future work. Many of those are natural continuations of the works themselves, while some address the overall development of bigraphs as an executable foundation for realistic formal languages. In this section, we summarize and organize the latter kind.

### Modeling Convenience

As a practical framework for modeling formal languages, bigraphs are still young. Much theory has been developed, which in principle provides many of the methods we called for in the previous section. Generalized binding [Mil09], higher-order reaction rules [HNO06b, BBD$^+$09], and rule tactics are already at a stage where they can be applied in practical modeling. However, other aspects need to be developed further before they become practical tools in bigraphical modeling.

For example, while Debois [Deb08] has treated sortings in great detail, his work does not provide guidance on how to specify sortings for realistic languages. We should like to investigate a language for describing sortings on bigraphs, which for instance should be able to capture the structural constraints of WS-BPEL in our bigraphical formalization.

Also, we would like to see the free calculi and calculational BRSs of Debois [Deb11] turned into a practical modeling tool, such that modelers will not have to reinvent the wheel for each model. We envision a library of free calculi for various kinds of data structures and a notation for conveniently embedding them into other BRSs. For example, one could imagine associating a data type with each control and then let nodes with that control carry data of the specified type, as in Greenhalgh's bigraphspace tool [Gre09]. Calculation on data could be specified as part of reaction rules, e.g., in the form of expressions over the values of the redex nested inside the reactum.

In a similar vein, we should also like to investigate how more general BRSs may be composed. For example, our HomeBPEL model is a non-trivial extension of our WS-BPEL model that extends and expands the signature and the set of rules. Perhaps this could be expressed as the composition of two BRSs? Or one could imagine other operations on BRSs that succinctly could express the extension of WS-BPEL to HomeBPEL.

Overall, it seems to us that a high-level language for specifying BRSs, incorporating the above ideas, would be useful.

### Analysis

An important motivation for defining formal semantics for a language in the first place is to enable formal analysis of the models the language can express, and this of course also applies to bigraphical languages.

Bundgaard and Sassone [BS06] and Elsborg et al. [EHS09] have explored type systems for bigraphical models of the $\pi$-calculus. It would be interesting to explore if their approaches generalize to more complex models such as the bigraphical languages of this dissertation. In particular, WS-BPEL invites the development of a variety of type systems such as types for variables and session types.

We expect that Perrone et al.'s recent developments on refinement for BRSs [PDH11] will be useful for bigraphical languages. It should allow us to relate concrete and abstract models of such languages and could make it easier to define and reason about certain aspects of models. For example, it would be interesting to investigate whether our WS-BPEL model is a refinement of a more abstract $\pi$-calculus-like language where session types are easier to define and to reason about.

Finally, our own work on rule inhibition/activation should provide valuable insight to modelers about how their rules may interact.

### Tool Support

In order for bigraphs to become a truly practical foundation for formal languages, computer tools are needed. Modeling assistance, simulation, and automatic analysis should all be implemented, preferably with a graphical user interface since the formal and intuitive graphical notation is a key trait of bigraphs.

Currently, there is a coordinated effort between the research groups of Marino Miculan and Thomas Hildebrandt on building a bigraph workbench. It aims to provide a unifying platform for the existing tools for bigraphs and to provide a graphical user interface for modeling and experimenting with bigraphs. We should like to integrate the BPL Tool and SBAM into this platform.

Regarding simulation, SBAM is still an early prototype and must be developed further before it is ready for general use. Besides the outstanding issue of rule inhibition/activation analysis, we think it could be worthwhile to explore whether the SAT-based approach of Sevegnani et al. [SUC10] could be applied to efficiently find the initial set of matches.

# Bibliography


[AK07] Oana Andrei and Hélène Kirchner. Graph rewriting and strategies for modeling biochemical networks. In *Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'07)*, pages 407–414, 2007.

[AM91] Andrew W. Appel and David B. MacQueen. Standard ML of New Jersey. In *Third International Symposium on Programming Language Implementation and Logic Programming (PLILP'91)*, pages 1–13, 1991.

[BBD+09] Lars Birkedal, Mikkel Bundgaard, Søren Debois, Davide Grohmann, and Thomas Hildebrandt. Higher-order contexts via games and the int-construction. Technical Report TR-2009-117, IT University of Copenhagen, January 2009.

[BDGM07] Lars Birkedal, Troels C. Damgaard, Arne John Glenstrup, and Robin Milner. Matching of bigraphs. *Electronic Notes in Theoretical Computer Science*, 175(4):3–19, 2007.

[BGH+08a] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing higher-order mobile embedded business processes with binding bigraphs. In *Proceedings of the 10th international conference on Coordination Models and Languages (COORDINATION'08)*, Lecture Notes in Computer Science, pages 83–99. Springer Verlag, 2008.

[BGH+08b] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing WS-BPEL and Higher Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool. Technical Report TR-2008-103, IT University of Copenhagen, 2008.

[BGM09a] Giorgio Bacci, Davide Grohmann, and Marino Miculan. Bigraphical models for protein and membrane interactions. In *Proceedings of the Third International Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC 2009)*, pages 3–18. EPTCS 11, 2009.

[BGM09b] Giorgio Bacci, Davide Grohmann, and Marino Miculan. A framework for protein and membrane interactions. In *Proceedings of the 3rd Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'09)*, pages 19–33, 2009.

[BHH08] Mikkel Bundgaard, Thomas Hildebrandt, and Espen Højsgaard. Seamlessly distributed & mobile workflow - or: The right processes at the right places. In *Proceedings of the 1st Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES)*, pages 64–69, June 2008.

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:217–239, June 2005.

[BMR10] Giorgio Bacci, Marino Miculan, and Romeo Rizzi. Finding a forest in a tree. 2010.

[BMSMT06] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. A calculus of looping sequences for modelling microbiological systems. *Fundamenta Informaticæ*, 72(1–3):21–35, 2006.

[BPL07] The Bigraphical Programming Languages Group. The BPL Tool. `http://www.itu.dk/research/pls/wiki/index.php/BPL_Tool`, 2007.

[BS06] Mikkel Bundgaard and Vladimiro Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of the 8th ACM SIGPLAN international conference on Principles and Practice of Declarative Programming 2006*, pages 1–12, 2006.

[Bun07] Mikkel Bundgaard. *Semantics of Higher-Order Mobile Embedded Resources and Local Names*. PhD thesis, IT University of Copenhagen, 2007.

[BZ07] Mario Bravetti and Gianluigi Zavattaro. Contract based multi-party service composition. In Farhad Arbab and Marjan Sirjani, editors, *Proceedings of the IPM International Symposium on Fundamentals of Software Engineering (FSEN'07)*, volume 4767 of *Lecture Notes in Computer Science*, pages 207–222. Springer Verlag, 2007.

[Car04] Luca Cardelli. Brane calculi - interactions of biological membranes. In *Computational Methods in Systems Biology*, pages 257–278. Springer, 2004.

[Car08] Luca Cardelli. Bitonal membrane systems - interactions of biological membranes. *Theoretical Computer Science*, 404(1-2), 2008.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. W3C Note, W3C, March 2001.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[CH09] Federica Ciocchetta and Jane Hillston. Bio-pepa: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009.

[CHY07] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *Proceedings of the 16th European Symposium on Programming (ESOP'07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer Verlag, 2007.

[CLM00] Gian Luca Cattani, James J. Leifer, and Robin Milner. Contexts and embeddings for closed shallow action graphs. Technical Report UCAM-CL-TR-496, University of Cambridge, Computer Laboratory, July 2000.

[CMS05] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Bigraphical logics for XML. In *Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems (SEBD'05)*, pages 392–399, 2005.

[DB06] Troels C. Damgaard and Lars Birkedal. Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1–2):58–77, 2006.

[DDK08] Troels C. Damgaard, Vincent Danos, and Jean Krivine. A language for the cell. Technical Report TR-2008-116, IT University of Copenhagen, December 2008.

[Deb08] Søren Debois. *Sortings & Bigraphs.* PhD thesis, IT University of Copenhagen, January 2008.

[Deb11] Søren Debois. Computation in the informatic jungle. Technical Report TR-2011-147, IT University of Copenhagen, 2011. (forthcoming).

[DF95] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24:873–921, August 1995.

[DFFK07] Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In *Proceedings of the 5th Asian conference on Programming languages and systems*, APLAS'07, pages 139–157. Springer-Verlag, 2007.

[DHK11] Troels C. Damgaard, Espen Højsgaard, and Jean Krivine. Formal cellular machinery. In *Proceedings of SASB 2011, the Second International Workshop on Static Analysis and Systems Biology*, September 2011. Keynote talk. (to appear).

[DK08] Troels C. Damgaard and Jean Krivine. A generic language for biological systems based on bigraphs. Technical Report TR-2008-115, IT University of Copenhagen, December 2008.

[DL03] Vincent Danos and Cosimo Laneve. Graphs for formal molecular biology. In *Proceedings of the 1st International Workshop on Computational Methods in Systems Biology (CMSB'03)*, volume 2602 of *LNCS*, pages 34–46, 2003.

[DL04] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325, 2004.

[DP04] Vincent Danos and Sylvain Pradalier. Projective brane calculus. In *Proceedings of the 2nd International Workshop on Computational Methods in Systems Biology (CMSB'04)*, pages 134–148, 2004.

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of graph grammars and computing by graph transformation: Volume 2: Applications, Languages, and Tools.* World Scientific Publishing Co., Inc., 1999.

[Ehr02] Hartmut Ehrig. Bigraphs meet double pushouts. *Bulletin of the EATCS*, 78:72–85, 2002.

[EHS09] Ebbe Elsborg, Thomas T. Hildebrandt, and Davide Sangiorgi. Type systems for bigraphs. In *Proceedings of the 4th International Symposium on Trustworthy Global Computing (TGC'08)*, pages 126–140, April 2009.

[EKMR99] Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors. *Handbook of graph grammars and computing by graph transformation, Volume 3: Concurrency, Parallelism, and Distribution.* World Scientific Publishing Co., Inc., 1999.

[Fah05] Dirk Fahland. Complete Abstract Operational Semantics for the Web Service Business Process Execution Language. Technical Report 190, Humboldt-Universität zu Berlin, 2005.

[Fai10] Alec Faithfull. Big Red. `http://www.itu.dk/research/pls/wiki/index.php/Big_Red`, 2010.

[FBH05] James R. Faeder, Mickael L. Blinov, and William S. Hlavacek. Rule based modeling of biochemical networks. *Complexity*, pages 22–41, 2005.

[FGV04] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. Specification and validation of the business process execution language for web services. In *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *Lecture Notes in Computer Science*, pages 78–94. Springer Verlag, 2004.

[FGV06] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. An abstract machine architecture for web service based business process management. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 144–157. Springer Verlag, 2006.

[FR05] Dirk Fahland and Wolfgang Reisig. ASM-based semantics for BPEL: The negative Control Flow. In Danièle Beauquier, Egon Börger, and Anatol Slissenko, editors, *Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05)*, pages 131–151. Paris XII, March 2005.

[GCDC06] Pablo Garralda, Adriana B. Compagnoni, and Mariangiola Dezani-Ciancaglini. BASS: Boxed ambients with safe sessions. In *Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'06)*, pages 61–72. ACM Press, 2006.

[GDBH10] Arne John Glenstrup, Troels Christoffer Damgaard, Lars Birkedal, and Espen Højsgaard. An implementation of bigraph matching. Technical Report TR-2010-135, IT University of Copenhagen, December 2010.

[GHR92] Norbert Götz, Ulrich Herzog, and Michael Rettelbach. TIPP – a language for timed processes and performance evaluation. report Technical Report 4/92, IMMD VII, University of Erlangen-Nurnberg, 1992.

[Gil76] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1990.

[Gre09] Chris Greenhalgh. bigraphspace. `http://bigraphspace.svn.sourceforge.net/`, 2009.

[HG11] Espen Højsgaard and Arne John Glenstrup. The BPL Tool: A tool for experimenting with bigraphical reactive systems. Technical Report TR-2011-145, IT University of Copenhagen, October 2011.

[HH11] Tim Hallwyl and Espen Højsgaard. Core BPEL: Semantic clarification of WS-BPEL 2.0 through syntactic simplification using XSL transformations. Technical Report TR-2011-138, IT University of Copenhagen, March 2011.

[HH12] Tim Hallwyl and Espen Højsgaard. Core BPEL: Syntactic simplification of WS-BPEL 2.0. In *Proceedings of SAC 2012, 27th ACM Symposium on Applied Computing*, March 2012. (to appear).

[Hil96] Jane Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[HK11] Espen Højsgaard and Jean Krivine. Towards scalable simulation of stochastic bigraphs. Technical Report TR-2011-148, IT University of Copenhagen, 2011.

[HM03] Kathrin Hoffmann and Till Mossakowski. Algebraic higher-order nets: Graphs and Petri nets as tokens. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Proceedings of the 16th International Workshop on Recent Trends in Algebraic Development Techniques (WADT'02)*, volume 2755 of *Lecture Notes in Computer Science*, pages 253–267. Springer Verlag, 2003.

[HNO06a] Thomas Hildebrandt, Henning Niss, and Martin Olsen. Business process execution languages as bigraphs and reactive XML. Technical Report TR-2006-85, IT University of Copenhagen, 2006.

[HNO06b] Thomas Troels Hildebrandt, Henning Niss, and Martin Olsen. Formalising business process execution with bigraphs and Reactive XML. In Paolo Ciancarini and Herbert Wiklicky, editors, *Proceedings of the 8th International Conference on Coordination Models and Languages (COORDINATION)*, volume 4038 of *Lecture Notes in Computer Science*, pages 113–129. Springer-Verlag, January 2006.

[Hp07] Thomas Hildebrandt (principal investigator). Computer supported mobile adaptive business processes (CosmoBiz) research project. Webpage, 2007. `http://www.cosmobiz.org`.

[HW05] Thomas Troels Hildebrandt and Jacob Wahl Winther. Bigraphs and (reactive) XML. Technical Report TR-2005-56, IT University of Copenhagen, January 2005.

[JLNV11] Mathias John, Cédric Lhoussaine, Joachim Niehren, and Cristian Versari. Biochemical reaction rules with constraints. In *Proceedings of the 20th European Symposium on Programming (ESOP'11)*, volume 6602 of *LNCS*, pages 338–357, 2011.

[JM04] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge – Computer Laboratory, February 2004.

[KMT08] Jean Krivine, Robin Milner, and Angelo Troina. Stochastic bigraphs. *Electronic Notes in Theoretical Computer Science*, 218:73 – 96, 2008. Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV).

[Lan64] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, pages 308–320, January 1964.

[Lan65a] Peter J. Landin. A correspondence between ALGOL 60 and church's lambda-notation: Part I. *Communications of the ACM*, 8:89–101, February 1965.

[Lan65b] Peter J. Landin. A correspondence between ALGOL 60 and church's lambda-notation: Part II. *Communications of the ACM*, 8:158–167, March 1965.

[Laz02] Yuri Lazebnik. Can a biologist fix a radio? – or, what i learned while studying apoptosis. *Cancer Cell*, 2(3):179–182, September 2002.

[Loh07] Niels Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In Marlon Dumas and Reiko Heckel, editors, *Proceedings of the 4th International Workshop on Web Services and Formal Methods (WS-FM'07)*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer Verlag, 2007.

[LPT06] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A WSDL-based type system for WS-BPEL. In Paolo Ciancarini and Herbert Wiklicky, editors, *Proceedings of the 8th international conference on Coordination Models and Languages (COORDINATION'06)*, volume 4038 of *Lecture Notes in Computer Science*, pages 145–163. Springer Verlag, 2006.

[LS03] Francesca Levi and Davide Sangiorgi. Mobile safe ambients. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 25(1):1–69, 2003.

[LVO+07] Niels Lohmann, Eric Verbeek, Chun Ouyang, Christian Stahl, and Wil M. P. van der Aalst. Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology, 2007.

[Mil99] Robin Milner. *Communicating and mobile systems: the π-calculus.* Cambridge University Press, 1999.

[Mil05] Robin Milner. Embeddings and contexts for link graphs. In Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 343–351. Springer Berlin / Heidelberg, 2005.

[Mil09] Robin Milner. *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Journal of Information and Computation*, 100:1–40 and 41–77, 1992.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *The definition of Standard ML.* MIT Press, Cambridge, MA, USA, 1990.

[MY07] Dimitris Mostrous and Nobuko Yoshida. Two session typing systems for higher-order mobile processes. In Simona Ronchi and Della Rocca, editors, *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications (TLCA'07)*, volume 4583 of *Lecture Notes in Computer Science*, pages 321–335. Springer Verlag, 2007.

[PDH11] Gian Perrone, Søren Debois, and Thomas Hildebrandt. Bigraphical refinement. In *Proceedings of the 15th International Refinement Workshop (Refine'11)*, pages 20–36, June 2011.

[PDH12] Gian Perrone, Søren Debois, and Thomas Hildebrandt. A model checker for bigraphs. In *Proceedings of the ACM Symposium on Applied Computing - Software Verification and Tools Track 2012 (ACM SAC-SVT 2012)*, 2012. (to appear).

[PRC08] Gheorghe Păun and Francisco J. Romero-Campero. Membrane computing as a modeling framework. Cellular systems case studies. In *Formal Methods for Computational Systems Biology*, volume 5016 of *LNCS*, pages 168–214, 2008.

[Pri95] Corrado Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation, Volume 1: Foundations.* World Scientific Publishing Co., Inc., 1997.

[RPS+04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science*, 325:141–167, 2004.

[RtHvdAM06] Nick Russell, Arthur H.M. ter Hofstede, Will M.P. van der Aalst, and Nataliya Mulyar. Workflow control-flow patterns: A revised view. BPM Center Report BPM-06-22, BPMcenter.org, 2006.

[San93] Davide Sangiorgi. From $\pi$-calculus to higher-order $\pi$-calculus - and back. In *Fourth International Conference on the Theory and Practice of Software Development (TAPSOFT'93)*, pages 151–166, 1993.

[Sob02] Pawel Sobocinsky. Relative pushouts in graphical reactive systems. February 2002.

[ST99] Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.

[Sta05] Christian Stahl. A Petri net semantics for BPEL. Informatik-Berichte 188, Humboldt-Universität zu Berlin, 2005.

[SUC10] Michele Sevegnani, Chris Unsworth, and Muffy Calder. A SAT based algorithm for the matching problem in bigraphs with sharing. Technical Report TR-2010-311, University of Glasgow, Department of Computing Science, 2010.

[WDW07] Matthias Weidlich, Gero Decker, and Mathias Weske. Efficient analysis of BPEL 2.0 processes using $\pi$-calculus. In *Asia-Pacific Service Computing Conference, The 2nd IEEE*, pages 266–274, December 2007.

[WS-07] Web services business process execution language version 2.0, April 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[XPa99] XML path language (XPath) version 1.0, 1999. `http://www.w3.org/TR/xpath/`.

[XSL99] XSL transformations (XSLT) version 1.0, 1999. `http://www.w3.org/TR/xslt`.

# Part II

# A Tool for Bigraphical Programming Languages

# Chapter 3

# An Implementation of Bigraph Matching

**Arne J. Glenstrup, Troels C. Damgaard, Lars Birkedal, and Espen Højsgaard**

**Abstract**

We describe a provably sound and complete matching algorithm for bigraphical reactive systems. The algorithm has been implemented in our BPL Tool, a first implementation of bigraphical reactive systems. We describe the tool and present a concrete example of how it can be used to simulate a model of a mobile phone system in a bigraphical representation of the polyadic $\pi$ calculus.

**Preface** This chapter consists of the technical report

A. J. Glenstrup, T. C. Damgaard, L. Birkedal, and E. Højsgaard. *An Implementation of Bigraph Matching.* Technical Report TR-2010-135, IT University of Copenhagen, December 2010.

## 3.1 Introduction

The theory of bigraphical reactive systems [13] provides a general meta-model for describing and analyzing mobile and distributed ubiquituous systems. Bigraphical reactive systems form a graphical model of computation in which graphs embodying both locality and connectivity can be reconfigured using *reaction rules*. So far it has been shown how to use the theory for recovering behavioural theories for various process calculi [12, 13, 15] and how to use the theory for modelling context-aware systems [2].

In this paper we describe the core part of our BPL Tool, a first prototype implementation of bigraphical reactive systems, which can be used for experimenting with bigraphical models.

The main challenge of implementing the dynamics of bigraphical reactive systems is the *matching problem*, that is, to determine for a given bigraph and reaction rule whether and how the reaction rule can be applied to rewrite the bigraph. When studying the matching problem in detail, one finds that it is a surprisingly tricky problem (it is related to the NP-complete graph embedding problem). Therefore we decided early on to study the matching problem quite formally and base our prototype implementation on a provably correct specification. In previous work [1, 9],

we gave a sound and complete inductive characterization of the matching problem for bigraphs. Our inductive characterization was based on normal form theorems for binding bigraphs [8].

In the present paper we extend the inductive characterization from graphs to a *term* representation of bigraphs. A single bigraph can be represented by several structurally congruent bigraph terms. Using an equational theory for bigraph terms [8], we essentially get a non-deterministic matching algorithm operating on bigraph terms. However, such an algorithm will be wildly nondeterministic and we thus provide an alternative, but still provably sound and complete, characterization of matching on terms, which is more suited for mechanically finding matching. In particular, it spells out how and where to make use of structural congruences.

We have implemented the resulting algorithm in our BPL Tool, which we briefly describe in Section 3.8. We also present an example of a bigraphical reactive system, an encoding of the polyadic $\pi$ calculus, and show how it can be used to simulate a simple model of a mobile phone system.

Bigraphical reactive systems are related to general graph transformation systems; Ehrig et al. [10] provide a recent comprehensive overview of graph transformation systems. In particular, bigraph matching is related to the general graph pattern matching (GPM) problem, so general GPM algorithms might also be applicable to bigraphs [11, 14, 20, 21]. As an alternative to implementing matching for bigraphs, one could try to formalize bigraphical reactive systems as graph transformation systems and then use an existing implementation of graph transformation systems. Some promising steps in this direction have been taken [19], but they have so far fallen short of capturing precisely all the aspects of binding bigraphs. For a more detailed account of related work, in particular on relations between BRSs, graph transformations, term rewriting and term graph rewriting, see the Thesis of Damgaard [7, Section 6].

The remainder of this paper is organized as follows. In Section 3.2 we give an informal presentation of bigraphical reactive systems and normalisation techniques needed for the implementation. In Section 3.3 we recall the graph-based inductive characterization, then in Section 3.4 we develop a term-based inductive characterization, forming the basis for our implementation of matching. Section 3.5 explains how we can restrict the kind of inferece trees the algorithm needs to consider, without sacrificing completeness; this is then used in Section 3.6, where we describe how to translate the inference system into a working algorithm. We discuss how to handle nondeterminism in Section 3.7, and in Section 3.8 we describe the BPL Tool and present an example use of it. Finally, we conclude and discuss future work in Section 3.9.

## 3.2 Bigraphs and Reactive Systems

In the following, we present bigraphs informally; for a formal definition, see the work by Jensen and Milner [13] and Damgaard and Birkedal [8].

### 3.2.1 Concrete Bigraphs

A concrete binding bigraph $G$ consists of a *place graph* $G^{\mathbf{P}}$ and a *link graph* $G^{\mathbf{L}}$. The place graph is an ordered list of trees indicating *location*, with roots $r_0, \ldots, r_n$, nodes $v_0, \ldots, v_k$, and a number of special leaves $s_0, \ldots, s_m$ called *sites*, while the link graph is a general graph over the node set $v_0, \ldots, v_k$ extended with *inner names* $x_0, \ldots, x_l$, and equipped with hyper edges, indicating *connectivity*.

We usually illustrate the place graph by nesting nodes, as shown in the upper part of Figure 3.1 (ignore for now the interfaces denoted by " $: \cdot \to \cdot$"). A *link* is a hyper edge of the link graph, either an internal *edge* $e$ or a *name* $y$. Links that are names are called *open*, those that are edges are called *closed*. Names and inner names can be *global* or *local*, the latter being located at a specific root or site, respectively. In Figure 3.1, $y_0$ is located at $r_0$, indicated by a small ring, and $x_0$ and $x_2$ are located at $s_2$, indicated by writing them within the site. Global names like $y_1$ and $y_2$ are

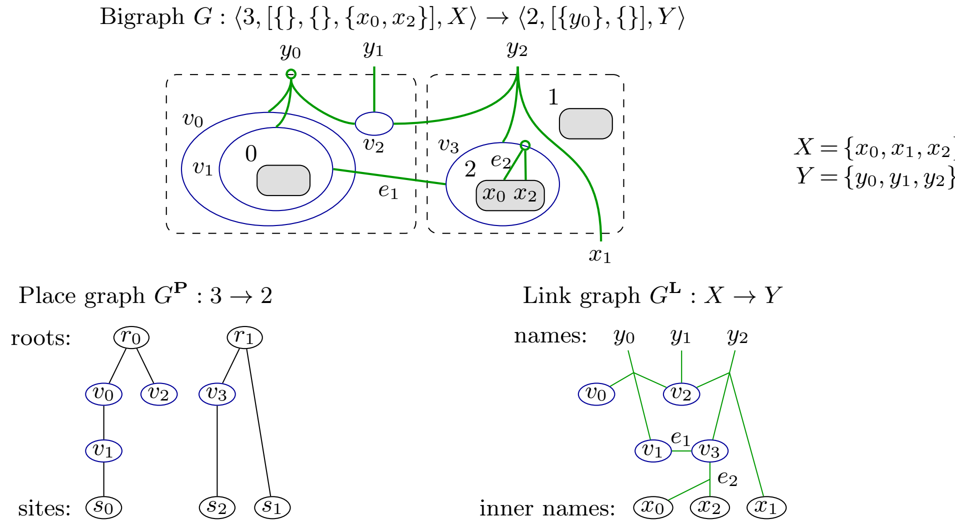



Figure 3.1: Example bigraph illustrated by nesting and as place and link graph.

drawn anywhere at the top, while global inner names like $x_1$ are drawn anywhere at the bottom. A link, including internal edges like $e_2$ in the figure, can be located with one *binder* (the ring), in which case it is a *bound link,* otherwise it is *free*. However, a bound link must satisfy the *scope rule*, a simple structural requirement that all points (cf. next paragraph) of the link lie within its location (in the place graph), except for the binder itself. This prevents $y_2$ and $e_1$ in the example from being bound.

### 3.2.2 Controls and Signatures

Every node $v$ has a *control* $K$, indicated by $v : K$, which determines a binding and free arity. In the example of Figure 3.1, we could have $v_i : K_i, i = 0, 1, 2, 3,$ where arities are given by $K_0 : 1$, $K_1 : 2$, $K_2 : 3$, $K_3 : 1 \to 2$, using $K : f$ as a shorthand for $K : 0 \to f$. The arities determine the number of bound and free *ports* of the node, to which bound and free links, respectively, are connected. Ports and inner names are collectively referred to as *points*.

In addition to arity, each control is assigned a *kind*, either atomic, active or passive, and describe nodes according to their control kinds. We require that atomic nodes contain no nodes except sites; any site being a descendant of a passive node is *passive*, otherwise it is *active*. If all sites of a bigraph $G$ are active, $G$ is *active*.

A collection of controls with their associated kinds and arities is referred to as a *signature*.

### 3.2.3 Abstract Bigraphs

While concrete bigraphs with named nodes and internal edges are the basis of bigraph theory [13], our prime interest is in *abstract bigraphs,* equivalence classes of concrete bigraphs that differ only in the names of nodes and internal edges[1]. Abstract bigraphs are illustrated with their node controls (see Figure 3.14 in Section 3.8). In what follows, “bigraph” will thus mean “abstract bigraph.”

[1]Formally, we also disregard *idle* edges: edges not connected to anything.

### 3.2.4 Interfaces

Every bigraph $G$ has two *interfaces* $I$ and $J$, written $G : I \to J$, where $I$ is the *inner face* and $J$ the *outer face*. An interface is a triple $\langle m, \vec{X}, X \rangle$, where $m$ is the *width* (the number of sites or roots), $X$ the entire set of local and global names, and $\vec{X}$ indicates the locations of each local name, cf. Figure 3.1. We let $\epsilon = \langle 0, [], \{\} \rangle$; when $m = 1$ the interface is *prime*, and if all $x \in X$ are located by $\vec{X}$, the interface is *local*. As in the work by Milner [18] we write $G : \to J$ or $G : I \to$ for $G : I \to J$ when we are not concerned about about $I$ or $J$, respectively.

A bigraph $G : I \to J$ is called *ground*, or an *agent*, if $I = \epsilon$, *prime* if $I$ is local and $J$ prime, and a *wiring* if $m = n = 0$, where $m$ and $n$ are the widths of $I$ and $J$, respectively. For $I = \langle m, \vec{X}, X \rangle$, bigraph $\mathsf{id}_I : I \to I$ consists of $m$ roots, each root $r_i$ containing just one site $s_i$, and a link graph linking each inner name $x \in X$ to name $x$.

### 3.2.5 Discrete and Regular Bigraphs

We say that a bigraph is *discrete* iff every free link is a name and has exactly one point. The virtue of discrete bigraphs is that any connectivity by internal edges must be bound, and node ports can be accessed individually by the names of the outer face. Further, a bigraph is *name-discrete* iff it is discrete and every bound link is either an edge, or (if it is a name) has exactly one point. Note that name-discrete implies discrete.

A bigraph is *regular* if, for all nodes $v$ and sites $i, j, k$ with $i \leq j \leq k$, if $i$ and $k$ are descendants of $v$, then $j$ is also a descendant of $v$. Further, for roots $r_{i'}$ and $r_{j'}$, and all sites $i$ and $j$ where $i$ is a descendant of $r_{i'}$ and $j$ of $r_{j'}$, if $i \leq j$ then $i' \leq j'$. The bigraphs in the figures are all regular, the permutation in Table 3.1 is not. The virtue of regular bigraphs is that permutations can be avoided when composing them from basic bigraphs.

### 3.2.6 Product and Composition

For bigraphs $G_1$ and $G_2$ that share no names or inner names, we can make the *tensor product* $G_1 \otimes G_2$ by juxtaposing their place graphs, constructing the union of their link graphs, and increasing the indexes of sites in $G_2$ by the number of sites of $G_1$. We write $\bigotimes_i^n G_i$ for the iterated tensor $G_0 \otimes \cdots \otimes G_{n-1}$, which, in case $n = 0$, is $\mathsf{id}_\epsilon$.

The *parallel product* $G_1 \parallel G_2$ is like the tensor product, except global names can be shared: if $y$ is shared, all points of $y$ in $G_1$ and $G_2$ become the points of $y$ in $G_1 \parallel G_2$.

The *prime product* $G_1 \,|\, G_2$ is like the parallel product, except the result has just one root (also when $G_1$ and $G_2$ are wirings), produced by merging any roots of $G_1$ and $G_2$ into one.

We can *compose* bigraphs $G_2 : I \to I'$ and $G_1 : I' \to J$, yielding bigraph $G_1 \circ G_2 : I \to J$, by plugging the sites of $G_1$ with the roots of $G_2$, eliminating both, and connecting names of $G_2$ with inner names of $G_1$. In the following, we will omit the '$\circ$', and simply write $G_1 G_2$ for composition, letting it bind tighter than tensor product.

### 3.2.7 Notation, Basic Bigraphs, and Abstraction

In the sequel, we will use the following notation: $\uplus$ denotes union of sets required to be disjoint; we write $\{\vec{Y}\}$ for $Y_0 \uplus \cdots \uplus Y_{n-1}$ when $\vec{Y} = Y_0, \ldots Y_{n-1}$, and similarly $\{\vec{y}\}$ for $\{y_0, \ldots, y_{n-1}\}$. For interfaces, we write $n$ to mean $\langle n, [\emptyset, \ldots, \emptyset], \emptyset \rangle$, $X$ to mean $\langle 0, [], X \rangle$, $\langle X \rangle$ to mean $\langle 1, [\{\}], X \rangle$ and $(X)$ to mean $\langle 1, [X], X \rangle$.

Any bigraph can be constructed by applying composition, tensor product and abstraction to identities (on all interfaces) and a set of basic bigraphs, shown in Table 3.1 [8]. For permutations, when used in any context, $\pi_{\vec{X}} G$ or $G \pi_{\vec{X}}$, $\vec{X}$ is given entirely by the interface of $G$; in these cases we simply write $\pi_{\vec{X}}$ as $\pi$.

| | Notation | Example |
|---|---|---|
| Merge | $merge_n : n \to 1$ | $merge_3 =$ |
| Concretion | $\ulcorner X \urcorner : (X) \to \langle X \rangle$ | $\ulcorner \{x_1, x_2\} \urcorner =$ |
| Abstraction | $(Y)P : I \to \langle 1, [Y], Z \uplus Y \rangle$ | $(\{y_1, y_2\})(\{y_3\})\ulcorner \{y_1, y_2, y_3, z\} \urcorner =$ |
| Substitution $\sigma$ | $\vec{y}/\vec{X} : X \to Y$ | $[y_1, y_2, y_3]/[\{x_1, x_2\}, \{\}, \{x_3\}] =$ |
| Renaming $\alpha, \beta$ | $\vec{y}/\vec{x} : X \to Y$ | $[y_1, y_2, y_3]/[x_1, x_2, x_3] =$ |
| Closure | $/X : X \to \{\}$ | $/\{x_1, x_2, x_3\} =$ |
| Wiring $\omega$ | $(\mathsf{id} \otimes /Z)\sigma : X \to Y$ | $(\mathsf{id}_{\{y1,y2\}} \otimes /\{z_1, z_2\})$ $[y_1, z_1, y_2, z_2]\,/$ $[\{\}, \{x_1, x_2\}, \{x_3, x_4\}, \{x_5\}] =$ |
| Ion | $K_{\vec{y}(\vec{X})} : (\{\vec{X}\}) \to \langle \{\vec{y}\} \rangle$ | $\mathsf{K}_{[y_1,y_2]([\{x_1\},\{x_2,x_3\},\{\}])} =$ |
| Permutation $\pi_{\vec{X}}$ | $\{i \mapsto j, \ldots\} : \langle m, \vec{X}, X \rangle \to \langle m, \pi(\vec{X}), X \rangle$ | $\{0 \mapsto 2, 1 \mapsto 0, 2 \mapsto 1\}_{[\{x\},\emptyset,\{y\}]} =$ |

Table 3.1: Basic bigraphs, the abstraction operation, and variables ranging over bigraphs.

Given a prime $P$, the abstraction operation localises a subset of its outer names. Note that the scope rule is necessarily respected since the inner face of a prime $P$ is required to be local, so all points of $P$ are located within its root. The abstraction operator is denoted by $(\cdot)\cdot$ and reaches as far right as possible.

For a renaming $\alpha : X \to Y$, we write $\ulcorner\alpha\urcorner$ to mean $(\alpha \otimes \mathsf{id}_1)\ulcorner X\urcorner$, and when $\sigma : U \to Y$, we let $\widehat{\sigma} = (Y)(\sigma \otimes \mathsf{id}_1)\ulcorner U\urcorner$. We write substitutions $\vec{y}/[\emptyset, \ldots, \emptyset] : \epsilon \to Y$ as $Y$.

Note that $[]/[] = /\emptyset = \pi_0 = \mathsf{id}_\epsilon$ and $merge_1 = \ulcorner\emptyset\urcorner = \pi_1 = \mathsf{id}_1$, where $\pi_i$ is the nameless permutation of width $i$.

### 3.2.8 Bigraphical Reactive Systems

Bigraphs in themselves model two essential parts of context: locality and connectivity. To model also *dynamics,* we introduce *bigraphical reactive systems* (BRS) as a collection of *rules.* Each rule $R \xrightarrow{\varrho} R'$ consists of a regular *redex* $R : I \to J$, a *reactum* $R' : I' \to J$, and an *instantiation* $\varrho$, mapping each site of $R'$ to a site of $R$, and mapping local names in $I'$ to those of $I$, as illustrated in Figure 3.2. Interfaces $I = \langle m, \vec{X}, X\rangle$ and $I' = \langle m', \vec{X}', X'\rangle$ must be local, and are related

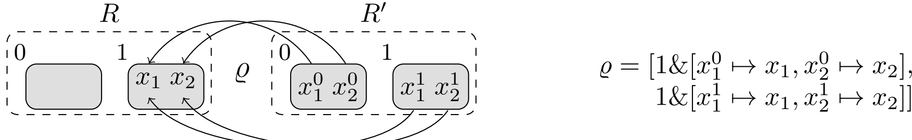



Figure 3.2: A reaction rule

by $X'_i = X_{\varrho(i)}$, where $\varrho$ must be a bijection between $X'_i$ and $X_{\varrho(i)}$. We illustrate $\varrho$ by '$i := j$', whenever $\varrho(i) = j \neq i$, or, alternatively, by listing $[\varrho(0), \ldots, \varrho(m' - 1)]$. Given an instantiation $\varrho$ and a discrete bigraph $d = d_0 \otimes \cdots \otimes d_k$ with prime $d_i$'s, we let $\varrho(d) = d_{\varrho(0)} \otimes \cdots \otimes d_{\varrho(k)}$, allowing copying, discarding and reordering parts of $d$.

Given an agent $a$, a *match* of redex $R$ is a decomposition $a = C(\mathsf{id}_Z \otimes R)d$, with active context $C$ and discrete parameter $d$ with its global names $Z$. Dynamics is achieved by transforming $a$ into a new agent $a' = C(\mathsf{id}_Z \otimes R')d'$, where $d' = \varrho(d)$, cf. Figure 3.3. This definition of a match is as

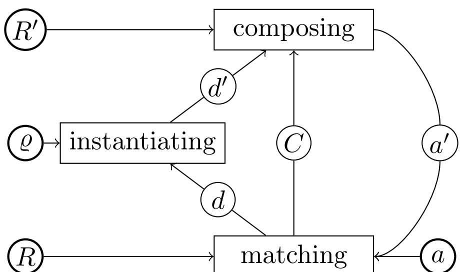



Figure 3.3: The reaction cycle

given by Jensen and Milner [13], except that we here also require $R$ to be regular. This restriction to regular redexes $R$ simplifies the inductive characterization of matching without limiting the set of possible reactions, as sites in $R$ and $R'$ can be renumbered to render $R$ regular.

### 3.2.9 Bigraph Terms and Normal Forms

Expressing bigraphs as terms composed by product, composition and abstraction over basic bigraph terms, Damgaard and Birkedal [8] showed that bigraphs can be expressed on normal forms uniquely

up to certain permutations and renamings. Further, they showed equivalence of term and bigraph equality, which will allow us in Section 3.4 to base our implementation on terms rather than graphs.

In this work, we use the normal forms shown in Figure 3.4, enabling us to express regular bigraphs simply by removing the permutations. These normal forms are unique up to permutation of $S_i$'s and renaming of names not visible on the interfaces.

$$
\begin{array}{rll}
M & ::= (\mathsf{id}_Z \otimes K_{\vec{y}(\vec{X})})N & \textit{molecule} \\
S & ::= \ulcorner \alpha \urcorner \ \mid \ M & \textit{singular top-level node} \\
G & ::= (\mathsf{id}_Y \otimes \mathit{merge}_n)(\bigotimes_i^n S_i)\pi & \textit{global discrete prime} \\
N & ::= (X)G & \textit{name-discrete prime} \\
P, Q & ::= (\mathsf{id}_Z \otimes \widehat{\sigma})N & \textit{discrete prime} \\
D & ::= \alpha \otimes (\bigotimes_i^n P_i)\pi & \textit{discrete bigraph} \\
B & ::= (\omega \otimes \mathsf{id}_{(\vec{X})})D & \textit{binding bigraph}
\end{array}
$$

Figure 3.4: Normal forms for binding bigraphs

### 3.2.10 Normalising

For normalising an arbitrary bigraph $t$, we define a normalisation relation $t \downarrow_{\mathbf{B}} t'$ for bigraph terms (details are given in Figure 3.22 of Appendix 3.A.1), with the following property:

**Proposition 1.** *For any bigraph terms $t, t'$, if $t$ represents a bigraph $b$ and $t \downarrow_{\mathbf{B}} t'$, then $t'$ represents $b$ as well, and is on B-normal form given in Figure 3.4.*

The relation is straightforward, recursively normalising subterms and recombining the results; for tensor product, the rule stated is

$$
\text{Bten}\ \frac{
\begin{array}{c}
t_i \downarrow_{\mathbf{B}} (\omega_i \otimes \mathsf{id}_{(\vec{Y}_i)})D_i \qquad D_i \equiv \alpha_i \otimes (\bigotimes_{j \in n_i} P_i^j)\pi_i : I_i \to \langle n_i, \vec{Y}_i, Y_i \rangle \\
\omega = \bigotimes_{i \in n} \omega_i \qquad \alpha = \bigotimes_{i \in n} \alpha_i \qquad \mathsf{id}_{(\vec{Y})} = \bigotimes_{i \in n} \mathsf{id}_{(\vec{Y}_i)} \qquad \pi = \bigotimes_{i \in n} \pi_i \\
P = \bigotimes_{j \in n} \bigotimes_{i \in n_j} P_i^j \qquad\qquad D \equiv \alpha \otimes P\pi
\end{array}
}{
\bigotimes_{i \in n} t_i \downarrow_{\mathbf{B}} (\omega \otimes \mathsf{id}_{(\vec{Y})})D
}.
$$

We find that the expression $\bigotimes_{j \in n} \bigotimes_{i \in n_j} P_i^j$ in general will lead to name clashes, because we can only assume that outer, not inner names, of the $\omega_i$'s are disjoint.

One solution could be to rename names on $P_i^j$'s outer face in the Bten rule. However, as Bten is applied recursively at each level of tensor product, this would lead to multiple renamings of the same names, causing inefficiency. Instead, we precede normalisation by a renaming phase described in the following; it will prevent name clashes in normalisation.

### 3.2.11 Renaming

While renaming names used in a term might look trivial at first sight, it is in fact not entirely straightforward. First, inner and outer names of a term must not be renamed, or we would be representing a different bigraph. Second, we cannot even require of a renamed term that all internal names are unique, as a normalised subterm can contain several instances of the same name, due to the use of $\mathsf{id}_Y$ in the normal form.

Thus, we need to identify a more refined notion of internal horizontal uniqueness, where a name can be reused vertically in link compositions, but not horizontally in tensor products. To this end, given a term $t$, we conceptually replace all occurrences of $/X$ by $e_1/x_1 \otimes \cdots \otimes e_n/x_n$, and $K_{\vec{y}(\vec{X})}$ by $K_{\vec{y}(\vec{e}/\vec{X})}$, in effect naming uniquely each closed link. We then define a function *linknames*, mapping

terms to link namers (details are given in Figure 3.23 of Appendix 3.A.2). Using this function we define a predicate *normalisable*, which identifies terms whose tensor products and compositions do not produce subterms with name clashes, and is preserved by normalisation (details are given in Figure 3.24 of Appendix 3.A.2):

**Proposition 2.** *For any bigraph term $t$, if normalisable$(t)$, there exists a $t'$ such that $t \downarrow_{\mathbf{B}} t'$ and normalisable$(t')$.*

For the actual renaming, we define inductively a renaming judgment $U \vdash \alpha, t \downarrow_{\boldsymbol{\beta}} t', \beta \dashv V$, where $U$ is a set of used names and $\alpha$ renames $t$'s inner names to those of $t'$, while $\beta$ renames $t$'s outer names to those of $t'$ and $V$ extends $U$ with names used in $t'$ (details are given in Figure 3.25 of Appendix 3.A.2).

We can show that renaming preserves the bigraph, and enables normalisation:

**Proposition 3.** *Given a term $t$ representing a bigraph $b : \langle m, \vec{X}, X\rangle \to \langle n, \vec{Y}, Y\rangle$, we can derive $X \cup Y \vdash \mathsf{id}_X, t \downarrow_{\boldsymbol{\beta}} t'', \beta \dashv V$ for some $t'', \beta, V$, and set $t' = ((\beta^{\mathbf{glob}})^{-1} \otimes \widehat{(\beta^{\mathbf{loc}})^{-1}})t''$; then $t'$ represents $b$, and normalisable$(t')$.*

## 3.2.12 Regularising

As a regular bigraph can be expressed as a term containing permutations, we must define *regularising* to represent it as a permutation-free term. This is done by splitting the permutations in the $D$- and $G$-normal forms, recursively pushing them into the subterms where they reorder the tensor product of $S_i$'s.

While $D$'s permutation $\pi$ must be a tensor product of $\pi_i$'s—otherwise the bigraph would not be regular—$G$'s permutation, on the other hand, need not be so. However, as the bigraph is regular, it must be possible to split it into a major permutation $\underline{\pi}^{\vec{X}}$ and $n$ minor permutations $\pi_i^{\vec{X}}$, based on the local inner faces, $\vec{X}$, of the $S_i$'s. Then $\underline{\pi}^{\vec{X}}$ is elided by permuting the $S_i$'s, and each $\pi_i^{\vec{X}}$ permutation is handled recursively in its $S_i$ (details are given in Figure 3.26 of Appendix 3.A.3).

We can show that regularisation is correct:

**Proposition 4.** *Given a term $t$ representing a regular bigraph $b$, we can infer $t \hookrightarrow t'$, for some $t'$ where $t'$ contains no nontrivial permutations, and $t'$ represents $b$.*

## 3.2.13 Summary

A detailed illustration of the entire reaction cycle including the preceding transformation technologies can be seen in Figure 3.5.

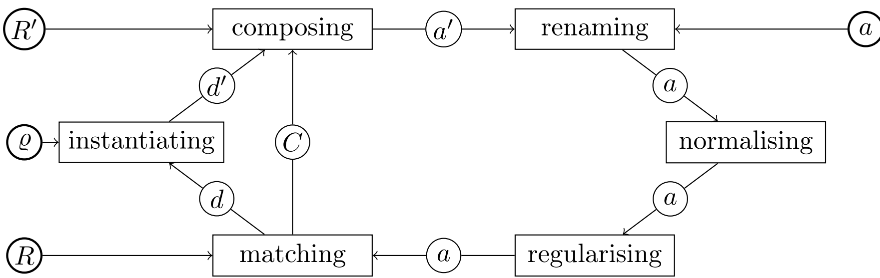



Figure 3.5: Details of the reaction cycle

## 3.3 Inferring Matches Using a Graph Representation

In this section we recap matching inference using a graph representation as developed in [9]; this representation is the basis for correctness proofs.

For simplicity, we will first consider just place graphs to explain the basic idea behind matching inference.

### 3.3.1 Matching place graphs

A place graph match is captured by a matching sentence:

**Definition 5** (Matching Sentence for Place Graphs)**.** *A* matching sentence *for place graphs is a 4-tuple of bigraphs $a, R \hookrightarrow C, d$, all are regular except $C$, with $a$ and $d$ ground. A sentence is* valid *iff $a = CRd$.*

We infer place graph matching sentences using the inference system given in Figure 3.6. Traversing an inference tree bottom-up, the agent is decomposed, while constructing the con-

$$\textsc{prime-axiom}\ \frac{}{p, \mathsf{id} \hookrightarrow \mathsf{id}, p} \qquad \textsc{ion}\ \frac{p, R \hookrightarrow P, d}{Kp, R \hookrightarrow KP, d} \qquad \textsc{switch}\ \frac{p, \mathsf{id} \hookrightarrow P, d}{p, P \hookrightarrow \mathsf{id}_1, d}$$

$$\textsc{par}\ \frac{a, R \hookrightarrow C, d \qquad b, S \hookrightarrow D, e}{a \otimes b, R \otimes S \hookrightarrow C \otimes D, d \otimes e} \qquad \textsc{perm}\ \frac{a, \bigotimes_i^n P_{\pi^{-1}(i)} \hookrightarrow C, \overline{\pi} d}{a, \bigotimes_i^n P_i \hookrightarrow C\pi, d}$$

$$\textsc{merge}\ \frac{a, R \hookrightarrow C, d}{\mathit{merge}\, a, R \hookrightarrow \mathit{merge}\, C, d}$$

Figure 3.6: Inference rules for deriving place graph matches

text, using the ION, MERGE and PAR rules. The PERM rule permutes redex parts to align tensor factors with corresponding agent factors.

At the point in the agent where a redex root should match, leaving a site in the context, the SWITCH rule is applied, switching the roles of the context and redex. This allows the remaining rules to be reused (above the switch rule) for checking that the redex matches the agent. When a site in the redex is reached, whatever is left of the agent should become (a part of) the parameter—this is captured by the PRIME-AXIOM rule.

For a match with a redex $R : m \to n$ consisting of $n$ nontrivial (i.e., non-identity) primes, the inference tree will contain $m$ applications of PRIME-AXIOM and $n$ applications of SWITCH. Further, between any leaf and the root of the inference tree, SWITCH will be applied at most once. The structure of a matching inference tree will thus generally be as illustrated in Figure 3.7; rules

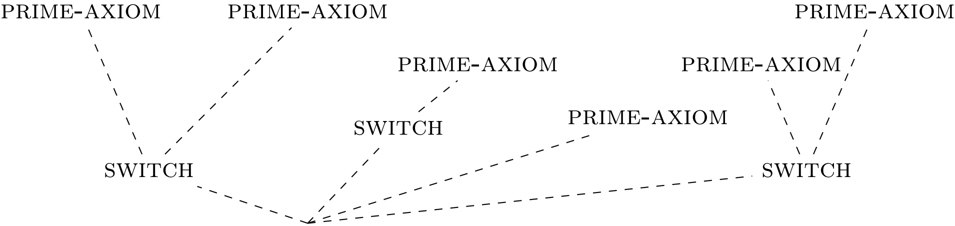



Figure 3.7: A sketch of the general structure of an inference tree for matching

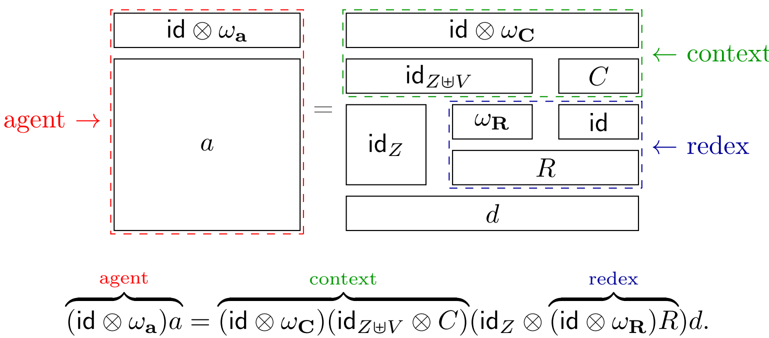
applied above SWITCH match agent and redex structure, while rules applied below match agent and context structure.

### 3.3.2 Matching binding bigraphs

Turning now to consider binding bigraphs, we extend the matching sentences to cater for links:

**Definition 6** (Matching Sentence for Binding Bigraphs)**.** *A (binding bigraph) matching sentence is a 7-tuple of bigraphs:* $\omega_{\mathbf{a}}, \omega_R, \omega_C \vdash a, R \hookrightarrow C, d$*, where* $a, R, C$ *and* $d$ *are discrete with local inner faces, all regular except* $C$*, with* $a$ *and* $d$ *ground. It is valid iff* $(\mathsf{id} \otimes \omega_{\mathbf{a}})a = (\mathsf{id} \otimes \omega_{\mathbf{C}})(\mathsf{id}_{Z \uplus V} \otimes C)(\mathsf{id}_Z \otimes (\mathsf{id} \otimes \omega_{\mathbf{R}})R)d$*.*

This definition separates the wirings, leaving local wiring in $a$, $R$, $C$ and $d$, while keeping global wiring of agent, redex and context in $\omega_{\mathbf{a}}$, $\omega_R$ and $\omega_C$, respectively; this is possible for any agent, redex and context [9]. The validity property shows how a valid matching sentence relates to a match, as illustrated in Figure 3.8.




$$\overbrace{(\mathsf{id} \otimes \omega_{\mathbf{a}})a}^{\text{agent}} = \overbrace{(\mathsf{id} \otimes \omega_{\mathbf{C}})(\mathsf{id}_{Z \uplus V} \otimes C)}^{\text{context}}(\mathsf{id}_Z \otimes \overbrace{(\mathsf{id} \otimes \omega_{\mathbf{R}})R}^{\text{redex}})d.$$

Figure 3.8: Decomposition of the bigraphs of a valid matching sentence

To reach a system for inferring valid matching sentences for binding bigraphs, we simply augment the place graph rules with wirings as shown in Figure 3.9, and add three rules for dealing with purely wiring constructs, shown in Figure 3.10. A detailed explanation of the rules is available

$$\text{PRIME-AXIOM}\ \frac{\sigma : W \uplus U \to \quad \beta : Z \to U \quad \alpha : V \to W \quad \tau : X \to V \quad p : \langle X \uplus Z \rangle}{\sigma(\beta \otimes \alpha\tau), \mathsf{id}_\epsilon, \sigma \vdash p, \mathsf{id}_{(V)} \hookrightarrow \ulcorner \alpha \urcorner, (\beta \otimes \widehat{\tau})(X)p}$$

$$\text{ION}\ \frac{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash ((\vec{v})/(\vec{X}) \otimes \mathsf{id}_U)p, R \hookrightarrow ((\vec{v})/(\vec{Z}) \otimes \mathsf{id}_W)P, d \qquad \alpha = \vec{y}/\vec{u} \qquad \sigma : \{\vec{y}\} \to}{\sigma \parallel \omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \sigma\alpha \parallel \omega_{\mathbf{C}} \vdash (K_{\vec{y}(\vec{X})} \otimes \mathsf{id}_U)p, R \hookrightarrow (K_{\vec{u}(\vec{Z})} \otimes \mathsf{id}_W)P, d}$$

$$\text{SWITCH}\ \frac{\omega_{\mathbf{a}}, \mathsf{id}_\epsilon, \omega_{\mathbf{C}}(\sigma \otimes \omega_{\mathbf{R}} \otimes \mathsf{id}_Z) \vdash p, \mathsf{id} \hookrightarrow P, d \quad \sigma : W \to U \quad P : \to \langle W \uplus Y \rangle \quad d : \langle m, \vec{X}, X \uplus Z \rangle}{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash p, (\widehat{\sigma} \otimes \mathsf{id}_Y)(W)P \hookrightarrow \ulcorner U \urcorner, d}$$

$$\text{PAR}\ \frac{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \parallel \omega \vdash a, R \hookrightarrow C, d \qquad \omega_{\mathbf{b}}, \omega_{\mathbf{S}}, \omega_{\mathbf{D}} \parallel \omega \vdash b, S \hookrightarrow D, e}{\omega_{\mathbf{a}} \parallel \omega_{\mathbf{b}}, \omega_{\mathbf{R}} \parallel \omega_{\mathbf{S}}, \omega_{\mathbf{C}} \parallel \omega_{\mathbf{D}} \parallel \omega \vdash a \otimes b, R \otimes S \hookrightarrow C \otimes D, d \otimes e}$$

$$\text{PERM}\ \frac{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash a, \bigotimes_i^m P_{\pi^{-1}(i)} \hookrightarrow C, (\overline{\pi} \otimes \mathsf{id})d}{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash a, \bigotimes_i^m P_i \hookrightarrow C\pi, d}$$

$$\text{MERGE}\ \frac{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash a, R \hookrightarrow C, d}{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash (\mathit{merge} \otimes \mathsf{id}_Y)a, R \hookrightarrow (\mathit{merge} \otimes \mathsf{id}_X)C, d}$$

Figure 3.9: Place graph rules (shaded) augmented for deriving binding bigraph matches

in the literature [9], along with proofs of soundness and completeness of the inference system.

$$\text{WIRING-AXIOM}\ \frac{}{y, X, y/X \vdash \mathsf{id}_\epsilon, \mathsf{id}_\epsilon \hookrightarrow \mathsf{id}_\epsilon, \mathsf{id}_\epsilon}$$

$$\text{ABSTR}\ \frac{\sigma_{\mathbf{a}} \otimes \omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \sigma_{\mathbf{C}} \otimes \omega_{\mathbf{C}} \vdash p, R \hookrightarrow P, d \quad \sigma_{\mathbf{a}} : Z \to W \quad p : \langle Z \uplus Y \rangle \quad \sigma_{\mathbf{C}} : U \to W \quad P : \to \langle U \uplus X \rangle}{\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash (\widehat{\sigma_{\mathbf{a}}} \otimes \mathsf{id}_Y)(Z)p, R \hookrightarrow (\widehat{\sigma_{\mathbf{C}}} \otimes \mathsf{id}_X)(U)P, d}$$

$$\text{CLOSE}\ \frac{\sigma_{\mathbf{a}}, \sigma_{\mathbf{R}}, \mathsf{id}_{Y_{\mathbf{R}}} \otimes \sigma_{\mathbf{C}} \vdash a, R \hookrightarrow C, d \quad \sigma_{\mathbf{a}} : \to U \uplus Y_R \quad \sigma_{\mathbf{R}} : \to V \uplus Y_R \quad \sigma_{\mathbf{C}} : \to W \uplus Y_C}{(\mathsf{id}_U \otimes /(Y_{\mathbf{R}} \uplus Y_{\mathbf{C}}))\sigma_{\mathbf{a}}, (\mathsf{id}_V \otimes /Y_{\mathbf{R}})\sigma_{\mathbf{R}}, (\mathsf{id}_W \otimes /Y_{\mathbf{C}})\sigma_{\mathbf{C}} \vdash a, R \hookrightarrow C, d}$$

Figure 3.10: Added inference rules for deriving binding bigraph matches

## 3.4 From Graph Matching to Term Matching

In this section we transform the graph based inductive characterisation of matching to be based on a term representation in such a way that correctness and completeness is preserved.

While the graph representation of matching sentences is useful for constructing a relatively simple inference system amenable to correctness proofs, it is not sufficient for an implementation based on syntax, that is, bigraph terms. One bigraph can be represented by several different bigraph terms that are structurally congruent by the axiom rules: $a = a \otimes \mathsf{id}_0 = \mathit{merge}_1\, a$, $a \otimes (b \otimes c) = (a \otimes b) \otimes c$ and $\mathit{merge}(a \otimes b) = \mathit{merge}(b \otimes a)$. If, for instance, we were to match agent $a = \mathit{merge}((\mathsf{K} \otimes \mathsf{L}) \otimes \mathsf{M})$ with redex $R = \mathsf{K}$, we would first need to apply the axioms to achieve $R = \mathit{merge}((\mathsf{K} \otimes \mathsf{id}_0) \otimes \mathsf{id}_0)$ before being able to apply the MERGE and PAR rules.

In the following, we recast the matching sentences to be tuples of 3 wirings and 4 bigraph *terms* $\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_C \vdash a, R \rightsquigarrow C, d$, with the same restrictions and validity as before, interpreting the terms as the bigraphs they represent. Given this, adding just this one rule would be sufficient to achieve completeness of the inference system:

$$\text{STRUCT}\ \frac{a \equiv a' \quad R \equiv R' \quad C \equiv C' \quad h \equiv h' \quad \omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a', R' \rightsquigarrow C', h'}{\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow C, h}$$

The STRUCT rule says that we can apply structural congruence to rewrite any term $a, R, C$ or $h$ to a term denoting the same bigraph. With the help of the equational theory for determining bigraph isomorphism on the term level [8], we have essentially a nondeterministic algorithm for matching bigraph terms—implementable in say, Prolog. A brief glance at the equational theory, shows us, though, that the associative and commutative properties of the basic operators of the language would yield a wildly nondeterminstic inference system, since we would need to apply structural congruence between every step to infer a match. This is reminiscent of the problems in implementing *rewriting logic*, that is, term rewriting modulo a set of static equivalences [5, 6, 16]. Consequentially, we abandon the fully general STRUCT rule. For the purposes of stating the completeness theorem below, we shall need to refer to sentences derived from the ruleset for bigraphs (i.e., from section 3.3.2) recast to terms with the help of the STRUCT rule above. We shall write such sentences $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow_{\mathrm{S}} C, h$ for wirings $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}}$ and terms $a, R, C$ and $h$.

**Definition 7.** *For wirings $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}}$ and terms $a, R, C$ and $h$, sentences $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow_{\mathrm{S}} C, h$ range over sentences derived from the rules of Figure 3.10—reading $a, R, C$ and $h$ as terms—extended with the* STRUCT *rule.*

## 3.5 Normal Inferences

Next, we look at how to restrict the term based inductive characterisation of matching as an enabling step for designing an algorithm. We define normal inferences, limiting the set of inferences we need to consider.

Normal inferences is a class of inferences that are complete in the sense that all valid matching sentences can be inferred, but suitably restricted, such that inferences can be built mechanically. In particular, normal inference definitions for term matching spell out *how* and *where* to apply structural congruence. As a main trick, we utilize a variant of the normal forms proven complete for binding bigraphs (cf. Section 3.2.9), lending us a set of uniform representations of classes of bigraphs based directly on terms for bigraphs; we define normal inferences that require each inference to start by rewriting the term to be on normal form.

Before giving the format for normal inferences, we incorporate structural congruence axioms into PRODUCT and MERGE rules. We derive rules for iterated tensor product and permutations under merge, arriving at the inference system shown in Figure 3.11. In this inference system, the terms in the conclusion of every rule except DNF is in some normal form as given by Figure 3.4, where $e$ is a discrete prime ($p$) or global discrete prime ($g$). An expression $[\![t]\!]^G$ means term $t$ expressed on $G$-normal form—for instance, $[\![\ulcorner\alpha\urcorner]\!]^G$ means $(\mathsf{id}_Y \otimes \mathit{merge}_1)(\bigotimes_i^1 \ulcorner\alpha\urcorner)$—and similarly for the remaining normal forms. The expression $\bar{\varrho}(n,m)$ denotes the set of $n$-$m$-partitions. An $n$-$m$-partition $\varrho$ is a partition of $\{0,\ldots,n-1\}$ into $m$ (possibly empty) subsets, and for $i \in m$, $\varrho_i$ is the $i$th subset. Given a metavariable $\mathcal{X}$, $\overline{\mathcal{X}}$ ranges over iterated tensor products of $\mathcal{X}$'es. As indicated by the superscript, rules PER$^{\mathcal{E}}$, PAR$^{\mathcal{E}}_{\mathbf{n}}$ and PAR$^{\mathcal{E}}_{\equiv}$ can be used either on discrete primes $p$ and $P$ or global discrete primes $g$ and $G$.

The main differences from the preceding inference system is that we have replaced the binary PAR rule by two iterative PAR rules, PAR$^{\mathcal{E}}_{\mathbf{n}}$ and PAR$^{\mathcal{E}}_{\equiv}$, and specialised the MERGE rule into a rule, MER, that makes the partitioning of children in an agent node explicit. The PAR$^{\mathcal{E}}_{\equiv}$ rule splits up an iterated tensor product into a number of products matching agent factors, while PAR$^{\mathcal{E}}_{\mathbf{n}}$ performs the actual inductive inference on each of the factors. (Note, by the way, that PAR$^{\mathcal{E}}_{\equiv}$ and MER$^{\mathcal{E}}_{\equiv}$ correspond just to particular instances of the STRUCT-rule, that we abandoned above.)

Furthermore, note that the usage of the previous WIRING-AXIOM-rule for introducing idle linkage has been inlined to a side-condition on a slightly generalized PAR-rule (i.e., the PAR$^{\mathcal{E}}_{\mathbf{n}}$-rule). The $\sigma'$ in that rule allows us to introduce idle linkage in redex and agent, and link them in context; as previously allowed by the WIRING-AXIOM-rule. Hence, PAR$^{\mathcal{E}}_{\mathbf{n}}$ also serves as an axiom, introducing 0-ary products of $\mathsf{id}_\epsilon$'s on $G$- and $P$-normal forms.

While this inference system is more explicit about partitioning tensor products (in the MER and PAR$^{\mathcal{E}}_{\equiv}$ rules), there is still a lot of nondeterministic choice left in the *order* in which the rules can be applied. To limit this, we define normal inferences based, essentially, on the order rules were used in the proof of completeness [9]. We derive a sufficient order that still preserves completeness:

**Definition 8** (Normal Inference)**.** *A* normal inference *is a derivation using the term matching rules of Figure 3.11 in the order specified by the context free grammar given in Figure 3.12.*

Now we can give the main theorem stating that normal inferences are sufficient for finding all valid matches. The following theorem states formally for every sentence derivable with the ruleset for bigraphs recast to bigraph terms by extending with STRUCT, that such a sentence is also derivable as a normal inference.

**Theorem 9** (Normal inferences are sound and complete)**.** *For wirings $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}}$ and terms $a, R, C, d$, we can infer $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow_{\mathrm{S}} C, d$ iff we can infer $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow C, d$ using a normal inference.*

*Proof.* (*Sketch*) By induction over the structure of the derivation of the sentence $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow_{\mathrm{S}} C, d$. We case on the last rule used to conclude this sentence. By the induction hypothesis (IH), we can conclude a normal derivation of the sentence used for concluding $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \rightsquigarrow_{\mathrm{S}} C, d$.

STRUCT: By IH, we can construct a normal derivation of $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a', R' \rightsquigarrow C', d'$, with $a = a'$, $R' = R$, $C' = C$ and $d' = d$. This normal derivation can be used directly to conclude also $\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a', R' \rightsquigarrow C', d'$.

$$\text{PAX}\ \frac{\sigma : W \uplus Z \to \quad \alpha : V \to W \quad \tau : X \to V \quad g : \langle X \uplus Z \rangle}{\sigma(\mathsf{id}_Z \otimes \alpha\tau), \mathsf{id}_\epsilon, \sigma \vdash g, [\![\mathsf{id}_{(V)}]\!]^{\overline{P}} \leadsto [\![\ulcorner \alpha \urcorner]\!]^G, [\![(\mathsf{id}_Z \otimes \widehat{\tau})(X)g]\!]^{\overline{P}}}$$

$$\text{ION}\ \frac{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash (\mathsf{id} \otimes (\vec{v})/(\vec{X}))n, \overline{P} \leadsto (\mathsf{id} \otimes (\vec{v})/(\vec{Z}))N, \overline{q} \quad \alpha = \vec{y}/\vec{u} \quad \sigma : \{\vec{y}\} \to}{(\sigma \parallel \sigma^{\mathbf{a}}), \sigma^{\mathbf{R}}, (\sigma\alpha \parallel \sigma^{\mathbf{C}}) \vdash [\![(\mathsf{id}_U \otimes K_{\vec{y}(\vec{X})})n]\!]^G, \overline{P} \leadsto [\![(\mathsf{id}_W \otimes K_{\vec{u}(\vec{Z})})N]\!]^G, \overline{q}}$$

$$\text{SWX}\ \frac{\begin{array}{c}\sigma : W \to U \quad G : \to \langle W \uplus Y \rangle \quad \overline{q} : \langle n, \vec{X}, X \uplus Z \rangle \quad X = \{\vec{X}\} \\ \sigma^{\mathbf{a}}, \mathsf{id}_\epsilon, \sigma^{\mathbf{C}}(\mathsf{id}_Z \otimes \sigma \otimes \sigma^{\mathbf{R}}) \vdash g, [\![\bigotimes_i^n \mathsf{id}_{(X_i)}]\!]^{\overline{P}} \leadsto G, \overline{q}\end{array}}{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash g, [\![(\mathsf{id}_Y \otimes \widehat{\sigma})(W)G]\!]^{\overline{P}} \leadsto [\![\ulcorner U \urcorner]\!]^G, \overline{q}}$$

$$\text{PAR}_{\mathbf{n}}^{\mathcal{E}}\ \frac{\sigma' : I_{\mathbf{R}} \to I_{\mathbf{a}} \quad (\forall i \in n)\ \sigma_i^{\mathbf{a}}, \sigma_i^{\mathbf{R}}, \sigma \parallel \sigma_i^{\mathbf{C}} \vdash e_i, \overline{P}_i \leadsto E_i, \overline{q}_i}{\big(I_{\mathbf{a}} \parallel \|_i^n \sigma_i^{\mathbf{a}}\big), \big(I_{\mathbf{R}} \parallel \|_i^n \sigma_i^{\mathbf{R}}\big), \big(\sigma' \parallel \sigma \parallel \|_i^n \sigma_i^{\mathbf{C}}\big) \vdash \bigotimes_i^n e_i, \bigotimes_i^n \overline{P}_i \leadsto \bigotimes_i^n E_i, \bigotimes_i^n \overline{q}_i}$$

$$\text{PAR}_{\equiv}^{\mathcal{E}}\ \frac{\begin{array}{c}P'_{ij} = P_{j+\sum_{r\in i} l_r} \quad q'_{ij} = q_{j+\sum_{r\in i} k_r} \quad P'_{ij} : \langle k_{ij}, \vec{X}_{ij} \rangle \to \quad k_i = \sum_{j\in l_i} k_{ij} \\ \sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash \bigotimes_i^n e_i, \bigotimes_i^n \bigotimes_j^{l_i} P'_{ij} \leadsto \bigotimes_i^n E_i, \bigotimes_i^n \bigotimes_j^{k_i} q'_{ij}\end{array}}{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash \bigotimes_i^n e_i, \bigotimes_i^m P_i \leadsto \bigotimes_i^n E_i, \bigotimes_i^{m'} q_i}$$

$$\text{PER}^{\mathcal{E}}\ \frac{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash \overline{e}, \bigotimes_i^n Q_{\pi^{-1}(i)} \leadsto \overline{E}, \bigotimes_i^m q_{\bar{\pi}^{-1}(i)}}{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash \overline{e}, \bigotimes_i^n Q_i \leadsto \overline{E}\pi, \bigotimes_i^m q_i}$$

$$\text{MER}\ \frac{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash \bigotimes_i^m (\mathsf{id} \otimes \mathit{merge}) \bigotimes_{j \in \varrho_i, \varrho \in \bar{\varrho}(n,m)} m_j, \overline{P} \leadsto (\bigotimes_i^m [\![S_{\pi^{-1}(i)}]\!]^G)\bar{\pi}, \overline{q}}{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash (\mathsf{id} \otimes \mathit{merge}) \bigotimes_i^n m_i, \overline{P} \leadsto (\mathsf{id} \otimes \mathit{merge}) \bigotimes_i^m S_i, \overline{q}}$$

$$\text{ABS}\ \frac{\sigma_L^{\mathbf{a}} \otimes \sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma_L^{\mathbf{C}} \otimes \sigma^{\mathbf{C}} \vdash g, \overline{P} \leadsto G, \overline{q} \quad \sigma_L^{\mathbf{a}} : Z \to W \quad \sigma_L^{\mathbf{C}} : U \to W \quad G : \to \langle U \uplus X \rangle}{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \sigma^{\mathbf{C}} \vdash (\mathsf{id} \otimes \widehat{\sigma_L^{\mathbf{a}}})(Z)g, \overline{P} \leadsto (\mathsf{id} \otimes \widehat{\sigma_L^{\mathbf{C}}})(U)G, \overline{q}}$$

$$\text{CLO}\ \frac{\sigma^{\mathbf{a}}, \sigma^{\mathbf{R}}, \mathsf{id}_{Y_{\mathbf{R}}} \otimes \sigma^{\mathbf{C}} \vdash \overline{p}, \overline{P} \leadsto \overline{Q}\pi, \overline{q}}{(\mathsf{id} \otimes /(Y_{\mathbf{R}} \uplus Y_{\mathbf{C}}))\sigma^{\mathbf{a}}, (\mathsf{id} \otimes /Y_{\mathbf{R}})\sigma^{\mathbf{R}}, (\mathsf{id} \otimes /Y_{\mathbf{C}})\sigma^{\mathbf{C}} \vdash \overline{p}, \overline{P} \leadsto \overline{Q}\pi, \overline{q}}$$

$$\text{DNF}\ \frac{\begin{array}{c}a \equiv \overline{p} \quad R \equiv \overline{P} \quad C \equiv \overline{Q}\pi \quad h \equiv \overline{q} \quad \overline{p}, \overline{P}, \overline{Q}, \overline{q} \text{ are on normal form} \quad R \text{ is regular} \\ \omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash \overline{p}, \overline{P} \leadsto \overline{Q}\pi, \overline{q}\end{array}}{\omega^{\mathbf{a}}, \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash a, R \leadsto C, h}$$

Figure 3.11: Inference rules for binding bigraph terms

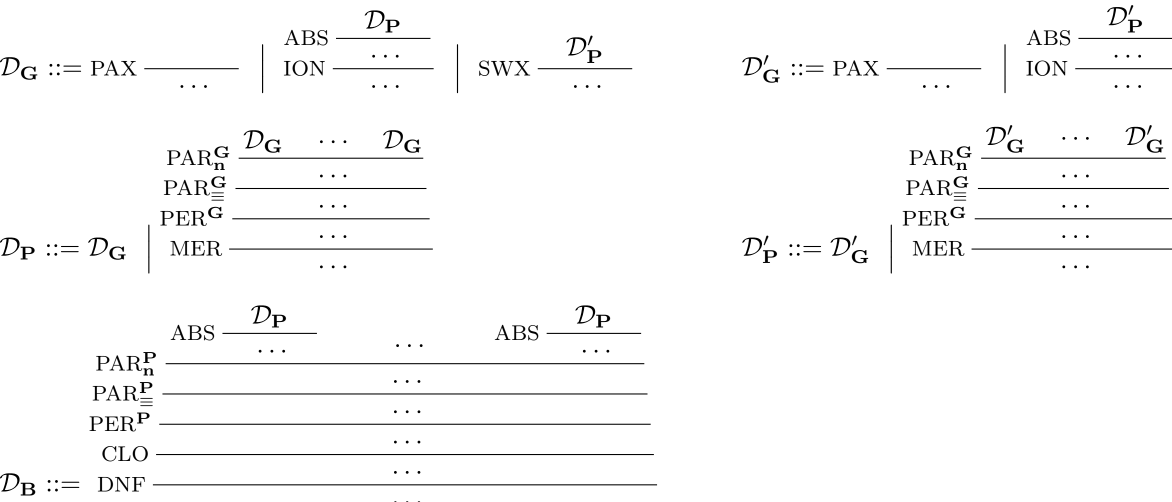


Figure 3.12: Grammar (BNF) for normal inferences for binding bigraphs with start symbol $\mathcal{D}_\mathbf{B}$

PRIME-AXIOM: We produce the needed normal inference by starting with an application of PAX, which introduces the needed prime bigraphs and wiring—that is, each term being equal up to structural congruence to the sentence concluded with PRIME-AXIOM. Now we proceed to build the needed normal inference by a building first a $\mathcal{D}_\mathbf{P}$ and then a $\mathcal{D}_\mathbf{B}$-inference. All steps add only term structure to match a particular normal form, while not changing the denotation of the terms.

ION: By IH, we can construct a normal derivation of $\omega_\mathbf{a}, \omega_\mathbf{R}, \omega_\mathbf{C} \vdash ((\vec{v})/(\vec{X}) \otimes \mathsf{id}_U)p, R \rightsquigarrow ((\vec{v})/(\vec{Z}) \otimes \mathsf{id}_W)P, d$. For this case, we have to unroll that normal derivation up across the $\mathcal{D}_\mathbf{B}$ production except for the last ABS-step, concluding with a $\text{PAR}_1^\mathbf{P}$ step (since we know $p$ and $P$ are prime). We now have a $\mathcal{D}_\mathbf{P}$ normal inference with an added ABS-step, which we can use for concluding an ION-step introducing our needed ion. Referring to the grammar in Figure 3.12, we see that this produces a $\mathcal{D}_\mathbf{G}$-inference, which we have to lead through two series of PAR-PER-MER steps (and one ABS-step), to produce a full normal inference.

SWITCH: This case needs a little extra care. First, we point out two properties of normal derivations: (i) any $\mathcal{D}_\mathbf{G}$ and $\mathcal{D}_\mathbf{P}$ inference without SWX is also a $\mathcal{D}'_\mathbf{G}$ or $\mathcal{D}'_\mathbf{P}$ inference, respectively; and, (ii) any sentence, $\omega_\mathbf{a}, \mathsf{id}_\epsilon, \omega_\mathbf{C} \vdash a, \mathsf{id} \rightsquigarrow C, h$ has a normal derivation with no SWX-steps. Both are easily verified.

Now, by the IH we can construct a normal derivation of a sentence $\omega_\mathbf{a}, \mathsf{id}_\epsilon, \omega_\mathbf{C}(\sigma \otimes \omega_\mathbf{R} \otimes \mathsf{id}_Z) \vdash p, \mathsf{id} \rightsquigarrow P, d$ for global $P$. By property (ii), we can assume that this normal derivation does not contain any applications of SWX. We unroll this normal derivation up across the whole $\mathcal{D}_\mathbf{B}$ production, . This leaves us with a $\mathcal{D}_\mathbf{P}$-type normal derivation, which by property (i), we can use also as $\mathcal{D}'_\mathbf{P}$ derivation. Hence, we can apply SWX to obtain a $\mathcal{D}_\mathbf{G}$ derivation. We proceed to build first a $\mathcal{D}_\mathbf{P}$ type inference, and then a $\mathcal{D}_\mathbf{B}$ type inference, in particular applying again ABS to introduce local linkage in $p$.

PAR: By IH, we can construct normal derivations of $\omega_\mathbf{a}, \omega_\mathbf{R}, \omega_\mathbf{C} \| \omega \vdash a, R \rightsquigarrow C, d$ and $\omega_\mathbf{b}, \omega_\mathbf{S}, \omega_\mathbf{D} \| \omega \vdash b, S \rightsquigarrow D, e$. Each of these normal derivations we can unroll up to the last application of $\text{PAR}_\mathbf{n}^\mathbf{P}$ $\mathcal{D}_i$ and $\mathcal{E}_j$, applied for concluding these $\text{PAR}_\mathbf{n}^\mathbf{P}$ steps. To construct the required normal inference we simply let instead a single $\text{PAR}_\mathbf{n}^\mathbf{P}$ step utilize all of the normal inferences $\mathcal{D}_i$ and $\mathcal{E}_j$.

PERM: By IH, we can construct a normal derivation of $\omega^\mathbf{a}, \omega^\mathbf{R}, \omega^\mathbf{C} \vdash a, \bigotimes_i^m P_{\pi(i)} \rightsquigarrow C, (\overline{\pi} \otimes \mathsf{id}_Z)d$. Unrolling this normal derivation up through the applications of DNF, CLO, and $\text{PER}^\mathbf{P}$, we can edit the $\text{PER}^\mathbf{P}$-step to also move the permutation $\pi$ to the context.

MERGE: By IH, we can construct a normal derivation of $\omega_\mathbf{a}, \omega_\mathbf{R}, \omega_\mathbf{C} \vdash a, R \rightsquigarrow C, d$ for global $a$ and $C$. We unroll this derivation up across the $\mathcal{D}_\mathbf{B}$ production to obtain $n$ $\mathcal{D}_\mathbf{P}$-derivations (for

$a$ and $C$ of width $n$). We may consider these as $\mathcal{D}_{\mathbf{G}}$-derivations, also. We combine these in a single application of $\text{PAR}^{\mathbf{G}}_{\mathbf{n}}$, and, after a $\text{PAR}^{\mathbf{P}}_{\equiv}$ and a PER-step, we apply MER to merge the roots as required by the case. We conclude by adding term structure to the terms of this $\mathcal{D}_{\mathbf{P}}$-inference as required by the normal form and lead it through the steps to produce a $\mathcal{D}_{\mathbf{B}}$-derivation.

WIRING-AXIOM: As sketched in the text above, introduction of idle names is now handled by $\text{PAR}^{\mathbf{P}}_{n}$. For this case, we simply start with a $\text{PAR}^{\mathbf{P}}_{0}$-step and proceed through the grammar for $\mathcal{D}_{\mathbf{B}}$ to produce a normal inference as needed.

ABSTR: By IH, we can construct a normal derivation of $\sigma_{\mathbf{a}} \otimes \omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \sigma_{\mathbf{C}} \otimes \omega_{\mathbf{C}} \vdash p, R \rightsquigarrow P, d$. We unroll this normal derivation up across the entire $\mathcal{D}_{\mathbf{B}}$-inference to obtain a $\mathcal{D}_{\mathbf{P}}$ type inference. (We know there is only one $\mathcal{D}_{\mathbf{P}}$-inference, as $p$ and $P$ are prime.) We construct the required $\mathcal{D}_{\mathbf{B}}$ inference by starting with a modified ABS-step, where we introduce the required abstractions and local substitutions.

CLOSE: By IH, we can construct a normal inference for a sentence with only substitutions (i.e., with no closed links). We simply unroll this normal inference up across the CLO-step, and instead, to produce the needed normal inference, close the needed names in a new CLO-step. □

Normal inferences are sufficiently restricted such that we can base our prototype implementation on mechanically constructing them.

## 3.6 Bigraph Matching Algorithm

In this section, we show how to interpret the inference rules as functions—achieving an implementation proven correct in great detail.

In general, an inference tree can be divided into two parts: the twigs of the tree, consisting of all the rule applications above a SWX rule application, and the base of the tree, consisting of the remaining rule applications, cf. Figure 3.13. The base and the SWX rule applications determine the

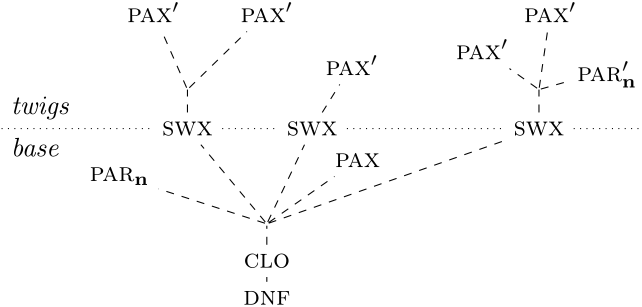



Figure 3.13: A sketch of the general structure of a normal inference tree for matching; dashed lines represent rules omitted for brevity

locations in the agent at which the roots of the redex are matched, and each twig determines how the place graph subtree below a redex root is matched to the corresponding subtree in the agent. Each PAX leaf determines a component of the parameter, and each $\text{PAR}_{\mathbf{n}}$ leaf corresponds to a leaf node in the agent.

We turn the declarative matching specification of the preceding section into a matching algorithm by considering each rule operationally. By implementing the inference system faitfully rule by rule, we ensure that the proof of completeness is valid also for the implemented algorithm.

All rules of Figure 3.11 except SWX, CLO and DNF can be applied in two flavours: below and above a SWX rule. We distinguish these applications using primes ($'$) for rules applied above a SWX rule.

The DNF rule is concerned with normalising $a$ and $R$, which is done using the algorithm described in Section 3.2.10, so for the remaining rules of the base of the tree we have as input to the algorithm wirings $\omega^{\mathbf{a}}$, $\omega^{\mathbf{R}}$ and discrete bigraphs $\bar{p}$ (agent) and $\bar{P}$ (redex). The goal is to find context wiring $\omega^{\mathbf{C}}$, context term $\bar{Q}\pi$ and parameter term $\bar{q}$.

In general, there are zero or more matches, so in the implementation the application of each rule returns a lazy list of matches, each containing a context wiring, along with context and parameter bigraphs.

As the matching inference starts at the root of the inference tree, we will do the same, considering each rule of Figure 3.11 in turn, and giving an operational interpretation of it:

### 3.6.1 Rule Applications in the Base of the Tree

CLO: Operationally, CLO opens $\omega^{\mathbf{a}}$ and $\omega^{\mathbf{R}}$, producing $\sigma^{\mathbf{a}}$ and $\sigma^{\mathbf{R}}$ and assigning fresh names $Y_{\mathbf{R}}$ to edges in $\omega^{\mathbf{R}}$.

> [Aside: We cannot generally conclude there are no matches if the number of edges in $\omega^{\mathbf{a}}$ is smaller than the number of edges in $\omega^{\mathbf{R}}$! But the number of port-connected edges in $\omega^{\mathbf{R}}$ must be greater than or equal to the number of edges in $\omega^{\mathbf{a}}$. Unfortunately, port-connectedness is expensive to determine in the term representation.]

In the implementation, we represent $\sigma^{\mathbf{R}}$ as $\sigma^{\mathbf{R}}_{\mathbf{e}} \otimes \sigma^{\mathbf{R}}_{\mathbf{n}}$, where $\sigma^{\mathbf{R}}_{\mathbf{n}}$ contains all the outer names of $\omega^{\mathbf{R}}$, and $\sigma^{\mathbf{R}}_{\mathbf{e}} : X_{\mathbf{e}} \to Y_{\mathbf{R}}$ contains the edges of $\omega^{\mathbf{R}}$ for some $X_{\mathbf{e}}$.

As we do not yet know which edges in $\omega^{\mathbf{a}} : \to W$ match which names in $Y_{\mathbf{R}}$, we represent $\sigma^{\mathbf{a}}$ as $\alpha\sigma^{\mathbf{a}}_{\mathbf{e}} \otimes \sigma^{\mathbf{a}}_{\mathbf{n}}$, where $\sigma^{\mathbf{a}}_{\mathbf{n}}$ contains all the open links of $\omega^{\mathbf{a}}$, and $\alpha$ is to be constructed during (i.e., returned by) the rest of the inference.

Inference must be done under the following condition:

- links in $\sigma^{\mathbf{R}}_{\mathbf{e}}$ must only be matched with links in $\alpha\sigma^{\mathbf{a}}_{\mathbf{e}}$

because redex edges can only match agent *edges,* not open links.

When the premise has been inferred, yielding $\mathsf{id}_{Y_{\mathbf{R}}} \otimes \sigma^{\mathbf{C}}$, we determine $\sigma^{\mathbf{C}} : \to W \uplus Y_{\mathbf{C}}$ (as $Y_{\mathbf{R}}$ is known), and then $Y_{\mathbf{C}} = (W \uplus Y_{\mathbf{C}}) \setminus W$. Finally, $\omega^{\mathbf{C}} = (\mathsf{id} \otimes /Y_{\mathbf{C}})\sigma^{\mathbf{C}}$.

ABS: During the inference, ABS adds the links $\sigma^{\mathbf{a}}_{\mathbf{L}}$ to $\sigma^{\mathbf{a}}_{\mathbf{n}}$; the inner names of $\sigma^{\mathbf{a}}_{\mathbf{L}}$ are collected in a set $L$.

As $\sigma^{\mathbf{a}}_{\mathbf{L}}$ contains links bound in the agent, inference must be done under the following condition:

- links in $\sigma^{\mathbf{a}}_{\mathbf{n}}$ whose inner names are in $L$ must not be matched via $\sigma^{\mathbf{R}}$, but must be matched via $\sigma$ (the local outer names of the redex) in the SWX rule.

This enforces the scoping rule for the resulting context bigraph.

When the premise of ABS has been inferred, $\sigma^{\mathbf{C}}_{\mathbf{L}}$ is computed by restricting the outer face of the context wiring to $W$, the outer names of $\sigma^{\mathbf{a}}_{\mathbf{L}}$.

MER: Taking $r$ be the outer width of redex $\bar{P}$, we let $m = r + 1$, and compute all partitions of width $m$. Setting $m > r$ allows parts of the agent to not be matched by the redex ($l_i = 0$ in PAR$^{\mathcal{E}}_{\mathbf{n}}$), that is, to become part of the context. For each partition, the premise is inferred, and if the permutation $\bar{\pi}$ in the returned context really is a pushthrough of some permutation $\pi$, the factors $S_i$ of the tensor product are permuted before they are returned.

PER: For each $n$-permutation $\pi$, $\bar{\pi}$ is computed by pushing $\pi$ through $\bigotimes^i_n Q_i$. After the premise has been inferred on the permuted redex primes, $\bar{\pi}$ is used to permute the resulting parameter primes accordingly before they are returned.

PAR$^{\mathcal{E}}_{\equiv}$: For each split of $m$ into $n$ parts, $\bigotimes^n_i \bigotimes^{l_i}_j P'_{ij}$ is computed, and after the premise has been inferred, the factors of the resulting parameter tensor product are concatenated into one tensor product before returning context and parameter.

PAR$^{\mathcal{E}}_{\mathbf{n}}$: For each $i$, the global outer names of $e_i$ is used to compute $\sigma^{\mathbf{a}}_{\mathbf{e},i}, \sigma^{\mathbf{a}}_{\mathbf{n},i}$ by restriction, and similarly for $\bar{P}_i$ and $\sigma^{\mathbf{R}}_{\mathbf{e},i}, \sigma^{\mathbf{R}}_{\mathbf{n},i}$. The context wirings resulting from inferring the premise are combined using parallel product, but allowing inner name clashes as long as each operand maps the inner names to the same outer name.

ION: Letting $Y = \{\vec{y}\}$, we split $\sigma^{\mathbf{a}}_{\mathbf{e}} = \sigma^{Y}_{\mathbf{e}} \parallel \sigma^{\mathbf{a}\prime}_{\mathbf{e}}$ according to whether the inner names are in $Y$ or not; and similarly for $\sigma^{\mathbf{a}}_{\mathbf{n}} = \sigma^{Y}_{\mathbf{n}} \parallel \sigma^{\mathbf{a}\prime}_{\mathbf{n}}$. Fresh names $\vec{v}$ are created

### 3.6.2 Application of the SWX Rule

As the SWX rule preserves $\sigma^{\mathbf{a}}$, its representation is not changed above the SWX rule. As CLO requires the resulting $\sigma^{\mathbf{C}}$ to be of the form $\mathsf{id}_{Y_{\mathbf{R}}} \otimes \sigma^{\mathbf{C}}$, where $\sigma^{\mathbf{R}}_{\mathbf{e}} : \to Y_{\mathbf{R}}$, the third wiring above SWX is $(\mathsf{id}_{Y_{\mathbf{R}}} \otimes \sigma^{\mathbf{C}})(\mathsf{id}_Z \otimes \sigma \otimes \sigma^{\mathbf{R}}_{\mathbf{e}} \otimes \sigma^{\mathbf{R}}_{\mathbf{n}}) = \sigma^{\mathbf{R}}_{\mathbf{e}} \otimes \sigma^{\mathbf{C}}(\mathsf{id}_Z \otimes \sigma \otimes \sigma^{\mathbf{R}}_{\mathbf{n}})$. Letting $\sigma^{\mathbf{C}}_{\mathbf{n}} = \sigma \otimes \sigma^{\mathbf{R}}_{\mathbf{n}}$, we thus represent the context wiring above the rule by $\sigma^{\mathbf{C}}_{\mathbf{e}} \otimes \sigma^{\mathbf{C}}(\mathsf{id}_Z \otimes \sigma^{\mathbf{C}}_{\mathbf{n}})$.

In the twigs we are thus given wirings $\sigma^{\mathbf{a}}_{\mathbf{e}}, \sigma^{\mathbf{a}}_{\mathbf{n}}, \sigma^{\mathbf{R}}_{\mathbf{e}}, \sigma^{\mathbf{R}}_{\mathbf{n}}, \sigma^{\mathbf{C}}_{\mathbf{e}}, \sigma^{\mathbf{C}}_{\mathbf{n}}$ and terms $\bar{p}$ (agent) and $\bar{P}$ (redex). The goal is to check that $\bar{p}$ matches $\bar{P}$, and find $\sigma^{\mathbf{C}}, Z, \alpha$ and $\bar{q}$, as we want $\omega^{\mathbf{a}} = \alpha\sigma^{\mathbf{a}}_{\mathbf{e}} \otimes \sigma^{\mathbf{a}}_{\mathbf{n}}$, $\omega^{\mathbf{R}} = \sigma^{\mathbf{R}}_{\mathbf{e}} \otimes \sigma^{\mathbf{R}}_{\mathbf{n}}$, and $\omega^{\mathbf{R}} = \sigma^{\mathbf{C}}_{\mathbf{e}} \otimes \sigma^{\mathbf{C}}(\mathsf{id}_Z \otimes \sigma^{\mathbf{C}}_{\mathbf{n}})$ in the judgment $\omega^{\mathbf{a}},\ \omega^{\mathbf{R}}, \omega^{\mathbf{C}} \vdash \bar{p}, \mathsf{id} \leadsto \bar{P}, \bar{q}$ above the SWX rule.

### 3.6.3 Rule Applications in the Twigs

ABS$'$: During the inference, the ABS$'$ rule adds links to $\sigma^{\mathbf{C}}$. By adding them to $\sigma^{\mathbf{C}}_{\mathbf{e}}$, not $\sigma^{\mathbf{C}}_{\mathbf{n}}$, they are treated like internal edges in $R$, and thus not linked to the parameter via $\mathsf{id}_Z$.

MER$'$: For each $m$-permutation $\pi$, $\bar{\pi}$ is computed; for each partition $\rho \in \bar{\rho}(n, m)$ of $n$ into $m$ (possibly empty) subsets, the tensor product of $m_j$'s are computed, and then the premise is inferred, returning $\sigma^{\mathbf{C}}, \alpha$ and $\bar{q}$.

PER$'$: The premise is inferred, and the resulting $\bar{q}$ are permuted using $\bar{\pi}$ before they are returned.

PAR$'_{\equiv}$: The premise is inferred, and the resulting list of $n$ tensor products are concatenated and returned as one product $\otimes^{m}_{i} q_i$.

PAR$'_{\mathbf{n}}$: Taking PAR$'$ literally, $\sigma^{\mathbf{a}}$ and $\sigma^{\mathbf{C}}$ must be split when performing the subinferences. However, as the inner face of $\sigma^{\mathbf{a}}$ must always match the global outer face of $a$, explicit splitting of $\sigma^{\mathbf{a}}_{\mathbf{e}}, \sigma^{\mathbf{a}}_{\mathbf{n}}, \sigma^{\mathbf{C}}_{\mathbf{e}}, \sigma^{\mathbf{C}}_{\mathbf{n}}$ can be avoided. This also implies that (3.1) need only be solved for links mapping outer names of $g$ (i.e., $X \uplus Z$).

ION$'$: Deconstructing agent and context ions, their controls are checked for equality; for each $u_i \in \mathrm{dom}(\sigma^{\mathbf{C}}_{\mathbf{e}})$ we update $\alpha'$ so that $\alpha'\sigma^{\mathbf{a}}_{\mathbf{e}}(y_i) = \sigma^{\mathbf{C}}_{\mathbf{e}}(u_i)$. Using a fresh $\vec{v}$, the premise is inferred, and $\alpha'$ and $\sigma^{\mathbf{C}\prime}$ are updated for each $u_i \notin \mathrm{dom}(\sigma^{\mathbf{C}}_{\mathbf{e}})$ so that $\sigma^{\mathbf{C}\prime}(\sigma^{\mathbf{C}}_{\mathbf{n}}(u_i))$ is equal to $\alpha'\sigma^{\mathbf{a}}_{\mathbf{e}}(y_i)$ or $\sigma^{\mathbf{a}}_{\mathbf{n}}(y_i)$, depending on whether $y_i \in \mathrm{dom}(\sigma^{\mathbf{a}}_{\mathbf{e}})$ or not.

PAX$'$: At the PAX$'$ rule, we are given $V, (X \uplus Z)$ and $\alpha$, and $\sigma^{\mathbf{a}}$ as $\alpha'\sigma^{\mathbf{a}}_{\mathbf{e}} \otimes \sigma^{\mathbf{a}}_{\mathbf{n}}$, and $\sigma^{\mathbf{C}}$ as $\sigma^{\mathbf{C}}_{\mathbf{e}} \otimes \sigma^{\mathbf{C}\prime}(\mathsf{id}_Z \otimes \sigma^{\mathbf{C}}_{\mathbf{n}})$. We must now solve the equation $\sigma^{\mathbf{a}} = \sigma^{\mathbf{C}}(\mathsf{id}_z \otimes \alpha\tau)$, i.e.

$$\alpha'\sigma^{\mathbf{a}}_{\mathbf{e}} \otimes \sigma^{\mathbf{a}}_{\mathbf{n}} = (\sigma^{\mathbf{C}}_{\mathbf{e}} \otimes \sigma^{\mathbf{C}\prime}(\mathsf{id}_Z \otimes \sigma^{\mathbf{C}}_{\mathbf{n}}))(\mathsf{id}_Z \otimes \alpha\tau) \tag{3.1}$$

for $\alpha'$, $\sigma^{\mathbf{C}\prime}$, $Z$ and $\tau$, where $X_{\mathbf{e}} \cap Z = \emptyset$ (recall $\sigma^{\mathbf{C}}_{\mathbf{e}} : X_{\mathbf{e}} \to$ ).

## 3.7 Nondeterminism

Given these term-based rules and the normal inference grammar, proven correct matching has been expressed in an operational, that is, implementable, form. However, there is still a fair amount of nondeterminism left, but fortunately we can clearly identify where it occurs:

**Grammar selection:** Which branches to select for $\mathcal{D}_{\mathbf{G}}$, $\mathcal{D}_{\mathbf{P}}$, $\mathcal{D}'_{\mathbf{G}}$ and $\mathcal{D}'_{\mathbf{P}}$.

**Tensor grouping:** How to group the tensor product in PAR$_{\equiv}$.

**Children partitioning:** How to partition molecules in MER.

**Prime permutation:** How to permute redex primes in PER.

**Context-redex-parameter wiring:** How to choose $Z, \alpha$ and $\tau$ in PAX.

**Mapping closed links:** How to find an appropriate decomposition of agent wiring in CLO such that closed agent links are matched correctly with closed redex links (i.e., determining $\sigma^{\mathbf{a}}$ and $Y_{\mathbf{R}}$).

When implementing matching, the challenge is to develop a heuristic that will handle typical cases well. In general, an agent-redex pair can lead to many different matches, so in our implementation we return for every inference rule a lazy list of possible matches.

To handle nondeterminism, we return possible matches as follows, bearing in mind that operationally speaking, rules applied below SWX are given agent and redex, while rules above SWX are given agent (, redex) and context:

**Grammar selection:** For $\mathcal{D}_{\mathbf{G}}$ and $\mathcal{D}_{\mathbf{P}}$, we concatenate the returned lazy lists returned from matching each branch in turn. However, if PAX succeeds, there is no reason to attempt a SWX match, as no new matches will result.

For $\mathcal{D}_{\mathbf{G}}'$ and $\mathcal{D}_{\mathbf{P}}'$, we try each branch in turn, returning the first branch that succeeds, as later branches will not find any new matches.

**Tensor grouping:** For given $m$ and $n$ in $\text{PAR}^{\mathcal{E}}_{\equiv}$, we compute all the ways of splitting $[0, \ldots, m-1]$ into $n$ (possibly empty) subsequences, trying out matching for each split. Note that this need only be done for applications of $\text{PAR}^{\mathcal{E}}_{\equiv}$ below the SWX rule.

**Children partitioning:** For given $m$ and $n$ in MER, we compute all the ways of partitioning $\{0, \ldots, m-1\}$ into $n$ (possibly empty) sets, trying out matching for each partitioning.

**Prime permutation:** For given $n$ in $\text{PER}^{\mathcal{E}}$, we compute all $n$-permutations, trying out matching for each permutation. This is done for applications of $\text{PER}^{\mathcal{E}}$ below the SWX rule; above, similar permutations are computed in the MER rule.

**Context-redex-parameter wiring:** Given global agent wiring, we compute the ways of decomposing it into $\sigma(\mathsf{id}_Z \otimes \alpha\tau)$, returning a match for each decomposition.

**Mapping closed links:** We split agent wiring into named and closed links, and postpone the actual mapping of each closed link to redex or context links until some constraint, given by ION or PAX produces it.

Note that even after limiting nondeterminism in this way, we can still in general find several instances of the same match, reached by different inference trees, as we are computing abstract bigraph matches using concrete representations. For instance, matching redex $R = \mathsf{K1}$ in agent $a = \mathit{merge}(\mathsf{K1} \otimes \mathsf{K1})$ produces matches with context $C_1 = \mathit{merge}(\mathsf{id}_1 \otimes \mathsf{K1})$ and context $C_2 = \mathit{merge}(\mathsf{K1} \otimes \mathsf{id}_1)$.

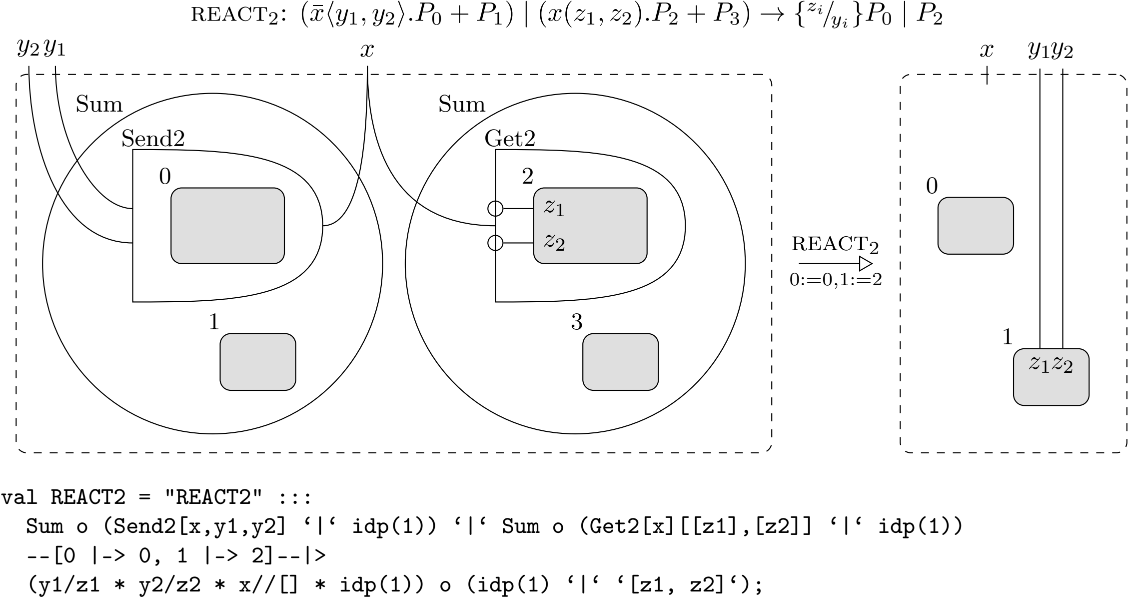
## 3.8 Tool Implementation and Example Modelling

We have implemented a BPL Tool as a reference implementation of binding bigraph matching, and as a toolbox for experimenting with bigraphs. It is written in SML, consists of parser, normalisation and matching kernel, and includes web and command line user interfaces [4].

To ensure correctness, we have implemented normalisation, renaming, regularisation and matching faithfully by implementing one SML function for every inference rule—in the case of matching, two: one for applications above and one for below the SWX rule.

The BPL Tool handles normalisation, regularisation, matching and reaction for the full set of binding bigraphs, and allows construction of simple tactics for prescribing the order in which reaction rules should be applied. The following example output is taken verbatim from the command line interface, which is based on the SMLNJ interactive system; omitted details are indicated by "`[...]`".

As an example, we model the polyadic $\pi$ calculus, running the mobile phone system introduced in Milner's $\pi$ book [17]. The calculus can be modeled by a family of reaction rules $\{\textsc{react}_i \mid i = 0, 1, \ldots\}$, one for each number of names that are to be communicated in a reaction [13]; $\textsc{react}_2$ is shown in Figure 3.14.




```
val REACT2 = "REACT2" :::
  Sum o (Send2[x,y1,y2] `|` idp(1)) `|` Sum o (Get2[x][[z1],[z2]] `|` idp(1))
  --[0 |-> 0, 1 |-> 2]--|>
  (y1/z1 * y2/z2 * x//[] * idp(1)) o (idp(1) `|` `[z1, z2]`);
```

Figure 3.14: $\pi$ calculus reaction rule shown as bigraphs and BPL value.

The signature for the nodes modelling the calculus and the mobile phone system is constructed using `passive` and `atomic` functions as shown in Figure 3.15. For this system, we only need `Send`

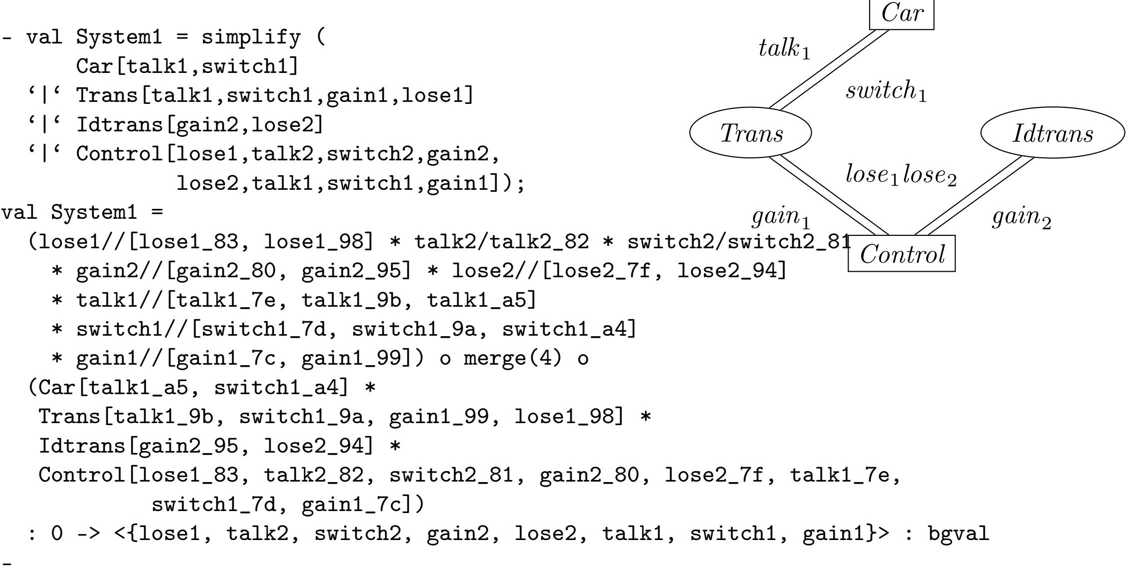
```
(* Pi calculus nodes *)
val Sum   = passive0 ("Sum")
val Send0 = passive ("Send0" -: 0 + 1)
val Get0  = passive ("Get0"  =: 0 --> 1)
val Send2 = passive ("Send2" -: 2 + 1)
val Get2  = passive ("Get2"  =: 2 --> 1)
```

```
(* Mobile phone system nodes *)
val Car      = atomic ("Car" -: 2)
val Trans    = atomic ("Trans" -: 4)
val Idtrans  = atomic ("Idtrans" -: 2)
val Control  = atomic ("Control" -: 8)
```

Figure 3.15: Signature for $\pi$ calculus and mobile phone system nodes.

and `Get` nodes for $\textsc{react}_0$ and $\textsc{react}_2$. Note that all reaction rule nodes are passive, preventing reaction within a guarded expression.

The system consists of a car, one active and one idle transmitter, and a control centre, as shown in Figure 3.16. Internally, a prime product constructed using the `` `|` `` operator is represented by a wiring and $merge_2$ composed with a binary tensor product. The function `simplify` applies various heuristics for producing human-readable bigraph terms, in this case for a prime product of four factors.



```
- val System1 = simplify (
      Car[talk1,switch1]
  `|` Trans[talk1,switch1,gain1,lose1]
  `|` Idtrans[gain2,lose2]
  `|` Control[lose1,talk2,switch2,gain2,
              lose2,talk1,switch1,gain1]);
val System1 =
  (lose1//[lose1_83, lose1_98] * talk2/talk2_82 * switch2/switch2_81
    * gain2//[gain2_80, gain2_95] * lose2//[lose2_7f, lose2_94]
    * talk1//[talk1_7e, talk1_9b, talk1_a5]
    * switch1//[switch1_7d, switch1_9a, switch1_a4]
    * gain1//[gain1_7c, gain1_99]) o merge(4) o
  (Car[talk1_a5, switch1_a4] *
   Trans[talk1_9b, switch1_9a, gain1_99, lose1_98] *
   Idtrans[gain2_95, lose2_94] *
   Control[lose1_83, talk2_82, switch2_81, gain2_80, lose2_7f, talk1_7e,
           switch1_7d, gain1_7c])
  : 0 -> <{lose1, talk2, switch2, gain2, lose2, talk1, switch1, gain1}> : bgval
-
```



Figure 3.16: Definition of the mobile phone system, $System_1$

The definition of these nodes and connections, shown in Figure 3.17, allows the control centre to switch *Car* communication between the two transmitters (supposedly when the car gets closer to the ilde than the active transmitter), and allows the car to talk with the active transmitter. Note that in the BPL tool, we define a node by a rule that unfolds an atomic node into a bigraph corresponding to the defining $\pi$ calculus expression.

Our BPL definition of the initial system in Figure 3.16, $System_1$, is the folded version; as BPL matching is complete, querying the tool reveals the four possible unfolding matches, illustrated in Figure 3.18. Here `mkrules` constructs the internal representation of a rule set, and `print_mv` prettyprints a lazy list of matches, produced by the `matches` function.

Using `react_rule` that simply applies a named reaction rule, and `++` that runs its arguments sequentially, we construct a tactic, `TAC_unfold`, for unfolding all four nodes once, shown in Figure 3.19. Applying this tactic using function `run`, we get an unfolded version of the system.

Querying the BPL Tool for all possible matches in the unfolded system reveals exactly the switch and talk actions, initiated by $\text{REACT}_2$ and $\text{REACT}_0$ rules, respectively, cf. Figure 3.20. Applying the $\pi$ calculus reaction rules for switching, we arrive at $System_2$, where *Car* communication has been switched to the other transmitter, as witnessed by the outer names to which *Car* ports link, as well as the order of names to which *Control* ports link.

This concludes our description of the example highlighting how we can use the BPL Tool to experiment with bigraphical reactive systems.

## 3.9 Conclusion and Future Work

We have developed a provably sound and complete inference system over bigraph terms for inferring legal matches of bigraphical reactive systems. Moreover, we have implemented our BPL Tool, the first implementation of bigraphical reactive systems. We have demonstrated a simple, but concrete, example of how the tool can be used to simulate bigraphical models. We have found it very useful to base this first implementation of bigraphical reactive systems so closely on the developed theory—this has naturally given us greater confidence in the implementation, but the implementation work

<table>
<tr><th>Defining equation</th><th>BPL definition</th></tr>
<tr><td>$Car(talk, switch) \stackrel{\text{def}}{=}$<br>$\overline{talk}.\, Car\langle talk, switch\rangle$<br>$+\, switch(t, s).\, Car\langle t, s\rangle$</td><td><code>val DEF_Car = "DEF_Car" :::<br>  Car[talk,switch]<br>  ----|><br>  Sum o (Send0[talk] o Car[talk,switch]<br>        `|` Get2[switch][[t],[s]]<br>             o (<[t,s]> Car[t,s]))</code></td></tr>
<tr><td>$Trans(talk, switch, gain, lose) \stackrel{\text{def}}{=}$<br>$talk\,.\, Trans\langle talk, switch, gain, lose\rangle$<br>$+\, lose(t, s).\overline{switch}\langle t, s\rangle$<br>$.\, Idtrans\langle gain, lose\rangle$</td><td><code>val DEF_Trans = "DEF_Trans" :::<br>  Trans[talk,switch,gain,lose]<br>  ----|><br>  Sum o (Get0[talk][] o Trans[talk,switch,gain,lose]<br>        `|` Get2[lose][[t],[s]]<br>             o (<[t,s]> Sum o Send2[switch,t,s]<br>                         o Idtrans[gain,lose]))</code></td></tr>
<tr><td>$Idtrans(gain, lose) \stackrel{\text{def}}{=}$<br>$gain(t, s).\, Trans\langle t, s, gain, lose\rangle$</td><td><code>val DEF_Idtrans = "DEF_Idtrans" :::<br>  Idtrans[gain, lose]<br>  ----|><br>  Sum o Get2[gain][[t],[s]]<br>   o (<[t,s]> Trans[t,s,gain,lose])</code></td></tr>
<tr><td>$Control(lose_1, talk_2, switch_2, gain_2,$<br>$lose_2, talk_1, switch_1, gain_1) \stackrel{\text{def}}{=}$<br>$\overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle$<br>$.\, Control\langle lose_2, talk_1, switch_1, gain_1,$<br>$lose_1, talk_2, switch_2, gain_2\rangle$</td><td><code>val DEF_Control = "DEF_Control" :::<br>  Control[lose1,talk2,switch2,gain2,<br>          lose2,talk1,switch1,gain1]<br>  ----|><br>  Sum o Send2[lose1,talk2,switch2]<br>   o Sum o Send2[gain2,talk2,switch2]<br>    o Control[lose2,talk1,switch1,gain1,<br>              lose1,talk2,switch2,gain2]</code></td></tr>
</table>

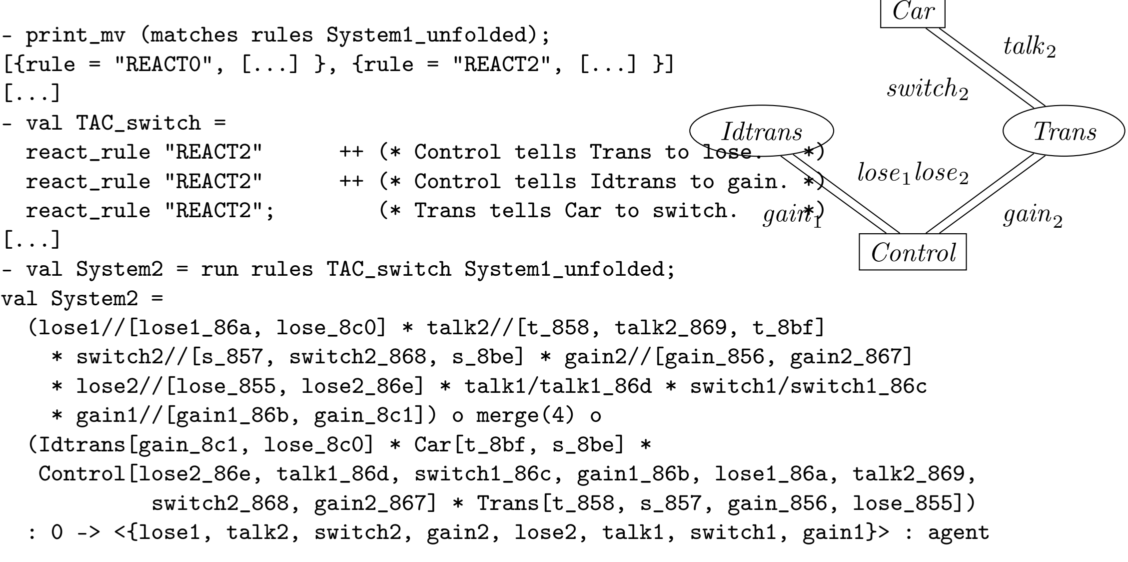
Figure 3.17: Definitions of Car, Trans, Idtrans and Control nodes.

```
- val rules = mkrules [REACT0, REACT2, DEF_Car, DEF_Trans, DEF_Idtrans, DEF_Control];
[...]
- print_mv (matches rules System1);
[{rule = "DEF_Car",
 context
  = (lose1//[lose1_d3, lose1_d6] * talk2/talk2_d2 * switch2/switch2_d1
      * gain2//[gain2_d0, gain2_d5] * lose2//[lose2_cf, lose2_d4]
      * talk1//[talk, talk1_ce, talk1_d9]
      * switch1//[switch, switch1_cd, switch1_d8]
      * gain1//[gain1_cc, gain1_d7]) o
    (merge(4) o
     (Trans[talk1_d9, switch1_d8, gain1_d7, lose1_d6] *
      Idtrans[gain2_d5, lose2_d4] *
      Control[lose1_d3, talk2_d2, switch2_d1, gain2_d0, lose2_cf,
               talk1_ce, switch1_cd, gain1_cc])),
 parameter = idx0},
{rule = "DEF_Control", [...] },
{rule = "DEF_Idtrans", [...] },
{rule = "DEF_Trans", [...] }]
```

Figure 3.18: Determining which rules match $System_1$.

```
- val TAC_unfold =
  react_rule "DEF_Car"     ++ react_rule "DEF_Trans"   ++
  react_rule "DEF_Idtrans" ++ react_rule "DEF_Control";
[...]
- val System1_unfolded = run rules TAC_unfold System1;
val System1_unfolded =
  (lose1//[lose1_3f9, lose1_419, lose_441, lose_459, lose_45d]
    * talk2//[talk2_3f8, talk2_40f, talk2_418]
    * switch2//[switch2_3f7, switch2_40e, switch2_417]
    * gain2//[gain2_3f6, gain2_410, gain_431, gain_438]
    * lose2//[lose2_3fd, lose_430]
    * talk1//[talk1_3fc, talk_460, talk_465, talk_482, talk_485]
    * switch1//[switch1_3fb, switch_447, switch_45f, switch_480, switch_481]
    * gain1//[gain1_3fa, gain_442, gain_45e]) o merge(4) o
  (Sum o merge(2) o
   (Send0[talk_485] o Car[talk_482, switch_481] *
    Get2[switch_480][[t_47d], [s_47c]] o
    (<[s_47c, t_47d]> Car[t_47d, s_47c])) *
   Sum o merge(2) o
   (Get0[talk_465] o Trans[talk_460, switch_45f, gain_45e, lose_45d] *
    Get2[lose_459][[t_446], [s_445]] o
    (<[s_445, t_446]>
      Sum o (Send2[switch_447, t_446, s_445] o Idtrans[gain_442, lose_441]))) *
   Sum o Get2[gain_438][[t_433], [s_432]] o
   (<[s_432, t_433]> Trans[t_433, s_432, gain_431, lose_430]) *
   Sum o
   (Send2[lose1_419, talk2_418, switch2_417] o
    (Sum o
     (Send2[gain2_410, talk2_40f, switch2_40e] o
      Control[lose2_3fd, talk1_3fc, switch1_3fb, gain1_3fa, lose1_3f9,
               talk2_3f8, switch2_3f7, gain2_3f6]))))
  : 0 -> <{lose1, talk2, switch2, gain2, lose2, talk1, switch1, gain1}> : agent
```

Figure 3.19: Unfolding $System_1$, using the TAC_unfold tactic.



```
- print_mv (matches rules System1_unfolded);
[{rule = "REACT0", [...] }, {rule = "REACT2", [...] }]
[...]
- val TAC_switch =
  react_rule "REACT2"      ++ (* Control tells Trans to lose.   *)
  react_rule "REACT2"      ++ (* Control tells Idtrans to gain. *)
  react_rule "REACT2";        (* Trans tells Car to switch.     *)
[...]
- val System2 = run rules TAC_switch System1_unfolded;
val System2 =
  (lose1//[lose1_86a, lose_8c0] * talk2//[t_858, talk2_869, t_8bf]
    * switch2//[s_857, switch2_868, s_8be] * gain2//[gain_856, gain2_867]
    * lose2//[lose_855, lose2_86e] * talk1/talk1_86d * switch1/switch1_86c
    * gain1//[gain1_86b, gain_8c1]) o merge(4) o
  (Idtrans[gain_8c1, lose_8c0] * Car[t_8bf, s_8be] *
   Control[lose2_86e, talk1_86d, switch1_86c, gain1_86b, lose1_86a, talk2_869,
           switch2_868, gain2_867] * Trans[t_858, s_857, gain_856, lose_855])
  : 0 -> <{lose1, talk2, switch2, gain2, lose2, talk1, switch1, gain1}> : agent
-
```



Figure 3.20: Checking possible matches, then switching to $System_2$, using the TAC_switch tactic.

has also helped to debug the developed theory.

There are lots of interesting avenues for future work. While the current implementation of BPL Tool is efficient enough to experiment with small examples, we will try to make it more efficient by using a number of different techniques: we plan to investigate how to prune off invalid matches quickly, for instance by making use of sorting information [3]. Moreover, we will investigate to what extent we can capture the link graph matching via a constraint-based algorithm.

We also plan to investigate smarter ways of combining matching and rewriting. As a starting point, we have made it possible for users to combine *tactics* to inform the tool in which order it should attempt to apply reaction rules.

Jean Krivine and Robin Milner are currently investigating stochastic bigraphs, which will be particularly important for simulation of real systems. We hope that our detailed analysis of matching for binding bigraphs will make it reaonably straightforward to extend it to stochastic bigraphs.

## Acknowledgements

The authors would like to thank Mikkel Bundgaard and anonymous referees for detailed comments to earlier versions of the paper.

## 3.10 Bibliography

[1] Lars Birkedal, Troels Christoffer Damgaard, Arne John Glenstrup, and Robin Milner. Matching of bigraphs. In *Proceedings of Graph Transformation for Verification and Concurrency Workshop 2006*, Electronic Notes in Theoretical Computer Science. Elsevier, August 2006.

[2] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Troels Hildebrandt, and Henning Niss. Bigraphical models of context-aware systems. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and*

*Computation Structure*, volume 3921 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, March 2006. ISBN 3-540-33045-3.

[3] Lars Birkedal, Søren Debois, and Thomas Troels Hildebrandt. Sortings for reactive systems. In Christel Baier and Holger Hermanns, editors, *Proceedings of the 17th International Conference on Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 248–262. Springer-Verlag, August 2006.

[4] The BPL Group. BPLweb—the BPL tool web demo, 2007. URL `http://tiger.itu.dk:8080/bplweb/`. IT University of Copenhagen, Denmark. Prototype.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Francisco Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The Maude 2.0 System. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications (RTA 2003)*, volume 2706 of *Lecture Notes in Computer Science*, pages 76–87. Springer-Verlag, June 2003.

[7] Troels Christoffer Damgaard. Syntactic theory for bigraphs. Master's thesis, IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen V, December 2006.

[8] Troels Christoffer Damgaard and Lars Birkedal. Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1–2):58–77, 2006.

[9] Troels Christoffer Damgaard, Arne John Glenstrup, Lars Birkedal, and Robin Milner. An inductive characterization of matching in binding bigraphs. *to appear*, 2011.

[10] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[11] James Jianghai Fu. Directed graph pattern matching and topological embedding. *Journal of Algorithms*, 22(2):372–391, 1997.

[12] Ole Høgh Jensen. *Mobile Processes in Bigraphs*. PhD thesis, Univ. of Cambridge, 2008. Forthcoming.

[13] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge, February 2004.

[14] Javier Larrosa and Gabriel Valiente. Constraint satisfaction algorithms for graph pattern matching. *Journal of Mathematical Structures in Computer Science*, 12:403–422, 2002.

[15] James Judi Leifer and Robin Milner. Transition systems, link graphs and Petri nets. Technical Report UCAM-CL-TR-598, University of Cambridge, August 2004.

[16] The Maude Team. The Maude system, 2007. `http://maude.cs.uiuc.edu/`.

[17] Robin Milner. *Communicating and Mobile Systems: The $\pi$-Calculus*. Cambridge University Press, 1999.

[18] Robin Milner. Bigraphs whose names have multiple locality. Technical Report UCAM-CL-TR-603, University of Cambridge, September 2004.

[19] Vladimiro Sassone and Paweł Sobociński. Reactive systems over cospans. In *Proceedings of Logic in Computer Science (LICS'05)*, pages 311–320. IEEE Press, 2005.

[20] Jeffrey D. Ullman. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1): 31–42, 1976.

[21] Albert Zündorf. Graph pattern matching in PROGRES. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *TAGT*, volume 1073 of *Lecture Notes in Computer Science*, pages 454–468. Springer-Verlag, 1994. ISBN 3-540-61228-9.

## 3.A Auxiliary Technologies Details

### 3.A.1 Normalising

We define a normalisation relation $t \downarrow_{\mathbf{B}} t'$ for elementary bigraphs: $\mathit{merge}_n, \ulcorner X \urcorner, \vec{y}/\vec{X}, K_{\vec{y}(\vec{X})}$ and $\pi$ as shown in Figure 3.21, and inductively for operations: abstraction $(X)P$, product $\bigotimes_i^n B_i$ and composition $B_1 B_2$ as shown in Figure 3.22, where the notation $\sigma\lfloor^Y$ means $\{X \mapsto y \in \sigma \mid y \in Y\}$.

$$\text{Bmer}\ \frac{N \equiv (\emptyset)(\mathsf{id}_\emptyset \otimes \mathit{merge}_n) \bigotimes_{i \in n} \ulcorner \mathsf{id}_\emptyset \urcorner \qquad P \equiv (\mathsf{id}_\emptyset \otimes {}^{([])}/_{([])})N \qquad D \equiv \mathsf{id}_\emptyset \otimes (\bigotimes_{i \in 1} P)\mathsf{id}_n}{\mathit{merge}_n \downarrow_{\mathbf{B}} (\mathsf{id}_\emptyset \otimes \mathsf{id}_{[\emptyset]})D}$$

$$\text{Bcon}\ \frac{\begin{array}{c} N \equiv (\emptyset)(\mathsf{id}_X \otimes \mathit{merge}_1) \bigotimes_{i \in 1} (\mathsf{id}_X \otimes \mathsf{id}_1) \ulcorner X \urcorner \\ P \equiv (\mathsf{id}_X \otimes {}^{([])}/_{([])})N \qquad D \equiv \mathsf{id}_\emptyset \otimes (\bigotimes_{i \in 1} P)\mathsf{id}_{(X)} \end{array}}{\ulcorner X \urcorner \downarrow_{\mathbf{B}} (\mathsf{id}_X \otimes \mathsf{id}_{[\emptyset]})D}$$

$$\text{Bwir}\ \frac{}{\vec{y}/\vec{X} \downarrow_{\mathbf{B}} (\vec{y}/\vec{X} \otimes \mathsf{id}_{[]})(\mathsf{id}_X \otimes \mathsf{id}_0 \mathsf{id}_0)}$$

$$\text{Bion}\ \frac{\begin{array}{c} X = \{\vec{X}\} \qquad Y = \{\vec{y}\} \qquad M \equiv (\mathsf{id}_\emptyset \otimes K_{\vec{y}(\vec{X})})(X)(\mathsf{id}_X \otimes \mathit{merge}_1) \bigotimes_{i \in 1} (\mathsf{id}_X \otimes \mathsf{id}_1) \ulcorner X \urcorner \\ N \equiv (\emptyset)(\mathsf{id}_Y \otimes \mathit{merge}_1) \bigotimes_{i \in 1} M \qquad P \equiv (\mathsf{id}_Y \otimes {}^{([])}/_{([])})N \qquad D \equiv \mathsf{id}_\emptyset \otimes (\bigotimes_{i \in 1} P)\mathsf{id}_{(X)} \end{array}}{K_{\vec{y}(\vec{X})} \downarrow_{\mathbf{B}} (\mathsf{id}_Y \otimes \mathsf{id}_{[\emptyset]})D}$$

$$\text{Bper}\ \frac{\begin{array}{c} Y_i = \{\vec{y}_i\} \qquad N_i \equiv (Y_i)(\mathsf{id}_{Y_i} \otimes \mathit{merge}_1) \bigotimes_{j \in 1} (\mathsf{id}_{Y_i} \otimes \mathsf{id}_1) \ulcorner Y_i \urcorner \\ P_i \equiv (\mathsf{id}_\emptyset \otimes \widehat{\vec{y}_i/\vec{y}_i})N_i \qquad D \equiv \mathsf{id}_\emptyset \otimes (\bigotimes_{i \in m} P_i)\pi \end{array}}{\pi : \langle m, \vec{X}, X \rangle \to \langle m, \vec{Y}, X \rangle \downarrow_{\mathbf{B}} (\mathsf{id}_\emptyset \otimes \mathsf{id}_{\vec{Y}})D}$$

Figure 3.21: Inference rules for normalising elementary bigraph expressions

### 3.A.2 Renaming

Let a *link namer* be a map $\mu$ mapping every link $l$ (outer name or edge) in its domain to a pair $(E, X)$, where $E$ is a set of names used internally to compose the link, and $X$ are the inner names linking to $l$. We let $\mathcal{V}_i(Y, \mu) = \bigcup_{y \in Y, y \mapsto (X_1, X_2) \in \mu} X_i$ and define link namer composition by

$$\begin{array}{rcl} \mu_1 \circ \mu_2 & = & \{y_1 \mapsto (E_1 \cup X_1 \cup V_1, V_2) \mid y_1 \mapsto (E_1, X_1) \in \mu_1 \wedge V_i = \mathcal{V}_i(X_1, \mu_2)\} \\ & \cup & \{y_2 \mapsto (E_2, X_2) \in \mu_2 \mid \forall y_1 \mapsto (E_1, X_1) \in \mu_1 : y_2 \notin X_1\}, \end{array}$$

essentially composing links of $\mu_1$ with those of $\mu_2$, and adding closed links from $\mu_2$.

We then define a function *linknames*, mapping terms to link namers, by the equations given in Figure 3.23. By using the link namers of immediate subterms, we can determine whether a term can be normalised without name clashes. To this end, we define a predicate *normalisable* by the

$$\text{Babs}\ \dfrac{\begin{array}{c} b \downarrow_{\mathbf{B}} (z/W \otimes \mathsf{id}_{([Y])})(\mathsf{id}_{\emptyset} \otimes (\bigotimes_{i\in 1}(\mathsf{id}_Z \otimes \widehat{\vec{y}/\vec{X}})(W)G)\mathsf{id}_I) \\ \vec{z}^X = [z_j \leftarrow \vec{z} \mid z_j \in X] \qquad \vec{z}^{\bar{X}} = [z_j \leftarrow \vec{z} \mid z_j \notin X] \\ \vec{W}^X = [W_j \leftarrow \vec{W} \mid z_j \in X] \qquad \vec{W}^{\bar{X}} = [W_j \leftarrow \vec{W} \mid z_j \notin X] \\ W^X = \{\vec{W}^X\} \quad W^{\bar{X}} = \{\vec{W}^{\bar{X}}\} \quad U = \{\vec{y}\vec{z}^X\} \quad N \equiv (W^X \cup W)G \quad P \equiv (\mathsf{id}_{W^{\bar{X}}} \otimes \widehat{\vec{y}\vec{z}^X/\vec{X}\vec{W}^X})N \end{array}}{(X)b \downarrow_{\mathbf{B}} (z^{\bar{X}}/W^{\bar{X}} \otimes \mathsf{id}_{([U])})(\mathsf{id}_{\emptyset} \otimes (\bigotimes_{i\in 1} P)\mathsf{id}_I)}$$

$$\text{Bten}\ \dfrac{\begin{array}{c} b_i \downarrow_{\mathbf{B}} (\omega_i \otimes \mathsf{id}_{(\vec{Y}_i)})D_i \qquad D_i \equiv \alpha_i \otimes (\bigotimes_{j\in n_i} P_i^j)\pi_i : I_i \to \langle n_i, \vec{Y}_i, Y_i\rangle \\ \omega = \bigotimes_{i\in n}\omega_i \quad \alpha = \bigotimes_{i\in n}\alpha_i \quad \mathsf{id}_{(\vec{Y})} = \bigotimes_{i\in n}\mathsf{id}_{(\vec{Y}_i)} \quad \pi = \bigotimes_{i\in n}\pi_i \\ P = \bigotimes_{j\in n}\bigotimes_{i\in n_j} P_i^j \qquad D \equiv \alpha \otimes P\pi \end{array}}{\bigotimes_{i\in n} b_i \downarrow_{\mathbf{B}} (\omega \otimes \mathsf{id}_{(\vec{Y})})D}$$

$$\text{Ccom}\ \dfrac{\sigma = (\mathsf{id}_Z \otimes \alpha)(\mathsf{id}_Z \otimes y/X)}{(\mathsf{id}_Z \otimes (\alpha \otimes \mathsf{id}_1)\ulcorner Y \urcorner)\bigotimes_{i\in 1}(\mathsf{id}_Z \otimes \widehat{\vec{y}/\vec{X}})(X)(\mathsf{id}_U \otimes \mathit{merge}_n)\bar{S} \downarrow_{\bar{\mathbf{S}}} \sigma, \bar{S}}$$

$$\text{Mcom}\ \dfrac{(\mathsf{id}_Z \otimes N)\bar{P} \downarrow_{\mathbf{N}} \sigma, N' \quad \vec{X}' = \sigma^{-1}(\vec{X}) \quad Z' = \sigma^{-1}(Z) \quad Y' = \sigma^{-1}(Y) \quad \sigma' = \mathsf{id}_{\{\vec{y}\}} \otimes \sigma\downharpoonright^{Z \uplus Y}}{(\mathsf{id}_Z \otimes (\mathsf{id}_Y \otimes K_{\vec{y}(\vec{X})})N)\bar{P} \downarrow_{\bar{\mathbf{S}}} \sigma', \bigotimes_{i\in 1}(\mathsf{id}_{Z' \uplus Y'} \otimes K_{\vec{y}(\vec{X}')})N'}$$

$$\text{Ncom}\ \dfrac{\begin{array}{c} P_i : \langle m_i, \vec{X}_i, X_i\rangle \to \langle 1, (Y_i), Y_i \uplus W_i\rangle \\ \bigotimes_{i\in n}\bar{P}_i = \bigotimes_{i\in k} P_i \qquad \bar{P}_i : I_i \to \langle n_i, \vec{Y}_i, \{\vec{Y}_i\} \uplus Z_i\rangle \qquad (\mathsf{id}_{Z_i} \otimes S_i)\bar{P}_i \downarrow_{\bar{\mathbf{S}}} \sigma_i, \bar{S}_i \\ \bar{S} = \bigotimes_{i\in n}\bar{S}_i : I \to \langle n', Z' \uplus Y'\rangle \quad \sigma = \bigotimes_{i\in n}\sigma_i \quad X' = \sigma^{-1}(X) \quad Z' = \sigma^{-1}(Z) \quad Y' = \sigma^{-1}(Y) \end{array}}{\big(\mathsf{id}_Z \otimes (X)(\mathsf{id}_Y \otimes \mathit{merge}_n)\bigotimes_{i\in n} S_i\big)\bigotimes_{i\in k} P_i \downarrow_{\mathbf{N}} \sigma, (X')(\mathsf{id}_{Z' \uplus Y'} \otimes \mathit{merge}_{n'})\bar{S}}$$

$$\text{Pcom}\ \dfrac{(\mathsf{id}_Z \otimes N)\bar{P} \downarrow_{\mathbf{N}} \sigma, N' \quad W = \sigma^{-1}(Z \uplus Z') \quad \vec{X}' = \sigma^{-1}(\vec{X}) \quad \sigma' = \sigma\downharpoonright^{Z \uplus Z'}}{(\mathsf{id}_Z \otimes (\mathsf{id}_{Z'} \otimes \widehat{\vec{y}/\vec{X}})N)\bar{P} \downarrow_{\mathbf{P}} \sigma', (\mathsf{id}_W \otimes \widehat{\vec{y}/\vec{X}'})N'}$$

$$\text{Bcom}\ \dfrac{\begin{array}{c} b_1 \downarrow_{\mathbf{B}} \Big(\omega_1 \otimes \mathsf{id}_{(\vec{U}^1)}\Big)D_1 : \langle m', \vec{X}', X' \uplus Z\rangle \to \langle n, \vec{U}^1, U^1 \uplus W\rangle \\ b_2 \downarrow_{\mathbf{B}} \Big(\omega_2 \otimes \mathsf{id}_{(\vec{U}^2)}\Big)D_2 : \langle m, \vec{X}, X \uplus U\rangle \to \langle m', \vec{U}^2, U^2 \uplus Z\rangle \\ D_1 \equiv \alpha_1 \otimes \Big(\bigotimes_{i\in n} P_i^1\Big)\pi_1 : \langle m', \vec{X}', X' \uplus Z\rangle \to \langle n, \vec{U}^1, U^1 \uplus V^1 \uplus W^1\rangle \\ D_2 \equiv \alpha_2 \otimes \Big(\bigotimes_{i\in m'} P_i^2\Big)\pi_2 : \langle m, \vec{X}, X \uplus U\rangle \to \langle m', \vec{U}^2, U^2 \uplus V^2 \uplus W^2\rangle \\ P_i^1 : \langle m_i', \vec{X}_i', X_i'\rangle \to \langle (U_i^1), U_i^1 \uplus V_i^1\rangle \qquad P_i^2 : \langle m_i'', \vec{X}_i'', X_i''\rangle \to \langle (U_i^2), U_i^2 \uplus V_i^2\rangle \\ \omega_1 : V^1 \uplus W^1 \to W \qquad \omega_2 : V^2 \uplus W^2 \to Z \qquad \alpha_1 : Z \to W^1 \qquad \alpha_2 : U \to W^2 \\ V^2 = \biguplus_{i\in m'} V_i^2 \qquad \bigotimes_{i\in m'} P^2_{\pi_1^{-1}(i)} = \bigotimes_{i\in n}\bar{P}_i \qquad \bar{P}_i : I_i' \to \langle m_i', \vec{X}_i', X_i' \uplus Z_i'\rangle \\ (\mathsf{id}_{Z_i'} \otimes P_i^1)\bar{P}_i \downarrow_{\mathbf{P}} \sigma_i, P_i \qquad \sigma = \mathsf{id}_U \otimes \bigotimes_{i\in n}\sigma_i \qquad \omega = \omega_1(\alpha_1\omega_2(\alpha_2 \otimes \mathsf{id}_{V^2}) \otimes \mathsf{id}_{V^1})\sigma \\ \pi = \overline{\pi_1}_{\vec{X}''}\pi_2 \qquad D \equiv \mathsf{id}_U \otimes \big(\bigotimes_{i\in n} P_i\big)\pi \end{array}}{b_1 b_2 \downarrow_{\mathbf{B}} \Big(\omega \otimes \mathsf{id}_{(\vec{U}^1)}\Big)D}$$

Figure 3.22: Inference rules for normalising bigraph abstraction, product and composition expressions

$$
\begin{array}{lcl}
\mathit{linknames}(\mathit{merge}_n) & = & \{\} \\
\mathit{linknames}(\ulcorner X \urcorner) & = & \{x \mapsto (\{\}, \{x\}) \mid x \in X\} \\
\mathit{linknames}(\vec{y}/\vec{X}) & = & \{y_i \mapsto (\{\}, X_i) \mid i \in |\vec{y}|\} \\
\mathit{linknames}(K_{\vec{y}(\vec{e}/\vec{X})}) & = & \{y_i \mapsto (\{\}, \{\}) \mid i \in |\vec{y}|\} \cup \{e_i \mapsto (\{\}, X_i) \mid i \in |\vec{X}|\} \\
\mathit{linknames}(\pi : \ \to \langle m, \vec{X}, X \rangle) & = & \{x \mapsto (\{\}, \{x\}) \mid x \in X\} \\
\mathit{linknames}((Y)P) & = & \mathit{linknames}(P) \\
\mathit{linknames}(\bigotimes_i t_i) & = & \bigcup_i \mathit{linknames}(t_i) \\
\mathit{linknames}(t_1 t_2) & = & \mathit{linknames}(t_1) \circ \mathit{linknames}(t_2)
\end{array}
$$

Figure 3.23: Function for determining which names are used internally to compose a link

$$
\begin{array}{lcl}
\mathit{normalisable}(\mathit{merge}_n) & = & \mathit{true} \\
\mathit{normalisable}(\ulcorner X \urcorner) & = & \mathit{true} \\
\mathit{normalisable}(\vec{y}/\vec{X}) & = & \mathit{true} \\
\mathit{normalisable}(K_{\vec{y}(\vec{e}/\vec{X})}) & = & \mathit{true} \\
\mathit{normalisable}(\pi : \ \to \langle m, \vec{X}, X \rangle) & = & \mathit{true} \\
\mathit{normalisable}((Y)P) & = & \mathit{normalisable}(P) \\
\mathit{normalisable}(\bigotimes_i t_i) & = & \bigwedge_i \mathit{normalisable}(t_i) \\
 & & \wedge (\forall i \neq j : E_i \cap E_j = \emptyset) \\
 & & \text{where } \begin{array}[t]{lcl} \mu_i & = & \mathit{linknames}(t_i) \\ E_i & = & \bigcup_{y \mapsto (E,X) \in \mu_i} E \end{array} \\
\mathit{normalisable}(t_1 t_2) & = & \mathit{normalisable}(t_1) \wedge \mathit{normalisable}(t_2) \\
 & & \wedge (\forall l_1 \neq l_2 : \mu_{\mathbf{E}}(l_1) \cap \mu_{\mathbf{E}}(l_2) = \emptyset) \\
 & & \text{where } \begin{array}[t]{lcl} \mu_i & = & \mathit{linknames}(t_i) \\ \mu & = & \mu_1 \circ \mu_2 \\ \mu_{\mathbf{E}}(l) & = & E, \text{ if } \mu(l) = (E, X) \end{array}
\end{array}
$$

Figure 3.24: Function for determining whether a (well-formed) term is normalisable

equations given in Figure 3.24. We basically just require, that at no level in the term does two different links share any internal names.

