Domain-Specific Modelling Languages in Bigraphs

Gian Perrone

IT University of Copenhagen

Copenhagen, Denmark

December 6, 2012

Abstract

Modelling is a ubiquitous activity in human endeavours, and the construction of informatic models of many kinds is the key to understanding and managing the complexity of an increasingly computational world. We advocate the use of domain-specific modelling languages, instantiated within a "tower" of models, in order to improve the utility of the models we build, and to ease the process of model construction by moving the languages we use to express such models closer to their respective domains.

This thesis is concerned with the study of bigraphical reactive systems as a host for domain-specific modelling languages. We present a number of novel technical developments, including a new complete meta-calculus presentation of bigraphical reactive systems, an abstract machine that instantiates to an abstract machine for any instance calculi, and a mechanism for defining declaratively sorting predicates that always give rise to well-behaved sortings. We explore bigraphical refinement relations that permit formalisation of the relationship between different languages instantiated as bigraphical reactive systems. We detail a prototype verification tool for instance calculi, and provide a tractable heuristic for deciding reaction rule causation. Finally, we provide a mechanism for the modular construction of domain-specific modelling languages as bigraphical reactive systems, exploring the relationship between vertical refinement and language specialisation in this setting.

The thesis is composed of several publications, augmented with new introductory and background material.

Preface

I joined the IT University of Copenhagen under the auspices of the *Jingling Genies* project not longer after the completion of the Bigraphical Programming Languages (BPL) project at ITU. The BPL project had made significant progress in advancing the theory of bigraphs, although it was clear that there was still much to be done. Jingling Genies was a project focusing on *context-aware services*, and as a result, was an appropriate vehicle to continue the investigation of bigraphs. Although the vision of a bigraphical programming language had not emerged as a reality during the BPL project, it became clear during the course of the Genie project that we could advance the state of the art in the construction of domain-specific modelling languages that specialise to a given domain and therefore provide better modelling primitives.

During my time at ITU, I published the following papers, some of which appear in this thesis, or which form the basis for specific chapters:

- Wusheng Wang, Gian Perrone, and Thomas Hildebrandt. Petri Nets in Bigraphs Revisited. *Nordic Workshop on Programming Theory (NWPT 2012)*, 2012 (to appear).
- Alexander Faithfull, Gian Perrone, and Thomas Hildebrandt. Big Red: A Development Environment for Bigraphs. *Fourth International Workshop on Graph Computation Models (GCM 2012)*, 2012.
- Marco Carbone, Thomas Hildebrandt, Gian Perrone, and Andrzej Wasowski. Refinement for Transition Systems with Responses. *Fourth International Workshop on Foundations of Interface Technologies (FIT 2012)*, 2012.
- Gian Perrone, Søren Debois, and Thomas Hildebrandt. A Model Checker for Bigraphs. *ACM Symposium on Applied Computing — Software Verification and Tools Track (ACM SAC-SVT 2012)*, 2012.
- Gian Perrone, Søren Debois, and Thomas Hildebrandt. Bigraphical Refinement. *Refine '11: Proceedings of the 2011 International Refinement Workshop*, 2011.

This thesis represents the culmination of my work (and my work with co-authors) on domain-specific modelling languages in bigraphs.

Acknowledgements

It takes a village to raise a PhD thesis, and indeed it would take an entire forest of paper to properly thank everyone who deserves to be. A few individuals require special mention, however.

To my supervisor Thomas Hildebrandt — you kept me on the path, steered me clear of any dangers, and tolerated my — *ahem* — occasional tardiness with characteristic good grace and humour, which is all one can really ask for in a supervisor.

Søren Debois — I cannot thank you enough for how generously you gave of your time, your ideas, and your efforts. Yours was a constant reassuring presence whenever I didn't understand, because I could count on your patient willingness to teach and help.

I also benefited from being in a stimulating, supportive environment at ITU, and I thank everyone who made it so. Lars Birkedal, Marco Carbone, Carsten Schürmann, and Andrzej Wasowski deserve special mention for the help they gave me.

I'm grateful to all the friends who helped me stay sane: Aaron, Alastair, Alec, Alyona, Anthony, Christina, Davide, Elena, Espen, Filip, Fiona, Francesco, Hugo, Jarmo, Jasmine, Josu, Maxime, Nick, Paolo B., Paolo T., Perry, Petra, Rob, Rosalba, Shane, Shaun, Taus, and Ziyad.