Renaming is achieved by the judgment $U \vdash \alpha, t \downarrow_{\boldsymbol{\beta}} t', \beta \dashv V$, where $U$ is a set of used names and $\alpha$ renames $t$'s inner names to those of $t'$, while $\beta$ renames $t$'s outer names to those of $t'$ and $V$ extends $U$ with names used in $t'$. The system of rules for inferring this judgment is given in Figure 3.25.

$$\text{Rmer}\ \frac{}{U \vdash \mathsf{id}_\emptyset, merge_n \downarrow_{\boldsymbol{\beta}} merge_n, \mathsf{id}_\emptyset \dashv U} \qquad \text{Rcon}\ \frac{X' = \alpha(X)}{U \vdash \alpha, \ulcorner X \urcorner \downarrow_{\boldsymbol{\beta}} \ulcorner X' \urcorner, \alpha \dashv U}$$

$$\text{Rwir}\ \frac{Z = \{\vec{z}\} \quad Z \cap U = \emptyset \quad |Z| = |\vec{z}| = |\vec{y}| \qquad \vec{X}' = \alpha(\vec{X}) \quad \beta = \{y_i \mapsto z_i\}}{U \vdash \alpha, \vec{y}/\vec{X} \downarrow_{\boldsymbol{\beta}} \vec{z}/\vec{X}', \beta \dashv U \cup Z}$$

$$\text{Rion}\ \frac{Z = \{\vec{z}\} \quad Z \cap U = \emptyset \quad |Z| = |\vec{z}| = |\vec{y}| \qquad \vec{X}' = \alpha(\vec{X}) \quad \beta = \{y_i \mapsto z_i\}}{U \vdash \alpha, K_{\vec{y}(\vec{X})} \downarrow_{\boldsymbol{\beta}} K_{\vec{z}(\vec{X}')}, \beta \dashv U \cup Z}$$

$$\text{Rper}\ \frac{X' = \alpha(X) \quad \vec{X}' = \alpha(\vec{X}) \quad \vec{Y}' = \alpha(\vec{Y})}{U \vdash \alpha, \pi : \langle m, \vec{X}, X \rangle \to \langle m, \vec{Y}, X \rangle \downarrow_{\boldsymbol{\beta}} \pi : \langle m, \vec{X}', X' \rangle \to \langle m, \vec{Y}', X' \rangle, \alpha \dashv U}$$

$$\text{Rabs}\ \frac{U \vdash \alpha, t \downarrow_{\boldsymbol{\beta}} t', \beta \dashv V \quad X' = \beta(X)}{U \vdash \alpha, (X)t \downarrow_{\boldsymbol{\beta}} (X')t', \beta \dashv V}$$

$$\text{Rten}\ \frac{t_i : \langle m_i, \vec{X}_i, X_i \rangle \to J_i \quad \alpha_i = \alpha \restriction_{X_i} \quad U_i \vdash \alpha_i, t_i \downarrow_{\boldsymbol{\beta}} t'_i, \beta_i \dashv U_{i+1} \quad \beta = \bigotimes_{i \in n} \beta_i}{U_0 \vdash \alpha, \bigotimes_{i \in n} t_i \downarrow_{\boldsymbol{\beta}} \bigotimes_{i \in n} t'_i, \beta \dashv U_n}$$

$$\text{Rcom}\ \frac{U_1 \vdash \alpha_1, t_2 \downarrow_{\boldsymbol{\beta}} t'_2, \beta_1 \dashv U_2 \quad U_2 \vdash \beta_1, t_1 \downarrow_{\boldsymbol{\beta}} t'_1, \beta_2 \dashv V_2}{U_1 \vdash \alpha_1, t_1 t_2 \downarrow_{\boldsymbol{\beta}} t'_1 t'_2, \beta_2 \dashv V_2}$$

Figure 3.25: Renaming rules

### 3.A.3 Regularising

The system of rules for inferring a permutation-free term representing a regular bigraph is given in Figure 3.26.

$$\alpha \; \frac{}{\ulcorner\alpha\urcorner \mathsf{id}_{(X)} \hookrightarrow \ulcorner\alpha\urcorner} \qquad \mathrm{M} \; \frac{N\pi \hookrightarrow N'}{(\mathsf{id}_Z \otimes K_{\vec{y}(\vec{X})})N\pi \hookrightarrow (\mathsf{id}_Z \otimes K_{\vec{y}(\vec{X})})N'}$$

$$\mathrm{N} \; \frac{S_i : \langle m_i, \vec{X}_i \rangle \to J_i \qquad \pi = \underline{\pi'}^{\vec{X}} \qquad S_i \pi_i'^{\vec{X}} \hookrightarrow S_i'}{((X)(\mathsf{id}_Y \otimes \mathit{merge}_n) \bigotimes_{i \in n} S_i)\pi' \hookrightarrow (X)(\mathsf{id}_Y \otimes \mathit{merge}_n) \bigotimes_{i \in n} S'_{\pi(i)}}$$

$$\mathrm{D} \; \frac{\pi = \bigotimes_{i \in n} \pi_i \qquad \pi_i : I_i' \to I_i \qquad N_i : I_i \to J_i \qquad N_i \pi_i \hookrightarrow N_i'}{\alpha \otimes (\bigotimes_{i \in n} (\mathsf{id}_{Z_i} \otimes \widehat{\vec{y}_i / \vec{X}_i}) N_i)\pi \hookrightarrow \alpha \otimes \bigotimes_{i \in n} (\mathsf{id}_{Z_i} \otimes \widehat{\vec{y}_i / \vec{X}_i}) N_i'}$$

$$\mathrm{B} \; \frac{D \hookrightarrow D'}{(\omega \otimes \mathsf{id}_{(\vec{X})})D \hookrightarrow (\omega \otimes \mathsf{id}_{(\vec{X})})D'}$$

Figure 3.26: Removing nontrivial permutations from regular bigraphs.

# Chapter 4

# The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems

**Espen Højsgaard and Arne J. Glenstrup**

We present the BPL Tool, a first implementation of bigraphical reactive systems with binding. The BPL Tool provides manipulation, simulation and visualisation of bigraphs and bigraphical reactive systems, and can be used either through the included web and command line user interfaces or as a programming library.

**Preface** This chapter consists of the technical report

> E. Højsgaard and A. J. Glenstrup. *The BPL Tool: A Tool for Experimenting with Bigraphical Reactive Systems.* Technical Report TR-2011-145, IT University of Copenhagen, October 2011.

## 4.1 Introduction

The theory of bigraphical reactive systems [19] provides a general meta-model for describing and analyzing mobile and distributed ubiquitous systems. Bigraphical reactive systems form a graphical model of computation in which graphs that embody both locality and connectivity can be reconfigured using reaction rules. So far it has been shown how to use the theory to recover behavioural theories for various process calculi [15–17] and how to use the theory to model context-aware systems [6].

In this report, we describe the BPL Tool, a first prototype implementation of bigraphical reactive systems, which can be used for experimenting with bigraphical models with binding. The theoretical foundations for the implementation have been developed in detail in [13], but in summary the BPL Tool is based on Damgaard et al.'s axiomatization of binding bigraphs [9] (i.e. it is term based) and the inductive characterization of matching [5] by the same authors. In [13] we have extended the inductive characterization from *graphs* to a *term* representation of bigraphs and have given an algorithmic interpretation of this characterization of matching. This required the development of some additional algorithms for bigraph terms: normalisation, renaming, and regularisation.

The BPL Tool is written in SML, consists of parser, normalisation and matching kernel, and includes web and command line user interfaces. To ensure correctness, we have implemented normalisation, renaming, regularisation and matching faithfully by implementing one SML function for every inference rule – in the case of matching, two: one for applications above and one for below the SWX rule.

The BPL Tool has been used to model the following:

| | |
|---|---|
| **the ARAN protocol** | (Bentzen, [2]) |
| **the GeoCast protocol** | (Niss, unpublished) |
| **IEEE 802.11 MAC 4-way handshake** | (Bentzen, [2]) |
| **the Insider Problem** | (Bentzen, [2]) |
| **a mobile phone system** | (Glenstrup, included in this report) |
| **platographical models** | (Elsborg, [12]) |
| **WS-BPEL and HomeBPEL** | (Bundgaard et al., [8]) |

The BPL Tool is available at `http://www.itu.dk/research/pls/wiki/index.php/BPL_Tool` where also some additional material can be found, such as API documentation and slides from presentations.

### 4.1.1 Related work

A number of implementations of bigraphs are being developed at various institutions. Unfortunately, it is hard to find the implementations themselves or papers describing them – until now, this has also been the case for the BPL Tool – but here is a complete list of the implementations of which we are aware:

**BigMC:** A model checker for bigraphs which includes a command line interface and visualisation [4].

**bigraphspace:** A Java library which provides a tuple-space-like API based on bigraphs [14].

**Big Red:** A graphical editor for bigraphs with easily extensible support for various file formats [3].

**BigWB:** A graphical workbench for bigraphs, aiming at providing a unifying GUI for the various bigraph tools (no website or papers at the time of writing).

**DBtk:** A tool for directed bigraphs, which provides calculation of IPOs, matching, and visualisation [1].

**SAT based algorithm:** Sevegnani et al. has presented a SAT based algorithm for matching in place graphs with sharing [21] and an implementation is in progress based on MiniSAT [11].

**SBAM:** A stochastic simulator for bigraphs, aimed at simulation of biological models [20].

### 4.1.2 Outline

In the remainder of this report, we assume a basic knowledge of bigraphs; we refer the uninitiated reader to Milner's book [19]. We shall use the bigraph notation from [13].

This report encourages a hands-on approach, and our focus is therefore on getting the tool installed and trying an example. The tool has built-in documentation, which we include (and slightly expand) for easy reference.

**Section 4.2, Installation**

Instructions for how to obtain, install and run the BPL Tool.

**Section 4.3, Example: Polyadic $\pi$ and Mobile Phones**

We demonstrate the features of the BPL Tool using Milner's polyadic $\pi$ calculus model of mobile phones [18].

**Section 4.4, Reference**

We present the BPL language (BPLL) for bigraphical reactive systems and the various functions that the BPL Tool provides.

**Section 4.5, Conclusions and Future Work**

We present our experiences with the BPL Tool and conclude on its strengths and weaknesses, and present our plans for future improvements.

## 4.2 Installation

The BPL Tool is distributed as source code as it relies on an SML compiler with an interactive mode to provide a command line interface. The source code can be obtained from the BPL Tool website [7].

We shall here distinguish between two types of installation: user and developer installations.

### 4.2.1 User installation

The BPL Tool requires the following software to be installed on your system:

**SML compiler** preferably SML of New Jersey but Moscow ML and MLton should work as well.

**GNU make**

**GNU sed**

The BPL Tool has been known to run on the following platforms: Linux (Ubuntu), OS X (ver. 10.4-10.6), and Windows XP (using Cygwin).

When you have installed the above and obtained a copy of the BPL Tool sources, you need to configure the tool to your setup. This is done by executing the following command in the $BPL/src directory:

```
./configure
```

To use the BPL Tool CLI, you need to use an SML compiler with an interactive mode (i.e. not MLton) – it works particularly well with SML of New Jersey, since SML/NJ allows the use of custom pretty-printers for values.

To start the CLI, execute the following command in the $BPL/src directory:

```
./bpltoolcli.sh
```

Once the BPL Tool has loaded you should be met with the prompt:

```
BPL (revision 3294) interactive prompt.  Type 'help[];' for help.
-
```

### 4.2.2 Developer installation

BPL Tool developers will – in addition to the basic installation – need the following software:

**GNU autoconf:** needed if you change configure.in.

**SML#:** BPL Tool uses some of the tools, SMLUnit and SMLDoc, from the SML# distribution, but not SML# itself.

To run the unit tests, execute the following command in the $BPL/src directory (or one of its sub-directories):

```
make test
```

If you have multiple SML compilers installed, you can switch between them by running the configure script with the MLC option set to one of the values `mlton`, `mosml`, or `smlnj`. E.g. to use the Moscow ML compiler, run

```
./configure MLC=mosml
```

## 4.3 Example: Polyadic $\pi$ and Mobile Phones

We model the polyadic $\pi$ calculus, running the mobile phone system introduced in Milner's $\pi$ book [18].

### 4.3.1 A mobile phone system

The mobile phone system we shall model is the following: there is a static network of transmitters which are all connected to a central control. Each mobile phone is located in a car and is connected to a single transmitter using a unique frequency. On some events, e.g. signal fading, the mobile phone may switch to another transmitter.

A simple example of such a system, call it $System_1$, is shown in Figure 4.1, where we also show how to define the system in the BPL Tool: the system consists of a car, one active and one idle transmitter, and a control centre. Note that the controls are atomic, since nodes with these

```
val (switch1,  talk1,  lose1,  gain1) =
  ( "switch1","talk1","lose1","gain1")
val (switch2,  talk2,  lose2,  gain2) =
  ( "switch2","talk2","lose2","gain2")

val Car      = atomic ("Car"     -: 2)
val Trans    = atomic ("Trans"   -: 4)
val Idtrans  = atomic ("Idtrans" -: 2)
val Control  = atomic ("Control" -: 8)

val System1 =
      Car[talk1,switch1]
  `|` Trans[talk1,switch1,gain1,lose1]
  `|` Idtrans[gain2,lose2]
  `|` Control[lose1,talk2,switch2,gain2,
              lose2,talk1,switch1,gain1]
```

Figure 4.1: Definition of the mobile phone system, $System_1$

controls should not contain other nodes.

The illustration of $System_1$ in Figure 4.1 is "hand-drawn" in TikZ, but we can also use the BPL Tool to render the system using either SVG or TikZ. For example, Figure 4.2 shows how to generate TikZ for $System_1$ and the resulting diagram. This uses the default configuration, but it is possible to obtain more fine grained control over the appearance of roots, nodes and sites. As is evident from the diagram in this example, it cannot replace hand-drawn figures. Nevertheless, it is our experience that such automatic visualisation is very helpful when working with bigraphs in the BPL Tool.

### 4.3.2 Polyadic $\pi$

Let us now examine how we can model a dynamic aspect of the mobile phone system, namely the hand-over protocol for when a mobile phone switches from one transmitter to another. We shall model this in the polyadic $\pi$ calculus which again can be modelled directly in the BPL Tool.

The polyadic $\pi$ calculus can be modeled by a family of reaction rules $\{\text{REACT}_i \mid i = 0, 1, \ldots\}$, one for each number of names that are to be communicated in a reaction [16]; $\text{REACT}_2$ is shown in Figure 4.3. The signature for the nodes modelling the polyadic $\pi$ calculus is constructed using `passive` controls as shown in Figure 4.4. For this system, we only need `Send` and `Get` nodes for $\text{REACT}_0$ and $\text{REACT}_2$. Note that all reaction rule nodes are passive, preventing reaction within a guarded expression.

In the $\pi$ calculus the nodes in $System_1$ are defined as recursive equations. In the BPL tool, they are defined by a rule that unfolds an atomic node into a bigraph corresponding to the defining $\pi$ calculus expression. The definitional equations and BPL definitions are shown in Figure 4.5.

The definitions allows the control centre to switch *Car* communication between the two transmitters (supposedly when the car gets closer to the idle than the active transmitter), and allows the car to communicate with the active transmitter.

Our BPL definition of the initial system in Figure 4.1, $System_1$, is the folded version; querying the tool reveals the four possible unfolding matches, illustrated in Figure 4.6. Here `mkrules` constructs the internal representation of a rule set, and `print_mv` prettyprints a lazy list of matches, produced by the `matches` function, cf. Sections 4.4.6 and 4.4.9.

We can unfold the four nodes into their defining $\pi$ calculus expressions by using the reaction tactic `TAC_unfold`, shown in Figure 4.7. The tactic is constructed using the `react_rule` tactic which simply applies a named reaction rule and the `++` tactic which runs its arguments sequentially, cf. Section 4.4.9. Applying this tactic using the function `run`, we get an unfolded version of the system.

Querying the BPL Tool for matches in the unfolded system reveals exactly the switch and talk actions, initiated by $\text{REACT}_2$ and $\text{REACT}_0$ rules, respectively, cf. Figure 4.8. Applying the $\pi$ calculus reaction rules for switching, using the `TAC_switch` tactic, we arrive at $System_2$, where *Car* communication has been switched to the other transmitter, as witnessed by the outer names to which *Car* ports link, as well as the order of names to which *Control* ports link.

## 4.4 Reference

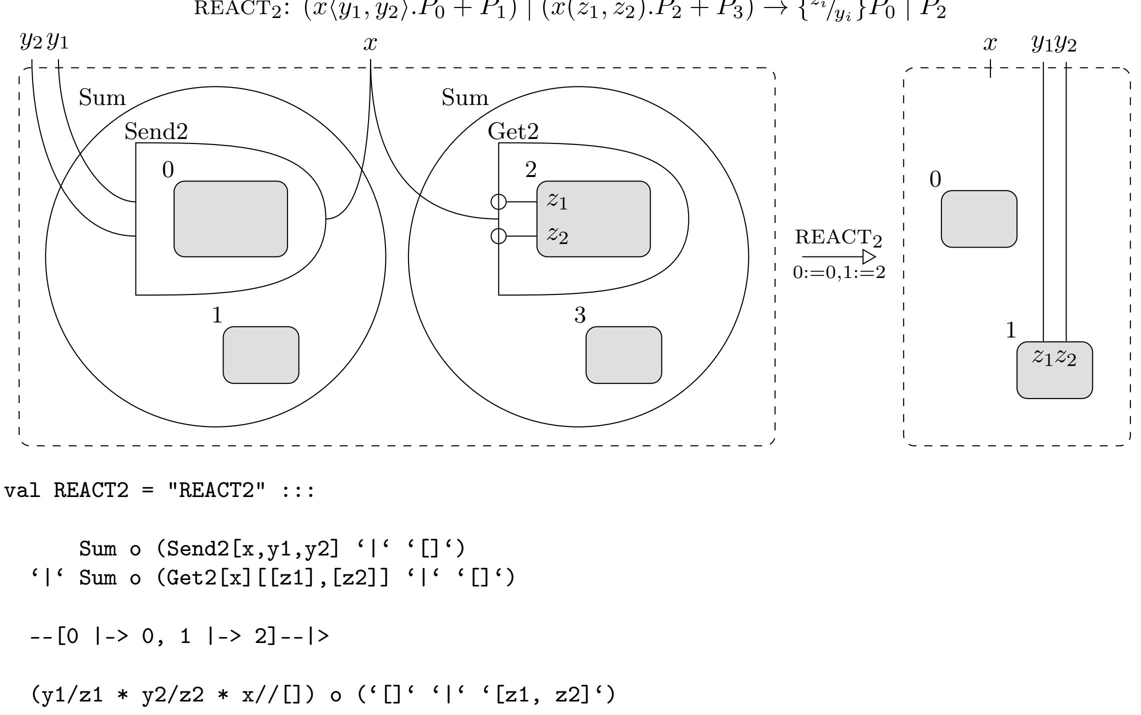
The language used in the BPL Tool is called BPLL, and it consists of a number of SML constructs which allows you to write BPLL directly in SML programs. This also means that your favorite interactive SML environment doubles as BPLL environment.

In this section we present the BPLL syntax for bigraphs and bigraphical reactive systems. Much of this information is also accessible through the help function in the BPL Tool CLI.

```
- print (tikz System1);
\tikzstyle nametext=[font=\footnotesize\itshape,inner sep=0pt]%
\tikzstyle root=[dashed,rounded corners]%
\tikzstyle binder=[draw,fill=white]%
\tikzstyle node=[draw]%
\tikzstyle nodetext=[font=\sffamily\bfseries,text=blue,inner sep=0pt]%
\tikzstyle site=[fill=gray!25,rounded corners]%
\tikzstyle sitetext=[font=\sffamily,inner sep=0pt]%
\tikzstyle link=[draw]%
\begin{tikzpicture}[x={(0.02cm,0cm)},y=-0.02cm,baseline=-1cm]
  \draw[style=root] (0,16) rectangle +(260,88);
  \draw[style=node] (29,64) ellipse (0.5cm and 0.4cm);
  \draw (4,100) node [style=nodetext,anchor=south west] {Car};
  \draw[style=node] (83,64) ellipse (0.5cm and 0.4cm);
  \draw (58,100) node [style=nodetext,anchor=south west] {Trans};
  \draw[style=node] (147,64) ellipse (0.7cm and 0.4cm);
  \draw (112,100) node [style=nodetext,anchor=south west] {Idtrans};
  \draw[style=node] (221,64) ellipse (0.7cm and 0.4cm);
  \draw (186,100) node [style=nodetext,anchor=south west] {Control};
  \draw (231,45) .. controls +(0,-24) and +(0,20) .. (109,13);
  \draw (72,46) .. controls +(0,-24) and +(0,20) .. (109,13);
  \draw (25,44) .. controls +(0,-24) and +(0,20) .. (109,13);
  \draw (109,10) node [style=nametext,anchor=south] {talk1};
  \draw (238,47) .. controls +(0,-24) and +(0,20) .. (158,13);
  \draw (79,44) .. controls +(0,-24) and +(0,20) .. (158,13);
  \draw (32,44) .. controls +(0,-24) and +(0,20) .. (158,13);
  \draw (158,10) node [style=nametext,anchor=south] {switch1};
  \draw (196,50) .. controls +(0,-24) and +(0,20) .. (207,13);
  \draw (93,46) .. controls +(0,-24) and +(0,20) .. (207,13);
  \draw (207,10) node [style=nametext,anchor=south] {lose1};
  \draw (245,49) .. controls +(0,-24) and +(0,20) .. (249,13);
  \draw (86,44) .. controls +(0,-24) and +(0,20) .. (249,13);
  \draw (249,10) node [style=nametext,anchor=south] {gain1};
  \draw (217,44) .. controls +(0,-24) and +(0,20) .. (291,13);
  \draw (143,44) .. controls +(0,-24) and +(0,20) .. (291,13);
  \draw (291,10) node [style=nametext,anchor=south] {gain2};
  \draw (224,44) .. controls +(0,-24) and +(0,20) .. (333,13);
  \draw (150,44) .. controls +(0,-24) and +(0,20) .. (333,13);
  \draw (333,10) node [style=nametext,anchor=south] {lose2};
  \draw (203,47) .. controls +(0,-24) and +(0,20) .. (375,13);
  \draw (375,10) node [style=nametext,anchor=south] {talk2};
  \draw (210,45) .. controls +(0,-24) and +(0,20) .. (424,13);
  \draw (424,10) node [style=nametext,anchor=south] {switch2};
\end{tikzpicture}
```



Figure 4.2: Generating TikZ




```
val REACT2 = "REACT2" :::

      Sum o (Send2[x,y1,y2] `|` `[]`)
  `|` Sum o (Get2[x][[z1],[z2]] `|` `[]`)

  --[0 |-> 0, 1 |-> 2]--|>

  (y1/z1 * y2/z2 * x//[]) o (`[]` `|` `[z1, z2]`)
```

Figure 4.3: $\pi$ calculus reaction rule shown as bigraphs and BPL values.

```
val Sum   = passive0 ("Sum")
val Send0 = passive  ("Send0" -: 0  +  1)
val Get0  = passive  ("Get0"  -: 0  +  1)
val Send2 = passive  ("Send2" -: 2  +  1)
val Get2  = passive  ("Get2"  =: 2 --> 1)
```

Figure 4.4: Signature for polyadic $\pi$ calculus.

<table>
<tr><th>Defining equation</th><th>BPL definition</th></tr>
<tr><td>$Car(talk, switch) \stackrel{\text{def}}{=}$<br>$\overline{talk}.\, Car\langle talk, switch\rangle$<br>$+\, switch(t, s).\, Car\langle t, s\rangle$</td><td><code>val DEF_Car = "DEF_Car" :::</code><br><code>  Car[talk,switch]</code><br><code>  ----|></code><br><code>  Sum o (    Send0[talk] o Car[talk,switch]</code><br><code>         ‘|‘ Get2[switch][[t],[s]] o Car[t,s])</code></td></tr>
<tr><td>$Trans(talk, switch, gain, lose) \stackrel{\text{def}}{=}$<br>$talk\, .\, Trans\langle talk, switch, gain, lose\rangle$<br>$+\, lose(t, s).\overline{switch}\langle t, s\rangle$<br>$.\, Idtrans\langle gain, lose\rangle$</td><td><code>val DEF_Trans = "DEF_Trans" :::</code><br><code>  Trans[talk,switch,gain,lose]</code><br><code>  ----|></code><br><code>  Sum o (    Get0[talk][]</code><br><code>              o Trans[talk,switch,gain,lose]</code><br><code>         ‘|‘ Get2[lose][[t],[s]]</code><br><code>              o Sum o Send2[switch,t,s]</code><br><code>                 o Idtrans[gain,lose])</code></td></tr>
<tr><td>$Idtrans(gain, lose) \stackrel{\text{def}}{=}$<br>$gain(t, s).\, Trans\langle t, s, gain, lose\rangle$</td><td><code>val DEF_Idtrans = "DEF_Idtrans" :::</code><br><code>  Idtrans[gain, lose]</code><br><code>  ----|></code><br><code>  Sum o Get2[gain][[t],[s]] o Trans[t,s,gain,lose]</code></td></tr>
<tr><td>$Control(lose_1, talk_2, switch_2, gain_2,$<br>$lose_2, talk_1, switch_1, gain_1) \stackrel{\text{def}}{=}$<br>$\overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle$<br>$.\, Control\langle lose_2, talk_1, switch_1, gain_1,$<br>$lose_1, talk_2, switch_2, gain_2\rangle$</td><td><code>val DEF_Control = "DEF_Control" :::</code><br><code>  Control[lose1,talk2,switch2,gain2,</code><br><code>          lose2,talk1,switch1,gain1]</code><br><code>  ----|></code><br><code>  Sum o Send2[lose1,talk2,switch2]</code><br><code>   o Sum o Send2[gain2,talk2,switch2]</code><br><code>    o Control[lose2,talk1,switch1,gain1,</code><br><code>              lose1,talk2,switch2,gain2]</code></td></tr>
</table>

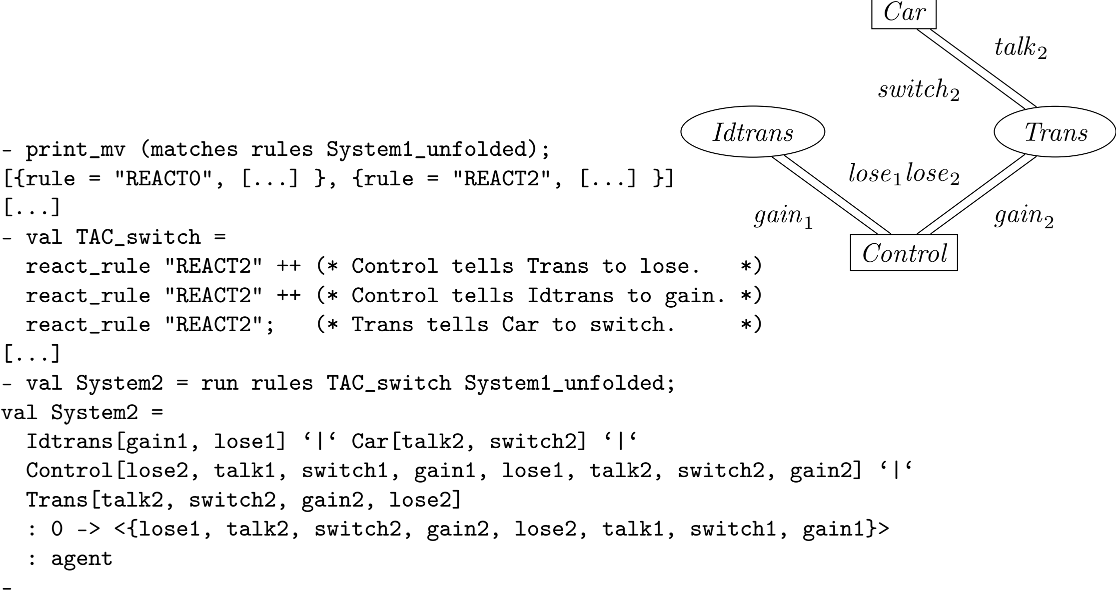
Figure 4.5: Definitions of Car, Trans, Idtrans and Control nodes.

```
- val rules = mkrules [REACT0, REACT2, DEF_Car, DEF_Trans, DEF_Idtrans, DEF_Control];
[...]
- print_mv (matches rules System1);
[{rule = "DEF_Car",
 context
  = (talk1/talk * switch1/switch) ||
    ‘[]‘ ‘|‘ Trans[talk1, switch1, gain1, lose1] ‘|‘
    Idtrans[gain2, lose2] ‘|‘
    Control[lose1, talk2, switch2, gain2, lose2, talk1, switch1, gain1],
 parameter = idx0},
{rule = "DEF_Control", [...] },
{rule = "DEF_Idtrans", [...] },
{rule = "DEF_Trans", [...] }]
```

Figure 4.6: Determining which rules match $System_1$.

```
- val TAC_unfold =
  react_rule "DEF_Car"     ++ react_rule "DEF_Trans"   ++
  react_rule "DEF_Idtrans" ++ react_rule "DEF_Control";
[...]
- val System1_unfolded = run rules TAC_unfold System1;
val System1_unfolded =
  Sum o
  (Send0[talk1] o Car[talk1, switch1] `|` Get2[switch1][[t], [s]] o Car[t, s]) `|`
  Sum o
  (Get0[talk1] o Trans[talk1, switch1, gain1, lose1] `|`
   Get2[lose1][[t], [s]] o Sum o Send2[switch1, t, s] o Idtrans[gain1, lose1]) `|`
  Sum o Get2[gain2][[t], [s]] o Trans[t, s, gain2, lose2] `|`
  Sum o
  Send2[lose1, talk2, switch2] o
  Sum o
  Send2[gain2, talk2, switch2] o
  Control[lose2, talk1, switch1, gain1, lose1, talk2, switch2, gain2]
  : 0 -> <{talk1, switch1, gain1, lose1, gain2, lose2, talk2, switch2}>
  : agent
```

Figure 4.7: Unfolding $System_1$, using the TAC_unfold tactic.




```
- print_mv (matches rules System1_unfolded);
[{rule = "REACT0", [...] }, {rule = "REACT2", [...] }]
[...]
- val TAC_switch =
  react_rule "REACT2" ++ (* Control tells Trans to lose.   *)
  react_rule "REACT2" ++ (* Control tells Idtrans to gain. *)
  react_rule "REACT2";   (* Trans tells Car to switch.     *)
[...]
- val System2 = run rules TAC_switch System1_unfolded;
val System2 =
  Idtrans[gain1, lose1] `|` Car[talk2, switch2] `|`
  Control[lose2, talk1, switch1, gain1, lose1, talk2, switch2, gain2] `|`
  Trans[talk2, switch2, gain2, lose2]
  : 0 -> <{lose1, talk2, switch2, gain2, lose2, talk1, switch1, gain1}>
  : agent
-
```

Figure 4.8: Checking possible matches, then switching to $System_2$, using the TAC_switch tactic.

### 4.4.1 Preliminaries

BPLL is a DSL embedded in Standard ML. This has the benefit of allowing easy extensions to the language and easy integration into SML programs. But it also imposes some restrictions on the syntax; for example, we cannot use `|` to denote the prime parallel product operator, so instead we use `` `|` ``, i.e. we wrap the operator in backquotes. In general, we have attempted to choose a syntax which visually closely resembles Milner's bigraph notation. While we believe we have been reasonably successful at this, we also appreciate that the notation is a bit heavy and we welcome any suggestions for improvements.

In order to keep bigraph terms as readable as possible, the BPL Tool assumes that bigraph names are bound to string variables of the same name. For example, the identity wiring $\mathtt{id}_{\{x\}}$ will be printed as `idw[x]`, thus assuming the preceding declaration of `x`: `val x = "x"`. The same goes for named ports (see below). This saves a lot of quotes when the BPL Tool prints bigraph terms.

### 4.4.2 Signatures

The definition of signatures in the BPL Tool is centered around controls: to define a control called $K$ which has *status* $s \in \{$`active`,`passive`,`atomic`$\}$, *global arity* $m$ and *local arity* $n$ one writes:

```
val K = s ("K" =:  m -> n)
```

For instance, in the example in Section 4.3 we needed a passive control with global arity 1 and local arity 2 called `Get2`, which was defined as follows:

```
val Get2 = passive ("Get2" =:  2 -> 1)
```

There is syntactic sugar for the common cases where the local arity is zero or both arities are zero, and for named ports:

**general case:**

```
val K = s ("K" =:  m -> n)
```

**local arity = 0:**

```
val K = s ("K" -:  n)
```

**global and local arity = 0:**

```
val K = s0 ("K")
```

**named ports, general case:**

```
val K = s ("K" ==:  [p'_1,...,p'_m] --> [p_1,...,p_n])
```

**named ports, local arity = 0:**

```
val K = s ("K" -:  [p_1,...,p_n])
```

where $K$ is the control name, $s \in \{$`active`,`passive`,`atomic`$\}$ is the status, $m$ is the global arity, $n$ is the local arity, and the $p_i$ and $p'_i$ are the names of global and local ports respectively.

It is not quite precise to say that the latter four cases are just syntactic sugar, as the way the control is used depends on how it was declared (i.e. their SML types are different) (cf. Ions below).

### 4.4.3 Types for bigraph terms

The BPL Tool uses the following types to distinguish bigraph terms on certain forms:

`bgterm`: a bigraph term that might not be well-formed, i.e. interfaces in compositions might not match;

`bgval`: a well-formed bigraph term with interfaces;

`'a bgbdnf`: a bigraph term which is on a binding discrete normal form indicated by the phantom type used for `'a`:

- `M`: molecule
- `S`: singular top-level node
- `G`: global discrete prime
- `N`: name-discrete prime
- `P`: discrete prime
- `D`: discrete bigraph
- `B`: bigraph
- `DR`: discrete, regular bigraph
- `BR`: regular bigraph

The normal forms are shown in Figure 4.9.

| | | |
|---|---|---|
| $M$ | $::= (\mathsf{id}_Z \otimes K_{\vec{y}(\vec{X})})N$ | *molecule* |
| $S$ | $::= \ulcorner\alpha\urcorner \quad \mid \quad M$ | *singular top-level node* |
| $G$ | $::= (\mathsf{id}_Y \otimes \mathit{merge}_n)(\bigotimes_i^n S_i)\pi$ | *global discrete prime* |
| $N$ | $::= (X)G$ | *name-discrete prime* |
| $P, Q$ | $::= (\mathsf{id}_Z \otimes \widehat{\sigma})N$ | *discrete prime* |
| $D$ | $::= \alpha \otimes (\bigotimes_i^n P_i)\pi$ | *discrete bigraph* |
| $B$ | $::= (\omega \otimes \mathsf{id}_{(\vec{X})})D$ | *bigraph* |
| $DR$ | $::= \alpha \otimes (\bigotimes_i^n P_i)$ | *discrete, regular bigraph* |
| $BR$ | $::= (\omega \otimes \mathsf{id}_{(\vec{X})})DR$ | *regular bigraph* |

Figure 4.9: Normal forms for binding bigraphs.

### 4.4.4 Bigraphs

Bigraphs are built from elementary bigraphs and operators:

| | | |
|---|---|---|
| Ions: | (`K,L,M` : `control` ; `x,y,p,p'` : `string`) | |
| `K[y,...]` | | Ion with control of global arity |
| `L[y,...][[x,...],...]` | | Ion with control of global/local arity |
| `K[p==y,...]` | | Ion with control of global arity with named ports |
| `L[p==y,...][p'==x,...]` | | Ion with control of global/local arity with named ports |
| Wirings: | (`x,y` : `string`) | |
| `y/x` | | Renaming link |
| `y//[x,...]` | | Substitution link |

| | |
|---|---|
| `y//[]` | Name introduction |
| `-/x` | Closure edge |
| `-//[x,...]` | Multiple closure edges |
| `idw[x,...]` | Identity wiring |
| | |
| Concretions: (`n >= 0` ; `x : string`) | |
| `‘[x,...]‘` | Concretion of names `x`,... |
| | |
| Merges: (`n >= 0`) | |
| `merge(n)` | Merge of inner width n |
| `<->` | Barren root (= `merge 0`) |
| | |
| Permutations: (`0 <=` i$_k$ `< m` ; `x : string`) | |
| `@[...,` i$_k$`, ...]` | Permutation mapping site $k$ to root i$_k$ |
| `@@[...,` i$_k$`&[x,...], ...]` | Permutation with local names |
| `idp(m)` | Identity permutation of width m |
| | |
| Operators: (`x : string` ; `A,B : bgval` ; `P : prime bgval`) | |
| `<[x,...]> P` | Abstract names x,... of a prime P |
| `A * B` | Tensor product |
| `A || B` | Parallel product |
| `A ‘|‘ B` | Prime product |
| `**[A,...]` | Tensor procuct of n factors |
| `|||[A,...]` | Parallel product of n factors |
| `‘|‘‘[A,...]` | Prime product of n factors |
| `A o B` | Composition |
| | |
| Precedence: (`x : string` ; `P : prime bgval`) | |
| `o` | Composition (strongest) |
| `*, ||, ‘|‘` | Product, left associative |
| `<[x,...]> P` | Abstraction (weakest) |

Bigraphs built using these combinators have the SML type `bgval`. Note that since the BPL Tool binds bigraph composition to ’`o`’, we rebind function composition to ’`oo`’.

### Syntactic Sugar

The BPL Tool allows some of the syntactic shorthands used in the bigraph literature – the shorthands for each relevant combinator are as follows:

**abstraction:** In an abstraction $(X)P$, one may abstract names that are not in the outer face of $P$ and we allow abstractions on name introductions. For example, `<[x]> y//[]` is allowed. The desugared form of such an abstraction is

$$(X)P \stackrel{\text{def}}{=} (X)(P \otimes Y \otimes \mathsf{id})$$

where $Y$ are the names of $X$ which are not in the outer face of $P$ and $\mathsf{id}$ is either $\mathsf{id}_1$ if $\mathsf{width}(P) = 0$ and $\mathsf{id}_0$ otherwise. Thus, `<[x]> y//[]` $\stackrel{\text{def}}{=}$ `<[x]>(y//[] * x//[] * idp(1))`.

**composition:** The BPLL composition operator is a generalization of Milner’s nesting operator: in a composition $A \circ B$ the outer names of $A$ and $B$ may be shared. For example, `K[x] o`

K[x] is allowed. The desugared form of such a composition is

$$A \circ B \stackrel{\text{def}}{=} (A \parallel \mathsf{id}_X) \circ B$$

where $X$ are the outer names of $B$. Thus, `K[x] o K[x]` $\stackrel{\text{def}}{=}$ `(K[x] || idw[x]) o K[x]`.

Also, it allows implicit abstraction of names in primes: in a composition $A \circ P$ where $P$ is prime, the local inner names of $A$ that are not local in the outer face of $P$ will be abstracted. The desugared form of such a composition is

$$A \circ P \stackrel{\text{def}}{=} A \circ ((X)P)$$

**ion:** The global ports of an ion $K_{\vec{y}}$ are allowed to use the same name, i.e. the names of $\vec{y}$ need not be distinct. For example, `K[x,x]` is allowed. The desugared form of such an ion is

$$K_{\vec{y}(\vec{X})} \stackrel{\text{def}}{=} (\omega \otimes \mathsf{id}_1) \circ K_{\vec{z}(\vec{X})}$$

where $\vec{z}$ is a vector of $n$ distinct names, $\mathsf{ar}(K) = m \to n$, and $\omega : \{\vec{z}\} \to \{\vec{y}\}$ is a substitution satisfying $\vec{y}_i = \omega(\vec{z}_i)$ $(i \in n)$. Thus, `K[x,x]` $\stackrel{\text{def}}{=}$ `(x//[x,y] * idp(1)) o K[x,y]`.

Note that the BPL Tool will do its best (subject to the configuration options discussed in Section 4.4.12) to use the syntactically sugared forms whenever possible.

Also, the BPL Tool internally works on bigraph terms in the normal forms shown in Figure 4.9. Such terms are not easily readable, in particular because '`|`' and `||` are treated as derived operators. The BPL Tool will try (again subject to configuration options) to simplify the terms and use '`|`' and `||` whenever possible.

### 4.4.5 Bigraph Operations

| | | | |
|---|---|---|---|
| `===` | : | `bgval * bgval -> bool` | Equality (infix) |
| `====` | : | `'a bgbdnf * 'a bgbdnf -> bool` | Equality (infix) |
| `norm_v` | : | `bgval -> B bgbdnf` | Normalise |
| `denorm_b` | : | `'a bgbdnf -> bgval` | Denormalise |
| `regl_v` | : | `bgval -> BR bgbdnf` | Regularise |
| `regl_b` | : | `B bgbdnf -> BR bgbdnf` | Regularise |
| `simpl_v` | : | `bgval -> bgval` | Attempt to simplify |
| `simpl_b` | : | `'a bgbdnf -> bgval` | Attempt to simplify |

### 4.4.6 Matching

Matching is computationally intensive, so the BPL Tool uses lazy lists to represent sets of matches.

`match_v` : `{agent:bgval, redex:bgval} -> match lazylist`
Match redex in agent, returning a lazy list of matches.

`match_b` : `{agent:B bgbdnf, redex:B bgbdnf} -> match lazylist`
Match redex in agent, returning lazy list of matches.

`print_mv` : `match lazylist -> unit`
Print lazy list of matches.

`print_mb` : `match lazylist -> unit`
Print lazy list of matches.

`print_mtv` : `match lazylist -> unit`
Print lazy list of matches with trees.

`print_mtb` : `match lazylist -> unit`
Print lazy list of matches with trees.

### 4.4.7 Lazy lists

The main functions for working with lazy lists are the following; see the online API for a complete list [7].

`lznull` : `'a lazylist -> bool`
Test whether the lazy list is empty.

`lzhd` : `'a lazylist -> 'a`
Return the first element of a lazy list.

`lztl` : `'a lazylist -> 'a lazylist`
Return the tail of a lazy list.

`lzunmk` : `'a lazylist -> 'a lazycell`
Return the head and tail of a lazy list, or Nil if it is empty.

`lzmap` : `('a -> 'b) -> 'a lazylist -> 'b lazylist`
Map a function on all elements of a lazy list.

### 4.4.8 Reaction rules

Reaction rules are constructed using the following combinators:

Instantiations: ($\mathtt{i}_k$,$\mathtt{j}_k$ : `int` ; $\mathtt{x}_k$,$\mathtt{y}_k$ : `string`)

`[..., i`$_k$ `|-> j`$_k$`, ...]`
Instantiation mapping reactum site $\mathtt{i}_k$ to redex site $\mathtt{j}_k$

`[..., i`$_k$`&[x`$_0$`, ..., x`$_{m-1}$`] |-> j`$_k$`&[y`$_0$`, ..., y`$_{m-1}$`], ...]`
Instantiation mapping local reactum name $\mathtt{x}_k$ to redex name $\mathtt{y}_k$

Rules: (`R`,`R'` : `rule` ; `rho` : `instantiation` ; `N` : `string`)

| | |
|---|---|
| `R --|> R'` | Rule with redex R, reactum R' and default instantiation |
| `R -rho-|> R'` | Rule with redex R, reactum R' and instantiation rho |
| `N ::: R --|> R'` | Named rule |

Operators on rules: (`R` : `rule`)

| | |
|---|---|
| `redex R` | Extract the redex of a rule |
| `reactum R` | Extract the reactum of a rule |
| `inst R` | Extract the instantiation of a rule |

For convenience, instantiations in rules need not be fully specified; if an instantiation $\rho : J \to I$, where $J$ and $I$ are the reactum and redex innerfaces respectively, is partially specified, the BPL Tool will automatically add missing mappings as follows:

1. if a site of $J$ is not mentioned, it is assumed to map to the same site at $I$, inferring the name map as in (2);
2. if the name lists of a map are empty, the local renaming will be inferred as follows:
   (a) if the relevant sites of $I$ and $J$ have the same local names, an identity renaming will be used;
   (b) otherwise, if there is only one local name at both sites, say $x$ at $I_i$ and $y$ at $J_j$, the local renaming $(y)/(x)$ will be used.

An exception will be raised if this procedure not yield an instantiation.

### 4.4.9 Simulation

Tactics: (`i` : `int` ; `N` : `string` ; $\mathtt{t}_i$ : `tactic`)

| | |
|---|---|
| `react_rule N` | Apply rule `N` |
| `react_rule_any` | Apply any rule |
| `roundrobin` | Apply rules roundrobin until none match |
| `t`$_1$ `++ t`$_2$ | Use $\mathtt{t}_1$, then $\mathtt{t}_2$ |
| `TRY t`$_1$ `ORTHEN t`$_2$ | If $\mathtt{t}_1$ fails, use $\mathtt{t}_2$ on its result |
| `IF t`$_1$ `THEN t`$_2$ `ELSE t`$_3$ | If $\mathtt{t}_1$ finishes, use $\mathtt{t}_2$, else $\mathtt{t}_3$ on its result |
| `REPEAT t` | Repeat `t` until it fails |
| `i TIMES_DO t` | Use `t i` times |
| `finish` | Finish tactic |
| `fail` | Fail tactic |

Reaction operations: (`v` : `bgval`; `m` : `match`; $\mathtt{r}_i$ : `rule` ; $\mathtt{N}_i$ : `string` ; `rs` : `rules`)

| | |
|---|---|
| `react m` | Perform a single reaction step |
| `mkrules [r`$_0$`, ..., r`$_n$`]` | Construct a rule map |
| `mknamedrules [..., (N`$_i$`, r`$_i$`), ...]` | Construct a rule map with explicit names |
| `matches rs v` | Return lazy list of all matches of all rules |
| `run rs t v` | Perform agent reactions using a tactic |
| `steps rs t v` | Return agent for each step using a tactic |
| `stepz rs t v` | Return lazily agent for each step using a tactic |

### 4.4.10 Pretty printing

Bigraphs:

| | | | |
|---|---|---|---|
| `str_v` | : | `bgval -> string` | Return as a string |
| `str_b` | : | `'a bgbdnf -> string` | Return as a string |
| `print_v` | : | `bgval -> unit` | Print to stdOut |
| `print_b` | : | `'a bgbdnf -> unit` | Print to stdOut |

Matches:

| | | | |
|---|---|---|---|
| `print_mv` | : | `match lazylist -> unit` | Print lazy list of matches |
| `print_mb` | : | `match lazylist -> unit` | Print lazy list of matches |
| `print_mtv` | : | `match lazylist -> unit` | Print lazy list of matches with trees |
| `print_mtb` | : | `match lazylist -> unit` | Print lazy list of matches with trees |

Rules:

| | | | |
|---|---|---|---|
| `str_r` | : | `rule -> string` | Return rule as a string |
| `print_r` | : | `rule -> unit` | Print rule |

### 4.4.11 Visualization

Configuration:

`makecfg` `(string * BG.PPSVG.path -> configinfo) -> config`
Construct a config

`unmkcfg` `config -> string * BG.PPSVG.path -> configinfo`
Deconstruct a config

`defaultcfg` `config`
Default config

Scalable Vector Graphics (SVG):

`svg_v` `config option -> bgval -> string`
Return as SVG fragment string

`svg_b` `config option -> B bgbdnf -> string`
Return as SVG fragment string

`svg` `bgval -> string`
Return as SVG fragment string

`svgdoc_v` `config option -> bgval -> string`
Return as SVG document string

`svgdoc_b` `config option -> B bgbdnf -> string`
Return as SVG document string

`svgdoc` `bgval -> string`
Return as SVG document string

`outputsvgdoc_v` `string -> config option -> bgval -> unit`
Output as SVG document to file

| | |
|---|---|
| `outputsvgdoc_b` | `string -> config option -> B bgbdnf -> unit`<br>Output as SVG document to file |
| `outputsvgdoc` | `string -> bgval -> unit`<br>Output as SVG document to file |

TikZ:

| | |
|---|---|
| `tikz_v` | `real option -> config option-> bgval -> string`<br>Return as TikZ string |
| `tikz_b` | `real option -> config option -> B bgbdnf -> string`<br>Return as TikZ string |
| `tikz` | `bgval -> string`<br>Return as TikZ string |
| `outputtikz_v` | `string -> real option-> config option -> bgval -> unit`<br>Output as TikZ to file |
| `outputtikz_b` | `string -> real option-> config option -> B bgbdnf -> unit`<br>Output as TikZ to file |
| `outputtikz` | `string -> bgval -> unit`<br>Output as TikZ to file |

### 4.4.12 Controlling tool behaviour

The behaviour of the BPL Tool can be modified by changing a number of configuration *flags*. Flags are accessed by two families of functions:

| | |
|---|---|
| `Flags.get`*Type*`Flag "`*name*`"` | Get the value of the named flag of the given type |
| `Flags.set`*Type*`Flag "`*name*`"` *value* | Set the value of the named flag of the given type |

The help function for flags displays and explains all the available flags as well as their current and default values:

```
help["flags"];
```

**Matching:**

| **name** | **type** | **description** |
|---|---|---|
| `/kernel/match/match/nodups` | `bool` | Remove duplicate matches. |

**Miscellaneous:**

| **name** | **type** | **description** |
|---|---|---|
| `/debug/level` | `int` | Level of debugging information (0 = no info, >0 = info). |
| `/dump/prefix` | `string` | Filename prefix for pretty print dumps to a file. |
| `/misc/timings` | `bool` | Enable timings. |

**Pretty printing:**

| name | type | description |
|---|---|---|
| `/misc/indent` | `int` | Set extra indentation at each level when prettyprinting to N. |
| `/misc/linewidth` | `int` | Set line width to W characters. |
| `/kernel/ast/bgterm/pp0abs` | `bool` | Explicitly display empty-set abstractions (ignored if ppabs is false). |
| `/kernel/ast/bgterm/ppabs` | `bool` | Explicitly display abstractions (abstractions on roots are always displayed). |
| `/kernel/ast/bgterm/ppids` | `bool` | Explicitly display identities in tensor and parallel products. |
| `/kernel/ast/bgterm/ppmeraspri` | `bool` | Replace merge with prime product (best effort). |
| `/kernel/ast/bgterm/pptenaspar` | `bool` | Replace tensor product with parallel product. |
| `/kernel/ast/bgval/pp-merge2prime` | `bool` | Substitute $\mid$ for $\parallel$ by removal of merges before prettyprinting. |
| `/kernel/ast/bgval/pp-simplify` | `bool` | Simplify BgVal terms before prettyprinting. |
| `/kernel/ast/bgval/pp-tensor2parallel` | `bool` | Substitute $\parallel$ for * by removal of y//X's before prettyprinting. |
| `/kernel/bg/name/strip` | `bool` | Strip trailing _xx off input names (xx are hex digits). |
| `/kernel/match/rule/ppsimplereactum` | `bool` | Simplify reactum when displaying rules. |
| `/kernel/match/rule/ppsimpleredex` | `bool` | Simplify redex when displaying rules. |

For convenience, one can switch the use of syntactic shorthands on and off with a single command:

use_shorthands on/off

This will modify the following flags appropriately:

```
/kernel/ast/bgterm/ppids
/kernel/ast/bgterm/ppabs
/kernel/ast/bgterm/pp0abs
/kernel/ast/bgterm/pptenaspar
/kernel/ast/bgterm/ppmeraspri
```

### 4.4.13 Exceptions

Exceptions can be explained by the BPL Tool using the following command:

| | | |
|---|---|---|
| `explain` | `exn -> 'a` | Explain exception in detail and raise it again |

If the debug level is greater than 0, and the SML interpreter supports it, the exception history will also be printed.

## 4.5 Conclusions and Future Work

We have introduced the BPL Tool, a first implementation of bigraphical reactive systems with binding, and have demonstrated its use by modeling a simple mobile phone system.

Our research group has used the BPL Tool to successfully model a number of sytems (cf. Section 4.1). Our experience is, that the BPL Tool is that it is quite useful for modeling as it validates well-formedness of terms and rules, and its visualization capabilities, in particular through the web interface, provides a good overview of reaction rules.

However, there is also room for improvement:

- The tool would benefit from a more complete graphical user interface than what the web interface provides. One approach would be to extend Big Red [3] as follows:
  - add support for binding,
  - add facilities for modeling reaction rules, and
  - add simulation facilities, by using the BPL Tool as a simulation backend.
- The BPLL syntax is a bit heavy due to the fact that it is embedded in Standard ML. By building a dedicated command line interface one would be free to choose a simpler syntax. The BPL Tool code base already contains a parser for an older version of BPLL, so the main task is to implement an interactive prompt. The compromise would of course be that end-users will have a harder time extending the tool.
- The implementation of matching is not very fast, due to the fact that it is derived directly from the inductive characterization of matching which is based on the binding discrete normal form. The main issues are the following:
  - Structural congruence is currently handled naïvely: when matching children of a node, one need to find partitions and permutations and the BPL Tool simply generates them all.
  - Matching currently follows the place graph structure, and the link graph is only matched at the root and leaves of the matching inference tree. By interleaving the matching of the two graphs in a more fine-grained manner, one could probably prune the search space significantly; this would perhaps be easier if one based matching on a *connected normal form* where edges are as close to their constituent points as possible instead of being at the outermost level.
  - Only one redex is matched at a time, as this is the algorithm that naturally falls out of the inductive characterization of matching. By matching all redexes simultaneusly, only one traversal of the agent term would be necessary.

  However, while we believe the suggested improvements are significant, we believe that more efficient matching will be achieved by using SAT-solvers, which is currently being investigated by Sevegnani et al. [21], or by the graph embedding based approach of Højsgaard et al. [20]. Note that matching is NP-complete [20] and thus no efficient algorithm exists unless P = NP.
- From a modeling perspective, is would be convenient if the BPL Tool was extended with support for sortings of some kind, such that modellers could specify the structure of well-formed bigraphs and then have the BPL Tool verify well-formedness of agents and rules and that the latter preserves well-formedness.
- Similarly, built-in support for datatypes and manipulation of data would make it easier to express models containing computations. We suggest that such that an extension should be founded on a solid formal foundation, such as the calculational bigraphical reactive systems of Debois [10].

## 4.6 Bibliography


[1] Giorgio Bacci, Davide Grohmann, and Marino Miculan. DBtk: A toolkit for directed bigraphs. In *CALCO*, pages 413–422, 2009.

[2] Jørgen Eske Runge Bentzen. Master's thesis, IT University of Copenhagen, 2007.

[3] Big Red. `http://www.itu.dk/research/pls/wiki/index.php/Big_Red`, 2010.

[4] BigMC – Bigraphical Model Checker. `http://bigraph.org/bigmc/`.

[5] Lars Birkedal, Troels Christoffer Damgaard, Arne John Glenstrup, and Robin Milner. Matching of bigraphs. *Electronic Notes in Theoretical Computer Science*, 175(4):3–19, 2007.

[6] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Troels Hildebrandt, and Henning Niss. Bigraphical models of context-aware systems. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structure*, volume 3921 of *LNCS*, pages 187–201. Springer-Verlag, March 2006.

[7] BPL Tool. `http://www.itu.dk/research/pls/wiki/index.php/BPL_Tool`.

[8] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing WS-BPEL and higher order mobile embedded business processes in the bigraphical programming languages (BPL) tool. Technical Report TR-2008-103, IT University of Copenhagen, 2008.

[9] Troels Christoffer Damgaard and Lars Birkedal. Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1–2):58–77, 2006.

[10] Søren Debois. Computation in the informatic jungle. Draft, 2011.

[11] Niklas Eén and Niklas Sörensson. MiniSAT. `http://minisat.se`.

[12] Ebbe Elsborg. *Bigraphs: Modelling, Simulation, and Type Systems.* PhD thesis, IT University of Copenhagen, 2009.

[13] Arne John Glenstrup, Troels Christoffer Damgaard, Lars Birkedal, and Espen Højsgaard. An implementation of bigraph matching. Technical Report TR-2010-135, IT University of Copenhagen, December 2010.

[14] Chris Greenhalgh. bigraphspace. `http://bigraphspace.svn.sourceforge.net/`, 2009.

[15] Ole Høgh Jensen. Mobile processes in bigraphs. Available at `http://www.cl.cam.ac.uk/~rm135/Jensen-monograph.html`, 2006.

[16] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge, February 2004.

[17] James Judi Leifer and Robin Milner. Transition systems, link graphs and Petri nets. Technical Report UCAM-CL-TR-598, University of Cambridge, August 2004.

[18] Robin Milner. *Communicating and Mobile Systems: The $\pi$-Calculus.* Cambridge University Press, 1999.

[19] Robin Milner. *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.

[20] Stochastic Bigraphical Abstract Machine (SBAM). `http://www.itu.dk/research/pls/wiki/index.php/Stochastic_Bigraphical_Abstract_Machine_%28SBAM%29`.

[21] M. Sevegnani, C. Unsworth, and M. Calder. A SAT based algorithm for the matching problem in bigraphs with sharing. Technical Report TR-2010-311, University of Glasgow, Department of Computing Science, 2010.

# Part III

# Bigraphical Semantics for Business Processes

# Chapter 5

# Formalizing WS-BPEL and Higher Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool

**Mikkel Bundgaard, Arne J. Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss**

**Abstract**

Bigraphical Reactive Systems (BRSs) have been proposed as a formal meta-model for global ubiquitous computing that encompasses process calculi for mobility, notably the $\pi$-calculus and the Mobile Ambients calculus, as well as graphical models for concurrency such as Petri Nets. In this paper we demonstrate that BRSs also allow natural formalizations of programming languages used in practice. We do so by providing a direct and extensible formalization of a subset of WS-BPEL as a binding bigraphical reactive system using the BPL Tool developed in the Bigraphical Programming Languages (BPL) project. The tool allows for compositional definition, visualization and simulation of the execution of bigraphical reactive systems. The formalization exploits the close correspondence between bigraphs and XML to provide a formalized run-time format very close to standard WS-BPEL syntax.

The formalization is the starting point of an endeavor to provide a completely formalized and extensible business process engine within the Computer Supported Mobile Adaptive Business Processes (CosmoBiz) research project at the IT University of Copenhagen. Building upon the formalization of WS-BPEL we propose and formalize HomeBPEL, a higher-order WS-BPEL-like business process execution language where processes are *first-class values* that can be stored in variables, passed as messages, and activated as embedded *sub-instances*. A sub-instance is similar to a WS-BPEL scope, except that it can be dynamically *frozen* and stored as a process in a variable, and then subsequently be *thawed* when reactivated as a sub-instance. We motivate HomeBPEL by an example of pervasive health care where treatment guidelines are dynamically deployed as sub processes that may be delegated dynamically to other workflow engines and in particular stay available for disconnected operation on mobile devices.

**Preface** This chapter consists of the technical report

M. Bundgaard, A. J. Glenstrup, T. Hildebrandt, E. Højsgaard, and H. Niss. *Formalizing WS-BPEL and Higher-Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool.* Technical Report TR-2008-103, IT University of Copenhagen, May 2008.

## 5.1 Introduction

Services implemented and orchestrated by processes written in languages such as WS-BPEL are being put forward as a means to achieve loosely coupled and highly flexible computer supported business and work processes. In the current architectures, services are deployed and managed on web servers by meta-level tools and cannot be replaced or moved during the life-time of a session with an instance of the service. In the present paper we propose and formalize a higher-order WS-BPEL-like language, called HomeBPEL, where processes are values that can be stored in variables and dynamically instantiated as embedded sub-instances. A sub-instance is similar to a WS-BPEL scope, except that it can be dynamically frozen during a session and stored as a process in a variable. When frozen in a variable, the process instance can be sent to remote services as any other content of variables and dynamically re-instantiated as a local sub-instance continuing its execution.

We envisage a use of HomeBPEL where the necessary services or even active instances can be dynamically moved to a local process engine running on a mobile device and thereby allow for disconnected operation. We exemplify this use by an example of pervasive health care, where treatment workflows are moved between and executed locally on mobile devices belonging to either the doctor or the patient, depending on whether the guideline requires actions by the doctor or it prescribes actions carried out as self-treatment by the patient.

As a first step towards the formalization of HomeBPEL we provide a formalization of a non-trivial subset of the Web Service Business Process Execution Language, WS-BPEL [35], which is being promoted by major industrial players including IBM, SAP, BEA, Oracle, and Microsoft as the future standard for orchestrating web-services as business processes. The formalization exploits the close correspondence between bigraphs and XML to provide a small step rewrite semantics of the behavior of WS-BPEL, and the formalization uses a representation of the state of active process instances which is very close to the XML syntax of WS-BPEL processes. Building upon this formalization we provide a bigraphical formalization of a WS-BPEL-like business process language supporting higher-order primitives.

The investigation is part of the Computer Supported Mobile Adaptive Business Processes (CosmoBiz) project [24], which aims to provide a fully formalized runtime engine for a business process language extended to allow for mobile and adaptive processes. Our primary goals of the formalization is 1) to be able to guarantee that the implemented engine actually conforms to the semantics and 2) to form a basis for the development of type systems that can be used to statically guarantee safe and reliable behavior. To achieve the first goal a main concern is to limit the gap between the source language, its formalization, and the implementation. A key element to achieve the second goal is to strive for a *compositional* formalization supporting subsequent formalization of type rules for the individual parts. We want to stress that it is *not* a main concern at this point to provide techniques or principles for verification of processes, which has been the main concern of most WS-BPEL formalizations so far. However, we do hope that future reasoning techniques developed for bigraphs can be employed also to support formal verification.

The work on HomeBPEL is inspired and guided by our previous work on the Homer process calculus of Higher-order mobile embedded resources [8, 19, 21], and in particular its formalization as a bigraphical reactive system [7]. Not surprisingly, the new features add to the complexity of the language and its formalization. Yet, the formal approach ensures that they are completely unambiguously specified. Also, the close relationship to semantics of process calculi such as Homer and the Mobile Ambients gives a very succinct formalization of sub-process mobility. Indeed, the

serialized representation of a mobile process is just a process description. In particular, this means that a future implementation could use the standard XML format for serialized process instances.

The theory of *Bigraphical reactive systems* [26] provides a framework in which process models for concurrent and ubiquitous computing can be uniformly defined and formally analyzed. In particular, the $\pi$-calculus [33], Mobile Ambients calculus [10] and (1-safe) Petri Nets [29] have been represented as instances of bigraphical reactive systems [26]. Bigraphical reactive systems can be seen as a specialized kind of graph rewriting systems, in which processes are represented as two graphs (hence the name *bi*graphs): The place graph and the link graph respectively. The *place graph* is a collection of node labeled trees generalizing the nesting of process constructors in process calculi. The *link graph* is a hyper graph specifying links between ports associated to the nodes of the place graph and a set of external names, generalizing the link structure characteristic of the $\pi$-calculus. The dynamics is specified by a set of parametric reaction rules, generalizing the rule formats used in e.g. the $\pi$-calculus and the calculus of Mobile Ambients. The general theory of bigraphical reactive systems provides a general notion of contexts and composition. This allows for *compositional* description of processes as characteristic to process calculi. Together with a notion of "minimal" contexts, it forms the basis for an *automatic* derivation of a labeled transition bisimulation congruence from the reaction rules supporting compositional reasoning about the behavior of systems.

There are several reasons for why it is interesting to apply bigraphs to formalize an evolving standard such as WS-BPEL. Firstly, we thereby demonstrate that bigraphical reactive systems can not only be used as a meta-format for process calculi, but also be used to formalize programming languages used in practice. Secondly, the model of bigraphical reactive systems is *extensible*: An instance of a bigraphical reactive system is defined by its signature (the possible labels and ports of nodes) and its reaction rules, which can be chosen to fit a particular language and its semantics. By extending the signature and the set of reaction rules, a bigraphical reactive system can be adapted according to e.g. changes in the language specification or incremental extensions of the language. We exploit the extensibility of bigraphical reactive systems to extend the language and formalization of WS-BPEL with primitives for mobile, embedded sub-processes. Furthermore, the place and link graph of bigraphs correspond closely to respectively the nested element structure and sharing of attribute values in the XML data model. Since WS-BPEL is equipped with an XML syntax, we are able to provide a small step rewrite semantics in the model of bigraphical reactive systems using a representation of the current state of processes which is very close to the WS-BPEL syntax. Indeed, the close correspondence between bigraphs and XML was explored in our previous work on formalizing WS-BPEL as bigraphs [22, 23], on which the present work builds. However, the formalization in [22, 23] was obtained at the cost of introducing so-called higher-order reaction rules, for which the relationship to the existing notions of bigraphs and theory of behavioral congruences remain to been developed. In addition to covering a larger subset of WS-BPEL, the present formalization stays within the standard format for binding bigraphs described in [26]. Thus, the general theory, techniques and tools developed for binding bigraphs remain applicable to our formalization. In particular, we describe how the formalization can be explored within the BPL Tool [3] developed in the Bigraphical Programming Languages (BPL) project at the IT University of Copenhagen. The tool allows compositional definition of bigraphical reactive systems within Standard ML. It is also equipped with a web interface supporting visualization and interactive simulation of the execution of binding bigraphical reactive systems based on the formal inference of rule matching described in [1, 2].

The present paper combines the work presented in the two papers: [5] and [6].

**Related work.** WS-BPEL has been the target for several formalizations [40] accompanying the official informal specification [35]. Generally, any formalization requires a compilation of a BPEL process to a representation in the formal model. Clearly, the usefulness of a formalization depends on the availability of tools and reasoning techniques for the formal model, but also on how easy it

is to relate the formal representation to the original BPEL process description.

Many of the prior formalizations have been based on versions of Petri Nets [32, 38], following the tradition of formal workflow models. Other authors have been promoting the use of process calculi, notably the $\pi$-calculus [36, 39]. This diversion can be partly explained by the fact that WS-BPEL is a convergence and development of two radically different approaches to web service orchestration proposed back in 2001: The IBM Web Services Flow Language (WSFL) and the Microsoft XLANG specification. While WSFL was based on flow graphs which are characteristic to the Petri Net model and most workflow languages, XLANG was based on the notion of message exchange behavior which is characteristic to the $\pi$-calculus. A third line of formalizations are based on abstract state machines (ASMs) [13–16]. These seek to represent the informal specification *as is*, i.e. they aim at using the same terminology and level of abstraction in their formalizations, thereby hoping to minimize the gap to the informal specification. This goal is shared by our approach, though our method focuses on keeping the formalization close to the BPEL language itself and not its informal description.

In this paper we focus on the XLANG subset of WS-BPEL, in particular the control flow, scope structure, message passing and dynamic manipulation of Partner Links (akin to name passing in the $\pi$-calculus). However, since bigraphical reactive systems have been shown to faithfully represent both the $\pi$-calculus and Petri Nets, we believe the model is a good candidate for providing at the same time a faithful representation of both the WSFL and the XLANG features of WS-BPEL. Already for the present subset, we crucially exploit the nesting structure of bigraphs to give a very succinct semantics of "abnormal" termination caused by the WS-BPEL `exit` activity, which is not as easy to formalize in the $\pi$-calculus.

Our formalization of the core WS-BPEL subset relates to the WS-BPEL process calculus given in [28]. An advantage to our approach is that we can reuse the general theory developed for bigraphical reactive systems, instead of redeveloping an entire theory of a new process calculus. As in [28], we hope to be able to equip our formalization with WSDL-like (or even richer) type systems.

*Sub-processes* have been proposed by IBM and SAP in [27] as an extension to WS-BPEL (called BPEL-SPE) to allow for modularization and reuse of process fragments to ease the burden of designing large business processes. As argued in [27] one could simulate some of the behavior of sub-processes by invoking another process instead of invoking a sub-process. However, this makes it impossible to establish any coupling between the life-cycles of the two process instance, e.g. enforcing that a sub-process exits if the super process exits prematurely. The sub-processes we propose in this paper extend the proposal in [27] in several aspects. First and foremost, BPEL-SPE requires that the *sole* interaction of a sub-process is an initial receive activity, and a last reply activity, basically making the sub-process act as a method or function call. We allow that the sub-process can communicate unrestrictedly with the parent process (and vice versa) using invoke-receive. Furthermore, we add facilities for “freezing” and “thawing” sub-processes as well as (sub-)process mobility.

Higher order workflow models applied to health care processes have been considered in the context of Higher-Order (Petri) Nets [25], allowing sub-processes (nets) as values (tokens), which may be dynamically composed. The approach in [25] differs from ours in several ways: Firstly, the approach in [25] is based on Petri Nets as opposed to process calculi, and has no direct relationship to WS-BPEL nor service orchestration. Another central difference is that we execute sub-processes as sub-threads, whereas in [25] a sub-process is executed step-by-step by the super process — and sub-processes can not contain sub-processes themselves. Finally, the model in [25] allows for dynamic modification and composition of sub-processes, which is not yet supported in our setting.

As described above, our proposal of higher order mobile sub-processes relates to our work on the higher-order process calculus Homer. The Homer calculus is related to the process calculus of Mobile Ambients [10] and the Seal calculus [11]. Indeed, HomeBPEL shares with Seal the

combination of name (link) and process passing. We leave for future work to explore the relationship between HomeBPEL and these process calculi. Again, we hope that the many type systems proposed for process calculi for mobility can guide us to equip our formalization with useful type systems for controlling the mobility.

In [20] a notion of *mobile* business process is defined by processes in which a) the place of execution of an activity can change, or can be different for different instances, b) the change is caused by external factors, and c) cooperation with external resources is needed. The mobile sub-instances we propose meet all these criteria.

**Structure of the paper.** In Sec. 5.2 we introduce the meta-model of binding bigraphs, and in Sec. 5.3 we utilize binding bigraphs to give a formal semantics of a subset of WS-BPEL and visualize it using the BPL Tool. In Sec. 5.4 we motivate the need for higher-order constructs with an example of computer supported pervasive health care, and we present the resulting language HomeBPEL — a WS-BPEL-like language where processes are values that can be stored in variables and dynamically instantiated as embedded sub-instances. In Sec. 5.5 we formalize HomeBPEL using the BPL Tool. We conclude and propose directions for future work in Sec. 5.6.

## 5.2 Binding Bigraphs and BPL Tool

In this section we briefly review the binding bigraphs of Milner and Jensen [26] and introduce the syntactical representation of binding bigraphs as implemented in the BPL Tool. For a complete introduction to bigraphs we refer to [26].

### 5.2.1 Binding Bigraphs

A binding bigraph is a pair of graphs: a *place graph* and a *link graph*. The place graph is an $n$-tuple of finite, unordered trees. Except for roots, every node is labelled by a *control* and has two finite ordered sets of respectively *free* and *binding ports*. The link graph is essentially a hypergraph connecting every free port of the nodes in the place graph to either a closed link, a binding port, or a name in a finite set $X$ of names. Jointly with a collection of pairwise disjoint sets $X_i \subseteq X$ of local names, one for each root in the bigraph, the set $X$ defines the (outer) *interface* of the link graph. The so-called *scope condition* enforces that any binding port and any name in a set $X_i$ is only connected to ports nested strictly inside the node of the binding port and root $i$ respectively.

What we just described above is known as *ground* binding bigraphs. Intuitively, one may think of a ground binding bigraph as an ordered tuple of terms of a process calculus up to structural congruence: Sibling nodes in the place graph represent processes combined by an associative and commutative parallel operator. Each node is a prefix, and each control denotes a distinct prefix operation (e.g. send or receive in the $\pi$-calculus) with free and binding ports representing names and name binders of the particular operation (e.g. for the $\pi$-calculus, any node labelled by a send control would have 2 free ports, while nodes labelled by a receive control would have one free and one binding port). The link graph then maps each name in a prefix to either a local name (closed link), a binder (i.e. a binding port) or a name in the interface.

A ground bigraph with a single root is also similar to the data model for XML, with controls playing the role of the names of XML elements, ports playing the role of attributes and the linking of ports playing the role of attribute values. As we will see below, we exploit this similarity to give a bigraphical semantics to HomeBPEL resembling closely the XML syntax.

A central ingredient of the theory of bigraphs is that bigraphs in general are (multi-hole) *contexts* that can be composed: The place graph has a finite ordered set of *holes* (referred to as *sites* in the usual bigraph terminology), each associated as a child of a node. The link graph has a set of *local names* $Y_i$ for each hole. As for the outer interface, the sets $Y_i$ are pairwise disjoint and contained in a finite set of names $Y$ which jointly with the sets $Y_i$ forms the *inner* interface.

As the free ports, the names in $Y$ are connected to either a closed link, a binding port or a name in the outer interface.

Outer (resp. inner) interfaces of binding bigraphs are thus triples $\langle n, \vec{X}, X\rangle$, where the *width* $n$ is a finite ordinal representing the number of roots (resp. sites), $X$ is a finite set of names, and $\vec{X}$ is an $n$-tuple of pairwise disjoint subsets of $X$ which declares some of the names in $X$ as *local* to specific roots (resp. sites). If $x \notin \vec{X}$ then $x$ is said to be *global*, else it is *local*; if an interface $I$ has no global names $x$, it is a *local* interface. We write $G : I \to J$ for the bigraph $G$ with inner interface $I$ and outer interface $J$. The composition $H \circ G : I \to J$ of bigraphs $G : I \to I'$ and $H : I' \to J$ with compatible interfaces is obtained by making the children of the $i$th root of G children of the (parent) node of the $i$th site of $H$, discarding the roots of $G$ and sites of $H$, and by coalescing links as prescribed by the correspondence of $H$'s inner and $G$'s outer names.

A binding bigraphical reactive system is defined with respect to a *signature*, which declare the set of possible controls labelling nodes of the bigraph and for each control $K$ the number of binding and free ports of nodes in the bigraph labelled with $K$. The signature also declares each control as either atomic, active or passive. Only nodes with non-atomic controls can have children, and reactions (as defined below) can only occur in sub-bigraphs nested solely within active controls, i.e. the active controls determine evaluation contexts.

### 5.2.2 BPL Tool Term Language

Binding bigraphs are often visualized graphically. However, binding bigraphs also admit a representation via a term language based on the axiomatization of binding bigraphs [12]. This representation is exploited in the BPL Tool to allow compact and compositional textual descriptions of binding bigraphs and their reaction rules[1].

In the present paper we will use the syntax of the term language as used in the BPL Tool. The language consists of Standard ML constructs which allows the user to write the terms directly in SML, at the cost of introducing a few additional back quotes. (Future versions of the BPL Tool will also support a clean input language stripped of SML artifacts.) The employed subset of the language can be defined by the following grammar.

$$P ::= P \texttt{ o } P \mid P \texttt{ || } P \mid P \texttt{ `|` } P \mid C$$
$$C ::= c \mid c\texttt{[}N^?\texttt{]} \mid c\texttt{[}N^?\texttt{][}NS^?\texttt{]} \mid \texttt{-//[}N^?\texttt{]} \mid n\texttt{//[}N^?\texttt{]} \mid \texttt{`[}N^?\texttt{]`} \mid \texttt{<->}$$
$$N^? ::= \epsilon \mid N \quad N ::= n \mid n\texttt{,}N \quad NS^? ::= \epsilon \mid NS \quad NS ::= \texttt{[}N^?\texttt{]} \mid \texttt{[}N^?\texttt{],}NS$$

where $n$ ranges over strings representing names and $c$ over strings representing controls. $C$ describes so-called *ions* which are bigraphs consisting of a single root with a single node as child, having a control as defined in the signature. If the control is non-atomic the ion has a single hole inside. For instance, an ion with name $c$ and $i$ free ports and $j$ binding ports is written $c\texttt{[}n_1,\ldots,n_i\texttt{][}NS_1,\ldots,NS_j\texttt{]}$ where the $NS_k$ is the set of names bound to the $k$th binding port.

We use the double bars `||` to separate roots in the place graph and the single bar `` `|` `` as a separator between sibling nodes. The symbol `o` denotes composition as defined above (the tool checks that the interfaces of the bigraphs match). The terms $\texttt{-//[}N^?\texttt{]}$ and $n\texttt{//[}N^?\texttt{]}$ denote a bigraph with an empty place graph (i.e. no roots) and a link graph mapping the names in the list $N^?$ to respectively each their closed link and to the name $n$. The term `` `[]' `` denotes a hole and the term $\texttt{`[}n_1,\ldots,n_k\texttt{]`}$ denotes a hole with local names $n_1,\ldots,n_k$. Finally, the term `<->` denotes a bigraph just consisting of a single empty root. As an example, we may define two binding bigraphs as follows (and depict the graphical representation of them in Fig. 5.1).

```
val R = If[inst_id] o (Condition o False `|` Then o `[]` `|` Else o `[]`)
```

[1]The representation is also exploited in the underlying formalization and implementation of matching used for the execution of reaction rules as described in [18].

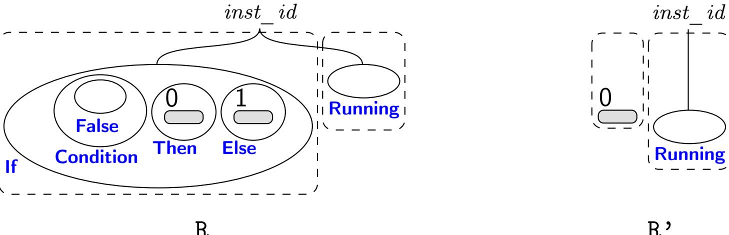



Figure 5.1: Example bigraphs `R` : $I \rightarrow J$ and `R'` : $I' \rightarrow J$, where $I = \langle 2, [\{\}, \{\}], \{\}\rangle, I' = \langle 1, [\{\}], \{\}\rangle, J = \langle 2, [\{\}, \{\}], \{inst_id\}\rangle$

```
     || Running[inst_id]
val R' = `[]` || Running[inst_id]
```

The bigraphs `R` and `R'` both have two roots. The first root of `R` has a single node as child with the control `If` and a single free port linked to the name `inst_id`. The node has three nodes as children, labelled respectively with the controls `Condition`, `Then` and `Else`. The first node has a single node as child labelled with the atomic control `False`. The two latter nodes both have a hole as a child. The holes in a bigraph term are ordered from left to right, i.e. the hole below the `Then` is indexed 0 and the hole below the `Else` is indexed 1. The second root of `R` has a single node as child labelled with the atomic control `Running` and a single free port linked to the name `inst_id`. The bigraph `R'` has simply a hole below its first root and the atomic `Running` control below its second root. The two bigraphs in fact form respectively the redex and reactum of a reaction rule, as defined below, defining the meaning of an if-then-else construct in the case where the condition has been evaluated to false.

### 5.2.3 Parametric Reaction Rules

The dynamics of bigraphical reactive systems is defined in terms of a reaction relation generated from a set of reaction rules $\mathcal{R}$. Such rules are generally parametric, and may discard and also duplicate their parameters.

A rule, written `"rule name" ::: R -`$\bar{\varrho}$`-|> R'`, consists of two bigraphs: the *redex* `R` : $I \rightarrow J$ and the *reactum* `R'` : $I' \rightarrow J$, where both $I$ and $I'$ are local interfaces, and a parameter mapping $\bar{\varrho}$. The mapping $\bar{\varrho}$ indicates for each site in the reactum from which site in the redex the parameter is copied.

The expression `"if false" ::: R -[0 |-> 1]-|> R'` is a reaction rule for executing an `If` activity with a false condition. During a reaction, the first tree of `R` (the if-then-else construct) is replaced by the first tree the reactum `R'`. Since the second tree of `R` and `R'` are identical it simply means that a node with the `Running` control (and the correct id link) must be present in the context—this is used to ensure that rewrites are only performed on running instances which are ready to execute a step. The mapping `[0 |-> 1]` specifies that the hole in the reactum (site 0) should contain the contents of the hole in the `Else`-branch (site 1), while the contents of the hole in the `Then`-branch is discarded as it is not mentioned in the mapping.

In general parameters may have local names, thus the mapping $\bar{\varrho}$ must also define how the local names of a parameter is mapped to local names in the hole of the reactum. For instance, `[0&[x1] |-> 0&[x], 1&[x2] |-> 0&[x]]` is a mapping which (a) maps site 0 of the reactum and its local name `x1` to site 0 of the redex and its local name `x`, and (b) also maps site 1 of the reactum and its local name `x2` to site 0 of the redex and its local name `x`.

A rule matches an agent `a` if `a` = `C o (id`$_Z$` || R) o d` for some identity linking `id`$_Z$ and active context `C` (i.e., no site of `C` is nested within a passive node); the linking `id`$_Z$ connects all non-local

| | | |
|---|---|---|
| *proc* | ::= | `Process`(*scopecontent*) |
| *scopecontent* | ::= | *partnerlinks vars act* |
| *partnerlinks* | ::= | `PartnerLinks(PartnerLink`$^*$`)` |
| *vars* | ::= | `Variables(Variable`$^*$(*value*?)) |
| *act* | ::= | *scope* \| *seq* \| *flow* \| *while* \| *if* \| *assign* |
| | \| | `Invoke` \| `Receive` \| `Reply` \| `Exit` |
| *scope* | ::= | `Scope`(*scopecontent*) |
| *seq* | ::= | `Sequence`(*act act*) |
| *flow* | ::= | `Flow`(*act*$^*$) |
| *while* | ::= | `While(Condition`(*expr*) *act*?) |
| *if* | ::= | `If(Condition`(*expr*) `Then`(*act*?) `Else`(*act*?)) |
| *assign* | ::= | `Assign(Copy(From To))` |
| *value* | ::= | `true`() \| `false`() |
| *expr* | ::= | *value* \| $\$x$ |

Table 5.1: Grammar for WS-BPEL processes.

names in the outerface of `d` to `C`. In this case reaction produces a new agent `a' = C o (id`$_Z$ `|| R') o d'`, where `d'` is computed from `d` as prescribed by $\bar{\varrho}$. When duplicating parts of the agent (by letting $\bar{\varrho}$ map several reactum sites to a single redex site), *local* links in `d` are *copied* to each copy in `d'`, while *free* links are *shared* between the copies. Binding ports thus enforce a notion of scope and locality on a bigraph's links, resembling the usual notion of binders in the $\lambda$- and the $\pi$-calculus. This feature of binding bigraphs is crucial in our formalization of WS-BPEL to create *fresh* id and scope links when new instances or scopes are created.

## 5.3 Formalizing WS-BPEL in the BPL Tool

In this section we present the subset of WS-BPEL considered in this report and its formalization in the BPL Tool. First we present the WS-BPEL subset in Sec. 5.3.1. Second we present the static representation in Sec. 5.3.2, i.e. the representation of processes and instances. Third we present the reaction rules capturing the dynamic behavior of WS-BPEL in Sec. 5.3.3.

### 5.3.1 WS-BPEL

We consider a subset of the WS-BPEL syntax given by the grammar in Tab. 5.1. For brevity we do not use XML notation or an XML schema and omit attributes in the grammar. We use ? and $*$ to indicate that an element can appear at most once and any number of times respectively. We also assume that sequence elements always contain exactly two actions. For technical reasons, which will be explained in Sec. 5.3.3, we also assume that receive elements with the `createInstance="yes"` attribute refer to a partner link defined in the outermost scope. We write attributes as sets following the element name (e.g. `Process {name=echo}`) and let $A$ range over such sets. If an element has no attributes, we leave out the set of attributes. Note that in regard to data flow we only consider the constant values given by the XPath expressions `true`() and `false`() and references to variables, assuming that $x$ ranges over strings. We let **BPEL** refer to the set of terms defined by the grammar.

### 5.3.2 The Static Representation

We define our bigraphical representation of WS-BPEL in the BPL Tool with respect to the signature given in Tab. 5.2. As explained in the previous section, the signature determines the allowed controls for labeling nodes, and for each control the number of binding and free ports of nodes labeled with this control. For instance, we write `Reply =: 0 -> 6` for a control called `Reply` with binding arity 0 and free arity 6, which can be abbreviated to `Reply -: 6`. A control having zero binding and free arity is declared by just writing the control name, e.g. `Next`.

| *Active controls* | *Passive controls* | *Atomic controls* |
|---|---|---|
| `PartnerLinks` | `Process =: 1 -> 1` | `To -: 2` |
| `Variables` | `Scope =: 1 -> 1` | `From -: 2` |
| `If -: 1` | `Variable -: 2` | `ToPLink -: 2` |
| `Condition` | `While -: 1` | `FromPLink -: 2` |
| `Sequence -: 1` | `Then` | `Invoke -: 8` |
| `Flow -: 1` | `Else` | `Receive -: 6` |
| | `Assign -: 1` | `Reply -: 6` |
| | `Copy` | `Exit -: 1` |
| | `PartnerLink -: 2` | `True` |
| | | `False` |
| | | `VariableRef -: 3` |
| | | |
| | | `CreateInstance -: 1` |
| | | `GetReply -: 6` |
| | | `ReplyTo -: 2` |
| | | `Link -: 1` |
| | | `Running -: 1` |
| `Instance -: 2` | `Next` | `Invoked -: 1` |
| `ActiveScope -: 2` | `Message -: 1` | `Stopped -: 1` |

Table 5.2: Signature for WS-BPEL

The controls listed in the upper part of the signature correspond directly to the subset of WS-BPEL elements we are considering[2] (cf. Tab. 5.1) and allow us to give a very direct representation of WS-BPEL processes, while the controls listed in the lower part are introduced to facilitate the formalization of the execution semantics. We will call the bigraphical reactive system for $\mathbf{BRS}^{\text{BPEL}}$.

As an example, consider the process in Fig. 5.2(a). The process is represented in the BPL Tool as shown in Fig. 5.2(b). The graphical representation, generated by the BPL Tool, is shown in Fig. 5.2(c) (the link to the binding port of the `Process` node has been colored green to improve readability). The place graph (nesting of controls) and the link graph correspond almost directly to the nesting of elements and the sharing of attributes of the XML representation respectively. But we need to introduce some additional structure. The main differences are:

1. Since the children of a node in a bigraph are unordered, and children of an XML element are ordered, we represent the sequence construct as a nesting of binary sequence constructs, in which the second activity is enclosed in a node labeled by the `Next` control.

2. To be able to identify the scope of partner links and variables we have added an explicit link from the `PartnerLink` and `Variable` nodes to a binding port of the `Process` and `Scope` nodes. This will be explained below when we describe the semantics of assignment and scopes. For similar reasons, we also link expressions and activities to the binding port of the enclosing `Process` node.

3. To be able to identify the initial receive actions we insert `CreateInstance` nodes in each `PartnerLink` which identify an operation for which there is a receive activity with the `createInstance="yes"` attribute using that partner link.

4. We require `PartnerLinks` and `Variables` nodes in each scope (including process). This allows for fewer and simpler reaction rules. This is a technicality as the absence of e.g. a `variables` declaration in a WS-BPEL process is equivalent to an empty `variables` declaration, so we just make them explicit in the representation.

We represent the XPath expressions `true()` and `false()` by nodes with the controls `True` and `False`, respectively. Variable references (e.g. `$x`) are represented by `VariableRef` nodes.

[2]WS-BPEL allows several forms of the from and to elements. We have chosen to formalize two of these, namely those for variables (`From`, `To`) and partner links (`FromPLink`, `ToPLink`).

$$
\begin{aligned}
\mathcal{C}\llbracket \texttt{Process \{name=}n\texttt{\}}(pls\ vars\ act)\rrbracket &= \texttt{Process[}n\texttt{][[}n_id\texttt{]]}\circ \\
&\quad (\llbracket pls\rrbracket^{n_id,n_id}_{\{n_id\},[],\mathcal{I}\llbracket act\rrbracket}\ \texttt{`|`}\ \llbracket vars\rrbracket^{n_id,n_id}_{\{n_id\},[],\phi_0}\ \texttt{`|`}\ \llbracket act\rrbracket^{n_id,n_id}_{\{n_id\},[pls\cup vars\mapsto n_id],\phi_0})
\end{aligned}
$$

$$
\begin{aligned}
\llbracket \texttt{PartnerLink \{name=}n\texttt{\}}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{PartnerLink[}n\texttt{,}s_id\texttt{]} \circ \phi(n) \\
\llbracket \texttt{Variable \{name=}n\texttt{\}}(p)\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{Variable[}n\texttt{,}s_id\texttt{]} \circ \llbracket p\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} \\
\llbracket \texttt{Scope}(pls\ vars\ act)\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{Scope[}x_id\texttt{][[}y_id\texttt{]]}\circ \\
&\quad (\llbracket pls\rrbracket^{x_id,y_id}_{\tilde{x},\sigma,\phi}\ \texttt{`|`}\ \llbracket vars\rrbracket^{x_id,y_id}_{\tilde{x}y_id,\sigma,\phi}\ \texttt{`|`}\ \llbracket act\rrbracket^{x_id,y_id}_{\tilde{x}y_id,\sigma[pls\cup vars\mapsto y_id],\phi}) \\
\llbracket \texttt{Sequence}(act\ act')\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{Sequence[}x_id\texttt{]} \circ (\llbracket act\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi}\ \texttt{`|`}\ \texttt{Next} \circ \llbracket act'\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi}) \\
\llbracket \texttt{From \{var=}n\texttt{\}}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{From[}n\texttt{,}\sigma(n)\texttt{]} \\
\llbracket \texttt{To \{var=}n\texttt{\}}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{To[}n\texttt{,}\sigma(n)\texttt{]} \\
\llbracket \texttt{From \{partnerLink=}n\texttt{\}}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{FromPLink[}n\texttt{,}\sigma(n)\texttt{]} \\
\llbracket \texttt{To \{partnerLink=}n\texttt{\}}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{ToPLink[}n\texttt{,}\sigma(n)\texttt{]} \\
\llbracket \texttt{Receive } A\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{Receive[}n\texttt{,}\sigma(n)\texttt{,}op\texttt{,}x\texttt{,}\sigma(x)\texttt{,}x_id\texttt{]}, \\
&\quad \text{if } A \supseteq \texttt{\{partnerLink=}n\texttt{,operation=}op\texttt{,variable=}x\texttt{\}} \\
\llbracket \texttt{Invoke } A\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{Invoke[}n\texttt{,}\sigma(n)\texttt{,}op\texttt{,}ix\texttt{,}\sigma(ix)\texttt{,}ox\texttt{,}\sigma(ox)\texttt{,}x_id\texttt{]}, \\
&\quad \text{if } A = \texttt{\{partnerLink=}n\texttt{,operation=}op\texttt{,} \\
&\qquad\quad \texttt{inputVariable=}ix\texttt{,outputVariable=}ox\texttt{\}} \\
\llbracket \texttt{Reply } A\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{Reply[}n\texttt{,}\sigma(n)\texttt{,}op\texttt{,}x\texttt{,}\sigma(x)\texttt{,}x_id\texttt{]}, \\
&\quad \text{if } A = \texttt{\{partnerLink=}n\texttt{,operation=}op\texttt{,variable=}x\texttt{\}} \\
\llbracket E(p)\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= E \circ \llbracket p\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} \\
&\quad \text{where } E \in \{\texttt{PartnerLinks}, \texttt{Variables}, \texttt{Then}, \texttt{Else}, \texttt{Condition}, \texttt{Copy}\} \\
\llbracket Eid(p)\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= Eid\texttt{[}x_id\texttt{]} \circ \llbracket p\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} \\
&\quad \text{where } Eid \in \{\texttt{Flow}, \texttt{If}, \texttt{While}, \texttt{Assign}, \texttt{Exit}\} \\
\llbracket \texttt{true()}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{True} \\
\llbracket \texttt{false()}\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{False} \\
\llbracket \$x\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{VariableRef[}x\texttt{,}\sigma(x)\texttt{,}x_id\texttt{]} \\
\llbracket p\ p'\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \llbracket p\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi}\ \texttt{`|`}\ \llbracket p'\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} \\
\llbracket \varepsilon\rrbracket^{x_id,s_id}_{\tilde{x},\sigma,\phi} &= \texttt{<->}
\end{aligned}
$$

Table 5.3: Translating **BPEL** into **BRS**$^{\text{BPEL}}$.

```
<process name="echo_process">
  <partnerLinks><partnerLink name="echo_client" /></partnerLinks>
  <variables><variable name="x" /></variables>
  <sequence>
    <receive partnerLink="echo_client" operation="echo"
             createInstance="yes" variable="x" />
    <reply   partnerLink="echo_client" operation="echo" variable="x" />
  </sequence>
</process>
```

(a) Example WS-BPEL process.

```
val echo_process =
Process[echo_process][[echo_id]]
o (    PartnerLinks o PartnerLink[echo_client, echo_id] o CreateInstance[echo]
   `|` Variables    o Variable[x, echo_id] o <->
   `|` Sequence[echo_id] o (
         Receive[echo_client, echo_id, echo, x, echo_id, echo_id]
   `|` Next o (
         Reply[echo_client, echo_id, echo, x, echo_id, echo_id])))
```

(b) BPL Tool representation of example process.

(c) BPL Tool visualization of example process.

Figure 5.2: Example process.

We give the formal definition of the map $\mathcal{C}[\![-]\!] : \mathbf{BPEL} \to \mathbf{BRS}^{\mathsf{BPEL}}$ in Tab. 5.3. The map is defined using a map $[\![-]\!]^{x_id,s_id}_{\tilde{x},\sigma,\phi} : \mathbf{BPEL} \to \mathbf{BRS}^{\mathsf{BPEL}}$ on sub-terms indexed by names $x_id$ and $s_id$, indicating the name connected to the bound link identifying the process and the scope respectively, a set $\tilde{x}$ of names constituting the current names of bound links, a scope map $\sigma$ mapping every defined partnerlink name or variable name to its scope link, and a map $\phi : \mathbf{Name} \to \mathbf{BRS}^{\mathsf{BPEL}}$ mapping partnerlink names to bigraph terms of the form `CreateInstance[`$op_1$`]`\``|`\``...`\``|`\` `CreateInstance[`$op_n$`]` indicating for a given partnerlink name $n$ that the process contains terms `Receive` $A_i$ where `{partnerLink=`$n$`, operation=`$op_i$`, createInstance=yes}` $\subseteq A_i$, $1 \leq i \leq n$. We use the map $\mathcal{I}[\![-]\!] : \mathbf{BPEL} \to \mathbf{Name} \to \mathbf{BRS}^{\mathsf{BPEL}}$ defined in Tab. 5.4 to find $\phi$ from the body of the process. For a set of partnerlink and variable declarations *pls* and *vars* we will write $\sigma[pls \cup vars \mapsto x_id]$ for the update of the scope map $\sigma$ mapping every partnerlink and variable name in *pls* and *vars* to the name $x_id$ and every other name $n$ in the domain of $\sigma$ to $\sigma(n)$. We write $\tilde{x}y$ for the disjoint union of $\tilde{x}$ and $\{y\}$ (i.e. implying $y \notin \tilde{x}$), which is used in generating fresh bound names of scopes. Note that we assume a partition of the set of names into two disjoint sets, one ranged over by strings with the suffix $_id$ (e.g. $x_id$), which is used for scope identifiers, and one ranged over by strings without the suffix, which corresponds to XML attribute values. This prevents clashes between scope identifiers introduced in the translation and attribute values. We let $p$ range over any sub term of **BPEL**, including the empty term $\varepsilon$.

$$
\begin{aligned}
\mathcal{I}[\![\texttt{Receive } A]\!] &= \begin{cases} \phi_0[n \mapsto \texttt{CreateInstance[}op\texttt{]}] & \text{if } ci = \texttt{yes} \\ \phi_0 & \text{otherwise} \end{cases} \\
&\quad \text{where } A \supseteq \{\texttt{partnerLink=}n\texttt{,operation=}op\texttt{,createInstance=}ci\texttt{\}} \\
\mathcal{I}[\![E\ A(p)]\!] &= \phi_0 \\
&\quad \text{where } E \in \{\texttt{Condition}, \texttt{Assign}, \texttt{Invoke}, \texttt{Reply}, \texttt{Exit}, \texttt{Variables}, \texttt{PartnerLinks}\} \\
\mathcal{I}[\![E\ A(p)]\!] &= \mathcal{I}[\![p]\!] \\
&\quad \text{where } E \in \{\texttt{Scope}, \texttt{Flow}, \texttt{Sequence}, \texttt{While}, \texttt{If}, \texttt{Then}, \texttt{Else}\} \\
\mathcal{I}[\![p\ p']\!] &= \mathcal{I}[\![p]\!] \mid \mathcal{I}[\![p']\!] \\
\mathcal{I}[\![\varepsilon]\!] &= \phi_0
\end{aligned}
$$

$$
\begin{aligned}
\phi_0(n) &= \texttt{<->} \\
(\phi \mid \phi')(n) &= \phi(n)\ \texttt{`|`}\ \phi'(n)
\end{aligned}
$$

Table 5.4: Finding `Receive` terms with `createInstance=yes`.

A key feature of the formalization is that active process instances are represented almost as the processes, the main difference is that they are nested within an (active) `Instance` control instead of a (passive) `Process` control. Fig. 5.3(b) is an example of an instance, which is visualized in Fig. 5.3(c) (again some of the links have been colored to improve readability). It exemplifies the case, where the echo process has been invoked resulting in a new instance of that process, which has performed the initial receive activity. We have left out the calling instance from the figure to keep the example simple — the edges "client id edge" and "echo id edge", is connected to respectively the id port and a port of a link node in the PartnerLink (used for getting the reply) in the calling instance.

One might use the close correspondence between bigraphs and XML to translate the representation of instances into XML as shown in Fig. 5.3(a). This illustrates that the run-time execution format is very close to the process specification format.

However, notice that we have also added a `Running` node in the instance; we call this the *status* node of the instance. The purpose of the status node is twofold and somewhat intricate, and will be explained below when we describe the reaction rules.

### 5.3.3 Reaction Rules

In this section we present the reaction rules used in the formalization of WS-BPEL. The reaction rules (in BPL Tool syntax) is also available via the on-line tool[3].

#### Structural Activities

The rules for structural activities covers completion of flows and sequences, conditionals (if-then-else) and iteration (while-loop).

**Completion of Activities:** When a Flow is completed (i.e. there are no more instructions in the flow to be executed) we garbage collect the flow, by replacing the `Flow` node with an empty bigraph (denoted by `<->`).

```
"flow completed" :::
```

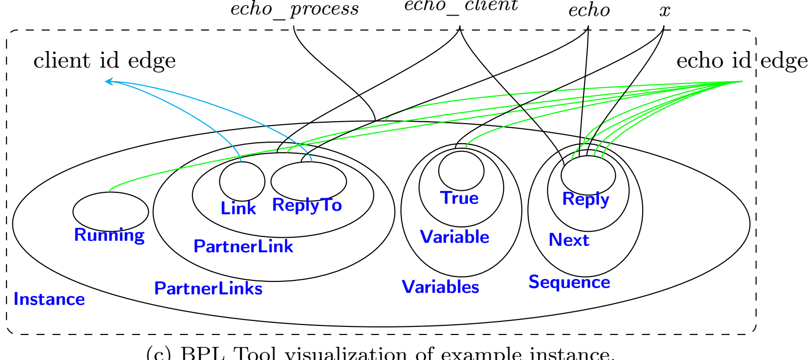
[3]See `http://tiger.itu.dk:8080/bplweb/index/18`

```
<instance name="echo_process" id="echo_id">
  <running id="echo_id" />
  <partnerLinks>
    <partnerLink name="echo_client" />
       <link id="client_id" /><replyTo id="client_id" />
    </partnerLink>
  </partnerLinks>
  <variables><variable name="x">True</variable></variables>
  <sequence>
    <reply   partnerLink="echo_client" operation="echo" variable="x" />
  </sequence>
</instance>
```

(a) Example WS-BPEL instance.

```
val echo_instance =
Instance[echo_process, echo_id]
o (     Running[echo_id]
   `|` PartnerLinks
       o PartnerLink[echo_client, echo_id]
         o (Link[client_id] `|` ReplyTo[echo, client_id])
   `|` Variables o Variable[x, echo_id] o True
   `|` Sequence[echo_id] o (
       Next o (
         Reply[echo_client, echo_id, echo, x, echo_id, echo_id])))
```

(b) BPL Tool representation of example instance.




(c) BPL Tool visualization of example instance.

Figure 5.3: Example instance.

```
   Flow[inst_id] o <->
|| Running[inst_id]
  ----|>
   <->
|| Running[inst_id];
```

In the same manner, we garbage collect a Sequence if the current instruction is completed (i.e. if there is no current instruction). We then make the following instruction the next to be executed by replacing the `Sequence` node with the content of the `Next` node.

```
"sequence completed" :::

   Sequence[inst_id] o Next o `[]`
|| Running[inst_id]
  --[0 |-> 0]--|>
   `[]`
|| Running[inst_id];
```

**Conditionals:** The rules for evaluating an if-then-else statement is as expected. If the condition is True we execute the then-branch, otherwise we execute the else-branch. One of the two rules

for evaluating an if-then-else statement was already given in Sec. 5.2 (rule `if false`), so we only present the rule for when the condition is true in this section. The rule is similar to the rule given in Sec. 5.2 except for the value of the condition and the instantiation.

```
"if true" :::

   If[inst_id] o (    Condition o True
                 `|` Then o `[]`
                 `|` Else o `[]`)
|| Running[inst_id]
  --[0 |-> 0]--|>
   `[]`
|| Running[inst_id];
```

**Iteration:** We give semantics to a while-loop in the traditional manner, by unfolding the loop once and using an if-then-else statement with the loop condition. In the syntax of the BPL Tool (emphasizing the order of the holes using Standard ML comments), the rule `while unfold` for unfolding looks as follows.

```
"while unfold" :::

   While[inst_id] o (Condition o `[]` `|` `[]`)
|| Running[inst_id]

  --[0 |-> 0, 1 |-> 1, 2 |-> 0, 3 |-> 1]--|>

   If[inst_id] o (    Condition o `[]`
                 `|` Then o Sequence[inst_id] o (
                               `[]`
                        `|` Next o
                                While[inst_id]
                                o (Condition o `[]` `|` `[]`))
                 `|` Else o <->)
|| Running[inst_id];
```

Note how the instantiation `[0 |-> 0, 1 |-> 1, 2 |-> 0, 3 |-> 1]` on the arrow of the rule describes that the parameter in hole 0 (the condition expression) is copied and placed in both hole 0 and hole 2 of the reactum. Also, the parameter in hole 1 (the body of the while loop) is copied and placed in both hole 1 and hole 3 of the reactum. One may also note that the empty process, to be executed in the `Else` branch, is represented by the bigraph with a single barren root. As explained above, the `Running` node linked to the `While` node via the name `inst_id` is used to guarantee that the instance which the while-loop is part of is indeed running.

### Expression Evaluation

Our current formalization only supports one type of expressions, namely variable references. But one can easily extend the semantics to more expression types, simply by adding rules describing how to evaluate them — without having to alter the current rules.

A variable reference is evaluated by locating the referenced variable, using its name and "scope"-link, and then replacing the `VariableRef` node by the current content of the variable.

```
"variable reference" :::

   VariableRef[var, var_scope, inst_id]
|| Variable[var, var_scope] o `[]`
|| Running[inst_id]

  --[0 |-> 0, 1 |-> 0]--|>

   `[]`
|| Variable[var, var_scope] o `[]`
|| Running[inst_id];
```

### Assignment and Dynamic Manipulation of Partner Links:

Of the many variants of the "Assign" activity in WS-BPEL, we cover in our formalization only the four allowing for copying the content of a `Variable` or `PartnerLink` to a `Variable` or `PartnerLink`. Each are covered by a *single* rule in the formalization. Below we show the case of the rule `assign copy plink2var` which copies the content of the `PartnerLink` node referenced to by the `FromPLink` node to the `Variable` node referenced to by the `To` node. The remaining 3 rules are quite similar.

```
"assign copy plink2var" :::

   Assign[inst_id] o Copy o (    FromPLink[f, scope1]
                             `|` To[t, scope2])
|| PartnerLink[f, scope1] o `[]`
|| Variable[t, scope2] o `[]`
|| Running[inst_id]

  --[0 |-> 0, 1 |-> 0]--|>

   <->
|| PartnerLink[f, scope1] o `[]`
|| Variable[t, scope2] o `[]`
|| Running[inst_id];
```

The instantiation describes that the parameter of hole 0 is copied to both hole 0 and 1 in the reactum, and that the content of hole 1 is discarded. The `f` and `t` links determine the name of the partner link and the variable respectively. However, the name alone may not uniquely determine a variable (or partner link). Since variables (or partner links) may be defined within nested scopes, several variables (or partner links) may have the same name. In this case the WS-BPEL specification states that the closest variable (partner link) should be fetched. We represent this in the formalization by letting the `scope1` and `scope2` links connect respectively the `FromPLink` and `To` nodes to the closest partner link and variable with the correct name.

### Scopes

The formalization of nested scopes makes crucial use of *bound* links. In WS-BPEL, local scopes may be defined within a while loop. As an example consider the while loop shown below.

```
<while>
  <condition>true()</condition>
  <scope>
    <variables><variable name="x" /></variables>
    <assign>
       <copy><from partnerlink="echo_client" /><to variable="x" /></copy>
    </assign>
  </scope>
</while>
```

Every iteration of the while loop, i.e. every unfolding of the loop, must create a *new* scope, containing a new copy of the variable. If we (naively) used a normal, i.e. free, link within the scope to connect the reference to $x$ in the assignment, then the two copies created by the rule `while unfold` would share the *same* scope link. The consequence would be that the assignment rules would not tell the two scopes apart, and thus the variable from the wrong scope could be used in the assignment. For this reason we let the scope links be connected to a *binding* port of the `Scope` control. This ensures that each copy of the scope gets its own bound link. However, this introduces another problem: In the rule `assign copy plink2var` above we *must* place the assignment, partner link and variable controls below different roots, since they could all potentially be located in different scopes. However, interfaces of binding bigraphs do not allow a bound link to be shared between two nodes located below two different roots in the place graph.

To cope with this problem, we make the `Scope` control passive, and introduce a rule `scope activation` as defined below. The rule replaces the passive `Scope` node with an active `ActiveScope`

node, where the binding port is replaced by a normal (free) port, and the bound link by an edge connected to that port. The rule is defined as follows in the BPL Tool syntax.[4]

```
"scope activation" :::

   Scope[inst_id][[scope]] o ‘[scope]‘
|| Running[inst_id]
  --[0 |-> 0]--|>
   -//[scope] o (ActiveScope[inst_id, scope] o ‘[scope]‘)
|| Running[inst_id];
```

Note how the use of single and double square brackets of the control `Scope` specify that `inst_id` is a normal port and `scope` is a binding port, whereas `ActiveScope[inst_id, scope]` has two normal ports. Note also that the link map `-//[scope]` closes the link connected to the scope port in the reactum, and that the name `scope` is local to the hole 0 in both redex and reactum. In a similar manner we need to be able to activate scopes in newly created instances, while their status is still `Invoked`.

```
"scope activation 2" :::

   Scope[inst_id][[scope]] o ‘[scope]‘
|| Invoked[inst_id]
  --[0 |-> 0]--|>
   -//[scope] o (ActiveScope[inst_id, scope] o ‘[scope]‘)
|| Invoked[inst_id];
```

When we are finished executing the body of the scope we remove the scope, including its variables, partner links, and its associated “scope”-edge.

```
"scope completed" :::

   ActiveScope[inst_id, scope]
   o (Variables o ‘[]‘  ‘|‘  PartnerLinks o ‘[]‘)
|| Running[inst_id]
  ----|>
   <-> || scope//[]
|| Running[inst_id];
```

### Process Termination

Processes can terminate in two different ways: 1) normally, i.e. when no more activities remain, or 2) abnormally by executing an `Exit` activity.

**Normal Termination:** In the first case, we simply remove the instance in the same way as for scopes. The precondition for the reaction rule is that there are no activities remaining in the instance, and we then replace the instance with an empty bigraph. As redex and the reactum are required to have the same outer face we add the “idle” link `proc_name` and `inst_id` using a wiring `proc_name//[] || inst_id//[]`.

```
"inst completed" :::

Instance[proc_name, inst_id]
o (Variables o ‘[]‘  ‘|‘  PartnerLinks o ‘[]‘  ‘|‘  Running[inst_id])
  ----|>
<-> || proc_name//[] || inst_id//[];
```

[4]An alternative solution to this problem would be to use so-called local bigraphs, which exactly allow the more general interfaces where names can be bound to several roots. Alas, local bigraphs are not supported by the BPL Tool.

**Abnormal Termination:** The `Exit` activity in WS-BPEL allows processes to be abnormally terminated. Its semantics is given by two rules: The first rule `exit stop instance` changes the status of the instance from running to stopped by replacing the `Running` node inside the instance with a `Stopped` node.

```
"exit stop instance" :::

   Exit[inst_id]
|| Running[inst_id]
  ----|>
   <->
|| Stopped[inst_id];
```

The second rule `exit remove inst` removes the instance together with all its remaining content. This is simply done by replacing the `Instance` node with the empty root bigraph `<->` and discarding the parameter in hole 0.

```
"exit remove inst" :::

Instance[proc_name, inst_id]
o (Stopped[inst_id] `|` `[]`)
  ----|>
<-> || proc_name//[] || inst_id//[];
```

(Again the "idle" links `proc_name//[] || inst_id//[]` in the reactum is simply there to ensure that the redex and reactum have the same outer face.)

One may think that the above semantics could be defined as a single rewrite rule, with a redex matching an instance containing an active `Exit` activity, and a reactum that simply replaces this instance with the empty bigraph `<->`. However, the `Exit` node may be nested arbitrarily deep (e.g. inside `Flow` nodes) within the `Instance` node. This cannot be captured in the format of parametric rules of binding bigraphs.[5] Therefore we first match on the status node `Running` and the `Exit` node and change the status to `Stopped`. As the status node is a child of the `Instance` node, we can write a rule which matches instances which are stopped and discard them. All other rules, except for the rule `exit remove inst`, checks for the presence of the `Running` node, so the two reaction rules will always be applied consecutively.

For similar reasons, we also change the status temporarily to `Invoked` when creating a new instance, that is, when we execute a receive activity with the `createInstance="yes"` attribute.

### Communication

The formalization includes synchronous request-response communication which is achieved in WS-BPEL using, in order, the invoke, receive, and reply activities. There are two cases: the receive can either 1) be an activity of a running instance, or 2) it can create a new instance of a process.

The first case is implemented by the `invoke instance` rule which handles both the invoke and receive in one step, while the second is modeled by two rules: `invoke` and `receive`. The content of partner links is used in the rules for invoke and reply activities to determine the target instance for communication. Thus, the ability of copying between partner links and variables makes it possible to send partner links as messages and dynamically assign instances as target for communication.

The rule `invoke instance` below allows two active instances to communicate. It synchronizes an active `Invoke` activity in one instance with a corresponding `Receive` activity in another instance, replacing the `Invoke` with a `GetReply` activity and removing the `Receive`. The instantiation map ensures that the content of the input variable `invar` (hole 1) is copied to the appropriate variable of the receiving instance (hole 3 in the reactum).

[5] This however is possible using the higher-order reaction rules introduced in [22, 23].

```
"invoke_instance" :::

   Invoke[partner_link_invoker, partner_link_scope_invoker, oper,
          invar, invar_scope, outvar, outvar_scope, inst_id_invoker]
|| PartnerLink[partner_link_invoker, partner_link_scope_invoker]
   o (Link[inst_id_invoked] `|` `[]`)
|| Variable[invar, invar_scope] o `[]`
|| Running[inst_id_invoker]
|| Receive[partner_link_invoked, partner_link_scope_invoked, oper,
           var, var_scope, inst_id_invoked]
|| PartnerLink[partner_link_invoked, partner_link_scope_invoked] o `[]`
|| Variable[var, var_scope] o `[]`
|| Running[inst_id_invoked]

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 1]--|>

   GetReply[partner_link_invoker, partner_link_scope_invoker, oper,
            outvar, outvar_scope, inst_id_invoker]
|| PartnerLink[partner_link_invoker, partner_link_scope_invoker]
   o (Link[inst_id_invoked] `|` `[]`)
|| Variable[invar, invar_scope] o `[]`
|| Running[inst_id_invoker]
|| <->
|| PartnerLink[partner_link_invoked, partner_link_scope_invoked]
   o (`[]` `|` ReplyTo[oper, inst_id_invoker])
|| Variable[var, var_scope] o `[]`
|| Running[inst_id_invoked];
```

A similar rule allows the `GetReply` activity to synchronize with the corresponding `Reply` activity in the invoked instance, thereby copying the content from variable `var` to variable `outvar`.

```
"reply" :::

   Reply[partner_link_invoked, partner_link_scope_invoked, oper,
         var, var_scope, inst_id_invoked]
|| PartnerLink[partner_link_invoked, partner_link_scope_invoked]
   o (ReplyTo[oper, inst_id_invoker] `|` `[]`)
|| Variable[var, var_scope] o `[]`
|| Running[inst_id_invoked]
|| GetReply[partner_link_invoker, partner_link_scope_invoker, oper,
            outvar, outvar_scope, inst_id_invoker]
|| PartnerLink[partner_link_invoker, partner_link_scope_invoker]
   o (Link[inst_id_invoked] `|` `[]`)
|| Variable[outvar, outvar_scope] o `[]`
|| Running[inst_id_invoker]

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 1]--|>

   <-> || oper//[]
|| PartnerLink[partner_link_invoked, partner_link_scope_invoked] o `[]`
|| Variable[var, var_scope] o `[]`
|| Running[inst_id_invoked]
|| <->
|| PartnerLink[partner_link_invoker, partner_link_scope_invoker]
   o (Link[inst_id_invoked] `|` `[]`)
|| Variable[outvar, outvar_scope] o `[]`
|| Running[inst_id_invoker];
```

The `invoke` rule represents the case where an `Invoke` activity is executed inside a running instance and we have a process with the appropriate operation available and marked as being able to create new instances. The reactum 1) replaces the `Invoke` activity in the calling instance with a `GetReply` activity, which is used to represent that the instance is waiting for the reply, and 2) creates a new instance with the body of the process definition and the value of the input variable in a `Message` node within the relevant `PartnerLink` node. The partner links are updated to reflect the connection between the two instances: A `Link` node is inserted into the `PartnerLink` nodes of the instances, with a connection to the scope link of the other instance.

Note that the `PartnerLink` in the invoked process must be defined in the outermost scope. This is essentially the same issue as with `Exit` (cf. Sec. 5.3.3), namely that binding bigraph contexts

cannot express arbitrary nesting depth: it is impossible to capture the whole process in the rule while also matching an arbitrarily nested partner link withing the process. With hindsight, we could probably have sidestepped this limitation by placing the `CreateInstance` nodes at a fixed location under the `Process` node, thus decoupling them from the partner links.

```
"invoke" :::

   Invoke[partner_link_invoker, partner_link_scope_invoker, oper,
          invar, invar_scope, outvar, outvar_scope, inst_id_invoker]
|| PartnerLink[partner_link_invoker, partner_link_scope_invoker] o <->
|| Variable[invar, invar_scope] o `[]`
|| Running[inst_id_invoker]
|| Process[proc_name][[scope]]
   o (    PartnerLinks
          o (    PartnerLink[partner_link, scope]
                 o (CreateInstance[oper] `|` `[]`)
              `|` scope//[scope1] o `[scope1]`)
      `|` scope//[scope2] o `[scope2]`)

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 3,
     4 |-> 0, 5&[inst_id_invoked1] |--> 2&[scope1],
     6&[inst_id_invoked2] |--> 3&[scope2]]--|>

-//[inst_id_invoked]
o (   GetReply[partner_link_invoker, partner_link_scope_invoker, oper,
               outvar, outvar_scope, inst_id_invoker]
   || PartnerLink[partner_link_invoker, partner_link_scope_invoker]
      o Link[inst_id_invoked]
   || Variable[invar, invar_scope] o `[]`
   || Running[inst_id_invoker]
   || Process[proc_name][[scope]]
      o (    PartnerLinks
             o (    PartnerLink[partner_link, scope]
                    o (CreateInstance[oper] `|` `[]`)
                 `|` scope//[scope1] o `[scope1]`)
         `|` scope//[scope2] o `[scope2]`)
  `|` Instance[proc_name, inst_id_invoked]
      o (    PartnerLinks
             o (    PartnerLink[partner_link, inst_id_invoked]
                    o (    Link[inst_id_invoker]
                       `|` Message[oper] o `[]`
                       `|` ReplyTo[oper, inst_id_invoker])
                 `|` inst_id_invoked//[inst_id_invoked1]
                    o `[inst_id_invoked1]`)
         `|` Invoked[inst_id_invoked]
         `|` inst_id_invoked//[inst_id_invoked2]
             o `[inst_id_invoked2]`));
```

The `receive` rule takes care of activating the instance, by removing a receive node associated to the partner link and the operation (indicated by the link of the `Message`), copying the content of the `Message` in the `PartnerLink` to the proper input variable, and changing the status from a `Invoked` node to a `Running` node.

```
"receive" :::

   Receive[partner_link, partner_link_scope, oper, var, var_scope, inst_id]
|| PartnerLink[partner_link, partner_link_scope]
   o (`[]` `|` Message[oper] o `[]`)
|| Variable[var, var_scope] o `[]`
|| Invoked[inst_id]

  --[0 |-> 0, 1 |-> 1]--|>

   <-> || oper//[]
|| PartnerLink[partner_link, partner_link_scope]
   o `[]`
|| Variable[var, var_scope] o `[]`
|| Running[inst_id];
```

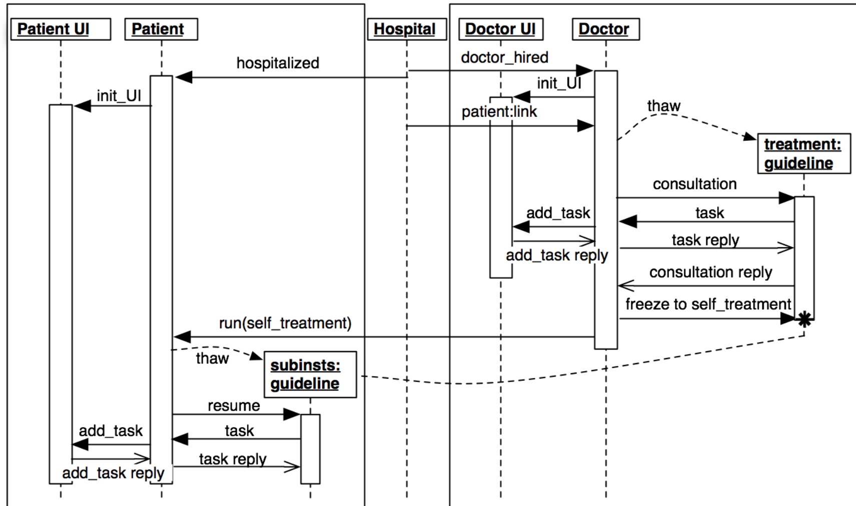


Figure 5.4: Sequence diagram for the pervasive health care scenario.

## 5.4 Motivating HomeBPEL

In this section we motivate the use of HomeBPEL with a simplified example of workflow management for pervasive health care. Each doctor is assumed to run a workflow process, which is initiated when he/she is hired. Every new treatment of a patient causes a new workflow process to be initiated, describing the clinical guideline to be followed for the particular treatment of the patient. In a centralized solution, this process would be running as a separate workflow on the workflow server and only be available when connected to the network. In HomeBPEL business processes can be manipulated as first class values, so we can let the doctor's workflow process execute the treatment process as a *sub-process*. By assuming that the doctor carries a mobile device running its own HomeBPEL engine the treatment process can be executed independently of the network. Moreover, if each patient is equipped with a mobile device running a self-treatment workflow process, the doctor may *delegate* the treatment process (or parts of it) by sending a sub-process to the patient's workflow process.

A sequence diagram illustrating a simple example of this scenario is shown in Fig. 5.4. The two large boxes represent the patient's and the doctor's PDA respectively. The dotted continuation of the "life-line" of the sub-process `guideline` indicates that it is the same process continuing its execution at the patient's PDA. The BPMN diagram in Fig. 5.5 gives a more detailed view of the patient process, with a group of guideline sub-processes indicated in the dashed box in the middle. Fig. 5.6 shows the corresponding HomeBPEL process for the patient. We have left out details related to the data-flow which are not relevant for this example. The initial `receive` on the `hospitalized` operation is used to invoke the patient process, as indicated by the `createInstance` attribute. We have only formalized synchronous communication, so most receive operations are immediately followed by a "dummy" reply. As also shown in the sequence and BPMN diagrams, the following invoke instantiates a local user interface process running on the patient's PDA which we assume takes care of handling the task list of the patient. It is followed by a WS-BPEL `flow`, which contains two while-loops executing in parallel. The first while-loop (corresponding to the right-hand loop in the BPMN diagram) allows for arbitrarily many self-treatment sub-processes to be received and instantiated: The `receive` on the `run` operation waits for a message containing a process and stores it in the input variable `guideline`. The following activity `thaw` is part of the new features introduced in HomeBPEL and it is used to create an instance of a process stored in a variable (in the example named `guideline`) and execute it as a sub-instance within the scope of the corresponding `subLink` (in the example named `subinsts`) of the current running instance. The

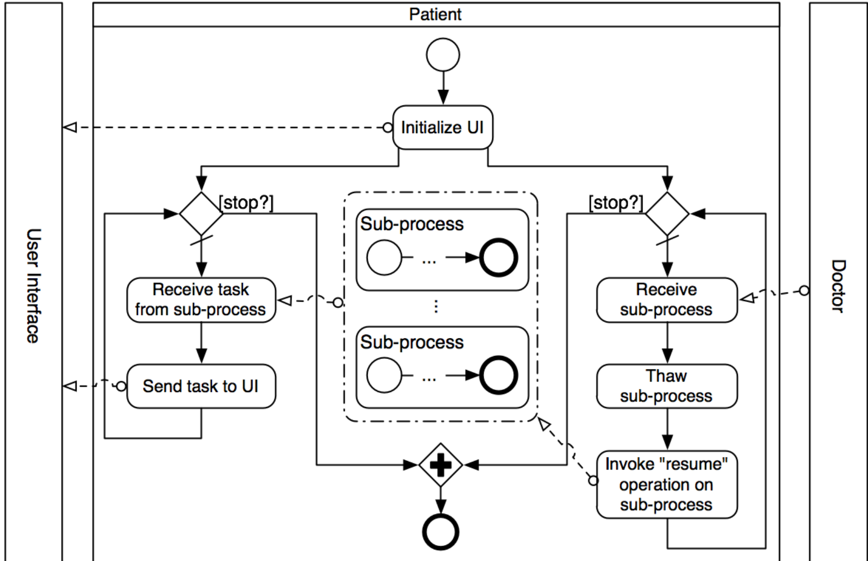



Figure 5.5: BPMN diagram of the patient workflow process.

second while-loop (corresponding to the left-hand loop in the BPMN diagram) forwards messages received from the guideline sub-processes by the HomeBPEL `receiveSub` activity to the user interface, and in turn forwards the answer back to the sub-process by the HomeBPEL `replySub` activity.

The doctor's workflow process shown in Fig. 5.7 also invokes a user interface process, and contains an identical loop for forwarding messages from treatment workflows to the user interface process (which we have omitted from the example code to save space). However, different from the patient workflow, the first step of the main loop of the doctor workflow is to receive a link (on the `patient` operation) which is then dynamically assigned to the `patient` partner link by the `copy` operation. Thereby the doctor workflow process can be dynamically linked to different patient workflow processes during its lifetime. The following `thaw` activity instantiates a treatment guideline as a sub-process from the variable named `guideline`. Fig. 5.8 shows an outline of the treatment process consisting of two phases: A *consultation phase* invoked explicitly by the doctor and carried out within the doctor's workflow, and a *self-treatment* phase carried out within the patient's workflow. To initiate the first part of the treatment, the operation `consultation` is invoked from the doctor workflow by the action `invokeSub`. The reply of this operation signals that the consultation is finished, and the treatment process is ready to be frozen (by the `freeze` action) and sent to the patient's workflow process.

Note that we have not specified the specific tasks for each phase in the treatment, which in general could be part of an arbitrarily complex workflow. However, we have illustrated how tasks in each phase can be scheduled at the user interface of the *current* super workflow by invoking the `task` operation by the `invokeSup` action. This shows how *context-dependent communication* is elegantly facilitated in HomeBPEL. One could easily imagine that the treatment processes could also access local information, e.g. special expertise of the doctor or relevant characteristics of the patient.

We claim that the use of higher-order processes in this example is much more flexible than a workflow simply based on a fixed *configuration* of a self-treatment process at the patient engine: Configuration is limited to a pre-defined set of parameters — in contrast to the treatment template that can be an arbitrary process which could be received by the doctor process from a central server of clinical guidelines.

```
<process name="patient">
  <partnerLinks>
    <partnerLink name="patient_client" />
    <partnerLink name="task_list_UI" />
  </partnerLinks>
  <subLinks>
    <subLink name="subinsts" />
  </subLinks>
  <variables>
    <variable name="guideline" />
    <variable name="task" />
    <variable name="reply" />
    ...
  </variables>
  <sequence>
    <receive partnerLink="patient_client" operation="hospitalized"
             createInstance="yes" ... /><reply operation="hospitalized" ... />
    <invoke  partnerLink="task_list_UI" operation="init_UI" ... />
    <flow>
      <!-- Thaw-loop: Continually receives and executes sub-instances -->
      <while>
        <condition>...</condition>
        <sequence>
          <receive   partnerLink="patient_client" operation="run"
                     variable="guideline" /><reply operation="run" ... />
          <thaw      subLink="subinsts" variable="guideline" />
          <invokeSub subLink="subinsts" operation="resume" ... />
        </sequence>
      </while>
      <!-- UI-loop: Continually receives tasks from sub-instances and
           pass them on to the UI service -->
      <while>
        <condition>...</condition>
        <sequence>
          <receiveSub subLink="subinsts" operation="task" variable="task" />
          <invoke     partnerLink="task_list_UI" operation="add_task"
                      inputVariable="task" outputVariable="reply" />
          <replySub   subLink="subinsts" operation="task" variable="reply" />
        </sequence>
      </while>
    </flow>
  </sequence>
</process>
```

Figure 5.6: Patient workflow process.

The above example is of course still highly simplified. One would most likely want more control over the behavior of sub-processes, i.e. to disallow malicious processes from entering ones mobile device, to only allow processes from known, trusted sources, etc. It would also be relevant to allow reflection, combination and adaption of sub-processes on the fly. In the health care scenario, this could be used to avoid repeating a blood pressure measurement in two concurrent treatments, or more important, that the same pill is not commanded to be taken twice. We expect to address these questions in future work. A necessary first step is a formal semantics of the execution which will be provided in the following sections.

## 5.5 Formalizing HomeBPEL

In this section we present the formalization of HomeBPEL in the BPL Tool. Using basically the same approach as in Sec. 5.3 we first present the static representation in Sec. 5.5.1 and then the reaction rules in Sec. 5.5.3, with a brief discourse on the semantics of sub-links in Sec. 5.5.2.

```
<process name="doctor">
  <partnerLinks>
    <partnerLink name="hospital" />
    <partnerLink name="patient" />
    <partnerLink name="task_list_UI" />
  </partnerLinks>
  <subLinks><subLink name="treatment" /></subLinks>
  <variables>
    <variable name="guideline"><process name="guideline">...</process></variable>
    <variable name="link" /><variable name="self_treatment" /> ...
  </variables>
  <sequence>
    <receive partnerLink="hospital" operation="doctor_hired"
             createInstance="yes" ... /><reply operation="doctor_hired" ... />
    <invoke  partnerLink="task_list_UI" operation="init_UI" ... />
    <flow>
      <while>
        <condition>...</condition>
        <sequence>
          <receive   partnerLink="hospital" operation="patient"
                     variable="link" /><reply operation="patient" ... />
          <assign><copy>
              <from variable="link" /><to partnerLink="patient" />
          </copy></assign>
          <thaw      subLink="treatment" variable="guideline" />
          <invokeSub subLink="treatment" operation="consultation" ... />
          <freeze    subLink="treatment" variable="self_treatment" />
          <invoke    partnerLink="patient" operation="run"
                     inputVariable="self_treatment" ... />
        </sequence>
      </while>
      <!-- while-loop forwarding tasks to the local user interface -->
    </flow>
  </sequence>
</process>
```

Figure 5.7: Doctor workflow process.

### 5.5.1 The Static Representation

The formalization of HomeBPEL as a binding bigraphical reactive system in the BPL Tool is given by a signature, determining the allowed controls and the ports for each type of control, and a set of reaction rules, determining the run-time semantics. As described previously, we utilize the extensibility of bigraphs to extend and adapt the previous formalization of WS-BPEL given in Sec. 5.3.

Table 5.5 shows the signature of HomeBPEL. The controls listed in the upper part of the signature correspond directly to elements in WS-BPEL, while the controls listed in the lower part are introduced to facilitate the formalization of the execution semantics. The underlined controls are the controls introduced (or adapted) in order to support higher order mobile embedded subprocesses.

```
<process name="guideline">
  ...
  <sequence>
    <!-- Doctor initializes treatment -->
    <receiveSup operation="consultation" ... />
    <!-- Instruct doctor on how to perform consultation -->
    <invokeSup  operation="task" ... />
    <replySup   operation="consultation" ... />
    <!-- Ready to be moved to patient -->
    <receiveSup operation="resume" ... /><replySup operation="resume" ... />
    <!-- Instruct patient how to perform self-treatment -->
    <invokeSup  operation="task" ... />
  </sequence>
</process>
```

Figure 5.8: Treatment guideline process.

| *Active controls* | *Passive controls* | *Atomic controls* | |
|---|---|---|---|
| `PartnerLinks` | `Process =: 1 -> 1` | `To -: 2` | |
| `Variables` | `Scope =: 1 -> 1` | `From -: 2` | |
| `If -: 1` | `Variable -: 2` | `ToPLink -: 2` | `VariableRef -: 3` |
| `Condition` | `While -: 1` | `FromPLink -: 2` | `ReplyTo -: 2` |
| `Sequence -: 1` | `Then` | `Invoke -: 8` | `Link -: 1` |
| `Flow -: 1` | `Else` | `Receive -: 6` | `CreateInstance -: 1` |
| | `Assign -: 1` | `Reply -: 6` | `SubTransition -: 1` |
| | `Copy` | `Exit -: 1` | `InvokeSub -: 8` |
| | `PartnerLink -: 2` | | `InvokeSup -: 6` |
| | | | `ReceiveSub -: 6` |
| | | | `ReceiveSup -: 4` |
| | | `True` | `ReplySub -: 6` |
| | | `False` | `ReplySup -: 4` |
| | | `GetReply -: 6` | `GetReplySub -: 7` |
| `Instance -: 3` | | `Running -: 3` | `GetReplySup -: 4` |
| `ActiveScope -: 2` | | `Invoked -: 3` | `Freeze -: 5` |
| `TopInstance` | `Next` | `Stopped -: 3` | `FreezingSub -: 5` |
| `Instances` | `Message -: 1` | `Freezing -: 3` | `Thaw -: 5` |
| `SubLinks` | `SubLink -: 2` | `TopRunning -: 1` | `FrozenSupLink -: 2` |

Table 5.5: Signature for HomeBPEL

Not all bigraphs of the given signature will correspond to valid processes and instances. The grammar in Table 5.6 shows the valid nesting of elements.[6] (For brevity we have abstracted away from the ports of nodes in the grammar. The arity of each control is provided in the signature, where e.g. `Process =: 1 -> 1` means that the `Process` control has one normal port and one binding port). We let $i$ range over the set $\{0, 1\}$ which we use to index some of the productions to keep the presentation succinct. We write *prod*? for indicating that the terminal or non-terminal is optional and we write `Link`$*$ to denote that there can be 0 or more `Link` terminals. Currently the formalization only supports one type of expressions, namely variable references. But one can easily extend the semantics to more expression types (e.g. XPath expressions), simply by adding rules describing how to evaluate them — without having to alter the current rules. Similarly, values (i.e. *value*) are currently restricted to be either the constants `True` and `False`, processes (higher-order values), or the content of a `PartnerLink` (akin to name passing in the $\pi$-calculus). One could exploit the correspondence between XML and bigraphs to represent any kind of XML-data.

As mentioned in the introduction, the key idea of the formalization is that a process is represented by a bigraph very similar to the XML syntax for WS-BPEL processes. Also, an active *instance* is represented almost exactly as the process, except it has an outermost node labeled by an `Instance` control. Instances keep the current content of variables inside the variable node, and are executed as in process calculi by rewriting the bigraph according to the set of reaction rules to be described in the following section.

As an example, the process `patient` from Sec. 5.4 is represented as a binding bigraph in the BPL Tool as shown in Fig. 5.9(a) – (b). (To shorten the example we have not fully specified the task loop. The full representation is available at the BPL Tool web page). Note that the compositionality of bigraphs allow us to separate the process into several parts. Fig. 5.10 shows the graphical representation provided by the BPL Tool of the `thaw_loop` bigraph in Fig. 5.9(a). To keep the figure clear we have abstracted away from the identity of the patient.

Looking at the graphical representation, it should be clear that the place graph corresponds closely to the nesting of elements in the XML syntax, the ports of controls correspond to attributes, and the link graph corresponds to shared values of attributes. However, already for the formalization of the subset of WS-BPEL given in Sec. 5.3 we needed to introduce some additional structure. For instance, a `Next` control is embedded in `Sequence` controls to cope with the fact that children nodes in bigraph place graphs are unordered while children nodes in XML are ordered (which is exploited in the sequence construct of WS-BPEL). To facilitate the definition of reaction

[6] This restriction can be represented in the theory of bigraphs using the notion of *sorting*.

<table>
<tr><td>system</td><td>::=</td><td>procs ‘|‘ state</td></tr>
<tr><td>procs</td><td>::=</td><td>proc ‘|‘ ... ‘|‘ proc</td></tr>
<tr><td>state</td><td>::=</td><td>topinst ‘|‘ ... ‘|‘ topinst</td></tr>
<tr><td>proc</td><td>::=</td><td>Process($scopecontent_0$)</td></tr>
<tr><td>partnerlinks</td><td>::=</td><td>PartnerLinks(partnerlink ‘|‘ ... ‘|‘ partnerlink)</td></tr>
<tr><td>partnerlink</td><td>::=</td><td>PartnerLink(partnerlinkcontent)</td></tr>
<tr><td>partnerlinkcontent</td><td>::=</td><td>CreateInstance? ‘|‘ link?</td></tr>
<tr><td>link</td><td>::=</td><td>Link ‘|‘ message?</td></tr>
<tr><td>message</td><td>::=</td><td>Message(value)</td></tr>
<tr><td>sublinks</td><td>::=</td><td>SubLinks(SubLink(Link*) ‘|‘ ... ‘|‘ SubLink(Link*))</td></tr>
<tr><td>vars</td><td>::=</td><td>Variables(Variable(value) ‘|‘ ... ‘|‘ Variable(value))</td></tr>
<tr><td>topinst</td><td>::=</td><td>TopInstance(inst ‘|‘ topinststatus)</td></tr>
<tr><td>topinststatus</td><td>::=</td><td>TopRunning | SubTransition</td></tr>
<tr><td>insts</td><td>::=</td><td>Instances(inst ‘|‘ ... ‘|‘ inst)</td></tr>
<tr><td>inst</td><td>::=</td><td>Instance(status ‘|‘ $scopecontent_1$)</td></tr>
<tr><td>status</td><td>::=</td><td>Invoked | Running | Freezing | Stopped</td></tr>
<tr><td>$act_i$</td><td>::=</td><td>$scope_i$ | $seq_i$ | $flow_i$ | $while_i$ | $if_i$ | assign | Invoke</td></tr>
<tr><td></td><td>|</td><td>Receive | Reply | GetReply | Exit | InvokeSub | InvokeSup</td></tr>
<tr><td></td><td>|</td><td>ReceiveSub | ReceiveSup | ReplySub | ReplySup | Thaw</td></tr>
<tr><td></td><td>|</td><td>GetReplySub | GetReplySup | Freeze | FreezingSub</td></tr>
<tr><td>$scope_0$</td><td>::=</td><td>Scope($scopecontent_0$)</td></tr>
<tr><td>$scope_1$</td><td>::=</td><td>ActiveScope($scopecontent_1$) | Scope($scopecontent_0$)</td></tr>
<tr><td>$scopecontent_i$</td><td>::=</td><td>partnerlinks ‘|‘ sublinks ‘|‘ insts ‘|‘ vars ‘|‘ $act_i$?</td></tr>
<tr><td>$seq_i$</td><td>::=</td><td>Sequence($act_i$? ‘|‘ Next($act_i$?))</td></tr>
<tr><td>$flow_i$</td><td>::=</td><td>Flow($act_i$? ‘|‘ ... ‘|‘ $act_i$?)</td></tr>
<tr><td>$while_i$</td><td>::=</td><td>While(Condition(expr) ‘|‘ $act_i$?)</td></tr>
<tr><td>$if_i$</td><td>::=</td><td>If(Condition(expr) ‘|‘ Then($act_i$?) ‘|‘ Else($act_i$?))</td></tr>
<tr><td>assign</td><td>::=</td><td>Assign(Copy(from ‘|‘ to))</td></tr>
<tr><td>from</td><td>::=</td><td>From | FromPLink</td></tr>
<tr><td>to</td><td>::=</td><td>To | ToPLink</td></tr>
<tr><td>value</td><td>::=</td><td>True | False | proc | partnerlinkcontent</td></tr>
<tr><td>expr</td><td>::=</td><td>True | False | VariableRef</td></tr>
</table>

Table 5.6: Grammar for HomeBPEL

rules in the semantics we needed to add links representing instance and scope identities. More intricately, we also needed to introduce a node within each instance with a *status* control being either `Invoked`, `Running`, or `Stopped`. Partly, this is needed because the semantics of `Invoke` and `Exit` activities requires two consecutive reactions. The extension with mobile sub-instances made it necessary to add an additional status control, `Freezing`, since freezing an instance into a process in a variable cannot be done atomically either. Also, we needed at top level to introduce a status control indicating if the top instance or any of its (arbitrarily nested) sub-instances are allowed to perform normal activities (by the control `TopRunning`) or if one of them are performing a sub-transition (control `SubTransition`) as part of a non-atomic activity. These aspects could most likely have been dealt with more elegantly if bigraphical reactive systems had a notion of priority on the reaction rules. We leave it for future work to study this.

### 5.5.2 Sub-links

In this section we take a closer look at the semantics of sub-links and the associated operations, using the patient process in Fig. 5.6 as example. Note how each iteration of the thaw loop thaws a new sub-instance which is bound to the sub-link `subinsts` and then invokes the operation `resume` on `subinsts`. What happens when a sub-instance is already bound to the `subinsts` sub-link? Is this an error or should it be allowed, and in that case which sub-instance(s) should be bound to the sub-link after the execution of `thaw`?

As one of the goals of the CosmoBiz project is to integrate process management into the process language itself, it seems reasonable to provide easy management of collections of processes at the language level. One simple way to achieve this feature, and the one we have chosen, is to allow multiple sub-instances to be bound to the same sub-link simultaneusly. This choice abstracts the

```
val thaw_loop_body =
    Sequence[patient_id] o (
      Receive[patient_client, patient_id, run, x, patient_id, patient_id]
`|` Next o Sequence[patient_id] o (
      Thaw[subinsts, patient_id, x, patient_id, patient_id]
`|` Next o
      Reply[patient_client, patient_id, run, y, patient_id, patient_id]));

val thaw_loop =
While[patient_id] o (    Condition o VariableRef[y, patient_id, patient_id]
                     `|` thaw_loop_body);

val task_loop =
While[patient_id] o ( ...... );

val patient_body = Flow[patient_id] o (thaw_loop `|` task_loop);
```

(a) `patient_body`

```
val patient_process =
Process[patient][[patient_id]] o (
    PartnerLinks o (
        PartnerLink[patient_client, patient_id] o CreateInstance[start]
    `|` PartnerLink[task_list_UI, patient_id]  o <->)
`|` SubLinks o SubLink[subinsts, patient_id] o <->
`|` Variables o (
        Variable[x, patient_id] o <->
    `|` Variable[y, patient_id] o True)
`|` Instances o <->
`|` Sequence[patient_id] o (
      Receive[patient_client, patient_id, start, x, patient_id, patient_id]
`|` Next o Sequence[patient_id] o (
      Reply[patient_client, patient_id, start, y, patient_id, patient_id]
`|` Next o patient_body)));
```

(b) `patient_process`

Figure 5.9: BPL Tool representation of the patient process.

implementation of collections of processes away from the programmer, building the concept of a process collection into the semantics of the activies which use sub-links: `thaw`, `freeze`, `invokeSub`, and `receiveSub`. The latter three activities are only intended to affect one sub-instance, but several of the sub-instances connected to the sub-link might be a suitable target for the activity. In this case, we choose an arbitrary process, based on the assumption that if the programmer wanted to distinguish two processes, she would place them in different collections. This might not hold true, and will be the subject of future work. One could imagine, for instance, that in the case of `freeze`, that the programmer wants to freeze a particular sub-instance or a sub-instance which is ready to be frozen.

### 5.5.3 Reaction Rules

In this section we present the reaction rules used in the formalization of HomeBPEL, focusing on the new rules for freezing and thawing and for communication between parent and child processes. The full set of reaction rules (in BPL Tool syntax) is available via the on-line tool[7].

#### Changes to the Representation

We have extended the representation in a smaller degree in order to facilitate the representation of higher-order primitives in the formalization. An `Instance` node now have an additional port

[7]See `http://tiger.itu.dk:8080/bplweb/index/20`

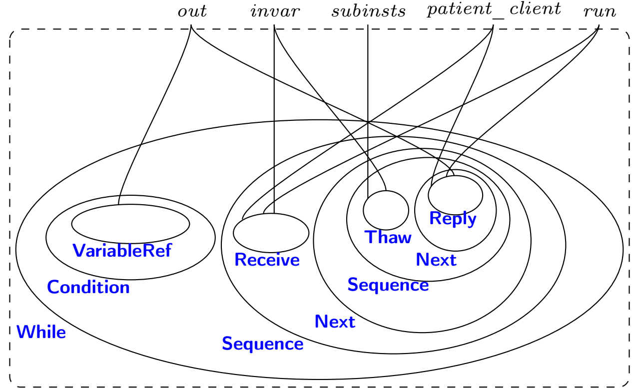



Figure 5.10: BPL Tool visualization of thaw_loop.

which should be connected state node of the parent instance if a parent instance exists. We have introduced a node Instances to group together sub-instances of an instance, similar to the effect of the Variables node. As mentioned above we also introduce a status node in the top-level instance to track whether the top instance or any of its (arbitrarily nested) sub-instances are allowed to perform normal activities (by the control TopRunning) or if one of them are performing a sub-transition (control SubTransition) as part of a non-atomic activity. Also as mentioned above we add the status Freezing. As a technicality we also introduce nodes of control TopInstance to encapsulate top-level instances together with their associated top-level status node.

### Augmenting the Existing Rules

Most of the reaction rules of the formalization remains unchanged from Sec. 5.3, except that the rules also need to make sure that the status of the top-level instance is TopRunning. However for the rule scope completed, the rule responsible for removing scopes that have been executed, we now also need to make sure that there are no running sub-instance inside the scope before removing the scope. The case is similar for the rule inst completed just for instances instead of scopes. We also need one additional rule to remove the new TopInstance nodes when removing a top-level instance. The added TopInstance node also add a bit to the complexity of the invoke rule.

```
"top instance completed" :::

TopInstance o (-//[inst_id_top] o TopRunning[inst_id_top])
  ----|>
<->;
```

We have added some new reaction rules to the formalization to implement the added primitives for higher-order processes. Below we present the new reaction rules.

### Communication Between Parent and Child

The rule invoke sub takes care of an instance invoking a method in a subinstance. The parent instance performs the InvokeSub activity in parallel with the ReceiveSup of the subinstance. Both instances are required to be running as well as the top-level instance. The result is that the content from the variable invar is copied to variable var. Besides these changes the rule resembles the rule invoke instance (described below) which is responsible for communication between two top-level instances.

```
"invoke sub" :::
```

```
   InvokeSub[sub_link, sub_link_scope, oper, invar, invar_scope,
             outvar, outvar_scope, inst_id_sup]
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[invar, invar_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| ReceiveSup[oper, var, var_scope, inst_id_sub]
|| Variable[var, var_scope]
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 1]--|>

   GetReplySub[sub_link, sub_link_scope, inst_id_sub, oper,
               outvar, outvar_scope, inst_id_sup]
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[invar, invar_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| <->
|| Variable[var, var_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| TopRunning[inst_id_top];
```

In the rule `reply sup` the `ReplySup` activity inside an instance can synchronize together with a `GetReplySub` activity inside the parent instance, thereby copying the content from variable `var` to variable `outvar`.

```
"reply sup" :::

   ReplySup[oper, var, var_scope, inst_id_sub]
|| Variable[var, var_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| GetReplySub[sub_link, sub_link_scope, inst_id_sub, oper,
               outvar, outvar_scope, inst_id_sup]
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[outvar, outvar_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 0]--|>

   <-> || oper//[]
|| Variable[var, var_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| <->
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[outvar, outvar_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top];
```

Rule `invoke sup` is similar to the rule `invoke sub`, except that it is the subinstance which invokes a method in the parent.

```
"invoke sup" :::

   InvokeSup[oper, invar, invar_scope, outvar, outvar_scope, inst_id_sub]
|| Variable[invar, invar_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| ReceiveSub[sub_link, sub_link_scope, oper, var, var_scope, inst_id_sup]
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[var, var_scope]
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 0]--|>

   GetReplySup[oper, outvar, outvar_scope, inst_id_sub]
|| Variable[invar, invar_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| <->
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
```

```
|| Variable[var, var_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top];
```

The rule `reply sub` is similar to the rule `reply sup`, except for the direction of the communication.

```
"reply sub" :::

   ReplySub[sub_link, sub_link_scope, oper, var, var_scope, inst_id_sup]
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[var, var_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| GetReplySup[oper, outvar, outvar_scope, inst_id_sub]
|| Variable[outvar, outvar_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 1]--|>

   <-> || oper//[]
|| SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] ‘|‘ ‘[]‘)
|| Variable[var, var_scope] o ‘[]‘
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| <->
|| Variable[outvar, outvar_scope] o ‘[]‘
|| Running[inst_id_sub, active_scopes_sub, inst_id_top]
|| TopRunning[inst_id_top];
```

### Freezing Processes

Freezing a sub-instance requires several transitions, initiated by a `Freeze` activity. The `Freeze` activity references a running subinstance through its `SubLinks` and changes the status of the instance from `Running` to `Freezing` (thus ensuring that the subinstance will not execute anymore), at the same time the `Freeze` activity is replaced by a `FreezingSub` activity, and the top-level status is changed from `TopRunning` to `SubTransition` to indicate that we have started a multistep reaction.

```
"freeze sub" :::

   Freeze[sub_link, sub_link_scope, var, var_scope, inst_id_sup]
|| (    SubLinks o (    SubLink[sub_link, sub_link_scope]
                        o (Link[inst_id_sub] ‘|‘ ‘[]‘)
                    ‘|‘ ‘[]‘)
    ‘|‘ Instances
        o (    Instance[sub_name, inst_id_sub, active_scopes_sup]
               o (Running[inst_id_sub, active_scopes_sub, inst_id_top] ‘|‘ ‘[]‘)
           ‘|‘ ‘[]‘))
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 3]--|>

   FreezingSub[sub_link, sub_link_scope, var, var_scope, inst_id_sup]
|| (    SubLinks o (    SubLink[sub_link, sub_link_scope]
                        o (Link[inst_id_sub] ‘|‘ ‘[]‘)
                    ‘|‘ ‘[]‘)
    ‘|‘ Instances
        o (    Instance[sub_name, inst_id_sub, active_scopes_sup]
               o (Freezing[inst_id_sub, active_scopes_sub, inst_id_top] ‘|‘ ‘[]‘)
           ‘|‘ ‘[]‘))
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| SubTransition[inst_id_top];
```

Inside a freezing subinstance an active scope can be frozen when all nested scopes and subinstances have been frozen. This is ensured by requiring that the content of the scope does not

refer to the active-scopes link of the enclosing sub-instance. We then change the `ActiveScope` to a `Scope` and bind the free edge denoted by "scope".

```
"freeze scope" :::

-//[active_scopes]
o (   Freezing[inst_id, active_scopes, inst_id_top]
   || -//[scope] o (ActiveScope[active_scopes, scope] o `[scope]`)
   || `[active_scopes]`)

  --[0 |-> 0, 1 |-> 1]--|>

-//[active_scopes]
o (   Freezing[inst_id, active_scopes, inst_id_top]
   || Scope[inst_id][[scope]] o `[scope]`
   || `[active_scopes]`);
```

Sub-instances of a sub-instance which is being frozen, are frozen by propagating the freezing state, which again allows its scopes and subinstances to be frozen. This is done by changing the status of the nested sub-instance from `Running` to `Freezing`.

```
"freeze sub instance" :::

   Freezing[inst_id, active_scopes, inst_id_top]
|| Instance[sub_name, inst_id_sub, active_scopes]
   o (    Running[inst_id_sub, active_scopes_sub, inst_id_top]
      `|` `[]`)

  --[0 |-> 0]--|>

   Freezing[inst_id, active_scopes, inst_id_top]
|| Instance[sub_name, inst_id_sub, active_scopes]
   o (    Freezing[inst_id_sub, active_scopes_sub, inst_id_top]
      `|` `[]`);
```

When all those are frozen, ie. the "active_scopes" link of the sub-sub-instance is only connected to the state node, the sub-sub-instance is frozen (remaining at the same location) and a `FrozenSupLink` is inserted in the frozen instance to remember which `SubLink` it was connected to.

```
"freeze sub instance2" :::

-//[inst_id_sub]
o (   Freezing[inst_id_sup, active_scopes, inst_id_top]
   || (    SubLinks o (    SubLink[sub_link, sub_link_scope]
                           o (Link[inst_id_sub] `|` `[]`)
                       `|` `[]`)
       `|` Instances
           o (    Instance[sub_name, inst_id_sub, active_scopes]
                  o (    -//[active_scopes_sub]
                         o Freezing[inst_id_sub, active_scopes_sub, inst_id_top]
                     `|` `[inst_id_sub]`)
              `|` `[]`)))

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 3]--|>

   Freezing[inst_id_sup, active_scopes, inst_id_top]
|| (    SubLinks o (SubLink[sub_link, sub_link_scope] o `[]` `|` `[]`)
    `|` Instances
        o (    Process[sub_name][[inst_id_sub]]
               o (    FrozenSupLink[sub_link, sub_link_scope]
                  `|` `[inst_id_sub]`)
           `|` `[]`));
```

When no more sub-instances and scopes are connected to the "active_scopes" link of the sub-instance being frozen, it can itself be frozen and placed into the proper variable denoted by `var`. To indicate that the multistep reaction is completed we change the top-level status from `SubTransition` and back to `TopRunning`.

```
"freeze complete" :::

-//[inst_id_sub]
o (   FreezingSub[sub_link, sub_link_scope, var, var_scope, inst_id_sup]
   || Variable[var, var_scope] o `[]`
   || SubLink[sub_link, sub_link_scope] o (Link[inst_id_sub] `|` `[]`)
   || Running[inst_id_sup, active_scopes_sup, inst_id_top]
   || SubTransition[inst_id_top]
   || Instance[sub_name, inst_id_sub, active_scopes_sup]
      o (    -//[active_scopes_sub]
             o Freezing[inst_id_sub, active_scopes_sub, inst_id_top]
         `|` `[inst_id_sub]`))

  --[0 |-> 2, 1 |-> 1]--|>

   <->
|| Variable[var, var_scope]
   o Process[sub_name][[inst_id_sub]] o `[inst_id_sub]`
|| SubLink[sub_link, sub_link_scope] o `[]`
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top]
|| <->;
```

### Thawing Processes

Using the of new Thaw activity one can thaw a sub-process stored in a variable and instantiating it as a sub-instance. The Thaw activity in the redex refers via its third port to the process inside the variable var. In the reactum the Thaw activity has been removed (indicating it has been executed) and a new running sub-instance has been inserted within the Instances control. The last part (4&[inst_id_sub] |-> 0&[sub_scope]]) of the instantiation map on the arrow from the redex to the reactum ensures that the process body (contained in hole 0 in the redex) is copied and used as body of the new sub-instance (hole 4 in the reactum). It also ensures that the local bound link sub_scope of the process body is renamed to inst_id_sub in the new copy. Note also that we insert the status node Running in the new sub-instance. Finally, the rule also insert a Link control within the SubLinks control. The Link control points to the new sub-instance via its link inst_id_sub.

```
"thaw sub" :::

   Thaw[sub_link, sub_link_scope, var, var_scope, inst_id_sup]
|| Variable[var, var_scope]
   o Process[sub_name][[sub_scope]] o `[sub_scope]`
|| (    SubLinks o (SubLink[sub_link, sub_link_scope] o `[]` `|` `[]`)
    `|` Instances o `[]`)
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 3,
     4&[inst_id_sub] |--> 0&[sub_scope]]--|>

   <->
|| Variable[var, var_scope]
   o Process[sub_name][[sub_scope]] o `[sub_scope]`
|| -//[inst_id_sub]
   o (    SubLinks o (    SubLink[sub_link, sub_link_scope]
                          o (Link[inst_id_sub] `|` `[]`)
                      `|` `[]`)
      `|` Instances
          o (    `[]`
             `|` Instance[sub_name, inst_id_sub, active_scopes_sup]
                 o (    -//[active_scopes_sub]
                        o Running[inst_id_sub, active_scopes_sub, inst_id_top]
                    `|` `[inst_id_sub]`)))
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top];
```

In general, when we thaw a process it may itself contain frozen sub-instances frozen “in place”, i.e. within the `Instances` control. An additional reaction rule (`thaw sub instance`) is thus included for thawing frozen sub-instances. The rule replaces the `Process` node with a `Instance` node and restores the `SubLinks` using the information represented by the `FrozenSupLink` node. Finally the rule sets the status of the instance to `Running`. Note that this rule is the inverse of rule `freeze sub instance2`.

```
"thaw sub instance" :::

   (    SubLinks o (SubLink[sub_link, sub_link_scope] o ‘[]‘ ‘|‘ ‘[]‘)
    ‘|‘ Instances
        o (    Process[sub_name][[inst_id_sub]]
               o (    FrozenSupLink[sub_link, sub_link_scope]
                  ‘|‘ ‘[inst_id_sub]‘)
           ‘|‘ ‘[]‘))
|| Running[inst_id_sup, active_scopes_sup, inst_id_top]
|| TopRunning[inst_id_top]

  --[0 |-> 0, 1 |-> 1, 2 |-> 2, 3 |-> 3]--|>

-//[inst_id_sub]
o (   (    SubLinks o (    SubLink[sub_link, sub_link_scope]
                           o (Link[inst_id_sub] ‘|‘ ‘[]‘)
                       ‘|‘ ‘[]‘)
       ‘|‘ Instances
           o (    Instance[sub_name, inst_id_sub, active_scopes_sup]
                  o (    -//[active_scopes_sub]
                         o Running[inst_id_sub, active_scopes_sub, inst_id_top]
                     ‘|‘ ‘[inst_id_sub]‘)
              ‘|‘ ‘[]‘))
   || Running[inst_id_sup, active_scopes_sup, inst_id_top]
   || TopRunning[inst_id_top]);
```

## 5.6 Conclusion and Future Work

We have formalized a subset of WS-BPEL as a binding bigraphical reactive system. As in our previous work described in [22] we have utilized the close correspondence between bigraphs and XML to provide a formalization close to the original WS-BPEL syntax and yet stays within the existing format for binding bigraphs [26].

Several new non-trivial aspects of WS-BPEL have been formalized compared to [22], including support for nested scopes, termination (exit), and dynamic assignment and communication of partner links. As a technical, but important point, we avoided higher-order reaction rules as used in [22]. This means that the general theory, techniques and tools developed for standard, binding bigraphs remain applicable to our formalization. In particular, we have described how the formalization can be implemented and explored within the BPL Tool [3] developed in the Bigraphical Programming Languages project at the IT University of Copenhagen. The tool allows compositional definition of binding bigraphs and reaction rules, as well as graphical visualization and interactive simulation of the execution of binding bigraphical reactive systems based on the formal inference of rule matching described in [1, 2].

We have utilised the extensibility of bigraphical reactive systems to extend the formalization of WS-BPEL to a formalization of a higher-order WS-BPEL-like language called HomeBPEL. The extensibility of bigraphical reactive systems enables us to directly reuse most of the existing formalization. In HomeBPEL processes are first-class values that can be stored in variables, passed as messages, and activated as embedded sub-instances. We have formalized HomeBPEL in the BPL Tool. We have motivated HomeBPEL by an example of pervasive health care where treatment guidelines are dynamically deployed as sub processes that may be delegated dynamically to other workflow engines and in particular stay available for disconnected operation on mobile devices. The added features of HomeBPEL allow us — among other — to define business processes for business

process management within HomeBPEL as opposed to relying on meta-level tools for deployment and process administration.

**Future Work.** It is important to stress that we do not in this paper claim to give a feature complete formalization of WS-BPEL, as e.g. provided in [31]. We leave as future work to compare our formalization to the work in [31] and to provide more complete semantics and simulation of WS-BPEL. To do this, it is likely to be helpful if the BPL Tool was extended with a notion of high-level bigraphs allowing e.g. built-in XML and/or ML datatypes and transformations in the reaction rules, analogous to the built-in ML datatypes and functions found in Coloured Petri Nets and the CPN Tool. This was already partly explored in the ReactiveXML implementation of pure bigraphical reactive systems described in [22].

A notion of *prioritized* reactions could be interesting to explore as an alternative to the explicit encoding of transitions and sub-transitions used in the present paper. It will also be interesting to investigate the use of the general theory of bisimulation congruences available for bigraphical reactive systems in the setting of WS-BPEL.

An interesting path for future research into HomeBPEL will be to examine different primitives for management and manipulation of processes. Currently, we can copy and discard processes (by copying and overwriting the content of variables) and we can — to some extend — combine processes, but we are currently examining more expressive primitives, such as sub-process reflection and general manipulation, e.g. editing or joining of frozen sub-processes. This relates to the work on Higher-Order (Petri) Nets and applications to workflow studied in [25].

Future work will also include the study of type systems, e.g. relations to the work on formalizations of WSDL types, contracts and session types [4, 9, 28]. The addition of mobile embedded sub-instances also opens for a study of type systems that can guarantee safe process mobility and manipulation. We plan to explore this in the CosmoBiz research project. In particular, we plan to examine the approaches done in Boxed Ambients [17] and in the higher-order $\pi$-calculus [34] on the safe integration of higher-order mobility and sessions.

Another relevant direction of work is a detailed and complete study of the expressiveness of HomeBPEL in relation to workflow patterns (e.g. [37]). We will also study the language primitives and expressiveness in relation to process calculi for mobility such as Ambients, Seal and Homer. In particular, we expect to examine a notion of subjective mobility as in Safe Ambients [30] by introducing a co-freeze activity to be carried out by the sub-instance, allowing it to decide whether (and when) it can be frozen.

**Acknowledgements.** Many thanks to the anonymous referees for their suggestions and comments from which this paper has benefited greatly.

## 5.7 Bibliography

[1] Lars Birkedal, Troels Christoffer Damgaard, Arne John Glenstrup, and Robin Milner. Matching of bigraphs. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *Proceedings of the Graph Transformation for Verification and Concurrency workshop (GT-VC'06)*, volume 175 of *Electronic Notes in Theoretical Computer Science*, pages 3–19. Elsevier, 2006.

[2] Lars Birkedal, Troels Christoffer Damgaard, Arne John Glenstrup, and Robin Milner. An inductive characterisation of matching in binding bigraphs. *to appear*, 2008.

[3] The Bigraphical Programming Languages Group. The BPL Tool. `http://www.itu.dk/research/pls/wiki/index.php/BPL_Tool`, 2007.

[4] Mario Bravetti and Gianluigi Zavattaro. Contract based multi-party service composition. In Farhad Arbab and Marjan Sirjani, editors, *Proceedings of the IPM International Symposium on Fundamentals of Software Engineering (FSEN'07)*, volume 4767 of *Lecture Notes in Computer Science*, pages 207–222. Springer Verlag, 2007.

[5] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, and Espen Højsgaard. An extensible formalization of WS-BPEL in binding bigraphs. Draft, 2008.

[6] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing higher-order mobile embedded business processes with binding bigraphs. In *Proceedings of the 10th international conference on Coordination Models and Languages (COORDINATION'08)*, Lecture Notes in Computer Science, pages 83–99. Springer Verlag, 2008.

[7] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *Proceedings of the Graph Transformation for Verification and Concurrency workshop (GT-VC'05)*, volume 154 of *Electronic Notes in Theoretical Computer Science*, pages 7–29. Elsevier, 2006.

[8] Mikkel Bundgaard, Thomas Hildebrandt, and Jens Chr. Godskesen. Modelling the security of smart cards by hard and soft types for higher-order mobile embedded resources. In Daniele Gorla and Catuscia Palamidessi, editors, *Proceedings of the 5th International Workshop on Security Issues in Concurrency (SecCo'07)*, volume 194 of *Electronic Notes in Theoretical Computer Science*, pages 23–38. Elsevier, 2007.

[9] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *Proceedings of the 16th European Symposium on Programming (ESOP'07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer Verlag, 2007.

[10] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[11] Giuseppe Castagna, Jan Vitek, and Fracesco Zappa Nardelli. The Seal calculus. *Journal of Information and Computation*, 201(1):1–54, 2005.

[12] Troels Christoffer Damgaard and Lars Birkedal. Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1–2):58–77, 2006.

[13] Dirk Fahland. Complete Abstract Operational Semantics for the Web Service Business Process Execution Language. Technical Report 190, Humboldt-Universität zu Berlin, 2005.

[14] Dirk Fahland and Wolfgang Reisig. ASM-based semantics for BPEL: The negative Control Flow. In Danièle Beauquier, Egon Börger, and Anatol Slissenko, editors, *Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05)*, pages 131–151. Paris XII, March 2005.

[15] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. Specification and validation of the business process execution language for web services. In *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *Lecture Notes in Computer Science*, pages 78–94. Springer Verlag, 2004.

[16] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. An abstract machine architecture for web service based business process management. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 144–157. Springer Verlag, 2006.

[17] Pablo Garralda, Adriana B. Compagnoni, and Mariangiola Dezani-Ciancaglini. BASS: Boxed ambients with safe sessions. In *Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'06)*, pages 61–72. ACM Press, 2006.

[18] Arne John Glenstrup, Troels Christoffer Damgaard, Lars Birkedal, and Espen Højsgaard. An implementation of bigraph matching. *submitted*, 2008.

[19] Jens Chr. Godskesen and Thomas Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. In *Proceedings of the 25th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 140–151. Springer Verlag, 2005.

[20] Volker Gruhn and André Köhler. Effects of mobile business processes on the software process. In *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim'04)*, pages 228–231. IEEE Computer Society Press, 2004.

[21] Thomas Hildebrandt, Jens Chr. Godskesen, and Mikkel Bundgaard. Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen, 2004.

[22] Thomas Hildebrandt, Henning Niss, and Martin Olsen. Formalising business process execution with bigraphs and Reactive XML. In Paolo Ciancarini and Herbert Wiklicky, editors, *Proceedings of the 8th international conference on Coordination Models and Languages (COORDINATION'06)*, volume 4038 of *Lecture Notes in Computer Science*, pages 113–129. Springer Verlag, 2006.

[23] Thomas Hildebrandt, Henning Niss, Martin Olsen, and Jacob W. Winther. Distributed Reactive XML. In Lubos Brim and Isabelle Linden, editors, *Proceedings of the 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'05)*, volume 150 of *Electronic Notes in Theoretical Computer Science*, pages 61–80, 2006.

[24] Thomas Hildebrandt (principal investigator). Computer supported mobile adaptive business processes (CosmoBiz) research project. Webpage, 2007. `http://www.cosmobiz.org/`.

[25] Kathrin Hoffmann and Till Mossakowski. Algebraic higher-order nets: Graphs and petri nets as tokens. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Proceedings of the 16th International Workshop on Recent Trends in Algebraic Development Techniques (WADT'02)*, volume 2755 of *Lecture Notes in Computer Science*, pages 253–267. Springer Verlag, 2003.

[26] Ole Høgh Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge – Computer Laboratory, 2004.

[27] Matthias Kloppmann, Dieter Koenig, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Reigen, Patrick Schmidt, and Ivana Trickovic. WS-BPEL extension for sub-processes: BPEL-SPE. Technical report, IBM and SAP, 2005.

[28] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A WSDL-based type system for WS-BPEL. In Paolo Ciancarini and Herbert Wiklicky, editors, *Proceedings of the 8th international conference on Coordination Models and Languages (COORDINATION'06)*, volume 4038 of *Lecture Notes in Computer Science*, pages 145–163. Springer Verlag, 2006.

[29] James J. Leifer and Robin Milner. Transition systems, link graphs and Petri nets. *Journal of Mathematical Structures in Computer Science*, 16(6):989–1047, 2006.

[30] Francesca Levi and Davide Sangiorgi. Mobile safe ambients. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 25(1):1–69, 2003.

[31] Niels Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In Marlon Dumas and Reiko Heckel, editors, *Proceedings of the 4th International Workshop on Web Services and Formal Methods (WS-FM'07)*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer Verlag, 2007.

[32] Niels Lohmann, H.M.W. Verbeek, Chun Ouyang, Christian Stahl, and Wil M. P. van der Aalst. Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology, 2007.

[33] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Journal of Information and Computation*, 100:1–40 and 41–77, 1992.

[34] Dimitris Mostrous and Nobuko Yoshida. Two session typing systems for higher-order mobile processes. In Simona Ronchi and Della Rocca, editors, *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications (TLCA'07)*, volume 4583 of *Lecture Notes in Computer Science*, pages 321–335. Springer Verlag, 2007.

[35] OASIS WSBPEL Technical Committee. Web Services Business Process Execution Language, version 2.0, 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`.

[36] Frank Puhlmann and Mathias Weske. Using the pi-calculus for formalizing workflow patterns. In Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management (BPM'05)*, volume 3649 of *Lecture Notes in Computer Science*, pages 153–168. Springer Verlag, 2005.

[37] Nick Russell, Arthur H.M. ter Hofstede, Will M.P. van der Aalst, and Nataliya Mulyar. Workflow control-flow patterns: A revised view. BPM Center Report BPM-06-22, BPMcenter.org, 2006.

[38] Christian Stahl. A Petri net semantics for BPEL. Informatik-Berichte 188, Humboldt-Universität zu Berlin, 2005.

[39] Christian Stefansen. A declarative framework for enterprise information systems. Master's thesis, Dept. of Computer Science, University of Copenhagen, 2005.

[40] Franck van Breugel and Maria Koshkina. Models and verification of BPEL. Draft., 2006.

# Chapter 6

# Core BPEL: Semantic Clarification of WS-BPEL 2.0 through Syntactic Simplification using XSL Transformations

**Tim Hallwyl and Espen Højsgaard**

**Abstract**

The Web Services Business Process Execution Language (WS-BPEL) is a language for expressing business process behaviour based on web services. The language is intentionally not minimal but provides a rich set of constructs, allows omission of constructs by relying on defaults, and supports language extensions. Combined with the fact that the language definition does not provide a formal semantics, it is an arduous task to work formally with the language (e.g. to give an implementation).

In this paper we identify a core subset of the language, called Core BPEL, which has fewer and simpler constructs, does not allow omissions, and does not contain ignorable elements. We do so by identifying syntactic sugar, including default values, and ignorable elements in WS-BPEL. The analysis results in a translation from the full language to the core subset. Thus, we reduce the effort needed for working formally with WS-BPEL, as one, without loss of generality, need only consider the much simpler Core BPEL. This report may also be viewed as an addendum to the WS-BPEL standard specification, which clarifies the WS-BPEL syntax and presents the essential elements of the language in a more concise way.

To make the results of this work directly usable for practical purposes, we provide an XML Schema for Core BPEL and a set of XSLT 1.0 transformations that will transform any standard compliant WS-BPEL process into a Core BPEL process. We also provide an online service where one can apply the transformation.

This work is part of the initial considerations on the implementation of a WS-BPEL engine within the Computer Supported Mobile Adaptive Business Processes (CosmoBiz) research project at the IT University of Copenhagen.

**Preface** This chapter consists of the technical report

T. Hallwyl and E. Højsgaard. *Core BPEL: Semantic Clarification of WS-BPEL 2.0 through Syntactic Simplification using XSL Transformations.* Technical Report TR-2011-138, IT University of Copenhagen, March 2011.

## 6.1 Introduction

The Web Services Business Process Execution Language (WS-BPEL) is a language for expressing business process behaviour based on web services. Based on the Extensible Markup Language (XML), the definition of WS-BPEL consists of two parts: A set of XML Schema documents[1], formally defining the syntax of the language, and a document [17], which we shall call the *standard specification*, defining the semantics and some further constraints on the syntax. The standard specification is written in prose and does not give any formal semantics for the language, which makes it harder than necessary to execute and reason about WS-BPEL processes as it is unclear what a WS-BPEL process means.

To remedy this, the academic community has proposed a number of formal semantics for WS-BPEL using a variety of formalisms (see related work below). These works have given a better understanding of WS-BPEL and have enabled formal analysis of WS-BPEL processes such as model checking. But the proposed formalizations are quite complex and comprehensive, which we believe stems from two sources: (a) WS-BPEL has some non-trivial features such as compensation and dead path elimination which are difficult to formalize concisely, and (b) WS-BPEL is (intentionally) not minimal, as for example the `<sequence>`-activity can be encoded as a `<flow>`-activity, and this redundancy carries over to the formalizations.

In this technical report, we investigate how we can remove some of the redundancy of WS-BPEL by showing how some constructs can be seen as *syntactic sugar* on a language core, which we term Core BPEL. By identifying a core language and a way to *desugar* the full language into the core subset, we can ease the tasks of implementing, analyzing, and formalizing the language, as one, without loss of generality, only needs to consider the core language. It should be noted that from a WS-BPEL programmer's perspective, features such as syntactic sugar are convenient and we do not suggest that WS-BPEL should be limited to a language core — we only aim to ease formal treatment of the language, and it is from this perspective that we in this report claim to *simplify* the WS-BPEL language. But we believe that the WS-BPEL user may also gain from this report, as it clarifies the WS-BPEL syntax and provides a more concise presentation of the essential elements of the language. From a more general perspective, this work also demonstrates that it is feasible to present a business process language as a set of primitive constructs from which a larger set of constructs may be built, and that this in fact promotes readability as well as clarifies and simplifies semantics. In particular, we expect a similar approach to be feasible and beneficial for the recent BPMN 2.0 standard [2].

The standard specification mentions many similarities between constructs and even defines some constructs by referring to the definition of other constructs. By analyzing these relations, we come up with desugaring transformations which preserve our understanding of the semantics, substantiated with quotes from the standard specification. Note that, in our analysis, we only consider executable processes, not the so-called abstract processes as these are not intended to be executed and may lack required operational details [17, Abstract]. Also, we disregard memory usage and execution speed, as these are implementation dependent and not treated by the standard.

Due to the lack of formal semantics in the standard specification we cannot prove our desugaring transformation to be semantics preserving in a formal sense; it might very well be the case, that our desugaring transformations do not preserve semantics with respect to formalizations given by the academic community or implementations of WS-BPEL. For instance, the memory usage of a syntactic sugar construct might be different from that of its desugared counterpart, which might lead to differences in exception behaviour. The only formal guarantee we give is that the WSDL types remain the same.

Core BPEL is thus a subset of WS-BPEL which has fewer and simpler constructs, and any valid executable WS-BPEL process can be transformed into an equivalent Core BPEL process by using the transformations presented in this report. We do not claim that the Core BPEL is minimal:

[1]The XML Schema documents are published in appendix E of the WS-BPEL standard specification [17].

there might be further or alternative transformations which yield a smaller language; nevertheless, our experience is that Core BPEL makes implementing and formalizing WS-BPEL significantly simpler.

In addition to syntactic sugar, WS-BPEL supports extensions that are implementation dependent and can be either optional or mandatory. Processes with mandatory extensions must be rejected if the implementation does not support them [17, SA00009], whereas unsupported optional extensions must be ignored by an implementation. Ignoring an extension can be done by removing it from the process before execution, and by providing a generic set of transformations for this, we again ease the implementation task. We therefore examine how optional extensions can be removed from a process by transforming it.

We have implemented the presented transformations as XSLT 1.0 templates. Some of the XSLT templates require access to the Web Service Definition Language (WSDL) files imported by the process.

In the appendix, we list syntax summaries (in the style of the WS-BPEL standard specification) for the WS-BPEL and Core BPEL elements side-by-side for easy comparison (App. 6.A). We also provide an XML Schema which defines the Core BPEL syntax (App. 6.B) and a set of XSLT 1.0 templates which implement the translation from WS-BPEL to Core BPEL (App. 6.D). The schema and transformations are also available at the CosmoBiz project website `http://www.cosmobiz.org` where we also provide a simple web-interface for experimenting with the transformations.

**Motivation** This investigation is part of the Computer Supported Mobile Adaptive Business Processes (CosmoBiz) project [11], which aims to provide a fully formalized runtime engine for a WS-BPEL-like business process language extended to allow for mobile and adaptive processes. As an initial step towards this goal, we are formalizing and implementing a WS-BPEL engine. By identifying a simpler core language of WS-BPEL, we ease the formalization and implementation. As a by-product, this investigation exposes some of the more subtle aspects of WS-BPEL, thereby making the semantics of the many constructs in WS-BPEL clearer and giving an overview of the many default behaviours specified in the standard specification.

Core BPEL allows WS-BPEL researchers and implementers to work with a simpler and more structured language that is both easier to understand and to reason about — without loss of generality. By providing the necessary transformations and a convenient web-interface, we hope to make our results easily accessible and usable for anyone working with WS-BPEL.

**Related work** The work presented in this paper is the continuation of Simplified BPEL as proposed by Hallwyl in his master's thesis [10]. The transformation into Simplified BPEL focused on adding implicit activities and default values, to support a so-called "standard-driven" implementation, a conceptual implementation closely mapping the prose descriptions found in the standard specification to code constructs. That aim did not allow us to remove or replace activities from the language. Core BPEL takes this further, by removing, transforming, and replacing constructs, aiming for a core subset of the language.

We are unaware of any other work on identifying a language core for WS-BPEL, but WS-BPEL has received considerable attention from the academic community, resulting in a number of formalizations using different formalisms the most notable being Petri Nets [12–14], process calculi such as the $\pi$-calculus [15] and bigraphs [3], and Abstract State Machines [6–9].

**Structure of the paper** We begin with some considerations in Section 6.2 about how to design and express the transformations from WS-BPEL to Core BPEL. In Section 6.3, we discuss how to make default attribute values and elements explicit. In Section 6.4 we discuss the standard attributes and elements of WS-BPEL which are allowed on all activities, and describe how these are only necessary on a few select activities. In Section 6.5 we identify syntactic sugar in WS-BPEL's activities and the `<process>` construct, and investigate how to desugar them one by one.

In Section 6.6 we discuss the extensibility of WS-BPEL and investigate how to remove optional extensions. Finally, in Section 6.7, we discuss how to combine the individual transformations into an overall transformation of WS-BPEL processes into Core BPEL and Section 6.8 concludes the paper with future perspectives.

The appendix contains the syntax summaries for the Core BPEL elements side-by-side with the corresponding WS-BPEL syntax summaries for easy comparison (App. 6.A), the XML Schema for Core BPEL (App. 6.B), an example WS-BPEL process and its Core BPEL equivalent, and the XSLT transformations we have constructed (App. 6.D).

We assume that the reader is familiar with WS-BPEL and XSLT.

## 6.2 Transformation Considerations

Before we start discussing how WS-BPEL can be simplified by transformations, we will discuss how we express such transformations and the design goals for our transformations. Also, we will discuss how concurrency must be taken into account when transforming activities, and how to handle the fact that WS-BPEL allows for very general and unconstrained language extensions, since it poses a challenge when we want to perform syntactic manipulations of WS-BPEL processes in general, without knowledge of specific extensions.

### 6.2.1 Language

XSLT has been chosen due to its widespread adoption and availability in many programming languages, making it easy to adopt our transformations in implementations. For historical reasons, we use XSLT version 1.0: when Hallwyl commenced his precursory work, XSLT 2.0 had not yet been widely adopted; also, WS-BPEL requires implementations to support XSLT 1.0 transformations [17, Sec. 8.3], and thus by using that version, the effort required by implementers to adopt our transformations is minimal. We are certain, though, that updating the transformations to XSLT 2.0 would make them more readable, but we leave this as an exercise for the reader.

XSLT has the drawback of being somewhat informally specified and untyped leading to inconsistencies between implementations as well as poor help for catching type errors in our transformations. For those reasons, we considered languages like XDuce [18] and CDuce [4], but ended up deciding that the advantages of XSLT outweighs its disadvantages.

### 6.2.2 Design Goals

When designing our transformations, we have had three goals in mind:

**idempotency:** each transformation should only affect elements that are not already valid Core BPEL. In other words, applying the same transformation a second time should not alter the result.

This ensures that the transformations are as specific as possible.

**independence:** each transformation should be valid for any WS-BPEL process, and should not rely on any of the other transformations being applied before or after itself.

This will allow users to employ a subset of the transformations instead of the whole suite, if they find them useful on their own, though this will of course not yield a Core BPEL process.

**simplicity:** the transformations should be as readable as possible. Most importantly, this means that we do not require that transformations use Core BPEL elements in their results; for example, several transformations produce `<sequence>`s, which in Core BPEL are encoded as `<flow>`s.

One could worry, that the price for this would be that each transformation might have to be applied several times, and in the worst case that the transformation suite would not terminate. Fortunately, it turns out that there is a simple way to avoid this, which we shall discuss in Section 6.7.

Also, we have put no effort into optimizing the execution of the transformations in order to keep them as simple as possible.

We believe that we have achieved these goals, except in one case: we have chosen to let the transformations that make default attribute values explicit (cf. Section 6.3.1) be more general than necessary: they make *all* WS-BPEL default attribute values explicit, though some of these are unnecessary and indeed not a part of Core BPEL, and therefore has to be removed in a separate step. We have made this choice, as we believe those transformations are of general interest and value in their general form, independently of Core BPEL.

### 6.2.3 Concurrency Considerations

Since WS-BPEL allows concurrency, e.g. using `<flow>`, we must ensure that our transformations do not alter the concurrency semantics as specified by the standard specification. In particular, since we are concerned with the elimination of syntactic sugar by transforming sugared activities into a composition of more primitive activities, one could fear that we might break atomicity.

The standard specification only puts a few requirements on the concurrent execution of activities: `<assign>` and initialization of correlation sets are required to be atomic:

> "If there is any fault during the execution of an assignment activity the destination variables MUST be left unchanged, as they were at the start of the activity (as if the assign activity were atomic). This applies regardless of the number of assignment elements within the overall assignment activity.
>
> The assign activity MUST be executed as if, for the duration of its execution, it was the only activity in the process being executed." *[17, Sec. 8.4]*

> "However, the initiation of a correlation set is performed in an atomic fashion – in the same sense as that of an `<assign>` operation – ensuring that the correlation set will not be partially initiated." *[17, Sec. 12.8]*

The execution of other activities is not required to be atomic. Thus, as long as our transformations do not split assignments or correlation set initializations into several activities, we do not break any atomicity requirements.

### 6.2.4 WS-BPEL Extensibility

WS-BPEL allows extensions in the form of namespace qualified attributes and elements, respectively on and in WS-BPEL elements. When transforming a WS-BPEL element, we cannot in general anticipate which of the resulting WS-BPEL elements each of the extensions relate to, nor whether a given extension still makes sense on the transformed element. For example, when we transform a `<receive>` into a `<pick>` with a single `<onMessage>` (cf. Section 6.5.3), some extensions might relate to the activity (`<pick>`) whereas others might relate to the messaging element (`<onMessage>`).

Thus, we cannot ensure that the transformations we construct will preserve semantics for extensions nor that the extension instances will be properly placed. And such considerations are out of scope for this report, as our interests are the WS-BPEL language constructs defined by the standard. Note though, that no matter where we place the extension instances, the result will still be valid WS-BPEL. In keeping with our design goal of simplicity, we therefore place the

extension instances wherever it is simplest in the XSLT code to put them. If this has unfortunate consequences for specific extensions, the transformations will have to be modified to handle those extensions explicitly.

## 6.3 Default Values and Elements

Having optional declarations and defaults means that something is stated when saying nothing. This is a special case of syntactic sugar, where the absence of some syntactic element should be interpreted the same as the presence of a particular instance of that syntactic element. As in the general case of syntactic sugar, making these implicit values explicit makes the language simpler without loosing expressivity.

In this section we briefly discuss the default values in WS-BPEL. For easy reference, they are listed in Table 6.1 and Table 6.2.

### 6.3.1 Default Attribute Values

We have divided the default attribute values into three groups as follows:

**simple defaults:** the default value is independent of the context of the construct

**global defaults:** there is a default value at the `<process>` level, and other constructs default to the value at that level

**inherited defaults:** there is a default value at the `<process>` level, and other constructs default to the value from the nearest enclosing element with the same attribute

Note that [17, Appendix C] contains a table of default values for attributes. But that table is missing the `ignoreMissingFromData` attribute on the `<copy>` element; it includes the attributes `keepSrcElementName` on `<copy>` and `initializePartnerRole` on `<partnerLink>` which, as argued in Sec. 6.3.1, cannot always be made explicit; and it for some reason also lists some attributes that do not have default values: `location` and `namespace` on `<import>` and `reference-scheme` on
`<sfref:service-ref>`.

#### Simple Defaults

XSLT template: Appendix 6.D.5

There are six optional attributes with default value `"no"` that are specific to the construct that they are part of. These are:

- `createInstance` on `<pick>` and `<receive>` [17, Sec. 11.5, 10.4].
- `ignoreMissingFromData` on `<copy>` [17, Sec. 8.4].
- `initiate` on `<correlation>` [17, Sec. 9.2].
- `isolated` on `<scope>` [17, Sec. 12.8].
- `succesfulBranchesOnly` on `<branches>` [17, Sec. 11.7].
- `validate` on `<assign>` [17, Sec. 8.4].

| Attributes | Default Value | Where |
|---|---|---|
| createInstance | no | <pick> <receive> |
| exitOnStandardFault | no | <process> |
| | inherited from immediately enclosing scope or process | <scope> |
| expressionLanguage | *xpath* | <process> |
| | inherited from <process> | <branches><br><condition><br><finalCounterValue><br><for><br><from><br><joinCondition><br><repeatEvery><br><startCounterValue><br><to><br><transitionCondition><br><until> |
| ignoreMissingFromData | no | <copy> |
| initiate | no | <correlation> |
| isolated | no | <scope> |
| messageExchange | the default <messageExchange> of the closest relevant parallel <forEach>, <onEvent>, or <process> | <onEvent><br><onMessage><br><receive><br><reply> |
| name | a fresh name | <assign>,<br><compensate>,<br><compensateScope>,<br><empty>,<br><exit>,<br><flow>,<br><forEach>,<br><if>,<br><invoke>,<br><pick>,<br><receive>,<br><repeatUntil>,<br><reply>,<br><rethrow>,<br><scope>,<br><sequence>,<br><throw>,<br><validate>,<br><wait>,<br><while> |
| queryLanguage | *xpath* | <process> |
| | inherited from <process> | <query> |
| successfulBranchesOnly | no | <branches> |
| suppressJoinFailure | no | <process> |
| | inherited from immediately enclosing activity or process | all activities |
| validate | no | <assign> |

Table 6.1: Default attribute values. Note that *xpath* is an abbreviation for `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`.

| **Elements** | **Default Value** | **Where** |
|---|---|---|
| <compensationHandler> | <compensationHandler><br><compensate /><br></compensationHandler> | <scope> |
| <completionCondition> | <completionCondition /> | <forEach> |
| <faultHandlers> | <faultHandlers><br><catchAll><br><sequence><br><compensate /><br><rethrow /><br></sequence><br></catchAll><br></faultHandlers> | <scope> |
| <joinCondition> | <joinCondition<br>expressionLanguage="*xpath*"><br>*disjunction of links*<br></joinCondition> | <target> |
| <messageExchange> | <messageExchange name="*fresh name*"/> | <process>,<br>child <scope> of <onEvent>,<br>parallel <forEach> |
| <terminationHandler> | <terminationHandler><br><compensate /><br></terminationHandler> | <scope> |
| <transitionCondition> | <transitionCondition<br>expressionLanguage="*xpath*"><br>true()<br></transitionCondition> | <source> |

Table 6.2: Default elements. Note that *xpath* is an abbreviation for `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`.

**The keepSrcElementName and initializePartnerRole attributes** The `keepSrcElementName` attribute on `<copy>` and `initializePartnerRole` on `<partnerLink>` both have default value `"no"` in [17, Appendix C] and in the case of `keepSrcElementName` the default value is even specified in the XML Schema for WS-BPEL. The default values do not always apply though:

> "An optional `keepSrcElementName` attribute is provided to further refine the behavior. [SA00042] It is only applicable when the results of both from-spec and to-spec are EIIs, and MUST NOT be explicitly set in other cases." *[17, Sec. 8.4.2]*

> "[SA00017] The `initializePartnerRole` attribute MUST NOT be used on a partner link that does not have a partner role; this restriction MUST be statically enforced." *[17, Sec. 6.2]*

Thus, the two are attributes are explicitly forbidden from being made explicit in certain cases. We guess that the authors of the WS-BPEL standard specification only intend the default values to apply *if* the attributes may be legally specified; but in that case, specifying a default value for the attribute in the XML Schema is not in keeping with the XML Schema standard specification:

> "*default* specifies that the attribute is to appear unconditionally in the post-schema-validation infoset, with the supplied value used whenever the attribute is not actually present" *[19, Sec. 3.2.1]*

We therefore choose not to have the default values in the Core BPEL schema, as they are not true defaults in the XML Schema sense.

One could make the default values explicit in the cases where they apply, but this would require more complex analyses than we wish to provide with our transformations, and we believe the value of doing so would be limited, as the constructs would not in general become simpler.

**Global Defaults** XSLT template: Appendix 6.D.3

The languages for expressions and queries can be specified using the optional `expressionLanguage` and `queryLanguage` attributes on relevant elements. The default for both attributes on `<process>` is
`urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`, while on other activities, the attributes by default inherit their values from `<process>` [17, Sec. 5.2, 8.2].

> "The value of the `queryLanguage` and `expressionLanguage` attributes on the `<process>` element are global defaults and can be overridden on specific constructs, such as `<condition>` of a `<while>` activity, as defined later in this specification." *[17, Sec. 5.2]*

**Inherited Defaults** XSLT template: Appendix 6.D.4

The optional attribute `suppressJoinFailure` is allowed on `<process>` and all activities. Its default value on `<process>` is `"no"`, while if it is unspecified on activities, it inherits its value from the nearest enclosing activity/`<process>` [17, Sec. 5.2]. Similarly, the optional attribute `exitOnStandardFault` is allowed on `<process>` and `<scope>`, with a default value of `"no"` on `<process>` while on `<scope>` it by default inherits the value from its nearest enclosing `<scope>`/`<process>` [17, Sec. 5.2].

### 6.3.2 Message Exchanges XSLT template: Appendix 6.D.8

When there is no ambiguity, WS-BPEL allows `messageExchange`'s to be omitted from IMAs[2] and `<reply>`s:

[2] inbound message activities

> “If the messageExchange attribute is not specified on an IMA or <reply> then the activity’s messageExchange is automatically associated with a default messageExchange with no name. Default messageExchange’s are implicitly declared by the <process> and the immediate child scopes of <onEvent> and the parallel form of <forEach>. Other occurrences of <scope> activities do not provide a default messageExchange.”
> *[17, Sec. 10.4.1]*

But the standard specification does not specify in which scope the default messageExchange associated with a given IMA or <reply> should be found! We see two possibilities for the choice of scope, but there may be more:

1. the nearest enclosing scope, with a default messageExchange, of the IMA or <reply>.
2. the nearest common enclosing scope, with a default messageExchange, of all the IMA’s and <reply>s that may be part of the same message exchange at runtime.

The second possibility requires advanced analysis, and is probably undecidable in general. The first possibility seems more plausible to us, as it is in line with the common lexical scoping rules as well as the reasoning for having default messageExchange’s declared in the immediate child scopes of <onEvent> and the parallel form of <forEach>

> “For example each time an <onEvent> is executed (i.e. when a new message arrives for processing) it creates a new default messageExchange instance associated with each <onEvent> instance. This allows a request-response <onEvent> event handler to receive messages in parallel without faulting or explicitly specifying a messageExchange. Similarly it allows the use of <receive>-<reply> or <onMessage>-<reply> pairs in the parallel form of <forEach> without the need to explicitly specify a messageExchange.”
> *[17, Sec. 10.4.1]*

Thus it seems that it should be the nearest enclosing scope which has a default messageExchange. Note that it is not truly the nearest *enclosing* scope in the case of <onEvent> (which is an IMA), since it is allowed to refer to its associated scope:

> “When the messageExchange attribute is explicitly specified, the resolution order of the message exchange referenced by messageExchange attribute MUST be first the associated scope and then the ancestor scopes.” *[17, Sec. 10.7.1]*

Assuming this interpretation of default messageExchange resolution, we can make the default message exchanges explicit in the following way:

1. Add a new <messageExchange> to the <process> and to each of the immediate child scopes of <onEvent> and the parallel forms of <forEach>, each with a unique fresh name (see Sec. 6.4.1 for a discussion of fresh names).
2. For all IMA’s or <reply>s where the messageExchange attribute is unspecified, set the attribute’s value to the name of the default <messageExchange> of the nearest enclosing scope of the kind listed in 1.

This is the interpretation that we implement, but we acknowledge that other interpretations are possible, as the standard specification is severely lacking on the subject. Note though, that the Core BPEL syntax remains the same regardless of which interpretation one chooses, as long as it can be decided statically.

### 6.3.3 Default Handlers

XSLT template: Appendix 6.D.7

When either no fault, compensation, or termination handlers are specified in a <scope> activity, default handlers apply [17, Sec. 12.5.1]. These default handlers are shown in Table 6.2.

### 6.3.4 Default Join, Transition, and Completion Conditions XSLT template: Appendix 6.D.6

The constructs `<targets>`, `<source>`, and `<forEach>` all have optional condition elements:

`<joinCondition>`

> "If no `<joinCondition>` is specified, the `<joinCondition>` is the disjunction (i.e. a logical OR operation) of the link status of all incoming links of this activity." *[17, Sec. 11.6.1]*

As links are statically declared — their source and target activities do not change during execution — we can explicitly declare the default join condition as the disjunction of the link statuses using an XPath expression.

`<transitionCondition>`

> "If the `<transitionCondition>` is omitted, it is assumed to evaluate to true." *[17, Sec. 11.6.1]*

This allows us to declare the default `<transitionCondition>` as the XPath expression `"true()"`.

`<completionCondition>`

> "When a `<completionCondition>` does not have any sub-elements or attributes understood by the WS-BPEL processor, it MUST be treated as if the `<completionCondition>` does not exist." *[17, Sec. 11.7]*

Thus the absence of a `<completionCondition>` is equivalent to `<completionCondition />`.

## 6.4 Standard Attributes and Elements XSLT template: Appendix 6.D.23

All WS-BPEL activities have two optional attributes and two optional elements, which in the standard specification are referred to as `standard-attributes` and `standard-elements` respectively, cf. the syntax summaries in Listing 6.4 and Listing 6.5 respectively. Making the default values explicit, as discussed in the previous section, simplifies the standard attributes and elements somewhat, but there is no reason to have these attributes and elements on all activities: we might as well move them to a wrapping `<flow>`, thereby restricting the syntax of Core BPEL further. In the following two sections, we discuss how this may be achieved.

### 6.4.1 Activity Names

First, we will discuss the standard attribute `name`. It is only in the case of `<scope>` that the value of `name` has any operational semantics specified by the standard (named scopes can be referenced by `<compensateScope>`), but it is also possible that in some cases, the name of an `<extensionActivity>` might be significant; this is not discussed in the standard. For all other activities, we can safely either remove the name or move it to a wrapping `<flow>`; the wrapper cannot be a `<scope>`, since that could break the following requirement in some cases:

> "[SA00092]Within a scope, the name of all named immediately enclosed scopes MUST be unique. This requirement MUST be statically enforced." *[17, Sec. 12.4.3]*

For example, the WS-BPEL fragment in Listing 6.1 can be transformed into the equivalent fragment in Listing 6.2 where the names are moved to a wrapping `<flow>`, but using `<scope>`s as wrappers,

Listing 6.1: Two activities with the same name.

```
1 <sequence>
2   <empty name="foo" />
3   <empty name="foo" />
4 </sequence>
5
6
7
8
```

Listing 6.2: Named wrapper.

```
1 <sequence>
2   <flow name="foo">
3     <empty />
4   </flow>
5   <flow name="foo">
6     <empty />
7   </flow>
8 </sequence>
```

Listing 6.3: Scope name clash.

```
1 <sequence>
2   <scope name="foo">
3     <empty />
4   </scope>
5   <scope name="foo">
6     <empty />
7   </scope>
8 </sequence>
```

as in Listing 6.3, violates [SA00092], whereas a wrapping `<flow>` does not alter the semantics of any activity.

For Core BPEL, we remove the redundant cases of the `name` attribute, i.e. we remove it from all activities but `<scope>` and `<extensionActivity>`.

Now, there is no default value for the `name` attribute, but there is nothing preventing us from assigning an unused (fresh) name to unnamed `<scope>`s (but not `<extensionActivity>`). Doing so makes the `name` attribute on `<scope>` mandatory in Core BPEL.

#### A Note on Fresh Names

Several of our transformations create named entities, e.g. variables or links, and it is important that we choose names that do not conflict with other named entities (e.g. variable shadowing). Though one could for each transformation analyse exactly how the created named entities might conflict with other entities, we do not believe that such an analysis will bring any significant insight. Instead, we simply generate globally fresh names in the following way:

- before we apply the transformations, we generate a string that is not the prefix of any attribute value in the WS-BPEL process to be transformed; we call this the *fresh prefix*.

- when a fresh name is needed in a transformation, we construct it as the concatenation of the following three parts

  **Fresh prefix:** the fresh prefix generated above.

  **Element id:** using the XSLT XPath function *generate-id()* on an element that is being transformed, we obtain an id for that instance of the transformation. If a transformation needs more than one fresh name, it uses a separate element for each fresh name.

  **Transformation postfix:** a postfix that is unique for that transformation.

### 6.4.2 Link endpoints

The remaining standard attributes and elements (`suppressJoinFailure`, `<targets>`, and `<sources>`) all relate to link endpoints. As with the `name` attribute, these may simply be moved to a wrapping `<flow>` activity, even in the case of `<scope>` and `<extensionActivity>`.

There is one detail to be aware of though: some `<scope>`s occur in contexts (`<onEvent>` and `<onAlarm>` in `<eventHandlers>`, and `<forEach>`) where other activities are not allowed, and it would therefore seem that we cannot in general move `<targets>` and `<sources>` to a wrapping `<flow>`. But a consequence of the following statement in the standard specification is that those `<scope>`s may not be target nor source of any links:

Listing 6.4: WS-BPEL `standard-attributes`.

```
name="NCName"?
suppressJoinFailure="yes|no"?
```

Listing 6.5: WS-BPEL `standard-elements`.

```
<targets>?
  <joinCondition expressionLanguage="anyURI"?>?
    bool-expr
  </joinCondition>
  <target linkName="NCName" />+
</targets>
<sources>?
  <source linkName="NCName">+
    <transitionCondition
      expressionLanguage="anyURI"?>?
      bool-expr
    </transitionCondition>
  </source>
</sources>
```

> A link used within a repeatable construct (`<while>`, `<repeatUntil>`, `<forEach>`, `<eventHandlers>`) or a `<compensationHandler>` MUST be declared in a `<flow>` that is itself nested inside the repeatable construct or `<compensationHandler>`. *[17, Sec. 11.6.1]*

Thus, we *can* move `<targets>` and `<sources>` to a wrapping `<flow>` everywhere they occur, and we do this for Core BPEL.

The `<scope>`s just discussed may still have the `suppressJoinFailure` attribute, though. But since the `<scope>` is not the target of any links, this attribute only serves to set the default value for the child activities of the `<scope>`: cf. Sec. 6.3.1, `suppressJoinFailure` by default inherits its value from the closest enclosing activity. We can therefore remove this attribute from the `<scope>`s in question by propagating the value of the `suppressJoinFailure` attribute to the immediate child activities.

Now, since our transformation will have to propagate the value of `suppressJoinFailure` for the `<scope>`s just discussed, we might as well do the same for activities that have no `<targets>` as they do not need the `suppressJoinFailure` attribute; the transformation does not become more complex due to this and it will lead to fewer wrapping `<flow>`s. This is purely an optimization, which does not affect the syntax of Core BPEL.

## 6.5 Desugaring Constructs

WS-BPEL provides programmers with a set of shorthands for frequent code patterns, i.e. *syntactic sugar*. For instance, when receiving a message, one often wants to separate the message into its constituent parts to ease further processing; WS-BPEL allows the programmer to specify this data processing concisely as part of the `<receive>` activity, thereby avoiding a sequence of explicit assignments.

The WS-BPEL standard specification does only in a few cases clearly describe shorthands as derived constructs, but it hints at many such relations. We have gone through the standard specification looking for such hints, as well as classical cases of syntactic sugar, and in this section we discuss the cases of syntactic sugar we have identified and we analyse how to desugar them.

The analysis is organized around the elements that we have constructed desugaring transformations for: `<process>`, `<invoke>`, `<pick>`, `<receive>`, `<reply>`, `<scope>`, `<if>`, `<repeatUntil>`, and `<sequence>`. There is a section for each of these constructs, and each section is accompanied by (a) an XSLT template in the appendix which formalizes and implements the described desugaring transformation, and (b) the syntax summary for the element, in the style of the standard specification, as well as the corresponding Core BPEL syntax summary. These are placed in Appendix 6.A and Appendix 6.D, and the header for each section contains references to the appropriate parts of the appendices. Table 6.3 gives an overview of the desugaring transformations we have constructed.

The elements for which we have not constructed separate desugaring transformations, are listed in Section 6.5.10; these are also accompanied by their WS-BPEL and Core BPEL syntax summaries, which are not identical, since many optional attributes and elements are made mandatory.

### 6.5.1 `<process>`

| | |
|---|---|
| WS-BPEL syntax summary: | Listing 6.34 |
| Core BPEL syntax summary: | Listing 6.35 |
| XSLT template: | Appendix 6.D.14 |

The `<process>` element serves as the root `<scope>` in the scoped environment hierarchy, and is allowed to contain almost all of the attributes and elements that are also allowed in a `<scope>` element as evident from their syntax summaries (cf. Listing 6.34 and Listing 6.64), and according to the standard specification the semantics of those attributes and elements are the same:

> "The `<process>` and `<scope>` elements share syntax constructs, which have the same semantics. However, they do have the following differences:
>
> - The `<process>` construct is not an activity; hence, standard attributes and elements are not applicable to the `<process>` construct
> - A compensation handler and a termination handler can not be attached to the `<process>` construct
> - The isolated attribute is not applicable to the `<process>` construct (see section 12.8. Isolated Scopes)"
>
> *[17, Sec. 12]*

Thus there is nothing preventing us from moving all `<scope>` related content from a `<process>` element into an explicitly defined `<scope>`. This includes the attributes `suppressJoinFailure` and `exitOnStandardFault`, which are made redundant for the `<process>` element by this transformation, since

`suppressJoinFailure` just serves to set an inherited default value, which is immediately overridden by the new `<scope>`, and

`exitOnStandardFault` the situation is the same, as any standard fault that may be trown, will be thrown inside of the new `<scope>`.

Listing 6.6 shows an example of `<scope>` related content in a `<process>` and Listing 6.7 shows how the scope is made explicit.

Also, note that two attributes `expressionLanguage` and `queryLanguage` serve only to specify default values for the activities of the process, so these can be safely removed after the defaults have been made explicit (i.e. after the transformation discussed in Section 6.3.1). As we shall discuss in Section 6.7, this is done as a separate transformation, in order to avoid problems when combining this transformation with the transformations of Section 6.3.1.

| Element | Syntactic sugar description | Desugaring transformation |
|---|---|---|
| `<if>` | Allows omitting the `<else>` element and allows `<elseif>` elements. | If the `<else>` element is omitted, insert `<else><empty/></else>`. Transform the list of `<elseif>` elements into a nested sequence of `<else><if>` elements. |
| `<invoke>` | Allows local fault and compensation handlers and manipulation of input/output using implicit local variables and implicit assignments. | Make an immediately enclosing `<scope>` and move handlers to that. Declare the implicit variables explicitly in that `<scope>` and make the implicit assignments explicit. |
| `<onEvent>` | Allows manipulation of input using implicit local variables and implicit assignments. | The standard forbids declaring the variables explicitly, but we make the implicit assignments of each `<onEvent>` explicit. |
| `<pick>` | Allows manipulation of input using implicit local variables and implicit assignments. | Make an immediately enclosing `<scope>` and declare the implicit variables explicitly in that `<scope>` and make the implicit assignments of each `<onMessage>` explicit. |
| `<process>` | Is an implicit scope. | Make the `<scope>` explicit. |
| `<receive>` | Corresponds to a `<pick>` activity which contains only one `<onMessage>` element. | Replace by a `<pick>` activity. |
| `<repeatUntil>` | Provides conditional iteration, just as `<while>`, though `<repeatUntil>` always performs at least one iteration. | Replace by a `<while>` where the condition is the negation of a temporary variable initialized to *false* and the body is a sequence of the `<repeatUntil>`'s body followed by an assignment of the value of its condition to the temporary variable. |
| `<reply>` | Allows manipulation of output using implicit local variables and implicit assignments. | Make an immediately enclosing `<scope>` and declare the implicit variables explicitly in that `<scope>` and make the implicit assignments explicit. |
| `<scope>` | Allows in-line variable initialization corresponding to implicit `<assign>` activities. | Convert the in-line variable initializations to explicit `<assign>` activities. To preserve the all-or-nothing behaviour of scope-initialization, this requires some additional `<scope>`s (see Section 6.5.6 for details). |
| `<sequence>` | Provides sequential processing, which can also be achieved with the `<flow>` activity. | Replace by a `<flow>` activity with properly defined links between the child activities. |

Table 6.3: Overview of syntactic sugar and transformations.

Listing 6.6: Process with scope related content.

```
1  <process>
2   <variables>
3    <variable name="size" type="xsd:int">
4     <from>
5      <literal>42</literal>
6     </from>
7    </variable>
8   </variables>
9   <catch ...>
10   activity
11  </catch>
12  <catchAll>
13   activity
14  </catchAll>
15
16  activity
17
18 </process>
19
20
```

Listing 6.7: Process with a scope as main activity.

```
1  <process>
2   <scope>
3    <variables>
4     <variable name="size" type="xsd:int">
5      <from>
6       <literal>42</literal>
7      </from>
8     </variable>
9    </variables>
10   <catch ...>
11    activity
12   </catch>
13   <catchAll>
14    activity
15   </catchAll>
16
17   activity
18
19  </scope>
20 </process>
```

### 6.5.2 <invoke>

| | |
|---|---|
| WS-BPEL syntax summary: | Listing 6.54 |
| Core BPEL syntax summary: | Listing 6.55 |
| XSLT template: | Appendix 6.D.11 |

The principal purpose of the `<invoke>` activity is to invoke an operation offered by a partner. But the standard specification allows the programmer to do more using this activity, e.g. to specify fault handling which is specific for this invocation.

We have identified three main cases of syntactic sugar for the `<invoke>` activity:

1. The local declaration of fault or compensation handlers implicitly declares an enclosing `<scope>` with these handlers.

2. The usage of `<toParts>` or `<fromParts>` to map variables to and from message parts implicitly declares an enclosing `<scope>` with temporary variables and assignments.

3. Referring to an element variable in either `<inputVariable>` or `<outputVariable>` also implicitly declares an enclosing `<scope>` with temporary variables and assignments.

A single `<invoke>` activity can match all three of these cases, but we examine each of them separately and then, at the end of the section, we examine how they interact.

But before we get to this, we will briefly digress to discuss a redundant attribute, `portType`, which can be used on the messaging activities (`<invoke>`, `<receive>`, `<reply>`, `<onEvent>`, and `<onMessage>`):

> "The `portType` attribute on the `<receive>` activity is optional... If the `portType` attribute is included for readability, the value of the `portType` attribute MUST match the `portType` value implied by the combination of the specified `partnerLink` and the `role` implicitly specified by the activity" *[17, Sec. 5.1]*

As both `partnerLink` and the `role` are required for those activities, the `portType` attribute can only carry redundant information. Thus, there is no reason to keep the `portType` attribute and consequently it has been removed from the messaging activities.

Listing 6.8: An <invoke> with an implicit <scope>.

```
<invoke ...
 name="bookFlight"
 suppressJoinFailure="yes">
 <targets>
  ...
 </targets>
 <sources>
  ...
 </sources>
  ...
 <catch ...>
  activity
 </catch>
 <catchAll>
  activity
 </catchAll>
 <compensationHandler>
  activity
 </compensationHandler>
 ...
</invoke>
```

Listing 6.9: Making the <scope> explicit.

```
<scope
 name="bookFlight"
 suppressJoinFailure="yes">
 <targets>
  ...
 </targets>
 <sources>
  ...
 </sources>
 <faultHandlers>
  <catch ...>
   activity
  </catch>
  <catchAll>
   activity
  </catchAll>
 </faultHandlers>
 <compensationHandler>
  activity
 </compensationHandler>
 <invoke name="bookFlight">
  ...
 </invoke>
</scope>
```

### Case 1: Fault and Compensation Handlers

Fault and compensation handlers are elements of the <scope> activity and when used in an <invoke> they implicitly declare a surrounding <scope> activity as follows:

> "Semantically, the specification of local fault handlers and/or a local compensation handler is equivalent to the presence of an implicit <scope> activity immediately enclosing the <invoke> providing these handlers. The implicit <scope> activity assumes the name of the <invoke> activity it encloses, its suppressJoinFailure attribute, as well as its <sources> and <targets>." *[17, Sec. 10.3]*

This means that whenever either fault or compensation handlers, or both, are declared within an <invoke> activity, it is to be executed as if there was an enclosing <scope> activity. Thus, writing an <invoke> activity along the lines of Listing 6.8 would be the same as writing out the scope explicitly, as in Listing 6.9.

### Case 2: Mapping Message Parts

The <invoke> activity also allows implicit assignment operations using <toParts> and <fromParts>. <toParts> is used to copy the contents of variables into specified parts of the message to be sent. Symmetrically, <fromParts> is used to copy parts of a received message into specified variables.

The standard specification says the following about the use of <toParts>:

> "By using the <toParts> element, an anonymous temporary WSDL variable is declared based on the type specified by the relevant WSDL operation's input message. The <toPart> elements, as a group, act as the single virtual <assign>, with each <toPart> acting as a <copy>." *[17, Sec. 10.3.1]*

Listing 6.10: <invoke> with implicit assignments.

```
<invoke
 name="orderItems"
 partnerLink="Seller"
 operation="Purchase"
 outputVariable="confirmation">
 <toParts>
  <toPart
   part="address"
   fromVariable="customerAddress"/>
  <toPart
   part="items"
   fromVariable="selectedItems"/>
 </toParts>
</invoke>
```

Listing 6.11: Making the assignments explicit.

```
<scope
 name="orderItems"
 <variables>
  <variable
   name="in"
   messageType="msg:PurchaseMessage"/>
 </variables>
 <sequence>
  <assign>
   <copy keepSrcElementName="yes">
    <from variable="customerAddress"/>
    <to variable="in" part="address"/>
   </copy>
   <copy keepSrcElementName="yes">
    <from variable="selectedItems"/>
    <to variable="in" part="items"/>
   </copy>
  </assign>
  <invoke
   name="orderItems"
   partnerLink="Seller"
   operation="Purchase"
   inputVariable="in"
   outputVariable="confirmation"/>
 </sequence>
</scope>
```

Declaring a temporary variable — one that is only visible during the execution of the <invoke> activity — is equivalent to having an immediately enclosing <scope> declaring the variable explicitly. Note that we have to use a fresh name for the variable so that it does not clash with any other variables referenced by the <invoke> activity (cf. Sec. 6.4.1 for a discussion of fresh names).

The virtual <assign> activity can then be declared explicitly in a <sequence> prior to the <invoke> activity. The following quote supports this idea:

> "The virtual <assign> MUST follow the same semantics and use the same faults as a real <assign>." *[17, Sec. 10.3.1]*

Listing 6.11 illustrates how the implicit <scope> and <assign> activities of Listing 6.10 are made explicit.

**Case 3: Element Variables**

The `inputVariable` and `outputVariable` attributes, when used, must refer to a WSDL message type variable matching the message to be sent or received respectively [17, SA00048], with one exception:

> "if the WSDL operation used in an <invoke> activity uses a message containing exactly one part which itself is defined using an element, then a variable of the same element type as used to define the part MAY be referenced by the `inputVariable` and `outputVariable` attributes respectively." *[17, Sec. 10.3]*

The standard specification prescribes that referring to an element variable is equivalent to declaring a temporary message variable and a virtual <assign> activity:

> "The result of using a variable in the previously defined circumstance MUST be the equivalent of declaring an anonymous temporary WSDL message variable based on the associated WSDL message type. The copying of the element data between the anonymous temporary WSDL message variable and the element variable acts as a single virtual `<assign>` with one `<copy>` operation whose `keepSrcElementName` attribute is set to `"yes"`." *[17, Sec. 10.3]*

This is similar to the usage of `<toParts>` and `<fromParts>`. However, in this case we are copying element variables, so the `keepSrcElementName` attribute on the `<copy>` operation is set to 'yes':

> "The optional `keepSrcElementName` attribute of the `<copy>` construct is used to specify whether the element name of the destination (as selected by the to-spec) will be replaced by the element name of the source (as selected by the from-spec) during the copy operation" *[17, Sec. 8.4]*

### Interplay between the cases

We have explored three cases of syntactic sugar for the `<invoke>` activity. In all three cases, desugaring makes an implicit `<scope>` explicit and in the last two cases, both implicit variables and virtual `<assign>` activities are made explicit. As mentioned previously, a single `<invoke>` could match all three cases, and thus we need to consider how they interact and how to desugar `<invoke>` as a whole. This is the topic of this section.

**Scope** Any fault caused by the virtual assignments of message parts (case 2) must be handled by the same local `<scope>` as the one which is implicitly declared by specifying local fault and compensation handlers (case 1):

> "The virtual `<assign>` created as a consequence of the `<fromPart>` or `<toPart>` elements occurs as part of the scope of the `<invoke>` activity and therefore any fault that is thrown are caught by an `<invoke>`'s inline fault handler when defined." *[17, Sec. 10.3.1]*

The standard specification does not describe how to handle errors from virtual assignments caused by a reference to an element variable (case 3). We suspect that this is because errors are not expected, as both the source and target variables are certain to exist and to be of the exact same type. However, implementation specific faults, e.g. 'out of memory', may still occur. As the standard specification does not describe how to handle such faults, it would be consistent to follow the pattern from assignments of parts, i.e. to let the `<invoke>`'s own `<scope>` handle any unexpected fault that might occur.

**Implicit activities** As `<toParts>` and `inputVariable` are mutually exclusive, there can be no conflict between their implicit activities; the same goes for `<fromParts>` and `outputVariable`. Thus, there will be at most one virtual `<assign>` activity prior to and following the `<invoke>` activity respectively, so virtual assignments do not conflict and may be part of the same `<sequence>` within the same `<scope>`. The body of the `<scope>` thus becomes a `<sequence>` of zero or one prior assignment followed by the core `<invoke>` activity and zero or one assignments at the end.

### Transforming Invoke

The shared `<scope>` and `<sequence>` makes it difficult to consider the three cases separately when constructing the XSLT template and as a result, the template becomes somewhat large and complex. To give an overview, we outline the structure of the main template in pseudo code in Listing 6.12. In the pseudo code, `inputElement` and `outputElement` are booleans, indicating whether the input or output variable attributes are referring to element (*true*) or message variables (*false*). Element names, as in XSLT, evaluate to *true* when present, and *false* otherwise.

Listing 6.12: Transforming <invoke>.

```
1  if (toParts ∨ fromParts ∨ inputElement ∨ outputElement ∨ catch ∨ catchAll ∨
   compensationHandler){
2
3      <scope ...>
4
5          if (toParts ∨ fromParts ∨ inputElement ∨ outputElement)
6              Implicit temporary variables in invoke made explicit:
7              <variables> ... </variables>
8
9          if (catch ∨ catchAll)
10             Move invokes local fault handlers to enclosing scopes
11             <catch ...> ... </catch>*
12             <catchAll ...> ... </catchAll>
13
14         if (compensationHandler)
15             Move invokes local compensation handler to enclosing scope
16             <compensationHandler> ... </compensationHandler>
17
18         if (toParts ∨ fromParts ∨ inputElement ∨ outputElement) {
19
20             <sequence>
21
22                 if (toParts)
23                     <assign>
24                         <copy> ... </copy>*
25                     </assign>
26
27                 if (inputElement)
28                     <assign>
29                         <copy> ... </copy>*
30                     </assign>
31
32                 <invoke ...>
33
34                 if (fromParts)
35                     <assign>
36                         <copy> ... </copy>*
37                     </assign>
38
39                 if (outputElement)
40                     <assign>
41                         <copy> ... </copy>*
42                     </assign>
43
44             </sequence>
45
46         } else {
47
48             Invoke without implict assignment, but with implict scope.
49             <invoke ...>
50
51         }
52
53     </scope>
54
55 } else {
56
57     A core invoke
58     <invoke ...>
59
60 }
```

Listing 6.13: A synchronous `<invoke>`.

```
<invoke
 partnerLink="pl"
 operation="op"
 inputVariable="in"
 outputVariable="out" />
```

Listing 6.14: Encoding synchronous `<invoke>`.

```
<sequence>
 <invoke
  partnerLink="pl"
  operation="op"
  inputVariable="in" />
 <receive
  partnerLink="pl"
  operation="op"
  variable="out" />
</sequence>
```

### A note on synchronous `<invoke>`

It is well known that one can often encode a synchronous communation operation as a pair of asynchronous communication operations. In the case of WS-BPEL, one could for example imagine transforming the synchronous `<invoke>` in Listing 6.13 to the asynchronous `<invoke>`/`<receive>`-pair in Listing 6.14.

While these code fragments abstractly accomplish the same request/response operation, they are not interchangeable because WSDL models request/response (and solicit/response) as primitive operation types:

> “WSDL has four transmission primitives that an endpoint can support:
>
> - **One-way.** The endpoint receives a message.
> - **Request-response.** The endpoint receives a message, and sends a correlated message.
> - **Solicit-response.** The endpoint sends a message, and receives a correlated message.
> - **Notification.** The endpoint sends a message.
>
> WSDL refers to these primitives as **operations**. Although request/response or solicit/response can be modeled abstractly using two one-way messages, it is useful to model these as primitive operation types because:
>
> - They are very common.
> - The sequence can be correlated without having to introduce more complex flow information.
> - Some endpoints can only receive messages if they are the result of a synchronous request response.
> - A simple flow can algorithmically be derived from these primitives at the point when flow definition is desired.”
>
> *[5, Sec. 2.4]*

Concretely, this means that the two fragments cannot both use the same WSDL definitions for the partner link `pl`, as the definition of `op` in the case of Listing 6.13 would have to be on the following form:

```
<wsdl:operation name="op" ...>
  <wsdl:input .../>
  <wsdl:output .../>
</wsdl:operation>
```

Listing 6.15: <receive> example.

```
1 <receive partnerLink="purchasing"
2  portType="lns:purchaseOrderPT"
3  operation="sendPurchaseOrder"
4  variable="PO"
5  createInstance="yes">
6
```

Listing 6.16: <receive> example desugared.

```
1 <pick createInstance="yes">
2  <onMessage partnerLink="purchasing"
3   portType="lns:purchaseOrderPT"
4   operation="sendPurchaseOrder"
5   variable="PO">
6 </pick>
```

whereas Listing 6.14 would require a separate callback operation, and such a transformation would thus change the WSDL interface of the process. For the same reasons, one cannot change a <reply> into an asynchronous <invoke>, though they both send a message without expecting a reply.

### 6.5.3 <receive>

WS-BPEL syntax summary: Listing 6.58
XSLT template: Appendix 6.D.15

The <receive> activity is similar to the <onMessage> event of the <pick> activity, as evident from their syntax summaries (cf. Listing 6.58 and Listing 6.56) and the following quotes from the standard specification:

> "The <onMessage> is similar to a <receive> activity, in that it waits for the receipt of an inbound message." *[17, Sec. 11.5]*

> "[SA00063] The semantics of the <onMessage> event are identical to a <receive> activity regarding the optional nature of the variable attribute or <fromPart> elements (see also [SA00047]), the handling of race conditions, the handling of correlation sets, the single element- based part message short cut and the constraint regarding simultaneous enablement of conflicting receive actions. For the last case, if two or more receive actions for the same partnerLink, portType, operation and correlationSet(s) are simultaneously enabled during execution, then the standard fault bpel:conflictingReceive MUST be thrown (see section 10.4. Providing Web Service Operations - Receive and Reply). Enablement of an <onMessage> event is equivalent to enablement of the corresponding <receive> activity for the purposes of this constraint." *[17, Sec. 11.5]*

In fact, we have found nothing in the standard specification which indicates any differences between a <receive> activity and a <pick> activity with a single <onMessage> event. Thus, given the above quotes, it seems reasonable to assume that they should be equivalent, and thus we can replace the former with the latter. Listing 6.16 shows how the <receive> activity example in Listing 6.15 is desugared into a <pick>.

### 6.5.4 <pick>

WS-BPEL syntax summary: Listing 6.56
Core BPEL syntax summary: Listing 6.57
XSLT template: Appendix 6.D.13

An <onMessage> element of a <pick> activity makes use of implicit assignments when <fromParts> is used or when its `variable` attribute refers to an element variable, just as it was the case with <invoke> (though for <invoke> the attribute name is `outputVariable`). The standard specification defines the syntax and semantics of <fromParts> in <receive> to be the same as in the context of <invoke>:

> "The syntax and semantics of the <fromPart> elements as used on the <receive> activity are the same as specified for the <invoke> activity in section 10.3.1." *[17, Sec. 10.4]*

And as argued in the previous section, `<receive>` is a special case of `<pick>`, so the same semantics should apply to `<fromParts>` in `<onMessage>`. With respect to the semantics of the `variable` attribute when it refers to an element variable, the relation to `<invoke>` is a little less clear. But the requirements on static analysis [17, SA00058] for `<receive>` are phrased similarly to that of the `<invoke>` activity [17, SA00048], and thus the discussion from case 3 in Section 6.5.2 applies to `<receive>`, and thereby `<onMessage>`, as well.

Thus, we can reuse the transformation we constructed to handle `<fromParts>` and element variables in the `<invoke>` activity, with two minor differences:

- we will have to place the temporary variables for all of the `<onMessage>`s in a shared `<scope>` enclosing the `<pick>`
- the explicit assignments must be placed inside the `<onMessage>`s

Listing 6.17 shows an example `<pick>` with two `<onMessage>`s using `<fromParts>` and an element variable, and the example is desugared in Listing 6.18.

### 6.5.5 `<reply>`

| | |
|---|---|
| WS-BPEL syntax summary: | Listing 6.60 |
| Core BPEL syntax summary: | Listing 6.61 |
| XSLT template: | Appendix 6.D.20 |

The `<reply>` activity makes use of implicit assignments when `<toParts>` is used or when its `variable` attribute refers to an element variable, just as it was the case with `<invoke>` (though for `<invoke>` the attribute name is `inputVariable`). Indeed, the standard specification defines the syntax and semantics of `<toParts>` to be the same as in the context of `<invoke>`:

> "The syntax and semantics of the `<toPart>` elements as used on the `<reply>` activity are the same as specified in section 10.3.1. Mapping WSDL Message Parts for the `<invoke>` activity" *[17, Sec. 10.4]*

With respect to the semantics of the `variable` attribute when it refers to an element variable, the relation to `<invoke>` is a little less clear. But the requirements on static analysis [17, SA00058] are phrased similarly to that of the `<invoke>` activity [17, SA00048], and thus the discussion from case 3 in Section 6.5.2 applies to `<reply>` as well.

Thus, we can reuse the transformation we constructed to handle `<toParts>` and element variables in the `<invoke>` activity.

### 6.5.6 `<scope>`

| | |
|---|---|
| WS-BPEL syntax summary: | Listing 6.64 |
| Core BPEL syntax summary: | Listing 6.65 |
| XSLT template: | Appendix 6.D.21 |

The concept of virtual `<assign>` activities arises again in the descriptions of in-line variable initializations. Variables are declared as part of a `<scope>` activity. Within the `<variable>` declaration, a `from-spec`, as known from assignments [17, Sec. 8.4], may be used to initialize the variable. Listing 6.19 shows a simple example, initializing an integer variable to the value 42.

The standard specification makes this feature sound entirely trivial:

> "Conceptually the in-line variable initializations are modeled as a virtual `<sequence>` activity that contains a series of virtual `<assign>` activities, one for each variable being initialized, in the order they are listed in the variable declarations." *[17, Sec. 8.1]*

I.e. when one or more variables are initialized within the variables declaration, we can make the virtual assignments explicit in a `<sequence>`, putting the scope main activity at the end of the sequence and letting the sequence become the scope main activity instead. Listing 6.20 illustrates this approach. While this will initialize the variables before they are used, it does not provide semantic equivalence wrt. fault handling, cf. the requirements for scope initialization:

Listing 6.17: <pick> with implicit assignments.

```
<pick>
  <onMessage
    partnerLink="partnerLink"
    operation="op1">
    <fromParts>
      <fromPart
        part="id"
        toVariable="idVar" />
      <fromPart
        part="description"
        toVariable="descVar" />
    </fromParts>
    <empty />
  </onMessage>
  <onMessage
    partnerLink="partnerLink"
    operation="op2"
    variable="addressVariable">
    <empty />
  </onMessage>
</pick>
```

Listing 6.18: Making the assignments explicit.

```
<scope>
 <variables>
  <variable
   name="fresh1"
   messageType="..." />
  <variable
   name="fresh2"
   messageType="..." />
 </variables>
 <pick>
  <onMessage
   partnerLink="partnerLink"
   operation="op1"
   variable="fresh1">
   <sequence>
    <assign>
     <copy>
      <from part="id"
         variable="fresh1"/>
      <to variable="idVar"/>
     </copy>
     <copy>
      <from part="description"
         variable="fresh1"/>
      <to variable="descVar"/>
     </copy>
    </assign>
    <empty/>
   </sequence>
  </onMessage>
  <onMessage
   operation="op2"
   partnerLink="partnerLink"
   variable="fresh2">
   <sequence>
    <assign>
     <copy keepSrcElementName="yes">
      <from part="addressIn"
         variable="fresh2"/>
      <to variable="addressVariable"/>
     </copy>
    </assign>
    <empty/>
   </sequence>
  </onMessage>
 </pick>
</scope>
```

Listing 6.19: Example of a variable initialization.

```
<scope ...>
 <variables>
  <variable name="size" type="xsd:int">
   <from><literal>42</literal></from>
  </variable>
 </variables>
 <flow>
  <invoke .../>
  <invoke .../>
 </flow>
</scope>
```

Listing 6.20: Naive proposal for transformation of the variable initialization example.

```
<scope ...>
 <variables>
  <variable name="size" type="xsd:int"/>
 </variables>
 <sequence>
  <assign>
   <copy>
    <from><literal>42</literal></from>
    <to variable="size"/>
   </copy>
  </assign>
  <flow>
   <invoke .../>
   <invoke .../>
  </flow>
 </sequence>
</scope>
```

> "Scope initialization is an all-or-nothing behavior: either it all occurs successfully or a `bpel:scopeInitializationFailure` fault MUST be thrown to the parent scope of the failed <scope>." *[17, Sec. 12.1]*

This means that we cannot declare the virtual assignments as part of the scope main activity: Any faults from these assignments would be caught by the scope's own fault handler, instead of resulting in
`bpel:scopeInitializationFailure` fault being thrown to the parent scope as required.

Our solution involves three <scope>s:

**Variable scope:** Declares the variables, so they are reachable from both of the other two <scope>s, which are put in a <sequence> within this scope. It will rethrow any faults, thus making it transparent regarding fault handling, and the default compensation handling will ensure that it is transparent in this regard as well.

**Initialization scope:** Assigns the initial values to the variables. It has a fault handler that catches all faults and rethrows them as `scopeInitializationFault`s. This ensures that faults from variable initializations will cause a `scopeInitializationFault`, as required.

**Main scope:** Executes the original main activity. It will also have the fault handlers from the original <scope> activity.

Listing 6.21 shows how the <scope> from Listing 6.19 is desugared.

#### <onEvent>

A <scope> may have a set of event handlers associated with it, which are invoked in parallel when their corresponding event occurs. One type of event handler is <onEvent> which receives messages much in the same way as <receive>:

> "The <onEvent> element indicates that the specified event waits for a message to arrive. The interpretation of this element and its attributes is very similar to a <receive> activity." *[17, Sec. 12.7.1]*

Listing 6.21: Unfolding variable initialization.

```
<scope>
 <variables>
  <variable name="size" type="xsd:int">
 </variables>
 <faultHandlers>
  <catchAll>
   <rethrow/>
  </catchAll>
 </faultHandlers>
 <sequence>
  <scope>
   <faultHandlers>
    <catchAll>
     <throw faultName="scopeInitializationFault"/>
    </catchAll>
   </faultHandlers>
   <assign>
    <copy>
     <from>
      <literal>42</literal>
     </from>
     <to variable="size"/>
    </copy>
   </assign>
  </scope>
  <scope>
   <flow>
    <invoke ... />
    <invoke ... />
   </flow>
  </scope>
 </sequence>
</scope>
```

The differences between `<onEvent>` and `<receive>` stem from the fact that the former is to be executed every time the specified message is received, thus starting several instances of its enclosed `<scope>` and it may therefore use partner links, message exchanges, and correlation sets from within the enclosed `<scope>` activity:

> "The `partnerLink` reference MUST resolve to a partner link declared in the process in the following order: the associated scope first and then the ancestor scopes." *[17, Sec. 12.7.1]*

> "When the `messageExchange` attribute is explicitly specified, the resolution order of the message exchange referenced by `messageExchange` attribute MUST be first the associated scope and then the ancestor scopes." *[17, Sec. 12.7.1]*

> "The usage of `<correlation>` is exactly the same as for `<receive>` activities, with the following addition: it is possible, from an event handler's inbound message operation, to use correlation sets that are declared within the associated scope." *[17, Sec. 12.7.1]*

For the same reason, the variable(s) used for the incoming message is/are placed in the associated `<scope>`, but they are implicitly declared and the standard specification specifically forbids making them explicit[3]:

> "Variables referenced by the `variable` attribute of `<fromPart>` elements or the `variable` attribute of an `<onEvent>` element are implicitly declared in the associated scope of the event handler. [SA00086] Variables of the same names MUST NOT be explicitly declared in the associated scope. This requirement MUST be enforced by static analysis." *[17, Sec. 12.7.1]*

But even so, it is still possible to make the implicit assignments, due to the use of the `element` attribute or `<fromParts>`, explicit, in a similar fashion to the transformations of `<invoke>` and `<pick>`/`<receive>` (cf. Sec. 6.5.2):

> "The syntax and semantics of the `<fromPart>` elements as used on the `<onEvent>` element are the same as specified in section 10.4. Providing Web Service Operations – Receive and Reply for the receive activity." *[17, Sec. 12.7.1]*

> "If an `element` attribute is used then the binding of the incoming message to the variable declared in the `<onEvent>` event handler occurs as specified for the receive activity (...)." *[17, Sec. 12.7.1]*

The difference is simply that the temporary message variable will be implicitly declared by the `<onEvent>` and the element variable or message part variables will be placed in its associated `<scope>`; other than that, the transformation is the same. Listing 6.22 shows an `<onEvent>` which uses an element variable, and Listing 6.23 shows how the implicit variable and assignment is made explicit.

### 6.5.7 `<if>`

| | |
|---|---|
| WS-BPEL syntax summary: | Listing 6.52 |
| Core BPEL syntax summary: | Listing 6.53 |
| XSLT template: | Appendix 6.D.10 |

The `<if>` activity allows two classic variants of syntactic sugar: leaving out the `<else>`-branch, and contracting a nested sequence of `<else><if>`s to a list of `<elseif>`s. Listing 6.24 gives an example and Listing 6.25 shows the desugared version.

[3] `<fromPart>` does not have a `variable` attribute but a `toVariable` attribute; we assume that what was intended.

Listing 6.22: An <onEvent> declaration.

```
<onEvent
 partnerLink="consumer"
 operation="getStatus"
 element="xsd:string"
 variable="statusRequest">
 <scope name="event">
  <partnerLinks>
   <partnerLink name="consumer"
    partnerLinkType="..."
    myRole="provider" />
  </partnerLinks>
  <empty name="activity" />
 </scope>
</onEvent>
```

Listing 6.23: <onEvent> desugared.

```
<onEvent
 partnerLink="consumer"
 operation="getStatus"
 messageType="msg:StatusMessage"
 variable="fresh">
 <scope name="event">
  <partnerLinks>
   <partnerLink name="consumer"
    partnerLinkType="..."
    myRole="provider" />
  </partnerLinks>
  <variables>
   <variable name="statusRequest"
        element="xsd:string" />
  </variables>
  <sequence>
   <assign>
    <copy keepSrcElementName="yes">
     <from variable="fresh"
        part="foo" />
     <to variable="statusRequest" />
    </copy>
   </assign>
   <empty name="activity" />
  </sequence>
 </scope>
</onEvent>
```

Listing 6.24: An <if> activity using <elseif>.

```
<if>
 <condition>$foo</condition>
 activity1

 <elseif>
  <condition>$bar</condition>
  activity2
 </elseif>
</if>
```

Listing 6.25: <if> desugared.

```
<if>
 <condition>$foo</condition>
 activity1

 <else>
  <if>
   <condition>$bar</condition>
   activity2

   <else>
    <empty/>
   </else>
  </if>
 </else>
</if>
```

### 6.5.8 <repeatUntil>

WS-BPEL syntax summary: Listing 6.59
XSLT template: Appendix 6.D.19

WS-BPEL includes another classic example of redundancy: it includes two repetition constructs, `<while>` and `<repeatUntil>`, with only minor differences in their semantics:

> "The `<while>` activity provides for repeated execution of a contained activity. The contained activity is executed as long as the Boolean `<condition>` evaluates to true at the beginning of each iteration." *[17, Sec. 11.3]*

> "The `<repeatUntil>` activity provides for repeated execution of a contained activity. The contained activity is executed until the given Boolean `<condition>` becomes true. The condition is tested after each execution of the body of the loop. In contrast to the `<while>` activity, the `<repeatUntil>` loop executes the contained activity at least once." *[17, Sec. 11.4]*

As stated in the quote above, the main difference between `<while>` and `<repeatUntil>` is that the latter always executes its body at least once before checking the condition, whereas `<while>` checks the condition first. The second difference is that `<repeatUntil>` stops looping when its condition becomes true, whereas `<while>` loops as long as its condition is true.

Either can be seen as a sugared version of the other:

**`<repeatUntil>` → `<while>`:**

The text-book transformation of repeat-until to while is (using pseudo-code for brevity):

```
                                     sequence
                                       body
repeat body until condition    →       while (not condition) do
                                         body
```

This will not immediately work for WS-BPEL, as copying the body might involve making copies of named `<scope>`s which is problematic: scope names within the same immediately enclosing scope must be unique:

> "[SA00092]Within a scope, the name of all named immediately enclosed scopes MUST be unique. This requirement MUST be statically enforced." *[17, Sec. 12.4.3]*

so one would have to rename all scopes in the body and somehow make sure that compensation works as intended.

Instead, one could imagine doing a less standard transformation which avoids copying the body:

```
                                 scope
                                   variable first := true
                                   while first or (not condition) do
repeat body until condition →        sequence
                                       body
                                       first := false
```

Note though, that this transformation negates the condition, but WS-BPEL allows any expression language to be used, and there is no requirement that expression languages must have a negation operator! Also, using this approach, one would have to extend our transformations with expression language specific behaviour.

To avoid these issues, we can use the following slightly more complicated transformation:

```
                                    scope
                                      variable cond_var := false
                                      while (not cond_var) do
repeat body until condition →           sequence
                                          body
                                          cond_var := condition
```

In this transformation, we evaluate the condition expression unchanged, storing the result in the fresh variable `cond_var`, and then use XPath to negate the stored result. The transformation is thus independent of the expression language. Note that, since it is only recommended that boolean expressions return a value of `xsd:boolean`, we must, in the general case, evaluate the condition in the following way to ensure that we store a value of type `xsd:boolean`:

```
if condition then
  cond_var := true
else
  cond_var := false
```

In this case we can optimize the negation of the condition variable and the else-case away, finally arriving at the following transformation:

```
                                    scope
                                      variable cond_var := true
                                      while cond_var do
repeat body until condition →           sequence
                                          body
                                          if condition then
                                            cond_var := false
```

`<while>` → `<repeatUntil>`:

The reverse transformation can be achieved as follows:

```
while condition do body  →   if condition then
                               repeat body until (not condition)
```

The body is not copied, whereas the condition is but we have found no indications in the WS-BPEL standard that this could raise any problems – but this could depend on the expression language used.

As was the case in the second transformation above, this transformation negates the expression directly, which might not be desirable, and we can use a similar trick to avoid this:

```
                             if condition then
                               scope
                                 variable cond_var
                                 repeat
while condition do body  →         sequence
                                     body
                                     cond_var := condition
                                 until (not $cond_var)
```

Thus, we can transform either construct into the other. We choose to use the transformation of `<repeatUntil>` as this transformation results in slightly shorter code and it also avoids making more than one copy of the condition.

Listing 6.26 shows an example of a `<repeatUntil>` activity which is desugared into the code in Listing 6.27.

Listing 6.26: Example of a <repeatUntil>.

```
<repeatUntil>
 <targets>
  <target linkName="l1" />
 </targets>

 <empty name="body" />

 <condition
  expressionLanguage="expr-lang">
  cond-expression
 </condition>
</repeatUntil>
```

Listing 6.27: A <repeatUntil> rewritten.

```
<scope>
 <targets>
  <target linkName="l1" />
 </targets>
 <variables>
  <variable
   name="cond_var"
   type="xsd:boolean">
   <from expressionLanguage="xpath">
    false()
   </from>
  </variable>
 </variables>

 <while>
  <condition expressionLanguage="xpath">
   not($cond_var)
  </condition>
  <sequence>
   <empty name="body" />
   <if>
    <condition
     expressionLanguage="expr-lang">
     cond-expression
    </condition>
    <assign>
     <copy>
      <from
       expressionLanguage="xpath">
       true()
      </from>
      <to variable="cond_var" />
     </copy>
    </assign>
   </if>
  </sequence>
 </while>
</scope>
```

Listing 6.28: Example of a <sequence>.

```
<sequence>
 <empty name="A" />
 <empty name="B" />
 <empty name="C" />
</sequence>
```

Listing 6.29: A <sequence> rewritten.

```
<flow>
 <links>
  <link name="fresh1" />
  <link name="fresh2" />
 </links>
 <flow>
  <sources>
   <source linkName="fresh1" />
  </sources>
  <empty name="A" />
 </flow>
 <flow>
  <targets>
   <target linkName="fresh1" />
  </targets>
  <sources>
   <source linkName="fresh2" />
  </sources>
  <empty name="B" />
 </flow>
 <flow>
  <targets>
   <target linkName="fresh2" />
  </targets>
  <empty name="C" />
 </flow>
</flow>
```

### 6.5.9 <sequence>

WS-BPEL syntax summary: Listing 6.66
XSLT template: Appendix 6.D.22

The authors of the standard specification are aware that the language is not 'minimal' and even suggest that the `<sequence>` activity could be modeled using a `<flow>` activity:

> "The set of structured activities in WS-BPEL is not intended to be minimal. There are cases where the semantics of one activity can be represented using another activity. For example, sequential processing may be modeled using either the `<sequence>` activity, or by a `<flow>` with properly defined links." *[17, Sec. 11]*

Essentially, the `<sequence>` activity is syntactic sugar for a `<flow>` construction with sequentially linked child activities. We could manipulate the `<targets>` and `<sources>` elements of the child elements directly to include additional sequencing links. But this would involve adding the sequencing links to the `<joinCondition>`s, i.e. manipulating expressions in arbitrary languages.

A more straightforward alternative is to wrap each of the child activities in a `<flow>` activity and place the sequencing links there; the default `<joinCondition>`s will suffice in this case. For example, we can rewrite the `<sequence>` in Listing 6.28 into the `<flow>` of Listing 6.29. Note that we need fresh names for the links we add, cf. Sec. 6.4.1, that the default `<transitionCondition>` is sufficient, and that the value of the `suppressJoinFailure` attribute is irrelevant for sequence flows, since the join condition will always be true, if the preceding activity finishes without propagating a fault; if the preceding activity propagates a fault, the transition condition will not be evaluated and no `joinFailure` will occur.

### 6.5.10 Non-sugared activities

Besides default values, we have not identified any cases of syntactic sugar in the activities listed in Table 6.4. We will therefore not discuss them further, only summarize the effect of the default value transformations in Section 6.3, by listing their WS-BPEL and Core BPEL syntax summaries.

| **Activity** | **WS-BPEL syntax summary** | **Core BPEL syntax summary** |
|---|---|---|
| `<assign>` | Listing 6.36 | Listing 6.37 |
| `<compensate>` | Listing 6.38 | Listing 6.39 |
| `<compensateScope>` | Listing 6.40 | Listing 6.41 |
| `<empty>` | Listing 6.42 | Listing 6.43 |
| `<exit>` | Listing 6.44 | Listing 6.45 |
| `<extensionActivity>` | Listing 6.46 | Listing 6.47 |
| `<flow>` | Listing 6.48 | Listing 6.49 |
| `<forEach>` | Listing 6.50 | Listing 6.51 |
| `<rethrow>` | Listing 6.62 | Listing 6.63 |
| `<throw>` | Listing 6.67 | Listing 6.68 |
| `<validate>` | Listing 6.69 | Listing 6.70 |
| `<wait>` | Listing 6.71 | Listing 6.72 |
| `<while>` | Listing 6.73 | Listing 6.74 |

Table 6.4: Non-sugared activities.

#### A note about `<rethrow>`

As noted in a draft of the standard specification, `<rethrow>` is essentially a `<throw>` with implicit `faultName` and `faultVariable` attributes:

> “A fault caught by a <catchAll> handler or by a custom fault handler that does not specify a faultName, may need to be rethrown. However the <throw> activity that requires a faultName can not be used here as the faultName is not available. Hence all fault handlers are allowed to rethrow the original fault with a <rethrow> activity that is defined to be an empty element. In essence <rethrow> can be considered a macro for a <throw> that throws the fault caught by the corresponding fault handler.”
> *[16, Sec. 13.4]*

One might therefore wonder, if not `<rethrow>` could somehow be transformed into `<throw>`? As pointed out in *loc. cit.*, the `faultName` is (statically) unavailable inside the `<catchAll>` handler, since this handler may handle different faults at runtime, and therefore we cannot simply replace a `<retrow>` with a `<throw>` in this situation. One might however attempt to transform a `<catchAll>` into a number of `<catch>` handlers, by statically determining which faults could be caught by the `<catchAll>` in question, thereby making the `faultName` explicit. But alas, we cannot know all the possible faults a `<catchAll>` might handle, since the standard specification allows arbitrary platform specific faults:

> “There are various sources of faults in WS-BPEL. A fault response to an `<invoke>` activity is one source of faults, where the fault name and data are based on the definition of the fault in the WSDL operation. A `<throw>` activity is another source, with explicitly given name and/or data. WS-BPEL defines several standard faults with their names, and there may be other platform-specific faults such as communication failures.”
> *[17, Sec. 12.5]*

Thus we cannot in general eliminate the `<catchAll>` construct nor the `<rethrow>` activity.

## 6.6 Extensions

XSLT template: Appendix 6.D.17

WS-BPEL supports extensions that are implementation dependent and they can be either optional or mandatory. In this section we look at how one may syntactically remove the use of unsupported optional extensions.

### 6.6.1 Extension Declarations

Extensions are declared in an `<extensions>` element as part of the `<process>` element. Each `<extension>` declaration must specify whether the extension is mandatory by using the `mustUnderstand` attribute. If an `<extension>` has `mustUnderstand="yes"` we cannot remove the extension declaration or the uses of the extension. But if the `<extension>` has `mustUnderstand="no"`, we may safely remove all uses of the extension, as discussed in the following sections, and thereafter remove the extension declaration itself.

Note that WS-BPEL identifies extensions by their namespace URI, and that the same extension URI may be declared more than once; if just one such declaration is has `mustUnderstand="yes"`, the extension is mandatory:

> "The same extension URI MAY be declared multiple times in the `<extensions>` element. If an extension URI is identified as mandatory in one `<extension>` element and optional in another, then the mandatory semantics have precedence and MUST be enforced." *[17, Sec. 14]*

### 6.6.2 Unsupported Extension Activities

Extension activities, such as vendor-specific activities, can be used by wrapping them in `<extensionActivity>` elements. The standard specification says the following on how to ignore an optional extension activity:

> "If the element contained within the `<extensionActivity>` element is not recognized by the WS-BPEL processor and is not subject to a `mustUnderstand="yes"` requirement from an extension declaration then the unknown activity MUST be treated as if it were an `<empty>` activity that has the standard-attributes and standard-elements of the unrecognized element; all its other attributes and child elements are ignored." *[17, Sec. 10.9]*

In short, the way to ignore an extension activity is to replace it with an `<empty>` activity, with the same standard attributes and elements as the element enclosed in the extension activity. The `<extensionActivity>` element itself cannot have any attributes or elements besides the single enclosed one.

### 6.6.3 Unsupported Extension Assign Operations

The `<assign>` activity is used to copy values between variables. Besides the ordinary `<copy>` element, the standard specification allows the usage of implementation specific assignment operations, wrapped in an `<extensionAssignOperation>` element [17, Sec. 8.4].

> "If the element contained within the `<extensionAssignOperation>` element is not recognized by the WS-BPEL processor and is not subject to a `mustUnderstand="yes"` requirement from an extension declaration then the `<extensionAssignOperation>` operation MUST be ignored." *[17, Sec. 8.4]*

Listing 6.30: An <assign> activity with both an ordinary <copy> and an <extensionAssignOperation>.

```
<assign>
 <copy>
  ...
 </copy>
 <extensionAssignOperation>
  ...
 </extensionAssignOperation>
</assign>
```

Listing 6.31: An <assign> activity consisting only of an <extensionAssignOperation>.

```
<assign>
 <extensionAssignOperation>
  ...
 </extensionAssignOperation>
</assign>
```

Listing 6.32: Activity with ignorable extensions.

```
<wait ext:position='2,5'>
 <until>'2006-11-19T19:50-07:00'</until>
 <ext:icon>clock.png</ext:icon>
</wait>
```

Listing 6.33: After removing extensions.

```
<wait>
 <until>'2006-11-19T19:50-07:00'</until>
</wait>
```

This mean we can safely remove optional <extensionAssignOperation>s from a process description. If an <assign> consists only of optional <extensionAssignOperation>s, as in Listing 6.31, simply removing them will leave an empty, and thus invalid, <assign> activity. In that case, we replace the <assign> with an <empty> activity in the same way as for extension activities.

Besides the standard attributes, an <assign> element may have a `validate` attribute, as follows:

> "The optional `validate` attribute can be used with the <assign> activity. Its default value is `"no"`. When `validate` is set to `"yes"`, the <assign> activity validates all the variables being modified by the activity." *[17, Sec. 8.4]*

In the case where we replace the <assign> activity with an <empty> activity, the content of the original <assign> activity solely consists of ignorable assignment operations — thus, no variable would be modified. Therefore it is safe to disregard the `validate` attribute.

### 6.6.4 Attribute and Element Extensions

Besides the two explicitly extendable elements discussed in the previous sections, WS-BPEL supports extension attributes and elements almost everywhere in a WS-BPEL process:

> "WS-BPEL supports extensibility by allowing namespace-qualified attributes to appear on any WS-BPEL element and by allowing elements from other namespaces to appear within WS-BPEL defined elements." *[17, Sec. 5.3]*

As all extensions must be declared in a namespace other than the WS-BPEL namespace it is easy to identify and remove optional extension attributes and elements: simply leave out all attributes and elements declared in the extension namespace.

Listings 6.32 and 6.33 illustrate this, showing a <wait> activity with and without its ignorable extensions, respectively.

### 6.6.5 Documentation

XSLT template: Appendix 6.D.16

In a sense, documentation is an ignorable extension: `<documentation>` elements are used to place human readable notes in the process description [17, Sec. 5.3] and they are not assigned any semantics by the standard specification and can thus be left out of the description.

## 6.7 Combining the Transformations

We now have a collection of stand-alone transformations, each of which transforms WS-BPEL processes into equivalent WS-BPEL processes with some class of syntactic sugar eliminated. The question is then, how do we combine these transformations, such that we eliminate all syntactic sugar, i.e. we obtain a Core BPEL process? Can we apply the individual transformations sequentially (in some order) or will we have to integrate them into one big transformation?

The challenge comes from the fact that the transformations interfere with each other in the following ways:

**use of syntactic sugar:** some transformations produce constructs that use syntactic sugar, in order to keep them simple and easily readable; for example, several transformations produce `<sequence>`s or rely on default attributes/elements.

**too general:** the transformations that make default attribute values explicit are too general: once the defaults have been made explicit, the attributes are only necessary on the specific elements they relate to. For example, `suppressJoinFailure` only affects activities which have a `<targets>` element (which in Core BPEL, can only be `<flow>`), but the transformation in Section 6.3.1 will make the attribute explicit on all activities and `<process>`.

In the case of the first type of interference, we are in the fortunate situation that there are no cyclic dependencies between the transformations: Figure 6.1 shows that the dependencies between transformations form a DAG. Thus, applying the transformations sequentially according to any topological ordering of that DAG, will result in a WS-BPEL process with no syntactic sugar.

Due to the second type of interference though, we will not obtain a Core BPEL process this way: there will be a number of redundant attributes on various constructs. We see two solutions to this problem:

- we revise the transformations that make defaults explicit, such that they only make defaults explicit on the elements where the attribute value is actually necessary, or
- we simply make a transformation which removes the redundant attributes, which is to be applied after the others. This is sound since the default values for these attributes have been made explicit on the elements where they matter.

We choose the latter option, as it is simple and keeps the other transformations simple and general. The attributes that should be removed are `suppressJoinFailure`, `exitOnStandardFault`, `expressionLanguage`, and `queryLanguage` on `<process>`, and `suppressJoinFailure` on all activities but `<flow>`. The XSLT implementation of this transformation is listed in Appendix 6.D.18.

## 6.8 Conclusions

We have identified a core subset of the WS-BPEL language, named Core BPEL, and have presented a transformation from WS-BPEL to this core language which preserves semantics according to the informal description in the standard specification, and the WSDL interfaces of processes. Core BPEL has fewer constructs and most constructs have fewer syntactical variations, making it more tractable for formal purposes, such as implementation of execution engines or static analysis, but

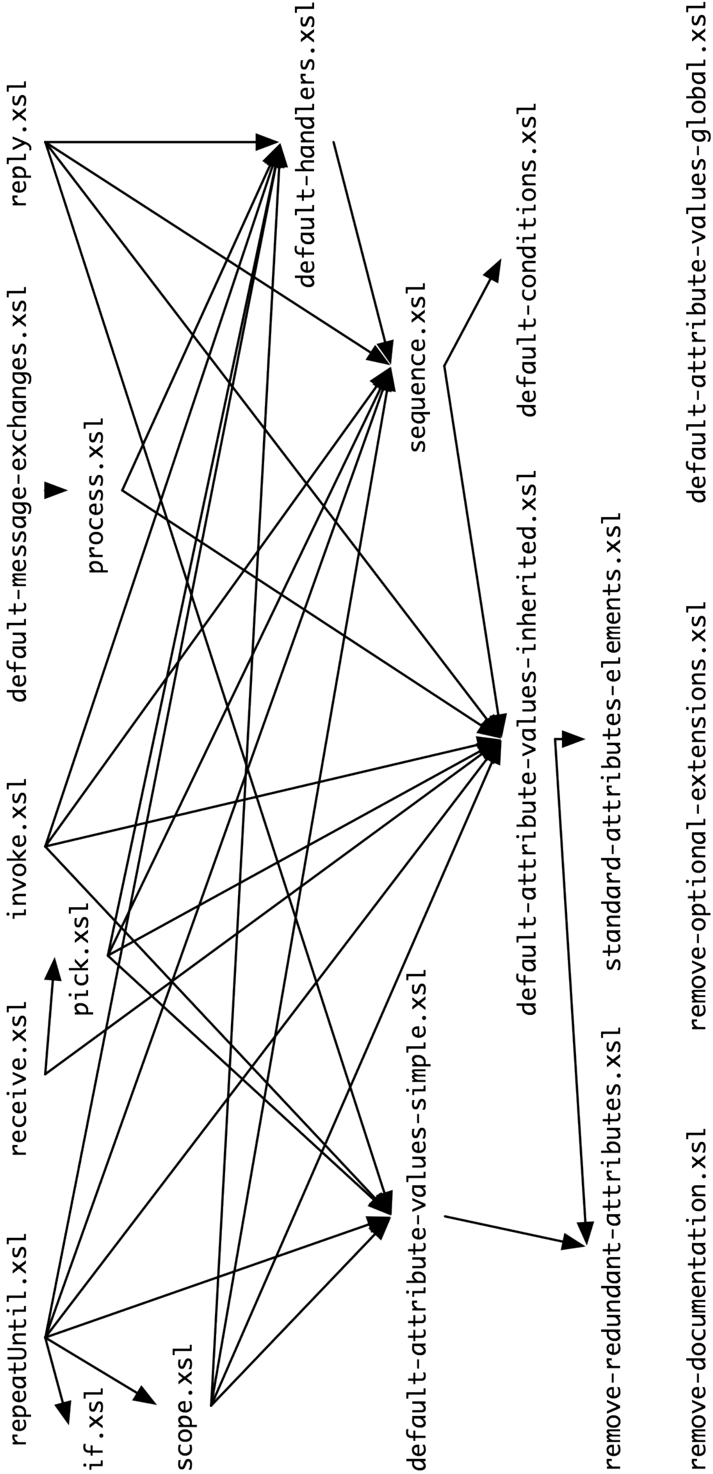



Figure 6.1: Graph showing which transformations must be applied after other transformations: the graph has an edge $a \rightarrow b$ if transformation $a$ creates syntactically sugared elements of a type that is eliminated by transformation $b$.

also easier to understand the features of the language. We have not proved that Core BPEL is minimal, but we are not aware of any further possibilities of reducing or restricting the language syntax.

As part of our analysis, we clarify certain aspects of the syntax of WS-BPEL and provide a more concise presentation of the essential constructs of the language. We therefore hope that this technical report will also prove useful as a cheat sheet for those who are studying the WS-BPEL standard specification.

The transformation from WS-BPEL to Core BPEL consists of a set of independent subtransformations, each responsible for simplifying a specific part of the language. The subtransformations are given in the form of a set of XSLT 1.0 templates, making them easily adoptable by other researchers and WS-BPEL implementers. The XSLT templates are available at the CosmoBiz website `http://www.cosmobiz.com` where we also provide an online tool for transforming WS-BPEL processes into Core BPEL. As each template performs an independent transformation, users are free to use just a subset of the transformations.

### 6.8.1 Future work

There are numerous possible lines of future work. Most importantly, we believe that Core BPEL could serve as a simpler basis for the work the academic community is doing with WS-BPEL; e.g. the numerous WS-BPEL formalizations could probably be simplified, though still remain complete, if they only covered Core BPEL. In the context of the CosmoBiz project, it would be interesting to extend our bigraph formalization of a subset of WS-BPEL [3] to the entire Core BPEL fragment.

Similarly, one might probably simplify implementations of WS-BPEL by only considering Core BPEL; for instance, we expect that Hallwyl's WS-BPEL engine (beepell) [1] can be simplified using this approach. As part of such work, it would also be interesting to investigate if and how this approach would affect performance of implementations.

It would also be interesting to investigate, whether the transformations presented here are semantic preserving with respect to the existing formalizations of WS-BPEL. Similarly, it would be interesting to perform a case study of existing WS-BPEL implementations: do they execute WS-BPEL processes in the same way as their Core BPEL equivalents? In both cases, discrepancies could help pin-point differences in interpretations of the standard as well as errors in our transformations or the formalizations and implementations. This is the natural continuation of Hallwyl's work in his master's thesis [10], where he investigated existing implementations to see if they are consistent in their interpretation of the WS-BPEL standard specification.

We also believe that our approach could be applied to the recent BPMN 2.0 standard [2], with the same benfits as discussed above.

## 6.9 Bibliography

[1] beepell. Webpage. `http://beepell.com/`.

[2] Business process model and notation (BPMN) version 2.0. Technical report, Object Management Group, January 2011.

[3] Mikkel Bundgaard, Arne John Glenstrup, Thomas Hildebrandt, Espen Højsgaard, and Henning Niss. Formalizing WS-BPEL and Higher Order Mobile Embedded Business Processes in the Bigraphical Programming Languages (BPL) Tool. Technical Report TR-2008-103, IT University of Copenhagen, 2008.

[4] CDuce. Webpage. `http://www.cduce.org/`.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. W3C Note, W3C, March 2001.

[6] Dirk Fahland. Complete Abstract Operational Semantics for the Web Service Business Process Execution Language. Technical Report 190, Humboldt-Universität zu Berlin, 2005.

[7] Dirk Fahland and Wolfgang Reisig. ASM-based semantics for BPEL: The negative Control Flow. In Danièle Beauquier, Egon Börger, and Anatol Slissenko, editors, *Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05)*, pages 131–151. Paris XII, March 2005.

[8] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. Specification and validation of the business process execution language for web services. In *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *Lecture Notes in Computer Science*, pages 78–94. Springer Verlag, 2004.

[9] Roozbeh Farahbod, Uwe Glässer, and Mona Vajihollahi. An abstract machine architecture for web service based business process management. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 144–157. Springer Verlag, 2006.

[10] Tim Hallwyl. Evaluating the BPEL standard specification. Master's thesis, Department of Computer Science, University of Copenhagen, May 2008.

[11] Thomas Hildebrandt (principal investigator). Computer supported mobile adaptive business processes (CosmoBiz) research project. Webpage, 2007. `http://www.cosmobiz.org/`.

[12] Niels Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In Marlon Dumas and Reiko Heckel, editors, *Proceedings of the 4th International Workshop on Web Services and Formal Methods (WS-FM'07)*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer Verlag, 2007.

[13] Niels Lohmann, H.M.W. Verbeek, Chun Ouyang, Christian Stahl, and Wil M. P. van der Aalst. Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology, 2007.

[14] Christian Stahl. A Petri net semantics for BPEL. Informatik-Berichte 188, Humboldt-Universität zu Berlin, 2005.

[15] M. Weidlich, G. Decker, and M. Weske. Efficient analysis of bpel 2.0 processes using $\pi$-calculus. In *Asia-Pacific Service Computing Conference, The 2nd IEEE*, pages 266–274, December 2007.

[16] Web services business process execution language version 2.0, working draft. Technical report, OASIS Web Services Business Process Execution Language (WSBPEL) TC, May 2005.

[17] Web services business process execution language version 2.0. Technical report, OASIS Web Services Business Process Execution Language (WSBPEL) TC, April 2007.

[18] XDuce - A Typed XML Processing Language. Webpage. `http://xduce.sourceforge.net/`.

[19] XML Schema part 1: Structures second edition. W3C Recommendation, W3C, 2004.

# 6.A WS-BPEL vs. Core BPEL Syntax Summaries

This section contains syntax summaries for the WS-BPEL process and activity elements side-by-side with the corresponding Core BPEL syntax summaries for easy comparison.

Listing 6.34: WS-BPEL <process>.

```
<process name="NCName"
  targetNamespace="anyURI"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/
      wsbpel/2.0/process/executable">
  <extensions>?
    ...
  </extensions>
  <import namespace="anyURI"?
    location="anyURI"?
    importType="anyURI" />*
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <variables>?
    ...
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>?
    ...
  </faultHandlers>
  <eventHandlers>?
    ...
  </eventHandlers>
  activity
</process>
```

Listing 6.35: Core BPEL <process>.

```
<process name="NCName"
  targetNamespace="anyURI"
  xmlns="http://docs.oasis-open.org/
      wsbpel/2.0/process/executable">
  <extensions>?
    ...
  </extensions>
  <import namespace="anyURI"?
    location="anyURI"?
    importType="anyURI" />*
  activity
</process>
```

Scope-related content of a <process> is moved into a new <scope> inside the <process> and the default attribute values defined on the <process> are made explicit on the relevant elements inside the <process> after which the attributes are removed from the <process> element.

Listing 6.36: WS-BPEL <assign>.

```
<assign validate="yes|no"?
  standard-attributes>
  standard-elements
  (
  <copy keepSrcElementName="yes|no"?
    ignoreMissingFromData="yes|no"?>
    from-spec
    to-spec
  </copy>
  |
  <extensionAssignOperation>
    assign-element-of-other-namespace
  </extensionAssignOperation>
  )+
</assign>
```

Listing 6.37: Core BPEL <assign>.

```
<assign validate="yes|no">
  (
  <copy keepSrcElementName="yes|no"?
    ignoreMissingFromData="yes|no">
    from-spec
    to-spec
  </copy>
  |
  <extensionAssignOperation>
    assign-element-of-other-namespace
  </extensionAssignOperation>
  )+
</assign>
```

Standard attributes and elements are moved to a wrapping <flow> and default attribute values are made explicit.

Listing 6.38: WS-BPEL <compensate>.

```
<compensate standard-attributes>
  standard-elements
</compensate>
```

Listing 6.39: Core BPEL <compensate>.

```
<compensate />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.40: WS-BPEL <compensateScope>.

```
<compensateScope target="NCName"
  standard-attributes>
  standard-elements
</compensateScope>
```

Listing 6.41: Core BPEL <compensateScope>.

```
<compensateScope target="NCName" />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.42: WS-BPEL <empty>.

```
<empty standard-attributes>
  standard-elements
</empty>
```

Listing 6.43: Core BPEL <empty>.

```
<empty />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.44: WS-BPEL <exit>.

```
<exit standard-attributes>
  standard-elements
</exit>
```

Listing 6.45: Core BPEL <exit>.

```
<exit />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.46: WS-BPEL `<extensionActivity>`.

```
<extensionActivity>
  <anyElementQName
    standard-attributes>
    standard-elements
  </anyElementQName>
</extensionActivity>
```

Listing 6.47: Core BPEL `<extensionActivity>`.

```
<extensionActivity>
  <anyElementQName name="NCName" />
</extensionActivity>
```

Standard attributes and elements are moved to a wrapping `<flow>`, except for the `name` attribute.

Listing 6.48: WS-BPEL `<flow>`.

```
<flow standard-attributes>
  standard-elements
  <links>?
    <link name="NCName" />+
  </links>
  activity+
</flow>
```

Listing 6.49: Core BPEL `<flow>`.

```
<flow suppressJoinFailure="yes|no">
  <targets>?
    <joinCondition
      expressionLanguage="anyURI">
      bool-expr
    </joinCondition>
    <target linkName="NCName" />+
  </targets>
  <sources>?
    <source linkName="NCName">+
      <transitionCondition
        expressionLanguage="anyURI">
        bool-expr
      </transitionCondition>
    </source>
  </sources>
  <links>?
    <link name="NCName" />+
  </links>
  activity+
</flow>
```

Core BPEL only allows the WS-BPEL standard attributes and elements on `<flow>` and they are therefore inlined in the Core BPEL syntax summary for `<flow>`. Default attribute values and elements are made explicit.

Listing 6.50: WS-BPEL <forEach>.

```
<forEach counterName="BPELVariableName"
  parallel="yes|no"
  standard-attributes>
  standard-elements
  <startCounterValue
    expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue
    expressionLanguage="anyURI"?>
    unsigned-integer-expression
  </finalCounterValue>
  <completionCondition>?
    <branches
      expressionLanguage="anyURI"?
      successfulBranchesOnly="yes|no"?
        >?
      unsigned-integer-expression
    </branches>
  </completionCondition>
  <scope ...>...</scope>
</forEach>
```

Listing 6.51: Core BPEL <forEach>.

```
<forEach counterName="BPELVariableName"
  parallel="yes|no">
  <startCounterValue
    expressionLanguage="anyURI">
    unsigned-integer-expression
  </startCounterValue>
  <finalCounterValue
    expressionLanguage="anyURI">
    unsigned-integer-expression
  </finalCounterValue>
  <completionCondition>
    <branches
      expressionLanguage="anyURI"
      successfulBranchesOnly="yes|no"
        >?
      unsigned-integer-expression
    </branches>
  </completionCondition>
  <scope ...>...</scope>
</forEach>
```

Standard attributes and elements are moved to a wrapping <flow> and default attribute values are made explicit.

Listing 6.52: WS-BPEL <if>.

```
<if standard-attributes>
  standard-elements
  <condition expressionLanguage="anyURI
     "?>
    bool-expr
  </condition>
  activity
  <elseif>*
    <condition
      expressionLanguage="anyURI"?>
      bool-expr
    </condition>
    activity
  </elseif>
  <else>?
    activity
  </else>
</if>
```

Listing 6.53: Core BPEL <if>.

```
<if>
  <condition expressionLanguage="anyURI
     ">
    bool-expr
  </condition>
  activity
  <else>
    activity
  </else>
</if>
```

<elseif>s are unfolded to <else><if>s, an explicit empty <else> branch is added if lacking, standard attributes and elements are moved to a wrapping <flow>, and default attribute values are made explicit.

Listing 6.54: WS-BPEL <invoke>.

```
<invoke partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  inputVariable="BPELVariableName"?
  outputVariable="BPELVariableName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName"
      initiate="yes|join|no"?
      pattern="request|response|request
          -response"? />+
  </correlations>
  <catch faultName="QName"?
    faultVariable="BPELVariableName"?
    faultMessageType="QName"?
    faultElement="QName"?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
    activity
  </compensationHandler>
  <toParts>?
    <toPart part="NCName"
     fromVariable="BPELVariableName"/>+
  </toParts>
  <fromParts>?
    <fromPart part="NCName"
      toVariable="BPELVariableName"/>+
  </fromParts>
</invoke>
```

Listing 6.55: Core BPEL <invoke>.

```
<invoke partnerLink="NCName"
  operation="NCName"
  inputVariable="BPELVariableName"?
  outputVariable="BPELVariableName"?>
  <correlations>?
    <correlation set="NCName"
      initiate="yes|join|no"
      pattern="request|response|request
          -response"? />+
  </correlations>
</invoke>
```

The superfluous portType attribute is removed, scope-related content as well as implicit temporary variables are moved to a wrapping <scope>, implicit assignments are made explicit, standard attributes and elements are moved to a wrapping <flow>, and default attribute values are made explicit.

Listing 6.56: WS-BPEL <pick>.

```
<pick createInstance="yes|no"?
  standard-attributes>
  standard-elements
  <onMessage partnerLink="NCName"
   portType="QName"?
   operation="NCName"
   variable="BPELVariableName"?
   messageExchange="NCName"?>+
   <correlations>?
     <correlation set="NCName"
        initiate="yes|join|no"? />+
   </correlations>
   <fromParts>?
     <fromPart part="NCName"
       toVariable="BPELVariableName"/>+
   </fromParts>
   activity
  </onMessage>
  <onAlarm>*
   (
   <for expressionLanguage="anyURI"?>
     duration-expr
   </for>
   |
   <until expressionLanguage="anyURI"?>
     deadline-expr
   </until>
   )
   activity
  </onAlarm>
</pick>
```

Listing 6.57: Core BPEL <pick>.

```
<pick createInstance="yes|no">
  <onMessage partnerLink="NCName"
    operation="NCName"
    variable="BPELVariableName"?
    messageExchange="NCName"?>+
    <correlations>?
      <correlation set="NCName"
         initiate="yes|join|no" />+
    </correlations>
    activity
  </onMessage>
  <onAlarm>*
    (
    <for expressionLanguage="anyURI">
      duration-expr
    </for>
    |
    <until expressionLanguage="anyURI">
      deadline-expr
    </until>
    )
    activity
  </onAlarm>
</pick>
```

The superfluous portType attribute is removed, implicit temporary variables are made explicit in a wrapping <scope>, implicit assignments are made explicit, standard attributes and elements are moved to a wrapping <flow>, and default attribute values are made explicit.

Listing 6.58: WS-BPEL <receive>.

```
<receive partnerLink="NCName"
  portType="QName"? operation="NCName"
  variable="BPELVariableName"?
  createInstance="yes|no"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName"
      initiate="yes|join|no"? />+
  </correlations>
  <fromParts>?
    <fromPart part="NCName"
      toVariable="BPELVariableName" />+
  </fromParts>
</receive>
```

<receive> is transformed into a <pick> with a single <onMessage>.

Listing 6.59: WS-BPEL <repeatUntil>.

```
<repeatUntil standard-attributes>
  standard-elements
  activity
  <condition
    expressionLanguage="anyURI"?>
    bool-expr
  </condition>
</repeatUntil>
```

<repeatUntil> is transformed into a <while>.

Listing 6.60: WS-BPEL <reply>.

```
<reply partnerLink="NCName"
  portType="QName"?
  operation="NCName"
  variable="BPELVariableName"?
  faultName="QName"?
  messageExchange="NCName"?
  standard-attributes>
  standard-elements
  <correlations>?
    <correlation
      set="NCName"
      initiate="yes|join|no"? />+
  </correlations>
  <toParts>?
    <toPart
     part="NCName"
     fromVariable="BPELVariableName"/>+
  </toParts>
</reply>
```

Listing 6.61: Core BPEL <reply>.

```
<reply partnerLink="NCName"
  operation="NCName"
  variable="BPELVariableName"?
  faultName="QName"?
  messageExchange="NCName">
  <correlations>?
    <correlation
      set="NCName"
      initiate="yes|join|no" />+
  </correlations>
</reply>
```

The superfluous portType attribute is removed, implicit temporary variables are made explicit in a wrapping <scope>, implicit assignments are made explicit, standard attributes and elements are moved to a wrapping <flow>, and default attribute values are made explicit.

Listing 6.62: WS-BPEL <rethrow>.

```
<rethrow standard-attributes>
  standard-elements
</rethrow>
```

Listing 6.63: Core BPEL <rethrow>.

```
<rethrow />
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.64: WS-BPEL <scope>.

```
<scope isolated="yes|no"?
  exitOnStandardFault="yes|no"?
  standard-attributes>
  standard-elements
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <variables>?
    <variable name="BPELVariableName"
      messageType="QName"?
      type="QName"?
      element="QName"?>+
      from-spec?
    </variable>
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>?
    ...
  </faultHandlers>
  <compensationHandler>?
    ...
  </compensationHandler>
  <terminationHandler>?
    ...
  </terminationHandler>
  <eventHandlers>?
    <onEvent partnerLink="NCName"
      portType="QName"?
      operation="NCName"
      ( messageType="QName"
        | element="QName" )?
      variable="BPELVariableName"?
      messageExchange="NCName"?>*
      <correlations>?
       <correlation set="NCName"
        initiate="yes|join|no"? />+
      </correlations>
      <fromParts>?
       <fromPart part="NCName"
        toVariable="BPELVariableName"/>+
      </fromParts>
      <scope ...>...</scope>
    </onEvent>
    ...
  </eventHandlers>
  activity
</scope>
```

Listing 6.65: Core BPEL <scope>.

```
<scope isolated="yes|no"
  exitOnStandardFault="yes|no"
  name="NCName">
  <partnerLinks>?
    ...
  </partnerLinks>
  <messageExchanges>?
    ...
  </messageExchanges>
  <variables>?
    <variable name="BPELVariableName"
      messageType="QName"?
      type="QName"?
      element="QName"? />+
  </variables>
  <correlationSets>?
    ...
  </correlationSets>
  <faultHandlers>
    ...
  </faultHandlers>
  <compensationHandler>
    ...
  </compensationHandler>
  <terminationHandler>
    ...
  </terminationHandler>
  <eventHandlers>?
    <onEvent partnerLink="NCName"
      operation="NCName"
      messageType="QName"?
      variable="BPELVariableName"?
      messageExchange="NCName">*
      <correlations>?
        <correlation set="NCName"
          initiate="yes|join|no"? />+
      </correlations>
      <scope ...>...</scope>
    </onEvent>
    ...
  </eventHandlers>
  activity
</scope>
```

Variable initializations are made explicit, standard attributes and elements are moved to a wrapping `<flow>` (except `name`), and default attribute values and elements are made explicit.

Additionally, for `<onEvent>`, the superfluous `portType` attribute is removed and implicit temporary variables and assignments are made explicit.

Listing 6.66: WS-BPEL <sequence>.

```
1 <sequence standard-attributes>
2   standard-elements
3   activity+
4 </sequence>
```

<sequence> is transformed into a <flow>.

Listing 6.67: WS-BPEL <throw>.

```
1 <throw faultName="QName"
2   faultVariable="BPELVariableName"?
3   standard-attributes>
4   standard-elements
5 </throw>
```

Listing 6.68: Core BPEL <throw>.

```
1 <throw faultName="QName"
2   faultVariable="BPELVariableName"? />
3
4
5
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.69: WS-BPEL <validate>.

```
1 <validate
2   variables="BPELVariableNames"
3   standard-attributes>
4   standard-elements
5 </validate>
```

Listing 6.70: Core BPEL <validate>.

```
1 <validate
2   variables="BPELVariableNames" />
3
4
5
```

Standard attributes and elements are moved to a wrapping <flow>.

Listing 6.71: WS-BPEL <wait>.

```
1  <wait standard-attributes>
2    standard-elements
3    (
4    <for expressionLanguage="anyURI"?>
5      duration-expr
6    </for>
7    |
8    <until expressionLanguage="anyURI"?>
9      deadline-expr
10   </until>
11   )
12 </wait>
```

Listing 6.72: Core BPEL <wait>.

```
1  <wait>
2    (
3    <for expressionLanguage="anyURI">
4      duration-expr
5    </for>
6    |
7    <until expressionLanguage="anyURI">
8      deadline-expr
9    </until>
10   )
11 </wait>
12
```

Standard attributes and elements are moved to a wrapping <flow> and default attribute values are made explicit.

Listing 6.73: WS-BPEL <while>.

```
1 <while standard-attributes>
2   standard-elements
3   <condition expressionLanguage="anyURI
      "?>
4     bool-expr
5   </condition>
6   activity
7 </while>
```

Listing 6.74: Core BPEL <while>.

```
1 <while>
2   <condition expressionLanguage="anyURI
      ">
3     bool-expr
4   </condition>
5   activity
6 </while>
7
```

Standard attributes and elements are moved to a wrapping <flow> and default attribute values are made explicit.