The members of the Genie project taught me a lot about doing collaborative research in an international way, and were very kind in helping me during visits to Beijing: Wang Wusheng, John Paulin Hansen, Jørgen Staunstrup, Arne Glenstrup, Ruth Sørensen, Yu Lian, Li Weiping, Lin Huiping, Mo Tong, Chu Weijie, Zhichao Zhang, and Nhi Quyen Le.

I was lucky enough to be hosted as a visitor at many fine institutions, and I'm grateful to the academics and their groups that took the time to discuss research and host me: Steve Reeves, David Streader, Sarah Mount, Michael Goldsmith, Matthew Hennessy, Kenneth Holmqvist, Marino Miculan, and Emilio Tuosto.

Finally, to my family: Rocky, Vye, John, Bella, and Andy. Thanks.

Contents

List of Figures

Chapter 1

Introduction

We shall therefore assume that explanation is the principal relationship—with many manifestations—that cements the tower of models that we call informatics.

Robin Milner (1934–2010)

Since before the invention of the digital computer there has been an interest in things that are computational in their nature — systems, processes, and artifacts that may be understood, described, and analysed in terms of fixed rules. The term *computer science* has become loaded with connotation, so we prefer the term *informatics* which encompasses the analysis, description, and application of systems that are somehow computational in nature. Robin Milner [84] observed that what is often called "computation" is really calculation, and therefore he used the term informatics in this context to mean something wider — encompassing the interaction, communication, and behaviour of systems.

There has always existed within the study of informatics a tension between the desire to understand complexity — the remit of the scientist — and the need to build, being the preserve of the engineers. However, central to both activities is the need to represent precisely artifacts and processes from the world around us, which is the process of *modelling*. When attempting to understand inherent complexity, we construct models as more comprehensible simplifications of parts of the real world. The act of construction begins with some idealised, abstract notion (if only in the mind

of the creator) which evolves through implementation to become something tangible.

This thesis is concerned with modelling in two respects; in providing improved modelling capabilities, and in studying the foundations upon which such modelling capabilities rest. In aid of the latter aim, this thesis includes contributions that are purely theoretical in their nature — it advances the state of the art not in the actual practice of modelling, but rather in our attempts to better understand the models that we create. In advancing the former aim, this thesis includes practical aspects that advance the state of the art in terms of the systems that we can hope to usefully model, and the quality and utility of these models.

Modelling as an activity is pervasive within all human endeavours. Disciplines such as physics, economics, chemistry, and biology largely consist of attempts to construct models that predict physical phenomena, be they as systems of equations, statistical analyses, diagrams, or simple human intuitions. In every case, attempts to abstract away from the concrete to more general principles that might be understood more completely are fundamentally modelling activities.

Of course, within this thesis we are primarily concerned with *informatic* models — i.e., models that attempt to capture knowledge of the form and function of information, and the way that information may interact and change over time. Confined further within this subset are those models that are strictly *computational* in their nature — i.e., those that proceed by fixed sets of rules. This space includes facets that are well-studied and well-understood; it encompasses much of modern mathematics and formal logic, for example. Less-understood, but commonly-practised is the construction of software, which is in many respects a kind of modelling activity. However, as the practice and nature of computing has changed, progression of the tools by which we might hope to understand or describe these systems has seldom accompanied this technological progress.

As we proceed, we will discuss (and see examples of) many different types of models of systems that may involve or enable the construction of software or computational artifacts. However, modelling is a much more general activity, and while software is an important application of efforts to improve the state of the art in modelling, there is a myriad of other systems that have benefited, or will benefit, from a modelling approach.

1.1 Explanation

The sense in which one model may relate to some other entity — be it another model, a physical object, or a concept — seems to be by *explanation*, i.e., if A is a model of B, then it is because A in some sense *explains* B. Robin Milner, as a pioneer of much of this school of thought (and one of its most prolific proponents) is quoted directly:

> [We] say that a model comprises A family of *entities*; and What these entities *mean* – Robin Milner, "A Tower of Informatic Models" [84]

and with respect to the relationships between these entities:

> The phrase 'model of...' needs discussion. The 'of' relationship, between a model and the reality it explains, is central to the whole of science; the same relationship holds in any engineering discipline between a model and the reality it explains. Just as we say that Newton's laws explain the movement of bodies with mass, so we can say in informatics that a model consisting of programs and their meaning explains the reality of computers and what their screens display. – Robin Milner, "A Tower of Informatic Models" [84]

1.2 Space and Motion

In attempting to analyse and better understand systems, we can look to the practice of computing to guide our explorations. The nature and substance of computing as an activity is changing dramatically, spurred by the development of the personal computer, ubiquitous internet access, mobile internet-connected devices, and the integration of computational elements in every facet of daily life. Our very conception of what constitutes "computing" is changing — informatic endeavours are so pervasive that they can hardly be considered a separate activity anymore, as ingrained as they are in entertainment, communication, business, education, and production.