# 6.B XML Schema for Core BPEL

## 6.B.1 core-bpel.xsd

```
1   <?xml version="1.0" encoding="UTF 8"?>
2   <xsd:schema
3     xmlns="http://docs.oasis open.org/wsbpel/2.0/process/executable"
4     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5     targetNamespace="http://docs.oasis open.org/wsbpel/2.0/process/
          executable"
6     elementFormDefault="qualified" blockDefault="#all">
7     <xsd:annotation>
8       <xsd:documentation>
9         This is the schema for Core BPEL, a subset of WS BPEL.  This
10        Schema is a modified version of the original WS BPEL XML Schema,
11        excluding implicit activities, optional extensions and making
12        default values mandatory.
13      </xsd:documentation>
14    </xsd:annotation>
15    <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
16      schemaLocation="http://www.w3.org/2001/xml.xsd" />
17    <xsd:element name="process" type="tProcess">
18      <xsd:annotation>
19        <xsd:documentation>
20          This is the root element for a Core BPEL process.
21        </xsd:documentation>
22      </xsd:annotation>
23    </xsd:element>
24    <xsd:complexType name="tProcess">
25      <xsd:complexContent>
26        <xsd:extension base="tExtensibleElements">
27          <xsd:sequence>
28            <xsd:element ref="extensions" minOccurs="0" />
29            <xsd:element ref="import" minOccurs="0"
30              maxOccurs="unbounded" />
31            <xsd:group ref="activity" />
32          </xsd:sequence>
33          <xsd:attribute name="name" type="xsd:NCName" use="required" />
34          <xsd:attribute name="targetNamespace" type="xsd:anyURI"
35            use="required" />
36        </xsd:extension>
37      </xsd:complexContent>
38    </xsd:complexType>
39    <xsd:complexType name="tExtensibleElements">
40      <xsd:annotation>
41        <xsd:documentation>
42          This type is extended by other component types to allow
43          elements and attributes from other namespaces to be added at
44          the modeled places.
45        </xsd:documentation>
46      </xsd:annotation>
47      <xsd:sequence>
48        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
49          maxOccurs="unbounded" />
50      </xsd:sequence>
51      <xsd:anyAttribute namespace="##other" processContents="lax" />
52    </xsd:complexType>
53    <xsd:group name="activity">
54      <xsd:annotation>
55        <xsd:documentation>
56          All Core BPEL activities in alphabetical order.
57          Basic activities and structured activities. Addtional
58          constraints:   rethrow activity can be used ONLY within a
59          fault handler (i.e. "catch" and "catchAll" element)
60          compensate or compensateScope activity can be used ONLY within
61          a fault handler, a compensation handler or a termination
62          handler
63        </xsd:documentation>
64      </xsd:annotation>
65      <xsd:choice>
66        <xsd:element ref="assign"/>
67        <xsd:element ref="compensate"/>
68        <xsd:element ref="compensateScope"/>
69        <xsd:element ref="empty"/>
70        <xsd:element ref="exit"/>
71        <xsd:element ref="extensionActivity" />
72        <xsd:element ref="flow"/>
73        <xsd:element ref="forEach"/>
74        <xsd:element ref="if"/>
75        <xsd:element ref="invoke"/>
76        <xsd:element ref="pick"/>
77        <xsd:element ref="reply"/>
78        <xsd:element ref="rethrow"/>
79        <xsd:element ref="scope"/>
80        <xsd:element ref="throw"/>
81        <xsd:element ref="validate"/>
82        <xsd:element ref="wait"/>
83        <xsd:element ref="while" />
84      </xsd:choice>
85    </xsd:group>
86    <xsd:element name="extensions" type="tExtensions" />
87    <xsd:complexType name="tExtensions">
88      <xsd:complexContent>
89        <xsd:extension base="tExtensibleElements">
90          <xsd:sequence>
91            <xsd:element ref="extension" maxOccurs="unbounded" />
92          </xsd:sequence>
93        </xsd:extension>
94      </xsd:complexContent>
95    </xsd:complexType>
96    <xsd:element name="extension" type="tExtension" />
97    <xsd:complexType name="tExtension">
98      <xsd:complexContent>
99        <xsd:extension base="tExtensibleElements">
100         <xsd:attribute name="namespace" type="xsd:anyURI"
101           use="required" />
102         <xsd:attribute name="mustUnderstand" type="tBoolean"
103           fixed="yes" />
104       </xsd:extension>
105     </xsd:complexContent>
106   </xsd:complexType>
107   <xsd:element name="import" type="tImport" />
108   <xsd:complexType name="tImport">
109     <xsd:complexContent>
110       <xsd:extension base="tExtensibleElements">
111         <xsd:attribute name="namespace" type="xsd:anyURI"
112           use="optional" />
113         <xsd:attribute name="location" type="xsd:anyURI"
114           use="optional" />
115         <xsd:attribute name="importType" type="xsd:anyURI"
116           use="required" />
117       </xsd:extension>
118     </xsd:complexContent>
119   </xsd:complexType>
120   <xsd:element name="partnerLinks" type="tPartnerLinks" />
121   <xsd:complexType name="tPartnerLinks">
122     <xsd:complexContent>
123       <xsd:extension base="tExtensibleElements">
124         <xsd:sequence>
125           <xsd:element ref="partnerLink" maxOccurs="unbounded" />
126         </xsd:sequence>
127       </xsd:extension>
128     </xsd:complexContent>
129   </xsd:complexType>
130   <xsd:element name="partnerLink" type="tPartnerLink" />
131   <xsd:complexType name="tPartnerLink">
132     <xsd:complexContent>
```

```
133       <xsd:extension base="tExtensibleElements">
134         <xsd:attribute name="name" type="xsd:NCName" use="required" />
135         <xsd:attribute name="partnerLinkType" type="xsd:QName"
136           use="required" />
137         <xsd:attribute name="myRole" type="xsd:NCName" />
138         <xsd:attribute name="partnerRole" type="xsd:NCName" />
139         <xsd:attribute name="initializePartnerRole" type="tBoolean" />
140       </xsd:extension>
141     </xsd:complexContent>
142   </xsd:complexType>
143   <xsd:element name="messageExchanges" type="tMessageExchanges" />
144   <xsd:complexType name="tMessageExchanges">
145     <xsd:complexContent>
146       <xsd:extension base="tExtensibleElements">
147         <xsd:sequence>
148           <xsd:element ref="messageExchange" maxOccurs="unbounded" />
149         </xsd:sequence>
150       </xsd:extension>
151     </xsd:complexContent>
152   </xsd:complexType>
153   <xsd:element name="messageExchange" type="tMessageExchange" />
154   <xsd:complexType name="tMessageExchange">
155     <xsd:complexContent>
156       <xsd:extension base="tExtensibleElements">
157         <xsd:attribute name="name" type="xsd:NCName" use="required" />
158       </xsd:extension>
159     </xsd:complexContent>
160   </xsd:complexType>
161   <xsd:element name="variables" type="tVariables" />
162   <xsd:complexType name="tVariables">
163     <xsd:complexContent>
164       <xsd:extension base="tExtensibleElements">
165         <xsd:sequence>
166           <xsd:element ref="variable" maxOccurs="unbounded" />
167         </xsd:sequence>
168       </xsd:extension>
169     </xsd:complexContent>
170   </xsd:complexType>
171   <xsd:element name="variable" type="tVariable" />
172   <xsd:complexType name="tVariable">
173     <xsd:complexContent>
174       <xsd:extension base="tExtensibleElements">
175         <xsd:attribute name="name" type="BPELVariableName"
176           use="required" />
177         <xsd:attribute name="messageType" type="xsd:QName"
178           use="optional" />
179         <xsd:attribute name="type" type="xsd:QName" use="optional" />
180         <xsd:attribute name="element" type="xsd:QName" use="optional" />
181       </xsd:extension>
182     </xsd:complexContent>
183   </xsd:complexType>
184   <xsd:simpleType name="BPELVariableName">
185     <xsd:restriction base="xsd:NCName">
186       <xsd:pattern value="[^\.]+" />
187     </xsd:restriction>
188   </xsd:simpleType>
189   <xsd:element name="correlationSets" type="tCorrelationSets" />
190   <xsd:complexType name="tCorrelationSets">
191     <xsd:complexContent>
192       <xsd:extension base="tExtensibleElements">
193         <xsd:sequence>
194           <xsd:element ref="correlationSet" maxOccurs="unbounded" />
195         </xsd:sequence>
196       </xsd:extension>
197     </xsd:complexContent>
198   </xsd:complexType>
199   <xsd:element name="correlationSet" type="tCorrelationSet" />
200   <xsd:complexType name="tCorrelationSet">
201     <xsd:complexContent>
202       <xsd:extension base="tExtensibleElements">
203         <xsd:attribute name="properties" type="QNames" use="required" />
204         <xsd:attribute name="name" type="xsd:NCName" use="required" />
205       </xsd:extension>
206     </xsd:complexContent>
207   </xsd:complexType>
208   <xsd:simpleType name="QNames">
209     <xsd:restriction>
210       <xsd:simpleType>
211         <xsd:list itemType="xsd:QName" />
212       </xsd:simpleType>
213       <xsd:minLength value="1" />
214     </xsd:restriction>
215   </xsd:simpleType>
216   <xsd:element name="faultHandlers" type="tFaultHandlers" />
217   <xsd:complexType name="tFaultHandlers">
218     <xsd:complexContent>
219       <xsd:extension base="tExtensibleElements">
220         <xsd:sequence>
221           <xsd:element ref="catch" minOccurs="0"
222             maxOccurs="unbounded" />
223           <xsd:element ref="catchAll" minOccurs="0" />
224         </xsd:sequence>
225       </xsd:extension>
226     </xsd:complexContent>
227   </xsd:complexType>
228   <xsd:element name="catch" type="tCatch">
229     <xsd:annotation>
230       <xsd:documentation>
231         This element can contain all activities including the
232         activities compensate, compensateScope and rethrow.
233       </xsd:documentation>
234     </xsd:annotation>
235   </xsd:element>
236   <xsd:complexType name="tCatch">
237     <xsd:complexContent>
238       <xsd:extension base="tActivityContainer">
239         <xsd:attribute name="faultName" type="xsd:QName" />
240         <xsd:attribute name="faultVariable" type="BPELVariableName" />
241         <xsd:attribute name="faultMessageType" type="xsd:QName" />
242         <xsd:attribute name="faultElement" type="xsd:QName" />
243       </xsd:extension>
244     </xsd:complexContent>
245   </xsd:complexType>
246   <xsd:element name="catchAll" type="tActivityContainer">
247     <xsd:annotation>
248       <xsd:documentation>
249         This element can contain all activities including the
250         activities compensate, compensateScope and rethrow.
251       </xsd:documentation>
252     </xsd:annotation>
253   </xsd:element>
254   <xsd:complexType name="tActivityContainer">
255     <xsd:complexContent>
256       <xsd:extension base="tExtensibleElements">
257         <xsd:sequence>
258           <xsd:group ref="activity" />
259         </xsd:sequence>
260       </xsd:extension>
261     </xsd:complexContent>
262   </xsd:complexType>
263   <xsd:element name="eventHandlers" type="tEventHandlers" />
264   <xsd:complexType name="tEventHandlers">
265     <xsd:annotation>
266       <xsd:documentation>
267         XSD Authors: The child element onAlarm needs to be a Local
268         Element Declaration, because there is another onAlarm element
269         defined for the pick activity.
270       </xsd:documentation>
271     </xsd:annotation>
272     <xsd:complexContent>
```

```
273       <xsd:extension base="tExtensibleElements">
274         <xsd:sequence>
275           <xsd:element ref="onEvent" minOccurs="0"
276             maxOccurs="unbounded" />
277           <xsd:element name="onAlarm" type="tOnAlarmEvent"
278             minOccurs="0" maxOccurs="unbounded" />
279         </xsd:sequence>
280       </xsd:extension>
281     </xsd:complexContent>
282   </xsd:complexType>
283   <xsd:element name="onEvent" type="tOnEvent" />
284   <xsd:complexType name="tOnEvent">
285     <xsd:complexContent>
286       <xsd:extension base="tOnMsgCommon">
287         <xsd:sequence>
288           <xsd:element ref="scope" />
289         </xsd:sequence>
290         <xsd:attribute name="messageType" type="xsd:QName"
291           use="optional" />
292       </xsd:extension>
293     </xsd:complexContent>
294   </xsd:complexType>
295   <xsd:complexType name="tOnMsgCommon">
296     <xsd:annotation>
297       <xsd:documentation>
298         XSD Authors: The child element correlations needs to be a
299         Local Element Declaration, because there is another
300         correlations element defined for the invoke activity.
301       </xsd:documentation>
302     </xsd:annotation>
303     <xsd:complexContent>
304       <xsd:extension base="tExtensibleElements">
305         <xsd:sequence>
306           <xsd:element name="correlations" type="tCorrelations"
307             minOccurs="0" />
308         </xsd:sequence>
309         <xsd:attribute name="partnerLink" type="xsd:NCName"
310           use="required" />
311         <xsd:attribute name="operation" type="xsd:NCName"
312           use="required" />
313         <xsd:attribute name="messageExchange" type="xsd:NCName"
314           use="required" />
315         <xsd:attribute name="variable" type="BPELVariableName"
316           use="optional" />
317       </xsd:extension>
318     </xsd:complexContent>
319   </xsd:complexType>
320   <xsd:complexType name="tCorrelations">
321     <xsd:annotation>
322       <xsd:documentation>
323         XSD Authors: The child element correlation needs to be a Local
324         Element Declaration, because there is another correlation
325         element defined for the invoke activity.
326       </xsd:documentation>
327     </xsd:annotation>
328     <xsd:complexContent>
329       <xsd:extension base="tExtensibleElements">
330         <xsd:sequence>
331           <xsd:element name="correlation" type="tCorrelation"
332             maxOccurs="unbounded" />
333         </xsd:sequence>
334       </xsd:extension>
335     </xsd:complexContent>
336   </xsd:complexType>
337   <xsd:complexType name="tCorrelation">
338     <xsd:complexContent>
339       <xsd:extension base="tExtensibleElements">
340         <xsd:attribute name="set" type="xsd:NCName" use="required" />
341         <xsd:attribute name="initiate" type="tInitiate" use="required" />
342       </xsd:extension>
343     </xsd:complexContent>
344   </xsd:complexType>
345   <xsd:simpleType name="tInitiate">
346     <xsd:restriction base="xsd:string">
347       <xsd:enumeration value="yes" />
348       <xsd:enumeration value="join" />
349       <xsd:enumeration value="no" />
350     </xsd:restriction>
351   </xsd:simpleType>
352   <xsd:complexType name="tOnAlarmEvent">
353     <xsd:complexContent>
354       <xsd:extension base="tExtensibleElements">
355         <xsd:sequence>
356           <xsd:choice>
357             <xsd:sequence>
358               <xsd:group ref="forOrUntilGroup" />
359               <xsd:element ref="repeatEvery" minOccurs="0" />
360             </xsd:sequence>
361             <xsd:element ref="repeatEvery" />
362           </xsd:choice>
363           <xsd:element ref="scope" />
364         </xsd:sequence>
365       </xsd:extension>
366     </xsd:complexContent>
367   </xsd:complexType>
368   <xsd:group name="forOrUntilGroup">
369     <xsd:choice>
370       <xsd:element ref="for" />
371       <xsd:element ref="until" />
372     </xsd:choice>
373   </xsd:group>
374   <xsd:element name="for" type="tDuration expr" />
375   <xsd:element name="until" type="tDeadline expr" />
376   <xsd:element name="repeatEvery" type="tDuration expr" />
377   <xsd:complexType name="tActivity">
378     <xsd:complexContent>
379       <xsd:extension base="tExtensibleElements">
380       </xsd:extension>
381     </xsd:complexContent>
382   </xsd:complexType>
383   <xsd:element name="targets" type="tTargets" />
384   <xsd:complexType name="tTargets">
385     <xsd:complexContent>
386       <xsd:extension base="tExtensibleElements">
387         <xsd:sequence>
388           <xsd:element ref="joinCondition" minOccurs="1" />
389           <xsd:element ref="target" maxOccurs="unbounded" />
390         </xsd:sequence>
391       </xsd:extension>
392     </xsd:complexContent>
393   </xsd:complexType>
394   <xsd:element name="joinCondition" type="tCondition" />
395   <xsd:element name="target" type="tTarget" />
396   <xsd:complexType name="tTarget">
397     <xsd:complexContent>
398       <xsd:extension base="tExtensibleElements">
399         <xsd:attribute name="linkName" type="xsd:NCName"
400           use="required" />
401       </xsd:extension>
402     </xsd:complexContent>
403   </xsd:complexType>
404   <xsd:element name="sources" type="tSources" />
405   <xsd:complexType name="tSources">
406     <xsd:complexContent>
407       <xsd:extension base="tExtensibleElements">
408         <xsd:sequence>
409           <xsd:element ref="source" maxOccurs="unbounded" />
410         </xsd:sequence>
411       </xsd:extension>
412     </xsd:complexContent>
```

```
413 </xsd:complexType>
414 <xsd:element name="source" type="tSource" />
415 <xsd:complexType name="tSource">
416   <xsd:complexContent>
417     <xsd:extension base="tExtensibleElements">
418       <xsd:sequence>
419         <xsd:element ref="transitionCondition" minOccurs="1" />
420       </xsd:sequence>
421       <xsd:attribute name="linkName" type="xsd:NCName"
422         use="required" />
423     </xsd:extension>
424   </xsd:complexContent>
425 </xsd:complexType>
426 <xsd:element name="transitionCondition" type="tCondition" />
427 <xsd:element name="assign" type="tAssign" />
428 <xsd:complexType name="tAssign">
429   <xsd:complexContent>
430     <xsd:extension base="tActivity">
431       <xsd:sequence>
432         <xsd:choice maxOccurs="unbounded">
433           <xsd:element ref="copy" />
434           <xsd:element ref="extensionAssignOperation" />
435         </xsd:choice>
436       </xsd:sequence>
437       <xsd:attribute name="validate" type="tBoolean" use="required" />
438     </xsd:extension>
439   </xsd:complexContent>
440 </xsd:complexType>
441 <xsd:element name="copy" type="tCopy" />
442 <xsd:complexType name="tCopy">
443   <xsd:complexContent>
444     <xsd:extension base="tExtensibleElements">
445       <xsd:sequence>
446         <xsd:element ref="from" />
447         <xsd:element ref="to" />
448       </xsd:sequence>
449       <xsd:attribute name="keepSrcElementName" type="tBoolean"
450         use="optional" />
451       <xsd:attribute name="ignoreMissingFromData" type="tBoolean"
452         use="required" />
453     </xsd:extension>
454   </xsd:complexContent>
455 </xsd:complexType>
456 <xsd:element name="from" type="tFrom" />
457 <xsd:complexType name="tFrom" mixed="true">
458   <xsd:sequence>
459     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
460       maxOccurs="unbounded" />
461     <xsd:choice minOccurs="0">
462       <xsd:element ref="literal" />
463       <xsd:element ref="query" />
464     </xsd:choice>
465   </xsd:sequence>
466   <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
467   <xsd:attribute name="variable" type="BPELVariableName" />
468   <xsd:attribute name="part" type="xsd:NCName" />
469   <xsd:attribute name="property" type="xsd:QName" />
470   <xsd:attribute name="partnerLink" type="xsd:NCName" />
471   <xsd:attribute name="endpointReference" type="tRoles" />
472   <xsd:anyAttribute namespace="##other" processContents="lax" />
473 </xsd:complexType>
474 <xsd:element name="literal" type="tLiteral" />
475 <xsd:complexType name="tLiteral" mixed="true">
476   <xsd:sequence>
477     <xsd:any namespace="##any" processContents="lax" minOccurs="0" />
478   </xsd:sequence>
479 </xsd:complexType>
480 <xsd:element name="query" type="tQuery" />
481 <xsd:complexType name="tQuery" mixed="true">
482   <xsd:sequence>
483     <xsd:any processContents="lax" minOccurs="0"
484       maxOccurs="unbounded" />
485   </xsd:sequence>
486   <xsd:attribute name="queryLanguage" type="xsd:anyURI" />
487   <xsd:anyAttribute namespace="##other" processContents="lax" />
488 </xsd:complexType>
489 <xsd:simpleType name="tRoles">
490   <xsd:restriction base="xsd:string">
491     <xsd:enumeration value="myRole" />
492     <xsd:enumeration value="partnerRole" />
493   </xsd:restriction>
494 </xsd:simpleType>
495 <xsd:element name="to" type="tTo" />
496 <xsd:complexType name="tTo" mixed="true">
497   <xsd:sequence>
498     <xsd:any namespace="##other" processContents="lax" minOccurs="0"
499       maxOccurs="unbounded" />
500     <xsd:element ref="query" minOccurs="0" />
501   </xsd:sequence>
502   <xsd:attribute name="expressionLanguage" type="xsd:anyURI" />
503   <xsd:attribute name="variable" type="BPELVariableName" />
504   <xsd:attribute name="part" type="xsd:NCName" />
505   <xsd:attribute name="property" type="xsd:QName" />
506   <xsd:attribute name="partnerLink" type="xsd:NCName" />
507   <xsd:anyAttribute namespace="##other" processContents="lax" />
508 </xsd:complexType>
509 <xsd:element name="extensionAssignOperation"
510   type="tExtensionAssignOperation" />
511 <xsd:complexType name="tExtensionAssignOperation">
512   <xsd:complexContent>
513     <xsd:extension base="tExtensibleElements" />
514   </xsd:complexContent>
515 </xsd:complexType>
516 <xsd:element name="compensate" type="tCompensate" />
517 <xsd:complexType name="tCompensate">
518   <xsd:complexContent>
519     <xsd:extension base="tActivity" />
520   </xsd:complexContent>
521 </xsd:complexType>
522 <xsd:element name="compensateScope" type="tCompensateScope" />
523 <xsd:complexType name="tCompensateScope">
524   <xsd:complexContent>
525     <xsd:extension base="tActivity">
526       <xsd:attribute name="target" type="xsd:NCName" use="required" />
527     </xsd:extension>
528   </xsd:complexContent>
529 </xsd:complexType>
530 <xsd:element name="empty" type="tEmpty" />
531 <xsd:complexType name="tEmpty">
532   <xsd:complexContent>
533     <xsd:extension base="tActivity" />
534   </xsd:complexContent>
535 </xsd:complexType>
536 <xsd:element name="exit" type="tExit" />
537 <xsd:complexType name="tExit">
538   <xsd:complexContent>
539     <xsd:extension base="tActivity" />
540   </xsd:complexContent>
541 </xsd:complexType>
542 <xsd:element name="extensionActivity" type="tExtensionActivity" />
543 <xsd:complexType name="tExtensionActivity">
544   <xsd:sequence>
545     <xsd:any namespace="##other" processContents="lax" />
546   </xsd:sequence>
547 </xsd:complexType>
548 <xsd:element name="flow" type="tFlow" />
549 <xsd:complexType name="tFlow">
550   <xsd:complexContent>
551     <xsd:extension base="tActivity">
552       <xsd:sequence>
```

```
553            <xsd:element ref="targets" minOccurs="0" />
554            <xsd:element ref="sources" minOccurs="0" />
555            <xsd:element ref="links" minOccurs="0" />
556            <xsd:group ref="activity" maxOccurs="unbounded" />
557          </xsd:sequence>
558          <xsd:attribute name="suppressJoinFailure" type="tBoolean" use="
               required"/>
559        </xsd:extension>
560      </xsd:complexContent>
561    </xsd:complexType>
562    <xsd:element name="links" type="tLinks" />
563    <xsd:complexType name="tLinks">
564      <xsd:complexContent>
565        <xsd:extension base="tExtensibleElements">
566          <xsd:sequence>
567            <xsd:element ref="link" maxOccurs="unbounded" />
568          </xsd:sequence>
569        </xsd:extension>
570      </xsd:complexContent>
571    </xsd:complexType>
572    <xsd:element name="link" type="tLink" />
573    <xsd:complexType name="tLink">
574      <xsd:complexContent>
575        <xsd:extension base="tExtensibleElements">
576          <xsd:attribute name="name" type="xsd:NCName" use="required" />
577        </xsd:extension>
578      </xsd:complexContent>
579    </xsd:complexType>
580    <xsd:element name="forEach" type="tForEach" />
581    <xsd:complexType name="tForEach">
582      <xsd:complexContent>
583        <xsd:extension base="tActivity">
584          <xsd:sequence>
585            <xsd:element ref="startCounterValue" />
586            <xsd:element ref="finalCounterValue" />
587            <xsd:element ref="completionCondition" minOccurs="0" />
588            <xsd:element ref="scope" />
589          </xsd:sequence>
590          <xsd:attribute name="counterName" type="BPELVariableName"
591            use="required" />
592          <xsd:attribute name="parallel" type="tBoolean" use="required" />
593        </xsd:extension>
594      </xsd:complexContent>
595    </xsd:complexType>
596    <xsd:element name="startCounterValue" type="tExpression" />
597    <xsd:element name="finalCounterValue" type="tExpression" />
598    <xsd:element name="completionCondition" type="tCompletionCondition" />
599    <xsd:complexType name="tCompletionCondition">
600      <xsd:complexContent>
601        <xsd:extension base="tExtensibleElements">
602          <xsd:sequence>
603            <xsd:element ref="branches" minOccurs="0" />
604          </xsd:sequence>
605        </xsd:extension>
606      </xsd:complexContent>
607    </xsd:complexType>
608    <xsd:element name="branches" type="tBranches" />
609    <xsd:complexType name="tBranches">
610      <xsd:complexContent>
611        <xsd:extension base="tExpression">
612          <xsd:attribute name="successfulBranchesOnly" type="tBoolean"
613            use="required" />
614        </xsd:extension>
615      </xsd:complexContent>
616    </xsd:complexType>
617    <xsd:element name="if" type="tIf" />
618    <xsd:complexType name="tIf">
619      <xsd:complexContent>
620        <xsd:extension base="tActivity">
621          <xsd:sequence>
622            <xsd:element ref="condition" />
623            <xsd:group ref="activity" />
624            <xsd:element ref="else" minOccurs="1" />
625          </xsd:sequence>
626        </xsd:extension>
627      </xsd:complexContent>
628    </xsd:complexType>
629    <xsd:element name="else" type="tActivityContainer" />
630    <xsd:element name="invoke" type="tInvoke" />
631    <xsd:complexType name="tInvoke">
632      <xsd:annotation>
633        <xsd:documentation>
634          XSD Authors: The child element correlations needs to be a
635          Local Element Declaration, because there is another
636          correlations element defined for the non invoke activities.
637        </xsd:documentation>
638      </xsd:annotation>
639      <xsd:complexContent>
640        <xsd:extension base="tActivity">
641          <xsd:sequence>
642            <xsd:element name="correlations"
643              type="tCorrelationsWithPattern" minOccurs="0" />
644          </xsd:sequence>
645          <xsd:attribute name="partnerLink" type="xsd:NCName"
646            use="required" />
647          <xsd:attribute name="operation" type="xsd:NCName"
648            use="required" />
649          <xsd:attribute name="inputVariable" type="BPELVariableName"
650            use="optional" />
651          <xsd:attribute name="outputVariable" type="BPELVariableName"
652            use="optional" />
653        </xsd:extension>
654      </xsd:complexContent>
655    </xsd:complexType>
656    <xsd:complexType name="tCorrelationsWithPattern">
657      <xsd:annotation>
658        <xsd:documentation>
659          XSD Authors: The child element correlation needs to be a Local
660          Element Declaration, because there is another correlation
661          element defined for the non invoke activities.
662        </xsd:documentation>
663      </xsd:annotation>
664      <xsd:complexContent>
665        <xsd:extension base="tExtensibleElements">
666          <xsd:sequence>
667            <xsd:element name="correlation"
668              type="tCorrelationWithPattern" maxOccurs="unbounded" />
669          </xsd:sequence>
670        </xsd:extension>
671      </xsd:complexContent>
672    </xsd:complexType>
673    <xsd:complexType name="tCorrelationWithPattern">
674      <xsd:complexContent>
675        <xsd:extension base="tCorrelation">
676          <xsd:attribute name="pattern" type="tPattern" />
677        </xsd:extension>
678      </xsd:complexContent>
679    </xsd:complexType>
680    <xsd:simpleType name="tPattern">
681      <xsd:restriction base="xsd:string">
682        <xsd:enumeration value="request" />
683        <xsd:enumeration value="response" />
684        <xsd:enumeration value="request response" />
685      </xsd:restriction>
686    </xsd:simpleType>
687
688    <xsd:element name="pick" type="tPick" />
689    <xsd:complexType name="tPick">
690      <xsd:annotation>
691        <xsd:documentation>
```

```
692         XSD Authors: The child element onAlarm needs to be a Local
693         Element Declaration, because there is another onAlarm element
694         defined for event handlers.
695       </xsd:documentation>
696     </xsd:annotation>
697     <xsd:complexContent>
698       <xsd:extension base="tActivity">
699         <xsd:sequence>
700           <xsd:element ref="onMessage" maxOccurs="unbounded" />
701           <xsd:element name="onAlarm" type="tOnAlarmPick"
702             minOccurs="0" maxOccurs="unbounded" />
703         </xsd:sequence>
704         <xsd:attribute name="createInstance" type="tBoolean"
705           use="required" />
706       </xsd:extension>
707     </xsd:complexContent>
708   </xsd:complexType>
709
710   <xsd:element name="onMessage" type="tOnMessage" />
711   <xsd:complexType name="tOnMessage">
712     <xsd:complexContent>
713       <xsd:extension base="tOnMsgCommon">
714         <xsd:sequence>
715           <xsd:group ref="activity" />
716         </xsd:sequence>
717       </xsd:extension>
718     </xsd:complexContent>
719   </xsd:complexType>
720   <xsd:complexType name="tOnAlarmPick">
721     <xsd:complexContent>
722       <xsd:extension base="tExtensibleElements">
723         <xsd:sequence>
724           <xsd:group ref="forOrUntilGroup" />
725           <xsd:group ref="activity" />
726         </xsd:sequence>
727       </xsd:extension>
728     </xsd:complexContent>
729   </xsd:complexType>
730   <xsd:element name="reply" type="tReply" />
731   <xsd:complexType name="tReply">
732     <xsd:annotation>
733       <xsd:documentation>
734         XSD Authors: The child element correlations needs to be a
735         Local Element Declaration, because there is another
736         correlations element defined for the invoke activity.
737       </xsd:documentation>
738     </xsd:annotation>
739     <xsd:complexContent>
740       <xsd:extension base="tActivity">
741         <xsd:sequence>
742           <xsd:element name="correlations" type="tCorrelations"
743             minOccurs="0" />
744         </xsd:sequence>
745         <xsd:attribute name="partnerLink" type="xsd:NCName"
746           use="required" />
747         <xsd:attribute name="operation" type="xsd:NCName"
748           use="required" />
749         <xsd:attribute name="variable" type="BPELVariableName"
750           use="optional" />
751         <xsd:attribute name="faultName" type="xsd:QName" />
752         <xsd:attribute name="messageExchange" type="xsd:NCName"
753           use="required" />
754       </xsd:extension>
755     </xsd:complexContent>
756   </xsd:complexType>
757   <xsd:element name="rethrow" type="tRethrow" />
758   <xsd:complexType name="tRethrow">
759     <xsd:complexContent>
760       <xsd:extension base="tActivity" />
761     </xsd:complexContent>
762   </xsd:complexType>
763   <xsd:element name="scope" type="tScope" />
764   <xsd:complexType name="tScope">
765     <xsd:complexContent>
766       <xsd:extension base="tActivity">
767         <xsd:sequence>
768           <xsd:element ref="partnerLinks" minOccurs="0" />
769           <xsd:element ref="messageExchanges" minOccurs="0" />
770           <xsd:element ref="variables" minOccurs="0" />
771           <xsd:element ref="correlationSets" minOccurs="0" />
772           <xsd:element ref="faultHandlers" minOccurs="1" />
773           <xsd:element ref="compensationHandler" minOccurs="1" />
774           <xsd:element ref="terminationHandler" minOccurs="1" />
775           <xsd:element ref="eventHandlers" minOccurs="0" />
776           <xsd:group ref="activity" />
777         </xsd:sequence>
778         <xsd:attribute name="name" type="xsd:NCName" use="required" />
779         <xsd:attribute name="isolated" type="tBoolean" use="required" />
780         <xsd:attribute name="exitOnStandardFault" type="tBoolean" use="
              required" />
781       </xsd:extension>
782     </xsd:complexContent>
783   </xsd:complexType>
784   <xsd:element name="compensationHandler" type="tActivityContainer">
785     <xsd:annotation>
786       <xsd:documentation>
787         This element can contain all activities including the
788         activities compensate and compensateScope.
789       </xsd:documentation>
790     </xsd:annotation>
791   </xsd:element>
792   <xsd:element name="terminationHandler" type="tActivityContainer">
793     <xsd:annotation>
794       <xsd:documentation>
795         This element can contain all activities including the
796         activities compensate and compensateScope.
797       </xsd:documentation>
798     </xsd:annotation>
799   </xsd:element>
800   <xsd:element name="throw" type="tThrow" />
801   <xsd:complexType name="tThrow">
802     <xsd:complexContent>
803       <xsd:extension base="tActivity">
804         <xsd:attribute name="faultName" type="xsd:QName"
805           use="required" />
806         <xsd:attribute name="faultVariable" type="BPELVariableName" />
807       </xsd:extension>
808     </xsd:complexContent>
809   </xsd:complexType>
810   <xsd:element name="validate" type="tValidate" />
811   <xsd:complexType name="tValidate">
812     <xsd:complexContent>
813       <xsd:extension base="tActivity">
814         <xsd:attribute name="variables" type="BPELVariableNames"
815           use="required" />
816       </xsd:extension>
817     </xsd:complexContent>
818   </xsd:complexType>
819   <xsd:simpleType name="BPELVariableNames">
820     <xsd:restriction>
821       <xsd:simpleType>
822         <xsd:list itemType="BPELVariableName" />
823       </xsd:simpleType>
824       <xsd:minLength value="1" />
825     </xsd:restriction>
826   </xsd:simpleType>
827   <xsd:element name="wait" type="tWait" />
828   <xsd:complexType name="tWait">
829     <xsd:complexContent>
830       <xsd:extension base="tActivity">
```

```
831          <xsd:choice>
832            <xsd:element ref="for" />
833            <xsd:element ref="until" />
834          </xsd:choice>
835        </xsd:extension>
836      </xsd:complexContent>
837    </xsd:complexType>
838    <xsd:element name="while" type="tWhile" />
839    <xsd:complexType name="tWhile">
840      <xsd:complexContent>
841        <xsd:extension base="tActivity">
842          <xsd:sequence>
843            <xsd:element ref="condition" />
844            <xsd:group ref="activity" />
845          </xsd:sequence>
846        </xsd:extension>
847      </xsd:complexContent>
848    </xsd:complexType>
849    <xsd:complexType name="tExpression" mixed="true">
850      <xsd:sequence>
851        <xsd:any processContents="lax" minOccurs="0"
852          maxOccurs="unbounded" />
853      </xsd:sequence>
854      <xsd:attribute name="expressionLanguage" type="xsd:anyURI"
855        use="required" />
856      <xsd:anyAttribute namespace="##other" processContents="lax" />
857    </xsd:complexType>
858    <xsd:complexType name="tCondition" mixed="true">
859      <xsd:complexContent mixed="true">
860        <xsd:extension base="tExpression" />
861      </xsd:complexContent>
862    </xsd:complexType>
863    <xsd:element name="condition" type="tBoolean expr" />
864    <xsd:complexType name="tBoolean expr" mixed="true">
865      <xsd:complexContent mixed="true">
866        <xsd:extension base="tExpression" />
867      </xsd:complexContent>
868    </xsd:complexType>
869    <xsd:complexType name="tDuration expr" mixed="true">
870      <xsd:complexContent mixed="true">
871        <xsd:extension base="tExpression" />
872      </xsd:complexContent>
873    </xsd:complexType>
874    <xsd:complexType name="tDeadline expr" mixed="true">
875      <xsd:complexContent mixed="true">
876        <xsd:extension base="tExpression" />
877      </xsd:complexContent>
878    </xsd:complexType>
879    <xsd:simpleType name="tBoolean">
880      <xsd:restriction base="xsd:string">
881        <xsd:enumeration value="yes" />
882        <xsd:enumeration value="no" />
883      </xsd:restriction>
884    </xsd:simpleType>
885  </xsd:schema>
```

## 6.C Transformation Example

This appendix shows how an example WS-BPEL process, cf. Section 6.C.1, looks after being transformed by our transformations, cf. Section 6.C.2. The WSDL interface for the processes is listed in Section 6.C.3.

### 6.C.1 echo.bpel

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<process
  name="echoService"
  targetNamespace="http://beepell.com/samples/echo"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:srv="http://beepell.com/samples/echo/definitions"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <import
    namespace="http://beepell.com/samples/echo/definitions"
    location="./echo.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />

  <partnerLinks>
    <partnerLink
      name="echoPartnerLink"
      partnerLinkType="srv:echoPartnerLinkType"
      myRole="echoServiceProvider" />
  </partnerLinks>

  <variables>
    <variable
      name="message"
      messageType="srv:echoMessage" />
  </variables>

  <sequence>
    <receive name="receive"
      partnerLink="echoPartnerLink"
      operation="echoOperation"
      createInstance="yes"
      variable="message" />
    <reply
      partnerLink="echoPartnerLink"
      operation="echoOperation"
      variable="message" />
  </sequence>
</process>
```

### 6.C.2 echo.cbpel

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<bpel:process
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  name="echoService"
  targetNamespace="http://beepell.com/samples/echo">

  <import
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:srv="http://beepell.com/samples/echo/definitions"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    importType="http://schemas.xmlsoap.org/wsdl/"
    location="./echo.wsdl"
    namespace="http://beepell.com/samples/echo/definitions"/>

  <bpel:scope
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:srv="http://beepell.com/samples/echo/definitions"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    exitOnStandardFault="no"
    isolated="no"
    name="echoService">

```

```
24      <partnerLinks>
25        <partnerLink
26          myRole="echoServiceProvider"
27          name="echoPartnerLink"
28          partnerLinkType="srv:echoPartnerLinkType"/>
29      </partnerLinks>
30
31      <bpel:messageExchanges>
32        <bpel:messageExchange
33          name="fresh-prefix-0N65537DefaultMessageExchange"
              />
34      </bpel:messageExchanges>
35
36      <variables>
37        <variable messageType="srv:echoMessage" name="
              message"/>
38      </variables>
39
40      <bpel:faultHandlers>
41        <bpel:catchAll>
42          <bpel:flow
43            name="fresh-prefix-0N65572FreshActivityName"
44            suppressJoinFailure="no">
45
46            <bpel:links>
47              <bpel:link name="fresh-prefix-0
                  N65570FreshSequenceLink"/>
48            </bpel:links>
49
50            <bpel:flow
51              name="fresh-prefix-0N65577FreshActivityName"
52              suppressJoinFailure="no">
53              <bpel:sources>
54                <bpel:source linkName="fresh-prefix-0
                    N65570FreshSequenceLink">
55                  <bpel:transitionCondition
56                    expressionLanguage="urn:oasis:names:tc:
                        wsbpel:2.0:sublang:xpath1.0">
57                    true()
58                  </bpel:transitionCondition>
59                </bpel:source>
60              </bpel:sources>
61              <bpel:compensate/>
62            </bpel:flow>
63
64            <bpel:flow
65              name="fresh-prefix-0N65587FreshActivityName"
66              suppressJoinFailure="no">
67              <bpel:targets>
68                <bpel:joinCondition
69                  expressionLanguage="urn:oasis:names:tc:
                      wsbpel:2.0:sublang:xpath1.0">
70                  $fresh-prefix-0N65570FreshSequenceLink
71                </bpel:joinCondition>
72                <bpel:target
73                  linkName="fresh-prefix-0
                      N65570FreshSequenceLink"/>
74              </bpel:targets>
75              <bpel:rethrow/>
76            </bpel:flow>
77          </bpel:flow>
78        </bpel:catchAll>
79      </bpel:faultHandlers>
80
81      <bpel:compensationHandler>
82        <bpel:compensate/>
83      </bpel:compensationHandler>
84
85      <bpel:terminationHandler>
86        <bpel:compensate/>
87      </bpel:terminationHandler>
88
89      <bpel:flow
90        name="fresh-prefix-0N65603FreshActivityName"
91        suppressJoinFailure="no">
92
93        <bpel:links>
94          <bpel:link name="fresh-prefix-0
              N65578FreshSequenceLink"/>
95        </bpel:links>
96
97        <bpel:flow
98          name="fresh-prefix-0N65608FreshActivityName"
99          suppressJoinFailure="no">
100         <bpel:sources>
101           <bpel:source
102             linkName="fresh-prefix-0
                  N65578FreshSequenceLink">
```

```
103            <bpel:transitionCondition
104              expressionLanguage="urn:oasis:names:tc:
                     wsbpel:2.0:sublang:xpath1.0">
105              true()
106            </bpel:transitionCondition>
107          </bpel:source>
108        </bpel:sources>
109
110        <bpel:flow
111          name="receive" suppressJoinFailure="no">
112          <bpel:pick createInstance="yes">
113            <bpel:onMessage
114              messageExchange="fresh-prefix-0
                     N65537DefaultMessageExchange"
115              operation="echoOperation"
116              partnerLink="echoPartnerLink"
117              variable="message">
118              <bpel:empty/>
119            </bpel:onMessage>
120          </bpel:pick>
121        </bpel:flow>
122      </bpel:flow>
123
124      <bpel:flow
125        name="fresh-prefix-0N65627FreshActivityName"
126        suppressJoinFailure="no">
127        <bpel:targets>
128          <bpel:joinCondition
129            expressionLanguage="urn:oasis:names:tc:wsbpel
                   :2.0:sublang:xpath1.0">
130            $fresh-prefix-0N65578FreshSequenceLink
131          </bpel:joinCondition>
132          <bpel:target linkName="fresh-prefix-0
                 N65578FreshSequenceLink"/>
133        </bpel:targets>
134
135        <reply
136          messageExchange="fresh-prefix-0
                 N65537DefaultMessageExchange"
137          operation="echoOperation"
138            partnerLink="echoPartnerLink"
139            variable="message"/>
140        </bpel:flow>
141      </bpel:flow>
142    </bpel:scope>
143  </bpel:process>
```

### 6.C.3 echo.wsdl

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <wsdl:definitions
4    targetNamespace="http://beepell.com/samples/echo/
         definitions"
5    xmlns:srv="http://beepell.com/samples/proxy/definitions"
6    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/
         plnktype"
7    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
8    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
9
10   <plnk:partnerLinkType name="echoPartnerLinkType">
11     <plnk:role name="echoServiceProvider" portType="srv:
           echoPortType" />
12   </plnk:partnerLinkType>
13
14   <wsdl:message name="echoMessage">
15     <wsdl:part name="content" type="xsd:string" />
16   </wsdl:message>
17
18   <wsdl:portType name="echoPortType">
19     <wsdl:operation name="echoOperation">
20       <wsdl:input message="srv:echoMessage" />
21       <wsdl:output message="srv:echoMessage" />
22     </wsdl:operation>
23   </wsdl:portType>
24
25 </wsdl:definitions>
```

# 6.D XSLT Transformations

## 6.D.1 Overview

| | File name | Description |
|---|---|---|
| 6.D.2 | `constants.xsl` | This is a template library with various constants. |
| 6.D.3 | `default-attribute-values-global.xsl` | Make the global default attribute values explicit. Attributes: `expressionLanguage`, `queryLanguage`. |
| 6.D.4 | `default-attribute-values-inherited.xsl` | Make the inherited default attribute values explicit. Attributes: `suppressJoinFailure`, `exitOnStandardFault`. |
| 6.D.5 | `default-attribute-values-simple.xsl` | Make the simple default attribute values explicit. Attributes: `createInstance`, `ignoreMissingFromData`, `initiate`, `isolated`, `succesfulBranchesOnly`, `validate`. |
| 6.D.6 | `default-conditions.xsl` | Make the default join, transition, and completion conditions explicit. |
| 6.D.7 | `default-handlers.xsl` | Make the default fault, compensation, and termination handlers explicit. |
| 6.D.8 | `default-message-exchanges.xsl` | Add default `<messageExchange>` to `<process>`, the immediate `<scope>` of `<onEvent>`, and parallel `<forEach>`. Also, make the use of these default message exchanges explicit. |
| 6.D.9 | `fresh-names.xsl` | Utility templates for generating fresh names. |
| 6.D.10 | `if.xsl` | Tranform `<elseif>`s into `<else><if>`s and add empty branches to the `<if>`s that lack them. |
| 6.D.11 | `invoke.xsl` | Move the `<scope>`-parts of an `<invoke>` into an explicit enclosing `<scope>`. Also, make temporary variables and assignments, due to the use of `<toParts>`, `<fromParts>`, and/or references to element variables, explicit. |
| 6.D.12 | `onEvent.xsl` | |

Make temporary variables and assignments, due to the use of `<fromParts>`, and/or references to element variables in `<onEvent>`s, explicit.

### 6.D.13 pick.xsl

Make temporary variables and assignments, due to the use of `<fromParts>`, and/or references to element variables in `<onMessage>`s, explicit.

### 6.D.14 process.xsl

Move the scope-parts of a `<process>` into an explicit `<scope>`.

### 6.D.15 receive.xsl

Transform `<receive>` into `<pick>`.

### 6.D.16 remove-documentation.xsl

Remove all documentation elements.

### 6.D.17 remove-optional-extensions.xsl

Remove all optional extensions and their declarations.

### 6.D.18 remove-redundant-attributes.xsl

Remove the redundant attributes (only sound when default attribute values have been made explicit).

### 6.D.19 repeatUntil.xsl

Transform `<repeatUntil>` into `<while>`.

### 6.D.20 reply.xsl

Make temporary variables and assignments, due to the use of `<toParts>`, and/or references to an element variable, explicit.

### 6.D.21 scope.xsl

Make variable initialization explicit in all scopes (including `<process>`) and make implicit variables and assignments in `<onEvent>`s explicit.

### 6.D.22 sequence.xsl

Transform `<sequence>`s into `<flow>`s.

### 6.D.23 standard-attributes-elements.xsl

Move `<targets>`, `<sources>`, and `suppressJoinFailure` from activities to a new wrapping `<flow>`, except for activities that have no `<targets>` or `<sources>`, where we push the value of that attribute to all the child activities. Names are also moved from activities (except `<scope>`s) to a new wrapping `<flow>` and fresh names are added to unnamed `<flow>`s and `<scope>`s.

### 6.D.24 to-from-parts-element-variables.xsl

Utility templates to make temporary variables and assignments, due to the use of `<toParts>`, `<fromParts>`, and/or references to element variables, explicit.

## 6.D.2 constants.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  This is a template library with various constants.  >
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9    <xsl:variable name="xpathURN">urn:oasis:names:tc:wsbpel:2.0:sublang:
         xpath1.0</xsl:variable>
10   <xsl:variable name="WS BPEL namespace URI">http://docs.oasis open.org/
         wsbpel/2.0/process/executable</xsl:variable>
11
12    <!  Postfixes for fresh names in different contexts.  >
13   <xsl:variable name="default message exchange postfix">
         DefaultMessageExchange</xsl:variable>
14   <xsl:variable name="fresh activity name postfix">FreshActivityName</xsl
         :variable>
15   <xsl:variable name="fresh sequence link postfix">FreshSequenceLink</xsl
         :variable>
16   <xsl:variable name="tmp condition variable postfix">TmpConditionVar</
         xsl:variable>
17   <xsl:variable name="tmp input message variable postfix">
         TmpInputMessageVar</xsl:variable>
18   <xsl:variable name="tmp output message variable postfix">
         TmpOutputMessageVar</xsl:variable>
19
20 </xsl:stylesheet>
```

## 6.D.3 default-attribute-values-global.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Make the global default attribute values explicit.
4       Attributes: expressionLanguage, queryLanguage  >
5
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:include href="constants.xsl" />
13
14   <xsl:template match="*">
15     <xsl:param name="expressionLanguage" />
16     <xsl:param name="queryLanguage" />
17     <xsl:copy>
18       <xsl:copy of select="@*" />
19       <xsl:apply templates>
20         <xsl:with param name="expressionLanguage" select="
               $expressionLanguage" />
21         <xsl:with param name="queryLanguage" select="$queryLanguage" />
22       </xsl:apply templates>
23     </xsl:copy>
24   </xsl:template>
25
26   <xsl:template match="bpel:process">
27     <xsl:copy>
28       <xsl:copy of select="@*" />
29       <xsl:apply templates>
30         <xsl:with param name="expressionLanguage">
31           <xsl:choose>
32             <xsl:when test="@expressionLanguage">
33               <xsl:value of select="@expressionLanguage" />
34             </xsl:when>
35             <xsl:otherwise>
36               <xsl:value of select="$xpathURN" />
37             </xsl:otherwise>
38           </xsl:choose>
39         </xsl:with param>
40         <xsl:with param name="queryLanguage">
41           <xsl:choose>
42             <xsl:when test="@queryLanguage">
43               <xsl:value of select="@queryLanguage" />
44             </xsl:when>
45             <xsl:otherwise>
46               <xsl:value of select="$xpathURN" />
47             </xsl:otherwise>
48           </xsl:choose>
49         </xsl:with param>
50       </xsl:apply templates>
51     </xsl:copy>
52 </xsl:template>
53
54  <!  Adding missing expressionLanguage attributes  >
55 <xsl:template match="bpel:branches | bpel:condition | bpel:
       finalCounterValue | bpel:for | bpel:from[text()] | bpel:
       joinCondition | bpel:repeatEvery | bpel:startCounterValue | bpel:
       to[text()] | bpel:transitionCondition | bpel:until">
56   <xsl:param name="expressionLanguage" />
57   <xsl:param name="queryLanguage" />
58   <xsl:copy>
59     <xsl:copy of select="@*" />
60
61      <!  if expressionLanguage attribute is missing add it  >
62     <xsl:if test="not(@expressionLanguage)">
63       <xsl:attribute name="expressionLanguage">
64         <xsl:value of select="$expressionLanguage" />
65       </xsl:attribute>
66     </xsl:if>
67
68     <xsl:apply templates>
69       <xsl:with param name="expressionLanguage" select="
             $expressionLanguage" />
70       <xsl:with param name="queryLanguage" select="$queryLanguage" />
71     </xsl:apply templates>
72   </xsl:copy>
73 </xsl:template>
74
75  <!  Adding missing queryLangugue attributes  >
76 <xsl:template match="bpel:query">
77   <xsl:param name="expressionLanguage" />
78   <xsl:param name="queryLanguage" />
79   <xsl:copy>
80     <xsl:copy of select="@*" />
81
82      <!  if queryLanguage attribute is missing add it  >
83     <xsl:if test="not(@queryLanguage)">
84       <xsl:attribute name="queryLanguage">
85         <xsl:value of select="$queryLanguage" />
86       </xsl:attribute>
87     </xsl:if>
88
89     <xsl:apply templates>
90       <xsl:with param name="expressionLanguage" select="
             $expressionLanguage" />
91       <xsl:with param name="queryLanguage" select="$queryLanguage" />
92     </xsl:apply templates>
93   </xsl:copy>
94 </xsl:template>
95
```

```
96 </xsl:stylesheet>
```

### 6.D.4 default-attribute-values-inherited.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3 <!  Make the inherited default attribute values explicit.
4     Attributes: suppressJoinFailure, exitOnStandardFault  >
5
6 <xsl:stylesheet version="1.0"
7   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
9
10  <xsl:output indent="yes" method="xml" />
11
12  <xsl:template match="bpel:process">
13    <xsl:copy>
14      <xsl:copy of select="@*" />
15      <xsl:apply templates>
16        <xsl:with param name="exitOnStandardFault">
17          <xsl:choose>
18            <xsl:when test="@exitOnStandardFault">
19              <xsl:value of select="@exitOnStandardFault" />
20            </xsl:when>
21            <xsl:otherwise>
22              <xsl:value of select="'no'" />
23            </xsl:otherwise>
24          </xsl:choose>
25        </xsl:with param>
26        <xsl:with param name="suppressJoinFailure">
27          <xsl:choose>
28            <xsl:when test="@suppressJoinFailure">
29              <xsl:value of select="@suppressJoinFailure" />
30            </xsl:when>
31            <xsl:otherwise>
32              <xsl:value of select="'no'" />
33            </xsl:otherwise>
34          </xsl:choose>
35        </xsl:with param>
36      </xsl:apply templates>
37    </xsl:copy>
38  </xsl:template>
39
40  <xsl:template match="bpel:scope">
41    <xsl:param name="exitOnStandardFault" />
42    <xsl:param name="suppressJoinFailure" />
43
44    <xsl:copy>
45      <xsl:copy of select="@*" />
46
47      <xsl:if test="not(@exitOnStandardFault)">
48        <xsl:attribute name="exitOnStandardFault">
49          <xsl:value of select="$exitOnStandardFault" />
50        </xsl:attribute>
51      </xsl:if>
52
53      <xsl:if test="not(@suppressJoinFailure)">
54        <xsl:attribute name="suppressJoinFailure">
55          <xsl:value of select="$suppressJoinFailure" />
56        </xsl:attribute>
57      </xsl:if>
58
59      <xsl:apply templates>
60        <xsl:with param name="exitOnStandardFault">
61          <xsl:choose>
62            <xsl:when test="@exitOnStandardFault">
63              <xsl:value of select="@exitOnStandardFault" />
64            </xsl:when>
65            <xsl:otherwise>
66              <xsl:value of select="$exitOnStandardFault" />
67            </xsl:otherwise>
68          </xsl:choose>
69        </xsl:with param>
70        <xsl:with param name="suppressJoinFailure">
71          <xsl:choose>
72            <xsl:when test="@suppressJoinFailure">
73              <xsl:value of select="@suppressJoinFailure" />
74            </xsl:when>
75            <xsl:otherwise>
76              <xsl:value of select="$suppressJoinFailure" />
77            </xsl:otherwise>
78          </xsl:choose>
79        </xsl:with param>
80      </xsl:apply templates>
81    </xsl:copy>
82  </xsl:template>
83
84  <xsl:template match="bpel:assign | bpel:compensate | bpel:
        compensateScope | bpel:empty | bpel:exit | bpel:extensionActivity
        /* | bpel:flow | bpel:forEach | bpel:if | bpel:invoke | bpel:pick
        | bpel:receive | bpel:repeatUntil | bpel:reply | bpel:rethrow |
        bpel:sequence | bpel:throw | bpel:validate | bpel:wait | bpel:
        while">
85    <xsl:param name="exitOnStandardFault" />
86    <xsl:param name="suppressJoinFailure" />
87
88    <xsl:copy>
89      <xsl:copy of select="@*" />
90      <xsl:if test="not(@suppressJoinFailure)">
91        <xsl:attribute name="suppressJoinFailure">
92          <xsl:value of select="$suppressJoinFailure" />
93        </xsl:attribute>
94      </xsl:if>
95
96      <xsl:apply templates>
97        <xsl:with param name="exitOnStandardFault" select="
            $exitOnStandardFault" />
98        <xsl:with param name="suppressJoinFailure">
99          <xsl:choose>
100           <xsl:when test="@suppressJoinFailure">
101             <xsl:value of select="@suppressJoinFailure" />
102           </xsl:when>
103           <xsl:otherwise>
104             <xsl:value of select="$suppressJoinFailure" />
105           </xsl:otherwise>
106         </xsl:choose>
107       </xsl:with param>
108     </xsl:apply templates>
109   </xsl:copy>
110 </xsl:template>
111
112 <xsl:template match="*">
113   <xsl:param name="exitOnStandardFault" />
114   <xsl:param name="suppressJoinFailure" />
115   <xsl:copy>
116     <xsl:copy of select="@*" />
117     <xsl:apply templates>
118       <xsl:with param name="exitOnStandardFault" select="
            $exitOnStandardFault" />
119       <xsl:with param name="suppressJoinFailure" select="
            $suppressJoinFailure" />
120     </xsl:apply templates>
121   </xsl:copy>
122 </xsl:template>
123
```

```
124 </xsl:stylesheet>
```

### 6.D.5 default-attribute-values-simple.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Make the simple default attribute values explicit.
4        Attributes: createInstance, ignoreMissingFromData, initiate,
           isolated, succesfulBranchesOnly, validate   >
5
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:template match="*">
13     <xsl:copy>
14       <xsl:copy of select="@*" />
15       <xsl:apply templates />
16     </xsl:copy>
17   </xsl:template>
18
19   <xsl:template match="bpel:pick[not(@createInstance)] | bpel:receive[not
         (@createInstance)]">
20     <xsl:copy>
21       <xsl:copy of select="@*" />
22       <xsl:attribute name="createInstance">no</xsl:attribute>
23       <xsl:apply templates />
24     </xsl:copy>
25   </xsl:template>
26
27   <xsl:template match="bpel:copy[not(@ignoreMissingFromData)]">
28     <xsl:copy>
29       <xsl:copy of select="@*" />
30       <xsl:attribute name="ignoreMissingFromData">no</xsl:attribute>
31       <xsl:apply templates />
32     </xsl:copy>
33   </xsl:template>
34
35   <xsl:template match="bpel:correlation[not(@initiate)]">
36     <xsl:copy>
37       <xsl:copy of select="@*" />
38       <xsl:attribute name="initiate">no</xsl:attribute>
39       <xsl:apply templates />
40     </xsl:copy>
41   </xsl:template>
42
43   <xsl:template match="bpel:scope[not(@isolated)]">
44     <xsl:copy>
45       <xsl:copy of select="@*" />
46       <xsl:attribute name="isolated">no</xsl:attribute>
47       <xsl:apply templates />
48     </xsl:copy>
49   </xsl:template>
50
51   <xsl:template match="bpel:branches[not(@successfulBranchesOnly)]">
52     <xsl:copy>
53       <xsl:copy of select="@*" />
54       <xsl:attribute name="successfulBranchesOnly">no</xsl:attribute>
55       <xsl:apply templates />
56     </xsl:copy>
57   </xsl:template>
58
59   <xsl:template match="bpel:assign[not(@validate)]">
60     <xsl:copy>
61       <xsl:copy of select="@*" />
62       <xsl:attribute name="validate">no</xsl:attribute>
63       <xsl:apply templates />
64     </xsl:copy>
65   </xsl:template>
66
67 </xsl:stylesheet>
```

### 6.D.6 default-conditions.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Make the default join, transition, and completion conditions
        explicit.   >
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9    <xsl:output indent="yes" method="xml" />
10
11   <xsl:include href="constants.xsl" />
12
13   <xsl:template match="*">
14     <xsl:copy>
15       <xsl:copy of select="@*" />
16       <xsl:apply templates />
17     </xsl:copy>
18   </xsl:template>
19
20   <xsl:template match="bpel:source[not(bpel:transitionCondition)]">
21     <xsl:copy>
22       <xsl:copy of select="@*" />
23       <bpel:transitionCondition expressionLanguage="{$xpathURN}">true()</
             bpel:transitionCondition>
24     </xsl:copy>
25   </xsl:template>
26
27   <xsl:template match="bpel:targets[not(bpel:joinCondition)]">
28     <xsl:copy>
29       <xsl:copy of select="@*" />
30       <bpel:joinCondition expressionLanguage="{$xpathURN}">
31         <xsl:for each select="bpel:target">
32           <xsl:value of select="concat('$', @linkName)"/>
33           <xsl:if test="following sibling::bpel:target"> or </xsl:if>
34         </xsl:for each>
35       </bpel:joinCondition>
36       <xsl:apply templates />
37     </xsl:copy>
38   </xsl:template>
39
40    <!  Add empty completion condition where missing   >
41   <xsl:template match="bpel:forEach[not(bpel:completionCondition)]">
42     <xsl:copy>
43       <xsl:copy of select="@*" />
44       <xsl:apply templates />
45       <bpel:completionCondition />
46     </xsl:copy>
47   </xsl:template>
48
49 </xsl:stylesheet>
```

### 6.D.7 default-handlers.xsl

```
<?xml version="1.0" encoding="ISO 8859 1"?>

 <!  Make the default fault , compensation, and termination handlers
      explicit.  >

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy of select="@*" />
      <xsl:apply templates />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:faultHandler[not(bpel:catchAll)]">
    <xsl:copy>
      <xsl:apply templates select="bpel:catch" />
      <bpel:catchAll>
        <bpel:sequence>
          <bpel:compensate />
          <bpel:rethrow />
        </bpel:sequence>
      </bpel:catchAll>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="bpel:scope[not(bpel:faultHandlers) or not(bpel:
      compensationHandler) or not(bpel:terminationHandler)]">
    <xsl:copy>
      <xsl:copy of select="@*" />
      <xsl:apply templates select="bpel:targets" />
      <xsl:apply templates select="bpel:sources" />
      <xsl:apply templates select="bpel:partnerLinks" />
      <xsl:apply templates select="bpel:messageExchanges" />
      <xsl:apply templates select="bpel:variables" />
      <xsl:apply templates select="bpel:correlationSets" />

      <xsl:apply templates select="bpel:faultHandlers" />
      <xsl:if test="not(bpel:faultHandlers)">
        <bpel:faultHandlers>
          <bpel:catchAll>
            <bpel:sequence>
              <bpel:compensate />
              <bpel:rethrow />
            </bpel:sequence>
          </bpel:catchAll>
        </bpel:faultHandlers>
      </xsl:if>

      <xsl:apply templates select="bpel:compensationHandler" />
      <xsl:if test="not(bpel:compensationHandler)">
        <bpel:compensationHandler>
          <bpel:compensate />
        </bpel:compensationHandler>
      </xsl:if>

      <xsl:apply templates select="bpel:terminationHandler" />
      <xsl:if test="not(bpel:terminationHandler)">
        <bpel:terminationHandler>
          <bpel:compensate />
        </bpel:terminationHandler>
      </xsl:if>

      <xsl:apply templates select="bpel:eventHandlers" />

      <xsl:apply templates select="*[not(
          self::bpel:targets or
          self::bpel:sources or
          self::bpel:partnerLinks or
          self::bpel:messageExchanges or
          self::bpel:variables or
          self::bpel:candrelationSets or
          self::bpel:faultHandlers or
          self::bpel:compensationHandler or
          self::bpel:terminationHandler or
          self::bpel:eventHandlers
        )]" />
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

### 6.D.8 default-message-exchanges.xsl

```
<?xml version="1.0" encoding="ISO 8859 1"?>

 <!  Add default <messageExchange> to <process>, the immediate <scope> of
      <onEvent>, and parallel <forEach>  >
 <!  Also, make the use of these default message exchanges explicit  >

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">

  <xsl:output indent="yes" method="xml" />

  <xsl:include href="constants.xsl" />
  <xsl:include href="fresh names.xsl" />

  <xsl:template match="*">
    <xsl:copy>
      <xsl:copy of select="@*" />
      <xsl:apply templates />
    </xsl:copy>
  </xsl:template>

   <!  Find out whether the default message of the given activity is ever
        used.  >
  <xsl:template name="is default message exchange used">
     <!  Activity should be either a <process>, <onEvent>, or <forEach
          parallel="yes">.  >
    <xsl:param name="activity" select="ancestor or self::*[self::bpel:
          process or self::bpel:onEvent or self::bpel:forEach[@parallel='
          yes']][1]" />

     <!  Does this activity or one of its descendants, which are _not_
          inside
          another scope with a default message exchange, use the default
               message
          exchange?  >
    <xsl:value of select="0 &lt; count($activity/descendant or self::*
                                    [(self::bpel:reply or self::bpel
                                          :receive or self::bpel:
                                          onMessage or self::bpel:
                                          onEvent) and
                                     not(@messageExchange) and
                                     ancestor or self::*[self::bpel:
                                           process or self::bpel:
                                           onEvent or self::bpel:
                                           forEach[@parallel='yes
                                           ']][1] = $activity])" />
```

```
34  </xsl:template>
35
36    <! Construct the default message exchange name for the given scope
37       (which defaults to the closest enclosing scope/process).  >
38  <xsl:template name="default message exchange name">
39    <xsl:param name="scope" select="ancestor or self::*[self::bpel:
          process or self::bpel:scope[parent::bpel:onEvent or parent::bpel
          :forEach[@parallel='yes']]][1]" />
40    <xsl:call template name="unique element name">
41      <xsl:with param name="element" select="$scope" />
42      <xsl:with param name="postfix" select="$default message exchange
          postfix" />
43    </xsl:call template>
44  </xsl:template>
45
46    <! Create the default message exchange for the given scope
47       (which defaults to the closest enclosing scope/process).  >
48  <xsl:template name="default message exchange">
49    <xsl:param name="scope" select="ancestor or self::*[self::bpel:
          process or self::bpel:scope[parent::bpel:onEvent or parent::bpel
          :forEach[@parallel='yes']]][1]" />
50    <bpel:messageExchange>
51      <xsl:attribute name="name">
52        <xsl:call template name="default message exchange name">
53          <xsl:with param name="scope" select="$scope" />
54        </xsl:call template>
55      </xsl:attribute>
56    </bpel:messageExchange>
57  </xsl:template>
58
59    <! Add default message exchanges to the relevant elements that
         already have an <messageExchanges> element  >
60  <xsl:template match="bpel:process/bpel:messageExchanges | bpel:onEvent/
       bpel:scope/bpel:messageExchanges | bpel:forEach[@parallel='yes']/
       bpel:scope/bpel:messageExchanges">
61    <xsl:variable name="is default message exchange used">
62      <xsl:choose>
63        <xsl:when test="parent::bpel:process">
64          <xsl:call template name="is default message exchange used">
65            <xsl:with param name="activity" select=".." />
66          </xsl:call template>
67        </xsl:when>
68        <xsl:otherwise>
69          <xsl:call template name="is default message exchange used">
70            <xsl:with param name="activity" select="../.." />
71          </xsl:call template>
72        </xsl:otherwise>
73      </xsl:choose>
74    </xsl:variable>
75
76    <xsl:copy>
77      <xsl:copy of select="@*" />
78      <xsl:if test="$is default message exchange used = 'true'">
79        <xsl:call template name="default message exchange" />
80      </xsl:if>
81      <xsl:apply templates />
82    </xsl:copy>
83  </xsl:template>
84
85    <! Add <messageExchanges> elements and default message exchanges to
         the
86       relevant elements that does not already have an <messageExchanges>
           element  >
87  <xsl:template match="bpel:process[not(bpel:messageExchanges)]">
88    <xsl:variable name="is default message exchange used">
89      <xsl:call template name="is default message exchange used">
90        <xsl:with param name="activity" select="." />
91      </xsl:call template>
92    </xsl:variable>
93
94    <xsl:copy>
95      <xsl:copy of select="@*" />
96      <xsl:copy of select="bpel:extensions" />
97      <xsl:copy of select="bpel:import" />
98      <xsl:copy of select="bpel:partnerLinks" />
99
100     <xsl:if test="$is default message exchange used = 'true'">
101       <bpel:messageExchanges>
102         <xsl:call template name="default message exchange" />
103       </bpel:messageExchanges>
104     </xsl:if>
105
106     <xsl:apply templates select="*[not(
107       self::bpel:extensions or
108       self::bpel:import or
109       self::bpel:partnerLinks or
110       self::bpel:messageExchanges
111       )]" />
112   </xsl:copy>
113 </xsl:template>
114 <xsl:template match="bpel:onEvent/bpel:scope[not(bpel:messageExchanges)
       ] | bpel:forEach[@parallel='yes']/bpel:scope[not(bpel:
       messageExchanges)]">
115   <xsl:variable name="is default message exchange used">
116     <xsl:call template name="is default message exchange used">
117       <xsl:with param name="activity" select=".." />
118     </xsl:call template>
119   </xsl:variable>
120
121   <xsl:copy>
122     <xsl:copy of select="@*" />
123     <xsl:copy of select="bpel:partnerLinks" />
124
125     <xsl:if test="$is default message exchange used = 'true'">
126       <bpel:messageExchanges>
127         <xsl:call template name="default message exchange" />
128       </bpel:messageExchanges>
129     </xsl:if>
130
131     <xsl:apply templates select="*[not(
132       self::bpel:partnerLinks or
133       self::bpel:messageExchanges
134       )]" />
135   </xsl:copy>
136 </xsl:template>
137
138   <! Make the messageExchange attribute explicit.  >
139 <xsl:template match="bpel:onMessage[not(@messageExchange)] | bpel:
       receive[not(@messageExchange)] | bpel:reply[not(@messageExchange)]
       ">
140   <xsl:copy>
141     <xsl:attribute name="messageExchange">
142       <xsl:call template name="default message exchange name" />
143     </xsl:attribute>
144     <xsl:copy of select="@*" />
145     <xsl:apply templates />
146   </xsl:copy>
147 </xsl:template>
148 <xsl:template match="bpel:onEvent[not(@messageExchange)]">
149   <xsl:copy>
150     <xsl:attribute name="messageExchange">
151       <xsl:call template name="default message exchange name">
152         <xsl:with param name="scope" select="bpel:scope" />
153       </xsl:call template>
154     </xsl:attribute>
155     <xsl:copy of select="@*" />
156     <xsl:apply templates />
157   </xsl:copy>
158 </xsl:template>
159
```

```
160 </xsl:stylesheet>
```

## 6.D.9 fresh-names.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Utility templates for generating fresh names.  >
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
7
8    <xsl:param name="freshPrefix" />
9
10   <xsl:template name="unique element name">
11     <xsl:param name="element" select="." />
12     <xsl:param name="postfix" select="''" />
13     <xsl:value of select="$freshPrefix" />
14     <xsl:value of select="generate id($element)" />
15     <xsl:value of select="$postfix" />
16   </xsl:template>
17
18   <xsl:template name="attribute with unique element name">
19     <xsl:param name="attributeName" />
20     <xsl:param name="element" select="." />
21     <xsl:param name="postfix" select="''" />
22     <xsl:attribute name="{$attributeName}">
23       <xsl:call template name="unique element name">
24         <xsl:with param name="element" select="$element" />
25         <xsl:with param name="postfix" select="$postfix" />
26       </xsl:call template>
27     </xsl:attribute>
28   </xsl:template>
29
30 </xsl:stylesheet>
```

## 6.D.10 if.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Tranform <elseif>s into <else><if>s and add empty branches to the <
       if>s
4      that lack them.  >
5
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:template match="*">
13     <xsl:copy>
14       <xsl:copy of select="@*" />
15       <xsl:apply templates />
16     </xsl:copy>
17   </xsl:template>
18
19   <xsl:template match="bpel:elseif">
20     <xsl:param name="if" />
21     <xsl:param name="count" />
22     <xsl:param name="index" />
23
24     <xsl:choose>
25       <xsl:when test="$index &lt; $count">
26         <bpel:else>
27           <bpel:if>
28             <xsl:copy of select="*" />
29             <xsl:apply templates select="$if/bpel:elseif[$index + 1]">
30               <xsl:with param name="if" select="$if"/>
31               <xsl:with param name="count" select="$count"/>
32               <xsl:with param name="index" select="$index + 1"/>
33             </xsl:apply templates>
34           </bpel:if>
35         </bpel:else>
36       </xsl:when>
37
38       <xsl:otherwise>
39         <bpel:else>
40           <bpel:if>
41             <xsl:copy of select="*" />
42
43             <xsl:choose>
44               <xsl:when test="$if/bpel:else">
45                 <xsl:copy of select="$if/bpel:else" />
46               </xsl:when>
47               <xsl:otherwise>
48                 <bpel:else>
49                   <bpel:empty/>
50                 </bpel:else>
51               </xsl:otherwise>
52             </xsl:choose>
53
54           </bpel:if>
55         </bpel:else>
56       </xsl:otherwise>
57     </xsl:choose>
58
59   </xsl:template>
60
61
62   <xsl:template match="bpel:if[bpel:elseif]">
63     <xsl:copy>
64       <xsl:copy of select="@*" />
65       <xsl:copy of select="bpel:condition" />
66       <xsl:apply templates select="*[not(self::bpel:condition | self::
             bpel:else | self::bpel:elseif)]" />
67       <xsl:apply templates select="bpel:elseif[1]">
68         <xsl:with param name="if" select="self::node()"/>
69         <xsl:with param name="count" select="count(bpel:elseif)"/>
70         <xsl:with param name="index" select="1"/>
71       </xsl:apply templates>
72     </xsl:copy>
73   </xsl:template>
74
75
76   <xsl:template match="bpel:if[not(bpel:else | bpel:elseif)]">
77     <xsl:copy>
78       <xsl:copy of select="@*" />
79       <xsl:apply templates />
80       <bpel:else>
81         <bpel:empty />
82       </bpel:else>
83     </xsl:copy>
84   </xsl:template>
85
86 </xsl:stylesheet>
```

## 6.D.11 invoke.xsl

```
1   <?xml version="1.0" encoding="ISO 8859 1"?>
2
3    <!  Move the <scope> parts of an <invoke> into an explicit enclosing <
         scope>.   >
4    <!  Also, make temporary variables and assignments, due to the use of <
         toParts>,
5        <fromParts>, and/or references to element variables, explicit.   >
6
7   <xsl:stylesheet version="1.0"
8     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
9     xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
10
11    <xsl:output indent="yes" method="xml" />
12
13    <xsl:include href="to from parts element variables.xsl" />
14
15     <!  Copy all elements and attributes   >
16    <xsl:template match="*">
17      <xsl:copy>
18        <xsl:copy of select="@*" />
19        <xsl:apply templates />
20      </xsl:copy>
21    </xsl:template>
22
23     <!  Unfold invoke   >
24    <xsl:template match="bpel:invoke">
25      <xsl:variable name="inputVariable" select="@inputVariable" />
26      <xsl:variable name="outputVariable" select="@outputVariable" />
27
28      <xsl:variable name="inputElement" select="count(ancestor::*/bpel:
            variables/bpel:variable[@name=$inputVariable][1]/@element) = 1"
            />
29      <xsl:variable name="outputElement" select="count(ancestor::*/bpel:
            variables/bpel:variable[@name=$outputVariable][1]/@element) = 1"
            />
30
31      <xsl:choose>
32        <xsl:when test="bpel:toParts or $inputElement or bpel:fromParts or
              $outputElement or bpel:catch or bpel:catchAll or bpel:
              compensationHandler">
33          <bpel:scope>
34            <xsl:if test="bpel:toParts or $inputElement or bpel:fromParts
                  or $outputElement">
35              <bpel:variables>
36                <xsl:call template name="message activities temp variables"
                      >
37                  <xsl:with param name="messageActivities" select="." />
38                </xsl:call template>
39              </bpel:variables>
40            </xsl:if>
41
42            <xsl:if test="bpel:catch or bpel:catchAll">
43              <bpel:faultHandlers>
44                <xsl:copy of select="bpel:catch" />
45                <xsl:copy of select="bpel:catchAll" />
46              </bpel:faultHandlers>
47            </xsl:if>
48
49            <xsl:copy of select="bpel:compensationHandler" />
50
51            <xsl:choose>
52              <xsl:when test="bpel:toParts or bpel:fromParts or
                    $inputElement or $outputElement">
53                <bpel:sequence>
54                   <!  Transform toParts into an assignment, if present   >
55                  <xsl:if test="bpel:toParts">
56                    <xsl:call template name="copy to parts explicitly" />
57                  </xsl:if>
58
59                   <!  Create assignment to copy element variable to single
                        part   >
60                  <xsl:if test="$inputElement">
61                    <xsl:call template name="copy input element explicitly"
                          >
62                      <xsl:with param name="inputVariable" select="
                            $inputVariable" />
63                    </xsl:call template>
64                  </xsl:if>
65
66                  <xsl:copy>
67                    <xsl:copy of select="@*[not(namespace uri() = '' and
68                                              (local name() = '
                                                    inputVariable' or
69                                               local name() = '
                                                    outputVariable' or
70                                               local name() = 'portType')
                                                    )]" />
71
72                    <xsl:choose>
73                      <xsl:when test="bpel:toParts or $inputElement">
74                        <xsl:call template name="attribute with unique
                              element name">
75                          <xsl:with param name="attributeName" select="'
                                inputVariable'" />
76                          <xsl:with param name="element" select="." />
77                          <xsl:with param name="postfix" select="$tmp input
                                message variable postfix" />
78                        </xsl:call template>
79                      </xsl:when>
80                      <xsl:otherwise>
81                        <xsl:copy of select="@inputVariable" />
82                      </xsl:otherwise>
83                    </xsl:choose>
84
85                    <xsl:choose>
86                      <xsl:when test="bpel:fromParts or $outputElement">
87                        <xsl:call template name="attribute with unique
                              element name">
88                          <xsl:with param name="attributeName" select="'
                                outputVariable'" />
89                          <xsl:with param name="element" select="." />
90                          <xsl:with param name="postfix" select="$tmp
                                output message variable postfix" />
91                        </xsl:call template>
92                      </xsl:when>
93                      <xsl:otherwise>
94                        <xsl:copy of select="@outputVariable" />
95                      </xsl:otherwise>
96                    </xsl:choose>
97
98                    <xsl:apply templates select="*[not(self::bpel:catch or
99                                                  self::bpel:catchAll
                                                       or
100                                                 self::bpel:
                                                       compensationHandler
                                                       or
101                                                 self::bpel:toParts
                                                       or
102                                                 self::bpel:fromParts
                                                       )]"/>
103                 </xsl:copy>
104
105                  <!  Transform fromParts into an assignment, if present
                        >
106                 <xsl:if test="bpel:fromParts">
107                   <xsl:call template name="copy from parts explicitly" />
108                 </xsl:if>
```

```
109
110                     <!  Create assignment to copy single part to element
                            variable   >
111                    <xsl:if test="$outputElement">
112                      <xsl:call template name="copy output element explicitly
                              ">
113                        <xsl:with param name="outputVariable" select="
                                 $outputVariable" />
114                      </xsl:call template>
115                    </xsl:if>
116                  </bpel:sequence>
117                </xsl:when>
118
119                <xsl:otherwise>
120                  <xsl:copy>
121                    <xsl:copy of select="@*[not(namespace uri() = '' and
122                                              local name() = 'portType')]"
                                                  />
123                    <xsl:apply templates select="*[not(self::bpel:catch or
124                                                     self::bpel:catchAll or
125                                                     self::bpel:
                                                          compensationHandler
                                                          )]"/>
126                  </xsl:copy>
127                </xsl:otherwise>
128              </xsl:choose>
129
130            </bpel:scope>
131          </xsl:when>
132
133          <xsl:otherwise>
134            <xsl:copy>
135              <xsl:copy of select="@*[not(namespace uri() = '' and
136                                       local name() = 'portType')]" />
137              <xsl:apply templates />
138            </xsl:copy>
139          </xsl:otherwise>
140        </xsl:choose>
141      </xsl:template>
142
143    </xsl:stylesheet>
```

## 6.D.12 onEvent.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Make temporary variables and assignments, due to the use of
4       <fromParts>, and/or references to element variables in <onEvent>s,
            explicit.   >
5   <!  This stylesheet is included by scope.xsl and shouldn't be applied
        alone.   >
6
7  <xsl:stylesheet
8    version="1.0"
9    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
10   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
11
12   <xsl:include href="to from parts element variables.xsl" />
13
14   <xsl:template match="bpel:onEvent[bpel:fromParts or @element]">
15     <xsl:copy>
16       <xsl:copy of select="@*[not(namespace uri() = '' and
17                                (local name() = 'portType' or
18                                 local name() = 'element'))]" />
19
20       <xsl:attribute name="messageType">
21         <xsl:call template name="inbound message type" />
22       </xsl:attribute>
23       <xsl:call template name="attribute with unique element name">
24         <xsl:with param name="attributeName" select="'variable'" />
25         <xsl:with param name="element" select="." />
26         <xsl:with param name="postfix" select="$tmp output message
               variable postfix" />
27       </xsl:call template>
28
29       <xsl:apply templates select="bpel:correlations" />
30
31       <bpel:scope>
32         <xsl:copy of select="bpel:scope/@*" />
33
34         <xsl:apply templates select="bpel:scope/bpel:partnerLinks | bpel:
               scope/bpel:messageExchanges" />
35
36         <bpel:variables>
37           <xsl:for each select="bpel:fromParts/bpel:fromPart">
38             <bpel:variable name="{@toVariable}">
39               <xsl:call template name="inbound message part typing">
40                 <xsl:with param name="messageActivity" select="../.." />
41                 <xsl:with param name="part" select="@part" />
42               </xsl:call template>
43             </bpel:variable>
44           </xsl:for each>
45           <xsl:if test="@element">
46             <bpel:variable name="{@variable}" element="{@element}" />
47           </xsl:if>
48           <xsl:apply templates select="bpel:scope/bpel:variables/*" />
49         </bpel:variables>
50
51         <xsl:apply templates select="bpel:scope/bpel:correlationSets |
               bpel:scope/bpel:faultHandlers | bpel:scope/bpel:
               compensationHandler | bpel:scope/bpel:terminationHandler |
               bpel:scope/bpel:eventHandlers" />
52
53         <bpel:sequence>
54            <!  Transform fromParts into an assignment, if present   >
55           <xsl:if test="bpel:fromParts">
56             <xsl:call template name="copy from parts explicitly" />
57           </xsl:if>
58
59            <!  Create assignment to copy single part to element variable
                 >
60           <xsl:if test="@element">
61             <xsl:call template name="copy output element explicitly">
62               <xsl:with param name="outputVariable" select="@variable" />
63             </xsl:call template>
64           </xsl:if>
65
66           <xsl:apply templates select="bpel:scope/*[not(self::bpel:
                 partnerLinks or self::bpel:messageExchanges or self::bpel:
                 correlationSets or self::bpel:faultHandlers or self::bpel:
                 compensationHandler or self::bpel:terminationHandler or
                 self::bpel:eventHandlers)]"/>
67         </bpel:sequence>
68       </bpel:scope>
69
70       <xsl:apply templates select="*[not(self::bpel:*)]" />
71     </xsl:copy>
72   </xsl:template>
73
74 </xsl:stylesheet>
```

## 6.D.13 pick.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Make temporary variables and assignments, due to the use of
4        <fromParts>, and/or references to element variables in <onMessage>s,
            explicit.   >
5
6  <xsl:stylesheet
7    version="1.0"
8    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
9    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
10
11   <xsl:output indent="yes" method="xml" />
12
13   <xsl:include href="to from parts element variables.xsl" />
14
15    <!  Copy all elements and attributes   >
16   <xsl:template match="*">
17     <xsl:copy>
18       <xsl:copy of select="@*" />
19       <xsl:apply templates />
20     </xsl:copy>
21   </xsl:template>
22
23   <xsl:template name="onMessage">
24     <xsl:variable name="outputVariable" select="@variable" />
25     <xsl:variable name="outputElement" select="count(ancestor::*/bpel:
           variables/bpel:variable[@name=$outputVariable][1]/@element) = 1"
           />
26
27     <xsl:copy>
28       <xsl:copy of select="@*[not(namespace uri() = '' and local name() =
             'portType')]" />
29       <xsl:copy of select="bpel:correlations" />
30
31       <xsl:choose>
32         <xsl:when test="bpel:fromParts or $outputElement">
33           <xsl:call template name="attribute with unique element name">
34             <xsl:with param name="attributeName" select="'variable'" />
35             <xsl:with param name="element" select="." />
36             <xsl:with param name="postfix" select="$tmp output message
                 variable postfix" />
37           </xsl:call template>
38
39           <bpel:sequence>
40              <!  Transform fromParts into an assignment, if present   >
41             <xsl:if test="bpel:fromParts">
42               <xsl:call template name="copy from parts explicitly" />
43             </xsl:if>
44
45              <!  Create assignment to copy single part to element
                 variable   >
46             <xsl:if test="$outputElement">
47               <xsl:call template name="copy output element explicitly">
48                 <xsl:with param name="outputVariable" select="
                     $outputVariable" />
49               </xsl:call template>
50             </xsl:if>
51
52             <xsl:apply templates select="*[not(self::bpel:fromParts or
                 self::bpel:correlations)]"/>
53
54           </bpel:sequence>
55         </xsl:when>
56         <xsl:otherwise>
57           <xsl:apply templates select="*[not(self::bpel:correlations)]"/>
58         </xsl:otherwise>
59       </xsl:choose>
60     </xsl:copy>
61   </xsl:template>
62
63   <xsl:template match="bpel:pick">
64     <xsl:variable name="needs temp variables">
65       <xsl:call template name="message activities need temp variables">
66         <xsl:with param name="messageActivities" select="bpel:onMessage"
             />
67       </xsl:call template>
68     </xsl:variable>
69
70     <xsl:choose>
71       <xsl:when test="$needs temp variables = 'true'">
72         <bpel:scope>
73           <bpel:variables>
74             <xsl:call template name="message activities temp variables">
75               <xsl:with param name="messageActivities" select="bpel:
                   onMessage" />
76             </xsl:call template>
77           </bpel:variables>
78
79           <xsl:copy>
80             <xsl:copy of select="@*" />
81
82             <xsl:for each select="bpel:onMessage">
83               <xsl:call template name="onMessage" />
84             </xsl:for each>
85
86             <xsl:apply templates select="*[not(self::bpel:onMessage)]" />
87           </xsl:copy>
88         </bpel:scope>
89        </xsl:when>
90
91       <xsl:otherwise>
92         <xsl:copy>
93           <xsl:copy of select="@*" />
94           <xsl:apply templates />
95         </xsl:copy>
96       </xsl:otherwise>
97     </xsl:choose>
98
99   </xsl:template>
100
101 </xsl:stylesheet>
```

## 6.D.14 process.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Move the scope parts of a <process> into an explicit <scope>.   >
4
5  <xsl:stylesheet version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9    <xsl:output indent="yes" method="xml" />
10
11   <xsl:template match="*">
12     <xsl:copy>
13       <xsl:copy of select="@*" />
14       <xsl:apply templates />
15     </xsl:copy>
16   </xsl:template>
17
18   <xsl:template match="bpel:process[@exitOnStandardFault or
```

```
19                                     @suppressJoinFailure or
20                                     bpel:partnerLinks or
21                                     bpel:messageExchanges or
22                                     bpel:variables or
23                                     bpel:correlationSets or
24                                     bpel:faultHandlers or
25                                     bpel:eventHandlers]">
26     <xsl:copy>
27       <xsl:copy of select="@*[not(namespace uri() = '' and
28                                   (local name() = 'exitOnStandardFault'
                                        or
29                                    local name() = 'suppressJoinFailure'))
                                        ]" />
30       <xsl:copy of select="bpel:extensions" />
31       <xsl:copy of select="bpel:import" />
32       <bpel:scope>
33         <xsl:copy of select="@*[not(namespace uri() = '' and
34                                     (local name() = 'targetNamespace' or
35                                      local name() = 'expressionLanguage'
                                          or
36                                      local name() = 'queryLanguage'))]"
                                          />
37         <xsl:apply templates select="*[not(self::bpel:import or self::
               bpel:extensions)]" />
38       </bpel:scope>
39     </xsl:copy>
40   </xsl:template>
41
42 </xsl:stylesheet>
```

## 6.D.15 receive.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Transform <receive> into <pick>.  >
4
5 <xsl:stylesheet version="1.0"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9   <xsl:output indent="yes" method="xml" />
10
11   <!  Copy all elements and attributes  >
12   <xsl:template match="*">
13     <xsl:copy>
14       <xsl:copy of select="@*" />
15       <xsl:apply templates />
16     </xsl:copy>
17   </xsl:template>
18
19   <!  Unfold receive  >
20   <xsl:template match="bpel:receive">
21     <bpel:pick>
22       <xsl:copy of select="@createInstance" />
23       <xsl:copy of select="@name" />
24       <xsl:copy of select="@suppressJoinFailure" />
25       <xsl:apply templates select="bpel:targets" />
26       <xsl:apply templates select="bpel:sources" />
27       <bpel:onMessage>
28         <xsl:copy of select="@*[not(namespace uri() = '' and
29                                     (local name() = 'createInstance' or
30                                      local name() = 'name' or
31                                      local name() = 'suppressJoinFailure
                                          '))]" />
32         <xsl:apply templates select="*[not(self::bpel:targets or self::
               bpel:sources)]" />
33         <bpel:empty/>
34       </bpel:onMessage>
35     </bpel:pick>
36   </xsl:template>
37
38 </xsl:stylesheet>
```

## 6.D.16 remove-documentation.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Remove any documentation elements.  >
4
5 <xsl:stylesheet version="1.0"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9   <xsl:output indent="yes" method="xml" />
10
11   <xsl:template match="*">
12     <xsl:copy>
13       <xsl:copy of select="@*" />
14       <xsl:apply templates />
15     </xsl:copy>
16   </xsl:template>
17
18   <xsl:template match="bpel:documentation" />
19
20 </xsl:stylesheet>
```

## 6.D.17 remove-optional-extensions.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Remove all optional extensions and their declarations.  >
4
5 <xsl:stylesheet version="1.0"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9   <xsl:include href="constants.xsl"/>
10
11   <xsl:output indent="yes" method="xml" />
12
13    <!  Initiate the removal of optional extensions by collecting the set
         of
14       namespace URIs for mandatory extensions.  >
15   <xsl:template match="bpel:process">
16     <xsl:call template name="clean bpel node">
17       <xsl:with param name="mandatory URIs" select="bpel:extensions/bpel:
             extension[@mustUnderstand='yes']/@namespace" />
18     </xsl:call template>
19   </xsl:template>
20
21    <!  Remove the <extensions> element if there are no mandatory
         extensions.  >
22   <xsl:template match="bpel:extensions[not(bpel:extension[@mustUnderstand
         ='yes'])]" />
```

```
23
24    <! Remove declarations of optional extensions. >
25   <xsl:template match="bpel:extension[@mustUnderstand='no']" />
26
27    <! Replace optional <extensionActivity>s with <empty>. >
28   <xsl:template match="bpel:extensionActivity">
29     <xsl:param name="mandatory URIs" />
30
31     <xsl:choose>
32       <xsl:when test="namespace uri(*[1]) = $mandatory URIs">
33         <xsl:copy>
34           <xsl:apply templates select="@*">
35             <xsl:with param name="mandatory URIs" select="$mandatory URIs
                    " />
36           </xsl:apply templates>
37           <xsl:apply templates>
38             <xsl:with param name="mandatory URIs" select="$mandatory URIs
                    " />
39           </xsl:apply templates>
40         </xsl:copy>
41       </xsl:when>
42       <xsl:otherwise>
43         <bpel:empty>
44           <xsl:apply templates select="*[1]/@*">
45             <xsl:with param name="mandatory URIs" select="$mandatory URIs
                    " />
46           </xsl:apply templates>
47           <xsl:apply templates select="*[1]/child::node()">
48             <xsl:with param name="mandatory URIs" select="$mandatory URIs
                    " />
49           </xsl:apply templates>
50         </bpel:empty>
51       </xsl:otherwise>
52     </xsl:choose>
53   </xsl:template>
54
55    <! Replace <assign>s that will be empty after removing optional
          <extensionAssignOperation>s with <empty>. >
57   <xsl:template match="bpel:assign">
58     <xsl:param name="mandatory URIs" />
59
60     <xsl:choose>
61       <xsl:when test="bpel:copy or bpel:extensionAssignOperation/*[1][
              namespace uri() = $mandatory URIs]">
62         <xsl:call template name="clean bpel node">
63           <xsl:with param name="mandatory URIs" select="$mandatory URIs"
                  />
64         </xsl:call template>
65       </xsl:when>
66       <xsl:otherwise>
67         <bpel:empty>
68           <xsl:apply templates select="@*[not(namespace uri() = '' and
                  local name() = 'validate')]">
69             <xsl:with param name="mandatory URIs" select="$mandatory URIs
                    " />
70           </xsl:apply templates>
71           <xsl:apply templates>
72             <xsl:with param name="mandatory URIs" select="$mandatory URIs
                    " />
73           </xsl:apply templates>
74         </bpel:empty>
75       </xsl:otherwise>
76     </xsl:choose>
77   </xsl:template>
78
79    <! Remove optional <extensionAssignOperation>s. >
80   <xsl:template match="bpel:extensionAssignOperation">
81     <xsl:param name="mandatory URIs" />
82
83     <xsl:if test="namespace uri(*[1]) = $mandatory URIs">
84       <xsl:copy>
85         <xsl:apply templates select="@*">
86           <xsl:with param name="mandatory URIs" select="$mandatory URIs"
                  />
87         </xsl:apply templates>
88         <xsl:apply templates>
89           <xsl:with param name="mandatory URIs" select="$mandatory URIs"
                  />
90         </xsl:apply templates>
91       </xsl:copy>
92     </xsl:if>
93   </xsl:template>
94
95    <! Protect <literal>s. >
96   <xsl:template match="bpel:literal">
97     <xsl:param name="mandatory URIs" />
98
99     <xsl:copy of select="." />
100  </xsl:template>
101
102   <! BPEL constructs recursively. >
103  <xsl:template match="bpel:*" name="clean bpel node">
104    <xsl:param name="mandatory URIs" />
105
106    <xsl:copy>
107      <xsl:apply templates select="@*">
108        <xsl:with param name="mandatory URIs" select="$mandatory URIs" />
109      </xsl:apply templates>
110      <xsl:apply templates>
111        <xsl:with param name="mandatory URIs" select="$mandatory URIs" />
112      </xsl:apply templates>
113    </xsl:copy>
114  </xsl:template>
115
116   <! Remove extensions that are in an optional namespace, and leave the
          rest untouched. >
117  <xsl:template match="*">
118    <xsl:param name="mandatory URIs" />
119
120    <xsl:if test="namespace uri() = $mandatory URIs">
121      <xsl:copy of select="." />
122    </xsl:if>
123  </xsl:template>
124
125   <! Remove extension attributes that are in an optional namespace. >
126  <xsl:template match="@*">
127    <xsl:param name="mandatory URIs" />
128
129    <xsl:if test="namespace uri() = '' or namespace uri() = $mandatory
            URIs">
130      <xsl:copy />
131    </xsl:if>
132  </xsl:template>
133
134 </xsl:stylesheet>
```

## 6.D.18 remove-redundant-attributes.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <! Remove the redundant attributes. >
4
5 <xsl:stylesheet version="1.0"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
```

```
8
9   <xsl:output indent="yes" method="xml" />
10
11  <xsl:template match="*">
12    <xsl:copy>
13      <xsl:copy of select="@*" />
14      <xsl:apply templates />
15    </xsl:copy>
16  </xsl:template>
17
18  <xsl:template match="*" mode="extensionActivity">
19      <xsl:copy of select="@*[not(namespace uri() = '' and local name() =
            'suppressJoinFailure')]" />
20      <xsl:copy of select="*" />
21  </xsl:template>
22
23  <xsl:template match="bpel:extensionActivity">
24    <xsl:copy>
25      <xsl:copy of select="@*" />
26      <xsl:apply templates mode="extensionActivity" />
27    </xsl:copy>
28  </xsl:template>
29
30  <xsl:template match="bpel:assign | bpel:compensate | bpel:
        compensateScope | bpel:empty | bpel:exit | bpel:forEach | bpel:if
        | bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil | bpel
        :reply | bpel:rethrow | bpel:scope | bpel:sequence | bpel:throw |
        bpel:validate | bpel:wait | bpel:while">
31    <xsl:copy>
32      <xsl:copy of select="@*[not(namespace uri() = '' and
33                                  local name() = 'suppressJoinFailure')]"
                                        />
34      <xsl:apply templates />
35    </xsl:copy>
36  </xsl:template>
37
38  <xsl:template match="bpel:process">
39    <bpel:process>
40      <xsl:copy of select="@*[not(namespace uri() = '' and
41                                  (local name() = 'queryLanguage' or
42                                   local name() = 'expressionLanguage' or
43                                   local name() = 'suppressJoinFailure'
                                        or
44                                   local name() = 'exitOnStandardFault'))
                                        ]" />
45      <xsl:apply templates />
46    </bpel:process>
47  </xsl:template>
48
49 </xsl:stylesheet>
```

## 6.D.19 repeatUntil.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Transform <repeatUntil> into <while>.  >
4
5 <xsl:stylesheet version="1.0"
6   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable"
8   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
9
10  <xsl:output indent="yes" method="xml" />
11
12  <xsl:include href="constants.xsl"/>
13  <xsl:include href="fresh names.xsl"/>
14
15  <xsl:template match="*">
16    <xsl:copy>
17      <xsl:copy of select="@*" />
18      <xsl:apply templates />
19    </xsl:copy>
20  </xsl:template>
21
22  <xsl:template match="bpel:repeatUntil">
23    <xsl:variable name="conditionVariable">
24      <xsl:call template name="unique element name">
25        <xsl:with param name="postfix" select="$tmp condition variable
              postfix" />
26      </xsl:call template>
27    </xsl:variable>
28
29    <bpel:scope>
30      <xsl:copy of select="@*" />
31      <xsl:apply templates select="bpel:targets | bpel:sources" />
32
33      <bpel:variables>
34        <bpel:variable type="xsd:boolean">
35          <xsl:attribute name="name">
36            <xsl:value of select="$conditionVariable" />
37          </xsl:attribute>
38          <bpel:from expressionLanguage="{$xpathURN}">
39            true()
40          </bpel:from>
41        </bpel:variable>
42      </bpel:variables>
43
44      <bpel:while>
45        <bpel:condition expressionLanguage="{$xpathURN}">
46          $<xsl:value of select="$conditionVariable" />
47        </bpel:condition>
48        <bpel:sequence>
49          <xsl:apply templates select="*[not(self::bpel:targets or self::
                bpel:sources or self::bpel:condition)]" />
50          <bpel:if>
51            <xsl:apply templates select="bpel:condition" />
52            <bpel:assign>
53              <bpel:copy>
54                <bpel:from expressionLanguage="{$xpathURN}">
55                  false()
56                </bpel:from>
57                <bpel:to variable="{$conditionVariable}" />
58              </bpel:copy>
59            </bpel:assign>
60          </bpel:if>
61        </bpel:sequence>
62      </bpel:while>
63    </bpel:scope>
64  </xsl:template>
65
66 </xsl:stylesheet>
```

## 6.D.20 reply.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Make temporary variables and assignments, due to the use of <toParts
      >,
4      and/or references to an element variable, explicit.  >
5
```

```
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:include href="to from parts element variables.xsl" />
13
14   <!  Copy all elements and attributes   >
15   <xsl:template match="*">
16     <xsl:copy>
17       <xsl:copy of select="@*" />
18       <xsl:apply templates />
19     </xsl:copy>
20   </xsl:template>
21
22   <!  Unfold reply   >
23   <xsl:template match="bpel:reply">
24     <xsl:variable name="inputVariable" select="@variable" />
25     <xsl:variable name="inputElement" select="count(ancestor::*/bpel:
           variables/bpel:variable[@name=$inputVariable][1]/@element) = 1"
           />
26
27     <xsl:choose>
28       <xsl:when test="bpel:toParts or $inputElement">
29         <bpel:scope>
30           <bpel:variables>
31             <xsl:call template name="message activities temp variables">
32               <xsl:with param name="messageActivities" select="." />
33             </xsl:call template>
34           </bpel:variables>
35
36           <bpel:sequence>
37             <!  Transform toParts into an assignment, if present   >
38             <xsl:if test="bpel:toParts">
39               <xsl:call template name="copy to parts explicitly" />
40             </xsl:if>
41
42             <!  Create assignment to copy element variable to single
                   part   >
43             <xsl:if test="$inputElement">
44               <xsl:call template name="copy input element explicitly">
45                 <xsl:with param name="inputVariable" select="
                       $inputVariable" />
46               </xsl:call template>
47             </xsl:if>
48
49             <xsl:copy>
50               <xsl:copy of select="@*[not(namespace uri() = '' and
51                                         (local name() = 'variable' or
52                                          local name() = 'portType'))]"
                                               />
53               <xsl:call template name="attribute with unique element name
                     ">
54                 <xsl:with param name="attributeName" select="'variable'"
                       />
55                 <xsl:with param name="element" select="." />
56                 <xsl:with param name="postfix" select="$tmp input message
                       variable postfix" />
57               </xsl:call template>
58               <xsl:apply templates select="*[not(self::bpel:toParts)]" />
59             </xsl:copy>
60           </bpel:sequence>
61         </bpel:scope>
62       </xsl:when>
63
64       <xsl:otherwise>
65         <xsl:copy>
66           <xsl:copy of select="@*[not(namespace uri() = '' and local name
                 () = 'portType')]" />
67           <xsl:apply templates select="*" />
68         </xsl:copy>
69       </xsl:otherwise>
70     </xsl:choose>
71
72   </xsl:template>
73
74 </xsl:stylesheet>
```

## 6.D.21 scope.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!  Make variable initialization explicit in all scopes (including <
      process>).   >
4   <!  Make implicit variables and assignments in <onEvent>s explicit.   >
5
6  <xsl:stylesheet version="1.0"
7    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
8    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
9
10   <xsl:output indent="yes" method="xml" />
11
12   <xsl:include href="onEvent.xsl" />
13
14   <xsl:template match="*">
15     <xsl:copy>
16       <xsl:copy of select="@*" />
17       <xsl:apply templates />
18     </xsl:copy>
19   </xsl:template>
20
21   <xsl:template match="bpel:process[bpel:variables/bpel:variable/bpel:
         from]">
22     <xsl:copy>
23       <xsl:copy of select="@*" />
24       <xsl:apply templates select="bpel:extensions | bpel:import" />
25       <xsl:call template name="scope" />
26     </xsl:copy>
27   </xsl:template>
28
29   <xsl:template match="bpel:scope[bpel:variables/bpel:variable/bpel:from]
       "
30                 name="scope">
31
32     <bpel:scope>
33       <bpel:variables>
34         <xsl:for each select="bpel:variables/bpel:variable">
35           <xsl:copy>
36             <xsl:copy of select="@*"/>
37           </xsl:copy>
38         </xsl:for each>
39       </bpel:variables>
40
41       <bpel:faultHandlers>
42         <bpel:catchAll>
43           <bpel:rethrow/>
44         </bpel:catchAll>
45       </bpel:faultHandlers>
46
47       <bpel:sequence>
48         <bpel:scope>
49           <bpel:faultHandlers>
50             <bpel:catchAll>
51               <bpel:throw faultName='bpel:scopeInitializationFault'/>
```

```
52              </bpel:catchAll>
53            </bpel:faultHandlers>
54
55            <bpel:assign>
56              <xsl:for each select="bpel:variables/bpel:variable[bpel:from]
                  ">
57                <bpel:copy>
58                  <xsl:apply templates select="bpel:from"/>
59                  <bpel:to>
60                    <xsl:attribute name="variable"><xsl:value of select="
                          @name"/></xsl:attribute>
61                  </bpel:to>
62                </bpel:copy>
63              </xsl:for each>
64            </bpel:assign>
65          </bpel:scope>
66
67          <bpel:scope>
68            <xsl:copy of select="@*[not(namespace uri() = '' and
69                                     local name() = 'targetNamespace')]"
                                       />
70            <xsl:apply templates select="*[not(self::bpel:variables or self
                  ::bpel:extensions or self::bpel:import)]" />
71          </bpel:scope>
72
73        </bpel:sequence>
74      </bpel:scope>
75
76    </xsl:template>
77
78  </xsl:stylesheet>
```

## 6.D.22 sequence.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3   <!   Transform <sequence>s into <flow>s.   >
4  <xsl:stylesheet
5    version="1.0"
6    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
7    xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
8
9    <xsl:output indent="yes" method="xml" />
10
11   <xsl:include href="constants.xsl"/>
12   <xsl:include href="fresh names.xsl"/>
13
14   <xsl:template match="*">
15     <xsl:copy>
16       <xsl:copy of select="@*" />
17       <xsl:apply templates />
18     </xsl:copy>
19   </xsl:template>
20
21   <xsl:template match="bpel:sequence">
22     <bpel:flow>
23       <xsl:copy of select="@*" />
24
25       <xsl:apply templates select="bpel:targets | bpel:sources" />
26
27       <xsl:choose>
28         <xsl:when test="1 &lt; count(bpel:assign | bpel:compensate | bpel
                 :compensateScope | bpel:empty | bpel:exit | bpel:
                 extensionActivity | bpel:flow | bpel:forEach | bpel:if |
                 bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil |
                 bpel:reply | bpel:rethrow | bpel:scope | bpel:sequence |
                 bpel:throw | bpel:validate | bpel:wait | bpel:while)">
29           <xsl:call template name="sequence links" />
30           <xsl:apply templates select="bpel:assign | bpel:compensate |
                 bpel:compensateScope | bpel:empty | bpel:exit | bpel:
                 extensionActivity | bpel:flow | bpel:forEach | bpel:if |
                 bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil
                 | bpel:reply | bpel:rethrow | bpel:scope | bpel:sequence |
                  bpel:throw | bpel:validate | bpel:wait | bpel:while"
                              mode="sequenceChild" />
31         </xsl:when>
32
33         <xsl:otherwise>
34           <xsl:apply templates select="bpel:assign | bpel:compensate |
35               bpel:compensateScope | bpel:empty | bpel:exit | bpel:
                 extensionActivity | bpel:flow | bpel:forEach | bpel:if |
                 bpel:invoke | bpel:pick | bpel:receive | bpel:repeatUntil
                 | bpel:reply | bpel:rethrow | bpel:scope | bpel:sequence |
                  bpel:throw | bpel:validate | bpel:wait | bpel:while" />
36         </xsl:otherwise>
37       </xsl:choose>
38
39       <xsl:apply templates select="*[not(self::bpel:targets or self::bpel
             :sources or self::bpel:assign or self::bpel:compensate or self
             ::bpel:compensateScope or self::bpel:empty or self::bpel:exit
             or self::bpel:extensionActivity or self::bpel:flow or self::
             bpel:forEach or self::bpel:if or self::bpel:invoke or self::
             bpel:pick or self::bpel:receive or self::bpel:repeatUntil or
             self::bpel:reply or self::bpel:rethrow or self::bpel:scope or
             self::bpel:sequence or self::bpel:throw or self::bpel:validate
              or self::bpel:wait or self::bpel:while)]" />
40
41     </bpel:flow>
42   </xsl:template>
43
44   <xsl:template name="sequence links">
45     <bpel:links>
46       <xsl:for each select="bpel:assign | bpel:compensate | bpel:
             compensateScope | bpel:empty | bpel:exit | bpel:
             extensionActivity | bpel:flow | bpel:forEach | bpel:if | bpel:
             invoke | bpel:pick | bpel:receive | bpel:repeatUntil | bpel:
             reply | bpel:rethrow | bpel:scope | bpel:sequence | bpel:throw
              | bpel:validate | bpel:wait | bpel:while">
47         <xsl:if test="following sibling::bpel:*">
48           <bpel:link>
49             <xsl:call template name="attribute with unique element name">
50               <xsl:with param name="attributeName" select="'name'" />
51               <xsl:with param name="element" select="." />
52               <xsl:with param name="postfix" select="$fresh sequence link
                     postfix" />
53             </xsl:call template>
54           </bpel:link>
55         </xsl:if>
56       </xsl:for each>
57     </bpel:links>
58   </xsl:template>
59
60    <!   Wrap the children of <sequence>s in <flow>s with appropriate links
        .   >
61   <xsl:template match="*" mode="sequenceChild">
62     <bpel:flow>
63       <xsl:if test="position() &gt; 1">
64         <bpel:targets>
65           <bpel:target>
66             <xsl:call template name="attribute with unique element name">
67               <xsl:with param name="attributeName" select="'linkName'" />
68               <xsl:with param name="element" select="preceding sibling
                     ::*[1]" />
69               <xsl:with param name="postfix" select="$fresh sequence link
                     postfix" />
```

```
70              </xsl:call template>
71            </bpel:target>
72          </bpel:targets>
73        </xsl:if>
74
75        <xsl:if test="following sibling::bpel:*">
76          <bpel:sources>
77            <bpel:source>
78              <xsl:call template name="attribute with unique element name">
79                <xsl:with param name="attributeName" select="'linkName'" />
80                <xsl:with param name="element" select="." />
81                <xsl:with param name="postfix" select="$fresh sequence link
                        postfix" />
82              </xsl:call template>
83            </bpel:source>
84          </bpel:sources>
85        </xsl:if>
86
87        <xsl:apply templates select="." />
88      </bpel:flow>
89    </xsl:template>
90
91  </xsl:stylesheet>
```

## 6.D.23 standard-attributes-elements.xsl

```
1  <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!   Move <targets>, <sources>, and suppressJoinFailure from activities
        to a
4       new wrapping <flow>, except for activities that have no <targets> or
5       <sources>, where we push the value of that attribute to all the
            child
6       activities.   >
7  <!   Names are also moved from activities (except <scope>s) to a new
        wrapping
8       <flow> and fresh names are added to unnamed <flow>s and <scope>s.
            >
9
10 <xsl:stylesheet version="1.0"
11   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
12   xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable">
13
14   <xsl:output indent="yes" method="xml" />
15
16   <xsl:include href="constants.xsl"/>
17   <xsl:include href="fresh names.xsl" />
18
19   <xsl:template match="*">
20     <xsl:param name="suppressJoinFailure" select="false()" />
21
22     <xsl:copy>
23       <xsl:copy of select="@*" />
24       <xsl:apply templates>
25         <xsl:with param name="suppressJoinFailure" select="
              $suppressJoinFailure" />
26       </xsl:apply templates>
27     </xsl:copy>
28   </xsl:template>
29
30   <xsl:template name="name">
31     <xsl:param name="name" />
32
33     <xsl:choose>
34       <xsl:when test="normalize space(string($name)) = ''">
35         <xsl:call template name="attribute with unique element name">
36           <xsl:with param name="attributeName" select="'name'" />
37           <xsl:with param name="element" select="." />
38           <xsl:with param name="postfix" select="$fresh activity name
                postfix" />
39         </xsl:call template>
40       </xsl:when>
41       <xsl:otherwise>
42         <xsl:copy of select="$name" />
43       </xsl:otherwise>
44     </xsl:choose>
45   </xsl:template>
46
47   <xsl:template match="*" mode="extensionActivity">
48     <xsl:copy of select="@*[not(namespace uri() = '' and local name() =
            'suppressJoinFailure')]" />
49     <xsl:copy of select="*[not(self::bpel:targets or self::bpel:sources
           )]" />
50   </xsl:template>
51
52   <xsl:template match="bpel:extensionActivity[*[1]/bpel:targets or *[1]/
        bpel:sources or *[1]/@suppressJoinFailure]">
53     <xsl:param name="suppressJoinFailure" select="false()" />
54
55     <xsl:choose>
56       <xsl:when test="not(*[1]/bpel:targets or *[1]/bpel:sources)">
57         <xsl:copy>
58           <xsl:copy of select="*" />
59           <xsl:apply templates />
60         </xsl:copy>
61       </xsl:when>
62       <xsl:otherwise>
63         <bpel:flow>
64           <xsl:copy of select="*[1]/@suppressJoinFailure" />
65           <xsl:if test="not(*[1]/@suppressJoinFailure) and
                $suppressJoinFailure">
66             <xsl:copy of select="$suppressJoinFailure" />
67           </xsl:if>
68           <xsl:copy of select="*[1]/bpel:targets" />
69           <xsl:copy of select="*[1]/bpel:sources" />
70           <xsl:copy>
71             <xsl:apply templates mode="extensionActivity" />
72           </xsl:copy>
73         </bpel:flow>
74       </xsl:otherwise>
75     </xsl:choose>
76   </xsl:template>
77
78   <xsl:template match="bpel:flow">
79     <xsl:param name="suppressJoinFailure" select="false()" />
80
81     <xsl:copy>
82       <xsl:copy of select="@*[not(namespace uri() = '' and local name() =
            'name')]" />
83       <xsl:if test="not(@suppressJoinFailure) and $suppressJoinFailure">
84         <xsl:copy of select="$suppressJoinFailure" />
85       </xsl:if>
86       <xsl:apply templates />
87     </xsl:copy>
88   </xsl:template>
89
90   <xsl:template match="bpel:scope">
91     <xsl:param name="suppressJoinFailure" select="false()" />
92
93     <xsl:choose>
94       <xsl:when test="$suppressJoinFailure">
95         <xsl:call template name="scope with links">
96           <xsl:with param name="suppressJoinFailure" select="
                $suppressJoinFailure" />
97         </xsl:call template>
```

```
98        </xsl:when>
99        <xsl:otherwise>
100         <xsl:copy>
101           <xsl:copy of select="@*[not(namespace uri() = '' and local name
                  () = 'name')]" />
102           <xsl:call template name="name">
103             <xsl:with param name="name" select="@name"/>
104           </xsl:call template>
105           <xsl:apply templates />
106         </xsl:copy>
107       </xsl:otherwise>
108     </xsl:choose>
109   </xsl:template>
110
111   <xsl:template match="bpel:scope[bpel:targets or bpel:sources or
          @suppressJoinFailure]"
112                 name="scope with links">
113     <xsl:param name="suppressJoinFailure" select="false()" />
114
115     <xsl:choose>
116       <xsl:when test="not(bpel:targets or bpel:sources)">
117         <xsl:copy>
118           <xsl:copy of select="@*[not(namespace uri() = '' and
119                                       (local name() = '
                                              suppressJoinFailure' or
120                                        local name() = 'name'))]" />
121           <xsl:call template name="name">
122             <xsl:with param name="name" select="@name"/>
123           </xsl:call template>
124
125           <xsl:choose>
126             <xsl:when test="@suppressJoinFailure">
127               <xsl:apply templates>
128                 <xsl:with param name="suppressJoinFailure" select="
                        @suppressJoinFailure" />
129               </xsl:apply templates>
130             </xsl:when>
131             <xsl:otherwise>
132               <xsl:apply templates>
133                 <xsl:with param name="suppressJoinFailure" select="
                        $suppressJoinFailure" />
134               </xsl:apply templates>
135             </xsl:otherwise>
136           </xsl:choose>
137         </xsl:copy>
138       </xsl:when>
139       <xsl:otherwise>
140         <bpel:flow>
141           <xsl:copy of select="@suppressJoinFailure" />
142           <xsl:if test="not(@suppressJoinFailure) and
                  $suppressJoinFailure">
143             <xsl:copy of select="$suppressJoinFailure" />
144           </xsl:if>
145           <xsl:copy of select="bpel:targets" />
146           <xsl:copy of select="bpel:sources" />
147           <xsl:copy>
148             <xsl:copy of select="@*[not(namespace uri() = '' and
149                                         (local name() = '
                                                suppressJoinFailure' or
150                                          local name() = 'name'))]" />
151             <xsl:call template name="name">
152               <xsl:with param name="name" select="@name"/>
153             </xsl:call template>
154             <xsl:apply templates select="*[not(self::bpel:targets or self
                    ::bpel:sources)]" />
155           </xsl:copy>
156         </bpel:flow>
157       </xsl:otherwise>
158     </xsl:choose>
159   </xsl:template>
160
161   <xsl:template match="bpel:assign
162                       | bpel:compensate
163                       | bpel:compensateScope
164                       | bpel:empty
165                       | bpel:exit
166                       | bpel:forEach
167                       | bpel:if
168                       | bpel:invoke
169                       | bpel:pick
170                       | bpel:receive
171                       | bpel:repeatUntil
172                       | bpel:reply
173                       | bpel:rethrow
174                       | bpel:sequence
175                       | bpel:throw
176                       | bpel:validate
177                       | bpel:wait
178                       | bpel:while">
179     <xsl:param name="suppressJoinFailure" select="false()" />
180
181     <xsl:choose>
182       <xsl:when test="$suppressJoinFailure">
183         <xsl:call template name="activity with links">
184           <xsl:with param name="suppressJoinFailure" select="
                  $suppressJoinFailure" />
185         </xsl:call template>
186       </xsl:when>
187       <xsl:otherwise>
188         <xsl:copy>
189           <xsl:copy of select="@*[not(namespace uri() = '' and
190                                       local name() = 'name')]" />
191           <xsl:apply templates />
192         </xsl:copy>
193       </xsl:otherwise>
194     </xsl:choose>
195   </xsl:template>
196
197   <xsl:template match="bpel:assign[bpel:targets or bpel:sources or
          @suppressJoinFailure]
198                       | bpel:compensate[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
199                       | bpel:compensateScope[bpel:targets or bpel:sources
                              or @suppressJoinFailure]
200                       | bpel:empty[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
201                       | bpel:exit[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
202                       | bpel:forEach[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
203                       | bpel:if[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
204                       | bpel:invoke[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
205                       | bpel:pick[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
206                       | bpel:receive[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
207                       | bpel:repeatUntil[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
208                       | bpel:reply[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
209                       | bpel:rethrow[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
210                       | bpel:sequence[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
211                       | bpel:throw[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
212                       | bpel:validate[bpel:targets or bpel:sources or
                              @suppressJoinFailure]
```

```
213                     | bpel:wait[bpel:targets or bpel:sources or
                            @suppressJoinFailure]
214                     | bpel:while[bpel:targets or bpel:sources or
                            @suppressJoinFailure]"
215                 name="activity with links">
216     <xsl:param name="suppressJoinFailure" select="false()" />
217
218     <xsl:choose>
219       <xsl:when test="not(bpel:targets or bpel:sources)">
220         <xsl:copy>
221           <xsl:copy of select="@*[not(namespace uri() = '' and
222                                  (local name() = 'name' or
223                                   local name() = '
                                         suppressJoinFailure'))]" />
224
225           <xsl:choose>
226             <xsl:when test="@suppressJoinFailure">
227               <xsl:apply templates>
228                 <xsl:with param name="suppressJoinFailure" select="
                        @suppressJoinFailure" />
229               </xsl:apply templates>
230             </xsl:when>
231             <xsl:otherwise>
232               <xsl:apply templates>
233                 <xsl:with param name="suppressJoinFailure" select="
                        $suppressJoinFailure" />
234               </xsl:apply templates>
235             </xsl:otherwise>
236           </xsl:choose>
237         </xsl:copy>
238       </xsl:when>
239       <xsl:otherwise>
240         <bpel:flow>
241           <xsl:copy of select="@suppressJoinFailure" />
242           <xsl:if test="not(@suppressJoinFailure) and
                  $suppressJoinFailure">
243             <xsl:copy of select="$suppressJoinFailure" />
244           </xsl:if>
245           <xsl:copy of select="bpel:targets" />
246           <xsl:copy of select="bpel:sources" />
247           <xsl:copy>
248             <xsl:copy of select="@*[not(namespace uri() = '' and
249                                    (local name() = 'name' or
250                                     local name() = '
                                           suppressJoinFailure'))]" />
251             <xsl:apply templates select="*[not(self::bpel:targets or self
                    ::bpel:sources)]" />
252           </xsl:copy>
253         </bpel:flow>
254       </xsl:otherwise>
255     </xsl:choose>
256   </xsl:template>
257
258 </xsl:stylesheet>
```

## 6.D.24 to-from-parts-element-variables.xsl

```
1 <?xml version="1.0" encoding="ISO 8859 1"?>
2
3  <!  Utility templates to make temporary variables and assignments, due
       to the
4      use of <toParts>, <fromParts>, and/or references to element
           variables,
5      explicit.  >
6  <!  NB: this template should _not_ be applied by itself.  >
7
8 <xsl:stylesheet version="1.0"
9   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
10  xmlns:bpel="http://docs.oasis open.org/wsbpel/2.0/process/executable"
11  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
12  xmlns:plnk="http://docs.oasis open.org/wsbpel/2.0/plnktype">
13
14  <xsl:include href="constants.xsl" />
15  <xsl:include href="fresh names.xsl" />
16
17   <!  Get the WSDL message name for either the input or output message
         of the
18       operation in question.
19       Assumes that the context node is a message activity.  >
20  <xsl:template name="wsdl message name">
21     <!  If $input = true then the input message type is returned,
22         otherwise the output message type is returned.  >
23    <xsl:param name="input" />
24     <!  If $myRole = true then the operation is provided by this process
           ,
25         otherwise it is provided by the partner.  >
26    <xsl:param name="myRole" />
27
28    <xsl:variable name="partnerLinkName" select="@partnerLink" />
29    <xsl:variable name="operation" select="@operation" />
30    <xsl:variable name="partnerLink" select="(self::*[self::bpel:invoke
          or self::bpel:onMessage or self::bpel:receive or self::bpel:
          reply] |
31                                            self::bpel:onEvent/bpel:
                                                  scope)
32                                           /ancestor or self::*/bpel:
                                                  partnerLinks/bpel:
                                                  partnerLink[@name=
                                                  $partnerLinkName][1]"
                                                  />
33    <xsl:variable name="partnerLinkNamespace" select="namespace::*[local
          name() = substring before($partnerLink/@partnerLinkType, ':')]"
          />
34    <xsl:variable name="definitions" select="document(/bpel:process/bpel:
          import[@importType='http://schemas.xmlsoap.org/wsdl/']/@location
          )/wsdl:definitions[@targetNamespace=$partnerLinkNamespace]" />
35    <xsl:variable name="portType">
36      <xsl:choose>
37        <xsl:when test="$myRole">
38          <xsl:value of select="substring after($definitions/plnk:
                partnerLinkType[@name=substring after($partnerLink/
                @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                @myRole]/@portType, ':')" />
39        </xsl:when>
40        <xsl:otherwise>
41          <xsl:value of select="substring after($definitions/plnk:
                partnerLinkType[@name=substring after($partnerLink/
                @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                @partnerRole]/@portType, ':')" />
42        </xsl:otherwise>
43      </xsl:choose>
44    </xsl:variable>
45    <xsl:variable name="message" select="$definitions/wsdl:portType[
          $input and @name=$portType]/wsdl:operation[@name=$operation]/
          wsdl:input |
46                                         $definitions/wsdl:portType[not(
                                                $input) and @name=$portType
                                                ]/wsdl:operation[@name=
                                                $operation]/wsdl:output" />
47     <!  Expand the namespace  >
48    <xsl:variable name="messageNamespace" select="$message/namespace::*[
          local name() = substring before($message/@message, ':')]" />
49    <xsl:variable name="namespacePrefix" select="name(namespace::*[self::
          node() = $messageNamespace][1])" />
```

```
50      <xsl:value of select="concat($namespacePrefix, ':', substring after(
             $message/@message, ':'))" />
51    </xsl:template>
52
53    <!   Get the WSDL name of _the_ part in either the input or output
          message of
54        the operation in question.
55        Assumes that the context node is a message activity.   >
56    <xsl:template name="wsdl message part name">
57      <!   If $input = true then the name of the part in the input message
             type is returned,
58           otherwise the output message part name is returned.   >
59      <xsl:param name="input" />
60      <!   If $myRole = true then the operation is provided by this process
             ,
61           otherwise it is provided by the partner.   >
62      <xsl:param name="myRole" />
63
64      <xsl:variable name="partnerLinkName" select="@partnerLink" />
65      <xsl:variable name="operation" select="@operation" />
66      <xsl:variable name="partnerLink" select="(self::*[self::bpel:invoke
             or self::bpel:onMessage or self::bpel:receive or self::bpel:
             reply] |
67                                                   self::bpel:onEvent/bpel:
                                                         scope)
68                                                  /ancestor or self::*/bpel:
                                                         partnerLinks/bpel:
                                                         partnerLink[@name=
                                                         $partnerLinkName][1]"
                                                         />
69      <xsl:variable name="partnerLinkNamespace" select="namespace::*[local
             name() = substring before($partnerLink/@partnerLinkType, ':')]"
             />
70      <xsl:variable name="definitions" select="document(/bpel:process/bpel:
             import[@importType='http://schemas.xmlsoap.org/wsdl/']/@location
             )/wsdl:definitions[@targetNamespace=$partnerLinkNamespace]" />
71      <xsl:variable name="portType">
72        <xsl:choose>
73          <xsl:when test="$myRole">
74            <xsl:value of select="substring after($definitions/plnk:
                      partnerLinkType[@name=substring after($partnerLink/
                      @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                      @myRole]/@portType, ':')" />
75          </xsl:when>
76          <xsl:otherwise>
77            <xsl:value of select="substring after($definitions/plnk:
                      partnerLinkType[@name=substring after($partnerLink/
                      @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                      @partnerRole]/@portType, ':')" />
78          </xsl:otherwise>
79        </xsl:choose>
80      </xsl:variable>
81      <xsl:variable name="message">
82        <xsl:choose>
83          <xsl:when test="$input">
84            <xsl:value of select="substring after($definitions/wsdl:
                      portType[@name=$portType]/wsdl:operation[@name=$operation
                      ]/wsdl:input/@message, ':')" />
85          </xsl:when>
86          <xsl:otherwise>
87            <xsl:value of select="substring after($definitions/wsdl:
                      portType[@name=$portType]/wsdl:operation[@name=$operation
                      ]/wsdl:output/@message, ':')" />
88          </xsl:otherwise>
89        </xsl:choose>
90      </xsl:variable>
91      <xsl:value of select="$definitions/wsdl:message[@name=$message]/wsdl:
             part/@name" />
92    </xsl:template>
93
94      <!   Get the WSDL typing attribute of the given part in either the
             input or
95           output message of the operation in question.
96           By default assumes that the context node is a message activity.
                 >
97    <xsl:template name="wsdl message part typing">
98      <xsl:param name="messageActivity" select="." />
99      <xsl:param name="part" />
100     <!   If $input = true then the name of the part in the input message
             type is returned,
101          otherwise the output message part name is returned.   >
102     <xsl:param name="input" />
103     <!   If $myRole = true then the operation is provided by this process
             ,
104          otherwise it is provided by the partner.   >
105     <xsl:param name="myRole" />
106
107     <xsl:variable name="partnerLinkName" select="$messageActivity/
             @partnerLink" />
108     <xsl:variable name="operation" select="$messageActivity/@operation"
             />
109     <xsl:variable name="partnerLink" select="($messageActivity/self::*[
             self::bpel:invoke or self::bpel:onMessage or self::bpel:receive
             or self::bpel:reply] |
110                                                  $messageActivity/self::bpel
                                                         :onEvent/bpel:scope)
111                                                 /ancestor or self::*/bpel:
                                                         partnerLinks/bpel:
                                                         partnerLink[@name=
                                                         $partnerLinkName][1]"
                                                         />
112     <xsl:variable name="partnerLinkNamespace" select="$messageActivity/
             namespace::*[local name() = substring before($partnerLink/
             @partnerLinkType, ':')]" />
113     <xsl:variable name="definitions" select="document(/bpel:process/bpel:
             import[@importType='http://schemas.xmlsoap.org/wsdl/']/@location
             )/wsdl:definitions[@targetNamespace=$partnerLinkNamespace]" />
114     <xsl:variable name="portType">
115       <xsl:choose>
116         <xsl:when test="$myRole">
117           <xsl:value of select="substring after($definitions/plnk:
                      partnerLinkType[@name=substring after($partnerLink/
                      @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                      @myRole]/@portType, ':')" />
118         </xsl:when>
119         <xsl:otherwise>
120           <xsl:value of select="substring after($definitions/plnk:
                      partnerLinkType[@name=substring after($partnerLink/
                      @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                      @partnerRole]/@portType, ':')" />
121         </xsl:otherwise>
122       </xsl:choose>
123     </xsl:variable>
124     <xsl:variable name="message">
125       <xsl:choose>
126         <xsl:when test="$input">
127           <xsl:value of select="substring after($definitions/wsdl:
                      portType[@name=$portType]/wsdl:operation[@name=$operation
                      ]/wsdl:input/@message, ':')" />
128         </xsl:when>
129         <xsl:otherwise>
130           <xsl:value of select="substring after($definitions/wsdl:
                      portType[@name=$portType]/wsdl:operation[@name=$operation
                      ]/wsdl:output/@message, ':')" />
131         </xsl:otherwise>
132       </xsl:choose>
133     </xsl:variable>
134     <xsl:copy of select="$definitions/wsdl:message[@name=$message]/wsdl:
             part[@name=$part]/@*[namespace uri() = '' and
```

```
135
136
137  </xsl:template>
138
139   <!  Determine whether the WSDL part has an element attribute in either
           the
140        input or output message of the operation in question.
141        By default assumes that the context node is a message activity.
             >
142  <xsl:template name="wsdl message part is element">
143    <xsl:param name="messageActivity" select="." />
144    <xsl:param name="part" />
145     <!  If $input = true then the name of the part in the input message
             type is returned,
146          otherwise the output message part name is returned.  >
147    <xsl:param name="input" />
148     <!  If $myRole = true then the operation is provided by this process
             ,
149          otherwise it is provided by the partner.  >
150    <xsl:param name="myRole" />
151
152    <xsl:variable name="partnerLinkName" select="$messageActivity/
             @partnerLink" />
153    <xsl:variable name="operation" select="$messageActivity/@operation"
             />
154    <xsl:variable name="partnerLink" select="(self::*[self::bpel:invoke
             or self::bpel:onMessage or self::bpel:receive or self::bpel:
             reply] |
155                                                self::bpel:onEvent/bpel:
                                                   scope)
156                                              /ancestor or self::*/bpel:
                                                   partnerLinks/bpel:
                                                   partnerLink[@name=
                                                   $partnerLinkName][1]"
                                                   />
157    <xsl:variable name="partnerLinkNamespace" select="$messageActivity/
             namespace::*[local name() = substring before($partnerLink/
             @partnerLinkType, ':')]" />
158    <xsl:variable name="definitions" select="document(/bpel:process/bpel:
             import[@importType='http://schemas.xmlsoap.org/wsdl/']/@location
             )/wsdl:definitions[@targetNamespace=$partnerLinkNamespace]" />
159    <xsl:variable name="portType">
160      <xsl:choose>
161        <xsl:when test="$myRole">
162          <xsl:value of select="substring after($definitions/plnk:
                 partnerLinkType[@name=substring after($partnerLink/
                 @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                 @myRole]/@portType, ':')" />
163        </xsl:when>
164        <xsl:otherwise>
165          <xsl:value of select="substring after($definitions/plnk:
                 partnerLinkType[@name=substring after($partnerLink/
                 @partnerLinkType, ':')]/plnk:role[@name=$partnerLink/
                 @partnerRole]/@portType, ':')" />
166        </xsl:otherwise>
167      </xsl:choose> or
168    </xsl:variable>
169    <xsl:variable name="message">
170      <xsl:choose>
171        <xsl:when test="$input">
172          <xsl:value of select="substring after($definitions/wsdl:
                 portType[@name=$portType]/wsdl:operation[@name=$operation
                 ]/wsdl:input/@message, ':')" />
173        </xsl:when>
174        <xsl:otherwise>
175          <xsl:value of select="substring after($definitions/wsdl:
                 portType[@name=$portType]/wsdl:operation[@name=$operation
                 ]/wsdl:output/@message, ':')" />
176        </xsl:otherwise>
177      </xsl:choose>
178    </xsl:variable>  />
179    <xsl:value of select="boolean($definitions/wsdl:message[@name=
             $message]/wsdl:part[@name=$part]/@element)" />
180  </xsl:template>
181
182
183   <!  Figure out what type an activity's outbound message has.
184        Assumes that the context node is a message activity.  >
185  <xsl:template name="outbound message type">
186    <xsl:choose>
187      <xsl:when test="self::bpel:invoke">
188        <xsl:call template name="wsdl message name">
189          <xsl:with param name="input" select="true()" />
190          <xsl:with param name="myRole" select="false()" />
191        </xsl:call template>
192      </xsl:when>
193      <xsl:when test="self::bpel:reply">
194        <xsl:call template name="wsdl message name">
195          <xsl:with param name="input" select="false()" />
196          <xsl:with param name="myRole" select="true()" />
197        </xsl:call template>
198      </xsl:when>
199    </xsl:choose>
200  </xsl:template>
201
202   <!  Figure out what type an activity's inbound message has.
203        Assumes that the context node is a message activity.  >
204  <xsl:template name="inbound message type">
205    <xsl:choose>
206      <xsl:when test="self::bpel:invoke">
207        <xsl:call template name="wsdl message name">
208          <xsl:with param name="input" select="false()" />
209          <xsl:with param name="myRole" select="false()" />
210        </xsl:call template>
211      </xsl:when>
212      <xsl:when test="self::bpel:onMessage or self::bpel:receive or self
               ::bpel:onEvent">
213        <xsl:call template name="wsdl message name">
214          <xsl:with param name="input" select="true()" />
215          <xsl:with param name="myRole" select="true()" />
216        </xsl:call template>
217      </xsl:when>
218    </xsl:choose>
```

```
219 </xsl:template>
220
221  <!  Figure out what part an activity's outbound message has.
222      Assumes that the context node is a message activity.  >
223 <xsl:template name="outbound message part">
224   <xsl:choose>
225     <xsl:when test="self::bpel:invoke">
226       <xsl:call template name="wsdl message part name">
227         <xsl:with param name="input" select="true()" />
228         <xsl:with param name="myRole" select="false()" />
229       </xsl:call template>
230     </xsl:when>
231     <xsl:when test="self::bpel:reply">
232       <xsl:call template name="wsdl message part name">
233         <xsl:with param name="input" select="false()" />
234         <xsl:with param name="myRole" select="true()" />
235       </xsl:call template>
236     </xsl:when>
237   </xsl:choose>
238 </xsl:template>
239
240  <!  Figure out what part an activity's inbound message has.
241      Assumes that the context node is a message activity.  >
242 <xsl:template name="inbound message part">
243   <xsl:choose>
244     <xsl:when test="self::bpel:invoke">
245       <xsl:call template name="wsdl message part name">
246         <xsl:with param name="input" select="false()" />
247         <xsl:with param name="myRole" select="false()" />
248       </xsl:call template>
249     </xsl:when>
250     <xsl:when test="self::bpel:onMessage or self::bpel:receive or self
            ::bpel:onEvent">
251       <xsl:call template name="wsdl message part name">
252         <xsl:with param name="input" select="true()" />
253         <xsl:with param name="myRole" select="true()" />
254       </xsl:call template>
255     </xsl:when>
256   </xsl:choose>
257 </xsl:template>
258
259  <!  Figure out what typing a named part of an activity's inbound
         message has.
260      By default assumes that the context node is a message activity.
           >
261 <xsl:template name="inbound message part typing">
262   <xsl:param name="messageActivity" select="." />
263   <xsl:param name="part" />
264   <xsl:choose>
265     <xsl:when test="$messageActivity/self::bpel:invoke">
266       <xsl:call template name="wsdl message part typing">
267         <xsl:with param name="messageActivity" select="$messageActivity
              " />
268         <xsl:with param name="part" select="$part" />
269         <xsl:with param name="input" select="false()" />
270         <xsl:with param name="myRole" select="false()" />
271       </xsl:call template>
272     </xsl:when>
273     <xsl:when test="$messageActivity/self::bpel:onMessage or
            $messageActivity/self::bpel:receive or $messageActivity/self::
            bpel:onEvent">
274       <xsl:call template name="wsdl message part typing">
275         <xsl:with param name="messageActivity" select="$messageActivity
              " />
276         <xsl:with param name="part" select="$part" />
277         <xsl:with param name="input" select="true()" />
278         <xsl:with param name="myRole" select="true()" />
279       </xsl:call template>
280     </xsl:when>
281   </xsl:choose>
282 </xsl:template>
283
284  <!  Figure out what whether a named part of an activity's inbound
         message is
285      an element.
286      By default assumes that the context node is a message activity.
           >
287 <xsl:template name="inbound message part is element">
288   <xsl:param name="messageActivity" select="." />
289   <xsl:param name="part" />
290   <xsl:choose>
291     <xsl:when test="$messageActivity/self::bpel:invoke">
292       <xsl:call template name="wsdl message part is element">
293         <xsl:with param name="messageActivity" select="$messageActivity
              " />
294         <xsl:with param name="part" select="$part" />
295         <xsl:with param name="input" select="false()" />
296         <xsl:with param name="myRole" select="false()" />
297       </xsl:call template>
298     </xsl:when>
299     <xsl:when test="$messageActivity/self::bpel:onMessage or
            $messageActivity/self::bpel:receive or $messageActivity/self::
            bpel:onEvent">
300       <xsl:call template name="wsdl message part is element">
301         <xsl:with param name="messageActivity" select="$messageActivity
              " />
302         <xsl:with param name="part" select="$part" />
303         <xsl:with param name="input" select="true()" />
304         <xsl:with param name="myRole" select="true()" />
305       </xsl:call template>
306     </xsl:when>
307   </xsl:choose>
308 </xsl:template>
309
310  <!  Figure out what whether a named part of an activity's outbound
         message is
311      an element.
312      By default assumes that the context node is a message activity.
           >
313 <xsl:template name="outbound message part is element">
314   <xsl:param name="messageActivity" select="." />
315   <xsl:param name="part" />
316   <xsl:choose>
317     <xsl:when test="$messageActivity/self::bpel:invoke">
318       <xsl:call template name="wsdl message part is element">
319         <xsl:with param name="messageActivity" select="$messageActivity
              " />
320         <xsl:with param name="part" select="$part" />
321         <xsl:with param name="input" select="true()" />
322         <xsl:with param name="myRole" select="false()" />
323       </xsl:call template>
324     </xsl:when>
325     <xsl:when test="$messageActivity/self::bpel:reply">
326       <xsl:call template name="wsdl message part is element">
327         <xsl:with param name="messageActivity" select="$messageActivity
              " />
328         <xsl:with param name="part" select="$part" />
329         <xsl:with param name="input" select="false()" />
330         <xsl:with param name="myRole" select="true()" />
331       </xsl:call template>
332     </xsl:when>
333   </xsl:choose>
334 </xsl:template>
335
336  <!  Make explicit that the message parts are copied to a temporary
         variable.
337      Assumes that the context node is a message activity.  >
338 <xsl:template name="copy to parts explicitly">
339   <xsl:variable name="uniqueElementName">
340     <xsl:call template name="unique element name">
```

```
341         <xsl:with param name="element" select="." />
342       </xsl:call template>
343     </xsl:variable>
344
345     <xsl:for each select="bpel:toParts">
346       <bpel:assign>
347         <xsl:for each select="bpel:toPart">
348           <bpel:copy>
349             <xsl:variable name="variableName" select="@fromVariable" />
350
351              <!  Are both source and target elements?  >
352             <xsl:variable name="messagePartIsElement">
353               <xsl:call template name="outbound message part is element">
354                 <xsl:with param name="messageActivity" select="../.." />
355                 <xsl:with param name="part" select="@part" />
356               </xsl:call template>
357             </xsl:variable>
358             <xsl:if test="$messagePartIsElement = 'true' and ancestor::*/
                    bpel:variables/bpel:variable[@name=$variableName][1]/
                    @element">
359               <xsl:attribute name="keepSrcElementName">yes</xsl:attribute
                  >
360             </xsl:if>
361
362             <bpel:from>
363               <xsl:attribute name="variable">
364                 <xsl:value of select="@fromVariable" />
365               </xsl:attribute>
366             </bpel:from>
367             <bpel:to>
368               <xsl:attribute name="variable">
369                 <xsl:value of select="concat($uniqueElementName, $tmp
                        input message variable postfix)" />
370               </xsl:attribute>
371               <xsl:attribute name="part">
372                 <xsl:value of select="@part" />
373               </xsl:attribute>
374             </bpel:to>
375           </bpel:copy>
376         </xsl:for each>
377       </bpel:assign>
378     </xsl:for each>
379   </xsl:template>
380
381    <!  Make explicit that the message parts are copied from a temporary
         variable.
382        Assumes that the context node is a message activity.  >
383   <xsl:template name="copy from parts explicitly">
384     <xsl:variable name="uniqueElementName">
385       <xsl:call template name="unique element name">
386         <xsl:with param name="element" select="." />
387       </xsl:call template>
388     </xsl:variable>
389
390     <xsl:for each select="bpel:fromParts">
391       <bpel:assign>
392         <xsl:for each select="bpel:fromPart">
393           <bpel:copy>
394             <xsl:variable name="variableName" select="@toVariable" />
395
396              <!  Are both source and target elements?  >
397             <xsl:variable name="messagePartIsElement">
398               <xsl:call template name="inbound message part is element">
399                 <xsl:with param name="messageActivity" select="../.." />
400                 <xsl:with param name="part" select="@part" />
401               </xsl:call template>
402             </xsl:variable>
403             <xsl:variable name="variableIsElement">
404               <xsl:choose>
405                 <xsl:when test="../../self::bpel:onEvent"><xsl:value of
                        select="$messagePartIsElement" /></xsl:when>
406                 <xsl:otherwise><xsl:value of select="ancestor::*/bpel:
                        variables/bpel:variable[@name=$variableName][1]/
                        @element" /></xsl:otherwise>
407               </xsl:choose>
408             </xsl:variable>
409             <xsl:if test="$messagePartIsElement = 'true' and
                    $variableIsElement = 'true'">
410               <xsl:attribute name="keepSrcElementName">yes</xsl:attribute
                  >
411             </xsl:if>
412
413             <bpel:from>
414               <xsl:attribute name="variable">
415                 <xsl:value of select="concat($uniqueElementName, $tmp
                        output message variable postfix)" />
416               </xsl:attribute>
417               <xsl:attribute name="part">
418                 <xsl:value of select="@part" />
419               </xsl:attribute>
420             </bpel:from>
421             <bpel:to>
422               <xsl:attribute name="variable">
423                 <xsl:value of select="@toVariable" />
424               </xsl:attribute>
425             </bpel:to>
426           </bpel:copy>
427         </xsl:for each>
428       </bpel:assign>
429     </xsl:for each>
430   </xsl:template>
431
432    <!  Make explicit that the input element is copied to the single part
         of a
433        temporary variable.
434        Assumes that the context node is a message activity.  >
435   <xsl:template name="copy input element explicitly">
436     <xsl:param name="inputVariable" />
437
438     <xsl:variable name="tmpInputMessageName">
439       <xsl:call template name="unique element name">
440         <xsl:with param name="element" select="." />
441         <xsl:with param name="postfix" select="$tmp input message
                variable postfix" />
442       </xsl:call template>
443     </xsl:variable>
444
445     <bpel:assign>
446       <bpel:copy keepSrcElementName="yes">
447         <bpel:from>
448           <xsl:attribute name="variable">
449             <xsl:value of select="$inputVariable" />
450           </xsl:attribute>
451         </bpel:from>
452         <bpel:to>
453           <xsl:attribute name="variable">
454             <xsl:value of select="$tmpInputMessageName" />
455           </xsl:attribute>
456           <xsl:attribute name="part">
457             <xsl:call template name="outbound message part" />
458           </xsl:attribute>
459         </bpel:to>
460       </bpel:copy>
461     </bpel:assign>
462   </xsl:template>
463
464    <!  Make explicit that the output element is copied to the single part
         of a
465        temporary variable.
```

```
466         Assumes that the context node is a message activity.  >
467   <xsl:template name="copy output element explicitly">
468     <xsl:param name="outputVariable" />
469
470     <xsl:variable name="tmpOutputMessageName">
471       <xsl:call template name="unique element name">
472         <xsl:with param name="element" select="." />
473         <xsl:with param name="postfix" select="$tmp output message
              variable postfix" />
474       </xsl:call template>
475     </xsl:variable>
476
477     <bpel:assign>
478       <bpel:copy keepSrcElementName="yes">
479         <bpel:from>
480           <xsl:attribute name="variable">
481             <xsl:value of select="$tmpOutputMessageName" />
482           </xsl:attribute>
483           <xsl:attribute name="part">
484             <xsl:call template name="inbound message part" />
485           </xsl:attribute>
486         </bpel:from>
487         <bpel:to>
488           <xsl:attribute name="variable">
489             <xsl:value of select="$outputVariable" />
490           </xsl:attribute>
491         </bpel:to>
492       </bpel:copy>
493     </bpel:assign>
494   </xsl:template>
495
496   <!  Create temporary variables for the current messaging activity if
        it
497       uses <toParts>, <fromParts> or element variables.
498       Assumes that the context node is a message activity.  >
499   <xsl:template name="message activity temp variables">
500     <xsl:param name="inputVariable" />
501     <xsl:param name="outputVariable" />
502
503     <xsl:variable name="uniqueElementName">
504       <xsl:call template name="unique element name">
505         <xsl:with param name="element" select="." />
506       </xsl:call template>
507     </xsl:variable>
508
509     <xsl:variable name="inputElement" select="count(ancestor::*/bpel:
          variables/bpel:variable[@name=$inputVariable][1]/@element) = 1"
          />
510     <xsl:variable name="outputElement" select="count(self::*[self::bpel:
          invoke or self::bpel:onMessage or self::bpel:receive or self::
          bpel:reply]/ancestor::*/bpel:variables/bpel:variable[@name=
          $outputVariable][1]/@element |
511                                                   self::bpel:onEvent[
                                                         @element]) = 1"
                                                         />
512
513     <xsl:if test="bpel:toParts or $inputElement">
514       <bpel:variable>
515         <xsl:attribute name="name">
516           <xsl:value of select="concat($uniqueElementName, $tmp input
                message variable postfix)" />
517         </xsl:attribute>
518         <xsl:attribute name="messageType">
519           <xsl:call template name="outbound message type" />
520         </xsl:attribute>
521       </bpel:variable>
522     </xsl:if>
523     <xsl:if test="bpel:fromParts or $outputElement">
524       <bpel:variable>
525         <xsl:attribute name="name">
526           <xsl:value of select="concat($uniqueElementName, $tmp output
                message variable postfix)" />
527         </xsl:attribute>
528         <xsl:attribute name="messageType">
529           <xsl:call template name="inbound message type" />
530         </xsl:attribute>
531       </bpel:variable>
532     </xsl:if>
533   </xsl:template>
534
535   <!  Create temporary variables for the given set of message activities
        .  >
536   <xsl:template name="message activities temp variables">
537     <xsl:param name="messageActivities" />
538
539     <xsl:for each select="$messageActivities">
540       <xsl:call template name="message activity temp variables">
541         <xsl:with param name="inputVariable">
542           <xsl:choose>
543             <xsl:when test="self::bpel:invoke">
544               <xsl:value of select="@inputVariable" />
545             </xsl:when>
546             <xsl:when test="self::bpel:reply">
547               <xsl:value of select="@variable" />
548             </xsl:when>
549           </xsl:choose>
550         </xsl:with param>
551         <xsl:with param name="outputVariable">
552           <xsl:choose>
553             <xsl:when test="self::bpel:invoke">
554               <xsl:value of select="@outputVariable" />
555             </xsl:when>
556             <xsl:when test="self::bpel:onMessage or self::bpel:receive or
                  self::bpel:onEvent">
557               <xsl:value of select="@variable" />
558             </xsl:when>
559           </xsl:choose>
560         </xsl:with param>
561       </xsl:call template>
562     </xsl:for each>
563   </xsl:template>
564
565   <!  Decide whether temporary variables are needed for the current
        messaging
566       activity if it uses <toParts>, <fromParts> or element variables.
567       By default, assumes that the context node is a message activity.
          >
568   <xsl:template name="message activity needs temp variables">
569     <xsl:param name="activity" select="." />
570     <xsl:param name="inputVariable" />
571     <xsl:param name="outputVariable" />
572
573     <xsl:variable name="inputElement" select="count($activity/ancestor
          ::*/bpel:variables/bpel:variable[@name=$inputVariable][1]/
          @element) = 1" />
574     <xsl:variable name="outputElement" select="count($activity/self::*[
          self::bpel:invoke or self::bpel:onMessage or self::bpel:receive
          or self::bpel:reply]/ancestor::*/bpel:variables/bpel:variable[
          @name=$outputVariable][1]/@element |
575                                                   $activity/self::bpel
                                                        :onEvent[
                                                        @element]) = 1"
                                                        />
576
577     <xsl:value of select="$activity/bpel:toParts or $inputElement or
          $activity/bpel:fromParts or $outputElement" />
578   </xsl:template>
579
580   <!  Decide whether temporary variables are needed for the given set of
        message activities.  >
```

```
581 <xsl:template name="message activities need temp variables">
582   <xsl:param name="messageActivities" />
583
584   <xsl:choose>
585     <xsl:when test="0 = count($messageActivities/*)">
586       <xsl:value of select="false()" />
587     </xsl:when>
588     <xsl:otherwise>
589       <xsl:variable name="first activity" select="$messageActivities[1]
            " />
590       <xsl:variable name="first activity needs temp variable">
591         <xsl:call template name="message activity needs temp variables"
              >
592           <xsl:with param name="activity" select="$first activity" />
593           <xsl:with param name="inputVariable">
594             <xsl:choose>
595               <xsl:when test="$first activity[self::bpel:invoke]">
596                 <xsl:value of select="$first activity/@inputVariable"
                      />
597               </xsl:when>
598               <xsl:when test="$first activity[self::bpel:reply]">
599                 <xsl:value of select="$first activity/@variable" />
600               </xsl:when>
601             </xsl:choose>
602           </xsl:with param>
603           <xsl:with param name="outputVariable">
604             <xsl:choose>
605               <xsl:when test="$first activity[self::bpel:invoke]">
606                 <xsl:value of select="$first activity/@outputVariable"
                      />
607               </xsl:when>
608               <xsl:when test="$first activity[self::bpel:onMessage or
                    self::bpel:receive]">
609                 <xsl:value of select="$first activity/@variable" />
610               </xsl:when>
611             </xsl:choose>
612           </xsl:with param>
613         </xsl:call template>
614       </xsl:variable>
615
616       <xsl:choose>
617         <xsl:when test="$first activity needs temp variable = 'true'">
618           <xsl:value of select="true()" />
619         </xsl:when>
620         <xsl:otherwise>
621           <xsl:call template name="message activities need temp
                variables">
622             <xsl:with param name="messageActivities" select="
                  $messageActivities/*[position() &gt; 1]" />
623           </xsl:call template>
624         </xsl:otherwise>
625       </xsl:choose>
626     </xsl:otherwise>
627   </xsl:choose>
628
629 </xsl:template>
630
631 </xsl:stylesheet>
```

# Part IV

# Scalable Simulation of Stochastic Bigraphs

# Chapter 7

# Towards Scalable Simulation of Stochastic Bigraphs

**Espen Højsgaard and Jean Krivine**

**Abstract**

We report on the progress of the development and implementation of an efficient and scalable simulation algorithm for stochastic bigraphical reactive systems (BRSs).

The starting point is the stochastic simulation algorithm for the $\kappa$-calculus by Danos et al. [12] (henceforth KaSim). Since the $\kappa$-calculus is a graphical formalism with a straightforward BRS representation, we are hopeful that their algorithm generalizes to BRSs. The KaSim algorithm relies on a number of concepts that have not previously been developed for BRSs: embeddings, localized matching, redex and agent modifications, and fine-grained conflict/-causality analysis at the level of rules exploiting the notion of modification. In this report, we rigorously develop bigraph embeddings and redex/agent modifications, give an algorithm for localized matching, and outline a fine-grained conflict/causality analysis.

Our implementation strategy is to represent the bigraphical structures as directly as possible, as we believe that this eases implementation and increases trust in correctness. However, it is difficult to directly represent the structures of the usual presentation of the theory of BRSs: any non-trivial BRS contains an infinite number of ground reaction rules, since the set of rules must be closed under support equivalence and a parametric reaction rules generates an infinite set of ground reaction rules. In addition, the usual presentation of the dynamic theory of BRSs combines poorly with the stochastic semantics: the stochastic semantics rely on support, i.e., concrete bigraphs, while dynamics are defined up to support equivalence. We therefore develop, and prove equivalent, an alternative dynamic theory for BRSs without these problems: (i) the set of rules need not be closed under support equivalence, (ii) parametric reaction rules are first-class citizens, and (iii) integrates a (generalized) stochastic semantics. The development is based on the more general theory of reactive systems, and is thus applicable in more settings than just (stochastic) BRSs.

The completed parts of our work have been implemented in a prototype called the Stochastic Bigraphical Abstract Machine (SBAM), which currently allows stochastic simulation of BRSs where each redex consists of a single connected component and is *solid*, i.e., matches are determined by support translations of its nodes.

**Preface** This chapter consists of the technical report

E. Højsgaard and J. Krivine. *Towards Scalable Simulation of Stochastic Bigraphs.* Technical Report TR-2011-148, IT University of Copenhagen, December 2011.

## 7.1 Introduction

The theory of bigraphs arose as a generalization of process calculi, and provides a unifying framework for modeling systems of mobile and communicating agents. The theory excels in that it provides a general method for deriving labeled transition systems (LTSs) from reaction semantics, with the nice property that, in the derived LTSs, bisimulation is a congruence.

But it is also a theory with a nice graphical representation, which enables models that are more intuitive than corresponding process calculi models. This has recently been exploited by Damgaard et al. [9, 10] and Bacci et al. [1] to give models of protein interaction and dynamic compartmentalization in cellular biology. In combination with Krivine et al.'s stochastic semantics for bigraphs [25], these works enable us to construct models of biological cells that may be simulated by a computer; the only thing missing is a simulator for stochastic bigraphs, which is what we have set out to build.

Our starting point is the efficient and scalable simulator for the $\kappa$-calculus [11] by Danos et al. and its underlying algorithm (which we call KaSim) [12]: Since the $\kappa$-calculus is a graphical formalism with a straightforward BRS representation, we are hopeful that KaSim generalizes to bigraphs. The KaSim algorithm relies on a number of concepts that have not previously been developed for BRSs: embeddings, localized matching, redex and agent modifications, and fine-grained causality analysis at the level of rules exploiting the notion of modification. In this report, we rigorously develop bigraph embeddings and redex/agent modifications, and outline algorithms for localized matching and fine-grained causality analysis:

**bigraph embeddings:**
: We develop a general theory of bigraph embeddings that are isomorphic to decompositions of the form $H = C \circ ((G \otimes \mathsf{id}_X) \circ D \otimes \mathsf{id}_{\langle k,Y \rangle})$ where $D$ is discrete (some detail omitted). In particular, embeddings of redexes into agents are isomorphic to matches. We also show that embeddings of so-called solid bigraphs are determined by support translations of their nodes.

**edit scripts:**
: We propose a set of minimal modifications to a redex, called *edits*, and show how the modification of a redex can be transferred to an agent through an embedding, giving rise to an alternative, but equivalent, way to define reaction. We prove that a sequence of edits, an *edit script*, can realize any parametric reaction rule and vice versa.

**rule activation and inhibition:**
: We outline an approach to characterizing causality and conflict at the level of rules, called respectively rule activation and rule inhibition, based on the idea of characterizing overlaps between bigraphs as a set of pullbacks in the category of embeddings.

**anchored matching:**
: We give a localized matching algorithm, based on the idea of expanding a partial embedding of a redex to total embeddings.

However, before we can hope to generalize and implement the KaSim algorithm for bigraphs, we must develop a formulation of stochastic bigraphs that is more amenable to implementation than the usual formulations. For example, neither Milner's definition of bigraph dynamics [29] nor Krivine et al.'s stochastic semantics for bigraphs [25] lend themselves easily to implementation for the following reasons:

- The various definitions, e.g., the definitions of matches, reactions, and stochastic rates, rely on *support*, i.e., node and edge identities. But at the same time, the same definitions always close under support equivalence, whereby it becomes unclear how to handle support in practice.

- The, from a modeling perspective, essential concept of *parametric* reaction rules are treated as generators of infinite families of *ground* (non-parametric) reaction rules, which clearly cannot be represented directly in an implementation.

Furthermore, the formulation of stochastic bigraphs in [25] have two minor deficiencies: it only defines stochastic semantics for BRSs with linear rules and so-called *solid* redexes, and there is a gap between the definition of the reaction semantics and the stochastic semantics, as they rely on seemingly different definitions of matches. We develop a theory of *stochastic parametric reactive systems (SPRS)* which unifies and generalizes Milner's reactive systems and the stochastic semantics of Krivine et al., while avoiding the above issues. Similar to Milner's reactive systems, we define SPRSs at the more abstract level of s-categories, of which bigraphs are an instance.

### 7.1.1 Related work

**Parametric Reactive Systems** Our SPRSs are related to, and their formulation inspired by, the *parametric reactive systems* of Debois, where parametric reaction rules are also first-class citizens [13]. However, contrary to our formulation, Debois does not make explicit that context and parameter may be connected without the involvement of the redex. This has the consequence that bigraphical reaction rules become generators of infinite families of rules. Furthermore, we go further than Debois, by formally showing that our formulation is equivalent to the usual (non-parametric) reactive systems.

**Bigraph Implementations** A number of implementations of bigraphs are being developed at various institutions. Unfortunately, it is hard to find the implementations themselves or papers describing them, but here is a complete list of the implementations which we are aware of:

**BigMC:** A model checker for bigraphs which includes a command line interface and visualization [4].

**bigraphspace:** A Java library which provides a tuple-space-like API based on bigraphs [21].

**Big Red:** A graphical editor for bigraphs with easily extensible support for various file formats [17].

**BigWB:** A graphical workbench for bigraphs, aiming at providing a unifying GUI for the various bigraph tools (work in progress, no website or papers at the time of writing).

**BPL Tool:** A command line tool for experimenting with (abstract) binding bigraphs based on Damgaard et al.'s inductive characterization of matching in binding bigraphs [6] [7, 20, 23].

**CLF based:** Beauquier and Schürmann have given a model of bigraphs in the type theory CLF [3], and thus CLF implementations, such as Celf [33], may be used for bigraphs.

**DBtk:** A tool for directed bigraphs, which provides calculation of IPOs, matching, and visualization [2].

**SAT based algorithm:** Sevegnani et al. have presented a SAT based algorithm for matching in place graphs with sharing [34] and an implementation is in progress based on MiniSAT [14].

**Bigraphs vs Graph Transformation** Ehrig and Milner have explored the connection between traditional graph transformation and the bigraph approach [15, 28]. They have in particular focused on the fact that in the traditional approach, graphs are objects in a category whereas they are morphisms in the bigraphical approach. Following ideas by Cattani [8] and Sobocinsky [35] they use the cospan construction to turn objects into morphisms, and the coslice construction to

turn morphisms into objects, thereby enabling transfer of results. As part of this work, Milner defines embeddings of ground link graphs and show that they are isomorphic to link graph contexts [28]; this definition serves as the basis of our bigraph embeddings.

**Stochastic Simulation of Process Algebra** A number of simulators for various stochastic process algebras have been developed in recent years, most notably

**KaSim:** A simulator for the $\kappa$-calculus based on the Gillespie-based algorithm presented in [12].

**PEPA:** The PEPA Eclipse Plug-in Project [30] provides an editor and various tools for PEPA [22], including a stochastic simulator.

**PRISM:** Though not quite a stochastic simulator – it is a *probabilistic model checker* – it supports (a subset of) PEPA models and is very efficient [26].

**SPiM:** The Stochastic Pi Machine is, as the name implies, a simulator for the stochastic $\pi$-calculus [31].

### 7.1.2 Outline of the Report

Though this report is self contained, it is *not* a gentle introduction to bigraphs: we assume that the reader has a keen intuition of bigraphs and bigraphical reactive systems and a reasonable grasp of its categorical underpinnings.

The remainder of the report is organized as follows:

**Section 7.2: Background**
We provide a terse recap of the theory of bigraphs, along with a few new related definitions and results that we shall need in the following sections.

**Section 7.3: The Simulation Algorithm**
We give an overview of the KaSim algorithm, recast to the setting of stochastic bigraphs.

**Section 7.4: Stochastic Parametric Reactive Systems**
We develop *stochastic parametric reactive systems* and prove that they generate the same abstract reactions as Milner's reactive systems.

**Section 7.5: Bigraph Embeddings**
We develop a general notion of *bigraph embeddings* which are isomorphic to certain decompositions of bigraphs. In particular, in the case of redexes, bigraph embeddings are isomorphic to matches.

**Section 7.6: Bigraph Edit Scripts**
We develop *bigraph edit scripts*, fine-grained reconfigurations of redexes that may be mediated by embeddings and generate the same reactions as bigraphical parametric reaction rules.

**Section 7.7: Rule Activation and Inhibition**
We outline a construction of fine-grained causality/conflict relations between parametric reaction rules based on pullbacks in the category of bigraph embeddings.

**Section 7.8: Anchored Matching**
We give a backtracking algorithm for extending partial bigraph embeddings to total embeddings, which, since embeddings of redexes into agents are matches, is equivalent to a localized matching algorithm.

**Section 7.9: Conclusions and Future Work**
We conclude and discuss future work.

## 7.2 Background

We provide the necessary background theory upon which this report builds: some basic mathematical preliminaries and a recap of the basic theory of bigraphs and bigraphical reactive systems (enriched with a few simple definitions and results).

### 7.2.1 Mathematical Preliminaries

We briefly review the basic mathematical notations and conventions used in this report.

**Natural Numbers and Sets** We shall frequently treat a natural number $m$ as a finite ordinal, the set of all preceding ordinals: $m = \{0, 1, \ldots, m-1\}$. We write sets as capital letters, e.g., $S, T$, or as a symbol with a tilde on top to denote a set of what that symbol denotes, e.g., $\tilde{m}$ is a set of natural numbers. For singletons $S = \{s\}$ we often use $s$ and $S$ interchangeably. We write $S - s$ and $S + s$ to mean $S \setminus \{s\}$ and $S \cup \{s\}$ respectively. We write $S \# T$ for disjoint sets, i.e., $S \cap T = \emptyset$. We write $S \uplus T$ for the union of sets known or assumed to be disjoint. We write $S + T$ for the disjoint union $\{(0, s) \mid s \in S\} \cup \{(1, t) \mid t \in T\}$, and we write $\pi_i(S_0 + S_1)$ for $S_i$. We use $\bar{\imath}$ to mean $1 - i$ for $i \in \{0, 1\}$.

**Vectors** We write vectors as a symbol with an arrow on top to denote a vector of what that symbol denotes, e.g., $\vec{m}$ is a vector of natural numbers. We write $|\vec{\cdot}|$ for the number of elements in a vector and $\vec{\cdot}_i$ $(i \in |\vec{\cdot}|)$ for the $i$th element of the vector (i.e., indices begin at 0). We write $\{\vec{\cdot}\}$ for the set $\{\vec{\cdot}_i \mid i \in |\vec{\cdot}|\}$.

**Functions** We write $\mathsf{Id}_S$ for the identity function on the set $S$. We write $\emptyset_S$ for the function whose domain and codomain are the empty set and $S$ respectively; $S$ is sometimes omitted when it is understood from the context. We write $\vec{f} : \vec{S} \to T$ for the vector consisting of functions $f_i : S_i \to T$ $(i \in |\vec{f}| = |\vec{S}|)$, and symmetrically $\vec{f} : S \to \vec{T}$ for the vector consisting of functions $f_i : S \to T_i$ $(i \in |\vec{f}| = |\vec{T}|)$. We write $\{s_0 \mapsto t_0, \ldots, s_{n-1} \mapsto t_{n-1}\}$ for the function mapping $s_i$ to $t_i$ (assuming the $s_i$ are distinct) and $\{S \mapsto t\}$ for the function that maps all elements of $S$ to $t$. For a function $f : S \to T$ we write $f - s : (S - s) \to T$ for the function defined as $(f - s)(s') = f(s')$ for $s' \in S - s$, and for sets $S' = \{s_0, \ldots, s_{n-1}\} \subseteq S$ we write $f - S$ to mean $(\cdots(f - s_0)\cdots - s_{n-1}$. Symmetrically, we write $f[s \mapsto t] : (S + s) \to (T + t)$ for the function defined as

$$f[s \mapsto t](s') = \begin{cases} t & \text{if } s = s' \\ f(s') & \text{otherwise} \end{cases}.$$

If $f : S \to T$ is a function and $S' \subseteq S$,$T' \subseteq T$, then $f \restriction_{S'}$ denotes the restriction of $f$ to $S'$ and $f \downharpoonleft^{T'}$ denotes the outward restriction of $f$ to $T'$, i.e., $f \downharpoonleft^{T'} (s) = t$ iff $f(s) \in T'$, and we write $f(S')$ to mean $\{f(s) \mid s \in S'\}$. For an injective function $f : S \rightarrowtail T$ we write $f^{-1} : \mathsf{rng}(f) \rightarrowtail S$ for its inverse which is total, injective and surjective. For a function $f : S \to T$ we write $f^{-1}$ for its preimage function defined as $f^{-1}(t) = \{s \in S \mid f(s) = t\} : T \to \mathcal{P}(S)$, and we extend the preimage function to sets $T' \subseteq T$ as follows: $f^{-1}(T') = \{s \in S \mid f(s) \in T'\} : \mathcal{P}(T) \to \mathcal{P}(S)$. For a function $f : S \to \mathcal{P}(T)$ we shall write $f(S')$ to mean $\bigcup_{s \in S'} f(s)$ when $S' \subseteq S$, and, by extension, we shall sometimes write $\mathsf{rng}(f)$ to mean $f(S)$ when the context prevents ambiguity. We write $f \downharpoonleft^{T'}$ for the outward restriction of $f$ to $T' \subseteq T$, i.e., $f \downharpoonleft^{T'} (s) = f(s) \cap T'$. We say that $f$ is *fully injective* iff $\forall s, s' \in S : f(s) \cap f(s') \neq \emptyset \Rightarrow s = s'$. When $f$ is fully injective we write $f^{-1}(t)$ $(t \in \mathsf{rng}(f))$ for the unique $s$ for which $t \in f(s)$. Note that for a fully injective function $f$ we have $f^{-1} : \mathsf{rng}(f) \rightarrowtail S$, $t \in f(f^{-1}(t))$, and $\{s\} = f^{-1}(f(s))$. For two functions $f$ and $g$ with disjoint domains $S$ and $T$ we write $f \uplus g$ for the function with domain $S \uplus T$ such that $(f \uplus g) \restriction_S = f$ and $(f \uplus g) \restriction_T = g$. We write $\rightarrowtail$ to indicate that a function is injective and $\rightharpoonup$ indicates partiality. We write $\hookrightarrow$ ($\hookrightarrow$) for (partial) graph embeddings.

**Stochastics** We use $rand(S, f)$ to denote a random variate of a stochastic variable with outcomes $S$ and probability distribution $f$.

## 7.2.2 Bigraphs

This section is not meant as an introduction to bigraphs, but rather as a simplified and unified reference for the parts of the bigraphical theory that we shall need in this report. In other words, we assume the reader is already familiar with bigraphs; please refer to Milner's recent book [29] for an introduction to, and more complete treatment of, the theory of bigraphs.

Below we recall the definitions, notation, conventions, and terminology of bigraphs that we shall use in this report. We follow Milner's book closely [29], most of the time verbatim, but we have in a few places omitted details that are irrelevant for our purposes (most significantly the notions of width and sorting), slightly tweaked the notation to improve the readability of this report, and corrected some minor mistakes.

We also prove a few straightforward results that will shall need later and introduce notation for extracting subgraphs from bigraphs.

### Concrete Bigraphs

We assume that names, node-identifiers, and edge-identifiers are drawn from three countably infinite, mutually disjoint, sets: $\mathcal{X}$, $\mathcal{V}$, and $\mathcal{E}$, respectively.

Nodes in bigraphs are assigned kinds, called *controls*, and the controls specify the number of ports nodes of that kind have:

**Definition 7.2.1** (basic signature (after [29, Def. 1.1])). A *basic signature* takes the form $(\mathcal{K}, ar)$. It has a set $\mathcal{K}$ whose elements are kinds of node called *controls*, and a map $ar : \mathcal{K} \to \mathbb{N}$ assigning an *arity*, a natural number, to each control. The signature is denoted by $\mathcal{K}$ when the arity is understood. A bigraph over $\mathcal{K}$ assigns to each node a control, whose arity indexes the *ports* of a node, where links may be connected. □

A bigraph is a pair of a place graph and a link graph, called its constituents. Bigraphs and their constituents are either *concrete* or *abstract*; in this section we are concerned with concrete bigraphs, and we shall defer the discussion of abstract bigraphs to Section 7.2.2. We denote concrete bigraphs and their constituents by upper case letters $A, \ldots, H$.

We define concrete place and link graphs separately, and then combine them into bigraphs:

**Definition 7.2.2** (concrete place graph (after [29, Def. 2.1])). A *concrete place graph*

$$F = (V_F, ctrl_F, prnt_F) : m \to n$$

is a triple having an inner face $m$ and an outer face $n$, both finite ordinals. These index respectively the *sites* and *roots* of the place graph. $F$ has a finite set $V_F \subset \mathcal{V}$ of *nodes*, a *control map* $ctrl_F : V_F \to \mathcal{K}$ and a *parent map*

$$prnt_F : m \uplus V_F \to V_F \uplus n$$

which is acyclic, i.e., if $prnt^i_F(v) = v$ for some $v \in V_F$ then $i = 0$. We shall call nodes, roots and sites the *places* of $F$.

We say that a root $i$ is *idle* iff there is no $c \in V_F \uplus m$ with $prnt_F(c) = i$. A site $i$ is *guarding* iff its parent is a node, i.e., $prnt_F(i) \in V_F$. Sites and nodes are *siblings* if they have the same parent. A place graph with inner face $m = 0$ is called *ground* or an *agent*. □

**Definition 7.2.3** (concrete link graph (after [29, Def. 2.2])). A *concrete link graph*

$$F = (V_F, E_F, ctrl_F, link_F) : X \to Y$$

is a quadruple having an inner face $X$ and an outer face $Y$, both finite subsets of $\mathcal{X}$, called respectively the *inner* and *outer names* of the link graph. $F$ has finite sets $V_F \subset \mathcal{V}$ of *nodes* and $E_F \subset \mathcal{E}$ of *edges*, a *control map* $ctrl_F : V_F \to \mathcal{K}$ and a *link map*

$$link_F : X \uplus P_F \to E_F \uplus Y$$

where $P_F \stackrel{\text{def}}{=} \{(v,i) \mid v \in V_F \wedge i \in ar(ctrl(v))\}$ is the set of *ports* of $F$. Thus $(v,i)$ is the $i$th port of node $v$. We write $P_{v,ctrl_F}$ for the ports of node $v$ given control map $ctrl_F$, i.e., it denotes the set $\{(v,i) \mid i \in ar(ctrl_F(v))\}$; we extend the notation to sets of nodes, $P_{V,ctrl_F} \stackrel{\text{def}}{=} \bigcup_{v \in V} P_{v,ctrl_F}$, and we sometimes omit *ctrl* when it is evident from the context. We shall call $X \uplus P_F$ the *points* of $F$, and $E_F \uplus Y$ its *links*. We say that outer names are *open* links and edges are *closed* links.

We say that a link $l$ is *idle* iff there is no $p \in X \uplus P_F$ with $link_F(p) = l$. An inner name $x$ is *guarding* iff its link is connected to a node, i.e., $\exists (v,i) \in P_F : link_F(v,i) = link_F(x)$. Points and inner names are *siblings* if they are connected to the same link. A link graph with inner face $X = \emptyset$ is called *ground* or an *agent*. □

**Definition 7.2.4** (concrete bigraph (after [29, Def. 2.3])). An *interface*, denoted by upper case letters $I, J, K$, for bigraphs is a pair $I = \langle m, X \rangle$ of a place graph interface and a link graph interface. We call $m$ the width of $I$, and we say that $I$ is *nullary*, *unary* or *multiary* according as $m$ is 0, 1 or >1. A *concrete bigraph*

$$F = (V_F, E_F, ctrl_F, prnt_F, link_F) : \langle k, X \rangle \to \langle m, Y \rangle$$

consists of a concrete place graph $F^P = (V_F, ctrl_F, prnt_F) : k \to n$ and a concrete link graph $F^L = (V_F, E_F, ctrl_F, link_F) : X \to Y$. We write the concrete bigraph as $F = \langle F^P, F^L \rangle$. We shall call $V_F \uplus E_F \uplus k \uplus X \uplus m \uplus Y$ the *entities* of $F$.

A bigraph is called *ground* or an *agent* iff both of its constituents are ground. A bigraph is called *discrete* iff it has no edges and its link map is a bijection. A bigraph is called *prime* iff is has no inner names and a unary outer face. □

It should be clear from the above definitions, that the choice of node- and edge-identifiers have no impact on the structure of the graphs. We shall now make this precise:

**Definition 7.2.5** (support for bigraphs (after [29, Def. 2.4])). To each place graph, link graph or bigraph $F$ is assigned a finite set $|F|$, its *support*. For a place graph we define $|F| = V_F$, and for a link graph or bigraph we define $|F| = V_F \uplus E_F$.

For two bigraphs $F : I \to J$ and $G : I \to J$, a support translation $\rho : |F| \to |G|$ from $F$ to $G$ consists of a pair of bijections $\rho_V : V_F \to V_G$ and $\rho_E : E_F \to E_G$ that respect structure, in the following sense:

(i) $\rho$ preserves controls, i.e., $ctrl_G \circ \rho_V = ctrl_F$. It follows that $\rho$ induces a bijection $\rho_P : P_F \to P_G$ on ports, defined by $\rho_P((v,i)) \stackrel{\text{def}}{=} (\rho_V(v), i)$.

(ii) $\rho$ commutes with the structural maps as follows:

$$prnt_G \circ (\mathsf{Id}_m \uplus \rho_V) = (\mathsf{Id}_n \uplus \rho_V) \circ prnt_F$$
$$link_G \circ (\mathsf{Id}_X \uplus \rho_P) = (\mathsf{Id}_Y \uplus \rho_E) \circ link_F\,.$$

Given $F$ and the bijection $\rho$, these conditions uniquely determine $G$. We therefore denote $G$ by $\rho \bullet F$, and call it the *support translation of $F$ by $\rho$*. We call $F$ and $G$ *support equivalent*, and we write $F \doteq G$, if such a support translation exists. Support translations $\rho : |F| \to |F|$ where $\rho \bullet F = F$ are called *support automorphisms* if they are not identities.

Support translation is defined similarly for place graphs and link graphs.

□

The interfaces of bigraphs and their constituents enable their composition: if the inner face of a bigraph is the same the outer face of another, they may be composed:

**Definition 7.2.6** (composition and identities (after [29, Def. 2.5]))**.** We define composition for place graphs and link graphs separately, and then combine them for the composition of bigraphs.

- If $F : k \to m$ and $G : m \to n$ are two place graphs with $|F| \# |G|$, their composite

  $$G \circ F = (V, ctrl, prnt) : k \to n$$

  has nodes $V = V_F \uplus V_G$ and control map $ctrl = ctrl_F \uplus ctrl_G$. Its parent map $prnt$ is defined as follows: If $w \in k \uplus V_F \uplus V_G$ is a site or node of $G \circ F$ then

  $$prnt(w) \stackrel{\text{def}}{=} \begin{cases} prnt_F(w) & \text{if } w \in k \uplus V_F \text{ and } prnt_F(w) \in V_F \\ prnt_G(j) & \text{if } w \in k \uplus V_F \text{ and } prnt_F(w) = j \in m \\ prnt_G(w) & \text{if } w \in V_G. \end{cases}$$

  The identity place graph at $m$ is $\mathsf{id}_m \stackrel{\text{def}}{=} (\emptyset, \emptyset, \mathsf{Id}_m) : m \to m$.

- If $F : X \to Y$ and $G : Y \to Z$ are two link graphs with $|F| \# |G|$, their composite

  $$G \circ F = (V, E, ctrl, link) : X \to Z$$

  has $V = V_F \uplus V_G$, $E = E_F \uplus E_G$, $ctrl = ctrl_F \uplus ctrl_G$, and its link map $link$ is defined as follows: If $q \in X \uplus P_F \uplus P_G$ is a point of $G \circ F$ then

  $$link(q) \stackrel{\text{def}}{=} \begin{cases} link_F(q) & \text{if } q \in X \uplus P_F \text{ and } link_F(q) \in E_F \\ link_G(y) & \text{if } q \in X \uplus P_F \text{ and } link_F(q) = y \in Y \\ link_G(q) & \text{if } q \in P_G. \end{cases}$$

  The identity link graph at $X$ is $\mathsf{id}_X \stackrel{\text{def}}{=} (\emptyset, \emptyset, \emptyset, \mathsf{Id}_X) : X \to X$.

- If $F : I \to J$ and $G : J \to K$ are two bigraphs with $|F| \# |G|$, their composite is

  $$G \circ F \stackrel{\text{def}}{=} \langle G^P \circ F^P, G^L \circ F^L \rangle : I \to K$$

  and the identity bigraph at $I = \langle m, X \rangle$ is $\langle \mathsf{id}_m, \mathsf{id}_X \rangle$.

We shall often omit the composition operator and simply write $GF$ for $G \circ F$. □

Bigraphs also have a notion of partial tensor product, called *juxtaposition*, which is defined for *disjoint graphs*:

**Definition 7.2.7** (disjoint graphical structure (after [29, Def. 2.6]))**.** Two place graphs $F_i$ $(i = 0, 1)$ are *disjoint* if $|F_0| \# |F_1|$. Two link graphs $F_i : X_i \to Y_i$ are *disjoint* if $X_0 \# X_1$, $Y_0 \# Y_1$ and $|F_0| \# |F_1|$. Two bigraphs $F_i$ are *disjoint* if $F_0^P \# F_1^P$ and $F_0^L \# F_1^L$.

In each of the three cases we write $F_0 \# F_1$. □

**Definition 7.2.8** (juxtaposition and units (after [29, Def. 2.7]))**.** We define juxtaposition for place graphs and link graphs separately, and then combine them in order to juxtapose bigraphs. In each case we indicate the obvious unit for juxtaposition.

- For place graphs, the juxtaposition of two interfaces $m_i$ $(i = 0, 1)$ is $m_0 + m_1$ and the unit is 0. If $F_i = (V_i, \mathit{ctrl}_i, \mathit{prnt}_i) : m_i \to n_i$ are disjoint place graphs $(i = 0, 1)$, their juxtaposition $F_0 \otimes F_1 : m_0 + m_1 \to n_0 + n_1$ is given by

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, \mathit{ctrl}_0 \uplus \mathit{ctrl}_1, \mathit{prnt}_0 \uplus \mathit{prnt}'_1),$$

where $\mathit{prnt}'_1(m_0 + i) = n_0 + j$ whenever $\mathit{prnt}_1(i) = j$.

- For link graphs, the juxtaposition of two disjoint link graph interfaces is $X_0 \uplus X_1$ and the unit is $\emptyset$. If $F_i = (V_i, E_i, \mathit{ctrl}_i, \mathit{link}_i) : X_i \to Y_i$ are disjoint link graphs $(i = 0, 1)$, their juxtaposition $F_0 \otimes F_1 : X_0 \uplus X_1 \to Y_0 \uplus Y_1$ is given by

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, \mathit{ctrl}_0 \uplus \mathit{ctrl}_1, \mathit{link}_0 \uplus \mathit{link}_1).$$

- For bigraphs, the juxtaposition of two disjoint interfaces $I_i = \langle m_i, X_i \rangle$ $(i = 0, 1)$ is $\langle m_0 + m_1, X_0 \uplus X_1 \rangle$ and the unit is $\epsilon = \langle 0, \emptyset \rangle$. If $F_i =: I_i \to J_i$ are disjoint bigraphs $(i = 0, 1)$, their juxtaposition $F_0 \otimes F_1 : I_0 \otimes I_1 \to J_0 \otimes J_1$ is given by

$$F_0 \otimes F_1 \stackrel{\text{def}}{=} \langle F_0^P \otimes F_1^P, F_o^L \otimes F_1^L \rangle.$$

□

**Notations and terminology** An interface $\langle n, X \rangle$ is sometimes written as $n$ if $X = \emptyset$ or as $X$ if $n = 0$; hence $\epsilon$, 0 and $\emptyset$ all denote the same interface.

We shall denote bigraphs known to be ground using small letters and we shall often omit their inner face $\epsilon$, e.g., $g : I$.

We call bigraphs with zero width *linkings* (sometimes *wirings*) and we use $\lambda$ and $\omega$ to denote them. We shall often write linkings simply as their link map, and, as instance of this convention, the empty bigraph can be denoted by $\emptyset$. We call linkings with no edges *substitutions*, denoted by $\sigma$ and $\tau$. Discrete substitutions are called *renamings*, denoted by $\alpha$ and $\beta$. Ground substitutions are called (name) *introductions* and we denote them by their outer face $X$, or just $x$ if $X$ is the singleton set $\{x\}$. A linking with empty outer and inner faces and a single edge $e$ is denoted by $/_e$. Linkings with empty outer face, a single inner name $x$, and a single edge $e$ are called *closures*, denoted $/_e x$; the tensor product of multiple closures $/_{e_1} x_1 \otimes \cdots \otimes /_{e_n} x_n$ is sometimes written as $/_{[e_1,\ldots,e_n]}\{x_1, \ldots, x_n\}$. We sometimes omit edge when their identity is irrelevant.

We often omit identities in compositions when there is no ambiguity, and e.g., write $\sigma \circ G$ for $(\sigma \otimes \mathsf{id}_m) \circ G$. Also, we sometimes want to apply a linking $\lambda : X \uplus Y \to Z$ to a graph $g : I \to \langle m, X \rangle$ with fewer outer names than are in the inner face of the linking; in this case we write $\lambda \circ G$ to mean $(\lambda \otimes \mathsf{id}_m) \circ (G \otimes Y)$.

Bigraphs with no nodes or links are called *placings*. We shall often write placings simply as their parent map. Placings where the parent map is a bijection are called *permutations*, denoted $\pi$.

**Subtrees, Subforests, and their Contexts** We shall sometimes need to extract subgraphs from bigraphs:

**Definition 7.2.9** (subtree, subforest)**.** Given a bigraph $H : \langle n, X \rangle \to \langle m, Y \rangle$ and a node or site $c \in V_H \uplus n$. Then the *subtree* rooted at $c$ is the set of nodes and sites defined by

$$H \lfloor^c \stackrel{\text{def}}{=} \{c' \mid c' \in k_H \uplus V_H \ \wedge\ \exists i \geq 0 : \mathit{prnt}_H^i(c') = c\}.$$

For a set of nodes or sites $C \subseteq V_H \uplus n$ we define the *subforest* as the union of the subtrees:

$$H \lfloor^C = \bigcup_{c \in C} H \lfloor^c .$$

□

We shall also need the dual, i.e., the context of subgraphs:

**Definition 7.2.10** (context graph)**.** Given a bigraph $H : \langle n, X\rangle \to \langle m, Y\rangle$ and a node or root $p \in V_H \uplus m$. Then the *context graph* at $p$ is the set of nodes, roots, and sites defined by

$$H\restriction_p \stackrel{\text{def}}{=} (V_H \uplus n \uplus m) \setminus H\downharpoonleft^{prnt^{-1}(p)} .$$

For a set of nodes or roots $P \subseteq V_H \uplus m$ we define the *context graph* as the intersection of the individual context graphs:

$$H\restriction_P = \bigcap_{p\in P} H\restriction_p .$$

□

Subforests and context graphs possess a number of properties:

**Proposition 7.2.11** (subforest and context graph)**.** *Given a bigraph $H : \langle n, X\rangle \to \langle m, Y\rangle$, a set of nodes or sites $c \in C \subseteq V_H \uplus n$, and a set of nodes or roots $p \in P \subseteq V_H \uplus m$. Then we have the following properties:*

1. $H\downharpoonleft^C \subseteq n \uplus V_H$,
2. $c \in H\downharpoonleft^c$,
3. $C \subseteq H\downharpoonleft^C$,
4. $\forall c' \in H\downharpoonleft^c$: $\exists i \geq 0 : prnt^i_H(c') = c$,
5. $H\restriction_P \subseteq n \uplus V_H \uplus m$,
6. $H\restriction_P = (V_H \uplus n \uplus m) \setminus \{c \mid c \in k \uplus V_H \ \wedge\ \exists i > 0 : prnt^i_H(c) \in P\}$
7. $p \in H\restriction_P$, *and*
8. $P \subseteq H\restriction_P$ *if* $\forall v, p \in P : \forall i > 0 : prnt^i_H(v) \neq p$.

*Proof.* The first 5 properties are immediate from the definitions. We prove the remaining three:

6: Expanding the definitions we get:

$$\begin{aligned}
H\restriction_p &= (V_H \uplus n \uplus m) \setminus H\downharpoonleft^{prnt^{-1}(p)} \\
&= (V_H \uplus n \uplus m) \setminus \bigcup_{c\in prnt^{-1}(p)} H\downharpoonleft^c \\
&= (V_H \uplus n \uplus m) \setminus \bigcup_{c\in prnt^{-1}(p)} \{c' \mid c' \in n \uplus V_H \ \wedge\ \exists i \geq 0 : prnt^i_H(c') = c\} \\
&= (V_H \uplus n \uplus m) \setminus \{c' \mid c' \in n \uplus V_H \ \wedge\ \exists i \geq 0 : prnt^i_H(c') \in prnt^{-1}(p)\} \\
&= (V_H \uplus n \uplus m) \setminus \{c' \mid c' \in n \uplus V_H \ \wedge\ \exists i > 0 : prnt^i_H(c') = p\}.
\end{aligned}$$

7: From (6) we can deduce $p \in H\restriction_p$ iff $p$ is a root or if it is a node satisfying $\forall i > 0 : prnt^i_H(p) \neq p$, which is satisfied since $prnt_H$ is acyclic.

8: We must show $\forall p, p' \in P : p' \in H\restriction_p$, assuming $\forall v, p \in P : \forall i > 0 : prnt^i_H(v) \neq p$. From the proof of (7) we have $p' \in H\restriction_p$ iff $p'$ is a root or if it is a node satisfying $\forall i > 0 : prnt^i_H(p') \neq p$ which follows from the assumption. □

**Derived operations** When composing and juxtapositioning bigraphs we shall often want to fuse links from the two graphs if they have the same name, and we therefore introduce two derived operations, *parallel product* $\|$ and *nesting* (sometimes *dotting*) $.$:

**Definition 7.2.12** (parallel product (after [29, Def. 3.11]))**.** The *parallel product* $\|$ is given on interfaces by

$$\langle m, X\rangle \parallel \langle n, Y\rangle \stackrel{\text{def}}{=} \langle m+n, X \cup Y\rangle.$$

Now let $G_i : I_i \to J_i$ $(i = 0, 1)$ be two bigraphs with disjoint supports. Denote the link map of $G_i$ by $link_i$ $(i = 0, 1)$, and assume further that $link_0 \cup link_1$ is a function. Then the parallel product

$$G_0 \parallel G_1 : I_0 \parallel I_1 \to J_0 \parallel J_1$$

is defined just as tensor product, except that its link map allows name-sharing. □

**Proposition 7.2.13** (parallel product (after [24, Prop. 9.14]))**.** *Let $G_0 \parallel G_1$ be defined. Then*

$$G_0 \parallel G_1 = \sigma(G_0 \otimes \tau G_1),$$

*where the substitutions $\sigma$ and $\tau$ are defined as follows: If $z_i$ $(i \in n)$ are the names shared between $G_0$ and $G_1$, and $w_i$ are fresh names in bijection with the $z_i$, then $\tau(z_i) = w_i$ and $\sigma(w_i) = \sigma(z_i) = z_i$ $(i \in n)$.*

**Definition 7.2.14** (nesting (after [29, Def. 3.13]))**.** Let $F : I \to \langle m, X\rangle$ and $G : m \to \langle n, Y\rangle$ be bigraphs. Define the *nesting* $G.F : I \to \langle n, X \cup Y\rangle$ by:

$$G.F \stackrel{\text{def}}{=} (\mathsf{id}_X \parallel G) \circ F.$$

□

#### S-categories and spm-categories

Large parts of the theory of bigraphs is formulated at the more general level of spm categories and s-categories, the definitions of which we shall briefly recall here. We shall assume that the reader is familiar with the basic definitions of category theory. We shall use upper case bold letters to denote categories (e.g., $\mathbf{C}$) and we presuppose an infinite repository of *support elements* $\mathcal{S}$.

**Definition 7.2.15** (partial monoidal category (after [29, Def. 2.10]))**.** A category is said to be *partial monoidal* when it has a partial *tensor product* $\otimes$ both on objects and on arrows satisfying the following conditions.

On objects, $I \otimes J$ and $J \otimes I$ are either both defined or both undefined. The same holds for $I \otimes (J \otimes K)$ and $(I \otimes J) \otimes K$; moreover, they are equal when defined. There is a *unit* object $\epsilon$, often called the *origin*, for which $\epsilon \otimes I = I \otimes \epsilon = I$ for all $I$.

On arrows, the tensor product of $f : I_0 \to I_1$ and $g : J_0 \to J_1$ is defined iff $I_0 \otimes J_0$ and $I_1 \otimes J_1$ are both defined. The following must hold when both sides are defined:

(M1) $f \otimes (g \otimes h) = (f \otimes g) \otimes h$

(M2) $\mathsf{id}_\epsilon \otimes f = f \otimes \mathsf{id}_\epsilon = f$

(M3) $(f_1 \otimes g_1) \circ (f_0 \otimes g_0) = (f_1 \circ f_0) \otimes (g_1 \circ g_0)$.

A functor of partial monoidal categories preserves unit and tensor product. □

**Definition 7.2.16** (spm category (after [29, Def. 2.11]))**.** A partial monoidal category is *symmetric* (spm) if, whenever $I \otimes J$ is defined, there is an arrow $\gamma_{I,J} : I \otimes J \to J \otimes I$ called a *symmetry*, satisfying the following equations when the compositions and products are defined:

(S1) $\gamma_{I,\epsilon} = \mathsf{id}_I$

(S2) $\gamma_{J,I} \circ \gamma_{I,J} = \mathsf{id}_{I \otimes J}$

(S3) $\gamma_{I_1,J_1} \circ (f \otimes g) = (g \otimes f) \circ \gamma_{I_0,J_0}$ (for $f : I_0 \to I_1, g : J_0 \to J_1$)

(S4) $\gamma_{I \otimes J,K} = (\gamma_{I,K} \otimes \mathsf{id}_J) \circ (\mathsf{id}_I \otimes \gamma_{J,K})$.

A functor between spm categories preserves unit, product and symmetries. □

**Definition 7.2.17** (precategory (after [29, Def. 2.12])). A *precategory* $`\mathbf{C}$ is like a category except that composition of $f$ and $g$ may be undefined even when $\mathsf{cod}(f) = \mathsf{dom}(g)$. We use a tag, as in $`\mathbf{C}$, to distinguish precategories. Composition satisfies the following conditions (the first being weaker than for a category):

(C1) if $g \circ f$ is defined then $\mathsf{cod}(f) = \mathsf{dom}(g)$

(C2) $h \circ (g \circ f) = (h \circ g) \circ f$ when either is defined

(C3) $\mathsf{id} \circ f = f$ and $f = f \circ \mathsf{id}$.

We understand (C3) to imply that composition of an arrow $f$ with the identities on its domain and codomain is always defined. □

**Definition 7.2.18** (s-category (after [29, Defs. 2.13 & A.1])). An s-category $`\mathbf{C}$ is a precategory in which each arrow $f$ is assigned a finite *support* $|f| \subset \mathcal{S}$. Further, $`\mathbf{C}$ possesses a partial tensor product, unit and symmetries, as in an spm category. The identities $\mathsf{id}_I$ and symmetries $\gamma_{I,J}$ are assigned empty support. In addition:

(i) For $f : I \to J$ and $g : J' \to K$, the composition $g \circ f$ is defined iff $J = J'$ and $|f| \# |g|$; then $|g \circ f| = |f| \uplus |g|$.

(ii) For $f : I_0 \to I_1$ and $g : J_0 \to J_1$, the tensor product $f \otimes g$ is defined iff $I_i \otimes J_i$ is defined $(i = 0, 1)$ and $|f| \# |g|$; then $|f \otimes g| = |f| \uplus |g|$.

The equations (M1)–(M3) and (S1)–(S4) from Definitions 7.2.15 and 7.2.16 are required to hold when both sides are defined.

For any arrow $f : I \to J$ in an s-category $`\mathbf{C}$ and any partial injective map $\rho : \mathcal{S} \rightharpoonup \mathcal{S}$ whose domain includes $|f|$, there is an arrow $\rho \bullet f : I \to J$ called a *support translation* of $f$. Support translations satisfy the following equations when both sides are defined:

(T1) $\rho \bullet \mathsf{id}_I = \mathsf{id}_I$

(T2) $\rho \bullet (f \circ g) = \rho \bullet f \circ \rho \bullet g$

(T3) $\mathsf{Id}_{|f|} \bullet f = f$

(T4) $(\rho' \circ \rho) \bullet f = \rho' \bullet (\rho \bullet f)$

(T3) $\rho \bullet f = (\rho \restriction_{|f|}) \bullet f$

(T4) $|\rho \bullet f| = \rho(|f|)$

(T5) $\rho \bullet (f \otimes g) = \rho \bullet f \otimes \rho \bullet g$.

Two arrows $f$ and $g$ are *support-equivalent*, written $f \simeq g$, if $\rho \bullet f = g$ for some support translation $\rho$. The *support automorphisms* of an arrow $f$ are the non-identity support translations $\rho : |f| \to |f|$ such that $\rho \bullet f = f$.

A functor between s-categories preserves tensor product, unit, symmetries and support equivalence. □

We shall later need a few results about s-categories:

**Lemma 7.2.19.** *Let $a \simeq c \circ r$, $c \simeq d$, and $r \simeq s$ with $|d| \# |s|$ in some s-category $`\mathbf{C}$. Then $a \simeq d \circ s$.*

*Proof.* By the definition of support equivalence, we have support bijections $\rho : |c \circ r| \to |a|$, $\rho' : |d| \to |c|$ and $\rho'' : |s| \to |r|$ such that $a = \rho \bullet (c \circ r)$, $c = \rho' \bullet d$ and $r = \rho'' \bullet s$. Since $c \circ r$ is defined we have and $|c| \# |r|$. Thus $\rho' \uplus \rho''$ is well-defined and we get the following equivalences:

$$\begin{aligned} a &= \rho \bullet (c \circ r) \\ &= \rho \bullet ((\rho' \bullet d) \circ (\rho'' \bullet s)) \\ &= \rho \bullet (((\rho' \uplus \rho'') \bullet d) \circ ((\rho' \uplus \rho'') \bullet s)) \\ &= (\rho \circ (\rho' \uplus \rho'')) \bullet (d \circ s) \end{aligned}$$

and thus $a \simeq d \circ s$ (using the support translation equalities of Def. 7.2.18). ☐

**Lemma 7.2.20.** *Let $a \simeq c \circ r$ and $r \simeq s$ in some s-category $\grave{\mathbf{C}}$. Then $a \simeq c' \circ s$ for some $c'$ with $c \simeq c'$.*

*Proof.* Choose some bijection $\rho'' : |c| \to \mathcal{S} \setminus |s|$ (where $\mathcal{S}$ is the infinite support repository of $\grave{\mathbf{C}}$) and let $c' \stackrel{\text{def}}{=} \rho'' \bullet c$. By construction we have $c \simeq c'$ and $|c'| \# |s|$, so by Lemma 7.2.19 $a \simeq c' \circ s$ as required. ☐

Obviously, spm categories can be seen as s-categories where arrows have empty support. Conversely, we can obtain spm categories from s-categories as follows:

**Definition 7.2.21** (support quotient (after [29, Def. 2.14]))**.** For any s-category $\grave{\mathbf{C}}$, its *support quotient*

$$\mathbf{C} \stackrel{\text{def}}{=} \grave{\mathbf{C}} / \simeq$$

is the spm category whose objects are those of $\grave{\mathbf{C}}$, and whose arrows $[f] : I \to J$ are support-equivalence classes of the homset $\grave{\mathbf{C}}(I \to J)$. The composition of $[f] : I \to J$ with $[g] : J \to K$ is defined as $[g] \circ [f] \stackrel{\text{def}}{=} [g' \circ f']$, where $f' \in [f]$ and $g' \in [g]$ are chosen with disjoint supports.

The tensor product is defined analogously. The identities and symmetries of $\mathbf{C}$ are singleton equivalence classes since they have empty support. ☐

**Notations and terminology** In any category $\mathbf{C}$ a pair of arrows $\vec{f} : I \to \vec{J}$ is a *span*. Dually, a pair of arrows $\vec{f} : \vec{I} \to J$ is a *cospan*.

If $\vec{f} : I \to \vec{J}$ is a span and $\vec{g} : \vec{J} \to K$ is a cospan satisfying $g_0 \circ f_0 = g_1 \circ f_1$, then $\vec{g}$ is a *bound* for $\vec{f}$ and, dually, $\vec{f}$ is a *cobound* for $\vec{g}$.

### Bigraphical Categories

Concrete bigraphs and their constituents form s-categories, and by quotienting these by (essentially) support equivalence we get spm categories of abstract bigraphs.

**Definition 7.2.22** (graphical s-categories (after [29, Def. 2.17]))**.** A basic signature $\mathcal{K}$ was defined in Def. 7.2.1. Concrete place graphs, link graphs and bigraphs over an arbitrary signature were defined in Defs. 7.2.2, 7.2.3 and 7.2.4. We now cast each of these kinds of graph as arrows in an s-category, denoted respectively by $\grave{\text{P}}\text{G}(\mathcal{K})$, $\grave{\text{L}}\text{G}(\mathcal{K})$ and $\grave{\text{B}}\text{G}(\mathcal{K})$.

The objects in these three s-categories are called *interfaces*, or *faces*. For place graphs they are natural numbers, for link graphs they are finite name-sets, and for bigraphs they are pairs of a natural number $m$ and a finite name-set.

*Support* for the three kinds of graph was defined in Def. 7.2.5, with support elements $\mathcal{V} \uplus \mathcal{E}$. *Composition* and *identities* were set out in Def. 7.2.6, and *juxtaposition* and *units* in Def. 7.2.8, determining *tensor product*.

To complete our definition it remains to define *symmetries* $\gamma_{I,J}$ as follows:

$$
\begin{array}{llll}
\text{in `Pg :} & \gamma_{m,n} \stackrel{\text{def}}{=} (\emptyset, \emptyset, prnt), & \text{where} \quad prnt(i) = n + i & (i \in m) \\
& & \text{and} \quad prnt(m+j) = j & (j \in n) \\
\text{in `Lg :} & \gamma_{X,Y} \stackrel{\text{def}}{=} \mathsf{id}_{X \uplus Y} & & \\
\text{in `Bg :} & \gamma_{\langle m,X \rangle, \langle n,Y \rangle} \stackrel{\text{def}}{=} \langle \gamma_{m,n}, \gamma_{X,Y} \rangle. & &
\end{array}
$$

□

**Definition 7.2.23** (lean, lean-support quotient, abstract bigraphs (after [29, Def. 2.19]))**.** A bigraph is *lean* if it has no idle edges. Two bigraphs $F$ and $G$ are *lean-support equivalent*, written $F \mathrel{\hat{\simeq}} G$, if they are support-equivalent ignoring their idle edges. It is easily seen that both composition and tensor product preserve this equivalence.

For the bigraphical s-category `Bg($\mathcal{K}$), its *lean-support quotient*

$$\text{Bg}(\mathcal{K}) \stackrel{\text{def}}{=} \text{`Bg}(\mathcal{K}) / \mathrel{\hat{\simeq}}$$

is the spm category whose objects are those of `Bg($\mathcal{K}$) and whose arrows $[\![G]\!] : I \to J$, called *abstract bigraphs*, are lean-support equivalence classes of the homset $(I \to J)$ in `Bg($\mathcal{K}$). Composition, tensor product, identities and symmetries for the lean-support quotient are defined just as for support quotient in Def. 7.2.21.

The spm categories Pg($\mathcal{K}$) of abstract place graphs and Lg($\mathcal{K}$) of abstract link graphs are constructed similarly.

We shall sometimes use hat to denote that an arrow is abstract, e.g., $\hat{G}$. We shall say that $G$ is a *concretion* of $[\![G]\!]$. □

We shall later need the following result regarding lean support equivalence:

**Lemma 7.2.24.** *Let $F, G, C, D$ be concrete bigraphs with $G \mathrel{\hat{\simeq}} F$ and $F = C \circ D$. Then there exists concrete bigraphs $C', D'$ with $C' \mathrel{\hat{\simeq}} C$ and $D' \mathrel{\hat{\simeq}} D$ such that $G = C' \circ D'$.*

*Proof.* Let $\rho$ be a witness of the support equivalence part of $G \mathrel{\hat{\simeq}} F$. Then $\rho \centerdot G$ and $F = C \circ D$ differ only in their idle edges. Construct $C''$ and $D''$ by removing the idle edges of $C$ and $D$ that are not in $\rho \centerdot G$, and add the idle edges of $|\rho \centerdot G| \setminus |F|$ to either one. Then clearly $C'' \mathrel{\hat{\simeq}} C$, $D'' \mathrel{\hat{\simeq}} D$ and $\rho \centerdot G = C'' \circ D''$. Since support translations are bijections, we have $G = \rho^{-1} \centerdot C'' \circ \rho^{-1} \centerdot D''$, $C' \stackrel{\text{def}}{=} \rho^{-1} \centerdot C'' \simeq C'' \mathrel{\hat{\simeq}} C$, and $D' \stackrel{\text{def}}{=} \rho^{-1} \centerdot D'' \simeq D'' \mathrel{\hat{\simeq}} D$ as required. □

**Corollary 7.2.25** (lean support equivalence vs support equivalence)**.** *Let $F, G, C, D$ be concrete bigraphs with $G \mathrel{\hat{\simeq}} F$ and $F \simeq C \circ D$. Then there exists concrete bigraphs $C', D'$ with $C' \mathrel{\hat{\simeq}} C$ and $D' \mathrel{\hat{\simeq}} D$ such that $G \simeq C' \circ D'$.*

*Furthermore, if $C$ is lean, a lean $C'$ exists, i.e., $C \simeq C'$. The same holds for $D$ and $D'$. Both $C'$ and $D'$ can be lean iff $G$ is lean.*

*Proof.* Let $\rho'$ be the witness of $F \simeq C \circ D$ and use the construction from the proof of Lemma 7.2.24 for $G$ and $\rho' \centerdot F = C \circ D$ (obviously $G \mathrel{\hat{\simeq}} \rho \centerdot F$), putting all the idle edges of $|\rho \centerdot G| \setminus |\rho' \centerdot F|$ in either $C''$ or $D''$. □

The notations, terminology and derived operations pertaining to concrete bigraphs carry over to abstract bigraphs.

### Reactive Systems

Bigraph dynamics are defined as an instance of the more general *basic reactive systems*, where a set of ground reaction rules generates a reaction relation by closing the set under all contexts and support equivalence:[1]

[1]We avoid the added complexity of the so-called *passive* contexts, contexts that disallow reaction, from this presentation as our work can be straightforwardly extended to include such contexts. Omitting passive contexts allows us to ignore the refinement of reactive systems to *wide reactive systems* [29, Def. 7.2], which are only introduced to handle passive contexts when deriving labeled transition systems (LTSs).

**Definition 7.2.26** (basic reactive system (BaRS) (after [29, Sec. 7.1]))**.** A *basic reactive system*, written $\grave{\mathbf{C}}(\grave{\mathcal{R}})$, consists of an s-category $\grave{\mathbf{C}}$ equipped with a set $\grave{\mathcal{R}}$ of *reaction rules.* An arrow $a : \epsilon \to I$ in $\grave{\mathbf{C}}$ with domain $\epsilon$ is a *ground arrow* or *agent*, often written $a : I$.

Each reaction rule $\mathsf{R}$ consists of a pair $(r : I, r' : I)$ of ground arrows, a *redex* and a *reactum.* The set $\grave{\mathcal{R}}$ must be closed under support translation, i.e., if $(r, r')$ is a rule then so is $(s, s')$ whenever $r \simeq s$ and $r' \simeq s'$.

The *reaction relation* $\longrightarrow$ over agents is the smallest such that $a \longrightarrow a'$ whenever $a \simeq c \circ r$ and $a' \simeq c \circ r'$ for some reaction rule $(r, r')$ and context $c$ for $r$ and $r'$.

□

When the underlying s-category disregards support we say that the BaRS is abstract:

**Definition 7.2.27** (abstract BaRS (after [29, Def. 7.3]))**.** A BaRS is *abstract* if its underlying s-category is an spm category. □

We saw in section 7.2.2 how we can construct spm categories from s-categories by quotienting by support equivalence. Similarly, we may construct abstract BaRSs from concrete ones by quotienting by support equivalence. However, as we saw in section 7.2.2, abstract bigraphs are constructed from concrete bigraphs by quotienting by a more coarse-grained equivalence: lean-support equivalence. We therefore generalize the constructions of spm categories and abstract BaRSs from concrete ones to more general equivalences, so-called *abstractions*:

**Definition 7.2.28** (structural congruence (after [29, Def. 7.4]))**.** An equivalence relation $\equiv$ on each homset of an s-category $\grave{\mathbf{C}}$ is a *structural congruence* if it is preserved by composition and tensor product. It is called an *abstraction* if it includes support equivalence. We denote the $\equiv$-equivalence class of $f$ by $[\![f]\!]$.

In a BaRS $\grave{\mathbf{C}}_{(\mathsf{r})}(\grave{\mathcal{R}})$ an abstraction is *dynamic* if in addition it respects reaction, i.e., if $f \longrightarrow f'$ and $g \equiv f$ then $g \longrightarrow g'$ for some $g' \equiv f'$. □

Clearly, support equivalence is a dynamic abstraction on any BaRS:

**Proposition 7.2.29** (support equivalence is a dynamic abstraction)**.** *Support equivalence $\simeq$ is a dynamic abstraction on a BaRS $\grave{\mathbf{C}}(\grave{\mathcal{R}})$.*

We can now state the more general abstraction constructions and show that they indeed yield spm categories and abstract BaRSs:

**Definition 7.2.30** (quotient s-category (after [29, Def. 7.5]))**.** Let $\grave{\mathbf{C}}$ be an s-category, and let $\equiv$ be an abstraction on $\grave{\mathbf{C}}$. Then

$$\mathbf{C} \stackrel{\text{def}}{=} \grave{\mathbf{C}}/\equiv$$

is the spm category whose objects are those of $\grave{\mathbf{C}}$, and whose arrows $[\![f]\!] : I \to J$ are $\equiv$-equivalence classes of the homset $I \to J$ in $\grave{\mathbf{C}}$. The composition of $[\![f]\!] : I \to J$ with $[\![g]\!] : J \to K$ is defined as $[\![g]\!] \circ [\![f]\!] \stackrel{\text{def}}{=} [\![f' \circ g']\!]$, where $f' \in [\![f]\!]$ and $g' \in [\![g]\!]$ are chosen with disjoint supports. The tensor product is defined analogously. The identities and symmetries in $\mathbf{C}$ are necessarily the equivalence classes of their $\grave{\mathbf{C}}$ counterparts. □

**Lemma 7.2.31** (quotient s-category)**.** *Let $\grave{\mathbf{C}}$ be an s-category, and let $\equiv$ be an abstraction on $\grave{\mathbf{C}}$. Then the quotient $\mathbf{C} = \grave{\mathbf{C}}/\equiv$ is an spm category. Its construction defines a functor of s-categories*

$$[\![\cdot]\!] : \grave{\mathbf{C}} \to \mathbf{C}.$$

*Proof.* $\mathbf{C}$ is an spm category, since it inherits the tensor product, unit, and symmetries of the s-category $\grave{\mathbf{C}}$, and the construction eliminates the partiality of composition and tensor product. $[\![\cdot]\!]$ is a functor by construction and it is between s-categories because any spm category is an s-category with empty supports. □

**Definition 7.2.32** (quotient BaRS (after [29, Def. 7.6]))**.** Let $`\mathbf{C}(`\mathcal{R})$ be a BaRS, and $\equiv$ a dynamic abstraction on $`\mathbf{C}$. Then define $\mathbf{C}(\mathcal{R})$, the quotient of $`\mathbf{C}(`\mathcal{R})$ by $\equiv$, as follows:

- $\mathbf{C} = `\mathbf{C}/\equiv$, and
- $\mathcal{R} = \{([\![r]\!], [\![r']\!]) \mid (r, r') \in `\mathcal{R}\}$.

$\square$

**Theorem 7.2.33** (abstract BaRS (after [29, Thm. 7.7]))**.** *The construction of Def. 7.2.30 and Def. 7.2.32, applied to a concrete BaRS $`\mathbf{C}(`\mathcal{R})$, yields an abstract BaRS $\mathbf{C}(\mathcal{R})$, whose underlying spm category $\mathbf{C}$ is the codomain of a functor of s-categories*

$$[\![\cdot]\!] : `\mathbf{C} \to \mathbf{C}.$$

*Moreover the construction preserves the reaction relation, in the following sense:*

1. *if $f \rightarrowtriangle f'$ in $`\mathbf{C}(`\mathcal{R})$ then $[\![f]\!] \rightarrowtriangle [\![f']\!]$ in $\mathbf{C}(\mathcal{R})$*
2. *if $[\![f]\!] \rightarrowtriangle g'$ in $\mathbf{C}(\mathcal{R})$ then $f \rightarrowtriangle f'$ in $`\mathbf{C}(`\mathcal{R})$ for some $f'$ with $[\![f']\!] = g'$.*

*Proof.* The first part follows immediately from Lemma 7.2.31 and Def. 7.2.27.

1: We have $f \bumpeq c \circ r$ and $f' \bumpeq c \circ r'$ for some reaction rule $(r, r')$ and context $c$ for $r$ and $r'$. We must show $[\![f]\!] \rightarrowtriangle [\![f']\!]$, i.e., $[\![f]\!] \bumpeq [\![c']\!] \circ [\![s]\!]$ and $[\![f']\!] \bumpeq [\![c']\!] \circ [\![s']\!]$ for some reaction rule $(s, s')$ and context $[\![c']\!]$ for $[\![s]\!]$ and $[\![s']\!]$.

Letting $c' \stackrel{\text{def}}{=} c$, $s \stackrel{\text{def}}{=} r$, $s' \stackrel{\text{def}}{=} r'$ and exploiting that $\equiv$ includes support equivalence and that $[\![\cdot]\!]$ is a functor, we get $([\![s]\!], [\![s']\!]) = ([\![r]\!], [\![r']\!]) \in \mathcal{R}$, $[\![f]\!] = [\![c \circ r]\!] = [\![c' \circ s]\!] = [\![c']\!] \circ [\![s]\!]$, and $[\![f']\!] = [\![c \circ r']\!] = [\![c' \circ s']\!] = [\![c']\!] \circ [\![s']\!]$ as required.

2: We have $[\![f]\!] \bumpeq [\![c]\!] \circ [\![r]\!]$ and $g' \bumpeq [\![c]\!] \circ [\![r']\!]$ for some reaction rule $(r, r')$ and context $[\![c]\!]$ for $[\![r]\!]$ and $[\![r']\!]$. We must show $f \rightarrowtriangle f'$ for some $f'$ with $[\![f']\!] = g'$, i.e., $f \bumpeq c' \circ s$ and $f' \bumpeq c' \circ s'$ for some reaction rule $(s, s')$ and context $c'$ for $s$ and $s'$.

Let $s \stackrel{\text{def}}{=} r$, $s' \stackrel{\text{def}}{=} r'$ and choose some context $c' \in [\![c]\!]$ with $c' \# r$ and $c' \# r'$. By definition of composition in $\mathbf{C}$ and since abstraction includes support equivalence, we get $[\![f]\!] = [\![c]\!] \circ [\![r]\!] = [\![c' \circ r]\!]$ and $g' = [\![c]\!] \circ [\![r']\!] = [\![c' \circ r']\!]$. By Def. 7.2.26 we have $f \equiv c' \circ r \rightarrowtriangle c' \circ r'$ and since $\equiv$ is dynamic, there is some $f' \equiv c' \circ r' \in g'$ such that $f \rightarrowtriangle f'$ as required. $\square$

### Bigraphical Reactive Systems

While the rules of basic reactive systems are required to be ground, bigraphical reaction rules are allowed to take parameters. But by viewing such parametric reaction rules as generators of ground reaction rules, we can view bigraphical reactive systems as a sugared variant of basic reactive systems.

Before we define bigraphical parametric reaction rules and bigraphical reactive systems, let us first define what a parameter is and how a parametric reaction rule is allowed to manipulate it through *instantiation*.

First, note that ground bigraphs can be seen as the juxtaposition of a number of discrete primes bound together by some linking:

**Corollary 7.2.34** (ground discrete normal form (DNF) (after [29, Corol. 3.10]))**.** *A ground bigraph $g : \langle n, Z \rangle$ can be expressed uniquely, up to renaming on $Y$, as $g = (\mathsf{id}_n \otimes \lambda) \circ (d_0 \otimes \cdots \otimes d_{n-1})$, where $\lambda : Y \to Z$ is a linking and the $d_i$ are discrete primes.*

We shall regard the individual primes of a ground bigraph as parameters of reaction and shall allow each of them to be copied, discarded, or left unchanged using *instantiation*:

**Definition 7.2.35** (instantiation (after [29, Def. 8.5]))**.** In a bigraphical s-category $\grave{\mathbf{C}} = \grave{\textsc{Bg}}(\mathcal{K})$, let $\eta : n \to m$ be a map of finite ordinals. Define the *instance* function family $\bar{\eta}_{X,S} : \grave{\mathbf{C}}(\epsilon, \langle m, X\rangle) \to \grave{\mathbf{C}}(\epsilon, \langle n, X\rangle)$, indexed by name set $X$ and support set $S$, on agents as follows: Given an agent $g : \langle m, X\rangle$, find its DNF $g = \lambda \circ (d_0 \otimes \cdots \otimes d_{m-1})$ (Corol. 7.2.34). Then

$$\bar{\eta}_{X,S}(g) \stackrel{\text{def}}{=} \lambda \circ (d'_0 \parallel \cdots \parallel d'_{n-1})$$

where $d'_j \bumpeq d_{\eta(j)}$ and $|d'_j| \# S$ for each $j \in n$. The function is defined up to $\bumpeq$.

We shall often omit $X$ and/or $S$ when they are evident from the context. □

We have reformulated Milner's definition to index $\bar{\eta}$ by $X$ and $S$; $X$ was already a somewhat implicit index whereas $S$ is a technical measure that will allow us to ensure that instantiation chooses fresh support with respect to the context in which it will be used.

Note that $\bar{\eta}_{X,S}(g)$ has the same outer names as $g$ and that linking commutes with instantiation:

**Proposition 7.2.36** (linking an instance (after [29, Def. 8.4]))**.** *Linking commutes with instantiation; that is,* $\omega \circ \bar{\eta}_{X,S}(g) \bumpeq \bar{\eta}_{X,S}(\omega \circ g)$.

Let us now define parametric reaction rules for bigraphs and how they generate ground reaction rules:

**Definition 7.2.37** (bigraphical parametric reaction rules (after [29, Def. 8.5]))**.** A *parametric reaction rule* $\mathsf{R}$ for bigraphs is a triple of the form

$$(R : m \to J, R' : m' \to J, \eta)$$

where $R$ is the *parametric redex*, $R'$ the *parametric reactum*, and $\eta : m' \to m$ a map of finite ordinals. The rule generates all ground reaction rules $(r, r')$, where

$$r \bumpeq R.d, \quad r' \bumpeq R'.\bar{\eta}(d)$$

and $d : \langle m, Y\rangle$ is discrete. □

With this definition in mind, it is clear that the following definition of bigraphical reactive systems is an instance of the basic reactive systems defined above:

**Definition 7.2.38** (bigraphical reactive system (BRS) (after [29, Def. 8.6]))**.** A *(concrete) bigraphical reactive system (BRS)* over $\mathcal{K}$ consists of $\grave{\textsc{Bg}}(\mathcal{K})$ equipped with a set $\grave{\mathcal{R}}$ of parametric reaction rules closed under support equivalence; that is, if $R \bumpeq S$ and $R' \bumpeq S'$ and $\grave{\mathcal{R}}$ contains $(R, R', \eta)$, then it also contains $(S, S', \eta)$. We denote the BRS by $\grave{\textsc{Bg}}(\mathcal{K}, \grave{\mathcal{R}})$. □

Comparing the definition of BaRSs with the generation of ground bigraphical reaction rules from parametric ones, it is clear that reactions are generated by occurrences of redexes, what we call *matches*:

**Definition 7.2.39** (match)**.** Given a parametric reaction rule $\mathsf{R} = (R, R', \eta)$, agent $a$, and bigraphs $c$,$d$, we say that $(c, d)$ is a *match* of $R$ in $a$ iff $a \bumpeq c \circ R.d$. Two matches $(c, d)$,$(c', d')$ are regarded as the same if they differ only by a bijection on the outer faces of $d$ and $d'$; otherwise they are *distinct*. We write $match(a, \mathsf{R})$ for the set of distinct matches of $R$ in $a$. □

We may construct abstract BRSs by quotienting by lean-support equivalence $\Bumpeq$ since it is a dynamic abstraction on BRSs:

**Proposition 7.2.40** (lean-support equivalence is a dynamic abstraction)**.** *Lean-support equivalence* $\Bumpeq$ *is a dynamic abstraction on a BRS* $\grave{\textsc{Bg}}(\mathcal{K}, \grave{\mathcal{R}})$.

*Proof.* It is immediate from its definition that lean-support equivalence is an abstraction, so we just need to check that it is dynamic.

Assume bigraphs $a, a', b$ with $a \rightarrowtriangle a'$ and $b \Bumpeq a$. Since $a \rightarrowtriangle a'$ we must have $a \bumpeq c \circ r$ and $a' \bumpeq c \circ r'$ for some rule $(r, r')$ and context $c$. By Corollary 7.2.25, noting that bigraph rules are lean, there are bigraphs $d, s$ with $d \Bumpeq c$ and $s \bumpeq r$ such that $b \bumpeq d \circ s$. Since rules are closed under support translation, there is a rule $(s, s')$ with $s' \bumpeq r'$, and we thus have the reaction $b \bumpeq d \circ s \rightarrowtriangle b'$ where $b' = d \circ s'$. Since lean-support equivalence is preserved by composition we have $b' = d \circ s' \Bumpeq c \circ r' \bumpeq a'$ as required. □

## 7.3 The Simulation Algorithm

We give an overview of the simulation algorithm for the $\kappa$-calculus by Danos et al. [12] recast to stochastic bigraphs. We shall refer to the algorithm in loc. cit. as KaSim. This reformulation of KaSim to stochastic bigraphs is independent of the physical and stochastic underpinnings of the algorithm, so we shall not concern ourselves with the details of these matters; the interested reader may refer to [12, 18, 19].

### 7.3.1 Gillespie's algorithm

KaSim is a generalization of what is known as Gillespie's algorithm, an algorithm for stochastic simulation of coupled chemical reactions [18, 19]. It is based on the idea of assigning probabilities to reaction rules which are proportional to the number of instances of each rule in the current state of the system, and letting the frequency of reaction be proportional to the total number of rule instances.

Recast to bigraphs, the algorithm in overview works as follows: given a set of reaction rules $\mathcal{R}$, with each reaction rule $\mathsf{R}$ assigned a rate constant $\varrho_{\mathsf{R}}$, an agent $a$, and a simulation time $t_{\text{stop}}$, perform the following steps:

**0. Initialization:**

Initialize the simulation state:

| | |
|---|---|
| $t := 0$ | current simulation time, initially 0 |
| $M(a, \mathsf{R}) := \mathit{match}(a, \mathsf{R})$ | set of matches of R's redex in $a$ ($\forall \mathsf{R} \in \mathcal{R}$) |
| $\alpha_{\mathsf{R}} := \lvert M(a, \mathsf{R})\rvert \times \varrho_{\mathsf{R}}$ | activity of R |
| $\alpha := \Sigma_{\mathsf{R} \in \mathcal{R}} \alpha_{\mathsf{R}}$ | system activity |

If $\alpha = 0$ then no reaction is possible and the simulation ends.

**1. Monte Carlo step:**

Sample the following random values:

| | |
|---|---|
| $\mathsf{R} := \mathit{rand}(\mathcal{R}, \lambda \mathsf{R}. \frac{\lvert M(a,\mathsf{R})\rvert \times \varrho_{\mathsf{R}}}{\alpha})$ | rule to be applied |
| $\phi := \mathit{rand}(M(a, \mathsf{R}), \lambda m.1/\lvert M(a, \mathsf{R})\rvert)$ | match to be applied |
| $\delta t := \mathit{rand}(\mathbb{R}^+, \lambda t.\alpha e^{\alpha t})$ | time advance |

**2. Update:**

Update the simulation state:

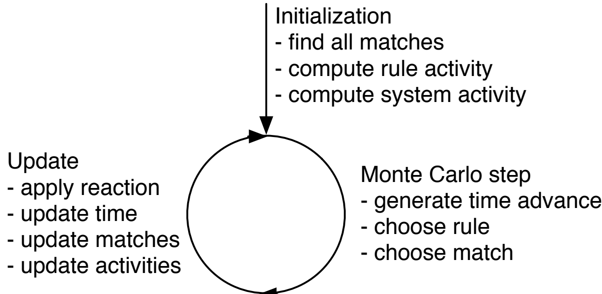



Figure 7.1: The basic simulation loop.

| | |
|---|---|
| $a := a'$, if $a \twoheadrightarrow_{\mathsf{R},\phi} a'$ | perform reaction |
| $t := t + \delta t$ | advance time |
| $M(a, \mathsf{R}) := \mathit{match}(a, \mathsf{R})$ | update sets of matches ($\forall \mathsf{R} \in \mathcal{R}$) |
| $\alpha_{\mathsf{R}} := \lvert M(a, \mathsf{R})\rvert \times \varrho_{\mathsf{R}}$ | update rule activities |
| $\alpha := \Sigma_{\mathsf{R}\in\mathcal{R}} \alpha_{\mathsf{R}}$ | update system activity |

**3. Iterate:**

If $t > t_{\text{stop}}$ or $\alpha = 0$, stop; otherwise repeat from step 1.

Figure 7.1 illustrates the simulation loop.

### 7.3.2 Incremental and Local Updates

It should be clear that the update step as expressed above does not scale: it requires recomputation of all matches at each simulation cycle. Instead, KaSim employs an incremental update phase where (i) matches are only removed from $M(a, \mathsf{R})$ if they are invalidated by the reaction and (ii) matches are only searched for in the parts of the agent that were affected by the reaction. Thus, the update phase actually consists of three steps[2]:

**2a. Negative update:**

Remove matches that will be invalidated by the chosen reaction and decrease activities accordingly.

**2b. Rewrite:**

Rewrite the agent using the chosen rule and match.

**2c. Positive update:**

Find new matches created by the reaction and increment activities accordingly.

These steps presume that we can determine conflict and causality in an efficient manner: we must be able to quickly identify (2a) the reactions that are in conflict with the chosen reaction and (2c) the reactions that it causes. This is achieved in KaSim by (i) assuming that rules are enriched with a notion of *modification* which characterizes how reaction modifies the redex, and (ii) assuming the existence of two relations, called the inhibition and activation maps, that characterize the interplay between rules:

[2]As a technicality, we have swapped steps 2a. and 2b. as it in the case of SBAM leads to a more direct implementation.

**inhibition:** We say that rule $\mathsf{R}_0$ *inhibits* rule $\mathsf{R}_1$, written $\mathsf{R}_0 \# \mathsf{R}_1$, iff there is some agent $a$ and embeddings $\phi_i : R_i \hookrightarrow a$ such that $\mathsf{cod}(\phi_0) \cap \mathsf{cod}(\phi_1)$ contains at least one entity modified by $\mathsf{R}_0$.

**activation:** Rule $\mathsf{R}_0$ *activates* rule $\mathsf{R}_1$, written $\mathsf{R}_0 \prec \mathsf{R}_1$, iff there is some agent $a$ and embeddings $\phi_0 : \Delta_0(R_0) \hookrightarrow a$, $\phi_1 : R_1 \hookrightarrow a$ such that $\mathsf{cod}(\phi_0) \cap \mathsf{cod}(\phi_1)$ contains at least one entity modified by $\mathsf{R}_0$.

Note that these relations are not necessarily symmetric.

Assuming we can construct these relations during initialization, we may express the negative and positive update steps in more detail as follows (recall that $\mathsf{R}$ is the chosen rule, $\phi$ the chosen match, and that step 2b. sets $a := a'$):

**2a. Negative update:**

For each $\mathsf{R}'$ with $\mathsf{R} \# \mathsf{R}'$

(i) remove the embeddings $\phi' : R' \hookrightarrow a$ from $M(a, \mathsf{R}')$ for which some elements of $\mathsf{rng}(\phi) \cap \mathsf{rng}(\phi')$ will be modified by the chosen reaction, and

(ii) decrease the system activity and the activity of the rule by the number of removed embeddings times $\varrho_{\mathsf{R}'}$.

**2c. Positive update:**

For each $\mathsf{R}'$ with $\mathsf{R} \prec \mathsf{R}'$

(i) add new embeddings $\phi' : R' \hookrightarrow a$ to $M(a, \mathsf{R}')$. At least one element of $\mathsf{rng}(\phi')$ must be modified by the reaction in order for $\phi'$ to be new, and

(ii) increase the system activity and the activity of the rule by the number of added embeddings times $\varrho_{\mathsf{R}'}$.

Thus, using this approach we avoid considering rules that are known to never generate reactions that are causally related to those of the chosen rule. But even in the worst case, $\prec = \# = \mathcal{R} \times \mathcal{R}$, this approach is an improvement since we have restricted the part of the agent that we need to consider.

Let us consider steps 2a(i) and 2c(i) in a bit more detail:

**2a(i):** Though we have restricted the set of embeddings we need to consider, it is still unclear how to efficiently identify the affected embeddings. The approach in KaSim is actually to not use the inhibition relation, but instead maintain a so-called *lift* map $l : |a| \to M(a, \cdot)$ from entities in the agent to the embeddings that have those entities in their co-domain. While less space-efficient, it enables us to quickly remove invalidated embeddings.

**2c(i):** The notion of modification that rules are enriched with, allows us to determine which entities in the agent have been modified, so we can perform localized matching as follows: for each modified entity $e \in |a|$ and entity $e' \in |R'|$ such that $[e' \mapsto e]$ is a partial embedding, attempt to extend it to a complete embedding of $R'$.

By extending we mean incrementally adding mappings $f \mapsto f'$ of entities $f \in |a| \setminus \mathsf{rng}(\phi')$, $f' \in |R'| \setminus \mathsf{dom}(\phi')$ which are adjacent to elements of $\mathsf{rng}(\phi')$ and $\mathsf{dom}(\phi')$ respectively. We call this *anchored matching.*

In $\kappa$, anchored matching is deterministic and there is at most one complete extension of a partial embedding. This is not generally the case for bigraphs.

Note that anchored matching only yields complete embeddings for redexes consisting of one connected component and the KaSim algorithm is actually slightly more complicated than what we have sketched above, as it handles redexes with more than one connected component. However, the KaSim approach to handling such redexes transfer unaltered to the bigraph version of the algorithm, so in this report we shall simply assume that redexes consist of exactly one connected component.

## 7.4 Stochastic Parametric Reactive Systems

Concrete bigraphs are a means to constructing a tractable behavioral theory for abstract bigraphs: Abstract bigraphs could be defined directly instead of being derived from concrete bigraphs[3]. However, abstract bigraphs have insufficient structure for constructing minimal transition labels which is a key construction in the behavioral theory of BRSs. By defining abstract bigraphs in terms of concrete bigraphs, where such minimal labels can be constructed, one gets the means for obtaining minimal labels for abstract bigraphs.

The dynamic theory of bigraphs have been designed with this construction in mind, which is reflected in the treatment of support : (a) sets of rules are required to be closed under support translation (cf. Def. 7.2.26 and Def. 7.2.38), (b) the construction of the reaction relation closes under support translation (cf. Def. 7.2.26), and (c) the construction of ground reaction rules from parametric ones closes under support translation (cf. Def. 7.2.37). In other words, these constructions are aimed at support equivalence classes of concrete bigraphs, i.e., abstract bigraphs.

While this approach is sufficient for most applications of abstract bigraphs, it is insufficient in the context of stochastic bigraphs, where support provides the means for counting matches: closing under support translation would lead to an infinite number of matches for non-trivial redexes. In their definition of stochastic bigraphs [25], Krivine et al. solve this issue by (a) replacing support equivalence by equality in the definition of a match (cf. Def. 7.2.39)[4], and (b) defining the number of matches in an abstract bigraph as the number of matches in one of its concretions. However, in loc. cit. the definition of a match is not (directly) related to the definition of the reaction relation, resulting in a conceptual gap between the usual reaction semantics and the stochastic reaction semantics. A unified presentation of these two aspects of stochastic BRSs would promote understanding.

Another issue is the infinite set of ground reaction rules that a parametric reaction rule generate. Clearly, we cannot represent these explicitly in an implementation, so we cannot implement BRSs directly as stated in Def. 7.2.26 and Def. 7.2.38. We believe that direct representations in implementations increase trust in correctness, and it would therefore be desirable if we could give a directly representable definition of BRSs with parametric reaction rules.

In this section we tackle both of these issues, by developing a variant of reactive systems, which we call *stochastic parametric reactive systems*, that have none of the above shortcomings while giving rise to the same abstract reaction relation. The idea is to prevent arbitrary support translation during reaction by restricting the use of support translation to the identification of matches of redexes in an agent.

Specifically, we incrementally develop the following kinds of reactive systems:

**representative basic reactive systems (RBaRS):**
Almost as BaRSs but different in two respects:

- the set of rules must not contain support equivalent rules and
- reaction cannot change the support of the context.

[3]Milner's algebra for abstract bigraphs could be one such definition [27].

[4]Krivine et al. use the term *occurrence* and use a slightly different definition, cf. [25, Def. 4.1 and Def. 4.2], but this is an insignificant technicality.

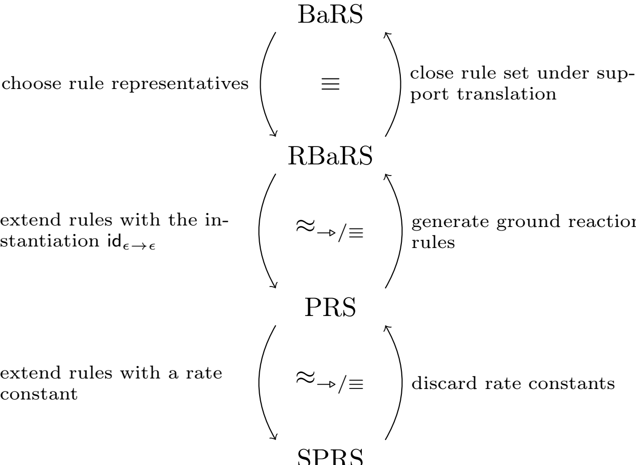



Figure 7.2: The four kinds of reactive systems and how to transform one into another. The topmost two are equivalent modulo dynamic abstraction $\equiv$ (e.g., $\doteq$) while the others have the same reaction relation modulo dynamic abstraction.

**parametric reactive systems (PRS):**
A generalization of RBaRSs where parametric reaction rules are first class citizens. Reaction is refined further by restricting the manipulation of support in parameters.

**stochastic parametric reactive systems (SPRS):**
PRSs equipped with a stochastic semantics, which for bigraphs generalizes the stochastic semantics of Krivine et al. [25]. The reaction relation is refined such that a match determines a reaction.

The concrete reaction relation becomes smaller for each increment, while the abstract reaction relation remains the same. To prove this, we for each of these systems show that its concrete reaction relation is closely related to that of its predecessor, indeed so closely that it becomes immediate that they have the same abstract reactions. Figure 7.2 gives an overview of the relations between the four kinds of reactive systems.

### 7.4.1 Representative Basic Reactive Systems

In this section we show how one may limit Milner's liberal use of support equivalence in the definition of basic reactive systems, while maintaining the same dynamic behavior. We do so in two steps: (1) first we show that we need not require the set of reaction rules to be closed under support translation, and then (2) we reduce the reaction relation to preclude support translation in the context.

Recall from Def. 7.2.26 that in a BaRS

- the set of reaction rules $\grave{\mathcal{R}}$ must be closed under support translation, i.e., if $(r, r')$ is a rule then so is $(s, s')$ whenever $r \doteq s$ and $r' \doteq s'$, and
- the reaction relation is also closed under support translation, since $a \rightarrow\!\!\triangleright a'$ whenever $a \doteq c \circ r$ and $a' \doteq c \circ r'$ for some reaction rule $(r, r') \in \grave{\mathcal{R}}$.

Since the construction of the reaction relation closes under support translation, we need only consider one concretion, a *representative*, of an abstract rule in order to generate all the corresponding reactions:

**Proposition 7.4.1** (reaction rule representatives are sufficient)**.** *Let* $\grave{\mathbf{C}}(\grave{\mathcal{R}})$ *be a BaRS and let* $a \rightarrowtriangle a'$ *be a reaction generated by rule* $(r, r')$*. Then any other support equivalent rule* $(s, s')$*, i.e.,* $r \bumpeq s$ *and* $r' \bumpeq s'$*, generates the same reaction.*

*Proof.* We have $a \bumpeq c \circ r$ and $a' \bumpeq c \circ r'$ for some context $c$ for $r$ and $r'$, and must show that there is a context $c'$ for $s$ and $s'$ such that $a \bumpeq c' \circ s$ and $a' \bumpeq c' \circ s'$. This follows from the proof of Lemma 7.2.20 if we strengthen the requirement on the choice of $c'$ by also precluding the support of $s'$, i.e., $\rho'' : |c| \to \mathcal{S} \setminus (|s| \cup |s'|)$. ☐

Thus we need not close the reaction rule set under support translation. But the reaction relation is still somewhat unmanageable, since it is closed under support translation, and one wonders why we must be able to change the support of the context of a reaction? Indeed, this does not strictly increase the number of reactions, in a sense that we shall now make precise.

First, note that the reaction relation is indeed closed under support equivalence:

**Lemma 7.4.2.** *Let* $\grave{\mathbf{C}}(\grave{\mathcal{R}})$ *be a BaRS and let* $a \rightarrowtriangle a'$*. Then* $\forall b, b' : b \bumpeq a \wedge b' \bumpeq a' \Rightarrow b \rightarrowtriangle b'$*.*

*Proof.* We have $a \bumpeq c \circ r$ and $a' \bumpeq c \circ r'$ for some rule $(r, r')$ and context $c$. Since $b \bumpeq a \bumpeq c \circ r$ and $b' \bumpeq a' \bumpeq c \circ r'$ we also get $b \rightarrowtriangle b'$. ☐

Based on the above observations, it should be clear that a BaRS contains support equivalence classes of rules and reactions; this naturally leads to a notion of a *representative* BaRS:

**Definition 7.4.3** (representative basic reactive system (RBaRS))**.** A *representative basic reactive system*, written $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$, consists of an s-category $\grave{\mathbf{C}}$ equipped with a set $\grave{\mathcal{R}}$ of *reaction rules*.

The *reaction relation* $\rightarrowtriangle$ over agents is the smallest such that $a \rightarrowtriangle a'$ whenever $a = c \circ \rho \bullet r$ and $a' = c \circ \rho' \bullet r'$ for some reaction rule $(r, r')$, support translations $\rho, \rho'$, and context $c$ for $\rho \bullet r$ and $\rho' \bullet r'$. ☐

The intuition is that an RBaRS represents a BaRS by allowing us to single out representatives for each equivalence class of rules and reactions:

**Definition 7.4.4** (RBaRS corresponding to BaRS)**.** Let $\grave{\mathbf{C}}(\grave{\mathcal{R}})$ be a BaRS. Then the RBaRS *corresponding* to $\grave{\mathbf{C}}(\grave{\mathcal{R}})$ is $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}}_r)$, where $\grave{\mathcal{R}}_r$ contains a single chosen representative of each support equivalence class of rules, i.e.,

$$\forall (r, r') \in \grave{\mathcal{R}} : \exists (s, s') \in \grave{\mathcal{R}}_r : r \bumpeq s \wedge r' \bumpeq s'$$
$$\forall (r, r'), (s, s') \in \grave{\mathcal{R}}_r : r \bumpeq s \wedge r' \bumpeq s' \Rightarrow r = s \wedge r' = s'.$$

☐

**Proposition 7.4.5** (RBaRS corresponding to BaRS)**.** *The RBaRS corresponding to a BaRS is indeed an RBaRS.*

Conversely, we can easily construct a BaRS from an RBaRS:

**Definition 7.4.6** (BaRS corresponding to RBaRS)**.** Let $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ be an RBaRS. Then the BaRS *corresponding* to $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ is $\grave{\mathbf{C}}(\grave{\mathcal{R}}_*)$, where $\grave{\mathcal{R}}_*$ is the support equivalence closure of $\grave{\mathcal{R}}$, i.e., $\grave{\mathcal{R}}_* = \{(s, s') \mid (r, r') \in \grave{\mathcal{R}} \wedge r \bumpeq s \wedge r' \bumpeq s'\}$. ☐

**Proposition 7.4.7** (BaRS corresponding to RBaRS)**.** *The BaRS corresponding to an RBaRS is indeed a BaRS.*

Note that this construction is inverse to the previous one:

**Lemma 7.4.8.** *For any BaRS* $\grave{\mathbf{C}}(\grave{\mathcal{R}})$*, the BaRS* $\grave{\mathbf{C}}((\grave{\mathcal{R}}_r)_*)$ *obtained through Def. 7.4.4 followed by Def. 7.4.6 is the same, i.e.,* $\grave{\mathbf{C}}(\grave{\mathcal{R}}) = \grave{\mathbf{C}}((\grave{\mathcal{R}}_r)_*)$*.*

*Proof.* Immediate from the definitions. □

It is immediate from the definitions, that reactions in an RBaRS are indeed reactions in the corresponding BaRS:

**Proposition 7.4.9** (Representative Reactions are Reactions). *Let $\rightarrowtriangle_r$ and $\rightarrowtriangle_f$ denote the reaction relations of a RBaRS and its corresponding BaRS respectively. Then*

$$a \rightarrowtriangle_r a' \quad \Rightarrow \quad a \rightarrowtriangle_f a'.$$

It is also clear that RBaRSs in general have a smaller reaction relations than their corresponding BaRS, since they do not allow reaction to change the support of the context. But this has a very limited impact: any series of reactions in a BaRS can be matched by a series of reactions followed by a single support translation in an RBaRS:

**Proposition 7.4.10** (Representative Reactions are Sufficient). *Let $\rightarrowtriangle_r$ and $\rightarrowtriangle_f$ denote the reaction relations of an RBaRS $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ and its corresponding BaRS respectively. Then*

$$\forall n \in \mathbb{N}: \quad a \rightarrowtriangle_f^n a' \quad \Rightarrow \quad \exists a'' : a \rightarrowtriangle_r^n a'' \wedge a' \bumpeq a''.$$

*Proof.* By induction on $n$, the base case being trivial. In the induction case we have $a \rightarrowtriangle_f^{n-1} b \rightarrowtriangle_f a'$, $a \rightarrowtriangle_r^{n-1} c$, and $b \bumpeq c$, and must show $c \rightarrowtriangle_r a''$ and $a' \bumpeq a''$ for some $a''$.

From $b \rightarrowtriangle_f a'$ we get $b \bumpeq d \circ s$ and $a' \bumpeq d \circ s'$ for some rule $(s, s') \in \grave{\mathcal{R}}_*$, i.e., $b \bumpeq d \circ \rho \blacksquare r$ and $a' \bumpeq d \circ \rho' \blacksquare r'$ for some rule $(r, r') \in \grave{\mathcal{R}}$ and support translations $\rho, \rho'$. Since $c \bumpeq b \bumpeq d \circ \rho \blacksquare r$, we have $c = \rho'' \blacksquare (d \circ \rho \blacksquare r) = (\rho'' \blacksquare d) \circ ((\rho'' \circ \rho) \blacksquare r)$ for some support translation $\rho''$. We then choose any support translation $\rho'''$ such that $a'' \stackrel{\text{def}}{=} (\rho'' \blacksquare d) \circ (\rho''' \blacksquare r')$ is defined and thus get $c \rightarrowtriangle_r a''$. By Lemma 7.2.19 $a''$ is support equivalent to $a'$: $a' \bumpeq d \circ \rho' \blacksquare r' \bumpeq (\rho'' \blacksquare d) \circ (\rho''' \blacksquare r') = a''$. □

### Abstract Representative Basic Reactive Systems

Let us now show that, once we abstract identities away, the reaction relations of RBaRS and BaRS are the same. The construction and properties of abstract BaRSs from Section 7.2.2 transfer unchanged to RBaRS, so we shall not repeat them here. We shall use $\mathbf{C}_{\mathsf{r}}(\mathcal{R})$ to denote the abstract RBaRS obtained as the quotient of an RBaRS $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ by a dynamic abstraction.

An RBaRS has the same abstract reactions as its corresponding BaRS:

**Theorem 7.4.11** (abstract RBaRSs are abstract BaRSs). *Let $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ be an RBaRS and let $\grave{\mathbf{C}}(\grave{\mathcal{R}}_*)$ be the corresponding BaRS. Then the quotient RBaRS $\mathbf{C}_{\mathsf{r}}(\mathcal{R})$ and quotient BaRS $\mathbf{C}(\mathcal{R}_*)$, both obtained using the construction of Def. 7.2.30 and Def. 7.2.32, are the same.*

*Proof.* Given that both $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ and $\grave{\mathbf{C}}(\grave{\mathcal{R}}_*)$ have the same underlying s-category, the underlying spm categories of the quotients are also the same. Also, since abstraction includes support equivalence, we have $\mathcal{R} = \mathcal{R}_*$.

We now show that the reaction relations are also the same. Let $\rightarrowtriangle_f$, $\rightarrowtriangle_{[\![f]\!]}$, $\rightarrowtriangle_r$, and $\rightarrowtriangle_{[\![r]\!]}$ denote the reaction relations of $\grave{\mathbf{C}}(\mathcal{R})$, $\mathbf{C}(\mathcal{R})$, $\grave{\mathbf{C}}_{\mathsf{r}}(\mathcal{R})$, and $\mathbf{C}_{\mathsf{r}}(\mathcal{R})$ respectively.

$\rightarrowtriangle_{[\![f]\!]} \subseteq \rightarrowtriangle_{[\![r]\!]}$: Assume $[\![f]\!] \rightarrowtriangle_{[\![f]\!]} [\![f']\!]$. From Theorem 7.2.33 we have $f \rightarrowtriangle_f g'$ for some $g' \in [\![f']\!]$, and Prop. 7.4.10 then gives us $f \rightarrowtriangle_r h'$ for some $h' \bumpeq g'$. Since abstraction includes support equivalence we have $[\![h']\!] = [\![g']\!] = [\![f']\!]$, and Theorem 7.2.33 gives us $[\![f]\!] \rightarrowtriangle_{[\![r]\!]} [\![h']\!]$, we have $[\![f]\!] \rightarrowtriangle_{[\![r]\!]} [\![f']\!]$ as required.

$\rightarrowtriangle_{[\![r]\!]} \subseteq \rightarrowtriangle_{[\![f]\!]}$: Assume $[\![f]\!] \rightarrowtriangle_{[\![r]\!]} [\![f']\!]$. From Theorem 7.2.33 we have $f \rightarrowtriangle_r g'$ for some $g' \in [\![f']\!]$, and Prop. 7.4.9 then gives us $f \rightarrowtriangle_f g'$. Finally, Theorem 7.2.33 gives us $[\![f]\!] \rightarrowtriangle_{[\![f]\!]} [\![g']\!] = [\![f']\!]$ as required. □

Conversely, a BaRS has the same abstract reactions as its corresponding RBaRS:

**Theorem 7.4.12** (abstract BaRSs are abstract RBaRSs)**.** *Let $\grave{\mathbf{C}}(\grave{\mathcal{R}})$ be a BaRS and let $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}}_r)$ be the corresponding RBaRS. Then the quotient BaRS $\mathbf{C}(\mathcal{R})$ and quotient RBaRS $\mathbf{C}_{\mathsf{r}}(\mathcal{R}_r)$, both obtained using the construction of Def. 7.2.30 and Def. 7.2.32, are the same.*

*Proof.* By Theorem 7.4.11, $\mathbf{C}_{\mathsf{r}}(\mathcal{R}_r)$ and $\mathbf{C}((\mathcal{R}_r)_*)$ are the same. The latter is the quotient of $\grave{\mathbf{C}}((\grave{\mathcal{R}}_r)_*)$, which, by Lemma 7.4.8, is the same as $\grave{\mathbf{C}}(\grave{\mathcal{R}})$, and thus $\mathbf{C}_{\mathsf{r}}(\mathcal{R}_r) = \mathbf{C}_{\mathsf{r}}((\mathcal{R}_r)_*) = \mathbf{C}(\mathcal{R})$. □

### 7.4.2 Parametric Reactive Systems

Having tamed the use of support equivalence in BaRSs, we now turn our attention to the rule set blow-up caused by treating parametric reaction rules as generators of ground reaction rules. To avoid this blow-up, we generalize RBaRSs to parametric reactive systems (PRSs) where parametric reaction rules are first-class citizens.

**Definition 7.4.13** (parametric reactive systems (PRS))**.** A *parametric reactive system*, written $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$, consists of an s-category $\grave{\mathbf{C}}$ equipped with a set $\grave{\mathcal{R}}$ of parametric reaction rules, and two subcategories $\grave{\mathbf{I}}$ and $\grave{\mathbf{D}}$ of identities and parameters respectively. $\grave{\mathbf{C}}$ and $\grave{\mathbf{D}}$ must be closed under support translation.

A *parametric reaction rule* is a triple of the form

$$(R : I \to J, R' : I' \to J, \bar{\eta}_{J',S})$$

where $R$ is the *parametric redex*, $R'$ the *parametric reactum*, and the *instance function* is a function family $\bar{\eta}_{J' \in \grave{\mathbf{I}}, S \subset \mathcal{S}} : \grave{\mathbf{D}}(\epsilon, I \otimes J') \to \grave{\mathbf{D}}(\epsilon, I' \otimes J')$ defined for all finite $S$ and whenever $I \otimes J'$ is defined.

Furthermore, instantiation maps must respect support equivalence and choose sufficiently fresh support, i.e.,

1. $d \mathrel{\overset{\frown}{-}} d' \Rightarrow \bar{\eta}_{J',S}(d) \mathrel{\overset{\frown}{-}} \bar{\eta}_{J',S'}(d')$ for any finite $S, S' \subset \mathcal{S}$, and
2. $|\bar{\eta}_{J',S}(d)| \# S$.

We shall often omit $J'$ and/or $S$ when they are evident from the context.

The *reaction relation* $\rightarrowtriangle$ over agents $a, a' \in \grave{\mathbf{C}}(\epsilon, \cdot)$ is the smallest such that $a \rightarrowtriangle a'$ whenever $a = c \circ (\rho \blacksquare R \otimes \mathsf{id}_{J'}) \circ d$ and $a' = c \circ (\rho' \blacksquare R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c| \cup \mathsf{rng}(\rho')}(d)$ for some parametric reaction rule $(R : I \to J, R' : I' \to J, \bar{\eta})$, support translations $\rho$, $\rho'$ with $\mathsf{dom}(\rho) = |R|$ and $\mathsf{dom}(\rho') = |R'|$, context $c$ for $\rho \blacksquare R \otimes \mathsf{id}_{J'}$ and $\rho' \blacksquare R' \otimes \mathsf{id}_{J'}$, parameter $d \in \grave{\mathbf{D}}(\epsilon, I \otimes J')$, and identity $\mathsf{id}_{J'} \in \grave{\mathbf{I}}$. □

To some extent, one could argue that we have simply moved the infinitude to the instance function. However, in the case of bigraphs the instance function is finitely representable, cf. Def. 7.2.37, so bigraphical PRSs with finite sets of reaction rules can be directly and finitely represented in an implementation.

Another important difference, when we compare this definition to Def. 7.4.3, and in particular the definition of the reaction relation, is that we have factored the parameter $d$ out of the ground redex $r$. One would perhaps expect the equations to simply read $a = c \circ \rho \blacksquare R \circ d$ and $a' = c \circ \rho' \blacksquare R' \circ \bar{\eta}(d)$ – what is the purpose of the identities? The answer is that this enables parameter and context to be connected without the involvement of the redex. To illustrate this, let us examine how BRSs can be expressed as PRSs.

### Bigraphical Parametric Reactive Systems

In bigraphs, context and parameter may share links (and nothing else) without the involvement of the redex. To see that this is the case, let us examine Milner's generation of ground rules from parametric ones, cf. Def. 7.2.37: It relies on the bigraph specific nesting operator '.' (Def. 7.2.14) which is derived from the parallel product '$\|$' (Def. 7.2.12) which again can be seen as derived from the tensor product '$\otimes$' . Unfolding the nestings and applying Prop. 7.2.13, we obtain

$$\begin{aligned} R.d &= (R \parallel \mathsf{id}_X) \circ d & &= \sigma(R \otimes \tau \circ \mathsf{id}_X) \circ d \\ R'.\bar{\eta}(d) &= (R' \parallel \mathsf{id}_X) \circ \bar{\eta}(d) & &= \sigma(R' \otimes \tau \circ \mathsf{id}_X) \circ \bar{\eta}(d) \end{aligned}$$

for a suitable bijection $\tau : X \to X'$ and a substitution $\sigma$ that ensure definedness as well as the aliasing of the shared names between $R$ and $d$.

Ignoring $\tau$, this resembles an instance of our PRS reactions, namely in the case where the context is a substitution that aliases some of the links of the parameter and redex. But what about $\tau$? The purpose of $\tau$ is to rename the links of the parameter such that the tensor product is defined. But the names of the parameter are internal to the reaction, as they only serve as mediators in the decomposition of the agent into context, redex, and parameter. To see this, let us rewrite $R.d$ a bit more:

$$\begin{aligned} R.d &= (R \otimes \tau \circ \mathsf{id}_X) \circ d && \text{by def. of '.' and Prop. 7.2.13} \\ &= \sigma(R \circ \mathsf{id}_m \otimes \mathsf{id}_{X'} \circ \tau) \circ d && \text{by def. of identities} \\ &= \sigma(R \otimes \mathsf{id}_{X'}) \circ (\mathsf{id}_m \otimes \tau) \circ d && \text{since } \otimes \text{ is a functor} \end{aligned}$$

Since $d$ is discrete so is $(\mathsf{id}_m \otimes \tau) \circ d$ and thus $R.d$ corresponds to the left hand side of a parametric reaction.

We shall now make this precise by defining a bigraphical PRS and showing that the ground bigraphical reaction rules generated from parametric ones are reactions in that PRS:

**Definition 7.4.14** (bigraphical parametric reactive system (BPRS)). A *bigraphical parametric reactive system* over $\mathcal{K}$ with bigraphical parametric reaction rules $`\mathcal{R}$, written $`\mathrm{BG}(\mathcal{K}, `\mathcal{R})$, is the parametric reactive system $`\mathrm{BG}(\mathcal{K})(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ where $`\mathbf{D}$ consists of the discrete ground bigraphs of $`\mathrm{BG}(\mathcal{K})$ and $`\mathbf{I}$ consists of the zero-width (i.e., link graph) identities. For each rule $(R : m \to J, R' : m' \to J, \eta) \in `\mathcal{R}$ we interpret $\eta$ as the corresponding instance function family $\bar{\eta}_{X,S}$ as given in Def. 7.2.35. □

**Proposition 7.4.15.** *Let $`\mathrm{BG}(\mathcal{K}, `\mathcal{R})$ be a BPRS. Then $R.d \twoheadrightarrow R'.\bar{\eta}(d)$ is a reaction in the BPRS for any parametric rule $(R : m \to J, R' : m' \to J, \eta) \in `\mathcal{R}$ and discrete parameter $d : \langle m, Y \rangle$.*

*Proof.* By unfolding the derived operators in the left and right hand sides of the claimed reaction, and then rewriting them according to the categorical axioms, it becomes clear that it is indeed a reaction:

$$\begin{aligned} R.d &= \sigma(R \otimes \tau \circ \mathsf{id}_X) \circ d && \text{by def. of '.' and Prop. 7.2.13} \\ &= \sigma((\mathsf{Id}_{|R|} \bullet R) \circ \mathsf{id}_m \otimes \mathsf{id}_{X'} \circ \tau) \circ d && \text{by def. of identities} \\ &= \sigma((\mathsf{Id}_{|R|} \bullet R) \otimes \mathsf{id}_{X'}) \circ (\mathsf{id}_m \otimes \tau) \circ d && \text{since } \otimes \text{ is a functor} \end{aligned}$$

$$\begin{aligned} R'.\bar{\eta}(d) &= \sigma(R' \otimes \tau \circ \mathsf{id}_X) \circ \bar{\eta}(d) && \text{by def. of '.' and Prop. 7.2.13} \\ &= \sigma((\mathsf{Id}_{|R'|} \bullet R') \circ \mathsf{id}_{m'} \otimes \mathsf{id}_{X'} \circ \tau) \circ \bar{\eta}(d) && \text{by def. of identities} \\ &= \sigma((\mathsf{Id}_{|R'|} \bullet R') \otimes \mathsf{id}_{X'}) \circ (\mathsf{id}_{m'} \otimes \tau) \circ \bar{\eta}(d) && \text{since } \otimes \text{ is a functor} \\ &= \sigma((\mathsf{Id}_{|R'|} \bullet R') \otimes \mathsf{id}_{X'}) \circ \bar{\eta}((\mathsf{id}_m \otimes \tau) \circ d) && \text{by Prop. 7.2.36} \end{aligned}$$

Note that $\sigma$ and $\tau$ are validly chosen to be the same in both cases, since $R$ and $R'$ have the same outer names and ditto for $d$ and $\bar{\eta}(d)$. □

Note that Prop. 7.4.15 only covers the ground reaction rule $(R.d, R'.\bar{\eta}(d))$, though $(R : m \to J, R' : m' \to J, \eta)$ generates all rules on the form $(\rho \centerdot (R.d), \rho' \centerdot (R'.\bar{\eta}(d)))$. This is because our definition of PRSs does include arbitrary support translation of parameters in the reaction relation, just as it was the case for RBaRSs. We could distinguish between PRSs and *representative* PRSs, analogously to the distinction between BaRSs and RBaRS, in which case all the generated ground reaction rules would be reactions in the PRS but not the representative PRS. However, we leave this as an exercise to the reader as we shall not need this distinction.

### Relating Concrete PRSs and RBaRSs

Having demonstrated the crux of the correspondence between BRSs and BPRSs, let us now return to the general case of RBaRSs and PRSs. First, note that a RBaRS is a PRS in a very straightforward sense:

**Definition 7.4.16** (PRS corresponding to a RBaRS)**.** Let $`\mathbf{C}_\mathsf{r}(`\mathcal{R})$ be a RBaRS. Then the *corresponding* PRS is $`\mathbf{C}(`\mathcal{R} \times \{\mathsf{id}_{\epsilon \to \epsilon}\}, \mathbf{1}, \mathbf{1})$ (where $\epsilon \in `\mathbf{C}$ is the singleton object of $\mathbf{1}$). □

**Proposition 7.4.17** (PRS corresponding to a RBaRS)**.** *The PRS corresponding to an RBaRS is indeed a PRS.*

Dually, it is straightforward to derive a RBaRS from a PRS by simply generating ground reaction rules:

**Definition 7.4.18** (RBaRS corresponding to a PRS)**.** Let $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ be a PRS. Then the corresponding RBaRS is $`\mathbf{C}_\mathsf{r}(`\mathcal{R}')$ where $`\mathcal{R}'$ is generated from $`\mathcal{R}$ as follows: $(r, r') \in `\mathcal{R}'$ whenever $r = (R \otimes \mathsf{id}_{J'}) \circ d$ and $r' = (R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|R'|}(d)$ for some parametric reaction rule $(R : I \to J, R' : I' \to J, \bar{\eta})$, parameter $d \in `\mathbf{D}(\epsilon, I \otimes J')$, and identity $\mathsf{id}_{J'} \in `\mathbf{I}$. □

**Proposition 7.4.19** (RBaRS corresponding to a PRS)**.** *The RBaRS corresponding to a PRS is indeed an RBaRS.*

Note that the first of these constructions is the inverse of the second:

**Lemma 7.4.20.** *For any RBaRS $`\mathbf{C}_\mathsf{r}(`\mathcal{R})$, the RBaRS $`\mathbf{C}_\mathsf{r}(`\mathcal{R}')$ obtained through Def. 7.4.16 followed by Def. 7.4.18 is the same.*

*Proof.* Immediate from the definitions. □

From these definitions, it is no surprise that PRS reactions are also reactions in the corresponding RBaRS:

**Proposition 7.4.21** (Parametric Reactions are Representative Reactions)**.** *Let $\rightarrowtriangle_p$ and $\rightarrowtriangle_r$ denote the reaction relations of a PRS $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ and its corresponding RBaRS, respectively. Then*

$$a \rightarrowtriangle_p a' \quad \Rightarrow \quad a \rightarrowtriangle_r a'.$$

*Proof.* Assume $a \rightarrowtriangle_p a'$, i.e., $a = c \circ (\rho \centerdot R \otimes \mathsf{id}_{J'}) \circ d$ and $a' = c \circ (\rho' \centerdot R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}(d)$ for some parametric reaction rule $(R : I \to J, R' : I' \to J, \bar{\eta})$, support translations $\rho$, $\rho'$, context $c$ for $\rho \centerdot R \otimes \mathsf{id}_{J'}$ and $\rho' \centerdot R' \otimes \mathsf{id}_{J'}$, parameter $d \in `\mathbf{D}(\epsilon, I \otimes J')$, and identity $\mathsf{id}_{J'} \in `\mathbf{I}$.

By Def. 7.4.18 $((R \otimes \mathsf{id}_{J'}) \circ d, (R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|R'|}(d))$ is a rule in the corresponding RBaRS, and so, by Def. 7.4.3, $c \circ (\rho \uplus \mathsf{id}_{|d|}) \centerdot ((R \otimes \mathsf{Id}_{J'}) \circ d) \rightarrowtriangle_r c \circ (\rho' \uplus \rho'') \centerdot ((R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|R'|}(d))$, where $\rho''$ is a witness of $\bar{\eta}_{|c| \cup \mathsf{rng}(\rho')}(d) \doteq \bar{\eta}_{|R'|}(d)$. Applying the definition of support translation it is easy to see that this is indeed $a \rightarrowtriangle_r a'$. □

As we saw in the case of bigraphs, the reaction relation of a PRS will be smaller than that of its corresponding RBaRS, since RBaRSs allow support translation of parameters. But the PRS reaction relation characterizes that of the corresponding RBaRS, similar to how the reactions of an RBaRS characterizes the reactions of the corresponding BaRS, cf. Prop. 7.4.10: any series of RBaRS reactions can be matched by a series of PRS reactions followed by a single support translation:

**Proposition 7.4.22** (Parametric Reactions are Sufficient)**.** *Let $\rightarrowtriangle_p$ and $\rightarrowtriangle_r$ denote the reaction relations of a PRS $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ and its corresponding RBaRS, respectively. Then*

$$\forall n \in \mathbb{N} : \quad a \rightarrowtriangle_r^n a' \quad \Rightarrow \quad \exists a'' : a \rightarrowtriangle_p^n a'' \wedge a' \bumpeq a''.$$

*Proof.* By induction on $n$, the base case being trivial. In the induction case we have $a \rightarrowtriangle_r^{n-1} b \rightarrowtriangle_r a'$, $a \rightarrowtriangle_p^{n-1} c$, and $b \bumpeq c$, and must show $c \rightarrowtriangle_p a''$ and $a' \bumpeq a''$ for some $a''$. Let $\rho'' : |b| \to |c|$ be a witness of $b \bumpeq c$.

From $b \rightarrowtriangle_r a'$ we get $b = e \circ \rho \blacksquare r$ and $a' = e \circ \rho' \blacksquare r'$ for some reaction rule $(r, r')$, support translations $\rho, \rho'$, and context $e$ for $\rho \blacksquare r$ and $\rho' \blacksquare r'$. By Def. 7.4.18 we must have $r = (R \otimes \mathsf{id}_{J'}) \circ d$ and $r' = (R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|R'|}(d)$ for some parametric reaction rule $(R : I \to J, R' : I' \to J, \bar{\eta})$, parameter $d \in `\mathbf{D}(\epsilon, I \otimes J')$, and identity $\mathsf{id}_{J'} \in `\mathbf{I}$.

We therefore get the following equalities for $c$:

$$\begin{aligned}
c &= \rho'' \blacksquare b && \rho'' \text{ is a witness of } b \bumpeq c \\
&= \rho'' \blacksquare (e \circ \rho \blacksquare r) && b \text{ is the LHS of a representative reaction} \\
&= \rho'' \blacksquare (e \circ \rho \blacksquare ((R \otimes \mathsf{id}_{J'}) \circ d)) && (r, r') \text{ generated from parametric rule} \\
&= \rho'' \blacksquare e \circ ((\rho'' \circ \rho) \blacksquare R \otimes \mathsf{id}_{J'}) \circ ((\rho'' \circ \rho) \blacksquare d) && \text{by def. of supp. trans.}
\end{aligned}$$

Letting $a'' = \rho'' \blacksquare e \circ (\rho' \blacksquare R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|\rho'' \blacksquare e| \cup |\rho' \blacksquare R'|}((\rho'' \circ \rho) \blacksquare d)$, Def. 7.4.13 gives us $c \rightarrowtriangle_p a''$, and $a' \bumpeq a''$ as witnessed by $\rho''' = \rho'' \restriction_{|e|} \uplus \mathsf{Id}_{|\rho' \blacksquare R'|} \uplus \rho'''' \circ (\rho' \restriction_{|\bar{\eta}(d)|})^{-1}$, where $\rho''''$ is a witness of $\bar{\eta}_{|R'|}(d) \bumpeq \bar{\eta}_{|\rho'' \blacksquare e| \cup |\rho' \blacksquare R'|}((\rho'' \circ \rho) \blacksquare d)$:

$$\begin{aligned}
\rho''' \blacksquare a' &= \rho''' \blacksquare (e \circ \rho' \blacksquare r') && a \text{ is the RHS of a representative reaction} \\
&= \rho''' \blacksquare (e \circ \rho' \blacksquare ((R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|R'|}(d))) && (r, r') \text{ generated from parametric rule} \\
&= \rho'' \blacksquare e \circ (\rho' \blacksquare R' \otimes \mathsf{id}_{J'}) \circ \rho'''' \blacksquare \bar{\eta}_{|R'|}(d) && \text{by def. of supp. trans.} \\
&= \rho'' \blacksquare e \circ (\rho' \blacksquare R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|\rho'' \blacksquare e| \cup |\rho' \blacksquare R'|}((\rho'' \circ \rho) \blacksquare d) && \text{since } \bar{\eta} \text{ respects supp. eq.}
\end{aligned}$$

□

**Abstract Parametric Reactive Systems**

As we have seen above, the difference between the reaction relations of a PRS and its corresponding RBaRS is that the latter includes support translation of parameters. So we expect that if we quotient these systems with an abstraction, as in Section 7.4.1, we get the same abstract reactive systems. This is indeed the case, as we shall show below.

But first, we must extend the abstraction constructions and results of Section 7.2.2 to PRSs:

**Definition 7.4.23** (abstract PRS)**.** A PRS is *abstract* if its underlying s-category is an spm category. □

Since we shall need to abstract instantiation maps, we must require abstractions to be well-behaved with respect to these:

**Definition 7.4.24** (dynamic PRS abstraction)**.** In a PRS $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$, an abstraction $\equiv$, as defined in Def. 7.2.28, is *dynamic* if it respects reaction and instantiation, i.e.,

1. if $f \rightharpoondown f'$ and $g \equiv f$ then $g \rightharpoondown g'$ for some $g' \equiv f'$, and
2. if $(R : I \to J, R', \bar{\eta}) \in \grave{\mathcal{R}}$, $d, d' \in \grave{\mathbf{D}}(\epsilon, I \otimes J')$ and $d \equiv d'$ then $\bar{\eta}_S(d) \equiv \bar{\eta}_{S'}(d')$.

□

Clearly, support equivalence is a dynamic abstraction on any PRS:

**Proposition 7.4.25** (support equivalence is a dynamic PRS abstraction)**.** *Support equivalence* $\simeq$ *is a dynamic abstraction on a PRS* $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$.

More importantly, the lean-support equivalence of bigraphs is a dynamic abstraction on BPRSs:

**Proposition 7.4.26** (lean-support equivalence is a dynamic BPRS abstraction)**.** *Lean-support equivalence* $\risingdotseq$ *is a dynamic abstraction on a bigraphical PRS* $\grave{\mathrm{B}}\mathrm{G}(\mathcal{K}, \grave{\mathcal{R}})$.

*Proof.* The proof is similar to that of Prop. 7.2.40 only more tedious. □

We can now define how to obtain abstract PRSs by quotienting by dynamic abstractions:

**Definition 7.4.27** (quotient PRS)**.** Let $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$ be a PRS, and $\equiv$ a dynamic abstraction on $\grave{\mathbf{C}}$. Then define $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$, the quotient of $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$ by $\equiv$, as follows:

- $\mathbf{C} = \grave{\mathbf{C}}/\equiv$,
- $\mathbf{D} = \grave{\mathbf{D}}/\equiv$,
- $\mathbf{I} = \grave{\mathbf{I}}/\equiv$, and
- $\mathcal{R} = \{([\![R]\!], [\![R']\!], \bar{\eta}) \mid (R, R', \bar{\eta}) \in \grave{\mathcal{R}}\}$.

We define $\bar{\eta}_S([\![d]\!]) \stackrel{\text{def}}{=} [\![\bar{\eta}_{S'}(d)]\!]$ whenever $\bar{\eta}_{S'}(d)$ is defined; this is unambiguous since $\equiv$ is dynamic. □

**Theorem 7.4.28** (abstract PRS)**.** *The construction of Def. 7.2.30 and Def. 7.4.27, applied to a concrete PRS* $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$*, yields an abstract PRS* $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$*, whose underlying spm category* $\mathbf{C}$ *is the codomain of a functor of s-categories*

$$[\![\cdot]\!] : \grave{\mathbf{C}} \to \mathbf{C}.$$

*Moreover the construction preserves the reaction relation, in the following sense:*

1. *if* $f \rightharpoondown f'$ *in* $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$ *then* $[\![f]\!] \rightharpoondown [\![f']\!]$ *in* $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$
2. *if* $[\![f]\!] \rightharpoondown g'$ *in* $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$ *then* $f \rightharpoondown f'$ *in* $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$ *for some* $f'$ *with* $[\![f']\!] = g'$.

*Proof.* The first part follows immediately from Lemma 7.2.31 and Def. 7.4.23.

1: We have $f = c \circ (\rho \mathbin{\text{▪}} R \otimes \mathsf{id}_{J'}) \circ d$ and $f' = c \circ (\rho' \mathbin{\text{▪}} R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c| \cup \mathsf{rng}(\rho')}(d)$ for some parametric reaction rule $(R : I \to J, R' : I' \to J, \bar{\eta})$, support translations $\rho$, $\rho'$ with $\mathsf{dom}(\rho) = |R|$ and $\mathsf{dom}(\rho') = |R'|$, context $c$ for $\rho \mathbin{\text{▪}} R \otimes \mathsf{id}_{J'}$ and $\rho' \mathbin{\text{▪}} R' \otimes \mathsf{id}_{J'}$, parameter $d \in \grave{\mathbf{D}}(\epsilon, I \otimes J')$, and identity $\mathsf{id}_{J'} \in \grave{\mathbf{I}}$.

Quotienting $f$ and $f'$ we get $[\![f]\!] = [\![c \circ (\rho \mathbin{\text{▪}} R \otimes \mathsf{id}_{J'}) \circ d]\!] = [\![c]\!] \circ ([\![R]\!] \otimes [\![\mathsf{id}_{J'}]\!]) \circ [\![d]\!]$ and $[\![f']\!] = [\![c \circ (\rho' \mathbin{\text{▪}} R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c| \cup \mathsf{rng}(\rho')}(d)]\!] = [\![c]\!] \circ ([\![R']\!] \otimes [\![\mathsf{id}_{J'}]\!]) \circ \bar{\eta}_{\emptyset}([\![d]\!])$, since $[\![\cdot]\!]$ is a functor, abstraction includes support equivalence, and $\bar{\eta}_{\emptyset}([\![d]\!]) = [\![\bar{\eta}_{|c| \cup \mathsf{rng}(\rho')}(d)]\!]$. Thus, $[\![f]\!] \rightharpoondown [\![f']\!]$ as required.

2: We have $[\![f]\!] = [\![c]\!] \circ ([\![R]\!] \otimes [\![\mathsf{id}_{J'}]\!]) \circ [\![d]\!]$ and $g' = [\![c]\!] \circ ([\![R']\!] \otimes [\![\mathsf{id}_{J'}]\!]) \circ \bar{\eta}_{\emptyset}([\![d]\!])$ for some parametric reaction rule $(R : I \to J, R' : I' \to J, \bar{\eta})$, context $[\![c]\!]$ for $[\![R]\!] \otimes [\![\mathsf{id}_{J'}]\!]$ and $[\![R']\!] \otimes [\![\mathsf{id}_{J'}]\!]$, parameter

$[\![d]\!] \in \mathbf{D}(\epsilon, I \otimes J')$, and identity $[\![\mathsf{id}_{J'}]\!] \in \mathbf{I}$ (we disregard the support translations $\rho, \rho'$ since spm categories are s-categories with empty support).

Choose some context $c' \in [\![c]\!]$ and parameter $d' \in [\![d]\!]$ with supports that are disjoint from each other and from $|R|, |R'|$. By definition of composition in $\mathbf{C}$ and since $\bar{\eta}_\emptyset([\![d]\!]) = [\![\bar{\eta}_{|c'| \cup |R'|}(d')]\!]$, we get $[\![f]\!] = [\![c]\!] \circ ([\![R]\!] \otimes [\![\mathsf{id}_{J'}]\!]) \circ [\![d]\!] = [\![c' \circ (R \otimes \mathsf{id}_{J'}) \circ d']\!]$ and $g' = [\![c]\!] \circ ([\![R']\!] \otimes [\![\mathsf{id}_{J'}]\!]) \circ \bar{\eta}_\emptyset([\![d]\!]) = [\![c' \circ (R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c'| \cup |R'|}(d')]\!]$. By Def. 7.4.13 we have $f \equiv c' \circ (R \otimes \mathsf{id}_{J'}) \circ d' \longrightarrow c' \circ (R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c'| \cup |R'|}(d')$ and since $\equiv$ is dynamic, there is some $f' \equiv c' \circ (R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c'| \cup |R'|}(d') \in g'$ such that $f \longrightarrow f'$ as required. □

Now we are ready for the main results of this section, which states that a PRS has the same abstract behavior as its corresponding RBaRS and vice versa:

**Theorem 7.4.29** (abstract PRSs are abstract RBaRSs)**.** *Let* $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$ *be a PRS and let* $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}}')$ *be the corresponding RBaRS. Then the quotient PRS* $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$ *and quotient RBaRS* $\mathbf{C}_{\mathsf{r}}(\mathcal{R}')$ *have the same reaction relations.*

*Proof.* Let $\longrightarrow_r$, $\longrightarrow_{[\![r]\!]}$, $\longrightarrow_p$, and $\longrightarrow_{[\![p]\!]}$ denote the reaction relations of $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}}')$, $\mathbf{C}_{\mathsf{r}}(\mathcal{R}')$, $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$, and $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$ respectively.

$\longrightarrow_{[\![r]\!]} \subseteq \longrightarrow_{[\![p]\!]}$: Assume $[\![f]\!] \longrightarrow_{[\![r]\!]} [\![f']\!]$. From Theorem 7.2.33 we have $f \longrightarrow_r g'$ for some $g' \in [\![f']\!]$, and Prop. 7.4.22 then gives us $f \longrightarrow_p h'$ for some $h' \simeq g'$. Since abstraction includes support equivalence we have $[\![h']\!] = [\![g']\!] = [\![f']\!]$, and Theorem 7.4.28 gives us $[\![f]\!] \longrightarrow_{[\![p]\!]} [\![h']\!]$, we have $[\![f]\!] \longrightarrow_{[\![p]\!]} [\![f']\!]$ as required.

$\longrightarrow_{[\![p]\!]} \subseteq \longrightarrow_{[\![r]\!]}$: Assume $[\![f]\!] \longrightarrow_{[\![p]\!]} [\![f']\!]$. From Theorem 7.4.28 we have $f \longrightarrow_p g'$ for some $g' \in [\![f']\!]$, and Prop. 7.4.21 then gives us $f \longrightarrow_r g'$. Finally, Theorem 7.2.33 gives us $[\![f]\!] \longrightarrow_{[\![r]\!]} [\![g']\!] = [\![f']\!]$ as required. □

Conversely, an RBaRS has the same abstract reactions as its corresponding PRS:

**Theorem 7.4.30** (abstract RBaRSs are abstract PRSs)**.** *Let* $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$ *be an RBaRS and let* $\grave{\mathbf{C}}(\grave{\mathcal{R}} \times \{\mathsf{id}_{\epsilon \to \epsilon}\}, \mathbf{1}, \mathbf{1})$ *be the corresponding PRS. Then the quotient RBaRS* $\mathbf{C}_{\mathsf{r}}(\mathcal{R})$ *and quotient PRS* $\mathbf{C}(\mathcal{R} \times \{\mathsf{id}_{\epsilon \to \epsilon}\}, \mathbf{1}, \mathbf{1})$ *are the same.*

*Proof.* By Theorem 7.4.29, $\mathbf{C}(\mathcal{R} \times \{\mathsf{id}_{\epsilon \to \epsilon}\}, \mathbf{1}, \mathbf{1})$ is equal to the quotient $\mathbf{C}_{\mathsf{r}}(\mathcal{R}')$ of its corresponding RBaRS $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}}')$. By Lemma 7.4.20, the latter is the same as $\grave{\mathbf{C}}_{\mathsf{r}}(\grave{\mathcal{R}})$, and thus $\mathbf{C}(\mathcal{R} \times \{\mathsf{id}_{\epsilon \to \epsilon}\}, \mathbf{1}, \mathbf{1}) = \mathbf{C}_{\mathsf{r}}(\mathcal{R}') = \mathbf{C}_{\mathsf{r}}(\mathcal{R})$. □

### 7.4.3 Stochastic Parametric Reactive Systems

We now proceed to give a stochastic semantics to PRSs, generalizing and recasting the work on stochastic reduction semantics for a subset of BRSs in [25].

Intuitively, this is done by interpreting reaction rules as follows: a redex models a physical configuration that may lead to reaction which, over stochastic time, results in the physical configuration described by the corresponding reactum. In other words, we assign reaction rules a stochastic speed which will allow us to assign stochastic behavior to reactions.

In more detail, we wish to associate reactions with a rate, which "is the parameter of an exponential distribution that characterizes the stochastic behavior of that reaction" [25]. The rate of a reaction is then derived from the reaction rules that generate that reaction as follows:

(a) We associate a *rate constant* $\varrho$ with every reaction rule (its speed).

(b) The sum of the rate constants of all the rule instances that generate a reaction is its rate.

It is desirable for the rate of a reaction to be determined from the matches in its left-hand-side, since the stochastic behavior of an agent may then be determined without considering the right-hand-sides of reaction. In other words, we want each match to determine a single reaction. This is not the case for the definition of PRSs in the previous section: since we are free to choose any support translation of the reactum a match leads to infinitely many reactions for non-trivial reactums. We shall therefore refine the definition of the PRS reaction relation such that the support translation of the reactum is deterministic.

First, let us make precise what a match is in a PRS.

### Matches

The usual definition of a bigraph match (cf. Def. 7.2.39) is too coarse-grained for PRSs, as it is defined up to support equivalence. Instead, we shall use the following definition:

**Definition 7.4.31** (match). In a PRS $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$, a *match* $o$ of a parametric rule $\mathsf{R} = (R : I \to J, R' : I' \to J, \bar{\eta}_{J',S})$ in an agent $a$ is a quadruple

$$(\rho, \mathsf{id}_{J'}, c, d)$$

where $\rho : |R| \rightarrowtail |a|$ is a support translation, $\mathsf{id}_{J'} \in `\mathbf{I}$ an identity, $c$ a context, and $d \in `\mathbf{D}(\epsilon, I \otimes J')$ a parameter such that $a = c \circ (\rho \bullet R \otimes \mathsf{id}_{J'}) \circ d$.

Two matches $(\rho, \mathsf{id}_I, c, d), (\rho', \mathsf{id}_{I'}, c', d')$ are regarded as the same if they differ only by an iso between $I$ and $I'$; otherwise they are *distinct*. We say that a match *results* in $a'$ if $c \circ (\rho \bullet R \otimes \mathsf{id}_{J'}) \circ d \rightarrowtriangle a'$. We write $\mu_{\mathsf{R}}[a]$ for the number of distinct matches of $\mathsf{R}$ in $a$, and $\mu_{\mathsf{R}}[a, a']$ for the number of distinct matches of $\mathsf{R}$ in $a$ resulting in $a'$. □

Note that non-trivial support automorphisms, i.e., a non-identity support translation $\rho : |G| \to |G|$ such that $\rho \bullet G = G$, give rise to distinct matches:

**Lemma 7.4.32.** *Given a* match $o = (\rho, \mathsf{id}_{J'}, c, d)$ *of a parametric rule* $\mathsf{R} = (R : I \to J, R' : I' \to J, \bar{\eta}_{J',S})$ *in an agent* $a$*, all in a PRS* $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$*. Then any support automorphism* $\rho' : |R| \to |R|$ *for* $R$ *gives rise to a match* $o' = (\rho \circ \rho', \mathsf{id}_{J'}, c, d)$*. Furthermore, if* $\rho'$ *is not the identity then* $o$ *and* $o'$ *are distinct.*

*Proof.* Since

$$\begin{aligned} a &= c \circ (\rho \bullet R \otimes \mathsf{id}_{J'}) \circ d \\ &= c \circ (\rho \bullet (\rho' \bullet R) \otimes \mathsf{id}_{J'}) \circ d \\ &= c \circ ((\rho \circ \rho') \bullet R \otimes \mathsf{id}_{J'}) \circ d \end{aligned}$$

$o'$ is an match. Assuming $\rho'$ is not an identity we have $\rho \neq \rho \circ \rho'$ since both $\rho$ and $\rho'$ are bijections, and thus $o'$ is distinct from $o$. □

This is a point where our stochastic semantics differ from that of Krivine et al. [25]: they consider matches the same if they differ by a support automorphism. However, the difference is just a matter of convention and boils down to a scaling of rate constants by the number of support automorphisms of the corresponding redexes – we leave the details as an exercise to the reader.

### Deterministic Support Translation of Reactums

Let us now turn to the matter of ensuring that a match determines a single reaction. The solution is rather simple: we shall simply assume the existence of a family of canonical support translations, $\bar{\rho}_{S \subset \mathcal{S}, T \subset \mathcal{S}} : S \to \mathcal{S} \setminus T$, defined for finite $S$ and $T$. For a reactum $R'$ to be inserted in a context with support $T$, $\bar{\rho}_{|R'|,T}$ is the canonical support translation of $R'$ such that its support becomes disjoint from the context. We shall often omit $S$ and/or $T$ when they are evident from the context.

As was the case for the instantiation families $\bar{\eta}_{J',S}$, one could think that we have introduced an intractable infinite structure. However, $\bar{\rho}_{S,T}$ is simply a technical measure that need not be specified or represented in practice: when we abstract away support, the choice of the support translation family becomes irrelevant. In other words, as long as we are only interested in the abstract behavior of SPRSs, an implementation is free to generate suitable support for reactums as it pleases.

We can now define stochastic PRSs:

**Definition 7.4.33** (stochastic parametric reactive systems (SPRS))**.** A *stochastic parametric reactive system*, written $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$, is a PRS apart from the addition of rates and that reactum support is chosen canonically in the reaction relation:

A *stochastic parametric reaction rule* is a quadruple of the form

$$(R : I \to J, R' : I' \to J, \bar{\eta}_{J',S}, \varrho)$$

where the first three elements are as before and $\varrho \in \mathbb{R}_+$ is its *rate constant.*

The *reaction relation* $\twoheadrightarrow$ over agents $a, a' \in `\mathbf{C}(\epsilon, \cdot)$ is the smallest such that $a \twoheadrightarrow a'$ whenever $(\rho, \mathsf{id}_{J'}, c, d)$ is a match of some parametric reaction rule $\mathsf{R} = (R, R', \bar{\eta}, \varrho)$ in $a$ and $a' = c \circ (\bar{\rho}_{|R'|,|c|} \bullet R' \otimes \mathsf{id}_{J'}) \circ \bar{\eta}_{|c| \cup \mathsf{rng}(\bar{\rho}_{|R'|,|c|})}(d)$.

We define the rate $\mathit{rate}[a, a']$ of a reaction $a \twoheadrightarrow a'$ to be

$$\mathit{rate}[a, a'] \stackrel{\text{def}}{=} \sum_{\mathsf{R}=(R,R',\bar{\eta}_{J',S},\varrho) \in `\mathcal{R}} \varrho \cdot \mu_{\mathsf{R}}[a, a'].$$

□

For now, let us disregard stochastics and focus on the relation between SPRSs and PRSs (cf. Def. 7.4.13). The difference from PRSs is that we have refined the reaction relation such that a match determines a single reaction instead of an infinite family of support equivalent reactions (for non-trivial reactums).

Let us make the relation between SPRSs and PRSs precise in the same manner used in the previous sections; the proofs are trivial so we omit them.

**Definition 7.4.34** (SPRS corresponding to PRS)**.** Let $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ be a PRS. Then the SPRS *corresponding* to $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ is $`\mathbf{C}_s(`\mathcal{R}', `\mathbf{D}, `\mathbf{I})$ where

$$\begin{aligned} `\mathcal{R}' = \{&(R : I \to J, R' : I' \to J, \bar{\eta}_{J',S}, 1) \\ &\mid (R : I \to J, R' : I' \to J, \bar{\eta}_{J',S}) \in `\mathcal{R}\}. \end{aligned}$$

□

**Proposition 7.4.35** (SPRS corresponding to PRS)**.** *The SPRS corresponding to a PRS is indeed an SPRS.*

**Definition 7.4.36** (PRS corresponding to SPRS)**.** Let $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ be an SPRS. Then the PRS *corresponding* to $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ is $`\mathbf{C}(`\mathcal{R}', `\mathbf{D}, `\mathbf{I})$ where

$$\begin{aligned} `\mathcal{R}' = \{&(R : I \to J, R' : I' \to J, \bar{\eta}_{J',S}) \\ &\mid (R : I \to J, R' : I' \to J, \bar{\eta}_{J',S}, \varrho) \in `\mathcal{R}\}. \end{aligned}$$

□

**Proposition 7.4.37** (PRS corresponding to SPRS)**.** *The PRS corresponding to an SPRS is indeed a PRS.*

**Lemma 7.4.38.** *For any PRS $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$, the PRS $`\mathbf{C}(`\mathcal{R}', `\mathbf{D}, `\mathbf{I})$ obtained through Def. 7.4.34 followed by Def. 7.4.36 is the same.*

*Proof.* Immediate from the definitions. □

**Proposition 7.4.39** (Stochastic Parametric Reactions are Parametric Reactions)**.** *Let $\rightarrowtriangle_s$ and $\rightarrowtriangle_p$ denote the reaction relations of an SPRS $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ and its corresponding PRS, respectively. Then*

$$a \rightarrowtriangle_s a' \quad \Rightarrow \quad a \rightarrowtriangle_p a'.$$

**Proposition 7.4.40** (Stochastic Parametric Reactions are Sufficient)**.** *Let $\rightarrowtriangle_s$ and $\rightarrowtriangle_p$ denote the reaction relations of an SPRS $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ and its corresponding PRS, respectively. Then*

$$\forall n \in \mathbb{N} : \quad a \rightarrowtriangle_p^n a' \quad \Rightarrow \quad \exists a'' : a \rightarrowtriangle_s^n a'' \wedge a' \doteq a''.$$

### Abstract Stochastic Parametric Reactive Systems

Given the relations between PRSs and SPRSs we saw above it is clear that the abstractions of their reaction relations are the same. But before we can make this formal, we must first define how to construct abstract SPRSs. The abstraction constructions and results for PRSs (cf. Def. 7.4.2) transfer directly to SPRSs, except that the quotient construction must be extended to handle rates:

**Definition 7.4.41** (quotient SPRS)**.** Let $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ be an SPRS, and $\equiv$ a dynamic abstraction on $`\mathbf{C}$. Then define $\mathbf{C}_s(\mathcal{R}, \mathbf{D}, \mathbf{I})$, the quotient of $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ by $\equiv$, as in Def. 7.4.27.

The rate of reaction in $\mathbf{C}_s(\mathcal{R}, \mathbf{D}, \mathbf{I})$ is defined as:

$$\mathit{rate}[\hat{a}, \hat{a}'] = \sum_{a' \in \hat{a}'} \mathit{rate}[a, a'] \quad \text{for any } a \in \hat{a}.$$

□

Thus we define the rate of an abstract reaction by choosing a representative of its left-hand-side and then summing the rates of all reactions into the equivalence class of the right-hand-side.

This is well-defined since $\equiv$ is a *dynamic* abstraction, and thus reactions, and thereby matches, and rates are independent of the choice of representative:

**Proposition 7.4.42.** *If $a \equiv b$ for a dynamic abstraction $\equiv$, then*

$$\sum_{a' \in \hat{a}'} \mathit{rate}[a, a'] = \sum_{a' \in \hat{a}'} \mathit{rate}[b, a'].$$

*Proof.* Follows straightforwardly from the fact that $\equiv$ is dynamic (cf. Def. 7.4.24). □

We can now show that, indeed, SPRSs have the same abstract reactions as their corresponding PRSs:

**Theorem 7.4.43** (abstract SPRSs are abstract PRSs)**.** *Let $`\mathbf{C}_s(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ be an SPRS and let $`\mathbf{C}(`\mathcal{R}', `\mathbf{D}, `\mathbf{I})$ be the corresponding PRS. Then the quotient SPRS $\mathbf{C}_s(\mathcal{R}, \mathbf{D}, \mathbf{I})$ and quotient PRS $\mathbf{C}(\mathcal{R}', \mathbf{D}, \mathbf{I})$ have the same reaction relations.*

*Proof.* Follows easily from Prop. 7.4.39 and Prop. 7.4.40. □

Conversely, a PRS has the same abstract reactions as its corresponding SPRS:

**Theorem 7.4.44** (abstract PRSs are abstract SPRSs)**.** *Let $`\mathbf{C}(`\mathcal{R}, `\mathbf{D}, `\mathbf{I})$ be a PRS and let $`\mathbf{C}_s(`\mathcal{R}', `\mathbf{D}, `\mathbf{I})$ be the corresponding SPRS. Then the quotient PRS $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$ and quotient SPRS $\mathbf{C}_s(\mathcal{R}', \mathbf{D}, \mathbf{I})$ have the same reaction relations.*

*Proof.* By Theorem 7.4.43, $\mathbf{C}_s(\mathcal{R}', \mathbf{D}, \mathbf{I})$ has the same reaction relation as $\mathbf{C}(\mathcal{R}'', \mathbf{D}, \mathbf{I})$, the quotient of its corresponding PRS $\grave{\mathbf{C}}(\grave{\mathcal{R}}'', \grave{\mathbf{D}}, \grave{\mathbf{I}})$. By Lemma 7.4.38, the latter is the same as $\grave{\mathbf{C}}(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$, and thus $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I}) = \mathbf{C}(\mathcal{R}'', \mathbf{D}, \mathbf{I})$ so $\mathbf{C}_s(\mathcal{R}', \mathbf{D}, \mathbf{I})$ has the same reaction relation as $\mathbf{C}(\mathcal{R}, \mathbf{D}, \mathbf{I})$. $\square$

Finally, as a sanity check, we verify that rates are consistent with the reaction relation:

**Proposition 7.4.45** (consistency)**.** *Let $\grave{\mathbf{C}}_s(\grave{\mathcal{R}}, \grave{\mathbf{D}}, \grave{\mathbf{I}})$ be an SPRS and $\mathbf{C}_s(\mathcal{R}, \mathbf{D}, \mathbf{I})$ its quotient by a dynamic abstraction $\equiv$ on $\grave{\mathbf{C}}$. Then*

$$\mathit{rate}[a, a'] > 0 \text{ iff } a \rightarrowtriangle a', \quad \text{and} \quad \mathit{rate}[\hat{a}, \hat{a}'] > 0 \text{ iff } \hat{a} \rightarrowtriangle \hat{a}'$$

*Proof.* $\mathit{rate}[a, a'] > 0 \Rightarrow a \rightarrowtriangle a'$: From the definition of $\mathit{rate}[a, a']$ we see that there must be some rule $\mathsf{R} = (R, R', \bar{\eta}_{J',S}, \varrho)$ with $\mu_{\mathsf{R}}[a, a'] > 0$ and thus, by definition of $\mu_{\mathsf{R}}[a, a']$, $a \rightarrowtriangle a'$.

$a \rightarrowtriangle a' \Rightarrow \mathit{rate}[a, a'] > 0$: By the definition of the reaction relation, we have a match of a rule in $a$ resulting in $a'$ and thus $\mu_{\mathsf{R}}[a, a'] > 0$ which implies $\mathit{rate}[a, a'] > 0$.

$\mathit{rate}[\hat{a}, \hat{a}'] > 0 \Rightarrow \hat{a} \rightarrowtriangle \hat{a}'$: From the definition of $\mathit{rate}[\hat{a}, \hat{a}']$ we see that there must be some $a \in \hat{a}, a' \in \hat{a}'$ such that $\mathit{rate}[a, a'] > 0$ and thus, as shown above, $a \rightarrowtriangle a'$. Lastly, Theorem 7.4.28 gives us $\hat{a} \rightarrowtriangle \hat{a}'$.

$\hat{a} \rightarrowtriangle \hat{a}' \Rightarrow \mathit{rate}[\hat{a}, \hat{a}'] > 0$: Theorem 7.4.28 tells us that there must be some $a \in \hat{a}, a' \in \hat{a}'$ such that $a \rightarrowtriangle a'$ and thus, as shown above, $\mathit{rate}[a, a'] > 0$. Now it is obvious from its definition that $\mathit{rate}[\hat{a}, \hat{a}'] > 0$. $\square$

## 7.5 Bigraph Embeddings

In the previous section we defined stochastic parametric reactive systems. A key component in that development was to make precise the algebraic notion of a *match* of a parametric redex in an agent. The KaSim algorithm relies on a representation of matches as embeddings (cf. Section 7.3), so in this section we shall develop a general theory of bigraph embeddings, where embeddings of redexes are isomorphic to matches.

We shall exploit the orthogonality of the link and place graphs, by developing link and place graph embeddings independently and then combine them to obtain bigraph embeddings.

In overview, the development proceeds as follows:

**embedding maps:**
We define an embedding of a graph as the union of maps of identities (i.e., a support translation) and maps of the inner and outer faces. The maps must satisfy certain conditions that ensure structure preservation and correspondence with certain algebraic decompositions.

**embedding/context isomorphism:**
For place graphs and bigraphs, we show that embeddings are isomorphic to certain decompositions, giving constructions in both directions. For redexes this implies that embeddings and matches are isomorphic.

Link graphs seem to lack the necessary structure for embeddings to have this property.

We shall take special interest in the embeddings of a particular class of bigraphs: those that are *solid*. Simply put, a bigraph is solid if all elements of the outer and inner interfaces are connected to a node and not connected to each other. Solid bigraphs are interesting for two reasons: many bigraphical models in the literature have solid redexes[5] and an embedding of a solid bigraph is determined by a support translation of its nodes, making matches compactly representable.

[5]For example, all BRSs in [24, 25, 29] have solid redexes.

### 7.5.1 Link Graph Embeddings

Embeddings of link graphs are mostly what one would expect of a graph embedding: a pair of injections of the nodes and edges which preserve the structure of the embedded graph (i.e., a support translation). In addition, we need to specify how the names of the interfaces should be mapped; in bigraphs, a context is allowed to alias names, so any map from the outer face names to the links of the host graph will do. Dually, we map the names of the inner face to sets of points in the host graph.

The definition is based on Milner's definition of link graph inclusion [28], extended to cover link graphs in general and with a minor correction[6].

**Definition 7.5.1** (link graph embedding)**.** Let $G : X_G \to Y_G, H : X_H \to Y_H$ be two concrete link graphs. Then a *link graph embedding*, written $\phi : G \hookrightarrow H$, is a map $\phi : |G| \uplus X_G \uplus Y_G \to |H| \uplus \mathcal{P}(X_H \uplus P_H) \uplus Y_H$, where $\phi = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{e}} \uplus \phi^{\mathsf{i}} \uplus \phi^{\mathsf{o}}$ satisfies the following conditions:

**maps:**

(LGE-1) $\phi^{\mathsf{v}} : V_G \rightarrowtail V_H$ is an injective map

(LGE-2) $\phi^{\mathsf{e}} : E_G \rightarrowtail E_H$ is an injective map

(LGE-3) $\phi^{\mathsf{i}} : X_G \rightarrowtail \mathcal{P}(X_H \uplus P_H)$ is a fully injective map

(LGE-4) $\phi^{\mathsf{o}} : Y_G \to E_H \uplus Y_H$ is an arbitrary map

**injectivity:**

(LGE-5) $\mathsf{rng}(\phi^{\mathsf{e}}) \# \mathsf{rng}(\phi^{\mathsf{o}})$

(LGE-6) $\mathsf{rng}(\phi^{\mathsf{i}}) \# \mathsf{rng}(\phi^{\mathsf{port}})$

**surjective on edge points:**

(LGE-7) $\phi^{\mathsf{p}} \circ link_G^{-1} \restriction_{E_G} = link_H^{-1} \circ \phi^{\mathsf{e}}$

**structure preservation:**

(LGE-8) $ctrl_G = ctrl_H \circ \phi^{\mathsf{v}}$

(LGE-9) $\forall p \in X_G \uplus P_G : \forall p' \in \phi^{\mathsf{p}}(p) : (\phi^{\mathsf{l}} \circ link_G)(p) = link_H(p')$

where

$$\phi^{\mathsf{l}} = \phi^{\mathsf{e}} \uplus \phi^{\mathsf{o}} \qquad \text{(map of links)}$$
$$\phi^{\mathsf{port}}(v,i) = (\phi^{\mathsf{v}}(v), i) \qquad ((v,i) \in P_G) \qquad \text{(map of ports)}$$
$$\phi^{\mathsf{p}} = \phi^{\mathsf{i}} \uplus \phi^{\mathsf{port}} \qquad \text{(map of points).}$$

We do not take the codomain of $\phi$ as part of its definition, and thus it may be an embedding into several link graphs. We write $\phi \cdot G$ when applying the underlying support translation to $G$. If any of the maps are partial, $\phi$ is *partial*, written $\phi : G \hookrightarrow H$. Partial embeddings need only satisfy the conditions where they are defined; in particular the surjectivity condition only applies to an edge $e$ iff $\phi^{\mathsf{e}}$ is defined for $e$ and $\phi^{\mathsf{p}}$ is defined for $link_G^{-1}(e)$. A partial embedding is said to be *non-trivial* iff its range is non-empty. □

Condition (LGE-7) deserves an explanation:

[6] In [28] Milner missed that embeddings must be surjective on the points of an edge, cf. Example 1.

**Example 1.** Consider the following ground link graphs

$$\begin{aligned} G &= (\{v\}, \{e\}, \{v \mapsto K\}, \{(v,0) \mapsto e\}) : \emptyset \to \emptyset \\ H &= (\{v, v'\}, \{e\}, \{v \mapsto K, v' \mapsto K\}, \{(v,0) \mapsto e, (v',0) \mapsto e\}) : \emptyset \to \emptyset \\ ar(K) &= 1 \end{aligned}$$

Then the following would be a link graph embedding if we did not include condition (LGE-7):

$$\phi = \mathsf{Id}_{\{v,e\}} : G \hookrightarrow H$$

But there is no link graph $C$ such that $H = C \circ \phi \centerdot G$! □

The problem is that the context cannot add more points to an edge, so the points of an edge in $G$ must cover all the points the corresponding edge in $H$.

In [28] Milner showed that, in the case of ground link graphs, contexts and embeddings are isomorphic. Unfortunately, there is no such correspondence in the general case, as the following example shows:

**Example 2.** Consider the following link graphs

$$\begin{aligned} G &= (\{v\}, \emptyset, \{v \mapsto K\}, \emptyset) : \emptyset \to \emptyset \\ H &= (\{v, v'\}, \emptyset, \{v \mapsto K, v' \mapsto K\}, \emptyset) : \emptyset \to \emptyset \\ ar(K) &= 0 \end{aligned}$$

Then $\phi = \mathsf{Id}_{\{v\}} : G \hookrightarrow H$ is a link graph embedding and there are two different decompositions of $H$ that include $\phi \centerdot G$: $H = C \circ \phi \centerdot G \circ D = D \circ \phi \centerdot G \circ C$ where

$$\begin{aligned} C &= (\emptyset, \emptyset, \emptyset, \emptyset) : \emptyset \to \emptyset \\ D &= (\{v'\}, \emptyset, \{v' \mapsto K\}, \emptyset) : \emptyset \to \emptyset \end{aligned}$$

□

Though one could perhaps recover the correspondence by restricting to some canonical contexts, we shall not pursue this here, as we shall recover the correspondence once we combine link and place graph embeddings.

### Solid Link Graphs

For an important class of link graphs, those that are *solid*, embeddings are determined by the injections of nodes:

**Definition 7.5.2** (solid link graph (after [25, Def. 2.1]))**.** A link graph is *solid* iff these conditions hold:

1. no links are idle
2. no inner names are siblings
3. every inner name is guarding
4. no outer name is linked to an inner name.

□

The notion of solidness comes from stochastic bigraphs [25] where redexes are required to be solid, which essentially ensures that a match is determined by the support translation. We have strengthened the condition in two respects, which enables us to obtain a stronger and more general result without diminishing the set of solid redexes: (a) we preclude idle edges and (b) we require inner names to be guarding. To see that these condition do not rule out any redexes, remember that (a) we may simple choose concretions of the abstract redexes with no idle edges, and (b) that redexes have no inner names and thus 3. is vacuously satisfied.

The conditions ensure that an embedding and its context and parameters are determined by just the support translation of the nodes:

**Proposition 7.5.3** (solid link graph embeddings). *Given a solid link graph $G : X_G \to Y_G$ and an embedding $\phi : G \hookrightarrow H$ into a link graph $H : X_H \to Y_H$. Then $\phi^{\mathsf{e}}$, $\phi^{\mathsf{i}}$, and $\phi^{\mathsf{o}}$ are uniquely determined from $\phi^{\mathsf{v}}$.*

*Proof.* Here we give only the constructions of $\phi^{\mathsf{e}}$, $\phi^{\mathsf{i}}$ and $\phi^{\mathsf{o}}$. Proofs that they are unique and satisfy the embedding conditions may be found in Appendix 7.A.1.

$\phi^{\mathsf{e}}$: Construct the map of each edge $e \in E_G$ as follows: choose a port $p = (v, i) \in link_G^{-1}(e)$, which is always possible since no edge is idle and every inner name is guarding, and let

$$\phi^{\mathsf{e}}(e) = link_H(\phi^{\mathsf{v}}(v), i).$$

$\phi^{\mathsf{i}}$: Construct the map of each inner name $x \in X_G$ as follows:

$$\begin{aligned} \phi^{\mathsf{i}}(x) &= points_{H,x} \setminus \phi^{\mathsf{p}}(P_{G,x}) \\ points_{H,x} &= (link_H^{-1} \circ \phi^{\mathsf{e}})(link_G(x)) \\ P_{G,x} &= (link_G^{-1} \circ link_G)(x) \setminus \{x\} \\ \phi^{\mathsf{p}}(v,i) &= (\phi^{\mathsf{v}}(v), i). \end{aligned}$$

$\phi^{\mathsf{o}}$: Construct the map of each outer name $y \in Y_G$ as follows: choose a port $p = (v, i) \in link_G^{-1}(y)$, which is always possible since no outer name is idle or connected to an inner name, and let

$$\phi^{\mathsf{o}}(y) = link_H(\phi^{\mathsf{v}}(v), i).$$

□

### 7.5.2 Place Graph Embeddings

As for link graph embeddings, place graph embeddings are simply support translations along with maps of the interfaces:

**Definition 7.5.4** (place graph embedding). Let $G : k_G \to m_G, H : k_H \to m_H$ be two concrete place graphs. Then a *place graph embedding*, written $\phi : G \hookrightarrow H$, is a map $\phi : |G| \uplus k_G \uplus m_G \to |H| \uplus \mathcal{P}(k_H \uplus V_H) \uplus m_H$, where $\phi = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}}$ satisfies the following conditions:

**maps:**

(PGE-1) $\phi^{\mathsf{v}} : V_G \rightarrowtail V_H$ is an injective map

(PGE-2) $\phi^{\mathsf{s}} : k_G \to \mathcal{P}(k_H \uplus V_H)$ is a fully injective map

(PGE-3) $\phi^{\mathsf{r}} : m_G \to V_H \uplus m_H$ is an arbitrary map

**injectivity:**

(PGE-4) $\mathsf{rng}(\phi^{\mathsf{v}}) \# \mathsf{rng}(\phi^{\mathsf{r}})$

(PGE-5) $\mathsf{rng}(\phi^{\mathsf{s}}) \# \mathsf{rng}(\phi^{\mathsf{v}})$
(PGE-6) $H \lfloor^{\mathsf{rng}(\phi^{\mathsf{s}})} \# \mathsf{rng}(\phi^{\mathsf{r}})$

**surjective on node children:**

(PGE-7) $\phi^{\mathsf{c}} \circ prnt_G^{-1} \upharpoonright_{V_G} = prnt_H^{-1} \circ \phi^{\mathsf{v}}$

**structure preservation:**

(PGE-8) $ctrl_G = ctrl_H \circ \phi^{\mathsf{v}}$
(PGE-9) $\forall c \in k_G \uplus V_G : \forall c' \in \phi^{\mathsf{c}}(c) : (\phi^{\mathsf{f}} \circ prnt_G)(c) = prnt_H(c')$

where

$$\phi^{\mathsf{f}} = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{r}} \qquad \text{(map of parents)}$$
$$\phi^{\mathsf{c}} = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \qquad \text{(map of children).}$$

We do not take the codomain of $\phi$ as part of its definition, and thus it may be an embedding into several place graphs. We write $\phi \blacksquare G$ when applying the underlying support translation to $G$. If any of the maps are partial, $\phi$ is *partial*, written $\phi : G \hookrightarrow H$. Partial embeddings need only satisfy the conditions where they are defined; in particular the surjectivity condition only applies to a node $v$ iff $\phi^{\mathsf{v}}$ is defined for $v$ and $\phi^{\mathsf{c}}$ is defined for $prnt_G^{-1}(v)$. A partial embedding is said to be *non-trivial* iff its range is non-empty. □

The conditions are analogous to those for link graph embeddings, except condition (PGE-6) which deserves an explanation. Let us first motivate it by an example:

**Example 3.** Consider the following place graphs

$$G = (\emptyset, \emptyset, \{0 \mapsto 0, 1 \mapsto 1\}) : 2 \to 2$$
$$H = (\{v, v'\}, \{v \mapsto K, v' \mapsto K\}, \{v \mapsto 0, v' \mapsto v\}) : 1$$

Then the following would be a place graph embedding if we did not include condition (PGE-6):

$$\phi = \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} : G \hookrightarrow H$$
$$\phi^{\mathsf{r}} = \{0 \mapsto 0, 1 \mapsto v'\}$$
$$\phi^{\mathsf{s}} = \{0 \mapsto \{v\}, 1 \mapsto \emptyset\}$$

But there are no place graphs $C$ and $D$ such that $H = C \circ \phi \blacksquare G \circ D$! The problem is that root 1 of $G$ is mapped to node $v'$ which is part of the tree that site 0 is mapped to. □

The issue is that there are no place graph operations that can make one root of a place graph a descendant of one of its other roots. In other words, roots do not just model possibly disjoint locations, but subtrees that are disjoint. This is a design choice in the bigraphical model, and it is out of scope for this report to investigate the consequences of relaxing this restriction. Thus, for embeddings to correspond to decompositions, we need to rule out embeddings where one root is mapped to a descendant of another, hence condition (PGE-6).

The decompositions that we can express with an embedding are the following:

**Definition 7.5.5** (embedding corresponding to decomposition)**.** Given a place graph decomposition

$$H = C \circ (G \circ D \otimes \mathsf{id}_k) \circ \pi.$$

Then the *corresponding* embedding $\phi = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} : G \hookrightarrow H$ is defined by

$$\phi^{\mathsf{v}} = \mathsf{Id}_{V_G} \qquad \phi^{\mathsf{r}} = prnt_C \upharpoonright_{m_G} \qquad \phi^{\mathsf{s}} = (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \upharpoonright_{k_G} .$$

□

**Proposition 7.5.6** (embedding corresponding to decomposition)**.** *The embedding $\phi : G \hookrightarrow H$ of Def. 7.5.5 is indeed an embedding.*

*Proof.* Cf. Appendix 7.A.1. □

Note that in the case where $k = 0$ and $\pi = \mathsf{id}$, the decomposition becomes $H = C \circ G \circ D$, demonstrating that embeddings are indeed just decompositions into context, redex, and parameter. The identity $\mathsf{id}_k$ allows some of the sites of $H$ to be in the context $C$. The permutation $\pi$ is a technical measure to handle the fact that sites are not names but consecutive numbers: it expresses that the sites of $H$ may belong to either the context or the parameter, and in a decomposition we have to partition and renumber them accordingly.

Let us make this precise, by defining when we consider decompositions equivalent:

**Definition 7.5.7** (decomposition equivalence)**.** Say that two decompositions

$$\begin{aligned} H &= C \circ (G \circ D \otimes \mathsf{id}_k) \circ \pi \\ &= C' \circ (G \circ D' \otimes \mathsf{id}_k) \circ \pi' \end{aligned}$$

are the same iff they differ only on their internal numbering of sites, i.e.,

$$\begin{aligned} V_D &= V_{D'} \\ V_C &= V_{C'} \\ prnt_D \restriction_{V_D} &= prnt_{D'} \restriction_{V_D} \\ prnt_C \restriction_{V_C \uplus m_G} &= prnt_{C'} \restriction_{V_C \uplus m_G} \\ prnt_D \circ \pi \downharpoonleft^{k_D} &= prnt_{D'} \circ \pi' \downharpoonleft^{k_D} \\ prnt_C(\pi(i) - k_D + m_G) &= prnt_{C'}(\pi'(i) - k_D + m_G) \qquad (i \in \pi^{-1}(k_H \setminus k_D)). \end{aligned}$$

□

Let us now turn to showing that embeddings can only express such decompositions:

**Definition 7.5.8** (decomposition corresponding to embedding)**.** Given a place graph $G : k_G \to m_G$ and an embedding $\phi : G \hookrightarrow H$ into a place graph $H : k_H \to m_H$. Then the *corresponding* decomposition into *parameter* $prmt(\phi)$ and *context* $ctxt(\phi)$ place graphs are as defined in Figure 7.3. □

**Proposition 7.5.9** (embeddings are decompositions)**.** *Given a place graph $G : k_G \to m_G$ and an embedding $\phi : G \hookrightarrow H$ into a place graph $H : k_H \to m_H$. Then construction Def. 7.5.8 defines a decomposition up to decomposition equivalence.*

*Proof.* Cf. Appendix 7.A.1. □

The bijections $f_D$ and $f'_C$ are the realizations of the internal partitioning and renumbering of sites that we discussed above and they express the variation within decomposition equivalence classes. Note that the support translation of $G$ slightly muddles the correspondence with the decomposition of Def. 7.5.5 above. However, letting $F \stackrel{\text{def}}{=} \phi \bullet G$ we see that an embedding indeed specifies such a decomposition.

Together, the above constructions form an isomorphism between embeddings and decompositions:

**Theorem 7.5.10** (embeddings and decompositions are isomorphic)**.** *The constructions of Def. 7.5.5 and Def. 7.5.8 are mutually inverse.*

*Proof.* Cf. Appendix 7.A.1. □

$$
\begin{aligned}
\mathit{prmt}(\phi) &\stackrel{\text{def}}{=} (V_D, \mathit{ctrl}_H \upharpoonright_{V_D}, \mathit{prnt}_D) : k_D \to k_G \quad \text{where} \\
V_D &= V_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \\
\tilde{k}_D &= k_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \\
k_D &= |\tilde{k}_D| \\
f_D &: k_D \rightarrowtail \tilde{k}_D \quad \text{a bijection} \\
\mathit{prnt}_D &= ((\phi^{\mathsf{s}})^{-1} \uplus \mathit{prnt}_H \upharpoonright_{(V_D \uplus \tilde{k}_D) \setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{ctxt}(\phi) &\stackrel{\text{def}}{=} (V_C, \mathit{ctrl}_H \upharpoonright_{V_C}, \mathit{prnt}_C) : k_C \to m_H \quad \text{where} \\
V_C &= (V_H \setminus \phi^{\mathsf{v}}(V_G)) \setminus V_D \\
\tilde{k}_C &= k_H \setminus \tilde{k}_D \\
k_C &= m_G + |\tilde{k}_C| \\
f'_C &: |\tilde{k}_C| \rightarrowtail \tilde{k}_C \quad \text{a bijection} \\
f_C(i + m_G) &= f'_C(i) \quad \text{for } i \in |\tilde{k}_C| \\
\mathit{prnt}_C &= \phi^{\mathsf{r}} \uplus \mathit{prnt}_H \upharpoonright_{V_C} \uplus \mathit{prnt}_H \circ f_C \\
f'(i + k_D) &= f'_C(i) \quad \text{for } i \in |\tilde{k}_C| \\
\pi &= f_D^{-1} \uplus f'^{-1} : k_H \to k_H
\end{aligned}
$$

$$
H = \mathit{ctxt}(\phi) \circ (\phi \centerdot G \circ \mathit{prmt}(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|}) \circ \pi
$$

Figure 7.3: Decomposition of place graph $H : k_H \to m_H$ into parameter $\mathit{prmt}(\phi)$ and context $\mathit{ctxt}(\phi)$ corresponding to an embedding $\phi : G \hookrightarrow H$ of a place graph $G : k_G \to m_G$.

When we get to edit scripts, we shall need a number of disjointness results in addition to the injectivity conditions:

**Lemma 7.5.11.** *Given a place graph $G : k_G \to m_G$ and an embedding $\phi : G \hookrightarrow H$ into a place graph $H : k_H \to m_H$. Then*

1. $\mathsf{rng}(\phi^{\mathsf{f}}) \# H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}$,
2. $\mathsf{rng}(\phi^{\mathsf{c}}) \# H \upharpoonleft_{\mathsf{rng}(\phi^{\mathsf{r}})}$,
3. $H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \# H \upharpoonleft_{\mathsf{rng}(\phi^{\mathsf{r}})}$, *and*
4. $\forall i \in k_G : \mathsf{rng}(\phi^{\mathsf{c}}) \# (H \downharpoonleft^{\phi^{\mathsf{s}}(i)} \setminus \phi^{\mathsf{s}}(i))$.

*Proof.* Cf. Appendix 7.A.1. □

### Solid Place Graphs

As was the case with link graph embeddings, place graph embeddings are determined by the injection of nodes *iff* the place graph is solid:

**Definition 7.5.12** (solid place graph (after [25, Def. 2.1])). A place graph is *solid* iff these conditions hold:

1. no roots are idle
2. no sites are siblings
3. every site is guarding .

□

**Proposition 7.5.13.** *Given a solid place graph $G : k_G \to m_G$ and an embedding $\phi : G \hookrightarrow H$ into a place graph $H : k_H \to m_H$. Then $\phi^{\mathsf{s}}$ and $\phi^{\mathsf{r}}$ are uniquely determined from $\phi^{\mathsf{v}}$.*

*Proof.* Here we give only the constructions of $\phi^{\mathsf{s}}$ and $\phi^{\mathsf{r}}$. Proofs that they are unique and satisfy the embedding conditions may be found in Appendix 7.A.1.

$\phi^{\mathsf{s}}$: Construct the map of each site $i \in k_G$ as follows:

$$\begin{aligned}
\phi^{\mathsf{s}}(i) &= \mathit{children}_{H,i} \setminus \phi^{\mathsf{v}}(\mathit{siblings}_{G,i}) \\
\mathit{children}_{H,i} &= (\mathit{prnt}_H^{-1} \circ \phi^{\mathsf{v}})(\mathit{prnt}_G(i)) \\
\mathit{siblings}_{G,i} &= (\mathit{prnt}_G^{-1} \circ \mathit{prnt}_G)(i) \setminus \{i\}.
\end{aligned}$$

$\phi^{\mathsf{r}}$: Construct the map of each root $j \in m_G$ as follows: choose a node $v \in \mathit{prnt}_G^{-1}(j)$, which is always possible since no root is idle or has a site as a child, and let

$$\phi^{\mathsf{r}}(j) = (\mathit{prnt}_H \circ \phi^{\mathsf{v}})(v).$$

□

### 7.5.3 Bigraph Embeddings

Having defined embeddings for each of the two constituent graphs, we can now define embeddings of bigraphs as the combination of the two, adding only a single condition:

**Definition 7.5.14** (bigraph embedding)**.** Let $G : \langle k_G, X_G\rangle \to \langle m_G, Y_G\rangle$, $H : \langle k_H, X_H\rangle \to \langle m_H, Y_H\rangle$ be two concrete bigraphs. Then a *bigraph embedding*, written $\phi : G \hookrightarrow H$, is a map $\phi : |G| \uplus k_G \uplus m_G \uplus X_G \uplus Y_G \to |H| \uplus \mathcal{P}(k_H \uplus V_H) \uplus m_H \uplus \mathcal{P}(X_H \uplus P_H) \uplus Y_H$, where $\phi^P = \phi\restriction_{V_G \uplus k_G \uplus m_G} : G^P \hookrightarrow H^P$ is place graph embedding and $\phi^L = \phi\restriction_{|G| \uplus X_G \uplus Y_G} : G^L \hookrightarrow H^L$ is a link graph embedding. Furthermore, the map must satisfy the following condition:

**consistency:**

(BGE-1) $\mathsf{rng}(\phi^{\mathsf{i}}) \subseteq X_H \uplus P_{H\downharpoonright^{\mathsf{rng}(\phi^{\mathsf{s}})\cap V_H}}$ .

We do not take the codomain of $\phi$ as part of its definition, and thus it may be an embedding into several bigraphs. We write $\phi \bullet G$ when applying the underlying support translation to $G$. If $\phi^{\mathsf{P}}$ or $\phi^{\mathsf{L}}$ are partial, $\phi$ is *partial*, written $\phi : G \hookrightarrow H$; if either is *non-trivial*, so is $\phi$. □

The consistency condition ensures that the link graph embedding only maps inner names to ports on nodes that are in the place graph parameter. If we did not have this condition, the link and place graph embeddings might disagree on whether a node belongs to the context or the parameter.

We saw in the previous section that the place graph structure gives us a unique way to separate the nodes that are not in the image of the embedding into a context and parameter. Link graphs have less structure and for a given embedding there may be several ways to decompose the link graph (e.g., different ways to partition the edges between context and parameter). Depending on the definition of reaction, this may affect the reaction relation. For pure bigraphs, Milner resolves this issue by disallowing inner names in redexes and requiring parameters to be discrete [29, Def. 8.5]. For binding bigraphs, inner names are allowed in redexes as long as they are local and the definition of discreteness is conservatively extended to exempt bound links [24, Sec. 11]. We shall adapt (a variant of) the latter approach in order to avoid restricting redexes and to make our work extensible to binding bigraphs: we shall require the parameter to be discrete *except* that (1) we shall discard the bijection constraint for the links that connect to the redex, and (2) inner names can only connect to the redex. We call this *semi-discreteness*:

**Definition 7.5.15** (semi-discrete bigraph)**.** A bigraph $D : \langle X_D, k_D\rangle \to \langle X_G \uplus X_I, m_D\rangle$ is *semi-discrete on* $X_G$ iff it has no edges, no outer name is idle, $\mathit{link}_D \downharpoonright^{X_I}$ is a bijection, and $\mathit{link}_D(X_D) \subseteq X_G$. □

Thus the decompositions that our embeddings correspond to are the following:

**Definition 7.5.16** (embedding corresponding to decomposition)**.** Given a bigraph

$$H = C \circ ((G \otimes \mathsf{id}_{X_I}) \circ D \otimes \mathsf{id}_k \otimes \alpha) \circ (\pi \otimes \mathsf{id}_{X_H})$$

where $D$ is semi-discrete on $X_G$. Then the *corresponding* embedding $\phi = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{e}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} \uplus \phi^{\mathsf{i}} \uplus \phi^{\mathsf{o}} : G \hookrightarrow H$ is defined by

$$\phi^{\mathsf{v}} = \mathsf{Id}_{V_G} \qquad \phi^{\mathsf{r}} = \mathit{prnt}_C\restriction_{m_G} \qquad \phi^{\mathsf{s}} = (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ \mathit{prnt}_D^{-1}\restriction_{k_G}$$

$$\phi^{\mathsf{e}} = \mathsf{Id}_{E_G} \qquad \phi^{\mathsf{o}} = \mathit{link}_C\restriction_{Y_G} \qquad \phi^{\mathsf{i}} = \mathit{link}_D^{-1}\restriction_{X_G} .$$

□

**Proposition 7.5.17** (embedding corresponding to decomposition)**.** *The embedding $\phi : G \hookrightarrow H$ of Def. 7.5.16 is indeed an embedding.*

*Proof.* Cf. Appendix 7.A.1. □

As we discussed for place graph embeddings, $\mathsf{id}_k$ allows sites of $H$ to be in the context $C$ and $\pi$ is a technical artifact reflecting that sites are consecutive numbers and not names. Similarly, the renaming $\alpha$ allows inner names of $H$ to be in the context $C$, though suitably renamed to handle the case where inner names of $H$ collide with the outer names of $G$. If $\mathsf{id}_k = \alpha = \mathsf{id}_\epsilon$ and $\pi$ is an identity, the decomposition becomes $H = C \circ (G \otimes \mathsf{id}_{X_I}) \circ D$, again demonstrating that embeddings are just decompositions into context, redex, and parameter. The identity $\mathsf{id}_{X_I}$ expresses the fact that we allow the parameter and context to share links without the involvement of the redex; in this sense the exact choice of $X_I$ is internal to the decomposition.

Let us extend our definition of decomposition equivalence to disregard the internal names:

**Definition 7.5.18** (decomposition equivalence)**.** Say that two decompositions

$$\begin{aligned} H &= C \circ ((G \otimes \mathsf{id}_{X_I}) \circ D \otimes \mathsf{id}_k \otimes \alpha) \circ (\pi \otimes \mathsf{id}_{X_H}) \\ &= C' \circ ((G \otimes \mathsf{id}_{X_{I'}}) \circ D' \otimes \mathsf{id}_k \otimes \alpha') \circ (\pi' \otimes \mathsf{id}_{X_H}) \end{aligned}$$

with $D$,$D'$ discrete, are the same iff the place graph decompositions are the same and the link graph decompositions differ only in their internal names, i.e.,

$$\begin{aligned} E_D &= E_{D'} & \mathit{link}_D \downharpoonleft^{X_G} &= \mathit{link}_{D'} \downharpoonleft^{X_G} \\ E_C &= E_{C'} & \mathit{link}_C \upharpoonright_{P_C \uplus Y_G} &= \mathit{link}_{C'} \upharpoonright_{P_C \uplus Y_G} \\ & & \mathit{link}_C \circ \alpha &= \mathit{link}_{C'} \circ \alpha' \\ & & \mathit{link}_C \circ \mathit{link}_D \downharpoonleft^{X_I} &= \mathit{link}_{C'} \circ \mathit{link}_{D'} \downharpoonleft^{X_{I'}} . \end{aligned}$$

□

In the case of bigraph matches, i.e., Def. 7.4.31 instantiated to BPRSs, this definition captures exactly what it means for matches to be the same:

**Proposition 7.5.19** (matches are decompositions)**.** *Two matches in an agent are the same iff they are equivalent decompositions.*

*Proof.* Cf. Appendix 7.A.1. □

As we did for place graphs, we shall now prove that embeddings can only express such decompositions, by showing how to construct them from an embedding:

**Definition 7.5.20** (decomposition corresponding to embedding)**.** Given a bigraph $G : \langle k_G, X_G \rangle \to \langle m_G, Y_G \rangle$ and an embedding $\phi : G \hookrightarrow H$ into a bigraph $H : \langle k_H, X_H \rangle \to \langle m_H, Y_H \rangle$. Then the *corresponding* decomposition into *parameter* $\mathit{prmt}(\phi)$ and *context* $\mathit{ctxt}(\phi)$ bigraphs are as defined in Figure 7.4. □

**Proposition 7.5.21** (embeddings are decompositions)**.** *Given a bigraph* $G : \langle k_G, X_G \rangle \to \langle m_G, Y_G \rangle$ *and an embedding* $\phi : G \hookrightarrow H$ *into a bigraph* $H : \langle k_H, X_H \rangle \to \langle m_H, Y_H \rangle$*. Then constructions Def. 7.5.8 and Def. 7.5.20 define a decomposition up to decomposition equivalence.*

*Proof.* Cf. Appendix 7.A.1. □

We discussed the place graph aspects of the construction in Section 7.5.2. The bijections $\mathit{link}'_D$ and $\alpha_C$ are the realizations of the choice of suitable internal names as discussed above and they express the variation within decomposition equivalence classes.

Let us now prove that these constructions form an isomorphism between embeddings and decompositions:

$$
\begin{aligned}
prmt(\phi) &\stackrel{\text{def}}{=} (V_D, \emptyset, ctrl_D, prnt_D, link_D) : \langle k_D, X_D\rangle \to \langle k_G, X_G \uplus X_I\rangle \quad \text{where} \\
P'_D &= P_D \setminus \mathsf{rng}(\phi^{\mathsf{i}}) \\
X_D &= \mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H \\
X_I &: \text{ a set of names satisfying} \\
&\qquad |X_I| = |P'_D|,\ X_I \# X_G, \text{ and } X_I \# Y_G \\
link'_D &: P'_D \rightarrowtail X_I \text{ a bijection} \\
link_D &= (\phi^{\mathsf{i}})^{-1} \uplus link'_D
\end{aligned}
$$

$$
\begin{aligned}
ctxt(\phi) &\stackrel{\text{def}}{=} (V_C, E_C, ctrl_C, prnt_C, link_C) : \langle k_C, Y_G \uplus X_I \uplus X_C\rangle \to \langle m_H, Y_H\rangle \quad \text{where} \\
E_C &= E_H \setminus \mathsf{rng}(\phi^{\mathsf{e}}) \\
X'_C &= X_H \setminus X_D \\
X_C &: \text{ a set of names satisfying} \\
&\qquad |X_C| = |X'_C|,\ X_C \# Y_G, \text{ and } X_C \# X_I \\
\alpha_C &: X_C \rightarrowtail X'_C \text{ a bijection} \\
link_C &= \phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)
\end{aligned}
$$

$$
H = ctxt(\phi) \circ ((\phi \blacksquare G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|} \otimes \alpha_C^{-1}) \circ (\pi \otimes \mathsf{id}_{X_H})
$$

Figure 7.4: Decomposition of bigraph $H : \langle k_H, X_H\rangle \to \langle m_H, Y_H\rangle$ into parameter $prmt(\phi)$ and context $ctxt(\phi)$ corresponding to an embedding $\phi : G \hookrightarrow H$ of a place graph $G : \langle k_G, X_G\rangle \to \langle m_G, Y_G\rangle$. The decomposition of the place graph is given in Figure 7.3.

**Theorem 7.5.22** (embeddings and decompositions are isomorphic)**.** *The constructions of Def. 7.5.16 and Def. 7.5.20 are mutually inverse.*

*Proof.* Cf. Appendix 7.A.1. □

As an instance of this result we get that redex embeddings into agents are isomorphic to matches:

**Corollary 7.5.23** (matches isomorphic to redex embeddings into agents)**.** *In a BPRS* $`\mathrm{BG}(`\mathcal{R})$*, a match* $o = (\rho, \mathsf{id}_{X_I}, c, d)$ *of a parametric rule* $\mathsf{R} = (R : m \to \langle n, Y\rangle, R', \eta)$ *in an agent* $a$ *is isomorphic to the embedding*

$$\phi = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{e}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} \uplus \phi^{\mathsf{i}} \uplus \phi^{\mathsf{o}} : R \hookrightarrow a$$

$$\phi^{\mathsf{v}} = \rho \restriction_{V_R} \qquad \phi^{\mathsf{r}} = prnt_c \restriction_n \qquad \phi^{\mathsf{s}} = prnt_d^{-1} \restriction_m$$

$$\phi^{\mathsf{e}} = \rho \restriction_{E_R} \qquad \phi^{\mathsf{o}} = link_c \restriction_Y \qquad \phi^{\mathsf{i}} = \emptyset.$$

*Proof.* Follows immediately from Prop. 7.5.19 and Theorem 7.5.22. □

The disjointness results for place graph embeddings extend to bigraph embeddings; we shall need them in Section 7.6.

**Corollary 7.5.24.** *Given a bigraph* $G : \langle k_G, X_G\rangle \to \langle m_G, Y_G\rangle$ *and an embedding* $\phi : G \hookrightarrow H$ *into a bigraph* $H : \langle k_H, X_H\rangle \to \langle m_H, Y_H\rangle$*. Then*

1. $\mathsf{rng}(\phi^{\mathsf{port}}) \# P_{H\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}}$ *and*
2. $\mathsf{rng}(\phi^{\mathsf{port}}) \# P_{H\restriction_{\mathsf{rng}(\phi^{\mathsf{r}})}}$*.*

*Proof.* 1: From Lemma 7.5.11 we have $\mathsf{rng}(\phi^{\mathsf{v}}) \# H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}$ so clearly $\mathsf{rng}(\phi^{\mathsf{port}}) \# P_{H\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}}$.

2: From Lemma 7.5.11 we have $\mathsf{rng}(\phi^{\mathsf{v}}) \# H \restriction_{\mathsf{rng}(\phi^{\mathsf{r}})}$ so clearly $\mathsf{rng}(\phi^{\mathsf{port}}) \# P_{H\restriction_{\mathsf{rng}(\phi^{\mathsf{r}})}}$. □

### Solid Bigraphs

The results regarding solid link and place graphs of course also hold for bigraphs, i.e., embeddings of *solid* bigraphs are determined by the injection of nodes:

**Definition 7.5.25** (solid bigraph (after [25, Def. 2.1][7]))**.** A bigraph is *solid* iff these conditions hold:

1. no roots or links are idle
2. no sites or inner names are siblings
3. every site and inner name is guarding
4. no outer name is linked to an inner name .

□

**Corollary 7.5.26.** *Given a solid bigraph* $G : \langle k_G, X_G\rangle \to \langle m_G, Y_G\rangle$ *and an embedding* $\phi : G \hookrightarrow H$ *into a bigraph* $H : \langle k_H, X_H\rangle \to \langle m_H, Y_H\rangle$*. Then* $\phi^{\mathsf{s}}$*,* $\phi^{\mathsf{r}}$*,* $\phi^{\mathsf{e}}$*,* $\phi^{\mathsf{i}}$*, and* $\phi^{\mathsf{o}}$ *are uniquely determined from* $\phi^{\mathsf{v}}$*.*

*Proof.* Follows from Prop. 7.5.3 and Prop. 7.5.13. □

[7] These conditions are slightly stronger than those in loc. cit. cf. Sec. 7.5.1.

## 7.6 Bigraph Edit Scripts

As we have seen in the previous sections, bigraphical reactions are usually defined in terms of replacement: rewriting is performed by replacing a redex with a reactum. While this yields a simple and elegant presentation of reaction semantics, it does not capture the relation between entities in the redex and reactum, which is needed in the KaSim algorithm: we require a description of what is *modified* by a reaction rule, which implies a relation between entities before and after reaction.

In this section, we shall develop an alternative formulation of bigraphical reaction based on reconfiguration instead of replacement. The key ideas are:

**reconfiguration rules:**
*Reconfiguration rules* are fine-grained descriptions of how a reaction modifies the redex. They consist of a redex and an *edit script*: a series of minimal modifications, *edits*, to the redex which turn it into the reactum.

**reaction as reconfiguration:**
Exploiting that matches and embeddings are isomorphic, we define reaction as the mediation of edits to agents through embeddings.

This formulation is equivalent to the usual formulation in that it generates the same abstract reactions, but in addition it provides the notion of modification that is needed for the KaSim algorithm: edit scripts allow us to characterize causation and conflict in a fine-grained and concise way, as we already saw in the overview of KaSim (cf. Section 7.3.2).

In overview, the development proceeds as follows:

**Section 7.6.1: Patterns**
To simplify the development, we first introduce an alternative formulation of concrete bigraphs, where roots and sites are named instead of being consecutive numbers. We shall call these *patterns* to avoid confusion with the usual concrete bigraphs.

We also recast BPRSs and bigraph embeddings to this setting where redexes and reactums are patterns.

**Section 7.6.2: Edits**
We introduce a set of minimal *edits* and define how they reconfigure *compatible* redexes, i.e., redexes that have a suitable structure for the edit to be meaningful.

Next, we show how we can extract an instantiation map from an edit, which is necessary in order to relate edits to reaction rules.

Finally, we transfer edits to agents, by defining how an embedding of a redex can *mediate* an edit to the agent, and show that such mediated edits correspond to abstract reactions in certain BPRSs.

**Section 7.6.3: Edit Scripts**
Reaction rules cannot in general be expressed as a single edit, but require a sequence of edits, i.e., an *edit script*. We show how the concepts and results for edits transfer to such edit scripts.

**Section 7.6.4: Reconfiguration Systems**
Putting the above developments together, we define *reconfiguration rules* and *reconfiguration systems (RCSs)*. We give constructions between RCSs and BPRSs that preserve and reflect abstract reactions.

### 7.6.1 Patterns

A pattern is an alternative representation of a concrete bigraph where roots and sites are named. To avoid confusion with the names of the link graph, we shall call these identifiers *variables.* We shall assume a countably infinite set $\mathcal{U}$ of variables, ranged over by $q, r \in Q, R$ and disjoint from $\mathcal{X}$, $\mathcal{V}$, and $\mathcal{E}$.

**Definition 7.6.1** (pattern)**.** A *pattern*

$$\tilde{P} = (V_{\tilde{P}}, E_{\tilde{P}}, \mathit{ctrl}_{\tilde{P}}, \mathit{prnt}_{\tilde{P}}, \mathit{link}_{\tilde{P}}) : \langle Q, X \rangle \to \langle R, Y \rangle$$

is an alternative representation of the concrete bigraph

$$[\![\tilde{P}]\!] \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, \mathit{ctrl}_{\tilde{P}}, \mathit{prnt}_{[\![\tilde{P}]\!]}, \mathit{link}_{\tilde{P}}) : \langle |Q|, X \rangle \to \langle |R|, Y \rangle$$

where

$$\begin{aligned}
\mathit{prnt}_{\tilde{P}} &: Q \uplus V_{\tilde{P}} \to V_{\tilde{P}} \uplus R, \\
Q &= \{q_0, \ldots, q_{k-1}\} \qquad \text{where } \forall i \in [0; k-2] : q_i < q_{i+1}, \\
R &= \{r_0, \ldots, r_{m-1}\} \qquad \text{where } \forall i \in [0; m-2] : r_i < r_{i+1}, \\
\mathit{prnt}_{[\![\tilde{P}]\!]} &= (\mathsf{Id}_{V_{\tilde{P}}} \uplus \{r_0 \mapsto 0, \ldots, r_{m-1} \mapsto m-1\}) \\
&\quad \circ \mathit{prnt}_{\tilde{P}} \\
&\quad \circ (\mathsf{Id}_{V_{\tilde{P}}} \uplus \{0 \mapsto q_0, \ldots, k-1 \mapsto q_{k-1}\}).
\end{aligned}$$

□

As we shall see when we define edits, the virtue of patterns is that we can add and remove sites without having to renumber those that remain. Note that there are infinitely many patterns that correspond to a concrete bigraph:

**Proposition 7.6.2.** *Two patterns $\tilde{P}$,$\tilde{P}'$ differ only by order preserving bijections on their outer and inner variables respectively iff* $[\![\tilde{P}]\!] = [\![\tilde{P}']\!]$.

*Proof.* Immediate from the definition. □

Parametric reaction rules, BPRSs, and bigraph embeddings are easily adapted to patterns:

**Definition 7.6.3** (pattern rule)**.** A *pattern rule*

$$\mathsf{R} = (\tilde{P} : Q \to \langle R, Y \rangle, \tilde{P}' : Q' \to \langle R, Y \rangle, \eta : Q' \to Q)$$

where $\eta$ is a function called the *variable instance map*, is an alternative representation of the parametric bigraphical reaction rule

$$[\![\mathsf{R}]\!] \stackrel{\text{def}}{=} ([\![\tilde{P}]\!] : |Q| \to \langle |R|, Y \rangle, [\![\tilde{P}']\!] : |Q'| \to \langle |R|, Y \rangle, [\![\eta]\!] : |Q'| \to |Q|)$$

where

$$\begin{aligned}
Q &= \{q_0, \ldots, q_{k-1}\} && \text{where } \forall i \in [0; k-2] : q_i < q_{i+1}, \\
Q' &= \{q'_0, \ldots, q'_{m-1}\} && \text{where } \forall i \in [0; m-2] : q'_i < q'_{i+1}, \\
[\![\eta]\!](i) &= j && \text{if } \eta(q'_i) = q_j.
\end{aligned}$$

□

**Definition 7.6.4** (pattern-based BPRS)**.** A *pattern-based BPRS* over $\mathcal{K}$ with pattern rules $\grave{}\mathcal{R}$, written $\grave{}\mathrm{B}\small{\mathrm{G}}(\mathcal{K}, \grave{}\mathcal{R})$, is the BPRS $\grave{}\mathrm{B}\small{\mathrm{G}}(\mathcal{K}, \grave{}\mathcal{R}')$ where $\grave{}\mathcal{R}' = \{[\![\mathsf{R}]\!] \mid \mathsf{R} \in \grave{}\mathcal{R}\}$. □

| | | Affected Entity Type | | | |
|---|---|---|---|---|---|
| | | PORT | NODE | EDGE | SITE |
| Effect | REBIND | $\odot_{(v,i)\mapsto l}$ | | | |
| | CHANGE CONTROL | | $\circledcirc_{v:K}$ | | |
| | ADD | | $\oplus_{v:K_{\vec{y}}@p}$ | $\oplus_e$ | |
| | DELETE | | $\ominus_v$ | $\ominus_e$ | $\ominus_q$ |
| | MOVE | | $\oslash_{v@p}$ | | $\oslash_{q@p}$ |
| | COPY | | | | $\otimes_{q\to r@p}$ |

Table 7.1: The set of edits organized by their effect (rows) and the type of entity they affect (columns) ($v \in \mathcal{V}$, $K \in \mathcal{K}$, $\{\vec{y}\} \subseteq \mathcal{X}$, $p \in \mathcal{V} \uplus \mathcal{U}$, $e \in \mathcal{E}$, $q, r \in \mathcal{U}$,$i \in \mathbb{N}$, and $l \in \mathcal{E} \uplus \mathcal{X}$).

**Definition 7.6.5** (pattern embedding)**.** Let $\tilde{P} : \langle Q_{\tilde{P}}, X_{\tilde{P}}\rangle \to \langle R_{\tilde{P}}, Y_{\tilde{P}}\rangle$ and $\tilde{H} : \langle Q_{\tilde{H}}, X_{\tilde{H}}\rangle \to \langle R_{\tilde{H}}, Y_{\tilde{H}}\rangle$ be patterns. Then a *pattern embedding*, written $\phi : \tilde{P} \hookrightarrow \tilde{H}$, is a map $\phi : |\tilde{P}| \uplus Q_{\tilde{P}} \uplus R_{\tilde{P}} \uplus X_{\tilde{P}} \uplus Y_{\tilde{P}} \to |\tilde{H}| \uplus \mathcal{P}(Q_{\tilde{H}} \uplus V_{\tilde{H}}) \uplus R_{\tilde{H}} \uplus \mathcal{P}(X_{\tilde{H}} \uplus P_{\tilde{H}}) \uplus Y_{\tilde{H}}$ which is an alternative representation of the bigraph embedding

$$[\![\phi]\!] : [\![\tilde{P}]\!] \hookrightarrow [\![\tilde{H}]\!]$$

where the constituent maps are defined as:

$$\begin{aligned}
[\![\phi]\!]^* &\stackrel{\text{def}}{=} \phi^* && \text{where } * \in \{\mathsf{v}, \mathsf{e}, \mathsf{i}, \mathsf{o}\} \\
[\![\phi]\!]^{\mathsf{s}} &\stackrel{\text{def}}{=} (\mathsf{Id}_{V_{\tilde{H}}} \uplus \{q_{\tilde{H},0} \mapsto 0, \ldots, q_{\tilde{P},k_{\tilde{H}}-1} \mapsto k_{\tilde{H}} - 1\}) \\
&\quad \circ \phi^{\mathsf{s}} \circ \{0 \mapsto q_{\tilde{P},0}, \ldots, k_{\tilde{P}} - 1 \mapsto q_{\tilde{P},k_{\tilde{P}}-1}\} \\
[\![\phi]\!]^{\mathsf{r}} &\stackrel{\text{def}}{=} (\mathsf{Id}_{V_{\tilde{H}}} \uplus \{r_{\tilde{H},0} \mapsto 0, \ldots, r_{\tilde{P},m_{\tilde{H}}-1} \mapsto m_{\tilde{H}} - 1\}) \\
&\quad \phi^{\mathsf{r}} \circ \{0 \mapsto r_{\tilde{P},0}, \ldots, m_{\tilde{P}} - 1 \mapsto r_{\tilde{P},m_{\tilde{P}}-1}\} \\
Q_{\tilde{P}} &= \{q_{\tilde{P},0}, \ldots, q_{\tilde{P},k_{\tilde{P}}-1}\} && \text{where } \forall i \in [0; k_{\tilde{P}} - 2] : q_{\tilde{P},i} < q_{\tilde{P},i+1} \\
R_{\tilde{P}} &= \{r_{\tilde{P},0}, \ldots, r_{\tilde{P},m_{\tilde{P}}-1}\} && \text{where } \forall i \in [0; m_{\tilde{P}} - 2] : r_{\tilde{P},i} < r_{\tilde{P},i+1} \\
Q_{\tilde{H}} &= \{q_{\tilde{H},0}, \ldots, q_{\tilde{H},k_{\tilde{H}}-1}\} && \text{where } \forall i \in [0; k_{\tilde{H}} - 2] : q_{\tilde{H},i} < q_{\tilde{H},i+1} \\
R_{\tilde{H}} &= \{r_{\tilde{H},0}, \ldots, r_{\tilde{H},m_{\tilde{H}}-1}\} && \text{where } \forall i \in [0; m_{\tilde{H}} - 2] : r_{\tilde{H},i} < r_{\tilde{H},i+1}.
\end{aligned}$$

Clearly, it is easy to define embeddings between patterns and concrete bigraphs in a similar manner, and we shall freely use such embeddings. □

### 7.6.2 Edits

An *edit* is a minimal reconfiguration of a pattern. We are concerned with edits that correspond to reactions and shall therefore only consider edits of redexes, i.e., patterns with no inner names, which preserve the outer face.

**Definition 7.6.6** (edits)**.** An *edit* $\delta$ over a signature $\mathcal{K}$ is any of the operators in Table 7.1. The application of an edit $\delta$ to a pattern $\tilde{P}$, written $\delta(\tilde{P})$, is defined in Table 7.2. We say that an edit $\delta$ is *compatible* with a pattern $\tilde{P}$ iff $\delta(\tilde{P})$ is defined. □

It is straightforward to verify that edits yield patterns:

**Rebind a port:**

$$\odot_{(v,i)\mapsto l}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}[(v,i)\mapsto l]) \quad : \quad Q \to \langle R, Y\rangle$$
if $v \in V_{\tilde{P}}$ and $i \in ar(ctrl_{\tilde{P}}(v))$ and $l \in E_{\tilde{P}} \uplus Y$

**Change a control:**

$$\circledcirc_{v:K}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}[v\mapsto K], prnt_{\tilde{P}}, link_{\tilde{P}}) \quad : \quad Q \to \langle R, Y\rangle$$
if $v \in V_{\tilde{P}}$ and $ar(K) = ar(ctrl_{\tilde{P}}(v))$

**Add node or edge:**

$$\oplus_{v:K_{\vec{y}}@p}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}} + v, E_{\tilde{P}}, ctrl_{\tilde{P}}[v\mapsto K], prnt_{\tilde{P}}[v\mapsto p], link_{\tilde{P}}[(v,0)\mapsto \vec{y}_0, \ldots, (v,n-1)\mapsto \vec{y}_{n-1}]) \quad : \quad Q \to \langle R, Y\rangle$$
if $v \notin V_{\tilde{P}}$ and $p \in V_{\tilde{P}} \uplus R$ and $\{\vec{y}\} \subseteq E_{\tilde{P}} \uplus Y$ and $ar(K) = n$

$$\oplus_{e}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}} + e, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}) \quad : \quad Q \to \langle R, Y\rangle$$
if $e \notin E_{\tilde{P}}$

**Delete node, edge, or site:**

$$\ominus_{v}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}} - v, E_{\tilde{P}}, ctrl_{\tilde{P}} - v, prnt_{\tilde{P}} - v, link_{\tilde{P}} - P_v) : \quad Q \to \langle R, Y\rangle$$
if $v \in V_{\tilde{P}}$ and $prnt_{\tilde{P}}^{-1}(v) = \emptyset$

$$\ominus_{e}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}} - e, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}) \quad : \quad Q \to \langle R, Y\rangle$$
if $e \in E_{\tilde{P}}$ and $link_{\tilde{P}}^{-1}(e) = \emptyset$

$$\ominus_{q}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}} - q, link_{\tilde{P}}) \quad : (Q - q) \to \langle R, Y\rangle$$
if $q \in Q$

**Move node or site:**

$$\oslash_{v@p}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[v\mapsto p], link_{\tilde{P}}) \quad : \quad Q \to \langle R, Y\rangle$$
if $v \in V_{\tilde{P}}$ and $p \in V_{\tilde{P}} \uplus R$

$$\oslash_{q@p}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[q\mapsto p], link_{\tilde{P}}) \quad : \quad Q \to \langle R, Y\rangle$$
if $q \in Q$ and $p \in V_{\tilde{P}} \uplus R$

**Copy site:**

$$\otimes_{q\to r@p}(\tilde{P}) \stackrel{\text{def}}{=} (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[r\mapsto p], link_{\tilde{P}}) \quad : (Q + r) \to \langle R, Y\rangle$$
if $q \in Q$ and $r \notin Q$ and $p \in V_{\tilde{P}} \uplus R$

Table 7.2: Application of edit $\delta$ to pattern $\tilde{P} : Q \to \langle R, Y\rangle$, denoted $\delta(\tilde{P})$.

**Proposition 7.6.7** (edits)**.** *Given a pattern $\tilde{P} : Q \to \langle R, Y\rangle$ and a compatible edit $\delta$. Then $\delta(\tilde{P}) : Q' \to \langle R, Y\rangle$ is a pattern and*

$$Q' = \begin{cases} Q - q & \text{if } \delta = \ominus_q \\ Q + r & \text{if } \delta = \otimes_{q \to r@p} \\ Q & \text{otherwise.} \end{cases}$$

Note that other choices of edits are possible. For instance, one could imagine allowing deletion of nodes that have children – but what should happen to the children? Should they also be deleted or should they become children of the deleted node's parent? We have chosen the above set of edits as they express minimal modifications to each of the elements of a concrete bigraph quintuple which result in another concrete bigraph. As we shall see below, the chosen set of edits is sufficient for completeness, so we leave it to future work to explore other sets of edits.

### Deriving Named Instance Maps

To relate edits to reaction rules and reaction, we must define how edits relate to instantiations. There are two important things to take into account when we define the instantiations for edits: (a) edits are not tied to a specific pattern and thus the corresponding instance map should be the identity on all sites except those that are affected by the edit; and (b) edits will be chained together into edit scripts and thus the corresponding instance maps should be easily composable.

We meet these criteria by defining instance maps for edits in two steps:

**forward instance map:** A map which is defined for all compatible patterns and describes how sites are modified by an edit.

**derived instance map:** From the forward map we derive the variable instance maps for specific compatible patterns.

As we shall see when we get to edit scripts, the forward instance maps compose easily and we can reuse the derivation of the pattern specific instance maps.

**Definition 7.6.8** (forward instance map)**.** A *forward instance map* $\mathcal{F} : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ is a fully injective map on named sites.

The forward instance map $\mathit{finst}(\delta)$ corresponding to an edit $\delta$ is

$$\begin{aligned} \mathit{finst}(\ominus_q) &\stackrel{\text{def}}{=} \mathsf{Id}_{\mathcal{U}}[q \mapsto \emptyset] \\ \mathit{finst}(\otimes_{q \to r@a}) &\stackrel{\text{def}}{=} \mathsf{Id}_{\mathcal{U}-r}[q \mapsto \{q, r\}] \\ \mathit{finst}(\delta) &\stackrel{\text{def}}{=} \mathsf{Id}_{\mathcal{U}} \qquad \text{in all other cases} \end{aligned}$$

□

It is clear from this definition and the definition of edits that a forward instance map corresponding to an edit maps the sites of compatible patterns to the sites of the resulting patterns:

**Proposition 7.6.9** (forward instance map)**.** *Given a pattern $\tilde{P} : Q \to J$ and a compatible edit $\delta$. Then $Q \subseteq \mathsf{dom}(\mathit{finst}(\delta))$ and $\mathit{finst}(\delta)(Q) = Q'$ where $\delta(\tilde{P}) : Q' \to J$.*

To derive an instance map for a particular compatible pattern, we simply restrict the domain of the forward instance map and then invert it:

**Definition 7.6.10** (derived variable instance map)**.** The *derived variable instance map* $\mathit{inst}_Q(\mathcal{F})$ corresponding to a forward instance map $\mathcal{F}$ for a set of named sites $Q \subseteq \mathsf{dom}(\mathcal{F})$ is

$$\mathit{inst}_Q(\mathcal{F}) \stackrel{\text{def}}{=} (\mathcal{F} \restriction_Q)^{-1}.$$

Note that the inverse of the forward instance map $\mathcal{F}^{-1}$, and thus $\mathit{inst}_Q(\mathcal{F})$, is a function since $\mathcal{F}$ is fully injective. □

**Corollary 7.6.11** (derived variable instance map)**.** *Given a pattern $\tilde{P} : Q \to J$ and a compatible edit $\delta$. Then $inst_Q(finst(\delta)) : Q' \to Q$ where $\delta(\tilde{P}) : Q' \to J$.*

*Proof.* Follows from Def. 7.6.10 and Prop. 7.6.9. □

### Mediating edits

In order to realize reactions by using edits, we must define how an embedding of a pattern can mediate the edit of an agent. Since we wish to combine sequences of edits, the result of a mediated edit should be both a new agent and a new embedding that can mediate the next edit into the new agent. Furthermore, we shall define mediation of edits through embeddings into arbitrary bigraphs as we shall need this to characterize conflict and causality in Section 7.7.

**Definition 7.6.12** (mediated edits)**.** For a pattern $\tilde{P}$ and compatible edit $\delta$, the *mediated edit*, written $\delta(a, \phi)$, of a pattern $\tilde{H}$ through an embedding $\phi : \tilde{P} \hookrightarrow \tilde{H}$ is defined in Table 7.3. When $(\tilde{H}', \phi') = \delta(\tilde{H}, \phi)$ we shall often abuse the notation and write $\delta(\tilde{H}, \phi)$ for $\tilde{H}'$. □

**Proposition 7.6.13** (mediated edits)**.** *Given a pattern $\tilde{P} : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle$, a compatible edit $\delta$, and an embedding $\phi : \tilde{P} \hookrightarrow \tilde{H}$ into a pattern $\tilde{H} : \langle Q_{\tilde{H}}, X_{\tilde{H}} \rangle \to I$. Then $\tilde{H}' : \langle Q'_{\tilde{H}}, X_{\tilde{H}} \rangle \to I$ is a pattern and $\phi' : \tilde{P}' \hookrightarrow \tilde{H}'$ is an embedding, where $(\tilde{H}' : \langle Q'_{\tilde{H}}, X_{\tilde{H}} \rangle \to I, \phi') = \delta(\tilde{H}, \phi)$ and $\tilde{P}' = \delta(\tilde{P})$ for some set of variables $Q'_{\tilde{H}}$.*

*Proof.* Cf. Appendix 7.A.2. □

Mediated edits are well-behaved in the sense that the reconfigurations they cause in agents can also be expressed as reactions:

**Lemma 7.6.14** (mediated edits are reactions)**.** *Given a pattern $\tilde{P} : Q \to \langle R, Y \rangle$, a compatible edit $\delta$, and an embedding $\phi : \tilde{P} \hookrightarrow a$ into a concrete agent $a : \langle m_a, Y_a \rangle$. Then $a \twoheadrightarrow a'$, where $(a', \phi') = \delta(a, \phi)$, is a reaction in any pattern-based BPRS containing the rule $(\tilde{P}, \delta(\tilde{P}), inst_Q(finst(\delta)))$.*

*Proof.* Cf. Appendix 7.A.2. □

The converse does not hold for two reasons:

- A single edit of course cannot express any reaction. We will solve this by introducing edit scripts in the next section.
- Edits are much more restrictive in their handling of support: we cannot change support arbitrarily, but are only free to choose support for added/copied nodes and edges.

We can, however, show that any abstract reaction can be realized by edits. In general, this requires edit scripts, but let us first show that abstract reactions generated by rules derived from edits can also be obtained through mediated edits:

**Lemma 7.6.15.** *Given a pattern-based BPRS and a reaction $a \twoheadrightarrow a'$ generated by some pattern rule $\mathsf{R} = (\tilde{P} : Q \to \langle R, Y \rangle, \delta(\tilde{P}) : Q' \to \langle R, Y \rangle, inst_Q(finst(\delta)))$. Then there is an agent $a''$ and embeddings $\phi : \tilde{P} \hookrightarrow a$, $\phi' : \delta(\tilde{P}) \hookrightarrow a''$ such that $a' \simeq a''$ and $(a'', \phi') = \delta(a, \phi)$.*

*Proof.* Cf. Appendix 7.A.2. □

**Rebind a port:**

$$\odot_{(v,i)\mapsto l}(\tilde{H},\phi) \stackrel{\text{def}}{=} ((V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}, prnt_{\tilde{H}}, link_{\tilde{H}}[(\phi(v),i)\mapsto \phi(l)]),\phi)$$

**Change a control:**

$$\circledcirc_{v:K}(\tilde{H},\phi) \stackrel{\text{def}}{=} ((V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}[\phi(v)\mapsto K], prnt_{\tilde{H}}, link_{\tilde{H}}),\phi)$$

**Add node or edge:**

$$\begin{aligned}
\oplus_{v:K_{\vec{y}}@p}(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}+v', E_{\tilde{H}}, ctrl_{\tilde{H}}[v'\mapsto K], prnt_{\tilde{H}}[v'\mapsto \phi(p)],\\
&\qquad link_{\tilde{H}}[(v',0)\mapsto \phi(\vec{y}_0),\ldots,(v',n-1)\mapsto \phi(\vec{y}_{n-1})]),\\
&\qquad \phi[v\mapsto v'])\\
&\text{if } v'\notin V_{\tilde{H}}\\
\oplus_e(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}, E_{\tilde{H}}+e', ctrl_{\tilde{H}}, prnt_{\tilde{H}}, link_{\tilde{H}}),\phi[e\mapsto e'])\\
&\text{if } e'\notin E_{\tilde{H}}
\end{aligned}$$

**Delete node, edge, or parameter:**

$$\begin{aligned}
\ominus_v(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}-\phi(v), E_{\tilde{H}}, ctrl_{\tilde{H}}-\phi(v), prnt_{\tilde{H}}-\phi(v), link_{\tilde{H}}-P_{\phi(v)}),\phi-v)\\
\ominus_e(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}, E_{\tilde{H}}-\phi(e), ctrl, prnt_{\tilde{H}}, link_{\tilde{H}}),\phi-e)\\
\ominus_q(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}\setminus \tilde{H}\downharpoonright^{\phi(q)}, E_{\tilde{H}}, ctrl_{\tilde{H}}-\tilde{H}\downharpoonright^{\phi(q)}, prnt_{\tilde{H}}-\tilde{H}\downharpoonright^{\phi(q)}, link_{\tilde{H}}-P_{\tilde{H}\downharpoonright^{\phi(q)}})\\
&\qquad :\langle Q\setminus \tilde{H}\downharpoonright^{\phi(q)}, X\rangle\to I,\\
&\qquad \phi-q)
\end{aligned}$$

**Move node or parameter:**

$$\begin{aligned}
\oslash_{v@p}(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}, prnt_{\tilde{H}}[\phi(v)\mapsto\phi(p)], link_{\tilde{H}}),\phi)\\
\oslash_{q@p}(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}, prnt_{\tilde{H}}[\phi(q)\mapsto\phi(p)], link_{\tilde{H}}),\phi)
\end{aligned}$$

**Copy parameter:**

$$\begin{aligned}
\otimes_{q\to r@p}(\tilde{H},\phi) &\stackrel{\text{def}}{=} ((V_{\tilde{H}}\uplus V_r, E_{\tilde{H}}, ctrl_{\tilde{H}}\uplus ctrl_r, prnt_{\tilde{H}}\uplus prnt_r, link_{\tilde{H}}\uplus link_r),\\
&\qquad :\langle Q\uplus Q_r, X\rangle\to I,\\
&\qquad \phi[r\mapsto f^{-1}(\phi(q))])\\
&\text{where} \quad V_q = \tilde{H}\downharpoonright^{\phi(q)}\cap V_{\tilde{H}} \qquad Q_q = \tilde{H}\downharpoonright^{\phi(q)}\cap Q\\
&\qquad |V_r| = |V_q| \qquad |Q_r| = |Q_q|\\
&\qquad V_r \# V_{\tilde{H}} \qquad Q_r \# Q\\
&\qquad f_v : V_r \rightarrowtail V_q \qquad f_s : Q_r\rightarrowtail Q_q\\
&\qquad f = f_v\uplus f_s\\
&\qquad ctrl_r = ctrl_{\tilde{H}}\circ f_v\\
&\qquad prnt_r = \{f^{-1}(\phi(q))\mapsto\phi(p)\}\uplus f_v^{-1}\circ prnt_{\tilde{H}}\circ(f-f^{-1}(\phi(q)))\\
&\qquad link_r(v,i) = link_{\tilde{H}}(f_v(v),i) \quad (v\in V_r)
\end{aligned}$$

Table 7.3: Mediating compatible edit $\delta$ of $\tilde{P}$ to $\tilde{H}:\langle Q,X\rangle\to I$ through embedding $\phi:\tilde{P}\hookrightarrow\tilde{H}$. Interfaces are omitted for clarity in all but the two cases where they change.

### 7.6.3 Edit Scripts

In the previous section we took care to define the constructions such that they could easily be transferred to sequences of edits. In this section we reap the benefits: we transfer the concepts and results from edits to edit scripts without further comment.

**Definition 7.6.16** (edit script)**.** An *edit script* is a finite sequence of edits $\Delta = \delta_1 \cdots \delta_n$. An edit script is *compatible* with a pattern $\tilde{P}$ iff $\delta_1$ is compatible with $\tilde{P}$ *and* $n = 1$ or $\delta_2 \cdots \delta_n$ is compatible with $\delta(\tilde{P})$. The *application* of an edit script $\Delta$ to a compatible pattern $\tilde{P}$ is defined to be

$$\Delta(\tilde{P}) \stackrel{\text{def}}{=} \delta_n(\cdots \delta_1(\tilde{P}) \cdots).$$

Similarly, the *mediated application* of an edit script $\Delta$ to a pattern $\tilde{H}$ through an embedding $\phi : \tilde{P} \hookrightarrow \tilde{H}$ of a compatible pattern $\tilde{P}$ is defined to be

$$\Delta(\tilde{H}, \phi) \stackrel{\text{def}}{=} \delta_n(\cdots \delta_1(\tilde{H}, \phi) \cdots).$$

When $(\tilde{H}', \phi') = \Delta(\tilde{H}, \phi)$ we shall often abuse the notation and write $\Delta(\tilde{H}, \phi)$ for $\tilde{H}'$. □

**Corollary 7.6.17** (edit scripts)**.** *Given a pattern $\tilde{P} : Q \to \langle R, Y \rangle$ and a compatible edit script $\Delta$. Then $\Delta(\tilde{P}) : Q' \to \langle R, Y \rangle$ is a pattern.*

*Proof.* Follows from Prop. 7.6.7 by straightforward induction on the length of $\Delta$. □

**Corollary 7.6.18** (mediated edit scripts)**.** *Given a pattern $\tilde{P} : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle$, a compatible edit script $\Delta$, and an embedding $\phi : \tilde{P} \hookrightarrow \tilde{H}$ into a pattern $\tilde{H} : \langle Q_{\tilde{H}}, X_{\tilde{H}} \rangle \to I$. Then $\tilde{H}' : \langle Q'_{\tilde{H}}, X_{\tilde{H}} \rangle \to I$ is an agent and $\phi' : \tilde{P}' \hookrightarrow \tilde{H}'$ is an embedding, where $(\tilde{H}' : \langle Q'_{\tilde{H}}, X_{\tilde{H}} \rangle \to I, \phi') = \Delta(\tilde{H}, \phi)$ and $\tilde{P}' = \Delta(\tilde{P})$ for some set of variables $Q'_{\tilde{H}}$.*

*Proof.* Follows from Prop. 7.6.13 by straightforward induction on the length of $\Delta$. □

**Definition 7.6.19** (edit script forward instance map)**.** The *forward instance map* $\mathit{finst}(\Delta)$ corresponding to an edit script $\Delta = \delta_1 \cdots \delta_n$ is

$$\mathit{finst}(\Delta) \stackrel{\text{def}}{=} \mathit{finst}(\delta_n) \circ (\cdots \circ (\mathit{finst}(\delta_1) \downharpoonleft^{\mathsf{dom}(\mathit{finst}(\delta_2))}) \cdots \downharpoonleft^{\mathsf{dom}(\mathit{finst}(\delta_n))}).$$

□

**Corollary 7.6.20** (edit script forward instance map)**.** *Given a pattern $\tilde{P} : Q \to J$ and a compatible edit script $\Delta$. Then $Q \subseteq \mathsf{dom}(\mathit{finst}(\Delta))$ and $\mathit{finst}(\Delta)(Q) = Q'$ where $\Delta(\tilde{P}) : Q' \to J$.*

*Proof.* Follows from Prop. 7.6.9 by straightforward induction on the length of $\Delta$. □

**Corollary 7.6.21** (derived variable instance map)**.** *Given a pattern $\tilde{P} : Q \to J$ and a compatible edit script $\Delta$. Then $\mathit{inst}_Q(\mathit{finst}(\Delta)) : Q' \to Q$ is a variable instance map where $\Delta(\tilde{P}) : Q' \to J$.*

*Proof.* Follows from Def. 7.6.10 and Corol. 7.6.20. □

**Corollary 7.6.22** (mediated edit scripts are reactions)**.** *Given a pattern $\tilde{P} : Q \to \langle R, Y \rangle$, a compatible edit script $\Delta$, and an embedding $\phi : \tilde{P} \hookrightarrow a$ into a concrete agent $a : \langle m_a, Y_a \rangle$. Then $a \twoheadrightarrow a'$, where $(a', \phi') = \Delta(a, \phi)$, is a reaction in any pattern-based BPRS containing the rule $(\tilde{P}, \Delta(\tilde{P}), \mathit{inst}_Q(\mathit{finst}(\Delta)))$.*

*Proof.* Follows from Lemma 7.6.14 by straightforward induction on the length of $\Delta$. □

**Corollary 7.6.23.** *Given a pattern-based BPRS and a reaction $a \twoheadrightarrow a'$ generated by some pattern rule $\mathsf{R} = (\tilde{P} : Q \to \langle R, Y \rangle, \Delta(\tilde{P}) : Q' \to \langle R, Y \rangle, \mathit{inst}_Q(\mathit{finst}(\Delta)))$. Then there is an agent $a''$ and embeddings $\phi : \tilde{P} \hookrightarrow a$, $\phi' : \Delta(\tilde{P}) \hookrightarrow a''$ such that $a' \simeq a''$ and $(a'', \phi') = \Delta(a, \phi)$.*

*Proof.* Follows from Lemma 7.6.15 by straightforward induction on the length of $\Delta$. □

### 7.6.4 Reconfiguration Systems

In the previous sections, we have seen (a) that edit scripts generate reactions and (b) that if we can express a pattern rule as an edit script, that edit script generates the same abstract reactions as the pattern rule. Thus, if any pattern rule can be expressed as an edit script, we can generate all abstract reactions through edit scripts. In this section we shall give a construction of an edit script for any pattern rule, thus providing the final piece of the puzzle.

To show the correspondence between mediated edits and reactions, we shall define *reconfiguration rules* and *reconfiguration systems (RCSs)* and show that they are equivalent to pattern rules and pattern-based BPRS.

In overview, the development proceeds as follows:

**reconfiguration rules:**
We first define *reconfiguration rules* as pairs of patterns and edit scripts. We give constructions of pattern rules from reconfiguration rules and vice versa.

**reconfiguration systems:**
Next, we define RCSs, give constructions of pattern-based BPRS from RCSs and vice versa, and show that the constructions preserve and reflect abstract reactions.

#### Reconfiguration Rules

A reconfiguration rule is simply a pattern paired with a compatible edit script:

**Definition 7.6.24** (reconfiguration rule)**.** A *reconfiguration rule* $\tilde{R} = (\tilde{P}, \Delta)$ consists of a pattern $\tilde{P} : Q \to J$ and a compatible edit script $\Delta$. □

From a reconfiguration rule, we can easily construct a pattern rule:

**Definition 7.6.25** (pattern rule corresponding to reconfiguration rule)**.** The pattern rule corresponding to a reconfiguration rule $\tilde{R}$ is defined as

$$[\![\tilde{R}]\!] \stackrel{\text{def}}{=} (\tilde{P}, \Delta(\tilde{P}), \mathit{inst}_Q(\mathit{finst}(\Delta))).$$

□

**Proposition 7.6.26** (pattern rule corresponding to reconfiguration rule)**.** *The pattern rule corresponding to a reconfiguration rule is indeed a pattern rule.*

*Proof.* Let

$$\begin{aligned} \tilde{R} &= (\tilde{P} : Q \to \langle R, Y \rangle, \Delta) \\ [\![\tilde{R}]\!] &= (\tilde{P}, \Delta(\tilde{P}), \mathit{inst}_Q(\mathit{finst}(\Delta))). \end{aligned}$$

By Corol. 7.6.17 and Corol. 7.6.21, $\Delta(\tilde{P}) : Q' \to \langle R, Y \rangle$ is a pattern and $\mathit{inst}_Q(\mathit{finst}(\Delta)) : Q' \to Q$ is a variable instance map. □

The reverse direction is more tricky: in general, there will be infinitely many edit scripts that express a reaction, since we can always extend an edit script by two edits that cancel out, e.g., by adding and removing an edge. Here we shall give a naive construction, which first removes all the nodes, edges, and redundant sites from the redex and then builds up the reactum. In overview, the edit script will consist of the following steps:

(1) Copy the sites that will be in the reactum to a root (using fresh variables to avoid clashes.

(2) Delete the original sites.

$es(\mathsf{R}) =$

| | | |
|---|---|---|
| (1) | $\otimes_{\eta(q'_0)\to f(q'_1)@r} \cdots \otimes_{\eta(q'_{n'})\to f(q'_{n'})@r}$ | copy sites as prescribed by $\eta$ but using temporary names and placed at root $r$ |
| (2) | $\ominus_{q_1} \cdots \ominus_{q_n}$ | delete redex sites |
| (3) | $\ominus_{v_1} \cdots \ominus_{v_k}$ | delete redex nodes |
| (4) | $\ominus_{e_1} \cdots \ominus_{e_m}$ | delete redex edges |
| (5) | $\oplus_{e'_1} \cdots \oplus_{e_{m'}}$ | add reactum edges |
| (6) | $\oplus_{v'_1:ctrl_{\tilde{P}'}(v'_1)_{[\ldots,link_{\tilde{P}'}(v'_1,i),\ldots]}@\,prnt_{\tilde{P}'}(v'_1)} \cdots \oplus_{v'_{k'}:ctrl_{\tilde{P}'}(v'_{k'})_{[\ldots,link_{\tilde{P}'}(v'_{k'},i),\ldots]}@\,prnt_{\tilde{P}'}(v'_{k'})}$ | add reactum nodes |
| (7) | $\otimes_{f(q'_1)\to q'_1@\,prnt_{\tilde{P}'}(q'_1)} \cdots \otimes_{f(q'_{n'})\to q'_{n'}@\,prnt_{\tilde{P}'}(q'_{n'})}$ | copy sites to their proper places with proper names |
| (8) | $\ominus_{f(q'_1)} \cdots \ominus_{f(q'_{n'})}$ | delete temporary sites |

for some $r \in R$ and set of variables $Q'' \# Q \cup Q'$ in bijection to $Q'$, i.e., $f : Q' \rightarrowtail Q''$, and assuming the following sequencing of the entities of $\tilde{P}$ and $\tilde{P}'$:

$$
\begin{aligned}
Q &= \{q_1, \ldots, q_n\} & Q' &= \{q'_1, \ldots, q'_{n'}\} \\
E_{\tilde{P}} &= \{e_1, \ldots, e_m\} & E_{\tilde{P}'} &= \{e'_1, \ldots, e'_{m'}\} \\
V_{\tilde{P}} &= \{v_1, \ldots, v_k\} & &\text{where } \forall i,j \in [1;k] : prnt_{\tilde{P}}(v_i) = v_j \Rightarrow i < j \\
V_{\tilde{P}'} &= \{v'_1, \ldots, v'_{k'}\} & &\text{where } \forall i,j \in [1;k'] : prnt_{\tilde{P}'}(v'_i) = v'_j \Rightarrow i > j.
\end{aligned}
$$

Figure 7.5: Naive construction of edit script from pattern rule $\mathsf{R} = (\tilde{P} : Q \to \langle R, Y \rangle, \tilde{P}' : Q' \to \langle R, Y \rangle, \eta : Q' \to Q)$, denoted by $es(\mathsf{R})$.

(3) Delete all nodes (deleting children before their parents, i.e., bottom-up).

(4) Delete all edges.

(5) Add the edges of the reactum.

(6) Add the nodes of the reactum (adding parents before their children).

(7) Copy the sites to their proper place in the reactum (assigning the proper variable to the copy).

(8) Delete the sites created in step (1).

The formal construction is given by the following definition:

**Definition 7.6.27** (naive pattern rule edit script)**.** For a pattern rule $\mathsf{R}$ the *corresponding naive edit script*, written $es(\mathsf{R})$, is defined in Figure 7.5. □

**Proposition 7.6.28** (naive pattern rule edit script)**.** *Given a pattern rule* $\mathsf{R} = (\tilde{P} : Q \to J, \tilde{P}' : Q' \to J, \eta)$*. Then* $es(\mathsf{R})$ *is compatible with* $\tilde{P}$*,* $es(\mathsf{R})(\tilde{P}) = \tilde{P}'$*, and* $inst_Q(\mathit{finst}(es(\mathsf{R}))) = \eta$.

*Proof.* Cf. Appendix 7.A.2. □

Thus, for any parametric reaction rule we can construct a corresponding reconfiguration rule:

**Definition 7.6.29** (reconfiguration rule corresponding to pattern rule)**.** The reconfiguration rule corresponding to a pattern rule $\mathsf{R}$ is defined as

$$(\tilde{P}, es(\mathsf{R})).$$

□

**Proposition 7.6.30** (reconfiguration rule corresponding to pattern rule)**.** *The reconfiguration rule corresponding to a pattern rule is indeed a reconfiguration rule.*

*Proof.* Follows immediately from Def. 7.6.24 and Prop. 7.6.28. □

**Reconfiguration Systems**

Let us now give an alternative formulation of BPRSs based on reconfiguration rules:

**Definition 7.6.31** (reconfiguration systems (RCS))**.** A *reconfiguration system (RCS)* over $\mathcal{K}$, written $\grave{}\mathrm{B}\mathrm{G}(\mathcal{K}, \tilde{\mathcal{R}})$, consists of the s-category $\grave{}\mathrm{B}\mathrm{G}(\mathcal{K})$ equipped with a set $\tilde{\mathcal{R}}$ of reconfiguration rules.

The *reaction relation* $\longrightarrow$ over agents $a, a'$ is the smallest such that $a \longrightarrow a'$ whenever $\phi : \tilde{P} \hookrightarrow a$ is a match of some reconfiguration rule $\tilde{R} = (\tilde{P}, \Delta) \in \tilde{\mathcal{R}}$ in $a$ and $(a', \phi') = \Delta(a, \phi)$ for some embedding $\phi'$. □

From the results for edit scripts, it should be clear that RCSs have the same abstract reaction relations as pattern-based BPRSs. We shall now show this formally.

First, we can construct a pattern-based BPRS with the same abstract reactions as an RCS:

**Definition 7.6.32** (BPRS corresponding to RCS)**.** The pattern-based BPRS *corresponding* to an RCS $\grave{}\mathrm{B}\mathrm{G}(\mathcal{K}, \tilde{\mathcal{R}})$ is

$$\grave{}\mathrm{B}\mathrm{G}(\mathcal{K}, \{[\![\tilde{R}]\!] \mid \tilde{R} \in \tilde{\mathcal{R}}\}).$$

□

**Proposition 7.6.33** (BPRS corresponding to RCS)**.** *The pattern-based BPRS corresponding to an RCS is indeed a pattern-based BPRS.*

*Proof.* Follows immediately from Def. 7.6.4 and Prop. 7.6.26. □

**Theorem 7.6.34** (abstract reaction equivalence of BPRS corresponding to RCS)**.** *Given a reaction $a \longrightarrow_r a'$ in an RCS, then the corresponding pattern-based BPRS has the same reaction. Conversely, for any reaction $a \longrightarrow_p a'$ in the corresponding BPRS, there is an agent $a''$ such that $a' \simeq a''$ and $a \longrightarrow_r a''$ in the RCS.*

*Proof.* $\Rightarrow$: Assume a reaction $a \longrightarrow_r a'$ in an RCS, i.e., there is some match $\phi : \tilde{P} \hookrightarrow a$ of a reconfiguration rule $\tilde{R} = (\tilde{P}, \Delta) \in \tilde{\mathcal{R}}$ in $a$ and $(a', \phi') = \Delta(a, \phi)$ for some embedding $\phi'$. By Def. 7.6.32 and Def. 7.6.25 the corresponding pattern-based BPRS contains the rule $(\tilde{P}, \Delta(\tilde{P}), inst_Q(\mathit{finst}(\Delta)))$, so, by Corol. 7.6.22, $a \longrightarrow_p a'$.

$\Leftarrow$: Assume a reaction $a \longrightarrow_p a'$ generated by some pattern rule $(\tilde{P}, \Delta(\tilde{P}), inst_Q(\mathit{finst}(\Delta)))$ in the pattern-based BPRS corresponding to an RCS. By Corol. 7.6.23, there is an agent $a''$ and embeddings $\phi : \tilde{P} \hookrightarrow a$, $\phi' : \Delta(\tilde{P}) \hookrightarrow a''$ such that $a' \simeq a''$ and $(a'', \phi') = \Delta(a, \phi)$. Thus, $a \longrightarrow_r a''$. □

Conversely, for any pattern-based BPRS we can construct an RCS which has the same abstract reactions:

**Definition 7.6.35** (RCS corresponding to BPRS)**.** The RCS *corresponding* to a pattern-based BPRS $`\textsc{Bg}(\mathcal{K}, `\mathcal{R})$ is

$$`\textsc{Bg}(\mathcal{K}, \{(\tilde{P}, es(\mathsf{R})) \mid \mathsf{R} \in `\mathcal{R}\}).$$

□

**Proposition 7.6.36** (RCS corresponding to BPRS)**.** *The RCS corresponding to a pattern-based BPRS is indeed an RCS.*

*Proof.* Follows immediately from Def. 7.6.31 and Prop. 7.6.30. □

**Theorem 7.6.37** (abstract reaction equivalence of RCS corresponding to BPRS)**.** *Given a reaction $a \rightarrowtriangle_p a'$ in a pattern-based BPRS, then there is an agent $a''$ such that $a' \bumpeq a''$ and $a \rightarrowtriangle_r a''$ in the corresponding RCS. Conversely, any reaction $a \rightarrowtriangle_r a'$ in the corresponding RCS is a reaction in the pattern-based BPRS.*

*Proof.* $\Rightarrow$: Assume a reaction $a \rightarrowtriangle_p a'$ in the pattern-based BPRS generated by some pattern rule $(\tilde{P}, \tilde{P}', \eta)$, i.e., there is a match $\phi : \tilde{P} \hookrightarrow a$. By Def. 7.6.35 the corresponding RCS has the reconfiguration rule $(\tilde{P}, es(\mathsf{R}))$ and, by Prop. 7.6.28, $es(\mathsf{R})(\tilde{P}) = \tilde{P}'$, and $inst_Q(\mathit{finst}(es(\mathsf{R}))) = \eta$. Thus, by Corol. 7.6.23, there is an agent $a''$ and embeddings $\phi : \tilde{P} \hookrightarrow a$, $\phi' : \Delta(\tilde{P}) \hookrightarrow a''$ such that $a' \bumpeq a''$ and $(a'', \phi') = \Delta(a, \phi)$. Thus, $a \rightarrowtriangle_r a''$.

$\Leftarrow$: Assume a reaction $a \rightarrowtriangle_r a'$ in generated by some reconfiguration rule $(\tilde{P}, es(\mathsf{R}))$ in the RCS corresponding to a pattern-based BPRS with $\mathsf{R} = (\tilde{P}, \tilde{P}', \eta)$. By Prop. 7.6.28, $\mathsf{R} = (\tilde{P}, es(\mathsf{R})(\tilde{P}), inst_Q(\mathit{finst}(es(\mathsf{R}))))$, so, by Corol. 7.6.22, $a \rightarrowtriangle_p a'$. □

## 7.7 Rule Activation and Inhibition

The KaSim algorithm presumes that we can characterize causality and conflict at the level of reaction rules: it requires two relations over reaction rules, activation $\mathsf{R}_0 \prec \mathsf{R}_1$ and inhibition $\mathsf{R}_0 \# \mathsf{R}_1$, capturing whether a reaction using $\mathsf{R}_0$ can cause or prevent, respectively, reactions using $\mathsf{R}_1$. These relations reduce the number of rules that must be considered in the positive and negative update phases of the algorithm.

In this section, we outline how we hope to construct these relations through a characterization of causality and conflict in terms of pullbacks and pushouts in the category of bigraph embeddings: intuitively, a pullback characterizes one way two bigraphs can overlap in a context, while the pushout of a pullback is the minimal example of such an overlap. For simplicity, we shall assume that rules are linear, i.e., they do not copy or delete parameters.

This section proceeds as follows:

1. First, we give definitions of the usual notions of causality and conflict in the contexts of reconfiguration systems and show how they can be expressed in terms of embeddings and edit scripts.

2. We then define and discuss the category of bigraph embeddings and state a number of conjectures which our approach relies on. In particular, we argue that the embedding category has pullbacks and pushout of pullbacks if we relax the embedding conditions slightly.

3. Finally, we discuss how our conjectures about the category of bigraph embeddings should allow us to characterize causality and conflict at the level of rules and thus construct the activation and inhibition relations.

### 7.7.1 Causality and Conflict

The notions of causality and conflict between events are well-studied in the literature also for graph rewriting, though sometimes through their duals, sequential and parallel independence [16, 32]: events are causally related if one must precede the other, and in conflict if one prevents the other. In this section we shall, essentially, take events to be reactions in an RCS where the reaction relation $a \twoheadrightarrow_{\mathsf{R},\phi} a'$ is extended with labels that record the rule $\mathsf{R} = (\tilde{P}, \Delta)$ and embedding $\phi : \tilde{P} \hookrightarrow a$ that generated the reaction.

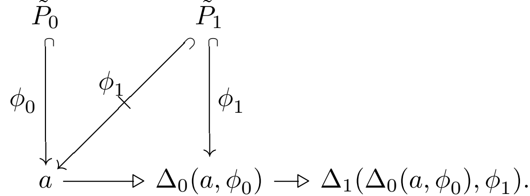
More precisely, we define causality and conflict as follows:

**Definition 7.7.1** (causality)**.** In an RCS, say that reaction $a \twoheadrightarrow_{\mathsf{R}_0,\phi_0} b$ causes reaction $b \twoheadrightarrow_{\mathsf{R}_1,\phi_1} c$ iff there is no $b'$ such that $a \twoheadrightarrow_{\mathsf{R}_1,\phi_1} b'$. □

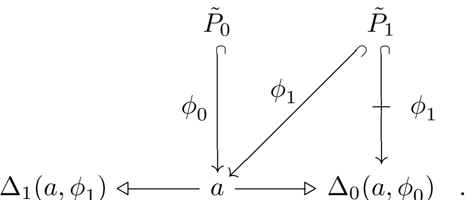
**Definition 7.7.2** (conflict)**.** In an RCS, say that reaction $a \twoheadrightarrow_{\mathsf{R}_0,\phi_0} b$ conflicts with reaction $a \twoheadrightarrow_{\mathsf{R}_1,\phi_1} b'$ iff there is no $c$ such that $b \twoheadrightarrow_{\mathsf{R}_1,\phi_1} c$. □

Inspecting the definition of RCSs (Def. 7.6.31) it is clear that causality and conflict can be restated in terms of embeddings and edit scripts as follows:

**Proposition 7.7.3** (causality)**.** *In an RCS, the reaction* $a \twoheadrightarrow_{\mathsf{R}_0,\phi_0} \Delta_0(a,\phi_0)$ *causes the reaction* $\Delta_0(a,\phi_0) \twoheadrightarrow_{\mathsf{R}_1,\phi_1} \Delta_1(\Delta_0(a,\phi_0),\phi_1)$ *iff* $\phi_1 : \tilde{P}_1 \not\hookrightarrow a$*, where* $\mathsf{R}_i = (\tilde{P}_i, \Delta_i)$ *(i = 0,1). The following diagram illustrates the situation:*



**Proposition 7.7.4** (conflict)**.** *In an RCS, the reaction* $a \twoheadrightarrow_{\mathsf{R}_0,\phi_0} \Delta_0(a,\phi_0)$ *conflicts with the reaction* $a \twoheadrightarrow_{\mathsf{R}_1,\phi_1} \Delta_1(a,\phi_1)$ *iff* $\phi_1 : \tilde{P}_1 \not\hookrightarrow \Delta_0(a,\phi_0)$*, where* $\mathsf{R}_i = (\tilde{P}_i, \Delta_i)$ *(i = 0,1). The following diagram illustrates the situation:*



In both cases, reaction using $\mathsf{R}_0$ must modify something in the range of $\phi_1$ since it becomes, or stops being, an embedding. Furthermore, the reaction generated by $\phi_0$ can only modify the parts of the agent that are in its range. Thus, there must be some overlap between the embeddings which is modified by reaction. It turns out that we can express this modification of overlaps concisely using category theory, so let us now discuss the category of bigraph embeddings.

### 7.7.2 Category of Bigraph Embeddings

The embeddings of Section 7.5 form categories where the objects are graphs and the arrows are embeddings:

**Definition 7.7.5** (category of bigraph embeddings)**.** The bigraphical embedding categories LgEmb($\mathcal{K}$), PgEmb($\mathcal{K}$), BgEmb($\mathcal{K}$) over a basic signature $\mathcal{K}$ respectively have link graphs, place graphs, and bigraphs over $\mathcal{K}$ as their objects and the arrows are embeddings.

Composition is function composition and identities $\mathsf{id}_G : G \hookrightarrow G$ are identity functions on the support and interfaces of $G$. □

These categories should not be confused with the usual bigraphical categories (cf. Section 7.2.2) where the arrows are bigraphs. For the remainder of this section we shall use lower case letters $f, g, \ldots$ for arrows in addition to $\phi$.

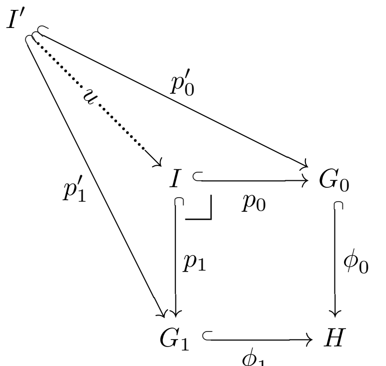
We are interested in these categories as the categorical notions of *pullback* and *pushout*, if the embedding categories have them, will allow us to characterize overlaps and minimal contexts exhibiting those overlaps, respectively. Let us discuss these two notions and their interpretations in some depth:

### Pullbacks of Embeddings

An overlap between two embeddings $\vec{\phi} : \vec{G} \hookrightarrow H$ can be thought of as a maximal shared subgraph $I$ and a pair of embeddings $\vec{p} : I \hookrightarrow \vec{G}$ such that the following diagram is a pullback diagram in the category of bigraph embeddings:

$$\begin{array}{ccc} I & \overset{p_0}{\hookrightarrow} & G_0 \\ {\scriptstyle p_1}\big\downarrow & \lrcorner & \big\downarrow{\scriptstyle \phi_0} \\ G_1 & \underset{\phi_1}{\hookrightarrow} & H \end{array} \quad .$$

The subgraph should be maximal to ensure that $\vec{p} : I \hookrightarrow \vec{G}$ describes all of the overlap and the pullback property captures this maximality requirement. More precisely, if there is another subgraph $I'$ and embeddings $\vec{p'} : I' \hookrightarrow \vec{G}$ that make the above diagram commute, then $I'$ is a subgraph of $I$, i.e., there is a (unique) embedding $u : I' \hookrightarrow I$ such that the following diagram commutes:




As an interesting special case, note that if the two embeddings do not overlap, the pullback is the empty bigraph and two empty maps.

We believe that the embedding formulation in Section 7.5 will have to be relaxed slightly in order for the embedding categories to have pullbacks. Let us first illustrate the issue with the embeddings of Section 7.5 through an example:

**Example 4.** Assume that we have the following overlapping link graph embeddings:

$$\begin{aligned} G_0 &= [x \mapsto v, y \mapsto w] : \{x, y\} \to \{v, w\} \\ G_1 &= [\{x, y, z\} \mapsto u] : \{x, y, z\} \to \{u\} \\ H &= [\{x, y\} \mapsto u] : \{x, y\} \to \{u\} \\ \phi_0 &= \mathsf{Id}_{\{x,y\}}[\{v, w\} \mapsto u] : G_0 \hookrightarrow H \\ \phi_1 &= \mathsf{Id}_{\{x,y,u\}}[z \mapsto \emptyset] : G_1 \hookrightarrow H. \end{aligned}$$

What should the pullback be? Since $G_0$ embeds into $G_1$ via $\phi_0$ it seems reasonable that $G_0$ could be the maximal overlap, i.e.,:

$$I' = G_0 \qquad g_0 = \mathsf{Id}_{G_0} : I' \hookrightarrow G_0 \qquad g_1 = \phi_0 : I' \hookrightarrow G_1.$$

Alas, this is not a pullback! Consider the link graph

$$I'' = G_0 \otimes [z \mapsto a] : \{x, y, z\} \to \{v, w, a\}.$$

It has the following embeddings into $G_0$,$G_1$

$$f_0 = \mathsf{Id}_{G_0} \uplus [a \mapsto v, z \mapsto \emptyset] : I'' \hookrightarrow G_0$$
$$f_0' = \mathsf{Id}_{G_0} \uplus [a \mapsto w, z \mapsto \emptyset] : I'' \hookrightarrow G_0$$
$$f_1 = f_1' = \phi_0 \uplus [a \mapsto u, z \mapsto z] : I'' \hookrightarrow G_1$$

which satisfy $\phi_0 \circ f_0 = \phi_0 \circ f_0' = \phi_1 \circ f_1$. If $\vec{g} : I' \hookrightarrow \vec{G}$ was a pullback there should be unique embeddings $u : I'' \hookrightarrow I'$, $u' : I'' \hookrightarrow I'$ such that $f_0 = g_0 \circ u$, $f_0' = g_0 \circ u'$, and $f_1 = g_1 \circ u = g_1 \circ u'$. But this is impossible since $\mathsf{rng}(f_1) \ni z \notin \mathsf{rng}(g_1)$.

Furthermore, neither $\vec{f} : I'' \hookrightarrow \vec{G}$ nor $\vec{f'} : I'' \hookrightarrow \vec{G}$ are pullbacks. For example, if $\vec{f} : I'' \hookrightarrow \vec{G}$ was a pullback, there should be a unique $u : I'' \hookrightarrow I''$ such that $f_0' = f_0 \circ u$ and $f_1' = f_1 \circ u = f_1' \circ u$. Since $f_1'(z) = z$ and $f_1'^{-1}(z) = z$ we must have $u(z) = z$, and since $f_0'(a) = f_0'(w) = w$ and $f_0^{-1}(w) = w$ we must have $u(a) = u(w) = w$. But this violates structure preservation which dictates $(u \circ prnt_{I''})(z) = prnt_{I''}(z) \Rightarrow u(a) = a$.

Intuitively, the problems arise because $\phi_1$ maps $z$ to the empty set. As we saw, though $\vec{g} : I' \hookrightarrow \vec{G}$ is an overlap, other subgraphs may include $z$ which is not in $I'$ and thus $I'$ is not maximal. But if we add $z$ to the overlap, we have to choose a single link of $G_0$ which it belongs to, cf. $f_0$ and $f_0'$. Any such choice is equally good but not isomorphic to the others and thus not a pullback.

It is easy to transfer this example to place graph embeddings (and this is a good exercise for the reader!) and the issues thus apply in that setting as well. □

We believe that this problem can be solved by allowing outer names/roots to embed into multiple outer names/roots, i.e., by changing conditions (LGE-4) and (PGE-3) to

(LGE-4') $\phi^{\mathsf{o}} : Y_G \to E_H \uplus \mathcal{P}(Y_H) \setminus \emptyset$ is an arbitrary map

(PGE-3') $\phi^{\mathsf{r}} : m_G \to V_H \uplus \mathcal{P}(m_H) \setminus \emptyset$ is an arbitrary map

The structure preservation conditions will ensure that outer names/roots, which contain at least one point/child that is not mapped to the empty set, will only be mapped to a single link/place. In other words, only outer names/roots, where all their points/children map to the empty set, are allowed to map to multiple outer names/roots. Intuitively, this models the situation where it is irrelevant which of the outer names/roots the outer name/root maps to.

Note that the results about decompositions and bigraph embeddings in Section 7.5 probably do not hold for the more general embeddings allowed by the relaxed conditions. However, this is unimportant, as we only need those result for the subset of embeddings that satisfy the original conditions, and they are still embeddings under the new conditions.

With the relaxed conditions on embeddings, we can construct the pullback in the above example:

**Example 5.** The pullback of the embeddings $\vec{\phi} : \vec{G} \hookrightarrow H$ from Example 4 is

$$I = I'' = G_0 \otimes [z \mapsto a] : \{x, y, z\} \to \{v, w, a\}$$
$$p_0 = \mathsf{Id}_{G_0} \uplus [a \mapsto \{v, w\}, z \mapsto \emptyset] : I \hookrightarrow G_0$$
$$p_1 = \phi_0 \uplus [a \mapsto u, z \mapsto z] : I \hookrightarrow G_1.$$

The unique embeddings $u_g : I' \hookrightarrow I, u_f : I'' \hookrightarrow I$, and $u_{f'} : I'' \hookrightarrow I$ for the spans $\vec{g} : I' \hookrightarrow \vec{G}$, $\vec{f} : I'' \hookrightarrow \vec{G}$, and $\vec{f'} : I'' \hookrightarrow \vec{G}$, respectively, are

$$u_g = \mathsf{Id}_{\{x,y,v,w\}} \qquad u_f = \mathsf{Id}_I \qquad u_{f'} = \mathsf{Id}_I.$$

□

We have a construction which we believe gives pullbacks for link graph embeddings, but have yet to prove it correct. So far, however, we have no indications that it should not be correct, and we believe that the construction transfers to place graphs and bigraphs. We therefore venture a conjecture:

**Conjecture 7.7.6** (pullbacks in the category of bigraph embeddings)**.** *The bigraphical embedding categories* LgEmb($\mathcal{K}$), PgEmb($\mathcal{K}$), BgEmb($\mathcal{K}$)*, where conditions (LGE-4) and (PGE-3) are replaced by conditions (LGE-4') and (PGE-3'), have pullbacks.*

Furthermore, since bigraphs are finite, we believe that there are only a finite number of ways that two bigraphs can overlap in any context, when we disregard the choice of support in those contexts. In other words, we conjecture that there is a finite set of pullbacks, up to isomorphism, for any two bigraphs:

**Conjecture 7.7.7.** *For any two objects $\vec{G}$ in one of the bigraphical embedding categories* LgEmb($\mathcal{K}$), PgEmb($\mathcal{K}$), BgEmb($\mathcal{K}$)*, where conditions (LGE-4) and (PGE-3) are replaced by conditions (LGE-4') and (PGE-3'), there are finitely many spans $\vec{p} : I \hookrightarrow \vec{G}$ (up to iso on $I$) which are pullbacks of a cospan $\phi : \vec{G} \hookrightarrow H$.*

In other words, we expect to be able to construct a finite representation of all possible overlaps between two bigraphs. We shall defer discussion of this construction until we have discussed pushouts on which the construction relies.

### Pushouts of Embeddings

Where a pullback is a maximal subgraph that characterizes the overlap of two embeddings, the dual notion of a *pushout*, if it exists, is a minimal context where two embeddings exhibit a given overlap. Pushouts do not in general exist in the bigraphical embedding categories as the following example illustrates:

**Example 6.** Consider the following span of place graph embeddings (we leave out controls for brevity):

$$\begin{aligned} I &= (\{v\}, [v \mapsto 0]) : 1 \\ G_0 &= (\{v, u\}, [v \mapsto u, u \mapsto 0]) : 1 \\ G_1 &= (\{v, w\}, [v \mapsto 0, w \mapsto 0]) : 1 \\ g_0 &= [v \mapsto v, 0 \mapsto u] : I \hookrightarrow G_0 \\ g_1 &= [v \mapsto v, 0 \mapsto 0] : I \hookrightarrow G_1. \end{aligned}$$

This span has no pushout because $G_1$ insists that $v$ has a sibling whereas $G_0$ insists that $v$ has no siblings and clearly no context can satisfy both of these requirements. □

Intuitively, this just means that we cannot single out a subgraph in two bigraphs and then construct a context where they overlap at that subgraph – which is unsurprising, since embeddings are structure preserving.

However, for pullbacks we know, by definition, that there are contexts where the overlap can be found. The remaining question is then: can we construct a minimal such context? We believe the answer is yes, but, as was the case for pullbacks, we shall have to relax the embedding conditions, as the following example illustrates:

**Example 7.** Consider the following pullback of place graph embeddings (we again leave out controls for brevity and use named sites and roots $q, r, s$ for clarity):

$$\begin{aligned}
I &= (\emptyset, \emptyset) : \emptyset \\
G_0 &= (\{v\}, [s \mapsto v, v \mapsto r]) : \{s\} \to \{r\} \\
G_1 &= (\{w\}, [w \mapsto q]) : \{q\} \\
H &= (\{v, u, x, w\}, [w \mapsto x, x \mapsto u, u \mapsto v, v \mapsto r]) : \{r\} \\
p_0 &= \emptyset : I \hookrightarrow G_0 \\
p_1 &= \emptyset : I \hookrightarrow G_1 \\
\phi_0 &= [v \mapsto v, s \mapsto u, r \mapsto r] : G_0 \hookrightarrow H \\
\phi_1 &= [w \mapsto w, q \mapsto x] : G_1 \hookrightarrow H.
\end{aligned}$$

What should the pushout of $\vec{p} : I \hookrightarrow \vec{G}$ be? The two embeddings do not overlap, so what is the canonical context where two place graphs do not overlap? Perhaps the pushout should be the tensor product of the two place graphs:

$$K = G_0 \otimes G_1 \qquad o_0 = \mathsf{Id}_{G_0} : G_0 \hookrightarrow K \qquad o_1 = \mathsf{Id}_{G_1} : G_1 \hookrightarrow K.$$

Alas, there is no embedding $u : K \hookrightarrow H$ such that $u \circ o_0 = \phi_0$ and $u \circ o_1 = \phi_1$ since condition (PGE-6) prevents us from mapping the root $q$ to a descendant of the site $s$ (in fact, this was the motivation for adding that condition, as discussed in Example 3).

If we disregard condition (PGE-6) then $\vec{o} : \vec{G_0} \hookrightarrow \vec{G_1}$ is a pushout. In particular, the embedding $u = \phi_0 \uplus \phi_1$ is the only one that satisfies $u \circ o_0 = \phi_0$ and $u \circ o_1 = \phi_1$. □

Thus, it seems that in order to have pushouts, we must discard condition (PGE-6). In fact, we believe that this all that is required in order to have pushouts of pullbacks in the bigraphical embedding categories. As for pullbacks, we have a construction of pushouts for pullbacks of link graph embeddings, which we think transfers to place graphs and bigraphs. But we have yet to prove it correct, though so far nothing indicates that it is incorrect. So, again, we venture a conjecture:

**Conjecture 7.7.8.** *For any pullback $\vec{p} : I \hookrightarrow \vec{G}$ of some cospan in one of the bigraphical embedding categories* LGEmb($\mathcal{K}$), PGEmb($\mathcal{K}$), BGEmb($\mathcal{K}$)*, where condition (PGE-6) is discarded and conditions (LGE-4) and (PGE-3) are replaced by conditions (LGE-4') and (PGE-3'), there is a pushout $\vec{o} : \vec{G} \hookrightarrow K$.*

### Characterizing Overlaps

As we discussed in Section 7.7.2, we think that there are only finitely many pullbacks for any two bigraphs (up to iso). While we hope to eventually find a direct method for enumerating these pullbacks, we will initially be content with any method. In particular, we believe that the following brute-force method will work (assuming we wish to characterize the overlaps of $\vec{G}$):

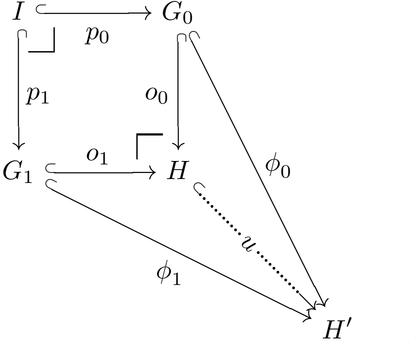
1. Generate overlap candidates, i.e., spans $\vec{g} : I' \hookrightarrow \vec{G}$, by equating subsets of entities of $G_0$ and $G_1$. It is still unclear to us exactly how this should be done, but for let us assume that we can generate such candidates and that there are finitely many.
2. For each overlap candidate $\vec{g} : I' \hookrightarrow \vec{G}$, apply the pushout construction. If it results in a bound $\vec{o} : \vec{G} \hookrightarrow K$ for $\vec{g}$, i.e., $K$ is a bigraph and $o_0 \circ g_0 = o_1 \circ g_1$, then we can construct a pullback.

Let us now assume that the conjectures of the previous two sections hold and that we have a method for obtaining the finite set of pullback spans for any pair $\vec{G}$ of bigraphs. In other words,

for each pair $\vec{G}$ of bigraphs assume that we have a finite set of pullback-pushout (PP) diagrams

$$\begin{array}{ccc} I & \xrightarrow{p_0} & G_0 \\ {\scriptstyle p_1}\downarrow & & \downarrow{\scriptstyle o_0} \\ G_1 & \xrightarrow[o_1]{} & H \end{array} \quad .$$

such that for any pair of embeddings $\vec{\phi} : \vec{G} \hookrightarrow H'$ its pullback is in one of the PP diagrams, i.e.,




Thus, if our conjectures hold, we can give a finite characterization of all overlaps between two bigraphs in any context.

### 7.7.3 PP Diagrams, Activation and Inhibition

Let us now return to the question of characterizing causality at the level of rules, i.e., the activation and inhibition relations. The characterization of overlaps based on PP diagrams, as outlined in the previous section, in itself provides the means to approximate the activation and inhibition relations $\prec$ and $\#$. For example, if two redexes can never overlap in a context, i.e., the only PP diagram has $I = \emptyset$, then they can never be in conflict. However, as discussed above in Section 7.7.1, we can do better than that by exploiting that edit scripts provide a notion of modification, and in this section we outline how.

We shall need a development for edit scripts which we, due to time constraints, only state as an assumption: we assume that we can construct an inverse $\Delta^{-1}$ of any linear edit script $\Delta$ which satisfies

$$\begin{array}{rl} & \phi : \Delta(\tilde{P}) \hookrightarrow H \\ \Rightarrow & \exists H', \phi' : \tilde{P} \hookrightarrow H' \ . \ (H, \phi) = \Delta(H', \phi') \wedge (H', \phi') = \Delta^{-1}(H, \phi). \end{array}$$

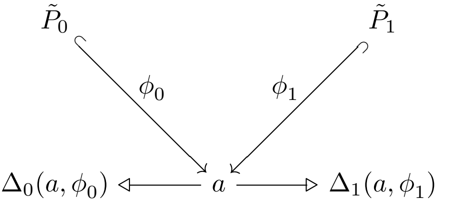
#### Inhibition

Recall from Section 7.3.2 that rule $\mathsf{R}_0$ *inhibits* rule $\mathsf{R}_1$, written $\mathsf{R}_0 \# \mathsf{R}_1$ iff $\mathsf{R}_0$ generates at least one reaction which conflicts with a reaction generated by $\mathsf{R}_1$.

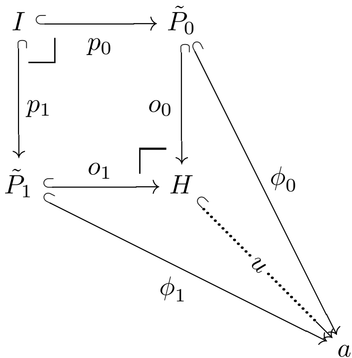
We believe that we can construct the inhibition relation through the PP diagrams of the previous section. More precisely, we make the following conjecture:

**Conjecture 7.7.9.** *Assume an agent $a$, two linear reconfiguration rules $\mathsf{R}_i = (\tilde{P}_i, \Delta_i)$ $(i = 0,1)$, and embeddings $\vec{\phi} : \vec{\tilde{P}} \hookrightarrow a$. By definition, the rules and embeddings generate the reactions $a \rightarrowtriangle_{\mathsf{R}_0,\phi_0}$*

*$\Delta_0(a, \phi_0)$ and $a \twoheadrightarrow_{\mathsf{R}_1,\phi_1} \Delta_1(a, \phi_1)$, as illustrated by the following diagram:*




*Also, assume that the cospan $\vec{\phi} : \vec{\tilde{P}} \hookrightarrow a$ has the PP diagram*




*We then conjecture*

$$\begin{array}{ccc} \phi_0 : \tilde{P}_0 \not\leftrightarrow \Delta_1(a, \phi_1) & \Leftrightarrow & o_0 : \tilde{P}_0 \not\leftrightarrow \Delta_1(H, o_1) \\ \phi_1 : \tilde{P}_1 \not\leftrightarrow \Delta_0(a, \phi_0) & \Leftrightarrow & o_1 : \tilde{P}_1 \not\leftrightarrow \Delta_0(H, o_0). \end{array}$$

Assuming this conjecture holds, we get that PP diagrams characterize inhibition:

**Theorem 7.7.10.** *Given two linear reconfiguration rules $\mathsf{R}_i = (\tilde{P}_i, \Delta_i)$ (i = 0,1) then $\mathsf{R}_0 \# \mathsf{R}_1$ iff there is a PP diagram*

$$\begin{array}{ccc} I & \xhookrightarrow{\;p_0\;} & \tilde{P}_0 \\ {\scriptstyle p_1}\big\downarrow & & \big\downarrow{\scriptstyle o_0} \\ \tilde{P}_1 & \xhookrightarrow{\;o_1\;} & H \end{array}$$

*such that $o_1 : \tilde{P}_1 \not\leftrightarrow \Delta_0(H, o_0)$.*

*Proof.* Follows immediately from Prop. 7.7.4 and Conjecture 7.7.9. □

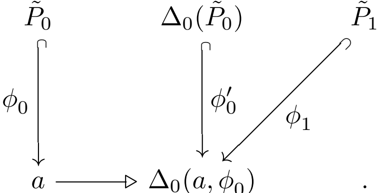
### Activation

Recall from Section 7.3.2 that rule $\mathsf{R}_0$ *activates* rule $\mathsf{R}_1$, written $\mathsf{R}_0 \prec \mathsf{R}_1$ iff $\mathsf{R}_0$ generates at least one reaction which enables a reaction generated by $\mathsf{R}_1$.

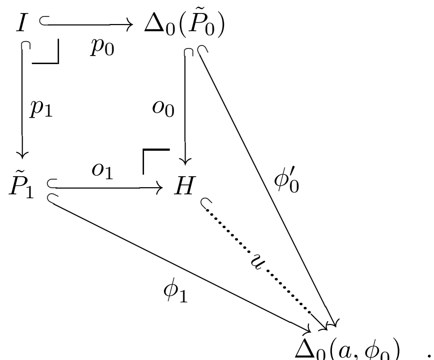
We believe that we can construct the activation relation through the PP diagrams of the previous section. More precisely, we make the following conjecture:

**Conjecture 7.7.11.** *Assume an agent $a$, two linear reconfiguration rules $\mathsf{R}_i = (\tilde{P}_i, \Delta_i)$ (i = 0,1), and embeddings $\phi_0 : \tilde{P}_0 \hookrightarrow a$, $\phi_0' : \Delta_0(\tilde{P}) \hookrightarrow a'$, $\phi_1 : \tilde{P}_1 \hookrightarrow a'$ where $(a', \phi_0') = \Delta_0(a, \phi_0)$, as*

*illustrated by the following diagram:*




*Also, assume that the cospan* $\phi'_0 : \Delta_0(\tilde{P}_0) \hookrightarrow a'$, $\phi_1 : \tilde{P}_1 \hookrightarrow a'$ *has the PP diagram*




*We then conjecture*

$$\phi_1 : \tilde{P}_1 \not\leftrightarrow a \qquad \Leftrightarrow \qquad o_1 : \tilde{P}_1 \not\leftrightarrow \Delta_0^{-1}(H, o_0).$$

Assuming this conjecture holds, we get that PP diagrams characterize activation:

**Theorem 7.7.12.** *Given two linear reconfiguration rules* $\mathsf{R}_i = (\tilde{P}_i, \Delta_i)$ *(i = 0,1) then* $\mathsf{R}_0 \prec \mathsf{R}_1$ *iff there is a PP diagram*

$$\begin{array}{ccc} I & \overset{p_0}{\hookrightarrow} & \Delta_0(\tilde{P}_0) \\ {\scriptstyle p_1}\downarrow & & \downarrow{\scriptstyle o_0} \\ \tilde{P}_1 & \underset{o_1}{\hookrightarrow} & H \end{array} .$$

*such that* $o_1 : \tilde{P}_1 \not\leftrightarrow \Delta_0^{-1}(H, o_0)$.

*Proof.* Follows immediately from Prop. 7.7.3 and Conjecture 7.7.11. □

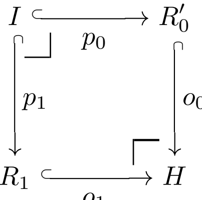
## 7.8 Anchored Matching

A pillar in the scalability of the KaSim algorithm is that, after reaction, we only search for new matches in the parts of the agent that have been modified. In other words, KaSim requires a localized matching algorithm that only searches a subset of the agent. Such an algorithm has not yet been presented in the bigraph literature. Previously published matching algorithms find matches anywhere in the agent [20, 34]; these algorithms are useful for the initialization phase of KaSim, where all matches must be found, but it is unclear how to specialize them to local matching.

In this section we shall present a localized matching algorithm, based on the idea of expanding partial embeddings to total embeddings, i.e., matches. Not only is this a localized matching algorithm, but it combines well with our edit scripts and causality analysis:

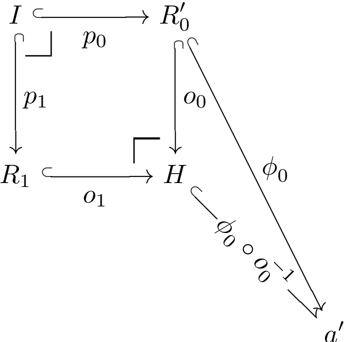
1. Mediation of an edit script results in a reaction $a \twoheadrightarrow a'$ and an embedding $\phi_0 : R'_0 \hookrightarrow a'$ of the reactum of the applied reaction rule $\mathsf{R}_0 = (R_0, R'_0, \eta_0)$ into the resulting agent $a'$.

2. For any rule $\mathsf{R}_1$ that may be activated by $\mathsf{R}_1 = (R_1, R'_1, \eta_1)$, i.e., $\mathsf{R}_0 \prec \mathsf{R}_1$, the causality analysis gives us a set of PP diagrams of the form

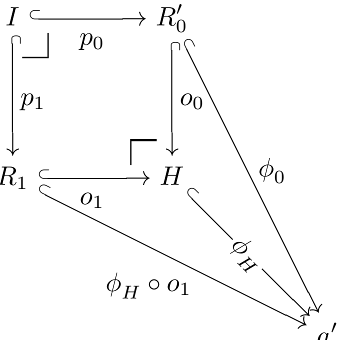



where $I \neq \epsilon$, which characterizes all matches of $R'_0$ and $R_1$ that overlap exactly as prescribed by $\vec{p} : I \hookrightarrow \vec{R}$.

3. Iff $\phi_0 \circ o_0^{-1} : H \rightharpoonup a'$ is a partial embedding




then any and all extensions to a total embedding of $H$, i.e., $\phi_H : H \hookrightarrow a'$ where $\phi_H \restriction_{\mathsf{rng}(o_0)} = \phi_0 \circ o_0^{-1}$, is a match of $R_1$, as witnessed by the composition $\phi_H \circ o_1 : R_1 \hookrightarrow H$




We call the initial partial embedding of $H$ the *anchor* and, derived from this, we call the algorithm *anchored matching.* Note that we assume that the redex is one connected component. However, as discussed in the original presentation of the KaSim algorithm [12], this is not a problem, since the separate connected components of a redex can be matched separately and then combined ad hoc during simulation; see loc. cit. for details.

## 7.8.1 Algorithm

Due to time constraints, we shall only present a very naive algorithm, and only discuss possible optimizations. While this algorithm is too naive to be practical in an implementation, it is sufficient

for communicating our approach and it can probably serve as a nice and simple starting point for soundness and completeness proofs.

The algorithm builds on the following two ideas:

**fringe:** For a bigraph $G : \langle k, X\rangle \to \langle m, Y\rangle$ and a subset of its entities $S \subset V_G \uplus E_G \uplus k \uplus X \uplus m \uplus Y$, the *fringe* of $S$ in $G$ are the entities of $G$ that are adjacent to entities of $S$ but not in $S$, i.e.,

$$
\begin{aligned}
\mathit{fringe}(G,S) = \{c \in (k \uplus V_G) \setminus S \quad & \mid \exists p \in S : \mathit{prnt}(c) = p\} \\
\uplus \{p \in (V_G \uplus m) \setminus S \quad & \mid \exists c \in S : \mathit{prnt}(c) = p\} \\
\uplus \{l \in E_G \uplus Y \setminus S \quad & \mid \exists v \in S, i \in \mathbb{N} : \mathit{link}(p) = l \\
& \quad \vee \exists x \in S : \mathit{link}(x) = l\} \\
\uplus \{v \in V_G \setminus S \quad & \mid \exists l \in S, i \in \mathbb{N} : \mathit{link}(v,i) = l\} \\
\uplus \{x \in X \setminus S \quad & \mid \exists l \in S : \mathit{link}(x) = l\}.
\end{aligned}
$$

If $G$ consists of a single connected component and $S$ is non-empty, then if we keep expanding $S$ by entities in its fringe, eventually $S$ will cover $G$.

**valid extension:** For a partial embedding $\phi : G \hookrightarrow a$ into an agent $a : \langle m, Y\rangle$, an entity $s$ in the fringe of $\phi$ in $G$, and subset $T$ of the entities of $a$, i.e.,

$$
\begin{gathered}
s \in \mathit{fringe}(G, \mathsf{dom}(\phi)) \\
T \subseteq V_a \uplus E_a \uplus m \uplus Y
\end{gathered}
$$

the define the predicate $\mathit{validExt}(\phi, s \mapsto T)$ to be true when $\phi[s \mapsto T] : G \hookrightarrow a$ is also a partial embedding. Note that $T$ is a subset of the entities of $a$, since embeddings of sites and inner names map to subsets; for the other entities $T$ should be a singleton.

Note that the injectivity and structure preservation conditions on embeddings imply that $T$ must be on the fringe of $\phi$ in $a$.

The algorithm is listed in Algorithm 1. In brief, it works as follows:

**line 2:** If the fringe of $\phi$ in $G$ is empty, $\phi$ is total (since $G$ is a connected component) and thus we have found a match.

**line 6:** Otherwise, choose an entity $s$ on the fringe of $\phi$ in $G$ which shall be matched next.

**line 7:** For any possible mapping of $s$ to entities $T$ on the fringe of $\phi$ in $a$:

**line 8:** If $s \mapsto T$ is a valid extension of $\phi$

**line 9:** find all total extensions of $\phi[s \mapsto T]$.

The obvious places to optimize this algorithm are the choices of $s$ and $T$:

**choosing $T$:** It should be possible to only choose $T$'s such that $\mathit{validExt}(\phi, s \mapsto T)$. In particular, the structure preservation conditions on embeddings should guide the choice of $T$. For instance, if $s$ is a node, $T$ should be a singleton $\{t\}$ where $t$ is a node with $\mathit{ctrl}_G(s) = \mathit{ctrl}_a(t)$, and if $s$ is on the fringe of $\phi$ because $\mathit{prnt}_G(s) \in \mathsf{dom}(\phi)$, then $t \in \mathit{prnt}_a^{-1}(\phi(\mathit{prnt}_G(s)))$.

**choosing $s$:** The heuristic for choosing $s$ is critical for narrowing down the number of $T$'s we will have to explore for each $s$. We believe that a good strategy could be to choose an $s$ that is estimated to have a small number of possible embeddings.

For instance, note that if $s$ is on the fringe of $\phi$ because of one of its children $c$, i.e., $c \in \mathit{prnt}_G^{-1}(s) \cap \mathsf{dom}(\phi)$, then structure preservation dictates that the only choice is $T = \{\mathit{prnt}_a(\phi(c))\}$. Similarly, embeddings of nodes and inner names determine the embeddings of the connected links.

**Algorithm 1** The Anchored Matching algorithm

**Require:** $\phi : G \hookrightarrow a$ non-trivial, $G$ has one connected component

1: **procedure** ANCHORED-MATCHING($\phi : G \hookrightarrow a$)
2: **if** $\mathit{fringe}(G, \mathsf{dom}(\phi)) = \emptyset$ **then**
3: **return** $\{\phi\}$
4: **else**
5: $M \leftarrow \emptyset$
6: choose $s \in \mathit{fringe}(G, \mathsf{dom}(\phi))$
7: **for all** $T \subseteq \mathit{fringe}(a, \mathsf{rng}(\phi))$ **do**
8: **if** $\mathit{validExt}(\phi, s \mapsto T)$ **then**
9: $M \leftarrow M \cup$ ANCHORED-MATCHING($\phi[s \mapsto T]$)
10: **return** $M$

We therefore envision a representation of the fringe of $\phi$ in $G$ as a prioritized queue, where the priority of each entity is an estimation number of possible embeddings based on its adjacency relation to $\mathsf{dom}(\phi)$. Whenever the embedding is extended with an entity, the estimate of adjacent entities in the fringe should be updated.

Furthermore, note that an inner will only be on the fringe of $\phi$ in $G$ if the link is connected to is already mapped by $\phi$. Together with the embedding conditions, this means that extending an embedding by an inner name will only affect the choice of sibling inner names. In fact, there are very few restrictions on how such sibling inner names should be mapped, and it therefore seems reasonable to postpone matching of inner names to the end. Similarly, sites could also be postponed.

## 7.9 Conclusions and Future Work

In this report we have laid a firm, formal foundation for an implementation of stochastic bigraphs:

1. We have defined *stochastic parametric reactive systems*, an alternative foundation for the dynamic semantics of bigraphs which is amenable to implementation: support is handled explicitly, parametric reaction rules are first-class citizens, and the stochastic rates of an agent is determined by its matches. Furthermore, we have shown that stochastic parametric reactive systems have the same abstract reactions as Milner's reactive systems.

2. We have defined *bigraph embeddings* and shown that they are isomorphic to certain decompositions of bigraphs; in particular, embeddings of redexes into agents are isomorphic to matches. Furthermore, we have shown that embeddings of a solid bigraph are determined by support translations of its nodes.

3. We have proposed, and proven sound and complete, a set of minimal *edits* of parametric redexes which, when put in sequence to form *edit scripts*, are equivalent to parametric reaction rules and generate the same abstract reactions.

4. We have outlined a characterization of causality and conflict for linear parametric reaction rules, based on pullbacks in the category of bigraph embeddings.

5. We have given a localized matching algorithm: starting from a partial match, i.e., a partial embedding, of a connected component, it finds all completions.

The presented work is part of an effort to build an efficient and scalable simulator for stochastic bigraphs: the Stochastic Bigraphical Abstract Machine. So far a prototype based on SPRSs,

embeddings, edit scripts, and anchored matching have been implemented. It allows stochastic simulation of certain BRSs: all controls must be active, reaction rules must be linear, and redexes must be solid and consist of a single connected component.

### 7.9.1 Future Work

Localized matching and causality analysis of rules should be investigated in more detail. In particular, we must prove our conjectures about the embedding categories and soundness and completeness of anchored matching. Furthermore, it it unclear whether the pullback approach to characterizing causality and conflict will (a) result in a practical algorithm, and (b) generalize to non-linear reaction rules.

Our presentation of bigraph embeddings, and the related proofs, could probably be simplified by using a formulation of concrete bigraphs where roots and sites are named (as we did in our development of edit scripts). In fact, we believe that, for many purposes, the theory of bigraphs would be simpler to work with if roots and sites were named.

From an implementation perspective, there is a need for representing sets of embeddings efficiently: in biological models there will often be a large number of embeddings of each redex. This needs further investigation, but as a first step we believe the following conjecture may prove useful: embeddings of solid bigraphs are determined by the support translation of the leaves of the place graph.

For the biological simulation scenarios we have in mind, we expect the user to provide edit scripts as they provide a natural way to express protein-protein interaction as well as dynamic compartmentalization. However, there might be applications where the user would prefer to provide reaction rules and have the system infer suitable edit scripts. While we have given a simple construction of edit scripts for any parametric reaction rule, it is very naive and assumes that there is no relation between nodes and edges of redex and reactum, resulting in inefficient simulation. It should therefore be investigated how one can derive better edit scripts. This seems related to the tree edit distance problem, where one wishes to find a minimal edit script that transforms one tree into another [5].

## 7.10 Bibliography


[1] G. Bacci, D. Grohmann, and M. Miculan. Bigraphical models for protein and membrane interactions. In *Proceedings of the Third International Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC 2009)*, pages 3–18. EPTCS 11, 2009.

[2] G. Bacci, D. Grohmann, and M. Miculan. Dbtk: A toolkit for directed bigraphs. In *CALCO*, pages 413–422, 2009.

[3] M. Beauquier and C. Schürmann. A bigraph reactive systems realtion model. Technical Report TR-2010-126, IT University of Copenhagen, June 2010.

[4] BigMC. BigMC – Bigraphical Model Checker. `http://bigraph.org/bigmc/`.

[5] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:217–239, June 2005.

[6] L. Birkedal, T. C. Damgaard, A. J. Glenstrup, and R. Milner. Matching of bigraphs. *Electronic Notes in Theoretical Computer Science*, 175(4):3–19, 2007.

[7] BPLTool. BPL Tool. `http://www.itu.dk/research/pls/wiki/index.php/BPL_Tool`.

[8] G. L. Cattani, J. J. Leifer, and R. Milner. Contexts and embeddings for closed shallow action graphs. Technical Report UCAM-CL-TR-496, University of Cambridge, Computer Laboratory, July 2000.

[9] T. C. Damgaard and J. Krivine. A generic language for biological systems based on bigraphs. Technical Report TR-2008-115, IT University of Copenhagen, December 2008.

[10] T. C. Damgaard, V. Danos, and J. Krivine. A language for the cell. Technical Report TR-2008-116, IT University of Copenhagen, December 2008.

[11] V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325, 2004.

[12] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable simulation of cellular signaling networks. In *Proceedings of the 5th Asian conference on Programming languages and systems*, APLAS'07, pages 139–157. Springer-Verlag, 2007.

[13] S. Debois. Computation in the informatic jungle. Technical Report TR-2011-147, IT University of Copenhagen, 2011. (forthcoming).

[14] N. Eén and N. Sörensson. MiniSAT. `http://minisat.se`.

[15] H. Ehrig. Bigraphs meet double pushouts. *Bulletin of the EATCS*, 78:72–85, 2002.

[16] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of graph grammars and computing by graph transformation, Volume 3: Concurrency, Parallelism, and Distribution.* World Scientific Publishing Co., Inc., 1999. ISBN 9-810240-21-X.

[17] A. Faithfull. Big Red. `http://www.itu.dk/research/pls/wiki/index.php/Big_Red`, 2010.

[18] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[19] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[20] A. J. Glenstrup, T. C. Damgaard, L. Birkedal, and E. Højsgaard. An implementation of bigraph matching. Technical Report TR-2010-135, IT University of Copenhagen, December 2010.

[21] C. Greenhalgh. bigraphspace. `http://bigraphspace.svn.sourceforge.net/`, 2009.

[22] J. Hillston. *A compositional approach to performance modelling.* Cambridge University Press, 1996. ISBN 0-521-57189-8.

[23] E. Højsgaard and A. J. Glenstrup. The BPL Tool: A tool for experimenting with bigraphical reactive systems. Technical Report TR-2011-145, IT University of Copenhagen, October 2011.

[24] O. H. Jensen and R. Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge – Computer Laboratory, February 2004.

[25] J. Krivine, R. Milner, and A. Troina. Stochastic bigraphs. *Electronic Notes in Theoretical Computer Science*, 218:73 – 96, 2008. ISSN 1571-0661. Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV).

[26] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[27] R. Milner. Axioms for bigraphical structure. *Journal of Mathematical Structures in Computer Science*, 15(6):1005–1032, 2005.

[28] R. Milner. Embeddings and contexts for link graphs. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 343–351. Springer Berlin / Heidelberg, 2005.

[29] R. Milner. *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.

[30] PEPAPlugIn. The PEPA Plug-in Project. `http://www.dcs.ed.ac.uk/pepa/tools/plugin/index.html`.

[31] C. Priami. Stochastic $\pi$-calculus. *Computer Journal*, 38(7):578–589, 1995.

[32] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation, Volume 1: Foundations.* World Scientific Publishing Co., Inc., 1997. ISBN 98-102288-48.

[33] A. Schack-Nielsen and C. Schürmann. Celf - a logical framework for deductive and concurrent systems (system description). In *IJCAR*, pages 320–326, 2008.

[34] M. Sevegnani, C. Unsworth, and M. Calder. A SAT based algorithm for the matching problem in bigraphs with sharing. Technical Report TR-2010-311, University of Glasgow, Department of Computing Science, 2010.

[35] P. Sobocinsky. Relative pushouts in graphical reactive systems. February 2002.

# 7.A Proofs

## 7.A.1 Bigraph Embeddings

### Proof of Prop. 7.5.3

$\phi^{\mathsf{e}}$: Construct the map of each edge $e \in E_G$ as follows: choose a port $p = (v, i) \in link_G^{-1}(e)$, which is always possible since no edge is idle and every inner name is guarding, and let

$$\phi^{\mathsf{e}}(e) = link_H(\phi^{\mathsf{v}}(v), i).$$

By construction it satisfies condition (LGE-9); it must satisfy the other conditions since $\phi$ is an embedding and $\phi^{\mathsf{e}}$ is the only embedding of edges that will satisfy condition (LGE-9): To see that $\phi^{\mathsf{e}}$ is unique, assume that there is a different $\phi'^{\mathsf{e}}$, i.e., $\phi^{\mathsf{e}}(e) \neq \phi'^{\mathsf{e}}(e)$ for some $e \in E_G$. Since they both satisfy condition (LGE-9), the following must hold for the port $p$ we chose when defining $\phi^{\mathsf{e}}(e)$:

$$\begin{aligned} \phi^{\mathsf{e}}(e) &= (\phi^{\mathsf{e}} \circ link_G)(p) \\ &= (link_H \circ \phi^{\mathsf{p}})(p) \\ &= (\phi'^{\mathsf{e}} \circ link_G)(p) = \phi'^{\mathsf{e}}(e) \end{aligned}$$

which contradicts our assumption that $\phi^{\mathsf{e}}$ and $\phi'^{\mathsf{e}}$ are different.

$\phi^{\mathsf{i}}$: Construct the map of each inner name $x \in X_G$ as follows:

$$\begin{aligned} \phi^{\mathsf{i}}(x) &= points_{H,x} \setminus \phi^{\mathsf{p}}(P_{G,x}) \\ points_{H,x} &= (link_H^{-1} \circ \phi^{\mathsf{e}})(link_G(x)) \\ P_{G,x} &= (link_G^{-1} \circ link_G)(x) \setminus \{x\} \\ \phi^{\mathsf{p}}(v, i) &= (\phi^{\mathsf{v}}(v), i). \end{aligned}$$

This is well-defined since no outer name of $G$ is linked to an inner name, thus $link_G(x) \in E_G$, and no inner names are siblings. By construction it satisfies condition (LGE-7); it must satisfy the other conditions since $\phi$ is an embedding and $\phi^{\mathsf{i}}$ is the only embedding of inner names that will satisfy condition (LGE-7): To see that $\phi^{\mathsf{i}}$ is unique, assume that there is a different $\phi'^{\mathsf{i}}$ that satisfies the conditions of Def. 7.5.1, i.e., there must be some $x \in X_G$ and $p \in X_H \uplus P_H$ with $p \in \phi^{\mathsf{i}}(x), p \notin \phi'^{\mathsf{i}}(x)$ (or vice versa). Since they both satisfy condition (LGE-7) and $G$ no outer name is linked to an inner name we have:

$$\begin{aligned} link_G(x) &\in E_G \\ (\phi^{\mathsf{p}} \circ link_G^{-1}\restriction_{E_G})(link_G(x)) &= (link_H^{-1} \circ \phi^{\mathsf{e}})(link_G(x)) \\ &= (\phi'^{\mathsf{p}} \circ link_G^{-1}\restriction_{E_G})(link_G(x)) \end{aligned}$$

where

$$\phi^{\mathsf{p}}(p') = \begin{cases} (\phi^{\mathsf{v}}(v), i) & \text{if } p' = (v,i) \in P_G \\ \phi^{\mathsf{i}}(p') & \text{if } p' \in X_G \end{cases}$$

$$\phi'^{\mathsf{p}}(p') = \begin{cases} (\phi^{\mathsf{v}}(v), i) & \text{if } p' = (v,i) \in P_G \\ \phi'^{\mathsf{i}}(p') & \text{if } p' \in X_G \end{cases}$$

And since $p \in \phi^{\mathsf{i}}(x)$, and $\phi^{\mathsf{p}}(p')$ and $\phi'^{\mathsf{p}}(p')$ agree on ports, we must have $p \in \phi'^{\mathsf{i}}(x')$ for some $x' \in link_G^{-1}\restriction_{E_G}(link_G(x))$. But no inner names are siblings, so $x' = x$ and thus $p \in \phi'^{\mathsf{i}}(x')$ which contradicts our assumption that $p \notin \phi'^{\mathsf{i}}(x)$.

$\phi^{\mathsf{o}}$: Construct the map of each outer name $y \in Y_G$ as follows: choose a port $p = (v,i) \in link_G^{-1}(y)$, which is always possible since no outer name is idle or connected to an inner name, and let

$$\phi^{\mathsf{o}}(y) = link_H(\phi^{\mathsf{v}}(v), i).$$

By construction it satisfies condition (LGE-9); it must satisfy the other conditions since $\phi$ is an embedding and $\phi^{\mathsf{o}}$ is the only embedding of outer names that will satisfy condition (LGE-9): To see that $\phi^{\mathsf{o}}$ is unique, assume that there is a different $\phi'^{\mathsf{o}}$, i.e., $\phi^{\mathsf{o}}(y) \neq \phi'^{\mathsf{o}}(y)$ for some $y \in Y_G$. Since they both satisfy condition (LGE-9), the following must hold for the port $p$ we chose when defining $\phi^{\mathsf{o}}(y)$:

$$\begin{aligned} \phi^{\mathsf{o}}(y) &= (\phi^{\mathsf{o}} \circ link_G)(p) \\ &= (link_H \circ \phi^{\mathsf{p}})(p) \\ &= (\phi'^{\mathsf{o}} \circ link_G)(p) = \phi'^{\mathsf{o}}(y) \end{aligned}$$

which contradicts our assumption that $\phi^{\mathsf{o}}$ and $\phi'^{\mathsf{o}}$ are different. □

### Proof of Prop. 7.5.6

From the definitions of support translation, composition, and tensor product we have:

$$\begin{aligned} V_H &= V_C \uplus V_G \uplus V_D & C &: m_G + k \to m_H \\ ctrl_H &= ctrl_C \uplus ctrl_G \uplus ctrl_D & D &: k_D \to k_G \\ k_D &\subseteq k_H \end{aligned}$$

To show that $\phi$ is an embedding we need to express the parent map of $H$ in terms of it decomposition $C \circ (G \circ D \otimes \mathsf{id}_k) \circ \pi$. We construct the map incrementally according to the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8)

**$G \circ D$:** We write $prnt_1$ for the parent map of the resulting place graph:

$$prnt_1(w) = \begin{cases} prnt_D(w) & \text{if } w \in k_D \uplus V_D \text{ and } prnt_D(w) \in V_D \\ prnt_G(j) & \text{if } w \in k_D \uplus V_D \text{ and } prnt_D(w) = j \in k_G \\ prnt_G(w) & \text{if } w \in V_G \end{cases}$$

**$G \circ D \otimes \mathsf{id}_k$:** We write $prnt_2$ for the parent map of the resulting place graph:

$$\begin{aligned} prnt'_{\mathsf{id}_k}(k_D + i) &= m_G + i \qquad (i \in k) \\ prnt_2(w) &= (prnt_1 \uplus prnt'_{\mathsf{id}_k})(w) \\ &= \begin{cases} prnt_D(w) & \text{if } w \in k_D \uplus V_D \text{ and } prnt_D(w) \in V_D \\ prnt_G(j) & \text{if } w \in k_D \uplus V_D \text{ and } prnt_D(w) = j \in k_G \\ prnt_G(w) & \text{if } w \in V_G \\ m_G - k_D + w & \text{if } w \in (k_D + k) \setminus k_D \end{cases} \end{aligned}$$

**$(G \circ D \otimes \mathsf{id}_k) \circ \pi$:** We write $prnt_3$ for the parent map of the resulting place graph:

$$\begin{aligned} prnt_3(w) &= \begin{cases} prnt_2(\pi(w)) & \text{if } w \in k_H \\ prnt_2(w) & \text{if } w \in V_G \uplus V_D \end{cases} \\ &= \begin{cases} prnt_D(\pi(w)) & \text{if } w \in k_H \text{ and } \pi(w) \in k_D \text{ and } prnt_D(\pi(w)) \in V_D \\ prnt_G(j) & \text{if } w \in k_H \text{ and } \pi(w) \in k_D \text{ and } prnt_D(\pi(w)) = j \in k_G \\ m_G - k_D + \pi(w) & \text{if } w \in k_H \text{ and } \pi(w) \in (k_D + k) \setminus k_D \\ prnt_D(w) & \text{if } w \in V_D \text{ and } prnt_D(w) \in V_D \\ prnt_G(j) & \text{if } w \in V_D \text{ and } prnt_D(w) = j \in k_G \\ prnt_G(w) & \text{if } w \in V_G \end{cases} \end{aligned}$$

$H = C \circ (G \circ D \otimes \mathsf{id}_k) \circ \pi$:

$$prnt_H(w) = \begin{cases} prnt_3(w) & \text{if } w \in k_H \uplus V_G \uplus V_D \text{ and } prnt_3(w) \in V_G \uplus V_D \\ prnt_C(j) & \text{if } w \in k_H \uplus V_G \uplus V_D \text{ and } prnt_3(w) = j \in k_C \\ prnt_C(w) & \text{if } w \in V_C \end{cases}$$

$$= \begin{cases} prnt_D(\pi(w)) & \text{if } w \in k_H \text{ and } \pi(w) \in k_D \\ & \quad \text{and } prnt_D(\pi(w)) \in V_D \\ prnt_G(j) & \text{if } w \in k_H \text{ and } \pi(w) \in k_D \\ & \quad \text{and } prnt_D(\pi(w)) = j \in k_G \\ & \quad \text{and } prnt_G(j) \in V_G \\ prnt_D(w) & \text{if } w \in V_D \text{ and } prnt_D(w) \in V_D \\ prnt_G(j) & \text{if } w \in V_D \text{ and } prnt_D(w) = j \in k_G \\ & \quad \text{and } prnt_G(j) \in V_G \\ prnt_G(w) & \text{if } w \in V_G \text{ and } prnt_G(w) \in V_G \\ prnt_C(j) & \text{if } w \in k_H \text{ and } \pi(w) \in k_D \\ & \quad \text{and } prnt_D(\pi(w)) = i \in k_G \\ & \quad \text{and } prnt_G(i) = j \in m_G \\ prnt_C(m_G - k_D + \pi(w)) & \text{if } w \in k_H \text{ and } \pi(w) \in (k_D + k) \setminus k_D \\ prnt_C(j) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in k_G \\ & \quad \text{and } prnt_G(i) = j \in m_G \\ prnt_C(j) & \text{if } w \in V_G \text{ and } prnt_G(w) = j \in m_G \\ prnt_C(w) & \text{if } w \in V_C \end{cases}$$

We can now verify that $\phi$ is a place graph embedding, i.e., that it satisfies the conditions of Def. 7.5.4:

**(PGE-1)** Satisfied since $\mathsf{Id}_{V_G}$ is an identity map.

**(PGE-2)** Since $prnt_D : k_D \uplus V_D \to V_D \uplus k_G$, $k_D \subseteq k_H$, $V_D \subseteq V_H$, and $\pi : k_H \to k_H$ we have $\phi^{\mathsf{s}} = (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G} : k_G \to \mathcal{P}(k_H \uplus V_H)$; it is fully injective since $prnt_D$, $\mathsf{Id}_{V_D}$, and $\pi$ are functions.

**(PGE-3)** Since $prnt_C : (m_G + k) \uplus V_C \to V_C \uplus m_H$ and $V_C \subseteq V_H$ we have $\phi^{\mathsf{r}} = prnt_C \restriction_{m_G} : m_G \to V_H \uplus m_H$.

**(PGE-4)** Satisfied since $\mathsf{rng}(\phi^{\mathsf{v}}) = V_G$, $\mathsf{rng}(\phi^{\mathsf{r}}) = \mathsf{rng}(prnt_C \restriction_{m_G}) \subseteq V_C \uplus m_H$, and $V_G \# (V_C \uplus m_H)$.

**(PGE-5)** Satisfied since $\mathsf{rng}(\phi^{\mathsf{s}}) = \mathsf{rng}((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G}) \subseteq k_H \uplus V_D$, $\mathsf{rng}(\phi^{\mathsf{v}}) = V_G$, and $V_G \# (k_H \uplus V_D)$.

**(PGE-6)** We have $H \downharpoonleft^{\phi^{\mathsf{s}}(k_G)} \subseteq k_H \uplus V_D$ which can be seen as follows (noting that $prnt_H(w) \in V_D \Rightarrow w \in k_H \uplus V_D$):

$$\begin{aligned} H \downharpoonleft^{\phi^{\mathsf{s}}(k_G)} &= \{c' \mid c' \in k_H \uplus V_H \ \wedge \ \exists i \geq 0 : prnt_H^i(c') \in \phi^{\mathsf{s}}(k_G)\} \\ &= \{c' \mid c' \in k_H \uplus V_H \ \wedge \ \exists i \geq 0 : prnt_H^i(c') \in ((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1})(k_G)\} \\ &= \{c' \mid c' \in k_H \uplus V_H \ \wedge \ \exists i \geq 0 : prnt_H^i(c') \in k_H \uplus V_D\} \\ &= \{c' \mid c' \in k_H \uplus V_D \ \wedge \ \exists i \geq 0 : prnt_H^i(c') \in V_D\} \\ &\subseteq k_H \uplus V_D \end{aligned}$$

Since $\mathsf{rng}(\phi^{\mathsf{r}}) = \mathsf{rng}(prnt_C \restriction_{m_G}) \subseteq V_C \uplus m_H$ and $(k_H \uplus V_D) \# (V_C \uplus m_H)$ the condition is satisfied.

**(PGE-7)** Satisfied since we have the following equalities:

$$\begin{aligned}(\phi^{\mathsf{c}} \circ prnt_G^{-1} \restriction_{V_G})(w) &= \phi^{\mathsf{c}}(prnt_G^{-1}(w)) \\ &= \phi^{\mathsf{s}}(k_G \cap prnt_G^{-1}(w)) \\ &\quad \cup \phi^{\mathsf{v}}(V_G \cap prnt_G^{-1}(w)) \\ &= ((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G})(k_G \cap prnt_G^{-1}(w)) \\ &\quad \cup (V_G \cap prnt_G^{-1}(w)) \\ &= \{i \in \pi^{-1}(k_D) \mid prnt_D(\pi(i)) = j \in k_G \text{ and } prnt_G(j) = w\} \\ &\quad \cup \{v \in V_D \mid prnt_D(v) = j \in k_G \text{ and } prnt_G(j) = w\} \\ &\quad \cup \{v \in V_G \mid prnt_G(v) = w\} \\ &= prnt_H^{-1}(w) \\ &= (prnt_H^{-1} \circ \mathsf{Id}_{V_G})(w) \\ &= (prnt_H^{-1} \circ \phi^{\mathsf{v}})(w).\end{aligned}$$

**(PGE-8)** Satisfied since $\phi^{\mathsf{v}} = \mathsf{id}_{V_G}$ and $ctrl_G \subseteq ctrl_H$.

**(PGE-9)** We check the condition separately for the nodes and sites:

$v \in V_G$**:** We check the condition separately for the cases where the parent is a node or a root:

$prnt_G(v) \in V_G$**:**

$$\begin{aligned}(\phi^{\mathsf{f}} \circ prnt_G)(v) &= (\phi^{\mathsf{v}} \circ prnt_H)(v) \\ &= (\mathsf{Id}_{V_G} \circ prnt_H)(v) \\ &= (prnt_H \circ \mathsf{Id}_{V_G})(v) \\ &= (prnt_H \circ \phi^{\mathsf{c}})(v).\end{aligned}$$

$prnt_G(v) \in m_G$**:**

$$\begin{aligned}(\phi^{\mathsf{f}} \circ prnt_G)(v) &= (\phi^{\mathsf{r}} \circ prnt_G)(v) \\ &= (prnt_C \circ prnt_G \circ \mathsf{Id}_{V_G})(v) \\ &= (prnt_H \circ \phi^{\mathsf{c}})(v).\end{aligned}$$

$i \in k_G$**:** We check the condition separately for the cases where the parent is a node or a root:

$prnt_G(i) \in V_G$**:**

$$\begin{aligned}(\phi^{\mathsf{f}} \circ prnt_G)(i) &= (\phi^{\mathsf{v}} \circ prnt_G)(i) \\ &= (\mathsf{Id}_{V_G} \circ prnt_G)(i) \\ &= prnt_G(i) \\ &= prnt_H(((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G})(i)) \\ &= (prnt_H \circ \phi^{\mathsf{s}})(i) \\ &= (prnt_H \circ \phi^{\mathsf{c}})(i).\end{aligned}$$

$prnt_G(i) \in m_G$:

$$\begin{aligned}(\phi^{\mathsf{f}} \circ prnt_G)(i) &= (\phi^{\mathsf{r}} \circ prnt_G)(i) \\ &= (prnt_C \circ prnt_G)(i) \\ &= prnt_H(((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G})(i)) \\ &= (prnt_H \circ \phi^{\mathsf{s}})(i) \\ &= (prnt_H \circ \phi^{\mathsf{c}})(i).\end{aligned}$$

□

**Proof of Prop. 7.5.9**

We first show that $prmt(\phi)$ and $ctxt(\phi)$ are indeed place graphs:

$prmt(\phi)$ : Clearly, $ctrl_H \restriction_{V_D}$ is a control map defined for $V_D$. We must check that the parent map $prnt_D : k_D \uplus V_D \to V_D \uplus k_G$ is (1) well-defined and (2) acyclic:

1. Since $\phi^{\mathsf{s}} : k_G \to \mathcal{P}(k_H \uplus V_H)$ is fully injective, $(\phi^{\mathsf{s}})^{-1}$ is a function and thus $prnt_D$ is clearly well-defined.

2. It is immediate that $prnt_D$ is acyclic iff $\forall c \in k_D \uplus V_D : \exists i > 0 : prnt_D^i(c) \in k_G$. This is clearly the case for the elements of $(f_D \uplus \mathsf{Id}_{V_D})^{-1}(\mathsf{dom}((\phi^{\mathsf{s}})^{-1}))$ and for the remaining elements it follows from the definition of the subtree operator, cf. Def. 7.2.9, and the fact that $\mathsf{rng}(prnt_H) \# k_D$.

$ctxt(\phi)$ : Clearly, $ctrl_H \restriction_{V_C}$ is a control map defined for $V_C$. We must check that the parent map $prnt_C : k_C \uplus V_C \to V_C \uplus m_H$ is (1) well-defined and (2) acyclic:

1. The constituent functions have the following domains and codomains:

$$\begin{aligned}\phi^{\mathsf{r}} &: m_G \to V_H \uplus m_H \\ prnt_H \circ f_C &: \{i + m_G \mid i \in |\tilde{k}_C|\} \to V_C \uplus m_H \\ prnt_H \restriction_{V_C} &: V_C \to V_C \uplus m_H\end{aligned}$$

   Since $k_C = m_G + |\tilde{k}_C|$ it is clear that $prnt_C$ is well-defined, but we must show $\mathsf{rng}(\phi^{\mathsf{r}}) \subseteq V_C \uplus m_H$ to know $\mathsf{cod}(prnt_C) = V_C \uplus m_H$. Since $V_C = (V_H \setminus \phi^{\mathsf{v}}(V_G)) \setminus V_D$ and $V_D = V_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}$ this amounts to showing $\mathsf{rng}(\phi^{\mathsf{r}}) \# \mathsf{rng}(\phi^{\mathsf{v}})$ and $\mathsf{rng}(\phi^{\mathsf{r}}) \# H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}$, which follows from the fact that $\phi$ is an embedding and thus satisfies conditions (PGE-4) and (PGE-6).

2. Since $prnt_H$ is acyclic and $\mathsf{rng}(prnt_H) \# k_C$, $prnt_C$ is acyclic.

To see that any valid choices of $f_D$ and $f'_C$ yield equivalent decompositions, cf. Def. 7.5.7, assume that we have two other bijections

$$g_D : k_D \rightarrowtail \tilde{k}_D \qquad\qquad g'_C : |\tilde{k}_C| \rightarrowtail \tilde{k}_C$$

and construct the corresponding parent maps $prnt_{D'}$ and $prnt_{C'}$ and permutation $\pi'$

$$\begin{aligned}
prnt_{D'} &= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (g_D \uplus \mathsf{Id}_{V_D}) \\
g_C(i + m_G) &= g'_C(i) \quad \text{for } i \in |\tilde{k}_C| \\
prnt_{C'} &= \phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ g_C \\
g'(i + k_D) &= g'_C(i) \quad \text{for } i \in |\tilde{k}_C| \\
\pi' &= g_D^{-1} \uplus g'^{-1} : k_H \to k_H.
\end{aligned}$$

Finally, we check the equivalence conditions

$$\begin{aligned}
prnt_D \restriction_{V_D} &= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D}) \restriction_{V_D} \\
&= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \restriction_{V_D} \\
&= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (g_D \uplus \mathsf{Id}_{V_D}) \restriction_{V_D} \\
&= prnt_{D'} \restriction_{V_D} \\
prnt_C \restriction_{V_C \uplus m_G} &= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_C) \restriction_{V_C \uplus m_G} \\
&= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C}) \restriction_{V_C \uplus m_G} \\
&= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ g_C) \restriction_{V_C \uplus m_G} \\
&= prnt_{C'} \restriction_{V_C \uplus m_G} \\
prnt_D \circ \pi \downharpoonleft^{k_D} &= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D}) \circ \pi \downharpoonleft^{k_D} \\
&= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D}) \circ (f_D^{-1} \uplus f'^{-1}) \downharpoonleft^{k_D} \\
&= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \restriction_{\tilde{k}_D} \\
&= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (g_D \uplus \mathsf{Id}_{V_D}) \circ (g_D^{-1} \uplus g'^{-1}) \downharpoonleft^{k_D} \\
&= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (g_D \uplus \mathsf{Id}_{V_D}) \circ \pi' \downharpoonleft^{k_D} \\
&= prnt_{D'} \circ \pi' \downharpoonleft^{k_D} \\
prnt_C(\pi(i) - k_D + m_G) &= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_C)(\pi(i) - k_D + m_G) \\
&= prnt_H(f_C((f_D^{-1} \uplus f'^{-1})(i) - k_D + m_G)) \\
&= prnt_H(f'_C(f'^{-1}(i) - k_D)) \\
&= prnt_H(f'_C({f'_C}^{-1}(i))) \\
&= prnt_H(i) \\
&= prnt_H(g'_C({g'_C}^{-1}(i))) \\
&= prnt_H(g'_C(g'^{-1}(i) - k_D)) \\
&= prnt_H(g_C((g_D^{-1} \uplus g'^{-1})(i) - k_D + m_G)) \\
&= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ g_C)(\pi'(i) - k_D + m_G) \\
&= prnt_{C'}(\pi'(i) - k_D + m_G).
\end{aligned}$$

We now show that $prmt(\phi)$ and $ctxt(\phi)$ are indeed parameter and context for the embedding of $G$:

Let

$$\begin{aligned}
D : k_D \to k_G &= prmt(\phi), \\
C : k_C \to m_H &= ctxt(\phi), and \\
(V, ctrl, prnt) &= ctxt(\phi) \circ (\phi \centerdot G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|}) \circ \pi.
\end{aligned}$$

By the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8) we have the following equalities:

$$\begin{aligned} V &= V_C \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D && \text{Defs. 7.2.6 and 7.2.8} \\ &= ((V_H \setminus \phi^{\mathsf{v}}(V_G)) \setminus V_D) \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D && \text{Def. 7.5.8} \\ &= V_H \end{aligned}$$

$$\begin{aligned} ctrl &= ctrl_H \restriction_{V_C} \uplus ctrl_G \circ (\phi^{\mathsf{v}})^{-1} \uplus ctrl_H \restriction_{V_D} && \text{Defs. 7.2.6 and 7.2.8} \\ &= ctrl_H \restriction_{V_C} \uplus ctrl_H \circ \phi^{\mathsf{v}} \circ (\phi^{\mathsf{v}})^{-1} \uplus ctrl_H \restriction_{V_D} && \text{Condition (PGE-8)} \\ &= ctrl_H \restriction_{V_C} \uplus ctrl_H \restriction_{\phi^{\mathsf{v}}(V_G)} \uplus ctrl_H \restriction_{V_D} && \phi^{\mathsf{v}} \text{ is injective on } V_G \\ &= ctrl_H && V_H = V_C \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D \end{aligned}$$

We construct the parent map *prnt* incrementally according to the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8), and then verify $prnt(w) = prnt_H(w)$:

$\phi \mathbin{\scriptstyle\blacksquare} G$**:** We write $prnt_{\phi \scriptstyle\blacksquare G}$ for the parent map of the resulting place graph:

$$prnt_{\phi \scriptstyle\blacksquare G} = (\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ ((\phi^{\mathsf{v}})^{-1} \uplus \mathsf{Id}_{k_G}).$$

$\phi \mathbin{\scriptstyle\blacksquare} G \circ prmt(\phi)$**:** We write $prnt_1$ for the parent map of the resulting place graph:

$$prnt_1(w) = \begin{cases} prnt_D(w) & \text{if } w \in k_D \uplus V_D \text{ and } prnt_D(w) \in V_D \\ prnt_{\phi \scriptstyle\blacksquare G}(j) & \text{if } w \in k_D \uplus V_D \text{ and } prnt_D(w) = j \in k_G \\ prnt_{\phi \scriptstyle\blacksquare G}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_G) \end{cases}.$$

$\phi \mathbin{\scriptstyle\blacksquare} G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|}$**:** We write $prnt_2$ for the parent map of the resulting place graph:

$$prnt_2 = prnt_1 \uplus prnt'_{\mathsf{id}_{|\tilde{k}_C|}}$$

where

$$prnt'_{\mathsf{id}_{|\tilde{k}_C|}}(k_D + i) = m_G + i \quad \text{for } i \in |\tilde{k}_C|.$$

$(\phi \mathbin{\scriptstyle\blacksquare} G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|}) \circ \pi$**:** We write $prnt_3$ for the parent map of the resulting place graph:

$$\begin{aligned} prnt_3(w) &= \begin{cases} \pi(w) & \text{if } w \in k_H \uplus \emptyset \text{ and } \pi(w) \in \emptyset \\ prnt_2(j) & \text{if } w \in k_H \uplus \emptyset \text{ and } \pi(w) = j \in k_H \\ prnt_2(w) & \text{if } w \in \phi^{\mathsf{v}}(V_G) \uplus V_D \end{cases} \\ &= \begin{cases} prnt_2(j) & \text{if } w \in k_H \text{ and } \pi(w) = j \in k_H \\ prnt_1(w) & \text{if } w \in \phi^{\mathsf{v}}(V_G) \uplus V_D \end{cases} \\ &= \begin{cases} prnt_1(j) & \text{if } w \in k_H \text{ and } \pi(w) = j \in k_D \\ prnt'_{\mathsf{id}_{|\tilde{k}_C|}}(j) & \text{if } w \in k_H \text{ and } \pi(w) = j \in k_H \setminus k_D \\ prnt_1(w) & \text{if } w \in \phi^{\mathsf{v}}(V_G) \uplus V_D \end{cases} \\ &= \begin{cases} prnt_1(j) & \text{if } w \in k_H \text{ and } \pi(w) = j \in k_D \\ j + m_G - k_D & \text{if } w \in k_H \text{ and } \pi(w) = j \in k_H \setminus k_D \\ prnt_1(w) & \text{if } w \in \phi^{\mathsf{v}}(V_G) \uplus V_D \end{cases} \end{aligned}$$

$ctxt(\phi) \circ (\phi \bullet G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|}) \circ \pi$**:** Finally, we have

$$prnt(w) = \begin{cases} prnt_3(w) & \text{if } w \in k_H \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D \text{ and } prnt_3(w) \in \phi^{\mathsf{v}}(V_G) \uplus V_D \\ prnt_C(j) & \text{if } w \in k_H \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D \text{ and } prnt_3(w) = j \in k_C \\ prnt_C(w) & \text{if } w \in V_C \end{cases}.$$

Let us now verify $prnt = prnt_H$. $prnt$ is defined for $w \in k_H \uplus V_C \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D$ and so is $prnt_H$ since $V_C \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D = V_H$, so let us consider $prnt(w)$ in each case (noting that $k_H = \tilde{k}_C \uplus \tilde{k}_D$):

$w \in \tilde{k}_C$**:** This implies

$$\begin{aligned} & & \pi(w) &= f'^{-1}(w) \\ \Leftrightarrow & & k_D &\leq \pi(w) < k_D + |\tilde{k}_C| \\ \Rightarrow & & prnt_3(w) &= \pi(w) + m_G - k_D \text{ and } m_G \leq prnt_3(w) < m_G + |\tilde{k}_C| \\ \Rightarrow & & prnt(w) &= prnt_C(\pi(w) + m_G - k_D) \\ & & &= (prnt_H \circ f_C)(\pi(w) + m_G - k_D) \\ & & &= (prnt_H \circ f_C)(f'^{-1}(w) + m_G - k_D) \\ & & &= (prnt_H \circ f_C)((f'_C)^{-1}(w) + m_G) \\ & & &= (prnt_H \circ f_C)((f_C)^{-1}(w)) \\ & & &= prnt_H(w) \end{aligned}$$

$w \in \tilde{k}_D$**:** This implies

$$\begin{aligned} & & \pi(w) &= (f_D)^{-1}(w) \in k_D \\ \Rightarrow & & prnt_3(w) &= prnt_1(\pi(w)) \end{aligned}$$

which further divides into two cases:

$prnt_D(\pi(w)) \in V_D$**:** This implies

$$\begin{aligned} & & prnt_1(\pi(w)) &= prnt_D(\pi(w)) \in V_D \\ \Rightarrow & & prnt(w) &= prnt_3(w) = prnt_D(\pi(w)) \\ & & &= prnt_D((f_D)^{-1}(w)) \\ & & &= (((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D) \setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \\ & & &\qquad \circ (f_D \uplus \mathsf{Id}_{V_D}))((f_D)^{-1}(w)) \\ & & &= prnt_H(w) \end{aligned}$$

$prnt_D(\pi(w)) \in k_G$**:** This implies

$$\begin{aligned} prnt_1(\pi(w)) &= prnt_{\phi \bullet G}(prnt_D(\pi(w))) \\ &= prnt_{\phi \bullet G}(prnt_D((f_D)^{-1}(w))) \\ &= prnt_{\phi \bullet G}((((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D) \setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \\ &\qquad \circ (f_D \uplus \mathsf{Id}_{V_D}))((f_D)^{-1}(w))) \\ &= prnt_{\phi \bullet G}((\phi^{\mathsf{s}})^{-1}(w)) \end{aligned}$$

which again divides into two cases:

$prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \in \phi^{\mathsf{v}}(V_G)$: This implies

$$\begin{aligned} prnt(w) = prnt_3(w) = prnt_1(\pi(w)) &= prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \\ &= ((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ ((\phi^{\mathsf{v}})^{-1} \uplus \mathsf{Id}_{k_G}))((\phi^{\mathsf{s}})^{-1}(w)) \\ &= (\phi^{\mathsf{v}} \circ prnt_G \downharpoonleft^{V_G})((\phi^{\mathsf{s}})^{-1}(w)) \\ &= (prnt_H \circ \phi^{\mathsf{s}})((\phi^{\mathsf{s}})^{-1}(w)) \\ &= prnt_H(w) \end{aligned}$$

$prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \in k_C$: This implies

$$\begin{aligned} prnt(w) &= prnt_C(prnt_3(w)) = prnt_C(prnt_1(\pi(w))) \\ &= prnt_C(prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w))) \\ &= prnt_C(((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ ((\phi^{\mathsf{v}})^{-1} \uplus \mathsf{Id}_{k_G}))((\phi^{\mathsf{s}})^{-1}(w))) \\ &= prnt_C(prnt_G((\phi^{\mathsf{s}})^{-1}(w))) \\ &= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_C)(prnt_G((\phi^{\mathsf{s}})^{-1}(w))) \\ &= \phi^{\mathsf{r}}(prnt_G((\phi^{\mathsf{s}})^{-1}(w))) \\ &= (prnt_H \circ \phi^{\mathsf{s}})((\phi^{\mathsf{s}})^{-1}(w)) \\ &= prnt_H(w) \end{aligned}$$

$w \in V_D$: This implies

$$prnt_3(w) = prnt_1(w)$$

which further divides into two cases:

$prnt_D(w) \in V_D$: This implies

$$\begin{aligned} & prnt_1(w) = prnt_D(w) \in V_D \\ \Rightarrow \quad & prnt(w) = prnt_3(w) = prnt_D(w) \\ & = (((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D}))(w) \\ & = (prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})})(w) \\ & = prnt_H(w) \end{aligned}$$

$prnt_D(w) \in k_G$: This implies

$$\begin{aligned} prnt_1(w) &= prnt_{\phi\blacksquare G}(prnt_D(w)) \\ &= prnt_{\phi\blacksquare G}((((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D}))(w)) \\ &= prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \end{aligned}$$

which again divides into two cases:

$prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \in \phi^{\mathsf{v}}(V_G)$**:** This implies

$$\begin{aligned}
prnt(w) &= prnt_3(w) = prnt_1(w) = prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \\
&= ((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ ((\phi^{\mathsf{v}})^{-1} \uplus \mathsf{Id}_{k_G}))((\phi^{\mathsf{s}})^{-1}(w)) \\
&= (\phi^{\mathsf{v}} \circ prnt_G \downharpoonleft^{V_G})((\phi^{\mathsf{s}})^{-1}(w)) \\
&= (prnt_H \circ \phi^{\mathsf{s}})((\phi^{\mathsf{s}})^{-1}(w)) \\
&= prnt_H(w)
\end{aligned}$$

$prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w)) \in k_C$**:** This implies

$$\begin{aligned}
prnt(w) &= prnt_C(prnt_3(w)) = prnt_C(prnt_1(w)) \\
&= prnt_C(prnt_{\phi\blacksquare G}((\phi^{\mathsf{s}})^{-1}(w))) \\
&= prnt_C(((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ ((\phi^{\mathsf{v}})^{-1} \uplus \mathsf{Id}_{k_G}))((\phi^{\mathsf{s}})^{-1}(w))) \\
&= prnt_C(prnt_G((\phi^{\mathsf{s}})^{-1}(w))) \\
&= (\phi^{\mathsf{r}} \uplus prnt_H \upharpoonright_{V_C} \uplus prnt_H \circ f_C)(prnt_G((\phi^{\mathsf{s}})^{-1}(w))) \\
&= \phi^{\mathsf{r}}(prnt_G((\phi^{\mathsf{s}})^{-1}(w))) \\
&= (prnt_H \circ \phi^{\mathsf{s}})((\phi^{\mathsf{s}})^{-1}(w)) \\
&= prnt_H(w)
\end{aligned}$$

$w \in V_C$**:**

$$\begin{aligned}
prnt(w) &= prnt_C(w) && \text{cf. def. of } prnt \\
&= prnt_H(w) && \text{cf. def. of } prnt_C
\end{aligned}$$

$w \in \phi^{\mathsf{v}}(V_G)$**:**

$$\begin{aligned}
prnt_3(w) &= prnt_1(w) = prnt_{\phi\blacksquare G}(w) \\
&= ((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ ((\phi^{\mathsf{v}})^{-1} \uplus \mathsf{Id}_{k_G}))(w) \\
&= ((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w)
\end{aligned}$$

There are two cases for $prnt_3(w)$:

$prnt_3(w) \in \phi^{\mathsf{v}}(V_G) \uplus V_D$**:** This implies

$$\begin{aligned}
prnt(w) &= prnt_3(w) = prnt_1(w) = prnt_{\phi\blacksquare G}(w) \\
&= ((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w) \\
&= (\phi^{\mathsf{v}} \circ prnt_G \downharpoonleft^{V_G} \circ (\phi^{\mathsf{v}})^{-1})(w) \\
&= (prnt_H \circ \phi^{\mathsf{v}} \circ (\phi^{\mathsf{v}})^{-1})(w) \\
&= prnt_H(w)
\end{aligned}$$

$prnt_3(w) \in k_C$**:** This implies

$$\begin{aligned}
prnt(w) &= prnt_C(prnt_3(w)) = prnt_C(prnt_1(w)) = prnt_C(prnt_{\phi \blacksquare G}(w)) \\
&= prnt_C(((\phi^{\mathsf{v}} \uplus \mathsf{Id}_{m_G}) \circ prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w)) \\
&= prnt_C((prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w)) \\
&= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_C)((prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w)) \\
&= \phi^{\mathsf{r}}((prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w)) \\
&= (\phi^{\mathsf{r}} \circ prnt_G \circ (\phi^{\mathsf{v}})^{-1})(w) \\
&= (prnt_H \circ \phi^{\mathsf{v}} \circ (\phi^{\mathsf{v}})^{-1})(w) \\
&= prnt_H(w)
\end{aligned}$$

□

**Proof of Theorem 7.5.10**

Def. 7.5.5 $\circ$ Def. 7.5.8 $=$ $\mathsf{Id}$: Assume

$$\begin{aligned}
H &= C \circ (G \circ D \otimes \mathsf{id}_k) \circ \pi \\
\phi &= \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} : G \hookrightarrow H \\
\phi^{\mathsf{v}} &= \mathsf{Id}_{V_G} \\
\phi^{\mathsf{r}} &= prnt_C \restriction_{m_G} \\
\phi^{\mathsf{s}} &= (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G}
\end{aligned}$$

and the results from the proof of Prop. 7.5.6.

Now, using construction Def. 7.5.8 we obtain:

$$\begin{aligned}
prmt(\phi) &= (V_{D'}, ctrl_H \restriction_{V_{D'}}, prnt_{D'}) : k_{D'} \to k_G \quad \text{where} \\
V_{D'} &= V_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \\
\tilde{k}_{D'} &= k_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \\
k_{D'} &= |\tilde{k}_{D'}| \\
f_{D'} &: k_{D'} \rightarrowtail \tilde{k}_{D'} \quad \text{a bijection} \\
prnt_{D'} &= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_{D'} \uplus \tilde{k}_{D'}) \setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_{D'} \uplus \mathsf{Id}_{V_{D'}})
\end{aligned}$$

$$\begin{aligned}
ctxt(\phi) &= (V_{C'}, ctrl_H \restriction_{V_{C'}}, prnt_{C'}) : k_{C'} \to m_H \quad \text{where} \\
V_{C'} &= (V_H \setminus \phi^{\mathsf{v}}(V_G)) \setminus V_{D'} \\
\tilde{k}_{C'} &= k_H \setminus \tilde{k}_{D'} \\
k_{C'} &= m_G + |\tilde{k}_{C'}| \\
f'_{C'} &: |\tilde{k}_{C'}| \rightarrowtail \tilde{k}_{C'} \quad \text{a bijection} \\
f_{C'}(i + m_G) &= f'_{C'}(i) \quad \text{for } i \in |\tilde{k}_{C'}| \\
prnt_{C'} &= \phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_{C'}} \uplus prnt_H \circ f_{C'}
\end{aligned}$$

$$\begin{aligned}
H &= ctxt(\phi) \circ (\phi \centerdot G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_{C'}|}) \circ \pi' \\
&= ctxt(\phi) \circ (G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_{C'}|}) \circ \pi' \\
\pi' &= f_{D'}^{-1} \uplus f'^{-1} : k_H \to k_H \\
f'(i + k_{D'}) &= f'_{C'}(i) \quad \text{for } i \in |\tilde{k}_{C'}|.
\end{aligned}$$

We must show $D = \mathit{prmt}(\phi)$, $C = \mathit{ctxt}(\phi)$, and $\pi = \pi'$. First, let us unfold some of the definitions:

$$
\begin{aligned}
\mathsf{rng}(\phi^{\mathsf{s}}) &= \mathsf{rng}((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ \mathit{prnt}_D^{-1} \restriction_{k_G}) \\
&= (\mathsf{Id}_{V_D} \uplus \pi^{-1})(\{c \mid c \in k_D \uplus V_D \wedge \mathit{prnt}_D(c) \in k_G\}) \\
&= \{c \mid (c \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c)) \in k_G) \vee (c \in V_D \wedge \mathit{prnt}_D(c) \in k_G)\} \\
H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} &= \{c' \mid c' \in k_H \uplus V_H \;\wedge\; \exists i \geq 0 : \mathit{prnt}_H^i(c') \in \mathsf{rng}(\phi^{\mathsf{s}})\} \\
&= \{c' \mid c' \in k_H \uplus V_H \\
&\qquad \wedge \exists i \geq 0 : (\mathit{prnt}_H^i(c') \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(\mathit{prnt}_H^i(c'))) \in k_G) \\
&\qquad\quad \vee (\mathit{prnt}_H^i(c') \in V_D \wedge \mathit{prnt}_D(\mathit{prnt}_H^i(c')) \in k_G)\} \\
&= \{c' \mid \exists i > 0 : (c' \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c')) \in k_G) \\
&\qquad \vee (c' \in V_D \wedge \mathit{prnt}_D(c') \in k_G) \\
&\qquad \vee (c' \in \pi^{-1}(k_D) \wedge \mathit{prnt}_H^{i-1}(\mathit{prnt}_D(\pi(c'))) \in V_D \wedge \mathit{prnt}_D(\mathit{prnt}_H^{i-1}(\mathit{prnt}_D(\pi(c')))) \in k_G) \\
&\qquad \vee (c' \in V_D \wedge \mathit{prnt}_H^{i-1}(\mathit{prnt}_D(c')) \in V_D \wedge \mathit{prnt}_D(\mathit{prnt}_H^{i-1}(\mathit{prnt}_D(c'))) \in k_G)\} \\
&= \{c' \mid \exists i > 0 : (c' \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c')) \in k_G) \\
&\qquad \vee (c' \in V_D \wedge \mathit{prnt}_D(c') \in k_G) \\
&\qquad \vee (c' \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D^{i-1}(\mathit{prnt}_D(\pi(c'))) \in V_D \wedge \mathit{prnt}_D(\mathit{prnt}_D^{i-1}(\mathit{prnt}_D(\pi(c')))) \in k_G) \\
&\qquad \vee (c' \in V_D \wedge \mathit{prnt}_D^{i-1}(\mathit{prnt}_D(c')) \in V_D \wedge \mathit{prnt}_D(\mathit{prnt}_D^{i-1}(\mathit{prnt}_D(c'))) \in k_G)\} \\
&= \{c' \mid \exists i > 0 : (c' \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D^i(\pi(c')) \in k_G) \\
&\qquad \vee (c' \in V_D \wedge \mathit{prnt}_D^i(c') \in k_G)\} \\
&= \pi^{-1}(k_D) \uplus V_D \\
\tilde{k}_{D'} &= k_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} = k_H \cap (\pi^{-1}(k_D) \uplus V_D) = \pi^{-1}(k_D) \\
k_{D'} &= |\tilde{k}_{D'}| = |\pi^{-1}(k_D)| = k_D \\
f_{D'} &: k_D = k_{D'} \quad \rightarrowtail \quad \tilde{k}_{D'} = \pi^{-1}(k_D) \\
\tilde{k}_{C'} &= k_H \setminus \tilde{k}_{D'} = k_H \setminus \pi^{-1}(k_D) = \pi^{-1}(k_H \setminus k_D) \\
k_{C'} &= m_G + |\tilde{k}_{C'}| = m_G + |\pi^{-1}(k_H \setminus k_D)| = m_G + k_H - k_D \\
f'_{C'} &: (k_H - k_D) = |\tilde{k}_{C'}| \quad \rightarrowtail \quad \tilde{k}_{C'} = \pi^{-1}(k_H \setminus k_D) \\
f_{C'}(i + m_G) &= f'_{C'}(i) \quad \text{for } i \in |\tilde{k}_{C'}| = (k_H - k_D) \\
f'(i + k_{D'}) &= f'_{C'}(i) \quad \text{for } i \in |\tilde{k}_{C'}| = (k_H - k_D)
\end{aligned}
$$

With these in mind, we proceed to prove $D = \mathit{prmt}(\phi)$, $C = \mathit{ctxt}(\phi)$, and $\pi = \pi'$:

**$\pi = \pi'$:** Remember that in Def. 7.5.8 we are free to choose the two bijections $f_{D'}$,$f'_{C'}$ as they are internal to the decomposition. We choose them to be suitable restrictions of the inverse of $\pi$:

$$
\begin{aligned}
f_{D'} &= \pi^{-1} \restriction_{k_D} : k_D \rightarrowtail \pi^{-1}(k_D) \\
f'_{C'}(i) &= \pi^{-1}(i + k_D) : (k_H - k_D) \rightarrowtail \pi^{-1}(k_H \setminus k_D)
\end{aligned}
$$

Expanding these in the derived functions we get:

$$
\begin{aligned}
f_{C'}(i+m_G) &= f'_{C'}(i) = \pi^{-1}(i+k_D) \\
f'(i) &= f'_{C'}(i-k_D) = \pi^{-1}(i-k_D+k_D) = \pi^{-1}(i) \quad \text{for } (i-k_D) \in |\tilde{k}_{C'}| = k_H - k_D \\
\pi' &= f_{D'}^{-1} \uplus f'^{-1} \\
\pi'(i) &= \begin{cases} f_{D'}^{-1}(i) & \text{if } i \in \pi^{-1}(k_D) \\ f'^{-1}(i) & \text{if } i \in \pi^{-1}(k_H \setminus k_D) \end{cases} \\
&= \begin{cases} \pi(i) & \text{if } i \in \pi^{-1}(k_D) \\ \pi(i) & \text{if } i \in \pi^{-1}(k_H \setminus k_D) \end{cases} \\
&= \pi(i).
\end{aligned}
$$

$D = \mathit{prmt}(\phi)$**:** It suffices to show $V_{D'} = V_D$ and $\mathit{prnt}_{D'} = \mathit{prnt}_D$, which is easily seen by unfolding the definitions:

$$
\begin{aligned}
V_{D'} &= V_H \cap H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} = V_H \cap (\pi^{-1}(k_D) \uplus V_D) = V_D \\
\mathit{prnt}_{D'} &= ((\phi^{\mathsf{s}})^{-1} \uplus \mathit{prnt}_H \restriction_{(V_{D'} \uplus \tilde{k}_{D'}) \setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_{D'} \uplus \mathsf{Id}_{V_{D'}}) \\
&= (((\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ \mathit{prnt}_D^{-1} \restriction_{k_G})^{-1} \\
&\qquad \uplus \mathit{prnt}_H \restriction_{(V_D \uplus \pi^{-1}(k_D)) \setminus (\{c \mid (c \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c)) \in k_G) \vee (c \in V_D \wedge \mathit{prnt}_D(c) \in k_G)\})}) \\
&\quad \circ (f_{D'} \uplus \mathsf{Id}_{V_D}) \\
&= ((\mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus \pi)) \restriction_{\{c \mid (c \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c)) \in k_G) \vee (c \in V_D \wedge \mathit{prnt}_D(c) \in k_G)\}} \\
&\qquad \uplus (\mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus \pi)) \restriction_{\{c \mid (c \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c)) \notin k_G) \vee (c \in V_D \wedge \mathit{prnt}_D(c) \notin k_G)\}}) \\
&\quad \circ (f_{D'} \uplus \mathsf{Id}_{V_D}) \\
&= ((\mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus \pi)) \restriction_{\{c \mid (c \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c)) \in k_G) \vee (c \in V_D \wedge \mathit{prnt}_D(c) \in k_G)\}} \\
&\qquad \uplus (\mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus \pi)) \restriction_{\{c \mid (c \in \pi^{-1}(k_D) \wedge \mathit{prnt}_D(\pi(c)) \notin k_G) \vee (c \in V_D \wedge \mathit{prnt}_D(c) \notin k_G)\}}) \\
&\quad \circ (f_{D'} \uplus \mathsf{Id}_{V_D}) \\
&= \mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus \pi) \circ (f_{D'} \uplus \mathsf{Id}_{V_D}) \\
&= \mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus \pi' \circ f_{D'}) \\
&= \mathit{prnt}_D \circ (\mathsf{Id}_{V_D} \uplus (f_{D'}^{-1} \uplus f'^{-1}) \circ f_{D'}) \\
&= \mathit{prnt}_D\,.
\end{aligned}
$$

$C = \mathit{ctxt}(\phi)$**:** It suffices to show $V_{C'} = V_C$ and $\mathit{prnt}_{C'} = \mathit{prnt}_C$, which is seen by unfolding the

definitions:

$$\begin{aligned}
V_{C'} &= (V_H \setminus \phi^{\mathsf{v}}(V_G)) \setminus V_{D'} = (V_H \setminus V_G) \setminus V_D = V_C \\
(prnt_H \circ f_{C'})(i) &= prnt_C(m_G + \pi(f_{C'}(i))) \quad \text{for } i \in k_H - k_D \\
&= prnt_C(m_G + \pi(\pi^{-1}(i - m_G + k_D))) \\
&= prnt_C(m_G + i - m_G + k_D) \\
&= prnt_C(i + k_D) \\
prnt_C(i) &= prnt_H(\pi^{-1}(i - m_G + k_D)) \quad \text{for } i \in (m_G + k_H - k_D) \setminus m_G \\
&= (prnt_H \circ f_{C'})(i) \\
prnt_{C'} &= \phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_{C'}} \uplus prnt_H \circ f_{C'} \\
&= prnt_C \restriction_{m_G} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_{C'} \\
&= prnt_C \restriction_{m_G} \uplus prnt_C \restriction_{V_C} \uplus prnt_C \restriction_{(m_G + k_H - k_D) \setminus m_G} \\
&= prnt_C \,.
\end{aligned}$$

Def. 7.5.5 $\circ$ Def. 7.5.8 $=$ $\mathsf{Id}$: Assume a place graph $G : k_G \to m_G$, an embedding $\phi : G \hookrightarrow H$ into a place graph $H : k_H \to m_H$ (for simplicity, assume $\phi \centerdot G = G$),

$$\begin{aligned}
prmt(\phi) &= (V_D, ctrl_H \restriction_{V_D}, prnt_D) : k_D \to k_G \quad \text{where} \\
V_D &= V_H \cap H \downharpoonright^{\mathsf{rng}(\phi^{\mathsf{s}})} \\
\tilde{k}_D &= k_H \cap H \downharpoonright^{\mathsf{rng}(\phi^{\mathsf{s}})} \\
k_D &= |\tilde{k}_D| \\
f_D &: k_D \rightarrowtail \tilde{k}_D \quad \text{a bijection} \\
prnt_D &= ((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D) \setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D})
\end{aligned}$$

$$\begin{aligned}
ctxt(\phi) &= (V_C, ctrl_H \restriction_{V_C}, prnt_C) : k_C \to m_H \quad \text{where} \\
V_C &= (V_H \setminus V_G) \setminus V_D \\
\tilde{k}_C &= k_H \setminus \tilde{k}_D \\
k_C &= m_G + |\tilde{k}_C| \\
f'_C &: |\tilde{k}_C| \rightarrowtail \tilde{k}_C \quad \text{a bijection} \\
f_C(i + m_G) &= f'_C(i) \quad \text{for } i \in |\tilde{k}_C| \\
prnt_C &= \phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_C \\
H &= ctxt(\phi) \circ (G \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|}) \circ \pi \\
\pi &= f_D^{-1} \uplus f'^{-1} : k_H \to k_H \\
f'(i + k_D) &= f'_C(i) \quad \text{for } i \in |\tilde{k}_C|
\end{aligned}$$

where we assume that the bijections are chosen as in the previous proof case, i.e.,

$$\begin{aligned}
f_D &= \pi^{-1} \restriction_{k_D} : k_D \rightarrowtail \pi^{-1}(k_D) \\
f'_C(i) &= \pi^{-1}(i + k_D) : (k_H - k_D) \rightarrowtail \pi^{-1}(k_H \setminus k_D).
\end{aligned}$$

Now, using construction Def. 7.5.5 we obtain:

$$\begin{aligned}\phi' &= \phi'^{\mathsf{v}} \uplus \phi'^{\mathsf{s}} \uplus \phi'^{\mathsf{r}} : G \hookrightarrow H\\ \phi'^{\mathsf{v}} &= \mathsf{Id}_{V_G}\\ \phi'^{\mathsf{r}} &= prnt_C \restriction_{m_G}\\ \phi'^{\mathsf{s}} &= (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G}\ .\end{aligned}$$

We must prove $\phi = \phi'$, and it suffices to show $\phi^{\mathsf{v}} = \phi'^{\mathsf{v}}$, $\phi^{\mathsf{r}} = \phi'^{\mathsf{r}}$, and $\phi^{\mathsf{s}} = \phi'^{\mathsf{s}}$.

**$\phi^{\mathsf{v}} = \phi'^{\mathsf{v}}$:** Satisfied by assumption.

**$\phi^{\mathsf{r}} = \phi'^{\mathsf{r}}$:** Easily seen by unfolding the definitions:

$$\begin{aligned}\phi'^{\mathsf{r}} &= prnt_C \restriction_{m_G}\\ &= (\phi^{\mathsf{r}} \uplus prnt_H \restriction_{V_C} \uplus prnt_H \circ f_C) \restriction_{m_G}\\ &= \phi^{\mathsf{r}}.\end{aligned}$$

**$\phi^{\mathsf{s}} = \phi'^{\mathsf{s}}$:** Easily seen by unfolding the definitions:

$$\begin{aligned}\phi'^{\mathsf{s}} &= (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G}\\ &= (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ (((\phi^{\mathsf{s}})^{-1} \uplus prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})}) \circ (f_D \uplus \mathsf{Id}_{V_D}))^{-1} \restriction_{k_G}\\ &= (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ (f_D^{-1} \uplus \mathsf{Id}_{V_D}) \circ (\phi^{\mathsf{s}} \uplus (prnt_H \restriction_{(V_D \uplus \tilde{k}_D)\setminus \mathsf{rng}(\phi^{\mathsf{s}})})^{-1}) \restriction_{k_G}\\ &= (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ (\pi \uplus \mathsf{Id}_{V_D}) \circ \phi^{\mathsf{s}}\\ &= \phi^{\mathsf{s}}.\end{aligned}$$

### Proof of Lemma 7.5.11

1: We show that $v \in H \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} = \{v \mid v \in V_H \ \wedge\ \exists i \geq 0 : prnt_H^i(v) \in \phi^{\mathsf{s}}(k_G)\}$ implies $v \notin \mathsf{rng}(\phi^{\mathsf{f}})$ by induction on $i$:

**$i = 0$:** We have $v \in \mathsf{rng}(\phi^{\mathsf{s}})$ and conditions (PGE-5) and (PGE-6) then give us $v \notin \mathsf{rng}(\phi^{\mathsf{f}})$.

**$i > 0$:** We have $prnt_H^i(v) \in \mathsf{rng}(\phi^{\mathsf{s}})$, i.e., $prnt_H^{i-1}(prnt_H(v)) \in \mathsf{rng}(\phi^{\mathsf{s}})$ which by the induction hypothesis means $prnt_H(v) \notin \mathsf{rng}(\phi^{\mathsf{f}})$. Thus condition (PGE-9) cannot be satisfied if $v \in \mathsf{rng}(\phi^{\mathsf{v}})$ and condition (PGE-6) prevents $v \in \mathsf{rng}(\phi^{\mathsf{r}})$, so $v \notin \mathsf{rng}(\phi^{\mathsf{f}})$.

2: First, we show that $prnt_G^i(c) = p$ $(i > 0)$ implies $prnt_H^i(\phi^{\mathsf{c}}(c)) = \phi^{\mathsf{f}}(p)$. We show this by induction on $i$:

**$i = 1$:** Assuming $prnt_G(c) = p$, we have $\phi^{\mathsf{f}}(prnt_G(c)) = \phi^{\mathsf{f}}(p)$ and thus, by condition (PGE-9), $prnt_H(\phi^{\mathsf{c}}(c)) = \phi^{\mathsf{f}}(p)$ as required.

**$i > 1$:** Assuming $prnt_G^i(c) = p$, we have $prnt_G^{i-1}(prnt_G(c)) = p$ and thus, by the induction hypothesis, $prnt_H^{i-1}(\phi^{\mathsf{c}}(prnt_G(c))) = \phi^{\mathsf{f}}(p)$. Since $\mathsf{dom}(\phi^{\mathsf{c}}) \cap \mathsf{cod}(prnt_G) = V_G$, $\phi^{\mathsf{f}} = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{r}}$ and $\phi^{\mathsf{c}} = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}}$, we have

$$\begin{aligned}&prnt_H^{i-1}(\phi^{\mathsf{c}}(prnt_G(c)))\\ =\,&prnt_H^{i-1}(\phi^{\mathsf{v}}(prnt_G(c)))\\ =\,&prnt_H^{i-1}(\phi^{\mathsf{f}}(prnt_G(c)))\\ =\,&\phi^{\mathsf{f}}(p)\end{aligned}$$

and thus, by condition (PGE-9), $prnt_H^{i-1}(prnt_H(\phi^{\mathsf{c}}(c))) = prnt_H^i(\phi^{\mathsf{c}}(c)) = \phi^{\mathsf{f}}(p)$ as required.

Now, since $prnt_G$ is acyclic we have:

$$c \in V_G \uplus k_G \Rightarrow \exists i > 0 : prnt^i_G(c) \in m_G$$

and thus, using the above result, we have

$$c \in V_G \uplus k_G \Rightarrow \exists i > 0 : prnt^i_H(\phi^{\mathsf{c}}(c)) \in \phi^{\mathsf{r}}(m_G) = \mathsf{rng}(\phi^{\mathsf{r}})$$

i.e., for any $c \in \mathsf{rng}(\phi^{\mathsf{c}})$ we have

$$\exists i > 0 : prnt^i_H(c) \in \mathsf{rng}(\phi^{\mathsf{r}}).$$

But this cannot be satisfied by any $c \in H\restriction_{\mathsf{rng}(\phi^{\mathsf{r}})}$, since Prop. 7.2.11 gives us

$$\begin{aligned} H\restriction_{\mathsf{rng}(\phi^{\mathsf{r}})} = &(V_H \uplus k_H \uplus m_H) \\ &\setminus \{c' \mid c' \in k_H \uplus V_H \ \wedge \ \exists i > 0 : prnt^i_H(c') \in \mathsf{rng}(\phi^{\mathsf{r}})\} \end{aligned}$$

i.e., $c \in H\restriction_{\mathsf{rng}(\phi^{\mathsf{r}})}$ iff $\forall i > 0 : prnt^i_H(c) \notin \mathsf{rng}(\phi^{\mathsf{r}})$.

3: From the previous proof case, we have that any $c \in \mathsf{rng}(\phi^{\mathsf{c}})$ satisfies

$$\exists i > 0 : prnt^i_H(c) \in \mathsf{rng}(\phi^{\mathsf{r}}).$$

Now, from Prop. 7.2.11 we have that $c \in H\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}$ implies $c \in k_H \uplus V_H$ and $\exists i \geq 0 : prnt^i_H(c) \in \phi^{\mathsf{s}}(k_G)$, which combined the above result and the fact $\phi^{\mathsf{s}}(k_G) \subseteq \mathsf{rng}(\phi^{\mathsf{c}})$ we get

$$\exists i > 0 : prnt^i_H(c) \in \mathsf{rng}(\phi^{\mathsf{r}})$$

which by Prop. 7.2.11 means $c \notin H\restriction_{\mathsf{rng}(\phi^{\mathsf{r}})}$.

4: Let $i \in k_G$ be a site and $c \in H\downharpoonleft^{\phi^{\mathsf{s}}(i)} \setminus \phi^{\mathsf{s}}(i)$. Then by Def. 7.2.9

$$\begin{aligned} &H\downharpoonleft^{\phi^{\mathsf{s}}(i)} \setminus \phi^{\mathsf{s}}(i) \\ &= \{c \mid c \in k_H \uplus V_H \ \wedge \ \exists i \geq 0 : prnt^i_H(c) \in \phi^{\mathsf{s}}(i)\} \setminus \phi^{\mathsf{s}}(i) \\ &= \{c \mid c \in k_H \uplus V_H \ \wedge \ \exists i > 0 : prnt^i_H(c) \in \phi^{\mathsf{s}}(i)\}. \end{aligned}$$

We show that $c \in H\downharpoonleft^{\phi^{\mathsf{s}}(i)} \setminus \phi^{\mathsf{s}}(i)$ implies $c \notin \mathsf{rng}(\phi^{\mathsf{c}}) \uplus \mathsf{rng}(\phi^{\mathsf{r}})$ by induction on $i$:

**$i = 1$:** We have $prnt_H(c) \in \phi^{\mathsf{s}}(i)$. We obtain a contradiction if $c \in \mathsf{rng}(\phi^{\mathsf{c}}) \uplus \mathsf{rng}(\phi^{\mathsf{r}})$:

- **$c \in \mathsf{rng}(\phi^{\mathsf{c}})$:** $(\phi^{\mathsf{c}})^{-1}(c)$ is defined and so, by condition (PGE-9), $prnt_H(c) = \phi^{\mathsf{f}}(prnt_G((\phi^{\mathsf{c}})^{-1}(c))) \in \phi^{\mathsf{s}}(i)$ which violates conditions (PGE-5) and (PGE-6).
- **$c \in \mathsf{rng}(\phi^{\mathsf{r}})$:** This violates condition (PGE-6).

**$i > 1$:** We have $prnt^i_H(c) = prnt^{i-1}_H(prnt_H(c)) \in \phi^{\mathsf{s}}(i)$, so by the induction hypothesis $prnt_H(c) \notin \mathsf{rng}(\phi^{\mathsf{c}}) \uplus \mathsf{rng}(\phi^{\mathsf{r}})$. We obtain a contradiction if $c \in \mathsf{rng}(\phi^{\mathsf{c}})$, because then $(\phi^{\mathsf{c}})^{-1}(c)$ is defined and so, by condition (PGE-9), $prnt_H(c) = \phi^{\mathsf{f}}(prnt_G((\phi^{\mathsf{c}})^{-1}(c)))$ which contradicts $prnt_H(c) \notin \mathsf{rng}(\phi^{\mathsf{c}}) \uplus \mathsf{rng}(\phi^{\mathsf{r}})$.

□

**Proof of Prop. 7.5.13**

$\phi^{\mathsf{s}}$: Construct the map of each site $i \in k_G$ as follows:

$$\begin{aligned}\phi^{\mathsf{s}}(i) &= \mathit{children}_{H,i} \setminus \phi^{\mathsf{v}}(\mathit{siblings}_{G,i}) \\ \mathit{children}_{H,i} &= (\mathit{prnt}_H^{-1} \circ \phi^{\mathsf{v}})(\mathit{prnt}_G(i)) \\ \mathit{siblings}_{G,i} &= (\mathit{prnt}_G^{-1} \circ \mathit{prnt}_G)(i) \setminus \{i\}.\end{aligned}$$

This is well-defined since every site of $G$ is guarding, thus $\mathit{prnt}_G(i) \in V_G$, and no sites are siblings. By construction it satisfies condition (PGE-7); it must satisfy the other conditions since $\phi$ is an embedding and $\phi^{\mathsf{s}}$ is the only embedding of inner names that will satisfy condition (PGE-7): To see that $\phi^{\mathsf{s}}$ is unique, assume that there is a different $\phi'^{\mathsf{s}}$ that satisfies the conditions of Def. 7.5.4. For the two to be different, there must be some $i \in k_G$ and $c \in k_H \uplus V_H$ with $c \in \phi^{\mathsf{s}}(i), c \notin \phi'^{\mathsf{s}}(i)$ (or vice versa). Since they both satisfy condition (PGE-7) and no root has a site as a child we have:

$$\begin{aligned}\mathit{prnt}_G(i) &\in V_G \\ (\phi^{\mathsf{c}} \circ \mathit{prnt}_G^{-1} \restriction_{V_G})(\mathit{prnt}_G(i)) &= (\mathit{prnt}_H^{-1} \circ \phi^{\mathsf{v}})(\mathit{prnt}_G(i)) \\ &= (\phi'^{\mathsf{c}} \circ \mathit{prnt}_G^{-1} \restriction_{V_G})(\mathit{prnt}_G(i))\end{aligned}$$

where

$$\begin{aligned}\phi^{\mathsf{c}} &= \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \\ \phi'^{\mathsf{c}} &= \phi^{\mathsf{v}} \uplus \phi'^{\mathsf{s}}.\end{aligned}$$

And since $c \in \phi^{\mathsf{s}}(i)$, and $\phi^{\mathsf{c}}$ and $\phi'^{\mathsf{c}}$ agree on nodes, we must have $c \in \phi'^{\mathsf{s}}(i')$ for some $i' \in \mathit{prnt}_G^{-1} \restriction_{V_G} (\mathit{prnt}_G(i))$. But no sites are siblings, so $i' = i$ and thus $c \in \phi'^{\mathsf{s}}(i)$ which contradicts our assumption that $c \notin \phi'^{\mathsf{s}}(i)$.

$\phi^{\mathsf{r}}$: Construct the map of each root $j \in m_G$ as follows: choose a node $v \in \mathit{prnt}_G^{-1}(j)$, which is always possible since no root is idle or has a site as child, and let

$$\phi^{\mathsf{r}}(j) = (\mathit{prnt}_H \circ \phi^{\mathsf{v}})(v).$$

By construction it satisfies condition (PGE-9); it must satisfy the other conditions since $\phi$ is an embedding and $\phi^{\mathsf{r}}$ is the only embedding of outer names that will satisfy condition (PGE-9): To see that $\phi^{\mathsf{r}}$ is unique, assume that there is a different $\phi'^{\mathsf{r}}$ that satisfies the conditions of Def. 7.5.4. For the two to be different, we must have $\phi^{\mathsf{r}}(j) \neq \phi'^{\mathsf{r}}(j)$ for some $j \in m_G$. But since they both satisfy condition (PGE-9), the following must hold for the node $v$ we chose when defining $\phi^{\mathsf{r}}(j)$:

$$\begin{aligned}\phi^{\mathsf{r}}(j) &= (\phi^{\mathsf{r}} \circ \mathit{prnt}_G)(v) \\ &= (\mathit{prnt}_H \circ \phi^{\mathsf{c}})(v) \\ &= (\phi'^{\mathsf{r}} \circ \mathit{prnt}_G)(v) = \phi'^{\mathsf{r}}(j)\end{aligned}$$

which contradicts our assumption that $\phi^{\mathsf{r}}$ and $\phi'^{\mathsf{r}}$ are different. □

**Proof of Prop. 7.5.17**

It is clear that the place graph may be expressed as

$$H^P = C^P \circ (G^P \circ D^P \otimes \mathsf{id}_k) \circ \pi$$

and thus Prop. 7.5.6 applies, whereby we have that $\phi^P = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} : G^P \hookrightarrow H^P$ is a place graph embedding and that $V_D = V_H \cap H\lfloor^{\mathsf{rng}(\phi^{\mathsf{s}})}$. What remains to show is that $\phi^L = \phi^{\mathsf{v}} \uplus \phi^{\mathsf{e}} \uplus \phi^{\mathsf{i}} \uplus \phi^{\mathsf{o}} : G^L \hookrightarrow H^L$ is a link graph embedding and that the two embeddings are consistent.

From the definitions of support translation, composition, tensor product, and $V_D = V_H \cap H\lfloor^{\mathsf{rng}(\phi^{\mathsf{s}})}$ we have:

$$
\begin{aligned}
V_H &= V_C \uplus V_G \uplus V_D & \qquad \mathit{ctrl}_H &= \mathit{ctrl}_C \uplus \mathit{ctrl}_G \uplus \mathit{ctrl}_D \\
P_H &= P_C \uplus P_G \uplus P_D & C &: \langle m_G + k, Y_G \uplus X_I \uplus X_C \rangle \to \langle m_H, Y_H \rangle \\
E_H &= E_C \uplus E_G & D &: \langle k_D, X_D \rangle \to \langle k_G, X_G \uplus X_I \rangle \\
X_H &= X_D \uplus X'_C & \alpha &: X'_C \to X_C \\
P_D &= P_{H\lfloor^{\mathsf{rng}(\phi^{\mathsf{s}})} \cap V_H}
\end{aligned}
$$

To show that $\phi^L$ is a link graph embedding, we need to express the link map of $H$ in terms of it decomposition. It is clear that the link graph may be expressed as

$$H^L = C^L \circ ((G^L \otimes \mathsf{id}_{X_I}) \circ D^L \otimes \alpha).$$

We construct the link map incrementally according to the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8)

**$G^L \otimes \mathsf{id}_{X_I}$:** We write $\mathit{link}_1$ for the link map of the resulting link graph:

$$\mathit{link}_1 = \mathit{link}_G \uplus \mathsf{Id}_{X_I}.$$

**$(G^L \otimes \mathsf{id}_{X_I}) \circ D^L$:** We write $\mathit{link}_2$ for the link map of the resulting link graph:

$$
\begin{aligned}
\mathit{link}_2(p) &= \begin{cases} \mathit{link}_D(p) & \text{if } p \in X_D \uplus P_D \text{ and } \mathit{link}_D(p) \in \emptyset \\ \mathit{link}_1(y) & \text{if } p \in X_D \uplus P_D \text{ and } \mathit{link}_D(p) = y \in X_G \uplus X_I \\ \mathit{link}_1(p) & \text{if } p \in P_G \end{cases} \\
&= \begin{cases} \mathit{link}_G(y) & \text{if } p \in X_D \uplus P_D \text{ and } \mathit{link}_D(p) = y \in X_G \\ y & \text{if } p \in X_D \uplus P_D \text{ and } \mathit{link}_D(p) = y \in X_I \\ \mathit{link}_G(p) & \text{if } p \in P_G. \end{cases}
\end{aligned}
$$

**$(G^L \otimes \mathsf{id}_{X_I}) \circ D^L \otimes \alpha$:** We write $\mathit{link}_3$ for the link map of the resulting link graph:

$$\mathit{link}_3 = \mathit{link}_2 \uplus \alpha.$$

$H^L = C^L \circ ((G^L \otimes \mathsf{id}_{X_I}) \circ D^L \otimes \alpha)$:

$$link_H = \begin{cases} link_3(p) & \text{if } p \in X_D \uplus X'_C \uplus P_D \uplus P_G \text{ and } link_3(p) \in E_G \\ link_C(y) & \text{if } p \in X_D \uplus X'_C \uplus P_D \uplus P_G \text{ and } link_3(p) = y \in Y_G \uplus X_I \uplus X_C \\ link_C(p) & \text{if } p \in P_C \end{cases}$$

$$= \begin{cases} link_2(p) & \text{if } p \in X_D \uplus P_D \uplus P_G \text{ and } link_2(p) \in E_G \\ link_C(y) & \text{if } p \in X_D \uplus P_D \uplus P_G \text{ and } link_2(p) = y \in Y_G \uplus X_I \\ link_C(\alpha(p)) & \text{if } p \in X'_C \\ link_C(p) & \text{if } p \in P_C \end{cases}$$

$$= \begin{cases} link_G(y) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) = y \in X_G \\ & \quad \text{and } link_G(y) \in E_G \\ link_G(p) & \text{if } p \in P_G \text{ and } link_G(p) \in E_G \\ link_C(y) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) = z \in X_G \\ & \quad \text{and } link_G(z) = y \in Y_G \\ link_C(y) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) = y \in X_I \\ link_C(y) & \text{if } p \in P_G \text{ and } link_G(p) = y \in Y_G \\ link_C(\alpha(p)) & \text{if } p \in X'_C \\ link_C(p) & \text{if } p \in P_C. \end{cases}$$

We can now verify $\phi^L$ is a link graph embedding and that $\phi$ is a bigraph embedding, i.e., they satisfy the conditions of Def. 7.5.1 and Def. 7.5.14:

$\phi^L$:

**(LGE-1)** Satisfied since $\mathsf{Id}_{V_G}$ is an identity map.

**(LGE-2)** Satisfied since $\mathsf{Id}_{E_G}$ is an identity map.

**(LGE-3)** Since $link_D : X_D \uplus P_D \to X_G \uplus X_I$, $X_D \subseteq X_H$, and $P_D \subseteq P_H$ we have $\phi^{\mathsf{i}} = link_D^{-1} \restriction_{X_G} : X_G \to \mathcal{P}(X_H \uplus X_H)$; it is fully injective since $link_D$ is a function.

**(LGE-4)** Satisfied since $E_C \subseteq E_H$.

**(LGE-5)** Satisfied since $\mathsf{rng}(\phi^{\mathsf{e}}) = \mathsf{rng}(\mathsf{Id}_{E_G}) = E_G$, $\mathsf{rng}(\phi^{\mathsf{o}}) = \mathsf{rng}(link_C \restriction_{Y_G}) \subseteq E_C \uplus Y_H$, and $(E_c \uplus Y_H) \# E_G$.

**(LGE-6)** Satisfied since $\mathsf{rng}(\phi^{\mathsf{i}}) = \mathsf{rng}(link_D^{-1} \restriction_{X_G}) \subseteq X_D \uplus P_D$, $\mathsf{rng}(\phi^{\mathsf{port}}) \subseteq P_G$, and $X_D \uplus P_D \# P_G$.

**(LGE-7)** Satisfied since we have the following equalities:

$$\begin{aligned} link_H^{-1} \circ \phi^{\mathsf{e}} &= ((\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G}) \circ link_G \circ (\phi^{\mathsf{p}})^{-1})^{-1} \circ \phi^{\mathsf{e}} \\ &= \phi^{\mathsf{p}} \circ link_G^{-1} \circ (\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G})^{-1} \circ \phi^{\mathsf{e}} \\ &= \phi^{\mathsf{p}} \circ link_G^{-1} \restriction_{E_G} . \end{aligned}$$

**(LGE-8)** Satisfied since $\phi^{\mathsf{v}} = \mathsf{id}_{V_G}$ and $ctrl_G \subseteq ctrl_H$.

**(LGE-9)** We check the condition separately for the ports and inner names:

$p \in P_G$: We check the condition for edges and outer names separately (noting $\phi^{\mathsf{p}}(p) = p$):

$link_G(p) \in E_G$**:**

$$\begin{aligned}(\phi^{\mathsf{l}} \circ link_G)(p) &= (\phi^{\mathsf{e}} \circ link_H)(p)\\ &= (\mathsf{Id}_{E_G} \circ link_H)(p)\\ &= (link_H \circ \phi^{\mathsf{p}})(p).\end{aligned}$$

$link_G(p) \in Y_G$**:**

$$\begin{aligned}(\phi^{\mathsf{l}} \circ link_G)(p) &= (\phi^{\mathsf{o}} \circ link_G)(p)\\ &= (link_C \restriction_{Y_G} \circ link_G)(p)\\ &= (link_C \circ link_G \circ \phi^{\mathsf{p}})(p)\\ &= (link_H \circ \phi^{\mathsf{p}})(p).\end{aligned}$$

$x \in X_G$**:** We check the condition for edges and outer names separately:
$link_G(x) \in E_G$**:**

$$\begin{aligned}(\phi^{\mathsf{l}} \circ link_G)(x) &= (\phi^{\mathsf{e}} \circ link_G)(x)\\ &= (\mathsf{Id}_{E_G} \circ link_G)(x)\\ &= link_G(x)\\ &= link_H(link_D^{-1}(x))\\ &= (link_H \circ \phi^{\mathsf{p}})(x).\end{aligned}$$

$link_G(x) \in Y_G$**:**

$$\begin{aligned}(\phi^{\mathsf{l}} \circ link_G)(x) &= (\phi^{\mathsf{o}} \circ link_G)(x)\\ &= (link_C \circ link_G)(x)\\ &= link_H(link_D^{-1}(x)\\ &= (link_H \circ \phi^{\mathsf{p}})(x).\end{aligned}$$

$\phi$**:** We have verified that $\phi$ is both a place and link graph embedding, so it only remains to show that these are consistent:

**(BGE-1)** Satisfied since $\mathsf{rng}(\phi^{\mathsf{i}}) = \mathsf{rng}(link_D^{-1} \restriction_{X_G}) \subseteq X_D \uplus P_D$, $X_D \subseteq X_H$, and $P_D = P_{H\downarrow^{\mathsf{rng}(\phi^{\mathsf{s}})} \cap V_H}$.

□

**Proof of Prop. 7.5.19**

⇒: Assume two matches $(\rho, \mathsf{id}_I, c, d), (\rho', \mathsf{id}_{I'}, c', d')$ in an agent $a$ that are regarded the same, i.e.,

$$a = c \circ (\rho \centerdot R \otimes \mathsf{id}_{X_I}) \circ d = c' \circ (\rho \centerdot R \otimes \mathsf{id}_{X_{I'}}) \circ d'$$

$$\begin{array}{cccc} V_c = V_{c'} & E_c = E_{c'} & ctrl_c = ctrl_{c'} & prnt_c = prnt_{c'}\\ V_d = V_{d'} & E_d = E_{d'} = \emptyset & ctrl_d = ctrl_{d'} & prnt_d = prnt_{d'}\end{array}$$
$$link_c \restriction_{P_c \uplus Y_G} = link_{c'} \restriction_{P_c \uplus Y_G}$$

and there is a bijection $\alpha : X_{I'} \rightarrowtail X_I$ such that

$$link_c \circ \alpha = link_{c'} \restriction_{X_{I'}} \qquad\qquad link_d = \alpha \circ link_{d'}\,.$$

The matches are clearly decompositions

$$\begin{aligned} a &= c \circ ((\rho \centerdot R \otimes \mathsf{id}_{X_I}) \circ d \otimes \mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \circ (\mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \\ &= c' \circ ((\rho \centerdot R \otimes \mathsf{id}_{X_{I'}}) \circ d' \otimes \mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \circ (\mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \end{aligned}$$

and all but the following decomposition equivalence condition are trivially satisfied:

$$\begin{aligned} \mathit{link}_c \circ \mathit{link}_d \downharpoonleft^{X_I} &= \mathit{link}_c \circ (\alpha \circ \mathit{link}_{d'}) \downharpoonleft^{X_I} \\ &= \mathit{link}_c \circ \alpha \circ \mathit{link}_{d'} \downharpoonleft^{X_{I'}} \\ &= \mathit{link}_{d'} \circ \mathit{link}_{d'} \downharpoonleft^{X_{I'}} . \end{aligned}$$

$\Leftarrow$: Assume two decompositions of an agent $a$

$$\begin{aligned} a &= c \circ ((\rho \centerdot R \otimes \mathsf{id}_{X_I}) \circ d \otimes \mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \circ (\mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \\ &= c' \circ ((\rho \centerdot R \otimes \mathsf{id}_{X_{I'}}) \circ d' \otimes \mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \circ (\mathsf{id}_0 \otimes \mathsf{id}_\emptyset) \end{aligned}$$

where $R$ is a redex and $d,'d$ are discrete and ground, and assume that they are equivalent, i.e.,

$$\begin{array}{llll} V_c = V_{c'} & E_c = E_{c'} & \mathit{ctrl}_c = \mathit{ctrl}_{c'} & \mathit{prnt}_c = \mathit{prnt}_{c'} \\ V_d = V_{d'} & E_d = E_{d'} = \emptyset & \mathit{ctrl}_d = \mathit{ctrl}_{d'} & \mathit{prnt}_d = \mathit{prnt}_{d'} \\ \mathit{link}_c \upharpoonright_{P_c \uplus Y_G} = \mathit{link}_{c'} \upharpoonright_{P_c \uplus Y_G} & & \mathit{link}_C \circ \mathit{link}_D \downharpoonleft^{X_I} = \mathit{link}_{C'} \circ \mathit{link}_{D'} \downharpoonleft^{X_{I'}} . & \end{array}$$

Clearly, the decompositions are matches

$$a = c \circ (\rho \centerdot R \otimes \mathsf{id}_{X_I}) \circ d = c' \circ (\rho \centerdot R \otimes \mathsf{id}_{X_{I'}}) \circ d'$$

differing only by a bijection $\alpha : X_{I'} \rightarrowtail X_I$ defined by

$$\alpha = \mathit{link}_d \circ \mathit{link}_{d'}^{-1} .$$

□

**Proof of Prop. 7.5.21**

We first show that $\mathit{prmt}(\phi)$ and $\mathit{ctxt}(\phi)$ are indeed bigraphs:

From Prop. 7.5.9 we have that $\mathit{prmt}(\phi)^P$ and $\mathit{ctxt}(\phi)^P$ are place graphs, so we just need to show that the following are link graphs:

$$\begin{aligned} \mathit{prmt}(\phi)^L &= (V_D, \emptyset, \mathit{ctrl}_D, \mathit{link}_D) : X_D \to X_G \uplus X_I \\ \mathit{ctxt}(\phi)^L &= (V_C, E_C, \mathit{ctrl}_C, \mathit{link}_C) : Y_G \uplus X_I \uplus X_C \to Y_H . \end{aligned}$$

In both cases the node sets and control maps are shared with the corresponding place graphs and are thus well-defined, and thus we just need to show that $\mathit{link}_D$ and $\mathit{link}_C$ are well-defined:

$\mathit{link}_D$ : Since $\phi^{\mathsf{i}}$ is fully injective $(\phi^{\mathsf{i}})^{-1}$ is a function and thus so is $\mathit{link}_D$.

What remains to show is $\mathsf{dom}(\mathit{link}_D) = X_D \uplus P_D$ and $\mathsf{cod}(\mathit{link}_D) = X_G \uplus X_I$. Since, by definition, we have $\mathsf{dom}(\mathit{link}'_D) = P'_D \subseteq P_D$ and $\mathsf{cod}(\mathit{link}'_D) = X_I$, this amounts to showing $\mathsf{dom}((\phi^{\mathsf{i}})^{-1}) = X_D \uplus (P_D \setminus P'_D)$ and $\mathsf{rng}((\phi^{\mathsf{i}})^{-1}) \subseteq X_G \uplus X_I$. The latter is immediate since

$\mathsf{dom}(\phi^{\mathsf{i}}) = X_G$. To see $\mathsf{dom}((\phi^{\mathsf{i}})^{-1}) = \mathsf{rng}(\phi^{\mathsf{i}}) = X_D \uplus (P_D \setminus P'_D)$, we expand the definitions of $X_D$ and $P'_D$:

$$\begin{aligned} X_D \uplus (P_D \setminus P'_D) &= (\mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H) \uplus (P_D \setminus (P_D \setminus \mathsf{rng}(\phi^{\mathsf{i}}))) \\ &= (\mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H) \uplus (P_D \cap \mathsf{rng}(\phi^{\mathsf{i}})) \\ &= \mathsf{rng}(\phi^{\mathsf{i}}) \cap (X_H \uplus P_D). \end{aligned}$$

Thus $\mathsf{dom}((\phi^{\mathsf{i}})^{-1}) = X_D \uplus (P_D \setminus P'_D)$ if $\mathsf{rng}(\phi^{\mathsf{i}}) \subseteq X_H \uplus P_D$. But this is exactly the consistency condition (BGE-1) and must thus be satisfied since $\phi$ is an embedding.

$link_C$ : The constituent functions have the following domains and codomains:

$$\begin{aligned} \phi^{\mathsf{o}} &: Y_G \to E_H \uplus Y_H \\ link_H &: X_H \uplus P_H \to E_H \uplus Y_H \\ \mathsf{Id}_{P_C} &: P_C \to P_C \\ link'^{-1}_D &: X_I \to P'_D \\ \alpha_C &: X_C \to X'_C \end{aligned}$$

Since $link'_D$ is a bijection, $link'^{-1}_D : X_I \to P'_D$ is a function. By construction we have $X_I \# X_C$ and clearly $P_C \# X_I \uplus X_C$, so $\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C : P_C \uplus X_I \uplus X_C \to P_C \uplus P'_D \uplus X'_C$ is a function. By construction $X'_C \subseteq X_H$ and as shown in the proof of Theorem 7.5.9 we have $V_C, V_D \subseteq V_H$ and by definition $ctrl_C = ctrl_H \restriction_{V_C}$ and $ctrl_D = ctrl_H \restriction_{V_D}$, so $P_C \subseteq P_H$ and $P'_D \subseteq P_D \subseteq P_H$, and thus $link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)$ is a function. Also by construction we have $Y_G \# X_I$, $Y_G \# X_C$ and clearly $P_C \# Y_G$, so $link_C$ is a function.

What remains to show is $\mathsf{dom}(link_C) = Y_G \uplus X_I \uplus X_C \uplus P_C$ and $\mathsf{cod}(link_C) = E_C \uplus Y_H$. The first is immediate from the domains of the constituent functions. The latter amounts to showing $\mathsf{cod}(\phi^{\mathsf{o}}) = \mathsf{cod}(link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)) = E_C \uplus Y_H$. Noting $\mathsf{cod}(\phi^{\mathsf{o}}) : E_H \uplus Y_H$, $\mathsf{cod}(link_H) : E_H \uplus Y_H$, $\mathsf{cod}(link_H)$, and $E_C \uplus Y_H = (E_H \setminus \mathsf{rng}(\phi^{\mathsf{e}})) \uplus Y_H$, we just have to show $\mathsf{rng}(\phi^{\mathsf{o}}) \# \mathsf{rng}(\phi^{\mathsf{e}})$ and $\mathsf{rng}(link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)) \# \mathsf{rng}(\phi^{\mathsf{e}})$:

**$\mathsf{rng}(\phi^{\mathsf{o}}) \# \mathsf{rng}(\phi^{\mathsf{e}})$:** This is the first injectivity condition for link graph embeddings, condition (LGE-5), and is thus assumed to be satisfied.

**$\mathsf{rng}(link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)) \# \mathsf{rng}(\phi^{\mathsf{e}})$:** By the surjectivity condition (LGE-7), a point $p \in link_H^{-1}(e)$ of an edge $e \in \mathsf{rng}(\phi^{\mathsf{e}})$ in the image of edges must be in the image of points, i.e.,

$$\begin{aligned} p &\in \mathsf{rng}(\phi^{\mathsf{p}}) \\ &= \mathsf{rng}(\phi^{\mathsf{port}}) \uplus \mathsf{rng}(\phi^{\mathsf{i}}) \\ &\subseteq \mathsf{rng}(\phi^{\mathsf{port}}) \uplus X_H \uplus P_D. \end{aligned}$$

where the last inclusion follows from the consistency condition (BGE-1). Note that $\mathsf{rng}(\phi^{\mathsf{port}}) \# P_D$ and $\mathsf{rng}(\phi^{\mathsf{port}}) \# X_H$.

But the images of the three functions $\mathsf{Id}_{P_C}$, $link'^{-1}_D$, $\alpha_C$ are not in the image of points, which can be seen as follows:

**$P_C$:** Immediate from the construction of $P_C$:

$$P_C = P_H \setminus \mathsf{rng}(\phi^{\mathsf{port}}) \setminus P_D.$$

**$P'_D$:** Unfolding the construction of $P'_D$ it becomes immediate:

$$P'_D = P_D \setminus \mathsf{rng}(\phi^{\mathsf{i}}).$$

$X'_C$**:** Unfolding the construction of $X'_C$ it becomes immediate:

$$\begin{aligned} X'_C &= X_H \setminus X_D \\ &= X_H \setminus (\mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H) \\ &= X_H \setminus \mathsf{rng}(\phi^{\mathsf{i}}). \end{aligned}$$

We now turn to the matter of showing that the construction is defined up to decomposition equivalence. Since the place graph decomposition is defined up to decomposition equivalence, it suffices to show that any valid choices of $X_I$, $link'_D$, $X_C$, and $\alpha_C$ yield equivalent decompositions cf. Def. 7.5.18.

Assume that we have alternative choices:

$$\begin{aligned} X_{I'} &: \text{ a set of names satisfying} \\ &\quad |X_{I'}| = |P'_D|,\ X_{I'} \# X_G, \text{ and } X_{I'} \# Y_G \\ link'_{D'} &: P'_D \rightarrowtail X_{I'} \text{ a bijection} \\ X_{C'} &: \text{ a set of names satisfying} \\ &\quad |X_{C'}| = |X'_C|,\ X_{C'} \# Y_G, \text{ and } X_{C'} \# X_{I'} \\ \alpha_{C'} &: X_{C'} \rightarrowtail X'_C \text{ a bijection} \end{aligned}$$

and construct the corresponding link maps

$$\begin{aligned} link_{D'} &= (\phi^{\mathsf{i}})^{-1} \uplus link'_{D'} \\ link_{C'} &= \phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_{D'} \uplus \alpha_{C'}). \end{aligned}$$

Finally, we check the equivalence conditions:

$$\begin{aligned} link_D \downharpoonleft^{X_G} &= ((\phi^{\mathsf{i}})^{-1} \uplus link'_D) \downharpoonleft^{X_G} \\ &= (\phi^{\mathsf{i}})^{-1} \downharpoonleft^{X_G} \\ &= ((\phi^{\mathsf{i}})^{-1} \uplus link'_{D'}) \downharpoonleft^{X_G} \\ &= link_{D'} \downharpoonleft^{X_G} \\ link_C \restriction_{P_C \uplus Y_G} &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)) \restriction_{P_C \uplus Y_G} \\ &= (\phi^{\mathsf{o}} \uplus link_H \circ \mathsf{Id}_{P_C}) \restriction_{P_C \uplus Y_G} \\ &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_{C'})) \restriction_{P_C \uplus Y_G} \\ &= link_{C'} \restriction_{P_C \uplus Y_G} \\ link_C \circ \alpha_C^{-1} &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)) \circ \alpha_C^{-1} \\ &= link_H \\ &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_{C'})) \circ \alpha_{C'}^{-1} \\ &= link_{C'} \circ \alpha_{C'}^{-1} \\ link_C \circ link_D \downharpoonleft^{X_I} &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)) \circ ((\phi^{\mathsf{i}})^{-1} \uplus link'_D) \downharpoonleft^{X_I} \\ &= link_H \restriction_{P'_D} \\ &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_{D'} \uplus \alpha_{C'})) \circ ((\phi^{\mathsf{i}})^{-1} \uplus link'_{D'}) \downharpoonleft^{X_I} \\ &= link_{C'} \circ link_{D'} \downharpoonleft^{X_{I'}}\ . \end{aligned}$$

We now show that $prmt(\phi)$ and $ctxt(\phi)$ are indeed parameter and context for the embedding of $G$:

Let

$$\begin{aligned} D : \langle k_D, X_D \rangle \to \langle k_G, X_G \uplus X_I \rangle &= prmt(\phi), \\ C : \langle k_C, Y_G \uplus X_I \uplus X_C \rangle \to \langle m_H, Y_H \rangle &= ctxt(\phi), and \\ (V, E, ctrl, prnt, link) &= ctxt(\phi) \\ &\quad \circ ((\phi \cdot G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi) \otimes \mathsf{id}_{\langle |\tilde{k}_C|, X_C \rangle}) \\ &\quad \circ (\pi \otimes \mathsf{id}_{X_H}). \end{aligned}$$

As composition and tensor product of bigraphs are defined pointwise on the constituent place and link graphs (cf. Def. 7.2.6 and Def. 7.2.8), it is straightforward to see that the proof of Theorem 7.5.9 is also a proof of $V_H = V, ctrl_H = ctrl, prnt_H = prnt$, since this theorem only adds link graph structure.

By the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8) we have the following equalities:

$$\begin{aligned} E &= E_C \uplus \phi^{\mathsf{e}}(E_G) \uplus E_D && \text{Defs. 7.2.6 and 7.2.8} \\ &= (E_H \setminus \mathsf{rng}(\phi^{\mathsf{e}})) \uplus \phi^{\mathsf{e}}(E_G) \uplus \emptyset && \text{Def. 7.5.20} \\ &= E_H. \end{aligned}$$

To prove $link_H = link$, we construct the link map $link$ incrementally according to the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8), and then verify $link_H(p) = link(p)$:

**$\phi \cdot G$:** We write $link_{\phi \cdot G}$ for the link map of the resulting bigraph:

$$link_{\phi \cdot G} = (\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G}) \circ link_G \circ ((\phi^{\mathsf{p}})^{-1} \uplus \mathsf{Id}_{X_G}).$$

where

$$\phi^{\mathsf{p}}(v, i) = (\phi^{\mathsf{v}}(v), i).$$

Also, we write $P_{\phi \cdot G}$ for the ports of that bigraph: $P_{\phi \cdot G} = \phi^{\mathsf{p}}(P_G)$.

**$\phi \cdot G \otimes \mathsf{id}_{X_I}$:** We write $link_1$ for the link map of the resulting bigraph:

$$link_1 = link_{\phi \cdot G} \uplus \mathsf{Id}_{X_I}.$$

**$(\phi \cdot G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi)$:** We write $link_2$ for the link map of the resulting bigraph:

$$\begin{aligned} link_2(p) &= \begin{cases} link_D(p) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) \in \emptyset \\ link_1(x) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) = x \in X_G \uplus X_I \\ link_1(p) & \text{if } p \in P_{\phi \cdot G} \end{cases} \\ &= \begin{cases} link_1(x) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) = x \in X_G \uplus X_I \\ link_1(p) & \text{if } p \in P_{\phi \cdot G} \end{cases}. \end{aligned}$$

**$(\phi \cdot G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|} \otimes \alpha_C^{-1}$:** We write $link_3$ for the link map of the resulting bigraph:

$$link_3 = link_2 \uplus \alpha_C^{-1}.$$

$\pi \otimes \mathsf{id}_{X_H}$**:** We write $link_4$ for the link map of the resulting bigraph:

$$link_4 = \mathsf{Id}_\emptyset \uplus \mathsf{Id}_{X_H} = \mathsf{Id}_{X_H}.$$

$((\phi \bullet G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|} \otimes \alpha_C^{-1}) \circ (\pi \otimes \mathsf{id}_{X_H})$**:** We write $link_5$ for the link map of the resulting bigraph:

$$\begin{aligned} link_5(p) &= \begin{cases} link_4(p) & \text{if } p \in X_H \text{ and } link_4(p) \in \emptyset \\ link_3(x) & \text{if } p \in X_H \text{ and } link_4(p) = x \in X_H \\ link_3(p) & \text{if } p \in P_{\phi \bullet G} \uplus P_D \end{cases} \\ &= \begin{cases} link_3(x) & \text{if } p \in X_H \text{ and } \mathsf{Id}_{X_H}(p) = x \in X_H \\ link_3(p) & \text{if } p \in P_{\phi \bullet G} \uplus P_D \end{cases} \\ &= \begin{cases} link_3(p) & \text{if } p \in X_H \\ link_3(p) & \text{if } p \in P_{\phi \bullet G} \uplus P_D \end{cases} \\ &= link_3(p). \end{aligned}$$

$ctxt(\phi) \circ ((\phi \bullet G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|} \otimes \alpha_C^{-1}) \circ (\pi \otimes \mathsf{id}_{X_H})$**:** Finally, we have

$$link(p) = \begin{cases} link_3(p) & \text{if } p \in X_H \uplus P_{\phi \bullet G} \uplus P_D \text{ and } link_3(p) \in \phi^{\mathsf{e}}(E_G) \\ link_C(x) & \text{if } p \in X_H \uplus P_{\phi \bullet G} \uplus P_D \text{ and } link_3(p) = x \in Y_G \uplus X_I \uplus X_C \\ link_C(p) & \text{if } p \in P_C \end{cases}$$

Unfolding the definitions of the involved link maps, we get the following equalities:

$$\begin{aligned} link(p) &= \begin{cases} \alpha_C^{-1}(p) & \text{if } p \in X'_C \text{ and } \alpha_C^{-1}(p) \in \phi^{\mathsf{e}}(E_G) \\ link_2(p) & \text{if } p \in X_D \uplus P_{\phi \bullet G} \uplus P_D \text{ and } link_2(p) \in \phi^{\mathsf{e}}(E_G) \\ link_C(y) & \text{if } p \in X_D \uplus P_{\phi \bullet G} \uplus P_D \text{ and } link_2(p) = y \in Y_G \uplus X_I \\ link_C(y) & \text{if } p \in X'_C \text{ and } \alpha_C^{-1}(p) = y \in X_C \\ link_C(p) & \text{if } p \in P_C \end{cases} \\ &= \begin{cases} link_1(x) & \text{if } p \in X_D \uplus P_D \text{ and } link_1(x) \in \phi^{\mathsf{e}}(E_G) \text{ and } link_D(p) = x \in X_G \uplus X_I \\ link_1(p) & \text{if } p \in P_{\phi \bullet G} \text{ and } link_1(p) \in \phi^{\mathsf{e}}(E_G) \\ link_C(y) & \text{if } p \in X_D \uplus P_D \text{ and } link_1(x) = y \in Y_G \uplus X_I \text{ and } link_D(p) = x \in X_G \uplus X_I \\ link_C(y) & \text{if } p \in P_{\phi \bullet G} \text{ and } link_1(p) = y \in Y_G \\ link_C(y) & \text{if } p \in X'_C \text{ and } \alpha_C^{-1}(p) = y \in X_C \\ link_C(p) & \text{if } p \in P_C \end{cases} \\ &= \begin{cases} link_{\phi \bullet G}(x) & \text{if } p \in X_D \uplus P_D \text{ and } link_{\phi \bullet G}(x) \in \phi^{\mathsf{e}}(E_G) \text{ and } link_D(p) = x \in X_G \\ link_{\phi \bullet G}(p) & \text{if } p \in P_{\phi \bullet G} \text{ and } link_{\phi \bullet G}(p) \in \phi^{\mathsf{e}}(E_G) \\ link_C(x) & \text{if } p \in X_D \uplus P_D \text{ and } link_D(p) = x \in X_I \\ link_C(y) & \text{if } p \in X_D \uplus P_D \text{ and } link_{\phi \bullet G}(x) = y \in Y_G \text{ and } link_D(p) = x \in X_G \\ link_C(y) & \text{if } p \in P_{\phi \bullet G} \text{ and } link_{\phi \bullet G}(p) = y \in Y_G \\ link_C(y) & \text{if } p \in X'_C \text{ and } \alpha_C^{-1}(p) = y \in X_C \\ link_C(p) & \text{if } p \in P_C \end{cases}. \end{aligned}$$

Let us now verify $link = link_H$. $link$ is defined for $X_D \uplus X'_C \uplus P_C \uplus P_{\phi \bullet G} \uplus P_D$ and so is $link_H$ since $X_D \uplus X'_C = X_H$ and $V_C \uplus \phi^{\mathsf{v}}(V_G) \uplus V_D = V_H$, $ctrl_H \restriction_{V_C} \uplus ctrl_G \circ (\phi^{\mathsf{v}})^{-1} \uplus ctrl_H \restriction_{V_D} = ctrl_H$ which implies $P_C \uplus P_{\phi \bullet G} \uplus P_D = P_H$. Let us examine each case of $link(p)$ separately:

$p \in X_D \uplus P_D$ **and** $link_{\phi \blacksquare G}(x) \in \phi^{\mathsf{e}}(E_G)$ **and** $link_D(p) = x \in X_G$**:**

We have

$$\begin{aligned} link(p) &= link_{\phi \blacksquare G}(link_D(p)) \\ &= ((\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G}) \circ link_G \circ ((\phi^{\mathsf{p}})^{-1} \uplus \mathsf{Id}_{X_G}))(link_D(p)) \\ &= (\phi^{\mathsf{e}} \circ link_G)(link_D(p)) \\ &= (\phi^{\mathsf{e}} \circ link_G)(((\phi^{\mathsf{i}})^{-1} \uplus link'_D)(p)) \\ &= (\phi^{\mathsf{e}} \circ link_G)((\phi^{\mathsf{i}})^{-1}(p)) \\ &= (link_H \circ \phi^{\mathsf{p}'})((\phi^{\mathsf{i}})^{-1}(p)) \\ &= link_H(p) \end{aligned}$$

where

$$\phi^{\mathsf{p}'}(p) = \begin{cases} (\phi^{\mathsf{v}}(v), i) & \text{if } p = (v, i) \in P_G \\ \phi^{\mathsf{i}}(p) & \text{if } p \in X_G \end{cases}.$$

$p \in P_{\phi \blacksquare G}$ **and** $link_{\phi \blacksquare G}(p) \in \phi^{\mathsf{e}}(E_G)$**:**

We have

$$\begin{aligned} link(p) &= link_{\phi \blacksquare G}(p) \\ &= ((\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G}) \circ link_G \circ ((\phi^{\mathsf{p}})^{-1} \uplus \mathsf{Id}_{X_G}))(p) \\ &= (\phi^{\mathsf{e}} \circ link_G \circ (\phi^{\mathsf{p}})^{-1})(p) \\ &= (link_H \circ \phi^{\mathsf{p}'} \circ (\phi^{\mathsf{p}})^{-1})(p) \\ &= link_H(p) \end{aligned}$$

where

$$\phi^{\mathsf{p}'}(p) = \begin{cases} (\phi^{\mathsf{v}}(v), i) & \text{if } p = (v, i) \in P_G \\ \phi^{\mathsf{i}}(p) & \text{if } p \in X_G \end{cases}.$$

$p \in X_D \uplus P_D$ **and** $link_D(p) = x \in X_I$**:** We have

$$\begin{aligned} link(p) &= link_C(link_D(p)) \\ &= link_C(((\phi^{\mathsf{i}})^{-1} \uplus link'_D)(p)) \\ &= link_C(link'_D(p)) \\ &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C))(link'_D(p)) \\ &= (link_H \circ link'^{-1}_D)(link'_D(p)) \\ &= link_H(p). \end{aligned}$$

$p \in X_D \uplus P_D$ **and** $link_{\phi \blacksquare G}(x) = y \in Y_G$ **and** $link_D(p) = x \in X_G$**:**

We have

$$\begin{aligned}
\mathit{link}(p) &= \mathit{link}_C(\mathit{link}_{\phi\blacksquare G}(\mathit{link}_D(p)))\\
&= \mathit{link}_C(\mathit{link}_{\phi\blacksquare G}(((\phi^{\mathsf{i}})^{-1} \uplus \mathit{link}'_D)(p)))\\
&= \mathit{link}_C(\mathit{link}_{\phi\blacksquare G}((\phi^{\mathsf{i}})^{-1}(p)))\\
&= \mathit{link}_C(((\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G}) \circ \mathit{link}_G \circ ((\phi^{\mathsf{p}})^{-1} \uplus \mathsf{Id}_{X_G}))((\phi^{\mathsf{i}})^{-1}(p)))\\
&= \mathit{link}_C(\mathit{link}_G((\phi^{\mathsf{i}})^{-1}(p)))\\
&= (\phi^{\mathsf{o}} \uplus \mathit{link}_H \circ (\mathsf{Id}_{P_C} \uplus \mathit{link}'^{-1}_D \uplus \alpha_C))(\mathit{link}_G((\phi^{\mathsf{i}})^{-1}(p)))\\
&= \phi^{\mathsf{o}}(\mathit{link}_G((\phi^{\mathsf{i}})^{-1}(p)))\\
&= (\phi^{\mathsf{o}} \circ \mathit{link}_G)((\phi^{\mathsf{i}})^{-1}(p))\\
&= (\mathit{link}_H \circ \phi^{\mathsf{p}'})((\phi^{\mathsf{i}})^{-1}(p))\\
&= \mathit{link}_H(p)
\end{aligned}$$

where

$$\phi^{\mathsf{p}'}(p) = \begin{cases} (\phi^{\mathsf{v}}(v), i) & \text{if } p = (v,i) \in P_G \\ \phi^{\mathsf{i}}(p) & \text{if } p \in X_G \end{cases}.$$

**$p \in P_{\phi\blacksquare G}$ and $\mathit{link}_{\phi\blacksquare G}(p) = x \in Y_G$:**

We have

$$\begin{aligned}
\mathit{link}(p) &= \mathit{link}_C(\mathit{link}_{\phi\blacksquare G}(p))\\
&= \mathit{link}_C(((\phi^{\mathsf{e}} \uplus \mathsf{Id}_{Y_G}) \circ \mathit{link}_G \circ ((\phi^{\mathsf{p}})^{-1} \uplus \mathsf{Id}_{X_G}))(p))\\
&= \mathit{link}_C((\mathit{link}_G \circ (\phi^{\mathsf{p}})^{-1})(p))\\
&= (\phi^{\mathsf{o}} \uplus \mathit{link}_H \circ (\mathsf{Id}_{P_C} \uplus \mathit{link}'^{-1}_D \uplus \alpha_C))((\mathit{link}_G \circ (\phi^{\mathsf{p}})^{-1})(p))\\
&= \phi^{\mathsf{o}}((\mathit{link}_G \circ (\phi^{\mathsf{p}})^{-1})(p))\\
&= (\phi^{\mathsf{o}} \circ \mathit{link}_G \circ (\phi^{\mathsf{p}})^{-1})(p)\\
&= (\mathit{link}_H \circ \phi^{\mathsf{p}'} \circ (\phi^{\mathsf{p}})^{-1})(p)\\
&= \mathit{link}_H(p)
\end{aligned}$$

where

$$\phi^{\mathsf{p}'}(p) = \begin{cases} (\phi^{\mathsf{v}}(v), i) & \text{if } p = (v,i) \in P_G \\ \phi^{\mathsf{i}}(p) & \text{if } p \in X_G \end{cases}.$$

**$p \in X'_C$ and $\alpha_C^{-1}(p) = y \in X_C$:**

We have

$$\begin{aligned}
\mathit{link}(p) &= \mathit{link}_C(\alpha_C^{-1}(p))\\
&= (\phi^{\mathsf{o}} \uplus \mathit{link}_H \circ (\mathsf{Id}_{P_C} \uplus \mathit{link}'^{-1}_D \uplus \alpha_C))(\alpha_C^{-1}(p))\\
&= (\mathit{link}_H \circ \alpha_C)(\alpha_C^{-1}(p))\\
&= \mathit{link}_H(p).
\end{aligned}$$

**$p \in P_C$:**

We have

$$\begin{aligned} link(p) &= link_C(p) \\ &= (\phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C))(p) \\ &= link_H(p). \end{aligned}$$

□

**Proof of Theorem 7.5.22**

Def. 7.5.20 ∘ Def. 7.5.16 = $\mathsf{Id}$: Assume

$$H = C \circ ((G \otimes \mathsf{id}_{X_I}) \circ D \otimes \mathsf{id}_k \otimes \alpha) \circ (\pi \otimes \mathsf{id}_{X_H})$$

$$\begin{array}{lll} \phi^{\mathsf{v}} = \mathsf{Id}_{V_G} & \phi^{\mathsf{r}} = prnt_C \restriction_{m_G} & \phi^{\mathsf{s}} = (\mathsf{Id}_{V_D} \uplus \pi^{-1}) \circ prnt_D^{-1} \restriction_{k_G} \\ \phi^{\mathsf{e}} = \mathsf{Id}_{E_G} & \phi^{\mathsf{o}} = link_C \restriction_{Y_G} & \phi^{\mathsf{i}} = link_D^{-1} \restriction_{X_G} \end{array}$$

where $D$ is semi-discrete on $X_G$. Also, assume the results from the proof of Prop. 7.5.17.

It is clear that the place graph may be expressed as

$$H^P = C^P \circ (G^P \circ D^P \otimes \mathsf{id}_k) \circ \pi$$

and thus Theorem 7.5.10 applies, so using construction Def. 7.5.20 we obtain

$$\begin{aligned} prmt(\phi) &\stackrel{\text{def}}{=} (V_D, \emptyset, ctrl_D, prnt_D, link_{D'}) : \langle k_D, X_{D'} \rangle \to \langle k_G, X_G \uplus X_{I'} \rangle \\ ctxt(\phi) &\stackrel{\text{def}}{=} (V_C, E_{C'}, ctrl_C, prnt_C, link_{C'}) : \langle k_C, Y_G \uplus X_{I'} \uplus X_{C'} \rangle \to \langle m_H, Y_H \rangle \end{aligned}$$

$$\begin{aligned} P'_{D'} &= P_D \setminus \mathsf{rng}(\phi^{\mathsf{i}}) \\ X_{D'} &= \mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H \\ X_{I'} &: \text{ a set of names satisfying} \\ &\qquad |X_{I'}| = |P'_{D'}|,\ X_{I'} \# X_G, \text{ and } X_{I'} \# Y_G \\ link'_{D'} &: P'_{D'} \rightarrowtail X_{I'} \text{ a bijection} \\ link_{D'} &= (\phi^{\mathsf{i}})^{-1} \uplus link'_{D'} \end{aligned}$$

$$\begin{aligned} E_{C'} &= E_H \setminus \mathsf{rng}(\phi^{\mathsf{e}}) \\ X'_{C'} &= X_H \setminus X_{D'} \\ X_{C'} &: \text{ a set of names satisfying} \\ &\qquad |X_{C'}| = |X'_{C'}|,\ X_{C'} \# Y_G, \text{ and } X_{C'} \# X_{I'} \\ \alpha_{C'} &: X_{C'} \rightarrowtail X'_{C'} \text{ a bijection} \\ link_{C'} &= \phi^{\mathsf{o}} \uplus link_H \circ (\mathsf{Id}_{P_C} \uplus link'^{-1}_{D'} \uplus \alpha_{C'}) \end{aligned}$$

$$H = ctxt(\phi) \circ ((\phi \centerdot G \otimes \mathsf{id}_{X_{I'}}) \circ prmt(\phi) \otimes \mathsf{id}_k \otimes \alpha^{-1}_{C'}) \circ (\pi \otimes \mathsf{id}_{X_H}).$$

Thus it suffices to show $D^L = prmt(\phi)^L$, $C^L = ctxt(\phi)^L$, $X_I = X_{I'}$, and $\alpha = \alpha^{-1}_{C'}$.

Let us first unfold some of the definitions (noting that $X_D \subseteq X_H$ and that $D$ is semi-discrete

on $X_G$):

$$\begin{aligned}
P'_D &= P_D \setminus \mathsf{rng}(\phi^{\mathsf{i}}) \\
&= P_D \setminus \mathsf{rng}(link_D^{-1}\restriction_{X_G}) \\
&= P_D \setminus link_D^{-1}(X_G) \\
X_{D'} &= \mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H \\
&= \mathsf{rng}(link_D^{-1}\restriction_{X_G}) \cap X_H \\
&= \mathsf{rng}(link_D^{-1}\restriction_{X_G}) \cap X_H \\
&= X_D \\
E_{C'} &= E_H \setminus \mathsf{rng}(\phi^{\mathsf{e}}) \\
&= E_H \setminus E_G \\
&= E_C \\
X'_{C'} &= X_H \setminus X_{D'} \\
&= X_H \setminus X_D \\
&= X'_C .
\end{aligned}$$

With these in mind, we proceed to prove $D^L = prmt(\phi)^L$, $C^L = ctxt(\phi)^L$, $X_I = X_{I'}$, $X_C = X_{C'}$, and $\alpha = \alpha_{C'}^{-1}$:

**$X_I = X_{I'}$:** Remember that we are free to choose $X_{I'}$ as it is internal to the decomposition, as long as it satisfies $|X_{I'}| = |P_D|$, $X_{I'} \# X_G$, and $X_{I'} \# Y_G$. From the initial decomposition we know that $X_I$ satisfies these conditions, and thus we simply choose $X_{I'} = X_I$.

Similarly, we are free to choose a suitable bijection $link'_{D'} : P'_D \rightarrowtail X_I$, so we simply choose $link'_{D'} = link_D\restriction_{P'_D}$.

**$X_C = X_{C'}$:** Again, we are free to choose $X_{C'}$ as it is internal to the decomposition, as long as it satisfies $|X_{C'}| = |X'_{C'}|$, $X_{C'} \# Y_G$, and $X_{C'} \# X_{I'}$. From the initial decomposition we know that $X_C$ satisfies these conditions, and thus we simply choose $X_{C'} = X_C$.

**$\alpha = \alpha_{C'}^{-1}$:** Again, we are free to choose $\alpha_{C'}$ as it is internal to the decomposition. So we simply choose $\alpha_{C'} = \alpha^{-1}$.

**$D^L = prmt(\phi)^L$:** Since $D^P = prmt(\phi)^P$ it suffices to show $E_D = \emptyset$, and $link_D = link_{D'}$:

**$E_D = \emptyset$:** Satisfied since $D$ is semi-discrete.

**$link_D = link_{D'}$:** Easily seen by expanding the definitions:

$$\begin{aligned}
link_{D'} &= (\phi^{\mathsf{i}})^{-1} \uplus link'_{D'} \\
&= (link_D^{-1}\restriction_{X_G})^{-1} \uplus link_D\restriction_{P'_D} \\
&= link_D\downharpoonleft^{X_G} \uplus link_D\restriction_{P_D \setminus link_D^{-1}(X_G)} \\
&= link_D .
\end{aligned}$$

**$C^L = ctxt(\phi)^L$:** Since $C^P = ctxt(\phi)^P$ and $E_{C'} = E_C$ it suffices to show $link_C = link_{C'}$ which is

easily seen by expanding the definitions:

$$\begin{aligned}
link_{C'} &= \phi^{\circ} \uplus link_H \circ(\mathsf{Id}_{P_C} \uplus link'^{-1}_{D'} \uplus \alpha_{C'}) \\
&= link_C\restriction_{Y_G} \uplus link_H \circ(\mathsf{Id}_{P_C} \uplus (link_D\restriction_{P'_D})^{-1} \uplus \alpha^{-1}) \\
&= link_C\restriction_{Y_G} \uplus link_H\restriction_{P_C} \uplus link_H \circ(link_D\restriction_{P_D \setminus link_D^{-1}(X_G)})^{-1} \uplus link_H \circ\alpha^{-1}) \\
&= link_C\restriction_{Y_G} \uplus link_C\restriction_{P_C} \uplus link_H \circ(link_D \downharpoonright^{X_I})^{-1} \uplus link_C\restriction_{X_C} \\
&= link_C\restriction_{Y_G} \uplus link_C\restriction_{P_C} \uplus link_C\restriction_{X_I} \uplus link_C\restriction_{X_C} \\
&= link_C \,.
\end{aligned}$$

Def. 7.5.16 $\circ$ Def. 7.5.20 $=$ Id: Assume a bigraph $G : \langle k_G, X_G\rangle \to \langle m_G, Y_G\rangle$ and an embedding $\phi : G \hookrightarrow H$ into a bigraph $H : \langle k_H, X_H\rangle \to \langle m_H, Y_H\rangle$ (for simplicity, assume $\phi \bullet G = G$),

$$\begin{aligned}
prmt(\phi) &\stackrel{\text{def}}{=} (V_D, \emptyset, ctrl_D, prnt_D, link_D) : \langle k_D, X_D\rangle \to \langle k_G, X_G \uplus X_I\rangle \\
ctxt(\phi) &\stackrel{\text{def}}{=} (V_C, E_C, ctrl_C, prnt_C, link_C) : \langle k_C, Y_G \uplus X_I \uplus X_C\rangle \to \langle m_H, Y_H\rangle
\end{aligned}$$

$$\begin{aligned}
(V_D, ctrl_D, prnt_D) : k_D \to k_G &= prmt(\phi^P) \\
P'_D &= P_D \setminus \mathsf{rng}(\phi^{\mathsf{i}}) \\
X_D &= \mathsf{rng}(\phi^{\mathsf{i}}) \cap X_H \\
X_I &: \text{ a set of names satisfying} \\
&\qquad |X_I| = |P'_D|,\ X_I \# X_G, \text{ and } X_I \# Y_G \\
link'_D &: P'_D \rightarrowtail X_I \text{ a bijection} \\
link_D &= (\phi^{\mathsf{i}})^{-1} \uplus link'_D
\end{aligned}$$

$$\begin{aligned}
(V_C, ctrl_C, prnt_C) : k_C \to m_H &= ctxt(\phi^P) \\
E_C &= E_H \setminus \mathsf{rng}(\phi^{\mathsf{e}}) \\
X'_C &= X_H \setminus X_D \\
X_C &: \text{ a set of names satisfying} \\
&\qquad |X_C| = |X'_C|,\ X_C \# Y_G, \text{ and } X_C \# X_I \\
\alpha_C &: X_C \rightarrowtail X'_C \text{ a bijection} \\
link_C &= \phi^{\circ} \uplus link_H \circ(\mathsf{Id}_{P_C} \uplus link'^{-1}_D \uplus \alpha_C)
\end{aligned}$$

$$\begin{aligned}
H &= ctxt(\phi) \circ ((G \otimes \mathsf{id}_{X_I}) \circ prmt(\phi) \otimes \mathsf{id}_{|\tilde{k}_C|} \otimes \alpha_C^{-1}) \circ (\pi \otimes \mathsf{id}_{X_H}) \\
\pi &= f_D^{-1} \uplus f'^{-1} : k_H \to k_H \\
f'(i + k_D) &= f'_C(i) \quad \text{for } i \in |\tilde{k}_C|.
\end{aligned}$$

where we assume that the name sets and bijections are chosen as in the previous proof case, i.e.,

$$X_{I'} = X_I \qquad X_{C'} = X_C \qquad link'_{D'} = link_D\restriction_{P'_D} \qquad \alpha_{C'} = \alpha^{-1}.$$

It is clear that the place graph may be expressed as

$$H^P = ctxt(\phi)^P \circ (G^P \circ prmt(\phi)^P \otimes \mathsf{id}_{|\tilde{k}_C|}) \circ \pi$$

and thus Theorem 7.5.10 applies, so using construction Def. 7.5.16 we obtain

$$\phi' = \phi^{\mathsf{v}} \uplus \phi'^{\mathsf{e}} \uplus \phi^{\mathsf{s}} \uplus \phi^{\mathsf{r}} \uplus \phi'^{\mathsf{i}} \uplus \phi'^{\mathsf{o}} : G \hookrightarrow H$$
$$\phi'^{\mathsf{e}} = \mathsf{Id}_{E_G} \qquad \phi'^{\mathsf{o}} = \mathit{link}_C \restriction_{Y_G} \qquad \phi'^{\mathsf{i}} = \mathit{link}_D^{-1} \restriction_{X_G} .$$

We must prove $\phi = \phi'$, and it suffices to show $\phi^{\mathsf{e}} = \phi'^{\mathsf{e}}$, $\phi^{\mathsf{o}} = \phi'^{\mathsf{o}}$, and $\phi^{\mathsf{i}} = \phi'^{\mathsf{i}}$.

**$\phi^{\mathsf{e}} = \phi'^{\mathsf{e}}$:** Satisfied by assumption.

**$\phi^{\mathsf{o}} = \phi'^{\mathsf{o}}$:** Easily seen by unfolding the definitions:

$$\begin{aligned}\phi'^{\mathsf{o}} &= \mathit{link}_C \restriction_{Y_G} \\ &= (\phi^{\mathsf{o}} \uplus \mathit{link}_H \circ (\mathsf{Id}_{P_C} \uplus \mathit{link}'^{-1}_D \uplus \alpha_C)) \restriction_{Y_G} \\ &= \phi^{\mathsf{o}}.\end{aligned}$$

**$\phi^{\mathsf{i}} = \phi'^{\mathsf{i}}$:** Easily seen by unfolding the definitions:

$$\begin{aligned}\phi'^{\mathsf{i}} &= \mathit{link}_D^{-1} \restriction_{X_G} \\ &= ((\phi^{\mathsf{i}})^{-1} \uplus \mathit{link}'_D)^{-1} \restriction_{X_G} \\ &= ((\phi^{\mathsf{i}})^{-1} \uplus \mathit{link}_D \restriction_{P'_D})^{-1} \restriction_{X_G} \\ &= ((\phi^{\mathsf{i}})^{-1} \uplus \mathit{link}_D \restriction_{P_D \setminus \mathit{link}_D^{-1}(X_G)})^{-1} \restriction_{X_G} \\ &= \phi^{\mathsf{i}}.\end{aligned}$$

□

### 7.A.2 Bigraph Edit Scripts

**Proof of Prop. 7.6.13**

It is straightforward to check that $\tilde{H}'$ is a pattern, since $\delta$ is compatible with $\tilde{P}$ and $\phi$ is an embedding and thus satisfies the embedding conditions of Def. 7.5.1 Def. 7.5.4, and Def. 7.5.14. Also, $\tilde{H}'$ clearly has the same outer face and inner names as $\tilde{H}$ since mediated edits can only affect the set of inner variables of a patterns interfaces.

What remains is to check that for each mediated edit, $\phi' : \tilde{P}' \hookrightarrow \tilde{H}'$ is an embedding, i.e., that it satisfies the embedding conditions. In most cases this follows easily from the fact that $\phi$ satisfies the conditions and we shall omit these, but a few cases are more interesting. Note that, by Prop. 7.6.7, $\tilde{P}'$ is a pattern.

**$\oplus_{v:K_{\vec{y}}@p}$:** We have $v \notin V_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R_{\tilde{P}}$, $v' \notin V_{\tilde{H}}$, and

$$\begin{aligned}\tilde{P}' &= (V_{\tilde{P}} + v, E_{\tilde{P}}, \mathit{ctrl}_{\tilde{P}}[v \mapsto K], \mathit{prnt}_{\tilde{P}}[v \mapsto p], \\ &\qquad \mathit{link}_{\tilde{P}}[(v,0) \mapsto \vec{y}_0, \ldots, (v, n-1) \mapsto \vec{y}_{n-1}]) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\ \tilde{H}' &= (V_{\tilde{H}} + v', E_{\tilde{H}}, \mathit{ctrl}_{\tilde{H}}[v' \mapsto K], \mathit{prnt}_{\tilde{H}}[v' \mapsto \phi(p)], \\ &\qquad \mathit{link}_{\tilde{H}}[(v',0) \mapsto \phi(\vec{y}_0), \ldots, (v', n-1) \mapsto \phi(\vec{y}_{n-1})]) \\ \phi' &= \phi[v \mapsto v'].\end{aligned}$$

The interesting cases are:

**(LGE-7)** Assuming $e \in E_{\tilde{P}}$ we have the following equalities:

$$\begin{aligned}
&(\phi'^{\mathsf{p}} \circ (link_{\tilde{P}}[(v,0) \mapsto \vec{y}_0, \ldots, (v,n-1) \mapsto \vec{y}_{n-1}])^{-1})(e) \\
=&\phi'^{\mathsf{p}}(link_{\tilde{P}}^{-1}(e) \cup [(v,0) \mapsto \vec{y}_0, \ldots, (v,n-1) \mapsto \vec{y}_{n-1}]^{-1}(e)) \\
=&(\phi'^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e) \cup (\phi'^{\mathsf{p}} \circ [(v,0) \mapsto \vec{y}_0, \ldots, (v,n-1) \mapsto \vec{y}_{n-1}]^{-1})(e) \\
=&(\phi^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e) \cup [(v',0) \mapsto \vec{y}_0, \ldots, (v',n-1) \mapsto \vec{y}_{n-1}]^{-1}(e) \\
=&(link_{\tilde{H}}^{-1} \circ \phi^{\mathsf{e}})(e) \cup ([(v',0) \mapsto \phi(\vec{y}_0), \ldots, (v',n-1) \mapsto \phi(\vec{y}_{n-1})]^{-1} \circ \phi'^{\mathsf{e}})(e) \\
=&(link_{\tilde{H}}^{-1} \circ \phi'^{\mathsf{e}})(e) \cup ([(v',0) \mapsto \phi(\vec{y}_0), \ldots, (v',n-1) \mapsto \phi(\vec{y}_{n-1})]^{-1} \circ \phi'^{\mathsf{e}})(e) \\
=&((link_{\tilde{H}}[(v',0) \mapsto \phi(\vec{y}_0), \ldots, (v',n-1) \mapsto \phi(\vec{y}_{n-1})])^{-1} \circ \phi'^{\mathsf{e}})(e).
\end{aligned}$$

**(PGE-7)** We check the two cases for $w \in V_{\tilde{P}} + v$:

$w \in V_{\tilde{P}}$**:** We have the following equalities:

$$\begin{aligned}
&(\phi'^{\mathsf{c}} \circ (prnt_{\tilde{P}}[v \mapsto p])^{-1})(w) \\
=&\phi'^{\mathsf{c}}(prnt_{\tilde{P}}^{-1}(w) \cup [v \mapsto p]^{-1}(w)) \\
=&(\phi'^{\mathsf{c}} \circ prnt_{\tilde{P}}^{-1})(w) \cup (\phi'^{\mathsf{c}} \circ [v \mapsto p]^{-1})(w) \\
=&(\phi^{\mathsf{c}} \circ prnt_{\tilde{P}}^{-1})(w) \cup [v' \mapsto p]^{-1}(w) \\
=&(prnt_{\tilde{H}}^{-1} \circ \phi^{\mathsf{v}})(w) \cup ([v' \mapsto \phi(p)]^{-1} \circ \phi'^{\mathsf{v}})(w) \\
=&(prnt_{\tilde{H}}^{-1} \circ \phi'^{\mathsf{v}})(w) \cup ([v' \mapsto \phi(p)]^{-1} \circ \phi'^{\mathsf{v}})(w) \\
=&((prnt_{\tilde{H}}[v' \mapsto \phi(p)])^{-1} \circ \phi'^{\mathsf{v}})(w).
\end{aligned}$$

$w = v$**:** Since $v \notin V_{\tilde{P}}$ and $v' \notin V_{\tilde{H}}$ we have $v \neq p$, $v \notin \mathsf{rng}(prnt_{\tilde{P}})$, $v' \neq \phi(p)$, and $v' \notin \mathsf{rng}(prnt_{\tilde{H}})$. Thus $(prnt_{\tilde{P}}[v \mapsto p])^{-1}(v) = \emptyset = (prnt_{\tilde{H}}[v' \mapsto \phi(p)])^{-1}(v')$.

$\oplus_e$ : We have $e \notin E_{\tilde{P}}$, $e' \notin E_{\tilde{H}}$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}} + e, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\
\tilde{H}' &= (V_{\tilde{H}}, E_{\tilde{H}} + e', ctrl_{\tilde{H}}, prnt_{\tilde{H}}, link_{\tilde{H}}) \\
\phi' &= \phi[e \mapsto e'].
\end{aligned}$$

All the conditions are obviously satisfied.

$\ominus_v$ : We have $v \in V_{\tilde{P}}$, $prnt_{\tilde{P}}^{-1}(v) = \emptyset$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}} - v, E_{\tilde{P}}, ctrl_{\tilde{P}} - v, prnt_{\tilde{P}} - v, link_{\tilde{P}} - P_v) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\
\tilde{H}' &= (V_{\tilde{H}} - \phi(v), E_{\tilde{H}}, ctrl_{\tilde{H}} - \phi(v), prnt_{\tilde{H}} - \phi(v), link_{\tilde{H}} - P_{\phi(v)}) \\
\phi' &= \phi - v.
\end{aligned}$$

The interesting cases are:

**(LGE-7)** Assuming $e \in E_{\tilde{P}}$ we have the following equalities:

$$\begin{aligned}
&(\phi'^{\mathsf{p}} \circ (link_{\tilde{P}} - P_v)^{-1})(e) \\
=&\phi^{\mathsf{p}}(link_{\tilde{P}}^{-1}(e) \setminus P_v) \\
=&(\phi^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e) \setminus \phi(P_v) \\
=&(link_{\tilde{H}}^{-1} \circ \phi^{\mathsf{e}})(e) \setminus P_{\phi(v)} \\
=&((link_{\tilde{H}} - P_{\phi(v)})^{-1} \circ \phi'^{\mathsf{e}})(e).
\end{aligned}$$

**(PGE-7)** Assuming $w \in V_{\tilde{P}} - v$ we have the following equalities:

$$\begin{aligned}&(\phi'^{\mathsf{c}} \circ (prnt_{\tilde{P}} - v)^{-1})(w)\\ =&\phi^{\mathsf{c}}(prnt_{\tilde{P}}^{-1}(w) - v)\\ =&(\phi^{\mathsf{c}} \circ prnt_{\tilde{P}}^{-1})(w) - \phi(v)\\ =&(prnt_{\tilde{H}}^{-1} \circ \phi^{\mathsf{v}})(w) - \phi(v)\\ =&((prnt_{\tilde{H}} - \phi(v))^{-1} \circ \phi'^{\mathsf{v}})(w).\end{aligned}$$

$\ominus_e$ : We have $e \in E_{\tilde{P}}$, $link_{\tilde{P}}^{-1}(e) = \emptyset$, and

$$\begin{aligned}\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}} - e, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}}\rangle\\ \tilde{H}' &= (V_{\tilde{H}}, E_{\tilde{H}} - \phi(e), ctrl_{\tilde{H}}, prnt_{\tilde{H}}, link_{\tilde{H}})\\ \phi' &= \phi - e.\end{aligned}$$

All the conditions are obviously satisfied.

$\ominus_q$**:** We have $q \in Q_{\tilde{P}}$, and

$$\begin{aligned}\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}} - q, link_{\tilde{P}}) : (Q_{\tilde{P}} - q) \to \langle R_{\tilde{P}}, Y_{\tilde{P}}\rangle\\ \tilde{H}' &= (V_{\tilde{H}} \setminus \tilde{H}\downharpoonleft^{\phi(q)}, E_{\tilde{H}}, ctrl_{\tilde{H}} - \tilde{H}\downharpoonleft^{\phi(q)}, prnt_{\tilde{H}} - \tilde{H}\downharpoonleft^{\phi(q)}, link_{\tilde{H}} - P_{\tilde{H}\downharpoonleft^{\phi(q)}})\\ &\quad : \langle Q_{\tilde{H}} \setminus \tilde{H}\downharpoonleft^{\phi(q)}, X_{\tilde{H}}\rangle \to I\\ \phi' &= \phi - q.\end{aligned}$$

The interesting cases are:

**(LGE-7)** Assuming $e \in E_{\tilde{P}}$ we have the following equalities:

$$\begin{aligned}&(\phi'^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e)\\ =&(\phi^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e) \setminus P_{\tilde{H}\downharpoonleft^{\phi(q)}}\\ =&(link_{\tilde{H}}^{-1} \circ \phi^{\mathsf{e}})(e) \setminus P_{\tilde{H}\downharpoonleft^{\phi(q)}}\\ =&((link_{\tilde{H}} - P_{\tilde{H}\downharpoonleft^{\phi(q)}})^{-1} \circ \phi'^{\mathsf{e}})(e)\end{aligned}$$

since $\mathsf{rng}(\phi^{\mathsf{p}}) = \mathsf{rng}(\phi^{\mathsf{port}}) \# P_{\tilde{H}\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}} \supseteq P_{\tilde{H}\downharpoonleft^{\phi(q)}}$, cf. Corollary 7.5.24.

**(PGE-7)** Assuming $w \in V_{\tilde{P}}$ we have the following equalities:

$$\begin{aligned}&(\phi'^{\mathsf{c}} \circ (prnt_{\tilde{P}} - q)^{-1})(w)\\ =&\phi^{\mathsf{c}}(prnt_{\tilde{P}}^{-1}(w) - q) \setminus (\tilde{H}\downharpoonleft^{\phi^{\mathsf{s}}(q)} \setminus \phi^{\mathsf{s}}(q))\\ =&((\phi^{\mathsf{c}} \circ prnt_{\tilde{P}}^{-1})(w) \setminus \phi(q)) \setminus (\tilde{H}\downharpoonleft^{\phi^{\mathsf{s}}(q)} \setminus \phi^{\mathsf{s}}(q))\\ =&(prnt_{\tilde{H}}^{-1} \circ \phi^{\mathsf{v}})(w) \setminus \tilde{H}\downharpoonleft^{\phi^{\mathsf{s}}(q)}\\ =&((prnt_{\tilde{H}} - \tilde{H}\downharpoonleft^{\phi^{\mathsf{s}}(q)})^{-1} \circ \phi'^{\mathsf{v}})(w)\end{aligned}$$

since $\mathsf{rng}(\phi^{\mathsf{c}}) \# (\tilde{H}\downharpoonleft^{\phi^{\mathsf{s}}(q)} \setminus \phi^{\mathsf{s}}(q))$, cf. Lemma 7.5.11.

$\oslash_{v@p}$**:** We have $v \in V_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R_{\tilde{P}}$, and

$$\begin{aligned}\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[v \mapsto p], link_{\tilde{P}}) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}}\rangle\\ \tilde{H}' &= (V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}, prnt_{\tilde{H}}[\phi(v) \mapsto \phi(p)], link_{\tilde{H}})\\ \phi' &= \phi.\end{aligned}$$

All the conditions are obviously satisfied.

$\oslash_{q@p}$**:** We have $q \in Q_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R_{\tilde{P}}$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[q \mapsto p], link_{\tilde{P}}) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\
\tilde{H}' &= (V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}, prnt_{\tilde{H}}[\phi(q) \mapsto \phi(p)], link_{\tilde{H}}) \\
\phi' &= \phi.
\end{aligned}$$

All the conditions are obviously satisfied.

$\otimes_{q \to r@p}$**:** We have $q \in Q_{\tilde{P}}$, $r \notin Q_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R_{\tilde{P}}$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[r \mapsto p], link_{\tilde{P}}) : (Q_{\tilde{P}} + r) \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\
\tilde{H}' &= (V_{\tilde{H}} \uplus V_r, E_{\tilde{H}}, ctrl_{\tilde{H}} \uplus ctrl_r, prnt_{\tilde{H}} \uplus prnt_r, link_{\tilde{H}} \uplus link_r) \\
&\quad : \langle Q_{\tilde{H}} \uplus Q_r, X \rangle \to I \\
\phi' &= \phi[r \mapsto f^{-1}(\phi(q))]
\end{aligned}$$

where

$$\begin{aligned}
V_q &= \tilde{H} \downharpoonleft^{\phi(q)} \cap V_{\tilde{H}} & Q_q &= \tilde{H} \downharpoonleft^{\phi(q)} \cap Q \\
|V_r| &= |V_q| & |Q_r| &= |Q_q| \\
V_r &\# V_{\tilde{H}} & Q_r &\# Q \\
f_v &: V_r \rightarrowtail V_q & f_s &: Q_r \rightarrowtail Q_q \\
f &= f_v \uplus f_s \\
ctrl_r &= ctrl_{\tilde{H}} \circ f_v \\
prnt_r &= \{f^{-1}(\phi(q)) \mapsto \phi(p)\} \uplus f_v^{-1} \circ prnt_{\tilde{H}} \circ (f - f^{-1}(\phi(q))) \\
link_r(v, i) &= link_{\tilde{H}}(f_v(v), i) \quad (v \in V_r)
\end{aligned}$$

The interesting cases are:

**(LGE-7)** Assuming $e \in E_{\tilde{P}}$ we have the following equalities:

$$\begin{aligned}
&(\phi'^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e) \\
=&(\phi^{\mathsf{p}} \circ link_{\tilde{P}}^{-1})(e) \\
=&(link_{\tilde{H}}^{-1} \circ \phi^{\mathsf{e}})(e) \\
=&(link_{\tilde{H}}^{-1} \circ \phi'^{\mathsf{e}})(e) \cup (link_r^{-1} \circ \phi'^{\mathsf{e}})(e) \\
=&((link_{\tilde{H}} \uplus link_r)^{-1} \circ \phi'^{\mathsf{e}})(e)
\end{aligned}$$

since $link_{\tilde{H}}^{-1}(\mathsf{rng}(\phi^{\mathsf{e}})) \subseteq \mathsf{rng}(\phi^{\mathsf{p}}) = \mathsf{rng}(\phi^{\mathsf{port}}) \# P_{\tilde{H}\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}} \supseteq P_{\tilde{H}\downharpoonleft^{\phi(q)}}$, cf. condition (LGE-7) and Corollary 7.5.24, and thus $link_r^{-1}(\mathsf{rng}(\phi'^{\mathsf{e}})) = \phi'^{\mathsf{p}}((link_{\tilde{H}} \upharpoonright_{P_{V_q}})^{-1}(\mathsf{rng}(\phi'^{\mathsf{e}}))) = \phi'^{\mathsf{p}}((link_{\tilde{H}} \upharpoonright_{P_{\tilde{H}\downharpoonleft^{\phi(q)}}})^{-1}(\mathsf{rng}(\phi^{\mathsf{e}}))) = \phi'^{\mathsf{p}}(\emptyset) = \emptyset$.

**(PGE-7)** Assuming $w \in V_{\tilde{P}}$ we have the following equalities:

$$\begin{aligned}
&(\phi'^{\mathsf{c}} \circ (prnt_{\tilde{P}}[r \mapsto p])^{-1})(w) \\
=&\phi'^{\mathsf{c}}(prnt_{\tilde{P}}^{-1}(w) \cup [r \mapsto p]^{-1}(w)) \\
=&(\phi'^{\mathsf{c}} \circ prnt_{\tilde{P}}^{-1})(w) \cup (\phi'^{\mathsf{c}} \circ [r \mapsto p]^{-1})(w) \\
=&(\phi^{\mathsf{c}} \circ prnt_{\tilde{P}}^{-1})(w) \cup [f^{-1}(\phi(q)) \mapsto p]^{-1}(w) \\
=&(prnt_{\tilde{H}}^{-1} \circ \phi^{\mathsf{v}})(w) \cup ([f^{-1}(\phi(q)) \mapsto \phi'^{\mathsf{v}}(p)]^{-1} \circ \phi'^{\mathsf{v}})(w) \\
&\cup ((f_v^{-1} \circ prnt_{\tilde{H}} \circ (f - f^{-1}(\phi(q))))^{-1} \circ \phi'^{\mathsf{v}})(w) \\
=&(prnt_{\tilde{H}}^{-1} \circ \phi'^{\mathsf{v}})(w) \\
&\cup (([f^{-1}(\phi(q)) \mapsto \phi'^{\mathsf{v}}(p)] \uplus f_v^{-1} \circ prnt_{\tilde{H}} \circ (f - f^{-1}(\phi(q))))^{-1} \circ \phi'^{\mathsf{v}})(w) \\
=&(prnt_{\tilde{H}}^{-1} \circ \phi'^{\mathsf{v}})(w) \cup (prnt_r^{-1} \circ \phi'^{\mathsf{v}})(w) \\
=&((prnt_{\tilde{H}} \uplus prnt_r)^{-1} \circ \phi'^{\mathsf{v}})(w)
\end{aligned}$$

since $\mathsf{rng}(f_v^{-1}) = V_r \# V_{\tilde{H}} \supseteq \mathsf{rng}(\phi'^{\mathsf{v}})$ and thus $((f_v^{-1} \circ prnt_{\tilde{H}} \circ (f - f^{-1}(\phi(q))))^{-1} \circ \phi'^{\mathsf{v}})(w) = \emptyset$.

$\odot_{(v,i)\mapsto l}$**:** We have $v \in V_{\tilde{P}}$, $i \in ar(ctrl_{\tilde{P}}(v))$, $l \in E_{\tilde{P}} \uplus Y_{\tilde{P}}$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}[(v,i) \mapsto l]) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\
\tilde{H}' &= (V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}, prnt_{\tilde{H}}, link_{\tilde{H}}[(\phi(v), i) \mapsto \phi(l)]) \\
\phi' &= \phi.
\end{aligned}$$

All the conditions are obviously satisfied.

$\circledcirc_{v:K}$**:** We have $v \in V_{\tilde{P}}$, $ar(K) = ar(ctrl_{\tilde{P}}(v))$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}[v \mapsto K], prnt_{\tilde{P}}, link_{\tilde{P}}) : Q_{\tilde{P}} \to \langle R_{\tilde{P}}, Y_{\tilde{P}} \rangle \\
\tilde{H}' &= (V_{\tilde{H}}, E_{\tilde{H}}, ctrl_{\tilde{H}}[\phi(v) \mapsto K], prnt_{\tilde{H}}, link_{\tilde{H}}) \\
\phi' &= \phi.
\end{aligned}$$

All the conditions are obviously satisfied.

□

**Proof of Lemma 7.6.14**

By Corol. 7.5.23 we have a match

$$a = ctxt([\![\phi]\!]) \circ ([\![\phi]\!] \mathbin{\blacksquare} [\![\tilde{P}]\!] \otimes \mathsf{id}_{X_I}) \circ prmt([\![\phi]\!])$$

for some set of names $X_I$, so we just have to show

$$a' = ctxt([\![\phi]\!]) \circ ([\![\phi']\!] \mathbin{\blacksquare} [\![\delta(\tilde{P})]\!] \otimes \mathsf{id}_{X_I}) \circ \overline{[\![inst_Q(finst(\delta))]\!]}(prmt([\![\phi]\!]))$$

and $a \twoheadrightarrow a'$ then follows from cf. Def. 7.4.14.

Let

$$\begin{aligned}
D : \langle |Q|, X_I \rangle &= prmt([\![\phi]\!]) \\
C : \langle |R|, Y \uplus X_I \rangle \to J &= ctxt([\![\phi]\!]) \\
\tilde{P}' : Q' \to \langle R, Y \rangle &= \delta(\tilde{P}) \\
Q &= \{q_0, \dots, q_{k-1}\} && \text{where } \forall i \in [0; k-2] : q_i < q_{i+1} \\
Q' &= \{q'_0, \dots, q'_{k'-1}\} && \text{where } \forall i \in [0; k'-2] : q'_i < q'_{i+1} \\
R &= \{r_0, \dots, r_{m-1}\} && \text{where } \forall i \in [0; m-2] : r_i < r_{i+1}.
\end{aligned}$$

By the definitions of parameter (cf. Defs. 7.5.8 and 7.5.20) and patterns (cf. Def. 7.6.1) we have the following equalities:

$$
\begin{aligned}
D : \langle |Q|, X_I \rangle &= (a \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}, \emptyset, \mathit{ctrl}_a \restriction_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}}, \\
&\qquad [\phi(q_0) \mapsto 0, \ldots, \phi(q_{k-1}) \mapsto k-1] \\
&\qquad\quad \uplus\, \mathit{prnt}_a \restriction_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \setminus \mathsf{rng}(\phi^{\mathsf{s}})}, \\
&\qquad \mathit{link}'_D) && \text{Defs. 7.5.8 and 7.5.20} \\
&= d_0 \otimes \cdots \otimes d_{|Q|-1} && \text{Def. 7.2.34 and } \mathit{link}'_D \text{ a bijection} \\
V_{d_i} &= a \downharpoonleft^{\phi^{\mathsf{s}}(q_i)} && \text{Defs. 7.6.1 and 7.2.9}
\end{aligned}
$$

where

$$\mathit{link}'_D : P_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}} \to X_I \text{ a bijection.}$$

By the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8) and the constructions from Defs. 7.5.8, 7.5.20, and 7.6.1 we have the following equalities:

$$
\begin{aligned}
V_a &= V_C \uplus \phi^{\mathsf{v}}(V_{\tilde{P}}) \uplus V_D \\
E_a &= E_C \uplus \phi^{\mathsf{e}}(E_{\tilde{P}}) \\
\mathit{ctrl}_a &= \mathit{ctrl}_a \restriction_{V_C} \uplus \mathit{ctrl}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} \uplus \mathit{ctrl}_a \restriction_{V_D} \\
\mathit{prnt}_a(w) &= \begin{cases}
\mathit{prnt}_D(w) & \text{if } w \in V_D \text{ and } \mathit{prnt}_D(w) \in V_D \\
\mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D \text{ and } \mathit{prnt}_D(w) = i \in |Q| \\
& \quad \text{and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
\mathit{prnt}_C(j) & \text{if } w \in V_D \text{ and } \mathit{prnt}_D(w) = i \in |Q| \\
& \quad \text{and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) = j \in |R| \\
\mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \text{ and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
\mathit{prnt}_C(i) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \text{ and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) = i \in |R| \\
\mathit{prnt}_C(w) & \text{if } w \in V_C
\end{cases} \\
\mathit{link}_a(p) &= \begin{cases}
\mathit{link}_C(x) & \text{if } p \in P_D \text{ and } \mathit{link}_D(p) = x \in X_I \\
\mathit{link}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } \mathit{link}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) \in \phi^{\mathsf{e}}(E_{\tilde{P}}) \\
\mathit{link}_C(y) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } \mathit{link}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) = y \in Y \\
\mathit{link}_C(p) & \text{if } p \in P_C
\end{cases}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathit{ctrl}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} &= \mathit{ctrl}_{\tilde{P}} \circ (\phi^{\mathsf{v}})^{-1} \\
\mathit{link}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} &= \phi^{\mathsf{l}} \circ \mathit{link}_{\tilde{P}} \circ (\phi^{\mathsf{p}})^{-1}, \\
\mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} &= (\phi^{\mathsf{v}} \uplus \{r_0 \mapsto 0, \ldots, r_{m-1} \mapsto m-1\}) \\
&\qquad \circ\, \mathit{prnt}_{\tilde{P}} \\
&\qquad \circ\, ((\phi^{\mathsf{v}})^{-1} \uplus \{0 \mapsto q_0, \ldots, k-1 \mapsto q_{k-1}\}).
\end{aligned}
$$

Similarly, unfolding the definitions of composition and tensor product (cf. Def. 7.2.6 and Def. 7.2.8) and the constructions from Defs. 7.5.8, 7.5.20, and 7.6.1, we see that we have to show

the following equalities in order for $a'$ to be on the prescribed form:

$$\begin{aligned}
V_{a'} &= V_C \uplus \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \uplus V_{D'} \\
E_{a'} &= E_C \uplus \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\
ctrl_{a'} &= ctrl_a \restriction_{V_C} \uplus ctrl_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]} \uplus ctrl_a \restriction_{V_{D'}} \\
prnt_{a'}(w) &= \begin{cases}
prnt_{D'}(w) & \text{if } w \in V_{D'} \\
 & \quad \text{and } prnt_{D'}(w) \in V_{D'} \\
prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \\
 & \quad \text{and } prnt_{D'}(w) = i \in |Q'| \\
 & \quad \text{and } prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(i) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(j) & \text{if } w \in V_{D'} \\
 & \quad \text{and } prnt_{D'}(w) = i \in |Q'| \\
 & \quad \text{and } prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(i) = j \in |R| \\
prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(w) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(w) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(i) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C
\end{cases} \\
link_{a'}(p) &= \begin{cases}
link_C(x) & \text{if } p \in P_{D'} \text{ and } link_{D'}(p) = x \in X_I \\
link_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(p) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(p) \in \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\
link_C(y) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]}(p) = y \in Y \\
link_C(p) & \text{if } p \in P_C
\end{cases}
\end{aligned}$$

where

$$\begin{aligned}
D' : \langle |Q'|, X_I \rangle &= \overline{[\![inst_Q(finst(\delta))]\!]}(prmt([\![\phi]\!])), \\
ctrl_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]} &= ctrl_{\tilde{P}'} \circ (\phi'^{\mathsf{v}})^{-1}, \\
link_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]} &= \phi'^{\mathsf{l}} \circ link_{\tilde{P}'} \circ (\phi'^{\mathsf{p}})^{-1}, \\
prnt_{[\![\phi']\!]\bullet[\![\tilde{P}']\!]} &= (\phi'^{\mathsf{v}} \uplus \{r_0 \mapsto 0, \ldots, r_{m-1} \mapsto m-1\}) \\
&\quad \circ prnt_{\tilde{P}'} \\
&\quad \circ ((\phi'^{\mathsf{v}})^{-1} \uplus \{0 \mapsto q'_0, \ldots, k-1 \mapsto q'_{k-1}\}),
\end{aligned}$$

We show that this is the case for each edit:

**$\oplus_{v:K_{\vec{y}}@p}$:** We have $v \notin V_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R$, $v' \notin V_a$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}} + v, E_{\tilde{P}}, ctrl_{\tilde{P}}[v \mapsto K], prnt_{\tilde{P}}[v \mapsto p], \\
&\qquad link_{\tilde{P}}[(v,0) \mapsto \vec{y}_0, \ldots, (v, n-1) \mapsto \vec{y}_{n-1}]) : Q \to \langle R, Y \rangle \\
a' &= (V_a + v', E_a, ctrl_a[v' \mapsto K], prnt_a[v' \mapsto \phi(p)], \\
&\qquad link_a[(v',0) \mapsto \phi(\vec{y}_0), \ldots, (v', n-1) \mapsto \phi(\vec{y}_{n-1})]) \\
\phi' &= \phi[v \mapsto v'] \\
inst_Q(finst(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\
D' : \langle |Q'|, X_I \rangle &= D : \langle |Q|, X_I \rangle.
\end{aligned}$$

The interesting cases are the parent and link maps:

$$
\begin{aligned}
prnt_{a'}(w) &= prnt_a[v' \mapsto \phi(p)](w) \\
&= \begin{cases}
prnt_D(w) & \text{if } w \in V_D \text{ and } prnt_D(w) \in V_D \\
prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}(i) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
prnt_C(j) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}(i) = j \in |R| \\
prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \text{ and } prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}(w) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
prnt_C(i) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \text{ and } prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C \\
\phi(p) & \text{if } w = v'
\end{cases} \\
&= \begin{cases}
prnt_{D'}(w) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) \in V_{D'} \\
prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(i) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) + v' \\
prnt_C(j) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(i) = j \in |R| \\
prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) + v' \\
& \quad \text{and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(w) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) + v' \\
prnt_C(i) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) + v' \\
& \quad \text{and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C
\end{cases} \\
&= \begin{cases}
prnt_{D'}(w) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) \in V_{D'} \\
prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(i) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(j) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(i) = j \in |R| \\
prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(w) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(w) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(i) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C
\end{cases}
\end{aligned}
$$

since $v' \notin V_a \uplus m_a = \mathsf{cod}(\phi^{\mathsf{f}}) \supseteq \mathsf{rng}(prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]})$ and $prnt_{[\![\phi']\!] \blacksquare [\![\tilde{P}']\!]} = prnt_{[\![\phi]\!] \blacksquare [\![\tilde{P}]\!]}[v' \mapsto \phi(p)]$.

$$\begin{aligned}
link_{a'}(p) &= link_a[(v',0) \mapsto \phi(\vec{y}_0), \dots, (v', n-1) \mapsto \phi(\vec{y}_{n-1})](p) \\
&= \begin{cases} link_C(x) & \text{if } p \in P_D \text{ and } link_D(p) = x \in X_I \\ link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) \in \phi^{\mathsf{e}}(E_{\tilde{P}}) \\ link_C(y) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \\ [\dots, (v', i) \mapsto \phi(\vec{y}_i), \dots](p) & \text{if } p \in P_{v'} \end{cases} \\
&= \begin{cases} link_C(x) & \text{if } p \in P_{D'} \text{ and } link_{D'}(p) = x \in X_I \\ link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \uplus P_{v'} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) \in \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\ link_C(y) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \uplus P_{v'} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases} \\
&= \begin{cases} link_C(x) & \text{if } p \in P_{D'} \text{ and } link_{D'}(p) = x \in X_I \\ link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) \in \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\ link_C(y) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases}
\end{aligned}$$

since $link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]} = link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}[\dots, (v', i) \mapsto \phi(\vec{y}_i), \dots]$.

$\oplus_e$ : We have $e \notin E_{\tilde{P}}$, $e' \notin E_a$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}} + e, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}) : Q \to \langle R, Y \rangle \\
a' &= (V_a, E_a + e', ctrl_a, prnt_a, link_a) \\
\phi' &= \phi[e \mapsto e'] \\
inst_Q(finst(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\
D' : \langle |Q'|, X_I \rangle &= D : \langle |Q|, X_I \rangle.
\end{aligned}$$

The interesting case is the link map:

$$\begin{aligned}
link_{a'}(p) &= link_a(p) \\
&= \begin{cases} link_C(x) & \text{if } p \in P_D \text{ and } link_D(p) = x \in X_I \\ link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) \in \phi^{\mathsf{e}}(E_{\tilde{P}}) \\ link_C(y) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases} \\
&= \begin{cases} link_C(x) & \text{if } p \in P_D \text{ and } link_D(p) = x \in X_I \\ link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) \in \phi^{\mathsf{e}}(E_{\tilde{P}}) + e' \\ link_C(y) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases} \\
&= \begin{cases} link_C(x) & \text{if } p \in P_{D'} \text{ and } link_{D'}(p) = x \in X_I \\ link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) \in \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\ link_C(y) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases}
\end{aligned}$$

since $e' \notin E_a \uplus Y \supseteq \mathsf{rng}(link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]})$.

$\ominus_v$ : We have $v \in V_{\tilde{P}}$, $prnt_{\tilde{P}}^{-1}(v) = \emptyset$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}} - v, E_{\tilde{P}}, ctrl_{\tilde{P}} - v, prnt_{\tilde{P}} - v, link_{\tilde{P}} - P_v) : Q \to \langle R, Y\rangle \\
a' &= (V_a - \phi(v), E_a, ctrl_a - \phi(v), prnt_a - \phi(v), link_a - P_{\phi(v)}) \\
\phi' &= \phi - v \\
inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\
D' : \langle |Q'|, X_I\rangle &= D : \langle |Q|, X_I\rangle.
\end{aligned}$$

The interesting case is the link map:

$$\begin{aligned}
link_{a'}(p) &= (link_a - P_{\phi(v)})(p) \\
&= \begin{cases} link_C(x) & \text{if } p \in P_D \setminus P_{\phi(v)} \text{ and } link_D(p) = x \in X_I \\ link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \setminus P_{\phi(v)} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) \in \phi^{\mathsf{e}}(E_{\tilde{P}}) \\ link_C(y) & \text{if } p \in P_{\phi^{\mathsf{v}}(V_{\tilde{P}})} \setminus P_{\phi(v)} \text{ and } link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \setminus P_{\phi(v)} \end{cases} \\
&= \begin{cases} link_C(x) & \text{if } p \in P_{D'} \text{ and } link_{D'}(p) = x \in X_I \\ link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}} - v)} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) \in \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\ link_C(y) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}} - v)} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases} \\
&= \begin{cases} link_C(x) & \text{if } p \in P_{D'} \text{ and } link_{D'}(p) = x \in X_I \\ link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) \in \phi'^{\mathsf{e}}(E_{\tilde{P}'}) \\ link_C(y) & \text{if } p \in P_{\phi'^{\mathsf{v}}(V_{\tilde{P}'})} \text{ and } link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(p) = y \in Y \\ link_C(p) & \text{if } p \in P_C \end{cases}
\end{aligned}$$

since $\phi(v) \in \phi^{\mathsf{v}}(V_{\tilde{P}})$, $\phi^{\mathsf{v}}(V_{\tilde{P}}) \# V_D$, $\phi^{\mathsf{v}}(V_{\tilde{P}}) \# V_C$, and $link_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]} = link_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} - P_{\phi(v)}$.

$\ominus_e$ : We have $e \in E_{\tilde{P}}$, $link_{\tilde{P}}^{-1}(e) = \emptyset$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}} - e, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}) : Q \to \langle R, Y\rangle \\
a' &= (V_a, E_a - \phi(e), ctrl_a, prnt_a, link_a) \\
\phi' &= \phi - e \\
inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\
D' : \langle |Q'|, X_I\rangle &= D : \langle |Q|, X_I\rangle.
\end{aligned}$$

All the equalities obviously hold.

$\ominus_q$**:** We have $q \in Q$, and (noting that $a$ has no sites)

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}} - q, link_{\tilde{P}}) : (Q - q) \to \langle R, Y\rangle \\
a' &= (V_a \setminus a\downharpoonright^{\phi(q)}, E_a, ctrl_a - a\downharpoonright^{\phi(q)}, prnt_a - a\downharpoonright^{\phi(q)}, link_a - P_{a\downharpoonright^{\phi(q)}})) \\
\phi' &= \phi - q \\
i_q &= i \qquad \text{if } q_i = q \\
inst_Q(\mathit{finst}(\delta)) &= (\mathsf{Id}_{\mathcal{U}}[q \mapsto \emptyset]\restriction_Q)^{-1} \\
&= \mathsf{Id}_{Q-q} \\
[\![inst_Q(\mathit{finst}(\delta))]\!] &= [\![\mathsf{Id}_{Q-q}]\!] \\
&= \mathsf{Id}_{i_q} \uplus [i_q \mapsto i_q + 1, \ldots, |Q| - 2 \mapsto |Q| - 1] \\
D' : \langle |Q'|, X_I\rangle &= \overline{[\![\mathsf{Id}_{Q-q}]\!]}(D)
\end{aligned}$$

By the definition of instantiation (Def. 7.2.35), we obtain:

$$
\begin{aligned}
D' : \langle |Q'|, X_I \rangle &= \overline{[\![\mathsf{Id}_{Q-q}]\!]}(D) \\
&= d_{[\![\mathsf{Id}_{Q-q}]\!](0)} \parallel \cdots \parallel d_{[\![\mathsf{Id}_{Q-q}]\!](|Q|-2)} \\
&= d_0 \otimes \cdots \otimes d_{i_q-1} \otimes d_{i_q+1} \otimes \cdots \otimes d_{|Q|-1} \\
&= (a \downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \setminus a \downharpoonleft^{\phi^{\mathsf{s}}(q)}, \emptyset, \mathit{ctrl}_a \upharpoonright_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \setminus a \downharpoonleft^{\phi^{\mathsf{s}}(q)}}, \\
&\qquad ([\![\mathsf{Id}_{Q-q}]\!]^{-1} \uplus \mathsf{Id}_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})} \setminus a \downharpoonleft^{\phi^{\mathsf{s}}(q)}}) \\
&\qquad\quad \circ (\mathit{prnt}_D - a \downharpoonleft^{\phi^{\mathsf{s}}(q)}), \\
&\qquad \mathit{link}'_D - P_{a\downharpoonleft^{\phi^{\mathsf{s}}(q)}})
\end{aligned}
$$

The interesting case is the parent map:

$$
\begin{aligned}
\mathit{prnt}_{a'}(w) &= (\mathit{prnt}_a - a \downharpoonleft^{\phi(q)})(w) \\
&= \begin{cases}
\mathit{prnt}_D(w) & \text{if } w \in V_D \setminus a \downharpoonleft^{\phi(q)} \text{ and } \mathit{prnt}_D(w) \in V_D \\
\mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D \setminus a \downharpoonleft^{\phi(q)} \text{ and } \mathit{prnt}_D(w) = i \in |Q| \\
& \quad \text{and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
\mathit{prnt}_C(j) & \text{if } w \in V_D \setminus a \downharpoonleft^{\phi(q)} \text{ and } \mathit{prnt}_D(w) = i \in |Q| \\
& \quad \text{and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) = j \in |R| \\
\mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \setminus a \downharpoonleft^{\phi(q)} \text{ and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
\mathit{prnt}_C(i) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \setminus a \downharpoonleft^{\phi(q)} \text{ and } \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) = i \in |R| \\
\mathit{prnt}_C(w) & \text{if } w \in V_C \setminus a \downharpoonleft^{\phi(q)}
\end{cases} \\
&= \begin{cases}
\mathit{prnt}_D(w) & \text{if } w \in V_{D'} \text{ and } \mathit{prnt}_{D'}(w) \in V_{D'} \\
\mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \text{ and } \mathit{prnt}_{D'}(w) = i \in |Q'| \\
& \quad \text{and } \mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
\mathit{prnt}_C(j) & \text{if } w \in V_{D'} \text{ and } \mathit{prnt}_{D'}(w) = i \in |Q'| \\
& \quad \text{and } \mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) = j \in |R| \\
\mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } \mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
\mathit{prnt}_C(i) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } \mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) = i \in |R| \\
\mathit{prnt}_C(w) & \text{if } w \in V_C
\end{cases}
\end{aligned}
$$

since

$$
\begin{aligned}
\phi^{\mathsf{v}}(V_{\tilde{P}}) \,\#\, a \downharpoonleft^{\phi(q)} & & \text{Lemma 7.5.11} \\
V_C \,\#\, a \downharpoonleft^{\phi(q)} & & \text{Def. 7.5.8} \\
\mathit{prnt}_{D'} &= ([\![\mathsf{Id}_{Q-q}]\!]^{-1} \uplus \mathsf{Id}_{V_{D'}}) \\
&\quad \circ (\mathit{prnt}_D - a \downharpoonleft^{\phi^{\mathsf{s}}(q)}) \\
\mathsf{rng}(\mathit{prnt}_{D'}) &= V_{D'} \uplus |Q'| \\
\mathit{prnt}_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]} &= \mathit{prnt}_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} \circ ([\![\mathsf{Id}_{Q-q}]\!] \uplus \mathsf{Id}_{\mathsf{rng}(\phi'^{\mathsf{v}})}).
\end{aligned}
$$

$\oslash_{v@p}$: We have $v \in V_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[v \mapsto p], link_{\tilde{P}}) : Q \to \langle R, Y\rangle \\
a' &= (V_a, E_a, ctrl_a, prnt_a[\phi(v) \mapsto \phi(p)], link_a) \\
\phi' &= \phi \\
inst_Q(finst(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\
D' : \langle |Q'|, X_I\rangle &= D : \langle |Q|, X_I\rangle .
\end{aligned}$$

The interesting case is the parent map:

$$\begin{aligned}
prnt_{a'}(w) &= prnt_a[\phi(v) \mapsto \phi(p)](w) \\
&= \begin{cases}
prnt_D(w) & \text{if } w \in V_D - \phi(v) \text{ and } prnt_D(w) \in V_D \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D - \phi(v) \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) \in \phi^{\vee}(V_{\tilde{P}}) \\
prnt_C(j) & \text{if } w \in V_D - \phi(v) \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) = j \in |R| \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi^{\vee}(V_{\tilde{P}}) - \phi(v) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) \in \phi^{\vee}(V_{\tilde{P}}) \\
prnt_C(i) & \text{if } w \in \phi^{\vee}(V_{\tilde{P}}) - \phi(v) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C - \phi(v) \\
\phi(p) & \text{if } w = \phi(v)
\end{cases} \\
&= \begin{cases}
prnt_{D'}(w) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) \in V_{D'} \\
prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) \in \phi'^{\vee}(V_{\tilde{P}'}) \\
prnt_C(j) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) = j \in |R| \\
prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) & \text{if } w \in \phi'^{\vee}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) \in \phi'^{\vee}(V_{\tilde{P}'}) \\
prnt_C(i) & \text{if } w \in \phi'^{\vee}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C
\end{cases}
\end{aligned}$$

since $\phi(v) \in \phi^{\vee}(V_{\tilde{P}})$, $\phi^{\vee}(V_{\tilde{P}}) \# V_D$, $\phi^{\vee}(V_{\tilde{P}}) \# V_C$, and $prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]} = prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}[\phi(v) \mapsto \phi(p)]$.

$\oslash_{q@p}$: We have $q \in Q$, $p \in V_{\tilde{P}} \uplus R$, and

$$\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[q \mapsto p], link_{\tilde{P}}) : Q \to \langle R, Y\rangle \\
a' &= (V_a, E_a, ctrl_a, prnt_a[\phi(q) \mapsto \phi(p)], link_a) \\
\phi' &= \phi \\
inst_Q(finst(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\
D' : \langle |Q'|, X_I\rangle &= D : \langle |Q|, X_I\rangle .
\end{aligned}$$

The interesting case is the parent map:

$$
\begin{aligned}
prnt_{a'}(w) &= prnt_a[\phi(q) \mapsto \phi(p)](w) \\
&= \begin{cases}
prnt_D(w) & \text{if } w \in V_D \setminus \phi(q) \text{ and } prnt_D(w) \in V_D \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D \setminus \phi(q) \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
prnt_C(j) & \text{if } w \in V_D \setminus \phi(q) \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) = j \in |R| \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \setminus \phi(q) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
prnt_C(i) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \setminus \phi(q) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C \setminus \phi(q) \\
\phi(p) & \text{if } w \in \phi(q)
\end{cases} \\
&= \begin{cases}
prnt_{D'}(w) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) \in V_{D'} \\
prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(j) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) = j \in |R| \\
prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(i) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C
\end{cases}
\end{aligned}
$$

since $prnt_D(\phi(q)) \in |Q|$, $\phi(q) \subseteq V_D$, cf. Def. 7.5.8, $V_D \# \phi^{\mathsf{v}}(V_{\tilde{P}})$, $V_D \# V_C$, and $prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]} = prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}[i \mapsto p']$ if $q = q_i$ and

$$
p' = \begin{cases} j & \text{if } p = r_j \\ \phi^{\mathsf{v}}(p) & \text{if } p \in V_{\tilde{P}} \end{cases} .
$$

$\otimes_{q \to r @ p}$**:** We have $q \in Q$, $r \notin Q$, $p \in V_{\tilde{P}} \uplus R$, and (noting that $a$ has no sites)

$$
\begin{aligned}
\tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}[r \mapsto p], link_{\tilde{P}}) : (Q + r) \to \langle R, Y \rangle \\
a' &= (V_a \uplus V_r, E_a, ctrl_a \uplus ctrl_r, prnt_a \uplus prnt_r, link_a \uplus link_r) \\
\phi' &= \phi[r \mapsto f_v^{-1}(\phi(q))] \\
Q' &= \{q_0, \ldots, q_{i_r - 1}, r, q_{i_r}, \ldots, q_{k-1}\} \qquad \text{where } q_{i_r - 1} < r < q_{i_r} \\
i_q &= i \qquad \text{if } q_i = q \\
inst_Q(finst(\delta)) &= (\mathsf{Id}_{\mathcal{U} - r}[q \mapsto \{q, r\}] \restriction_Q)^{-1} \\
&= \mathsf{Id}_Q[r \mapsto q] \\
[\![inst_Q(finst(\delta))]\!] &= [\![\mathsf{Id}_Q[r \mapsto q]]\!] \\
&= \mathsf{Id}_{i_r} \uplus [i_r \mapsto i_q] \uplus [i_r + 1 \mapsto i_r, \ldots, |Q| \mapsto |Q| - 1] \\
D' : \langle |Q'|, X_I \rangle &= \overline{[\![\mathsf{Id}_Q[r \mapsto q]]\!]}(D)
\end{aligned}
$$

where

$$
\begin{aligned}
V_q &= a\downharpoonleft^{\phi(q)} \\
|V_r| &= |V_q| \\
V_r &\mathbin{\#} V_a \\
f_v &: V_r \rightarrowtail V_q \\
ctrl_r &= ctrl_a \circ f_v \\
prnt_r &= \{f_v^{-1}(\phi(q)) \mapsto \phi(p)\} \uplus f_v^{-1} \circ prnt_a \circ (f_v - f_v^{-1}(\phi(q))) \\
link_r(v,i) &= link_a(f_v(v), i) \quad (v \in V_r)
\end{aligned}
$$

By the definition of instantiation (Def. 7.2.35), we obtain:

$$
\begin{aligned}
D' : \langle |Q'|, X_I \rangle &= \overline{[\![\mathsf{Id}_Q[r \mapsto q]]\!]}(D) \\
&= d_{[\![\mathsf{Id}_Q[r\mapsto q]]\!](0)} \parallel \cdots \parallel d_{[\![\mathsf{Id}_Q[r\mapsto q]]\!](|Q|)} \\
&= d_0 \otimes \cdots \parallel d_{i_r-1} \parallel d_r \parallel d_{i_r} \parallel \cdots \otimes d_{|Q|-1} \\
&= (V_D \uplus V_r, \emptyset, ctrl_a \restriction_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}} \uplus ctrl_r, \\
&\qquad ((\mathsf{Id}_{i_r} \uplus [i_r + 1 \mapsto i_r, \ldots, |Q| \mapsto |Q| - 1])^{-1} \uplus \mathsf{Id}_{a\downharpoonleft^{\mathsf{rng}(\phi^{\mathsf{s}})}}) \circ prnt_D \\
&\qquad \uplus ([i_r \mapsto i_q]^{-1} \uplus f_v^{-1}) \circ prnt_D \circ f_v, \\
&\qquad link_D \uplus link_r) \\
d_r &= f_v^{-1} \mathbin{\blacksquare} d_{i_q}.
\end{aligned}
$$

The interesting case is the parent map:

$$
\begin{aligned}
prnt_{a'}(w) &= (prnt_a \uplus prnt_r)(w) \\
&= \begin{cases}
prnt_D(w) & \text{if } w \in V_D \text{ and } prnt_D(w) \in V_D \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
prnt_C(j) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) = j \in |R| \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \\
prnt_C(i) & \text{if } w \in \phi^{\mathsf{v}}(V_{\tilde{P}}) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C \\
prnt_r(w) & \text{if } w \in V_r
\end{cases} \\
&= \begin{cases}
prnt_D(w) & \text{if } w \in V_D \text{ and } prnt_D(w) \in V_D \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(j) & \text{if } w \in V_D \text{ and } prnt_D(w) = i \in |Q| \\
& \quad \text{and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(i) = j \in |R| \\
prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(i) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C \\
prnt_{D'}(w) & \text{if } w \in V_r \text{ and } prnt_{D'}(w) \in V_r \\
\phi(p) & \text{if } w \in V_r \text{ and } prnt_{D'}(w) = i_r
\end{cases} \\
&= \begin{cases}
prnt_{D'}(w) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) \in V_{D'} \\
prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q'| \\
& \quad \text{and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(j) & \text{if } w \in V_{D'} \text{ and } prnt_{D'}(w) = i \in |Q'| \\
& \quad \text{and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(i) = j \in |R| \\
prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \\
prnt_C(i) & \text{if } w \in \phi'^{\mathsf{v}}(V_{\tilde{P}'}) \text{ and } prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]}(w) = i \in |R| \\
prnt_C(w) & \text{if } w \in V_C
\end{cases}
\end{aligned}
$$

since

$$\begin{aligned} prnt_{D'} &= ((\mathsf{Id}_{i_r} \uplus [i_r + 1 \mapsto i_r, \dots, |Q| \mapsto |Q| - 1])^{-1} \uplus \mathsf{Id}_{a\downarrow^{\mathsf{rng}(\phi^{\mathsf{s}})}}) \circ prnt_D \\ &\quad \uplus ([i_r \mapsto i_q]^{-1} \uplus f_v^{-1}) \circ prnt_D \circ f_v \end{aligned}$$

$$\begin{aligned} prnt_r &= [f_v^{-1}(\phi(q)) \mapsto \phi(p)] \\ &\quad \uplus f_v^{-1} \circ prnt_a \circ (f_v - f_v^{-1}(\phi(q))) \\ &= [i_q \mapsto \phi(p)] \circ prnt_D \circ [f_v^{-1}(\phi(q)) \mapsto \phi(q)] \\ &\quad \uplus f_v^{-1} \circ prnt_a \circ (f_v - f_v^{-1}(\phi(q))) \\ &= [\phi(p) \mapsto i_q]^{-1} \circ prnt_D \circ [f_v^{-1}(\phi(q)) \mapsto \phi(q)] \\ &\quad \uplus f_v^{-1} \circ prnt_a \circ (f_v - f_v^{-1}(\phi(q))) \\ &= [\phi(p) \mapsto i_q]^{-1} \circ prnt_D \circ f_v \restriction_{f_v^{-1}(\phi(q))} \\ &\quad \uplus f_v^{-1} \circ prnt_D \circ (f_v - f_v^{-1}(\phi(q))) \\ &= ([\phi(p) \mapsto i_q]^{-1} \uplus f_v^{-1}) \circ prnt_D \circ f_v \\ &= ([i_r \mapsto \phi(p)] \uplus \mathsf{Id}_{V_r}) \circ ([i_r \mapsto i_q]^{-1} \uplus f_v^{-1}) \circ prnt_D \circ f_v \\ &= ([i_r \mapsto \phi(p)] \uplus \mathsf{Id}_{V_r}) \circ prnt_{D'} \restriction_{V_r} \end{aligned}$$

$$prnt_{[\![\phi']\!] \bullet [\![\tilde{P}']\!]} = prnt_{[\![\phi]\!] \bullet [\![\tilde{P}]\!]} \circ (\mathsf{Id}_{i_r} \uplus [i_r \mapsto i_q] \uplus [i_r + 1 \mapsto i_r, \dots, |Q| \mapsto |Q| - 1] \uplus \mathsf{Id}_{\mathsf{rng}(\phi')}).$$

$\odot_{(v,i) \mapsto l}$**:** We have $v \in V_{\tilde{P}}$, $i \in ar(ctrl_{\tilde{P}}(v))$, $l \in E_{\tilde{P}} \uplus Y$, and

$$\begin{aligned} \tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}}, link_{\tilde{P}}[(v, i) \mapsto l]) : Q \to \langle R, Y \rangle \\ a' &= (V_a, E_a, ctrl_a, prnt_a, link_a[(\phi(v), i) \mapsto \phi(l)]) \\ \phi' &= \phi \\ inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\ D' : \langle |Q'|, X_I \rangle &= D : \langle |Q|, X_I \rangle. \end{aligned}$$

All the equalities obviously hold.

$\circledcirc_{v:K}$**:** We have $v \in V_{\tilde{P}}$, $ar(K) = ar(ctrl_{\tilde{P}}(v))$, and

$$\begin{aligned} \tilde{P}' &= (V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}[v \mapsto K], prnt_{\tilde{P}}, link_{\tilde{P}}) : Q \to \langle R, Y \rangle \\ a' &= (V_a, E_a, ctrl_a[\phi(v) \mapsto K], prnt_a, link_a) \\ \phi' &= \phi \\ inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_{\mathcal{U}} \\ D' : \langle |Q'|, X_I \rangle &= D : \langle |Q|, X_I \rangle. \end{aligned}$$

All the equalities obviously hold.

□

**Proof of Lemma 7.6.15**

By Def. 7.6.4 we have a match of R in $a$ and thus by Corol. 7.5.23 we have an embedding $\phi : \tilde{P} \hookrightarrow a$ such that

$$\begin{aligned} a &= ctxt([\![\phi]\!]) \circ ([\![\phi]\!] \bullet [\![\tilde{P}]\!] \otimes \mathsf{id}_{X_I}) \circ prmt([\![\phi]\!]) \\ a' &= ctxt([\![\phi]\!]) \circ (\rho' \bullet [\![\delta(\tilde{P})]\!] \otimes \mathsf{id}_{X_I}) \circ \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])). \end{aligned}$$

What remains is to show $(a'', \phi') = \delta(a, \phi)$ and $a' \simeq a''$ for each edit. The difficulty lies in the instantiation and the interesting cases are those that delete or copy a parameter; the others are very similar to each other, and we only show the first one.

$\oplus_{v:K_{\vec{y}}@p}$**:** We have $v \notin V_{\tilde{P}}$, $p \in V_{\tilde{P}} \uplus R$, $\{\vec{y}\} \subseteq E_{\tilde{P}} \uplus Y$, $ar(K) = n$, and

$$\begin{aligned} a'' &= (V_a + v', E_a, ctrl_a[v' \mapsto K], prnt_a[v' \mapsto \phi(p)], \\ &\qquad link_a[(v',0) \mapsto \phi(\vec{y}_0), \ldots, (v', n-1) \mapsto \phi(\vec{y}_{n-1})]) \\ \phi' &= \phi[v \mapsto v'] \\ inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_Q \end{aligned}$$

for some $v' \notin V_a$.

By Corol. 7.5.23 we have

$$a'' = ctxt([\![\phi']\!]) \circ ([\![\phi']\!] \bullet [\![\delta(\tilde{P})]\!] \otimes \mathsf{id}_{X_I}) \circ prmt([\![\phi']\!]).$$

From Def. 7.5.20 and $\phi' = \phi[v \mapsto v']$ it is clear that $ctxt([\![\phi']\!]) = ctxt([\![\phi]\!])$ and $prmt([\![\phi']\!]) = prmt([\![\phi]\!])$, so it is sufficient to show

$$\begin{aligned} prmt([\![\phi]\!]) &\simeq \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])) \\ [\![\phi']\!] \bullet [\![\delta(\tilde{P})]\!] &\simeq \rho' \bullet [\![\delta(\tilde{P})]\!]. \end{aligned}$$

where the latter is immediate from the definition of support equivalence. By definition of instantiation (cf. Def. 7.2.35) and $inst_Q(\mathit{finst}(\delta)) = \mathsf{Id}_Q$ we get

$$\begin{aligned} \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])) &= \overline{[\![\mathsf{Id}_Q]\!]}(prmt([\![\phi]\!])) \\ &= \overline{\mathsf{Id}_{|Q|}}(prmt([\![\phi]\!])) \\ &\simeq prmt([\![\phi]\!]) \end{aligned}$$

as required.

$\oplus_e$ : Similar to the first case.

$\ominus_v$ : Similar to the first case.

$\ominus_e$ : Similar to the first case.

$\ominus_q$**:** We have $q \in Q$ and (noting that $a$ has no sites)

$$\begin{aligned} a'' &= (V_a \setminus a\downharpoonright^{\phi(q)}, E_a, ctrl_a - a\downharpoonright^{\phi(q)}, \\ &\qquad prnt_a - a\downharpoonright^{\phi(q)}, link_a - P_{a\downharpoonright^{\phi(q)}}) \\ \phi' &= \phi - q \\ inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_{Q-q}. \end{aligned}$$

For simplicity, assume $\forall q' \in Q - q : q > q'$.

By Corol. 7.5.23 we have

$$a'' = ctxt([\![\phi']\!]) \circ ([\![\phi']\!] \bullet [\![\delta(\tilde{P})]\!] \otimes \mathsf{id}_{X_I}) \circ prmt([\![\phi']\!]).$$

From Def. 7.5.20 and $\phi' = \phi - q$ it is easy to see that

$$\begin{aligned} ctxt([\![\phi']\!]) &= ctxt([\![\phi]\!]) \\ prmt([\![\phi]\!]) &= d_0 \otimes \cdots \otimes d_{|Q|-1} \\ prmt([\![\phi']\!]) &= d_0 \otimes \cdots \otimes d_{|Q|-2} \end{aligned}$$

so it is sufficient to show

$$\begin{aligned} prmt(\phi') &\simeq \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])) \\ [\![\phi']\!] \bullet [\![\delta(\tilde{P})]\!] &\simeq \rho' \bullet [\![\delta(\tilde{P})]\!]. \end{aligned}$$

where the latter is immediate from the definition of support equivalence. By definition of instantiation (cf. Def. 7.2.35) and $inst_Q(\mathit{finst}(\delta)) = \mathsf{Id}_{Q-q}$ we get

$$\begin{aligned} \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])) &= \overline{[\![\mathsf{Id}_{Q-q}]\!]}(prmt([\![\phi]\!])) \\ &= \overline{\mathsf{Id}_{|Q|-2}}(prmt([\![\phi]\!])) \\ &\simeq d_0 \otimes \cdots \otimes d_{|Q|-2} \\ &\simeq prmt([\![\phi']\!]) \end{aligned}$$

as required.

$\oslash_{v@p}$: Similar to the first case.

$\oslash_{q@p}$: Similar to the first case.

$\otimes_{q\to r@p}$: We have $q \in Q$, $r \notin Q$, $p \in V_{\tilde{P}} \uplus R$, and (noting that $a$ has no sites)

$$\begin{aligned} a'' &= (V_a \uplus V_r, E_a, ctrl_a \uplus ctrl_r, prnt_a \uplus prnt_r, link_a \uplus link_r) \\ \phi' &= \phi[r \mapsto f_v^{-1}(\phi(q))] \\ inst_Q(\mathit{finst}(\delta)) &= \mathsf{Id}_Q[r \mapsto q] \end{aligned}$$

where

$$\begin{aligned} V_q &= a \downharpoonleft^{\phi(q)} \\ |V_r| &= |V_q| \\ V_r &\,\#\, V_a \\ f_v &: V_r \rightarrowtail V_q \\ ctrl_r &= ctrl_a \circ f_v \\ prnt_r &= \{f_v^{-1}(\phi(q)) \mapsto \phi(p)\} \uplus f_v^{-1} \circ prnt_a \circ (f_v - f_v^{-1}(\phi(q))) \\ link_r(v,i) &= link_a(f_v(v), i). \end{aligned}$$

For simplicity, assume $r > q$ and $\forall q' \in Q - q : q > q'$.
By Corol. 7.5.23 we have

$$a'' = ctxt([\![\phi']\!]) \circ ([\![\phi']\!] \bullet [\![\delta(\tilde{P})]\!] \otimes \mathsf{id}_{X_I}) \circ prmt([\![\phi']\!]).$$

From Def. 7.5.20 and $\phi' = \phi[r \mapsto f_v^{-1}(\phi(q))]$ it is easy to see that

$$\begin{aligned} ctxt([\![\phi']\!]) &= ctxt([\![\phi]\!]) \\ prmt([\![\phi]\!]) &= d_0 \otimes \cdots \otimes d_{|Q|-1} = (V_D, \emptyset, ctrl_D, prnt_D, link_D) \\ R &= (V_r, \emptyset, ctrl_r, prnt_R, link_R) \\ prnt_R &= \{f_v^{-1}(\phi(q)) \mapsto 0\} \uplus f_v^{-1} \circ prnt_a \circ (f_v - f_v^{-1}(\phi(q))) \\ link_R(v,i) &= link_D(f_v(v), i) \\ prmt([\![\phi']\!]) &= prmt([\![\phi]\!]) \parallel R \\ d_{|Q|-1} &= (V_q, \emptyset, ctrl_D \restriction_{V_q}, prnt_D \restriction_{V_q}, link_D \restriction_{P_{V_q}}) \\ &\simeq R \end{aligned}$$

so it is sufficient to show

$$\begin{aligned} prmt(\phi') &\simeq \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])) \\ [\![\phi']\!] \mathbin{\blacksquare} [\![\delta(\tilde{P})]\!] &\simeq \rho' \mathbin{\blacksquare} [\![\delta(\tilde{P})]\!]. \end{aligned}$$

where the latter is immediate from the definition of support equivalence. By definition of instantiation (cf. Def. 7.2.35), $inst_Q(\mathit{finst}(\delta)) = \mathsf{Id}_Q[r \mapsto q]$, and $d_{|Q|-1} \simeq R$ we get

$$\begin{aligned} \overline{[\![inst_Q(\mathit{finst}(\delta))]\!]}(prmt([\![\phi]\!])) &= \overline{[\![\mathsf{Id}_Q[r \mapsto q]]\!]}(prmt([\![\phi]\!])) \\ &= \overline{\mathsf{Id}_{|Q|-1}[|Q| \mapsto |Q|-1]}(prmt([\![\phi]\!])) \\ &\simeq prmt([\![\phi]\!]) \parallel R \\ &\simeq prmt([\![\phi']\!]) \end{aligned}$$

as required.

**$\odot_{(v,i)\mapsto l}$:** Similar to the first case.

**$\circledcirc_{v:K}$:** Similar to the first case.

□

**Proof of Prop. 7.6.28**

For each section of the script, we show (1) that it is compatible with the pattern at that point, (2) what the resulting pattern is, and (3) what the resulting forward instance map is. Finally, we show that the named instance map we derive from the forward instance map is $\eta$.

**$\otimes_{\eta(q'_i)\to f(q'_i)@r}$:** Compatible since $\eta(q'_i) \in \mathsf{cod}(\eta) = Q$, $f(q'_i) \in Q'' \,\#\, Q$, and the redex must have a root if it has sites, since the parent map is acyclic.

The resulting pattern is

$$(V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, prnt_{\tilde{P}} \uplus [f(Q') \mapsto r], link_{\tilde{P}}) : Q \uplus Q'' \to \langle R, Y \rangle.$$

The resulting forward instance map is

$$\begin{aligned} &\mathsf{Id}_{\mathcal{U}-f(q'_{n'})}[\eta(q'_{n'}) \mapsto \{\eta(q'_{n'}), f(q'_{n'})\}] \\ &\circ (\cdots \circ (\mathsf{Id}_{\mathcal{U}-f(q'_1)}[\eta(q'_1) \mapsto \{\eta(q'_1), f(q'_1)\}] \restriction^{\mathcal{U}-f(q'_2)}) \cdots \restriction^{\mathcal{U}-f(q'_{n'})}) \\ = {} &\mathsf{Id}_{\mathcal{U}-f(Q')}[\eta(q'_1) \mapsto \{\eta(q'_1), f(q'_1)\}, \ldots, \eta(q'_{n'}) \mapsto \{\eta(q'_{n'}), f(q'_{n'})\}]. \end{aligned}$$

**$\ominus_{q_i}$:** Compatible since $q_i \in Q$.

The resulting pattern is

$$(V_{\tilde{P}}, E_{\tilde{P}}, ctrl_{\tilde{P}}, (prnt_{\tilde{P}} \uplus [f(Q') \mapsto r]) - Q, link_{\tilde{P}}) : Q'' \to \langle R, Y \rangle.$$

The resulting forward instance map is

$$\begin{aligned} &\mathsf{Id}_{\mathcal{U}}[q_n \mapsto \emptyset] \\ &\circ (\cdots \circ (\mathsf{Id}_{\mathcal{U}}[q_1 \mapsto \emptyset] \\ &\circ (\mathsf{Id}_{\mathcal{U}-f(Q')}[\eta(q'_1) \mapsto \{\eta(q'_1), f(q'_1)\}, \ldots, \eta(q'_{n'}) \mapsto \{\eta(q'_{n'}), f(q'_{n'})\}] \restriction^{\mathcal{U}}) \\ &\restriction^{\mathcal{U}}) \cdots \restriction^{\mathcal{U}}) \\ = {} &\mathsf{Id}_{\mathcal{U}-f(Q')}[q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\ &\quad [\eta(q'_1) \mapsto \{f(q'_1)\}, \ldots, \eta(q'_{n'}) \mapsto \{f(q'_{n'})\}]. \end{aligned}$$

$\ominus_{v_i}$**:** Compatible since $v_i \| n V_{\tilde{P}}$ and the nodes has no remaining children since sites are only at root $r$ and for any node v$v_j \in prnt_{\tilde{P}}^{-1}(v_i)$ we have $j < i$ and thus $v_j$ has already been deleted.

The resulting pattern is

$$\begin{aligned} &(\emptyset, E_{\tilde{P}}, ctrl_{\tilde{P}} - V_{\tilde{P}}, ((prnt_{\tilde{P}} \uplus [f(Q') \mapsto r]) - Q) - V_{\tilde{P}}, link_{\tilde{P}} - P_{\tilde{P}}) : Q'' \to \langle R, Y \rangle \\ =\ &(\emptyset, E_{\tilde{P}}, \emptyset, [f(Q') \mapsto r], \emptyset) : Q'' \to \langle R, Y \rangle. \end{aligned}$$

The forward instance map is unchanged.

$\ominus_{e_i}$**:** Compatible since $e_i \| n E_{\tilde{P}}$ and there are no points left.

The resulting pattern is

$$(\emptyset, \emptyset, \emptyset, [f(Q') \mapsto r], \emptyset) : Q'' \to \langle R, Y \rangle.$$

The forward instance map is unchanged.

$\oplus_{e'_i}$**:** Compatible since there are no edges left.

The resulting pattern is

$$(\emptyset, E_{\tilde{P}'}, \emptyset, [f(Q') \mapsto r], \emptyset) : Q'' \to \langle R, Y \rangle.$$

The forward instance map is unchanged.

$\oplus_{v'_i : ctrl_{\tilde{P}'}(v'_i)_{[\ldots, link_{\tilde{P}'}(v'_i, i), \ldots]} @\, prnt_{\tilde{P}'}(v'_i)}$**:** Compatible since there are no nodes left, the links and roots are all present and if the parent is a node $v'_j = prnt_{\tilde{P}'}(v'_i)$ then we have $j < i$ and thus it has been added before its children.

The resulting pattern is

$$(V_{\tilde{P}'}, E_{\tilde{P}'}, ctrl_{\tilde{P}'}, (prnt_{\tilde{P}'} - Q') \uplus [f(Q') \mapsto r], link_{\tilde{P}'}) : Q'' \to \langle R, Y \rangle.$$

The forward instance map is unchanged.

$\otimes_{f(q'_i) \to q'_i @\, prnt_{\tilde{P}'}(q'_i)}$**:** Compatible since $f(q'_i) \in Q''$, $q'_i \in Q' \# Q''$, and $prnt_{\tilde{P}'}(q'_i) \in V_{\tilde{P}'} \uplus R$.

The resulting pattern is

$$(V_{\tilde{P}'}, E_{\tilde{P}'}, ctrl_{\tilde{P}'}, prnt_{\tilde{P}'} \uplus [f(Q') \mapsto r], link_{\tilde{P}'}) : Q' \uplus Q'' \to \langle R, Y \rangle.$$

The resulting forward instance map is

$$\begin{aligned} &\mathsf{Id}_{\mathcal{U} - q'_{n'}}[f(q'_{n'}) \mapsto \{f(q'_{n'}), q'_{n'}\}] \\ &\circ (\cdots \circ (\mathsf{Id}_{\mathcal{U} - q'_1}[f(q'_1) \mapsto \{f(q'_1), q'_1\}] \\ &\circ (\mathsf{Id}_{\mathcal{U} - f(Q')}[q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\ &\qquad [\eta(q'_1) \mapsto \{f(q'_1)\}, \ldots, \eta(q'_{n'}) \mapsto \{f(q'_{n'})\} \downharpoonleft^{\mathcal{U} - q'_1}) \\ &\downharpoonleft^{\mathcal{U} - q'_2}) \cdots \downharpoonleft^{\mathcal{U} - q'_{n'}}) \\ =\ &\mathsf{Id}_{(\mathcal{U} - f(Q')) - Q'}[q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\ &\qquad [\eta(q'_1) \mapsto \{f(q'_1), q'_1\}, \ldots, \eta(q'_{n'}) \mapsto \{f(q'_{n'}), q'_{n'}\}]. \end{aligned}$$

$\ominus_{f(q'_i)}$**:** Compatible since $f(q'_i) \in Q''$.

The resulting pattern is

$$(V_{\tilde{P}'}, E_{\tilde{P}'}, \mathit{ctrl}_{\tilde{P}'}, \mathit{prnt}_{\tilde{P}'}, \mathit{link}_{\tilde{P}'}) : Q' \to \langle R, Y \rangle = \tilde{P}'.$$

The resulting forward instance map is

$$\begin{aligned}
&\mathit{finst}(\mathit{es}(\mathsf{R})) \\
=\ &\mathsf{Id}_{\mathcal{U}}[f(q'_{n'}) \mapsto \emptyset] \\
&\circ (\cdots \circ (\mathsf{Id}_{\mathcal{U}}[f(q'_1) \mapsto \emptyset] \\
&\circ (\mathsf{Id}_{(\mathcal{U}-f(Q'))-Q'}[q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\
&\qquad [\eta(q'_1) \mapsto \{f(q'_1), q'_1\}, \ldots, \eta(q'_{n'}) \mapsto \{f(q'_{n'}), q'_{n'}\}] \downharpoonleft^{\mathcal{U}}) \\
&\downharpoonleft^{\mathcal{U}}) \cdots \downharpoonleft^{\mathcal{U}}) \\
=\ &\mathsf{Id}_{(\mathcal{U}-f(Q'))-Q'}[q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\
&\qquad [\eta(q'_1) \mapsto \{q'_1\}, \ldots, \eta(q'_{n'}) \mapsto \{q'_{n'}\}]
\end{aligned}$$

The derived named instance map for the entire script is thus:

$$\begin{aligned}
\mathit{inst}_Q(\mathit{finst}(\mathit{es}(\mathsf{R}))) &= (\mathsf{Id}_{(\mathcal{U}-f(Q'))-Q'}[q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\
&\qquad [\eta(q'_1) \mapsto \{q'_1\}, \ldots, \eta(q'_{n'}) \mapsto \{q'_{n'}\}] \upharpoonright_Q)^{-1} \\
&= ([q_1 \mapsto \emptyset, \ldots, q_n \mapsto \emptyset] \\
&\qquad [\eta(q'_1) \mapsto \{q'_1\}, \ldots, \eta(q'_{n'}) \mapsto \{q'_{n'}\}])^{-1} \\
&= [q'_1 \mapsto \eta(q'_1), \ldots, q'_{n'} \mapsto \eta(q'_{n'})] \\
&= \eta.
\end{aligned}$$

□

# Part V

# A Bigraphical Language for Cell Biology

# Chapter 8

# Formal Cellular Machinery

**Troels C. Damgaard, Espen Højsgaard, and Jean Krivine**

**Abstract**

Various calculi have been proposed to model different levels of abstraction of cell signaling and molecular interactions. In this paper we propose a framework inspired by some of these calculi that structures interactions and agents from the most basic elements of the cell (protein interaction sites) to higher order ones (compartments and molecular species).

**Preface** This chapter consists of the paper

T. C. Damgaard, E. Højsgaard, and J. Krivine. *Formal Cellular Machinery.* Proceedings of SASB 2011, the Second International Workshop on Static Analysis and Systems Biology. September 2011. Keynote talk. (to appear)

## 8.1 Introduction

It has been about 10 years now since part of the theoretical computer science community got interested in applying formal methods to systems biology. Since then it seems that the quest for a calculus having proteins, compartments or channels as first class citizens has not reached an end. Among the large variety of languages that have been proposed to tackle various aspects of systems biology (see Refs. [1–5, 10, 12, 14, 15, 17–22] for a non exhaustive list), several ideas seem of particular importance to us: (i) the cellular medium can be described as a graph where nodes represent molecules and edges represent physical contacts between these molecules [1, 10, 12, 14], (ii) languages with a natural notion of location of reaction can be used to represent cellular compartments [2, 15, 19–21], (iii) interactions between compartments and proteins or vesicle transformations can be described using local patches of membranes, without committing to any particular global curvature [4, 11], and (iv) although laws governing interactions of molecular components are numerous, they can be engendered by a small set of generators [3].

The present work proposes to integrate points (i) to (iv) in a single formalism. More specifically we define a language for proteins and cells in an incremental way, making explicit the trade-off between expressiveness and complexity. We decompose the construction of the language in four steps:

– $\mathcal{C}_0$**:** an "untyped" calculus aimed at modeling protein-protein interactions. The dynamics of these interactions is presented as a small set of *generator* rules, which modelers can refine and compose but not change.

– $\mathcal{C}_1$**:** an intermediate version of the term language that allows modelers to type reactions introduced at the previous stage.

– $\mathcal{C}_2$**:** the main expressiveness increment of our language. It introduces compartments and the notion of *projectivity* of membrane reactions, i.e., the possibility to mention patches of membrane, without having to deal with their global curvature. We propose a matching algorithm, that is proven both sound and complete. At this stage, generators allow modelers to create and destroy compartments in a projective fashion.

– $\mathcal{C}_3$**:** the final step of the construction deals with the diffusion problem. In particular we incorporate means to talk about connected components of reactants, which is a key feature for a new set of generators modeling diffusion of molecular species and intra-molecular complex formation. To the best of our knowledge $\mathcal{C}_3$ is the first calculus of its kind that allows one to model molecular agents both at a micro level (where interactions are purely local) and a macro level (where interactions involve connected components of agents).

The language we build is inspired by and closely related to the $\kappa$-calculus of Danos and Laneve [9, 10] and Milner's bigraphical reactive systems [16], however these connections will be left informal throughout the paper. The reader might refer to Appendix 8.A and to Ref. [6] for some preliminary work on the subject.

## 8.2 $\mathcal{C}_0$: forming molecules

Proteins are long polymers built over an alphabet of 20 amino acids. Each protein's interaction capabilities are mediated by its 3D folding in space which in turn depends on its amino acid composition. Protein interactions are either *structural* when they form non-covalent bonds to other molecular agents (DNA, RNA, other proteins) or *enzymatic* when they can catalyze the chemical modification of the substrate to which they are bound. In the first case one usually talks about complex formation, in the latter one talks about post-transcriptional modification. It has been observed that the amino acid sequence of most proteins appearing in living organisms can be regrouped into *domains* which are strings of amino acids that have a specific fold in space that is rather context free. Biologists tend to associate "functions" to domains, for instance zinc finger domains are often linked to the specific DNA binding capability of their host protein.

The first step of our construction, termed $\mathcal{C}_0$, is aimed at representing *domains* as a collection of interaction sites, *proteins* as a collection of domains and *interactions* as protein assembly and complex formation.

### 8.2.1 Terms

Consider an infinite set of *site names* $\mathcal{S} = \{x, y, z, \dots\}$ and a disjoint infinite set of *backbone names* $\mathcal{B} = \{a, b, c, \dots\}$. Let $\mathsf{D}$ be a terminal symbol, distinct from all others, that we use to denote domains. Terms $T$ of $\mathcal{C}_0$ are built on the following grammar:

$$\begin{array}{lcll} D, D' & ::= & \mathsf{D}^a(x_1, \dots, x_k) & \text{for } a \in \mathcal{B},\ x_i \in \mathcal{S} \\ T, S & ::= & D \mid 0 \mid (T, S) \mid T \backslash v & \text{for } v \in \mathcal{S} \cup \mathcal{B} \end{array}$$

Intuitively a $k$-ary domain $\mathsf{D}^a(x_1, \dots, x_k)$ is the placeholder of $k$ (interaction) sites and one backbone. Each site $i$ is equipped with a name $x_i \in \mathcal{S}$ and each domain with a backbone name $a \in \mathcal{B}$. Backbone name sharing denotes domains that belong to the same protein, site name sharing denotes complex formation. We inductively define free occurrences of names as:

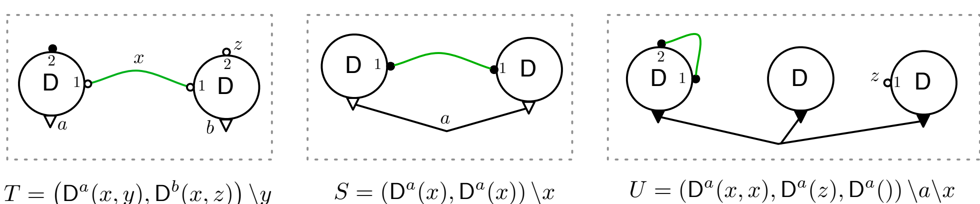
$$\begin{array}{rcl} fn\,(\mathsf{D}^a(x_1, \dots, x_k)) & = & \{a, x_1, \dots, x_k\} \\ fn(0) & = & \emptyset \\ fn(T, S) & = & fn(T) \cup fn(S) \\ fn(T \backslash v) & = & fn(T) - \{v\} \end{array}$$

$$\begin{array}{lcll}
(S,T) & \equiv & (T,S) & \\
((T,S),T') & \equiv & (T,(S,T')) & \\
(T,0) & \equiv & T & \\
T\backslash u & \equiv & T & u \notin fn(T) \\
(T\backslash u)\backslash v & \equiv & (T\backslash v)\backslash u & \\
T\backslash u & \equiv & (T\{v/u\})\backslash v & v \notin fn(T) \\
(T\backslash u, S) & \equiv & (T,S)\backslash u & u \notin fn(S)
\end{array}$$

Figure 8.1: Structural congruence for $\mathcal{C}_0$.

Symmetrically, one can define the bound occurrences of names, which we shall denote by $bn(T)$. Terms are equipped with a natural notion of structural congruence defined in Fig. 8.1. The structural congruence relation rules include a natural $\alpha$-equivalence on bound names. In the following we assume that names that are not under the same binder are kept distinct.

### 8.2.2 Graphical notation




$T = (\mathsf{D}^a(x,y), \mathsf{D}^b(x,z))\backslash y$ $\quad S = (\mathsf{D}^a(x), \mathsf{D}^a(x))\backslash x$ $\quad U = (\mathsf{D}^a(x,x), \mathsf{D}^a(z), \mathsf{D}^a())\backslash a\backslash x$

Intuitively, the term to port graph correspondence is the following: domains are nodes, sites and backbones are ports and name sharing denotes *(hyper) edges*. Bound names denote *closed ports* and we use the term *closed edges* to denote a bound name that is shared. Similarly, free names denote *open ports* and form *open edges* when they are shared. Open ports or edges can be merged or closed in the context (see later). With these conventions, one may view any term (up to structural congruence) as the isomorphism class of a port graph (with hyper-edges), in the style of bigraphs [16], where nodes (domains) are equipped with connection ports (sites and backbones). As an example we give above the port graph representation of terms $T, S$ and $U$. The reader familiar with bigraphs will notice that we drift slightly away from Milner's notation: site ports are represented by small circles that are filled when they are closed. Backbone ports are represented as small triangles that are also filled when they are closed. We use curved lines for site edges and straight lines for backbone edges. We label open edges or open ports with the corresponding free name (closed edges and ports are not labelled). Note that we will omit site numbers whenever they are not necessary.

Connections between sites correspond to physical contacts between protein parts. This connection being exclusive we want to restrict to terms where restrictions bind at most two occurrences of site names. In the following of this paper we will assume that for any term $T$, free site names occur exactly once in $T$ and bound site names have at most two occurrences. Note that we do not impose such restrictions on backbone name sharing.

### 8.2.3 Pattern matching and dynamics

A *match* for $T$ in $S$ is defined as a context $\mathbb{C}[\bullet]$ with exactly one hole such that $\mathbb{C}[T] \equiv S$. Such contexts are defined inductively as:

$$\mathbb{C}[\bullet] \quad ::= \quad \bullet \mid \mathbb{C}[\bullet]\backslash u \mid \mathbb{C}[\bullet], T \quad u \in \mathcal{B} \cup \mathcal{S}$$

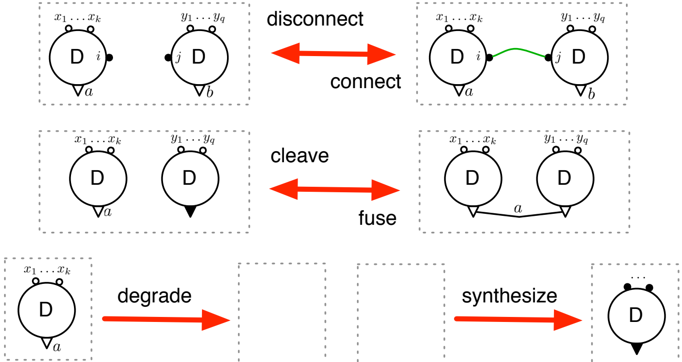



Figure 8.2: The set $\mathcal{G}_0$ of generators for $\mathcal{C}_0$.

*A rule* is a pair of terms $\langle T, S\rangle$ such that $fn(S) \subseteq fn(T)$. Given a set $\mathcal{R}$ of such pairs, one may rewrite terms by letting these rules be applied in a context free manner, i.e.,:

$$\frac{r = \langle T, S\rangle \in \mathcal{R} \quad T' \equiv \mathbb{C}[T\sigma] \quad S' \equiv \mathbb{C}[S\sigma]}{T' \rightarrow_r S'}$$

for some name substitution $\sigma$.

### 8.2.4 Generators

It is clear that not all rules make sense from a biological point of view: the fact that backbone names denote the core of a protein and that site names denote connection between protein domains is purely conventional and this convention could be easily broken. A way to proceed is to define some sorting discipline that allows one to screen off undesired terms from admissible ones [2], invalid rule applications being discarded "on the fly". Instead of doing this, we adopt a strategy of pre-conceiving what "laws" a modeler is able to invoke when defining her own rule set. This is achieved by defining a set $\mathcal{G}_0$ of basic rule *generators* that a modeler can only refine to her needs, cf. Fig. 8.2. These generators allow one to perform standard atomic actions of graph rewriting. It is noteworthy that these generators, including degrade, are side effect free. We shall carry this set of generators throughout the rest of this paper, incorporating new generators as the language grows.

Say a rule $r = \langle T, S\rangle$ is *generated* if and only if it can be obtained by:

- **refinement**: there exists $\langle T', S'\rangle \in \mathcal{G}_0$ such that $T \equiv \mathbb{C}[T'\sigma]$ and $S \equiv \mathbb{C}[S'\sigma]$ for some context $\mathbb{C}[\bullet]$ and substitution $\sigma$.
- **composition**: one can generate two rules $\langle T, T'\rangle$ and $\langle T', S\rangle$.

### 8.2.5 Discussion

We have introduced so far a simple calculus that rewrites proteins structured as connected domains. Proteins can be connected to each other (as in complex formation), new domains can be fused to proteins (as in protein synthesis) or severed (as trans-membrane proteins can be cleaved to emit signals into the inter cellular medium). This calculus is fairly abstract in the sense that two proteins may only differ in the number of domains they have and in the number of sites these domains possess. It is clear that we lack means of *naming* molecular components such as domain names (SH2, Tyrosine, PWWP etc.) or protein names (SOS, EGF, IGF, p53, etc.). Before

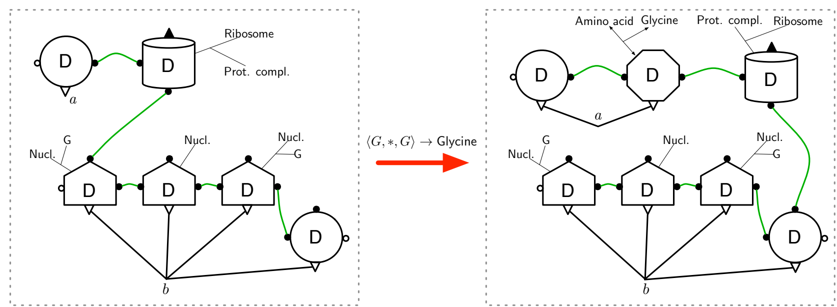



Figure 8.3: Graphical illustration of the role of *info* nodes and *meta* names, with the rule for the RNA translation of a Glycine amino acid. Node shape is purely illustrative. A ribosome is bound to a guanine being part of an RNA strand (backbone $b$) and has started to assemble a new protein (backbone $a$). The next nucleotide on the right is of unspecified type followed by a $G$ nucleotide, this triplet $\langle G, *, G\rangle$ codes for the Glycine that is produced on the right.

performing a bigger increment in expressiveness, when we introduce compartments in Section 8.4, we would like to briefly introduce a way to deal with names as a particular type of context in which unamed proteins can be embedded. The intent is to provide a way to define molecular reactions as refinements of the generators we have just presented, in keeping with the biological intuition that information about molecular objects is always partial and that more context could reveal more about the nature of a molecule. In particular, we have the ontology problem in mind that several names can denote the same protein or gene.

## 8.3 $\mathcal{C}_1$: naming molecules

### 8.3.1 Terms

Consider a new set of names $\mathcal{M}$ that is pairwise disjoint from $\mathcal{B}$ and $\mathcal{S}$. Terms of $\mathcal{C}_1$ are essentially those of $\mathcal{C}_0$ where domains have an extra *meta name* $m, m' \in \mathcal{M}$ that will point to new type of terms called *info* terms (denoted by $I, J, \ldots$). Let $\mathcal{I}$ be a set of terminal symbols (distinct from all previous ones) called *informations* (think of protein or domain names). The grammar of $\mathcal{C}_1$ is:

$$
\begin{array}{llll}
D, D' & ::= & \mathsf{D}^a_m(x_1, \ldots, x_k) & a \in \mathcal{B},\ m \in \mathcal{M},\ x_i \in \mathcal{S} & \text{(domains)} \\
I, J & ::= & \mathsf{Info}_m & \mathsf{Info} \in \mathcal{I},\ m \in \mathcal{M} & \text{(info)} \\
T, S & ::= & 0 \mid D \mid I \mid (T, S) \mid T \backslash v & \text{for } v \in \mathcal{S} \cup \mathcal{B} \cup \mathcal{M} & \text{(named terms)}
\end{array}
$$

Structural congruence coincides with the one defined earlier.

### 8.3.2 Graphical notation

This simple extension has a natural impact on the graphical notation, as shown in Fig. 8.3 with an example of amino acid synthesis. *Info* nodes are represented by their type (Nucl., G, Ribosome, Prot. compl., Amino acid, Glycine) without drawing borders around them. Meta names that are shared by nodes induce thin straight hyper edges. Open meta ports are not drawn, and closed meta edges are represented with filled arrowheads (as in the Amino acid and Glycine nodes on the right hand side).

There are only two specific generators for $\mathcal{C}_1$, for all $\mathsf{Info} \in \mathcal{I}$:

$$
\begin{array}{ll}
\text{(Concretize)} & \mathsf{D}^a_m(x_1, \ldots, x_k) \to \mathsf{D}^a_m(x_1, \ldots, x_k), \mathsf{Info}_m \\
\text{(Abstract)} & \mathsf{D}^a_m(x_1, \ldots, x_k), \mathsf{Info}_m \to \mathsf{D}^a_m(x_1, \ldots, x_k)
\end{array}
$$

and again, rules can be generated by refinement and composition of generators.

### 8.3.3 Discussion

With little symbol pushing burden we obtain a fairly expressive language which, at this stage, is already a reasonable candidate for representing most types of synthetic biology systems. It is noteworthy that the nature of an interaction can be expressed here as a form of type instantiation. One may think of $\mathcal{C}_0$ generators as polymorphic reaction types: $(\alpha, \beta)$ $\mathsf{connect}$ or $\alpha$ $\mathsf{synthesize}$. They can be instantiated as $(A{,}B)$ $\mathsf{connect}$ or *Amino acid* $\mathsf{synthesize}$.

This second step brings us closer to the $\kappa$-calculus of Danos and Laneve [10]. In fact, our calculus now encompasses $\kappa$ in a straightforward way (see Appendix 8.A).

## 8.4 $\mathcal{C}_2$: placing molecules

As we already stressed in the previous sections, we have for now abstracted away from space and geometry: molecules are assumed to be floating in a uniform medium that lets domains react freely with each other. One could for instance encode a discrete compartment as *info* nodes attached to each domain and make sure they are compatible when two domains encounter. Yet, not only would this induce an explosion in the number of rules to write, but also entail a lot of book keeping rules in order to make sure that protein domains remain co-localized. We propose here to exploit our informal yet underlying relationship with bigraphs in order to add a simple notion of compartmentalization to our language.

### 8.4.1 Terms

Let $\mathcal{V}$ be an infinite set of *parameter names* $\{X, Y, Z, \dots\}$ assumed to be pairwise disjoint from $\mathcal{S}$, $\mathcal{B}$ and $\mathcal{M}$. Let $\mathsf{C}$ be a terminal symbol, distinct from previous ones. Terms $P, Q, \dots$ of $\mathcal{C}_2$ are generated by the following extension of the grammar for $\mathcal{C}_1$:

$$\begin{array}{lcllr} T, S & ::= & \cdots \mid \mathsf{C}_m(T) \mid X & m \in \mathcal{M},\ X \in \mathcal{V} & \text{(local terms)} \\ P, Q & ::= & T \mid (T \parallel P) \mid P\backslash v & v \in \mathcal{M} \cup \mathcal{S} \cup \mathcal{B} & \text{(wide terms)} \end{array}$$

Terms of the form $\mathsf{C}_m(T)$ denote compartments. They are nodes with a meta name, like domains, but have neither sites nor backbone. In the way defined in the previous section, this meta name allows one to specify a type of compartment: for instance $\mathsf{nucleus}, \mathsf{membrane} \in \mathcal{I}$ (one may also think of $\mathsf{region} \in \mathcal{I}$ to denote compartments with no physical boundaries).

Note also the use parameters as in $\mathsf{C}_m(X)$, where $X$ denotes the unspecified content of compartment $\mathsf{C}_m$. We use $\mathcal{V}(P)$ to denote the set of parameter names in $P$. For simplicity we consider here "linear terms", i.e., terms that do not contain multiple copies of the same parameter variables. It entails that a rule may delete parameters but not duplicate them.

Terms of $\mathcal{C}_2$ are either *local*, in which case we use $T, S$ to denote them, or *wide* in which case we use $P, Q$. The term $P = (T \parallel S)$ is a pattern requiring $T$ and $S$ to be separated by *exactly one* compartment boundary in any context; note that this differs from the interpretation of wide composition in bigraphs, where they may be separated by any number of boundaries. Hence we will see that $P$ has a match in both $(\mathsf{C}_m(T), S)$ and $(T, \mathsf{C}_m(S))$. We want to absorb here the projective view of membrane reactions introduced by Danos and Pradalier [11] and also present in a later work by Cardelli [4]. The underlying idea is that membrane curvature is a global property that one may not want to consider when expressing cellular mechanisms. This trait will turn out to be very useful when defining a minimal set of generators for $\mathcal{C}_2$.

**Definition 8.4.1** (Local contexts)**.** A context $\mathbb{C}[\bullet]$ with exactly one hole is a *local context* if it is of the form:

$$\mathbb{C}[\bullet] ::= \bullet \mid \mathbb{C}[\bullet]\backslash u \mid \mathbb{C}[\bullet], T \quad u \in \mathcal{B} \cup \mathcal{S}$$

□

Note that the context $\mathsf{C}_m(\bullet)$ is not a local context. It is however a *derivable wide context* as we will see shortly.

Structural congruence for $\mathcal{C}_2$ extends the one of $\mathcal{C}_1$ with the following laws for wide composition of terms:

$$
\begin{array}{llll}
\mathsf{C}_m(T) & \equiv & \mathsf{C}_m(T') & \text{if } T \equiv T' \\
\mathsf{C}_m(T\backslash u) & \equiv & \mathsf{C}_m(T)\backslash u & \text{if } u \neq m \\
(P\backslash u)\backslash v & \equiv & (P\backslash v)\backslash u & \\
P\backslash u & \equiv & (P\{v/u\})\backslash v & v \notin fn(P) \\
T\backslash u \parallel P & \equiv & (T \parallel P)\backslash u & u \notin fn(P) \\
T \parallel P\backslash u & \equiv & (T \parallel P)\backslash u & u \notin fn(T)
\end{array}
$$

It is clear that any wide term is structurally congruent to a term of the form $(T_1 \parallel \cdots \parallel T_n)\backslash V$ (using the shorthand $P\backslash V$ for the restriction of the names of $V$). We sometimes write $P \parallel Q$ to denote the concatenation $P$ and $Q$ (in the style of list concatenation). Importantly a pattern of the form $T \parallel S \parallel T'$ specifies that $T$ and $T'$ are exactly two compartment layers away from each other, and that $S$ is one compartment layer away from both $T$ and $T'$, we will call this distance *projective* because it does not take the orientation of the compartment borders, that will separate the terms in the context, into account. We shall see that valid matches for a wide term $(T_1 \parallel \ldots \parallel T_n)\backslash V$ will correspond to those in which the distance between $T_i$ and $T_{i+k}$ is exactly $k$, for all $i \in \{1, \ldots, n-k\}$.

### 8.4.2 Pattern matching

For any wide term $P$, say that $P$ has *width* $w(P) = n$ if $P \equiv (T_1 \parallel \ldots \parallel T_n)\backslash V$ for some local terms $T_i$.

**Definition 8.4.2** (Projective distance)**.** Let $P$ be a wide term and $T_i, T_j$ two disjoint term occurrences in $P$. The *projective distance* of $T_i, T_j$ in $P$, written $\Delta_{T_i,T_j}(P)$ is inductively defined as:

$$
\begin{array}{llll}
\Delta_{T_i,T_j}(T_i, T_j) & = & 0 & \\
\Delta_{T_i,T_j}(T, S) & = & \Delta_{T_i,T_j}(T) & \text{if } T_i, T_j \notin S \\
\Delta_{T_i,T_j}(P\backslash u) & = & \Delta_{T_i,T_j}(P) & \\
\Delta_{T_i,T_j}(\mathsf{C}_m(T)) & = & \Delta_{T_i,T_j}(T) & \\
\Delta_{T_i,T_j}(\mathsf{C}_m(T), S) & = & \Delta_{T_i,T_j}(T, S) + 1 & \text{if } T_i \in T \text{ and } T_j \in S \\
\Delta_{T_i,T_j}(T \parallel P) & = & \Delta_{T_i,T_j}(P) & \text{if } T_i, T_j \in P \\
\Delta_{T_i,T_j}(T \parallel P) & = & \Delta_{T_i,T_j}(T) & \text{if } T_i, T_j \in T \\
\Delta_{T_i,T_j}(T \parallel S \parallel P) & = & \Delta_{T_i,T_j}(T \parallel P) + 1 & \text{if } T_i, T_j \notin S \\
\Delta_{T_i,T_j}(T \parallel S) & = & \Delta_{T_i,T_j}(T, S) + 1 & \text{if } T_i \in T, T_j \in S
\end{array}
$$

□

In other terms, the projective distance between $T_i$ and $T_j$ is equal to the number of wide compositions and compartment layers that separate $T_i$ from $T_j$.

Given a wide term $P = (T_1 \parallel \ldots \parallel T_n)\backslash V$, we need to define contexts $\mathbb{C}^n[\bullet, \ldots, \bullet]$ with exactly $n$ holes in which one may embed $P$ while preserving nesting distance. Let *generic contexts* (with an arbitrary number of holes) be inductively defined as:

$$T_\bullet, S_\bullet \quad ::= \quad \bullet \mid T \mid (T_\bullet, S_\bullet) \mid (T_\bullet)\backslash u \mid \mathsf{C}_m\,(T_\bullet) \qquad u \in \mathcal{B} \cup \mathcal{S} \cup \mathcal{M} \;\&\; m \in \mathcal{M}$$

For any such context $T_\bullet$ with exactly $k$ holes, we write $T_\bullet = \mathbb{C}^k[\bullet, \ldots, \bullet]$ or simply $T = \mathbb{C}^k$. Importantly, not all contexts of the form $\mathbb{C}^1$ is a local context since $\mathsf{C}_m(\bullet)$ is not local. Furthermore,

$$\text{(ax.)}\quad \frac{}{T \hookrightarrow_{\bullet} \mathbb{C}[\bullet]} \qquad \frac{P \hookrightarrow_{\pi} T_{\bullet} \quad v \notin fn(T_{\bullet}) \cup bn(T_{\bullet})}{P \backslash v \hookrightarrow_{\pi} T_{\bullet}}\quad \text{(rest)}$$

$$\frac{P \hookrightarrow_{\pi} T_{\bullet} \quad m \text{ fresh} \quad (\!| \cdot \pi \cdot |\!) \not\to \bot}{P \hookrightarrow_{(\!| \cdot \pi \cdot |\!)} \mathsf{C}_m(T_{\bullet})}\quad \text{(wrap)}$$

$$\frac{P \hookrightarrow_{\pi_0} T_{\bullet} \quad Q \hookrightarrow_{\pi_1} S_{\bullet} \quad \pi_0 \cdot \pi_1 \not\to \bot}{P \parallel Q \hookrightarrow_{\pi_0 \cdot \pi_1} \mathbb{C}[T_{\bullet}, S_{\bullet}]}\quad \text{(comp)}$$

Table 8.1: The extension relation. Contexts $\mathbb{C}[\bullet]$ are the local contexts of Definition 8.4.1.

not all contexts of $k$ holes will be valid placeholders for wide terms of width $k$. Rather than trying to enumerate valid contexts with $n$ holes we use a procedure that generates valid matches for terms of arbitrary width. We will then prove that this procedure is both sound and complete in the sense that it finds only correct matches for wide terms, and finds them all.

Let *projection constraints* $\pi$ be words on the alphabet $\Pi \stackrel{def}{=} \{(\!|, |\!), \bullet, \bot\}$. We use these constraints during the construction of a wide context $\mathbb{C}^n$, as an abstraction of the context that retains only the positions of compartments borders, symbols $(\!|$ and $|\!)$, and holes, symbol $\bullet$. In order to check that $\mathbb{C}^n$ is a valid context, it will suffice to make sure that the projection constraint is well-formed. For instance, the constraint $\pi = \bullet \cdot (\!| \cdot \bullet \cdot |\!) \cdot \bullet$ is an abstraction of an invalid context with exactly three holes, that would place the term $T \parallel S \parallel T'$ in an environment where $T$ and $T'$ would be at (projective) distance 0 instead of 2. Invalid constraints are detected during the construction of a wide context (cf. Table 8.1), using the reduction relation of Table 8.1.

**Definition 8.4.3** (Valid constraints)**.** Let $\pi \in \Pi^*$ be a projection constraint. Let $\cdot$ denote the concatenation of words over the alphabet $\Pi$. Say that $\pi$ is *valid* if $\pi \not\to \bot$ with $\to \subseteq \Pi^* \times \Pi^*$ the least reflexive, transitive, and compatible relation engendered by:

$$\begin{array}{llll} \bullet \cdot (\!| \cdot (\!| \to \bot & |\!) \cdot |\!) \cdot \bullet \to \bot & |\!) \cdot (\!| \to \bot & \\ \bullet \cdot \bullet \to \bot & \bullet \cdot (\!| \cdot \bullet \cdot |\!) \cdot \bullet \to \bot & \bot \cdot \pi \to \bot & \pi \cdot \bot \to \bot \end{array}$$

□

The inductive construction of the extension relation is given in Table 8.1. Let $\mu, \mu', \ldots$ denote (possibly empty) lists of parameter assignation of the form $[X_1 \leftarrow T_1]; \ldots; [X_n \leftarrow T_n]$ with $\mathcal{V}(T_i) = \emptyset$. We use $|\mu|$ to denote the set of parameter names in $\mu$, and $P\mu$ to denote $P$ in which parameters have been substituted according to $\mu$.

**Definition 8.4.4** (Matches)**.** A wide context $\mathbb{C}^n[\bullet, \ldots, \bullet]$ with exactly $n$ holes and a parameter assignation list $\mu$ form a *match* $\langle \mathbb{C}^n, \mu \rangle$ for a wide term $P = (T_1 \parallel \ldots \parallel T_n) \backslash V$ in $S$ if and only if:

$$P \hookrightarrow_{\pi} \mathbb{C}^n \quad \text{and} \quad |\mu| = \mathcal{V}(P) \quad \text{and} \quad ((\mathbb{C}^n[T_1, \ldots, T_n]\mu) \backslash V)\sigma \equiv S$$

for some name substitution $\sigma$. □

Furthermore, a pair $r = \langle P, Q \rangle$ with $w(P) = w(Q) = n$ and $\mathcal{V}(P) = V(Q)$ generates a transition $T \to_r S$ if the match $\langle \mathbb{C}^n[\bullet, \ldots, \bullet], \mu \rangle$ for $P$ in $T$ is a match for $Q$ in $S$.

We conclude this section with the expected soundness and completeness results for our extension relation with respect to projective distance.

**Theorem 8.4.5** (Soundness)**.** *Let $\langle \mathbb{C}^n[\bullet, \ldots, \bullet], \mu \rangle$ be a match for a wide term $P$ in a local term $T$. For all disjoint local term occurrences $S, S' \in P$, we have $\Delta_{S,S'}(P) = \Delta_{S,S'}(T)$.*

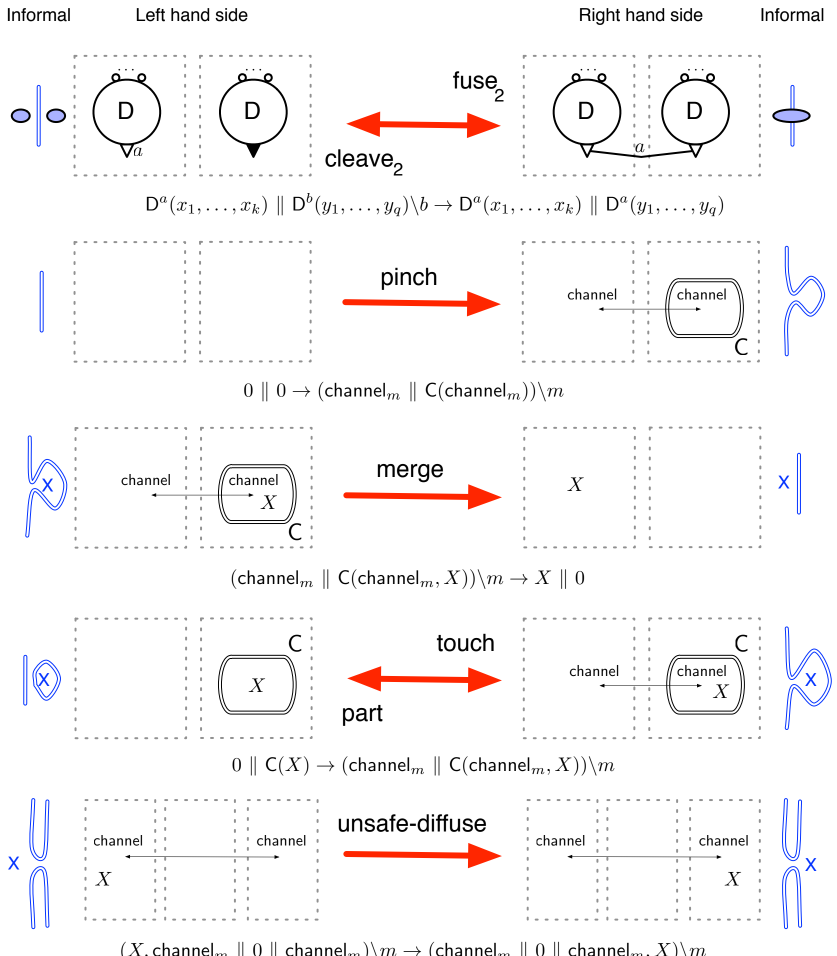



Figure 8.4: Generators for $\mathcal{C}_2$.

**Theorem 8.4.6** (Completeness). *Let $P = (T_1 \parallel \ldots \parallel T_n)\backslash V$ be a wide term and $\mathbb{C}^n[\bullet,\ldots,\bullet]$ be a generic context with exactly $n$ holes. Let also $T \equiv ((\mathbb{C}^n[T_1,\ldots,T_n]\mu)\backslash V)\sigma$ for some parameter assignation $\mu$ and name substitution $\sigma$.*

*If for all $i,j \leq n$ one has $\Delta_{T_i,T_j}(P) = \Delta_{T_i,T_j}(T)$, then $P \hookrightarrow_\pi \mathbb{C}^n$ is derivable, for some $\pi \in (\Pi \backslash \{\bot\})^*$.*

### 8.4.3 Generators

The generators are presented in Fig. 8.4, keeping with the graphical convention introduced earlier. We add here compartments, represented as nodes with double line boundaries, and variables. Wide terms are simply represented next to each other. Crucially, the possibility to express compartment patches independently of their general curvature allows us to maintain a minimal set of generators. Rules specifying curvature are then obtained as refinements of these generators. The wide versions of the fuse and cleave generators now allow for the representation of transmembrane proteins (aka receptors). Note that we do not generalize the connect and disconnect generators to keep with the fact that protein-protein interactions are local.

The other generators rely on the intuition, sketched in an earlier work on bigraphs [15], that

dynamic molecular compartments can be modeled using an intermediate step where two compartments are connected by a "neck". This neck, visible in generators pinch, merge, touch and unsafe-diffuse, is represented by two connected channel nodes, which are particular *info* nodes. In the unsafe-diffuse rule, they are used to indicate that molecules can translocate from one location to another, along the channel edge. This rule can be applied in order to populate a vesicle after pinch or touch, and until part or merge is applied.

At this stage our language is equipped with ways to model dynamic compartments and diffusion. Yet, consistency of the biological interpretation of $\mathcal{C}_2$ terms relies on a careful usage of the unsafe-diffuse rule. Indeed, nothing prevents modelers from using this generator to stretch a protein across several membranes by diffusing only a part of it, violating the desired invariant that only a backbone edge may cross a compartment (in the case of a receptor). In order to correct for this, we need to restrict diffusion to instances that will preserve biological soundness of terms. The final step in the design of our language is aimed at solving this question.

## 8.5 $\mathcal{C}_3$: moving molecules

### 8.5.1 Terms

Let $\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}$ be a family of $\mathsf{B}$ and $\mathsf{S}$ indexed terminal symbols (distinct from all others) with $\mathsf{B} \subseteq \mathcal{B}$ and $\mathsf{S} \subseteq \mathcal{S} \cup \mathcal{M}$. The grammar generating terms of $\mathcal{C}_3$ extends the previous one in the following way:

$$\begin{array}{lcll} T, S & ::= & \ldots & \text{(local terms)} \\ G, H & ::= & T \mid \mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T) \mid (G, H) & \text{(global terms)} \\ P, Q & ::= & G \mid (P \parallel Q) \mid \ldots & \text{(wide terms)} \end{array}$$

where $\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T)$ denotes the fact that term $T$ describes a partial species, i.e., is either a connected component or a pattern that should be placed in a context that will make it connected. The sets $\mathsf{B}$ and $\mathsf{S}$ denote respectively the free backbone names of the species and its free site and meta names. These names are kept separated for convenience because backbones will be allowed to cross membranes while *meta* and *site* names will not be shared by nodes that are not co-located in the same compartment. For instance, the expression $\mathsf{spec}^{\{a\}}_{\emptyset}((D^b_m(x), X)\backslash b, x, m)$ denotes a partial species that contains a domain $D^b_m(x)$ and that may only have a connection with other nodes outside the species boundaries by sharing the backbone name $a$.

The idea behind $\mathcal{C}_3$ is that although connectivity, i.e., transitive closure of name sharing, is a property one may not want to consider in general, it becomes relevant for some particular interactions including diffusion. We will come back to this in the section describing the new generators.

Structural congruence allows us to form spec nodes on demand. To do so, we extend previous structural laws with the following ones:

$$\frac{}{\mathsf{D}^a_m(x_1, \ldots, x_k) \equiv \mathsf{spec}^{\{a\}}_{\{m, x_1, \ldots, x_k\}}(\mathsf{D}^a_m(x_1, \ldots, x_k))}\,(\mathsf{init})$$

$$\frac{fn(A) \cap (\mathsf{B} \cup \mathsf{S}) \neq \emptyset \quad \mathsf{B}' = \mathsf{B} \cup (fn(A) \cap \mathcal{B}) \quad \mathsf{S}' = \mathsf{S} \cup (fn(A) \cap \mathcal{S})}{\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T), A \equiv \mathsf{spec}^{\mathsf{B}'}_{\mathsf{S}'}(T, A)}\,(\mathsf{grow})$$

$$\frac{u \in \mathcal{B} \cup \mathcal{S} \quad \mathsf{B}' \stackrel{def}{=} \mathsf{B} - \{u\} \quad \mathsf{S}' \stackrel{def}{=} \mathsf{S} - \{u\}}{\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T)\backslash u \equiv \mathsf{spec}^{\mathsf{B}'}_{\mathsf{S}'}(T\backslash u)} \qquad \frac{T \equiv T'}{\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T) \equiv \mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T')}$$

Where $A$ is either a domain node or an info node. Intuitively, the left-to-right orientation of the above first three equations allows one to capture more knowledge about connectivity, while the

other direction is forgetful. If one wishes to consider diffusion of vesicles, one needs the additional rule:

$$\frac{\mathit{fn}(T') \cap (\mathsf{B} \cup \mathsf{S}) \neq \emptyset \quad \mathsf{B}' = \mathsf{B} \cup (\mathit{fn}(T') \cap \mathcal{B}) \quad \mathsf{S}' = \mathsf{S} \cup (\mathit{fn}(T') \cap \mathcal{S})}{\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T), \mathsf{C}_m(T') \equiv \mathsf{spec}^{\mathsf{B}'}_{\mathsf{S}'}(T, \mathsf{C}_m(T'))}$$

that allows one to encompass compartments in the recognition of molecular species.

In order to ease the understanding of the generators presented in the next section, let us give a simple example of the usage of a species term in a pattern. Consider the term $P = (\mathsf{spec}^a_\emptyset(X) \parallel \mathsf{spec}^a_\emptyset(Y))\backslash a$ which denotes a transmembrane complex split in two parts $X$ and $Y$ on both sides of a membrane. We wish to find a match for $P$ in the term:

$$T = \big(\mathsf{D}^a_{m_1}(x), \mathsf{SH2}_{m_1}, \mathsf{C}_{m_2}(\mathsf{D}^a_{m_3}(y), \mathsf{D}^b_{m_4}(y))\big) \setminus \{a, b, x, y, m_i\}$$

To do so, we first need to turn $T$ into a form that makes the desired connectivity apparent:

$$\begin{array}{rcl} T & \equiv & \big(\mathsf{spec}^a_\emptyset(\mathsf{D}^a_{m_1}(x), \mathsf{SH2}_{m_1} \setminus \{x, m_1\}), \\ & & \mathsf{C}_{m_2}(\mathsf{spec}^a_\emptyset((\mathsf{D}^a_{m_3}(y), \mathsf{D}^b_{m_4}(y)) \setminus \{b, y, m_3, m_4\}) \backslash m_2\big) \backslash a \end{array}$$

Then, using the extension relation, we generate a context for $P$

$$P \hookrightarrow_{\bullet \cdot (\!| \cdot \bullet \cdot |\!)} (\bullet, \mathsf{C}_m(\bullet)) = \mathbb{C}^2[\bullet, \bullet]$$

which, together with a list of parameter assignations

$$\mu \stackrel{def}{=} [X \leftarrow (\mathsf{D}^a_{m_1}(x), \mathsf{SH2}_{m_1}) \setminus \{x, m_1\}]; [Y \leftarrow (\mathsf{D}^a_{m_3}(y), \mathsf{D}^b_{m_4}(y)) \setminus \{b, y, m_3, m_4\}]$$

defines a valid match for $P$ in $T$. One verifies that, indeed:

$$(\mathbb{C}[\mathsf{spec}^a_\emptyset(X), \mathsf{spec}^a_\emptyset(Y)]\mu)\, \{m_2/m\}\, \backslash a \equiv T$$

### 8.5.2 Generators

Generators are given in Fig. 8.5. They extend the generators of all previous stages, to the exception of the unsafe-diffuse rule that is replaced by its safe counterparts. We keep with the graphical conventions introduced earlier, and use cloud nodes to denote (partial or total) species.

As one may see in Fig. 8.5, we now have two generators for diffusion. The first one models classical diffusion: a total species may move from one compartment connected to another *via* a channel. The second generator models diffusion of transmembrane species: two partial and parametric species denote, respectively, both sides of a transmembrane complex. The side of the complex whose content is $X$ may translocate while the other side stays in its current location. The result of this operation in the two possible projections, is informally depicted on both sides of the generator and corresponds to the diffusion of a transmembrane complex along the neck. Finally, the intra generator stands for intra-molecular complex formation[1].

**Definition 8.5.1** (Mixture)**.** Say that a term $P$ is a *mixture* if:

- $w(P) = 1$, $\mathit{fn}(P) = \emptyset$ and $P$ is parameter free
- Site edges have exactly two sites and do not cross compartments
- Backbone hyper edges cross at most one compartment
- $P$ is structurally equivalent to a term that contains no species node.

□

[1] This generator cannot be obtained as a refinement of connect since $\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(\bullet)$ is not a valid local context.

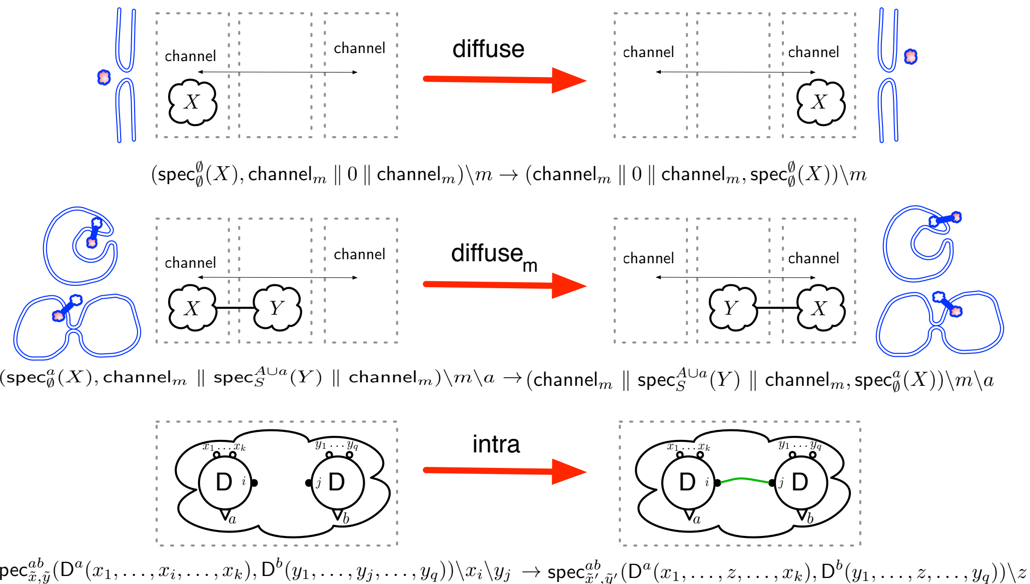



Figure 8.5: $\mathcal{C}_3$ generators. In the intra generator, let $\tilde{x} \stackrel{def}{=} \{x_1, \ldots, x_k\}$ and $\tilde{y} \stackrel{def}{=} \{y_1, \ldots, y_q\}$, $\tilde{x}' \stackrel{def}{=} \tilde{x}\{z/x_i\}$ and $\tilde{y}' \stackrel{def}{=} \tilde{y}\{z/y_j\}$

The last condition essentially states that species nodes that are present in a mixture are derivable from a species free mixture to which the above structural congruence rules have been applied. To ensure this one simply needs to verify the simple syntactical condition:

**Proposition 8.5.2.** *A global term of the form $G = \mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T)$ is a mixture if and only if $T$:*

- *$T$ is a mixture.*
- *$fn(T) = \mathsf{B} \cup \mathsf{S}$.*
- *$T$ is a connected component.*

The above proposition guarantees that one may always eliminate species nodes of the form $\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T)$ from a mixture, provided the sets $\mathsf{B}$ and $\mathsf{S}$ capture the free names of $T$ and provided $T$ defines a single connected set of agents. Note however, that general global terms need not be mixtures and one may have occurrences of species node in rules that do not satisfy this condition as it is for instance the case in the diffuse$_m$ generator. Yet not all species node make sense in a $\mathcal{C}_3$ expression. For instance $\mathsf{spec}^{\emptyset}_{\emptyset}(\mathsf{D}^a_m(x), X)$ will never have a match in any mixture since the structural congruence for species node introduction will always insure that the free names $a, m$ and $x$ will appear in the superscript and subscript of spec. The following proposition defines *well-formed* expressions with species nodes:

**Proposition 8.5.3.** *For any term $G = \mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T)$, there exists a mixture $M$ such that $G$ has a match in $M$ if and only if:*

- *$fn(T) \subseteq \mathsf{B} \cup \mathsf{S}$*
- *and either:*
  - *$T$ is connected*

– $\mathcal{V}(T) \neq \emptyset$ *and* $fn(T) \neq \emptyset$
– $fn(T) = \emptyset$ *and* $T = X_1, \ldots, X_n$ *for some parameters* $X_i$ .

Note that the second condition says that either $T$ needs to be already connected in the expression or leave "room enough" so that the context will make $T$ connected. It is easy to check that all the generators introduced in Fig. 8.5 satisfy this condition.

**Lemma 8.5.4** (Preservation)**.** *Let* $\mathcal{R}$ *be a set of generated rules and let* $P$ *be a mixture. If* $P \rightarrow_r Q$ *with* $r \in \mathcal{R}$ *then* $Q$ *is a mixture.*

As a corollary of the above lemma and Proposition 8.5.2, one has that a term containing $\mathsf{spec}^{\mathsf{B}}_{\mathsf{S}}(T, S)$ can only have a match in a mixture where $T$ and $S$ are part of the same connected component, which is a guarantee of the soundness of the intra generator.

## 8.6 Conclusion

The idea that models of signaling pathways or protein assembly should be considered as programs is now wending its way through the systems biology crowd. This is an appealing fact to language theoreticians, because it implies that one needs to accomplish in Systems Biology the same mutation that was accomplished in software engineering, when programs became too cumbersome and unwieldy to be developed in a non uniform way. This suggests that systems biology will soon require the development of high level languages, debuggers, and IDEs to compensate for the increasing gap between accumulation of data and its representation in executable models. The work we have presented here is an attempt to comply with Fontana's requirement that *"a model should be a data structure that contains a transparent, formal, and executable representation of the facts it rests upon"* [13]. In order to do so, we have structured our language in order to be able to tune the resolution level of the entities we wanted to describe: from anonymous domains, to molecular species, and from membrane patches to full fledged compartments.

We have already mentioned several approaches that were conducted with similar motivations, some of which we took inspiration from. Yet, we believe that the presented language offers a level of expressivity that was not accessible before in a single formalism. In particular we should mention that our language strictly contains the $\kappa$-calculus and corresponds to a particular class of bigraphical reactive systems that is yet to be defined formally[2]. Obviously, expressiveness and relative ease of use is not enough and future work should aim at developing quantitative simulation and analysis techniques. Here again, previous works have paved the way for such developments. In particular, proximity with the $\kappa$-calculus for which such analysis and simulation technique have been defined [7, 8] and the stochastic semantics for bigraphs [15], should be of great help.

## 8.7 Bibliography


[1] Andrei, O. and H. Kirchner, *Graph rewriting and strategies for modeling biochemical networks*, in: *Proc. SYNASC*, 2007, pp. 407–414.

[2] Bacci, G., D. Grohmann and M. Miculan, *A framework for protein and membrane interactions*, in: *Proc. MeCBIC'09*, 2009, pp. 19–33.

[3] Cardelli, L., *Brane calculi - interactions of biological membranes*, in: *Computational Methods in Systems Biology*, Springer, 2004 pp. 257–278.

[4] Cardelli, L., *Bitonal membrane systems - interactions of biological membranes*, Theoretical Computer Science **404** (2008).


[2]This may prove to be a complex task, since projectivity is not a trivial concept to capture with the standard definition of bigraphs. See Ref. [6] for some hints on how to do this.

[5] Chabrier, N. and F. Fages, *Symbolic model checking of biochemical networks*, in: *Proc. CMSB'03*, LNCS **2602**, 2003, pp. 146–162.

[6] Damgaard, T. C. and J. Krivine, *A generic language for biological systems based on bigraphs*, Technical Report 115, IT University of Copenhagen (2009).

[7] Danos, V., J. Feret, W. Fontana, R. Harmer and J. Krivine, *Abstracting the differential semantics of rule-based models: exact and automated model reduction*, in: *IEEE Symposium LICS*, 2010, pp. 362–381.

[8] Danos, V., J. Féret, W. Fontana and J. Krivine, *Scalable simulation of cellular signaling networks*, in: *Proc. APLAS'07*, LNCS **4807**, 2007, pp. 139–157.

[9] Danos, V. and C. Laneve, *Core formal molecular biology*, in: *Proc. ESOP'03*, LNCS **2618**, 2003, pp. 302–318.

[10] Danos, V. and C. Laneve, *Graphs for formal molecular biology*, in: *Proc. CMSB'03*, LNCS **2602**, 2003, pp. 34–46.

[11] Danos, V. and S. Pradalier, *Projective brane calculus*, in: *Proc. CMSB'04*, 2004, pp. 134–148.

[12] Faeder, J. R., M. L. Blinov and W. S. Hlavacek, *Rule based modeling of biochemical networks*, Complexity (2005), pp. 22–41.

[13] Fontana, W., *Systems biology, models, and concurrency*, in: *Proc. POPL'08*, 2008, pp. 1–2.

[14] John, M., C. Lhoussaine, J. Niehren and C. Versari, *Biochemical reaction rules with constraints*, in: *Proc. ESOP 2011*, LNCS **6602**, 2011, pp. 338–357.

[15] Krivine, J., R. Milner and A. Troina, *Stochastic bigraphs*, in: *Proceedings of MFPS XXIV*, ENTCS **218**, 2008, p. 7396.

[16] Milner, R., "The Space and Motion of Communicating Agents," Cambridge University Press, 2009.

[17] Phillips, A. and L. Cardelli, *Efficient, correct simulation of biological processes in the stochastic pi-calculus*, in: *CMSB*, 2007, pp. 184–199.

[18] Priami, C. and P. Quaglia, *Beta binders for biological interactions*, in: *Computational Methods in Systems Biology*, LNCS **3082**, 2005, pp. 20–33.

[19] Păun, G. and F. J. Romero-Campero, *Membrane computing as a modeling framework. cellular systems case studies*, in: *Formal Methods for Computational Systems Biology*, LNCS **5016**, 2008, pp. 168–214.

[20] R.Barbuti, A.Maggiolo-Schettini, P.Milazzo and A.Troina, *A calculus of looping sequences for modelling microbiological systems*, Fundamenta Informaticæ**72** (2006), pp. 21–35.

[21] Regev, A., E. M. Panina, W. Silverman, L. Cardelli and E. Shapiro, *Bioambients: An abstraction for biological compartments*, Theoretical Computer Science **325** (2004), pp. 141–167.

[22] Regev, A., W. Silverman and E. Y. Shapiro, *Representation and simulation of biochemical processes using the $\pi$-calculus process algebra*, in: *Pacific Symposium on Biocomputing*, 2001, pp. 459–470.

## 8.A Retrieving the $\kappa$-calculus.

In this section we show how one may naturally represent any $\kappa$-calculus model at the $\mathcal{C}_1$ level of our language. As the encoding is rather straightforward from a technical point of view, we shall simply describe here the translation of a particular example. We then show how $\mathcal{C}_3$ enables us to go beyond what one can express in $\kappa$.

## 8.A.1 The $\kappa$-calculus

We consider here the definition of $\kappa$ that is implemented in the $\kappa$-simulator KaSim[3]. Terms of the $\kappa$-calculus are built on the following grammar:

**Definition 10** ($\kappa$-Agents)**.**

| | | | | | |
|---|---|---|---|---|---|
| *(i)* | *agent* | $a$ | $::=$ | $N(\sigma)$ | |
| *(ii)* | *agent name* | $N$ | $::=$ | $A \in \mathcal{A}$ | |
| *(iii)* | *interface* | $\sigma$ | $::=$ | $\emptyset \mid s, \sigma$ | |
| *(iv)* | *site* | $s$ | $::=$ | $n_\iota^\lambda$ | |
| *(v)* | *site name* | $n$ | $::=$ | $x \in \mathcal{S}$ | |
| *(vi)* | *internal state* | $\iota$ | $::=$ | $\epsilon$ | *(any state)* |
| | | | | $\mid m \in \mathbb{V}$ | |
| *(vii)* | *binding state* | $\lambda$ | $::=$ | $\epsilon$ | *(free)* |
| | | | | $\mid -$ | *(semi-link)* |
| | | | | $\mid ?$ | *(wild-card)* |
| | | | | $\mid i \in \mathbb{N}$ | |

Expressions are simply formed by concatenation of agents $E ::= a, E \mid \emptyset$. Every agent represents a molecular entity (such as a protein) that has *sites* that can be used for complex formation (i.e., binding with other sites). For instance the expression:

$$\texttt{EGF(r}^1\texttt{),ErbB1(l}^1\texttt{,CR}^3\texttt{,Y1016}_p\texttt{,Y1092}_p^?\texttt{),EGF(r}^2\texttt{),ErbB1(l}^2\texttt{,CR}^3\texttt{,Y1092}_u^-\texttt{)}$$

corresponds to a molecular soup containing two instances of the agent `ErbB1` (a membrane receptor for the *epidermial growth factor* protein) and two instances of the agent `EGF` (the growth factor signal). In $\kappa$, each agent name comes with a fixed *signature* $\Sigma : N \to \mathcal{P}(\mathcal{S})$ that specifies the names of the sites each instance has. For instance $\Sigma(\texttt{ErbB1}) \stackrel{def}{=} \{\texttt{l}, \texttt{CR}, \texttt{Y1016}, \texttt{Y1092}, \dots\}$. Note that the protein `ErbB1` has in fact numerous tyrosine domains (whose name are of the form `Yxxx` where `xxx` corresponds to some amino acid position in the chain) that we do not list here. As a convention in $\kappa$, one does not represent sites that take no part in a given rule. In the example above, the site `Y1092` is left aside in one of the instances of `ErbB1`.

The superscript on a site indicate its *binding state*. The empty superscript $\epsilon$ marks a site that is free of any connection, _ indicates that the site is bound to an unspecified partner, ? indicates a site that is either free or bound and an integer is used to denote an explicit edge, as the one that connects the site `r` of the leftmost `EGF` to the site `l` of `ErbB1`.

The subscript on a site indicate its *internal state*. This is essentially a placeholder for a tag that serves to identify sites that have been chemically modified. Note that the absence of tag indicates that one does not care about its internal state in the expression.

## 8.A.2 The $\kappa$-calculus in $\mathcal{C}_1$

Fig. 8.6 shows the $\mathcal{C}_1$ representation of the above $\kappa$-expression.

The convention we adopt for the representation of $\kappa$-terms is the following: we use a domain node for each site of the kappa expression to translate. Internal states, site and agent names are represented by *info* nodes. Sites that belong to the same $\kappa$-agents will share the same backbone. Sites that are connected in the $\kappa$ expression will be bound in the $\mathcal{C}_1$-term. Notice that the backbone of both instances of `ErbB1` are both open. This captures the fact that not all sites of the signature of `ErbB1` are present in the expression.

[3] `http://kappalanguage.org`

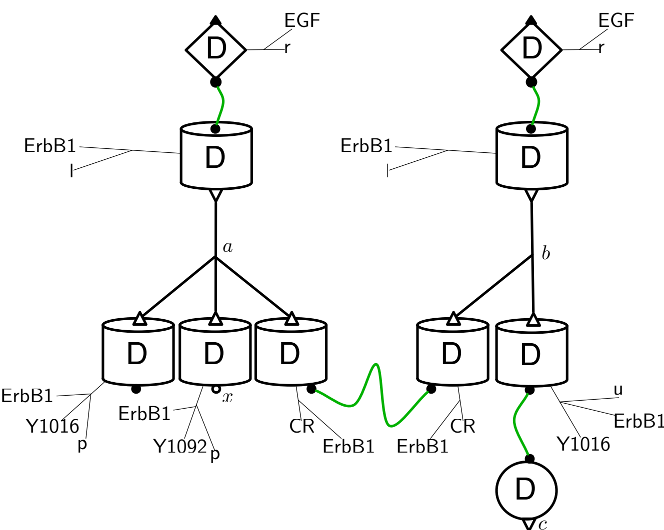



Figure 8.6: Representation of the $\kappa$ expression into $\mathcal{C}_1$.

### 8.A.3 Expressiveness of $\mathcal{C}_3$

As said, `ErbB1` proteins are in fact membrane receptors. `ErbB1` protein is composed of an extra cellular domain that holds the ligand binding site `l` and an intra cellular domain that bears the other interaction sites. Now that we have represented our expression in a richer language, it becomes natural to represent these facts as we show in Fig. 8.7.

A key regulatory mechanism of the EGF pathway is called *receptor internalization*. It is a mechanism by which receptors become trapped in inner vesicles that may eventually bubble down to the cytoplasm of the cell. This prevents the receptor from binding to new incoming signals. It is not possible to represent this behavior in $\kappa$ for two reasons. The first reason, which we have already solved, is that there is no way to represent compartments in $\kappa$. The second reason is more subtle. Indeed, during receptor internalization, not only will `ErbB1` get trapped inside the vesicle, but along with it will be any protein complex attached to its extra cellular domain. In the example of Fig. 8.7, one should capture also the `EGF` ligand that is bound to it. This is what we do in Fig. 8.8 by defining an internalization rule that utilizes a `species` node.

## 8.B Proof of the soundness Theorem

**Definition 8.B.1.** Let $\epsilon$ denote the empty word on $\Pi^*$ and $\cdot$ the concatenation of words. The *abstraction* map on generic wide contexts $\alpha : T_\bullet \to (\Pi \backslash \{\bot\})^*$ is defined as:

$$
\begin{array}{lcl}
\alpha(T) & \stackrel{def}{=} & \epsilon \\
\alpha((T_\bullet)\backslash u) & \stackrel{def}{=} & \alpha(T_\bullet) \\
\alpha(\bullet) & \stackrel{def}{=} & \bullet \\
\alpha(T_\bullet, S_\bullet) & \stackrel{def}{=} & \alpha(T_\bullet) \cdot \alpha(S_\bullet) \\
\alpha(\mathsf{C}_m(T_\bullet)) & \stackrel{def}{=} & (\!|\cdot\alpha(T_\bullet)\cdot|\!)
\end{array}
$$

□

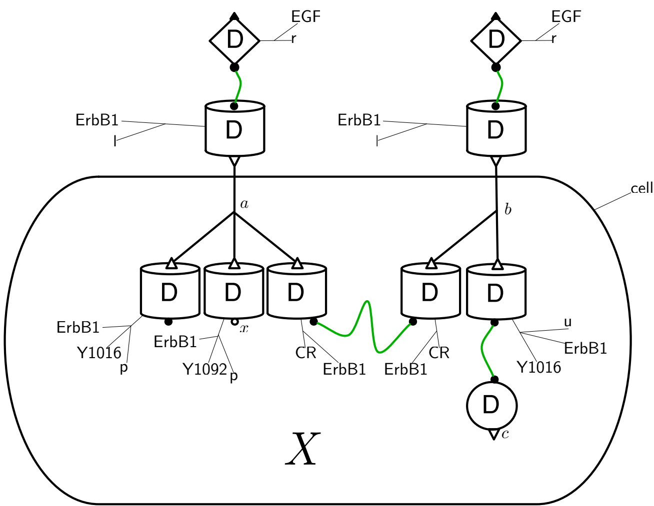



Figure 8.7: Adding compartments to the the $\kappa$ expression.

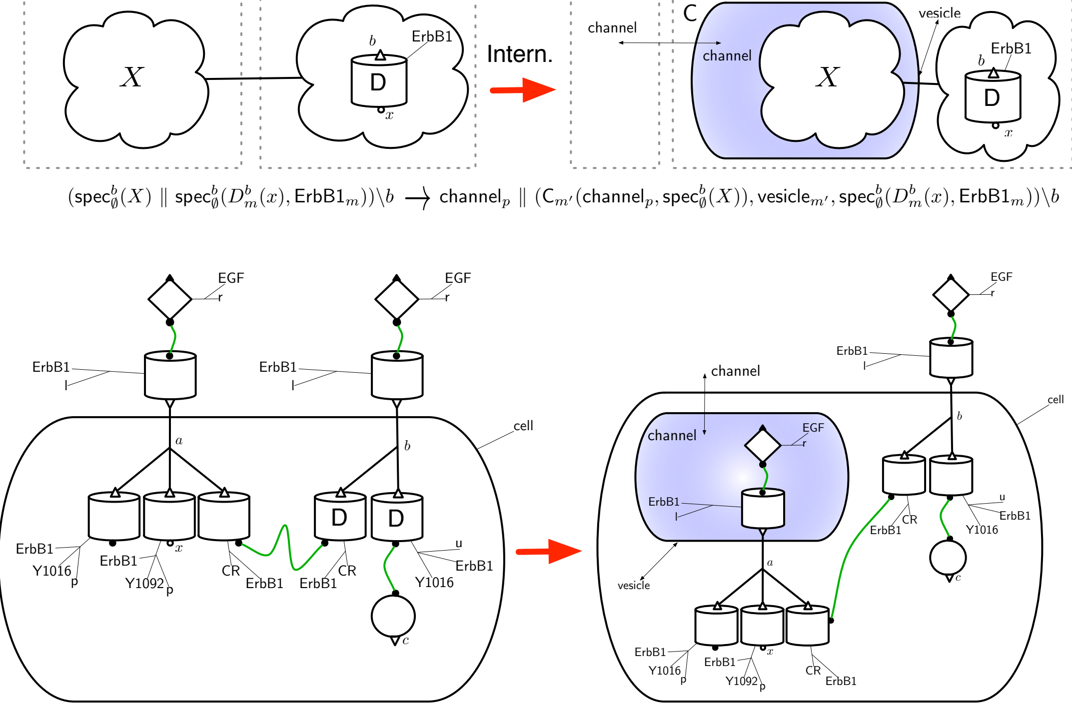



Figure 8.8: The Intern. rule is obtained by composition of the pinch and diffuse$_{\mathsf{m}}$ generators and invoking the species node where it is needed. Below is the result of the application of this rule to our example. Notice that the receptor gets internalized together with its ligand protein EGF.

**Lemma 8.B.2.** *Let $P$ be a wide term such that $P \hookrightarrow_\pi T_\bullet$ for some $\pi \in (\Pi\setminus\{\bot\})^*$. Then $\alpha(T_\bullet) = \pi$.*

*Proof.* By induction on the derivation of $P \hookrightarrow_\pi T_\bullet$.

**(Base case)**. $T \hookrightarrow_\bullet \mathbb{C}[\bullet]$ by (ax.). Since $\mathbb{C}$ is a local context with exactly one hole, the unique $\bullet$ cannot be wrapped in a compartment. From Def.8.B.1 it follows that $\alpha(\mathbb{C}[\bullet]) = \bullet$.

**(Inductive step)**.

- (rest). By induction hypothesis we have $\alpha(T_\bullet) = \pi$.
- (wrap). By induction hypothesis we have $\alpha(T_\bullet) = \pi$. According to Def 8.B.1 we have $\alpha(\mathsf{C}(T_\bullet)) = (\!|\alpha(T_\bullet)|\!) = (\!|\cdot\pi\cdot|\!)$.
- (comp). By induction hypothesis we have $\pi_0 = \alpha(T_\bullet)$ and $\pi_1 = \alpha(S_\bullet)$. Since $\mathbb{C}[\bullet]$ is a local context with exactly one hole, we have that $\alpha(\mathbb{C}[T_\bullet, S_\bullet]) = \alpha(T_\bullet, S_\bullet)$ which, by Def 8.B.1 is equal to $\alpha(T_\bullet)\cdot\alpha(S_\bullet) = \pi_0\cdot\pi_1$. □

□

Let $\bar{\pi}$ denote a well parenthesized word of the form $(\!|\cdot\pi\cdot|\!)$.

**Lemma 8.B.3.** *If $\pi$ is a derivable projective constraint, then $\pi \neq \pi_0 \bullet \bar{\pi}_1 \bullet \pi_2$ for all $\pi_0, \pi_1, \pi_2$.*

*Proof.* By induction on the size of $\pi_1$.

**(Base case)**. According to Def 8.4.3 we have $\bullet(\!|\bullet|\!)\bullet \to \bot$ so $\pi \not\to \bot$ implies $\pi \neq \bullet(\!|\bullet|\!)\bullet$ .

**(Inductive step)**. By *red. ad abs.* suppose:

$$\pi = \pi_0 \bullet (\!|\pi_1|\!) \bullet \pi_2$$

Since $\pi \not\to \bot$ we have necessarily $\pi_1 = \bullet\pi_1'\bullet$ (since $\bullet(\!|(\!|\to \bot$ and $|\!)|\!)\bullet \to \bot$ and $(\!||\!)$ is not derivable). Now by induction hypothesis $\pi_1' \neq \bar{\pi}''$, so the only possibility is $\pi_1' = |\!)\pi_3(\!|$ for some $\pi_3$. So we obtain:

$$\pi = \pi_0 \bullet (\!|\bullet|\!)\pi_3(\!|\bullet|\!) \bullet \pi_2$$

From $(\!|\cdot|\!) \to \bot$ it follows that $\pi_3 = \bullet\pi_3'\bullet$ and we have a contradiction. □

□

In order to define the projective distance between two occurrences of $\bullet$ symbols in a word $\pi$, we need the following labeling operation.

**Definition 8.B.4** (labeling)**.** Assume an infinite set of labels $\Lambda$. Let $\Pi_+ = \{|\!)_\alpha, (\!|_\alpha, \bullet_i\}$ be a decoration of the alphabet $\Pi$ where $\alpha \in \Lambda$ and $i \in \mathbb{N}$. We define the *labeling* function $\ell : (\Pi\setminus\{\bot\})^* \times \mathbb{N} \times \Lambda^* \to \Pi_+$ as:

$$\begin{array}{lcl}
\ell(\bullet\cdot\pi)(i)(\lambda) & \stackrel{def}{=} & \bullet_i \cdot \ell(\pi)(i+1)(\lambda) \\
\ell((\!|\cdot\pi)(i)(\lambda) & \stackrel{def}{=} & (\!|_\alpha \cdot \ell(\pi)(i)(\alpha\cdot\lambda) \text{ with } \alpha\in\Lambda \text{ fresh} \\
\ell(|\!)\cdot\pi)(i)(\alpha\cdot\lambda) & \stackrel{def}{=} & |\!)_\alpha \cdot \ell(\pi)(i)(\lambda) \\
\ell(\epsilon)(i)(\lambda) & \stackrel{def}{=} & \epsilon
\end{array}$$

□

We now consider labelled versions of projective constraints.

**Definition 8.B.5.** Let $|_\alpha$ denote either $(\!|_\alpha$ or $|\!)_\alpha$ and let $\alpha \odot \lambda$ be defined as $\alpha \odot \lambda \stackrel{def}{=} \alpha \cdot \lambda$ if $\lambda \neq \alpha \cdot \lambda'$and $\alpha \odot \lambda \stackrel{def}{=} \lambda$ otherwise. Let $|\lambda|$ denote the size of the word $\lambda$. Define $\Delta^{\#}_{i,j}(\pi)$ the projective distance between $\bullet_i$ and $\bullet_j$ in $\pi$ as:

$$\begin{array}{lcl} \Delta^{\#}_{i,j}(\bullet_k \cdot |_\alpha \cdot \bullet_{k+1} \cdot \pi)(\lambda) & \stackrel{def}{=} & \Delta^{\#}_{i,j}(\bullet_{k+1} \cdot \pi)(\lambda) \text{ if } k < i \\ \Delta^{\#}_{i,j}(\bullet_k \cdot |_\alpha \cdot \bullet_{k+1} \cdot \pi)(\lambda) & \stackrel{def}{=} & \Delta^{\#}_{i,j}(\bullet_{k+1} \cdot \pi)(\alpha \odot \lambda) \text{ if } k \geq i \text{ and } k < j \\ \Delta^{\#}_{i,j}(\bullet_k \cdot \pi)(\lambda) & \stackrel{def}{=} & |\lambda| \text{ if } k \geq j \end{array}$$

We write $\Delta^{\#}_{i,j}(\pi)$ for $\Delta^{\#}_{i,j}(\pi)(\epsilon)$ □

The following proposition will be useful for the upcoming proofs.

**Proposition 8.B.6.** *The following equalities hold:*

$$\begin{array}{lcll} \Delta^{\#}_{i,j}(\bullet_k\pi)(\epsilon) & = & \Delta^{\#}_{k,j}(\bullet_k\pi)(\epsilon) & \text{if } i \leq k < j \\ \Delta^{\#}_{i,j}(\pi)(\alpha \cdot \lambda) & = & \Delta^{\#}_{i,j}(\pi)(\lambda) + 1 & \text{if } \alpha \notin \pi \\ \Delta^{\#}_{i,j}(\pi_0 \bullet_i \pi_1)(\epsilon) & = & \Delta^{\#}_{i,j}(\bullet_i\pi_1)(\epsilon) & \\ \Delta^{\#}_{i,j}(\pi_0 \bullet_j \pi_1)(\epsilon) & = & \Delta^{\#}_{i,j}(\pi_0\bullet_j)(\epsilon) & \end{array}$$

**Lemma 8.B.7.** *Let $\pi = \pi_0 \bullet_i \pi_1 \bullet_j \pi_2 \bullet_k \pi_3$ be a derivable projective constraint, for some $\pi_0, \pi_1, \pi_3$. We have:*

$$\Delta^{\#}_{i,k}(\pi) = \Delta^{\#}_{i,j}(\pi) + \Delta^{\#}_{j,k}(\pi)$$

*Proof.* By induction on $k - i$.

**(Base case)**. Suppose $\pi = \pi_0 \bullet_i |_\alpha \bullet_{i+1} |_\beta \bullet_{i+2} \pi_1$. Since $\pi$ is derivable, $\alpha \neq \beta$ otherwise $\pi \to \perp$. We have:

$$\begin{array}{lcl} \Delta^{\#}_{i,i+2}(\pi) & = & \Delta^{\#}_{i,k}(\bullet_i|_\alpha \bullet_{i+1} |_\beta\bullet_{i+2})(\epsilon) \\ & = & \Delta^{\#}_{i,k}(\bullet_{i+1}|_\beta\bullet_{i+2})(\alpha) \\ & = & |\alpha \cdot \beta| \\ & = & 2 = \Delta^{\#}_{i,i+1}(\pi) + \Delta^{\#}_{i+1,i+2}(\pi) \end{array}$$

**(Inductive step)**. Suppose $\pi = \pi_0 \bullet_i |_\alpha\pi'_1 \bullet_j \pi_2 \bullet_k \pi_3$. We have:

$$\begin{array}{lcl} \Delta^{\#}_{i,k}(\pi) & \stackrel{def}{=} & \Delta^{\#}_{i,k}(\bullet_i|_\alpha \bullet_{i+1} \pi'_1 \bullet_j \pi_2\bullet_k)(\epsilon) \\ & \stackrel{def}{=} & \Delta^{\#}_{i,k}(\bullet_{i+1}\pi'_1 \bullet_j \pi_2\bullet_k)(\alpha) \end{array}$$

Using Lemma 8.B.3 we know that $\alpha \neq \pi'_1$ and $\alpha \neq \pi_2$. So we have:

$$\begin{array}{lcl} \Delta^{\#}_{i,k}(\pi) & = & (\Delta^{\#}_{i,k}(\bullet_{i+1}\pi'_1 \bullet_j \pi_2\bullet_k)(\epsilon)) + 1 \\ & = & (\Delta^{\#}_{i+1,k}(\bullet_{i+1}\pi'_1 \bullet_j \pi_2\bullet_k)(\epsilon)) + 1 \end{array}$$

By induction hypothesis we obtain:

$$\begin{array}{lcl} \Delta^{\#}_{i,k}(\pi) & = & (\Delta^{\#}_{i+1,j}(\bullet_{i+1}\pi'_1\bullet_j) + \Delta^{\#}_{j,k}(\bullet_j\pi_2\bullet_k)) + 1 \\ \Delta^{\#}_{i,k}(\pi) & = & (\Delta^{\#}_{i+1,j}(\bullet_{i+1}\pi'_1\bullet_j) + 1) + \Delta^{\#}_{j,k}(\bullet_j\pi_2\bullet_k) \\ \Delta^{\#}_{i,k}(\pi) & = & \Delta^{\#}_{i,j}(\bullet_i|_\alpha \bullet_{i+1} \pi'_1\bullet_j) + \Delta^{\#}_{j,k}(\bullet_j\pi_2\bullet_k) \\ \Delta^{\#}_{i,k}(\pi) & = & \Delta^{\#}_{i,j}(\pi) + \Delta^{\#}_{j,k}(\pi) \end{array}$$

□

□

**Corollary 8.B.8.** *For any derivable $\pi$, $\Delta^{\#}_{i,i+k}(\pi) = k$.*

*Proof.* By induction on $k$.

**(Base case)**. The base case is $\Delta^{\#}_{i,i+1}(\pi) = 1$ which is true, by definition of $\Delta^{\#}$.

**(Inductive step)**. Now we want to find $\Delta^{\#}_{i,i+k}(\pi)$ for some $\pi$ of the form $\pi = \pi_0 \bullet_i |_\alpha \pi_1 \bullet_{i+k} \pi_2$. We have:

$$\begin{array}{rcl} \Delta^{\#}_{i,i+k}(\pi)(\epsilon) & \stackrel{def}{=} & \Delta^{\#}_{i,i+k}(\bullet_i|_\alpha \bullet_{i+1} \pi_1 \bullet_{i+k})(\epsilon) \\ & = & \Delta^{\#}_{i,i+k}(\bullet_{i+1}\pi_1\bullet_{i+k})(\alpha) \end{array}$$

Using Lemma 8.B.3 we know that $\alpha \notin \pi_1$ so:

$$\begin{array}{rcl} \Delta^{\#}_{i,i+k}(\pi)(\epsilon) & = & \Delta^{\#}_{i,i+k}(\bullet_{i+1}\pi_1\bullet_{i+k})(\epsilon) + 1 \\ & = & \Delta^{\#}_{i+1,i+k}(\bullet_{i+1}\pi_1\bullet_{i+k})(\epsilon) + 1 \end{array}$$

By induction hypothesis we have:

$$\Delta^{\#}_{i,i+k}(\pi)(\epsilon) = (k-1) + 1 = k$$

□

□

We obtain the soundness Theorem as a corollary of Lemma 8.B.2 and Lemma 8.B.7.

*Theorem 8.4.5.* Let $P = (T_1 \parallel \ldots \parallel T_n)\backslash V$, and $P \hookrightarrow_\pi T_\bullet$ with $\langle T_\bullet, \mu\rangle$ a match for $P$ in $S$ for some $\mu$ ($|\mu| = V(P)$). We have $\Delta_{T_i,T_i+k}(P) \stackrel{def}{=} k$. Suppose $P \hookrightarrow_\pi \langle T_\bullet, \mu\rangle$. Using Lemma 8.B.2, we have $\alpha(T_\bullet) = \pi$. Now using Corollary 8.B.8 we also know that $\Delta^{\#}_{i,i+k}(\pi) = k$ and we have $\Delta_{T_i,T_{i+k}}(P) = \Delta^{\#}_{i,i+k}(\pi) = k$. We conclude by noticing that $\alpha(T_\bullet)$ preserves nesting distances between holes (it doesn't remove compartments that contain holes). So, $\Delta_{T_i,T_{i+k}}(P) = \Delta_{T_i,T_{i+k}}(S)$.
□
□

## 8.C Proof of the completeness Theorem

In order to prove Theorem 8.4.6 we need some properties on extensions.

**Lemma 8.C.1.** *Let $\mathsf{C}^n(T)$ denote a term of the form:*

$$\mathsf{C}(\mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[\ldots \mathsf{C}(\mathbb{C}_n[T])\ldots])])$$

*For all wide term $P = (T_1 \parallel \ldots \parallel T_n)\backslash V$, we have:*

$$P \hookrightarrow_{\pi\bullet} T \Rightarrow T = \mathbb{C}[T_n] \tag{8.1}$$

$$P \hookrightarrow_{\bullet\pi} T \Rightarrow T = \mathbb{C}[T_1] \tag{8.2}$$

$$P \hookrightarrow_{\pi\rrparenthesis} T \Rightarrow T = \mathbb{C}[\mathsf{C}^k(T_n)] \tag{8.3}$$

$$P \hookrightarrow_{\llparenthesis\pi} T \Rightarrow T = \mathbb{C}[\mathsf{C}^k(T_1)] \tag{8.4}$$

*for some* $k \geq 2$ *and:*

$$P \hookrightarrow_{\pi\bullet\rrparenthesis} T \Rightarrow T = \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[T_n])] \tag{8.5}$$

$$P \hookrightarrow_{\llparenthesis\bullet\pi} T \Rightarrow T = \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[T_1])] \tag{8.6}$$

$$P \hookrightarrow_{\llparenthesis\bullet\rrparenthesis\bullet\pi} T \Rightarrow T = \mathbb{C}[T_2] \tag{8.7}$$

*Lemma 8.C.1.* By induction on $|\pi|$. For simplicity we consider here terms without restriction on names, without loss of generality.

[**cases** $(8.1)_0$ **and** $(8.2)_0$] For both Equations (8.1) and (8.2) the only derivation producing a $\bullet$ symbol is (ax.) which gives the expected conclusion.

[**case** $(8.3)_0$ **and** $(8.4)_0$] The smallest $\pi$ such that $P \hookrightarrow_{\pi\rrparenthesis} T$ or $P \hookrightarrow_{\llparenthesis\pi} T$ is respectively $\pi = \llparenthesis\llparenthesis\bullet$ and $\pi = \bullet\rrparenthesis\rrparenthesis$. They both stem from the derivation:

$$\dfrac{\dfrac{\overline{T_1 \hookrightarrow_{\bullet} \mathbb{C}_1[T_1] = T'}\ (\mathsf{ax.})}{P \hookrightarrow_{\llparenthesis\bullet\rrparenthesis} \mathsf{C}(T')}\ (\mathsf{wrap})}{P \hookrightarrow_{\llparenthesis\llparenthesis\bullet\rrparenthesis\rrparenthesis} \mathsf{C}(\mathsf{C}(T')}\ (\mathsf{wrap})$$

and we have $P \hookrightarrow_{\llparenthesis\llparenthesis\bullet\rrparenthesis\rrparenthesis} \mathsf{C}(\mathsf{C}(\mathbb{C}[T_1]) = \mathsf{C}(\mathsf{C}(\mathbb{C}[T_n])$ which is in the expected form.

[**case** $(8.5)_0$ **and** $(8.6)_0$] The smallest $\pi$ for $P \hookrightarrow_{\pi\bullet\rrparenthesis} T$ is $\pi = \llparenthesis$ and the only possible derivation is:

$$\dfrac{\overline{P \hookrightarrow_{\bullet} T}\ (\mathsf{ax.})}{P \hookrightarrow_{\llparenthesis\bullet\rrparenthesis} \mathsf{C}(T)}\ (\mathsf{wrap})$$

It follows that $P = T_1$ and $T = \mathbb{C}[T_1]$. So we have:

$$P \hookrightarrow_{\llparenthesis\bullet\rrparenthesis} \mathsf{C}(\mathbb{C}[T_1])$$

which is in the expected form. One proceeds in a symmetric manner for $(8.6)_0$.

[**case** $(8.7)_0$] The derivation is :

$$\dfrac{T_1 \hookrightarrow_{\llparenthesis\bullet\rrparenthesis} \mathsf{C}(\mathbb{C}_1[T_1]) \quad T_2 \hookrightarrow_{\bullet} \mathbb{C}_2[T_2]}{P = T_1 \parallel T_2 \hookrightarrow_{\llparenthesis\bullet\rrparenthesis\bullet} T}$$

It entails that $T = \mathsf{C}(\mathbb{C}_1[T_1]), \mathbb{C}_2[T_2]$ which can also be written $\mathbb{C}[T_2]$ with $\mathbb{C}[\bullet] = \mathsf{C}(\mathbb{C}_1[T_1]), \mathbb{C}_2[\bullet]$.

[**case** $(8.1)_n$] Suppose:

$$\frac{P \hookrightarrow_{\pi_0} T \qquad Q \hookrightarrow_{\pi_1 \bullet} S \qquad \pi_0\pi_1 = \pi}{P \parallel Q \hookrightarrow_{\pi \bullet} T, S}$$

By induction hypothesis on $Q \hookrightarrow_{\pi_1 \bullet} S$ one has $S = \mathbb{C}[T_n]$ which in turn implies:

$$P \parallel Q \hookrightarrow_{\pi \bullet} T, \mathbb{C}[T_n]$$

By defining $\mathbb{C}'[\bullet] \stackrel{def}{=} T, \bullet$, one obtains $P \parallel Q \hookrightarrow_{\pi \bullet} \mathbb{C}'[T_n]$

[**case** $(8.2)_n$] Suppose:

$$\frac{P \hookrightarrow_{\bullet \pi_0} T \qquad Q \hookrightarrow_{\pi_1} S \qquad \pi_0\pi_1 = \pi}{P \parallel Q \hookrightarrow_{\bullet \pi} T, S}$$

One proceeds as before using induction hypothesis on $P \hookrightarrow_{\bullet \pi_0} T$.

[**case** $(8.3)_n$] Suppose $P \hookrightarrow_{\pi \rrparenthesis\rrparenthesis} T$. There are two sub-cases:

[**case** **(wrap)**] The derivation was:

$$\frac{P \hookrightarrow_{\pi' \rrparenthesis} T'}{P \hookrightarrow_{\llparenthesis \pi' \rrparenthesis\rrparenthesis} \mathsf{C}(T')} \quad \textsf{(wrap)}$$

Again we have two possible cases for $\pi'$:

[**case** $\pi' = \pi'' \rrparenthesis$] We have $P \hookrightarrow_{\pi'' \rrparenthesis\rrparenthesis} T'$. Using the induction hypothesis we deduce:

$$\frac{P \hookrightarrow_{\pi' \rrparenthesis\rrparenthesis} \mathbb{C}[\mathsf{C}^k(T_n)] = T'}{P \hookrightarrow_{\llparenthesis \pi' \rrparenthesis\rrparenthesis} \mathsf{C}^{k+1}(T_n)} \quad \textsf{(wrap)}$$

which gives the desired form.

[**case** $\pi' = \pi'' \bullet$] Thanks to Lemma 8.C.1.(8.5) we know that $P \hookrightarrow_{\pi'' \bullet \rrparenthesis} T'$ implies $T' = \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2(T_n)]$. Hence we obtain $P \hookrightarrow_{\llparenthesis \pi' \rrparenthesis\rrparenthesis} \mathsf{C}^2(T_n)$ as required.

[**case** **(comp)**] The derivation was:

$$\frac{P \hookrightarrow_{\pi_0} T \qquad Q \hookrightarrow_{\pi_1} S \qquad \pi_0\pi_1 = \pi' \rrparenthesis\rrparenthesis}{P \parallel Q \hookrightarrow_{\pi' \rrparenthesis\rrparenthesis} T, S} \quad \textsf{(comp)}$$

It results that $\pi_1 = \pi_1' \rrparenthesis\rrparenthesis$ for some $\pi_1'$ since no derivation may produce $Q \hookrightarrow_{\rrparenthesis\rrparenthesis} S$ or $Q \hookrightarrow_{\rrparenthesis} S$. We can apply induction hypothesis to $Q \hookrightarrow_{\pi_1' \rrparenthesis\rrparenthesis} S$ from which we get the derivation:

$$P \parallel Q \hookrightarrow_{\pi' \rrparenthesis\rrparenthesis} T, \mathsf{C}^k(T_n)$$

Defining $\mathbb{C}[\bullet] = T, \bullet$ one has $P \parallel Q \hookrightarrow_{\pi' \rrparenthesis\rrparenthesis} \mathbb{C}[\mathsf{C}^k(T_n)]$ which is in the desired form.

[**case** $(8.4)_n$] The inductive step for $P \hookrightarrow_{(\!|(\!|\pi} T$ is symmetric to the previous case.

[**case** $(8.5)_n$] Suppose:

$$\frac{P \hookrightarrow_{\pi_0} T \qquad Q \hookrightarrow_{\pi_1} S \qquad \pi_0\pi_1 = \pi\bullet|\!)}{P \parallel Q \hookrightarrow_{\pi\bullet|\!)} T, S}$$

From $\pi_0\pi_1 = \pi\bullet|\!)$ it results that $\pi_1 = \pi_1'\bullet|\!)$ since $\pi_1 = |\!)$ or $\pi_1 = \bullet|\!)$ is not a valid derivation. One may now apply induction hypothesis to $Q \hookrightarrow_{\pi_1} S$ from which we get the derivation:

$$\frac{P \hookrightarrow_{\pi_0} T \qquad Q \hookrightarrow_{\pi_1} \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[T_n])]}{P \parallel Q \hookrightarrow_{\pi\bullet|\!)} T, \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[T_n])]}$$

Defining $\mathbb{C}_1'[\bullet] \overset{def}{=} \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[T_n])], \bullet$, one obtains the desired form:

$$P \parallel Q \hookrightarrow_{\pi\bullet|\!)} \mathbb{C}_1'[\mathsf{C}(\mathbb{C}_2[T_n])]$$

[**case** $(8.6)_n$] Suppose:

$$\frac{P \hookrightarrow_{\pi_0} T \qquad Q \hookrightarrow_{\pi_1} S \qquad \pi_0\pi_1 = (\!|\bullet\pi}{P \parallel Q \hookrightarrow_{(\!|\bullet\pi} T, S}$$

and the case is similar to [**case** $(8.5)_n$] using induction hypothesis on $P \hookrightarrow_{(\!|\bullet\pi_0'} T$.

[**case** $(8.7)_n$] Suppose:

$$\frac{P \hookrightarrow_{\pi_0} T \qquad Q \hookrightarrow_{\pi_1} S \qquad \pi_0\pi_1 = (\!|\bullet|\!) \bullet \pi}{P \parallel Q \hookrightarrow_{(\!|\bullet|\!)\bullet\pi} T, S}$$

Then either $\pi_0 = (\!|\bullet|\!)\bullet\pi_0'$ for some $\pi_0'$ in which case, by induction hypothesis we have $T = \mathbb{C}[T_2]$ and we can conclude, or $\pi_0 = (\!|\bullet|\!)$ and $\pi_1 = \bullet\pi$. If so, we have $P = T_1$ and $T = \mathsf{C}(\mathbb{C}_1[T_1])$ and by Lemma 8.C.1.(8.7) we have $S = \mathbb{C}_2[T_2]$. It follows that $T, S = \mathsf{C}(\mathbb{C}_1[T_1]), \mathbb{C}_2[T_2]$ which can be written in the desired form with $\mathbb{C}[\bullet] = \mathsf{C}(\mathbb{C}_1[T_1]), \mathbb{C}_2[\bullet]$. □

□

We proceed now with the proof of the completeness theorem.

*Completeness.* Let $P = T_1 \parallel \ldots \parallel T_k$ and $T = \mathbb{C}^k[T_1 \parallel \ldots \parallel T_k]$ with $\Delta_{T_i,T_j}(P) = \Delta_{T_1,T_j}(T)$ for all $i, j \in \{1, \ldots, k\}$. We prove $P \hookrightarrow_\pi T$ for some $\pi$ by induction on $s(T)$, the size of $T$, defined inductively as:

$$\begin{array}{lcl}
s(0) & \overset{def}{=} & 0 \\
s(D) & \overset{def}{=} & 1 \\
s(I) & \overset{def}{=} & 1 \\
s(X) & \overset{def}{=} & 1 \\
s(\mathsf{C}(T)) & \overset{def}{=} & 1 + s(T) \\
s(T, S) & \overset{def}{=} & s(T) + s(S)
\end{array}$$

For simplicity we consider here terms without restriction on names, without loss of generality.

[**case** $s(T) = 0$] We have $P = 0 \hookrightarrow_\bullet 0$ thanks to the (ax.) rule and the trivial local context $\mathbb{C}[\bullet] = \bullet$.

[**case** $s(T) = n$] By hypothesis we have $P = T_1 \parallel \ldots \parallel T_k$ and a context $\mathbb{C}^k$ with exactly $k$-holes such that $\mathbb{C}^k[T_1, \ldots, T_k] = T$ for some local term $T$ with $\Delta(P) = \Delta(T)$. We need to prove $P \hookrightarrow_\pi T$ for some $\pi$. There are two cases, either:

$$T = \mathsf{C}(\mathbb{C}^k[T_1, \ldots, T_k]) \tag{8.8}$$

or there are two contexts $\mathbb{C}^i, \mathbb{C}^{k-(i+1)}$ with exactly $i$ and $k-(i+1)$ holes such that:

$$T = \mathbb{C}^i[T_1, \ldots, T_i], \mathbb{C}^{k-(i+1)}[T_{i+1}, T_k] \tag{8.9}$$

[**case (8.8)**] According to Definition 8.4.2:

$$\begin{array}{rcl} \Delta(\mathsf{C}(\mathbb{C}^k[T_1, \ldots, T_k])) & \stackrel{def}{=} & \Delta(\mathbb{C}^k[T_1, \ldots, T_k]) \\ & = & \Delta(P) \end{array}$$

In addition, $\mathbb{C}^k[T_1, \ldots, T_k]) <_s T$ so we apply induction hypothesis to deduce $P \hookrightarrow_\pi \mathbb{C}^k[T_1, \ldots, T_k]$ and we can conclude using (wrap):

$$\frac{P \hookrightarrow_\pi \mathbb{C}^k[T_1, \ldots, T_k]}{P \hookrightarrow_{(\!|\pi|\!)} \mathsf{C}(\mathbb{C}^k[T_1, \ldots, T_k])}$$

[**case (8.9)**] Let $P \stackrel{def}{=} (P' \parallel Q)$ with $P' \stackrel{def}{=} (T_1 \parallel \ldots \parallel T_i)$ and $Q \stackrel{def}{=} (T_{i+1} \parallel \ldots \parallel T_k)$. According to Definition 8.4.2, for all $T_q, T_l \in P'$ we have:

$$\begin{array}{rcl} \Delta_{T_q,T_l}(T) & \stackrel{def}{=} & \Delta_{T_q,T_l}(\mathbb{C}^i[T_1, \ldots, T_i]) \\ \Delta_{T_q,T_l}(P) & \stackrel{def}{=} & \Delta_{T_q,T_l}(P') \end{array}$$

It follows that $\Delta(P') = \Delta(\mathbb{C}^i[T_1, \ldots, T_i])$. By the same reasoning we also have that $\Delta(Q) = \Delta(\mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k])$. Therefore, by induction hypothesis we have:

$$P' \hookrightarrow_{\pi_0} \mathbb{C}^i[T_1, \ldots, T_i] \tag{8.10}$$

and

$$Q \hookrightarrow_{\pi_1} \mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k] \tag{8.11}$$

Recall that according to definition 8.4.2, $\Delta_{T_i,T_{i+1}}(P' \parallel Q) \stackrel{def}{=} 1$. We show that $\pi_0\pi_1 \to \bot$ implies a contradiction. We have four cases to check:

$$\begin{array}{llcl} (i) & \pi_0 & = & \pi\bullet \\ (ii) & \pi_0 & = & \pi\bullet|\!) \\ (iii) & \pi_0 & = & \pi|\!)|\!) \\ (iv) & \pi_1 & = & (\!|(\!|\pi \end{array}$$

[**case** $(i)$] Using Lemma 8.C.1.(8.1) one has

$$\mathbb{C}^i[T_1, \ldots, T_i] = \mathbb{C}[T_i]$$

Now there are two possibilities in order to have $\pi_0\pi_1 = \bot$:

[**case** $\pi_1 = \bullet\pi_1'$] using Lemma 8.C.1.(8.2) one has

$$\mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k] = \mathbb{C}[T_{i+1}]$$

from which we deduce $\Delta_{T_i,T_{i+1}}(T) = 0$ which contradicts hypothesis $\Delta(P) = \Delta(T)$.

[**case** $\pi_1 = (\!|\bullet|\!) \bullet \pi_1'$] Using Lemma 8.C.1.(8.7) we have:

$$\mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k] = \mathbb{C}[T_{i+2}]$$

from which we deduce $\Delta_{T_i,T_{i+2}}(T) = 0$ which contradicts hypothesis $\Delta(P) = \Delta(T)$.

[**case** $(ii)$] Using Lemma 8.C.1.(8.5) one has

$$\mathbb{C}^i[T_1, \ldots, T_i] = \mathbb{C}_1[\mathsf{C}(\mathbb{C}_2[T_i])]$$

and using Lemma 8.C.1.(8.6) one has:

$$\mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k] = \mathbb{C}_3[\mathsf{C}(\mathbb{C}_4[T_{i+1}])]$$

It results that $\Delta_{T_i,T_{i+1}}(T) = 2$ which contradicts hypothesis $\Delta(P) = \Delta(T)$.

[**case** $(iii)$] Lemma 8.C.1.(8.3) implies:

$$\mathbb{C}^i[T_1, \ldots, T_i] = \mathsf{C}^k(T_i),\ k \geq 2$$

It results that $\Delta_{T_i,T_{i+1}}(T) \geq 2$ which contradicts hypothesis $\Delta(P) = \Delta(T)$.

[**case** $(iv)$] Lemma 8.C.1.(8.4) implies:

$$\mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k] = \mathsf{C}^k(T_{i+1}),\ k \geq 2$$

It results that $\Delta_{T_i,T_{i+1}}(T) \geq 2$ which contradicts hypothesis $\Delta(P) = \Delta(T)$.

Therefore we must have $\pi_0\pi_1 \not\to \bot$. We can thus conclude using the $(\mathsf{comp})$ rule:

$$\frac{P' \hookrightarrow_{\pi_0} \mathbb{C}^i[T_1, \ldots, T_i] \quad Q \hookrightarrow_{\pi_1} \mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k] \quad \pi_0\pi_1 \not\to \bot}{P \hookrightarrow_{\pi_0\pi_1} \mathbb{C}^i[T_1, \ldots, T_i], \mathbb{C}^{k-(i+1)}[T_{i+1}, \ldots, T_k]}$$

□

□