One of the most startling changes that these innovations in the practice of computing has given rise to is the increasing importance of *space* and *motion*. The title of Milner's book *The Space and Motion of Communicating Agents* [83] gives a sense of the primacy of these concepts in the emerging future directions of the study of informatics. Where computation

and calculation were once inseparably linked, computation may now take place across many different computing systems, in many contexts, and engage in countless interactions with both human and computational agents. Advances in systems biology have recognised the value of a computational view of mechanisms inside cells [67], and have started a process whereby informatic models can enrich the natural sciences, as was Milner's aspiration [84].

While informatic models carry a connotation of computation, we can equally apply these processes to understanding, describing, and monitoring processes that only involve human actors. For example, *business process modelling* is now well-established [101] as an activity in many different industries. The value of systematic descriptions of human interactions in discovering where challenges or risks exist within an organisation has been amply demonstrated through years of practice [101].

The primary attraction of considering the space and motion of agents is that it involves (like all modelling tasks) constructing useful *abstractions*. Through the choices we make when building these abstractions — particularly by what we include and what we exclude — we gain a different perspective on a system, which might make it amenable to different (automatic or human) analysis. Space and motion are common to almost all systems, and so considering these as primary is the start of an approach to modelling that aspires to generality across problem domains.

1.3 Domain-Specificity

There exist many ways of constructing informatic models, both as theoretical constructions, or with robust tool support in an applied setting. A common feature of modelling languages or approaches is an attempt to be general. However, models are rarely general in this respect; they capture a set of assumptions and factors with respect to some *problem domain*. A domain might encompass an entire industry (e.g., aerospace or healthcare), or might only include a single model (e.g., a model of a particular software system). There exists something of a disconnect between the assumptions inherent in defining a model, and the general-purpose nature of many modelling languages. This mismatch has given rise to the idea of *domain-specific modelling languages* (DSMLs), where a language for modelling is constructed with a particular modelling task or problem domain in mind.

Why introduce domain-specificity in modelling? Given the existence of many general-purpose languages both for programming and for modelling, one might be tempted to employ these for all tasks. The semantics (and to a lesser degree, the syntax) of such general-purpose languages do not necessarily lend themselves to concise, accurate descriptions of the properties of interest in all domains. The reasoning is simple: A domain-specific approach leads to *better* models that capture the properties of interest, without encoding extraneous detail.

The construction of domain-specific modelling languages can be an arduous task. Even when considering a language based upon another, there is a particular attention to detail required in defining syntax and semantics of languages. Similarly, what can we do with these languages? Tool support for editing such models, analysis, and even communicating an understanding of the model to other humans requires vast effort that is rarely justified in all but the largest projects.

One means to reduce the effort required to construct and employ a domain-specific modelling language is the use of a *meta-modelling* environment[1]. One of the most credible efforts to construct a general-purpose meta-modelling environment that could function as a host for domain-specific modelling languages is *bigraphs* [83]. By constructing a domain-specific modelling language within a meta-modelling environment, one gains the benefit of reusability. Results shown to be true of a suitable meta-model are immediately true of all modelling languages instantiated within it. Similarly, tools that can operate on meta-models may be instantiated as tools for instance modelling languages. Finally, in terms of comprehensibility, the benefit of a meta-model is that it eases communication. The semantics of a domain-specific modelling language encoded as (for example) bigraphs are in some sense "on show" — they are tangible artifacts that may be exchanged, inspected, and discussed. Anyone familiar with bigraphs in general can hope to understand a particular instance modelling language or model constructed within a language simply by his or her a priori knowledge of the meta-model. Bigraphs are a promising candidate in all these respects, and will be the focus of much of this thesis. We give a full introduction to the development and technical aspects of bigraphs in Chapter 2.

[1]The term "meta-model" is often used in some communities to mean something akin to a syntactic schema or grammar. This is not sense in which we mean "meta-model", which following Milner [84] we consider to be a model that in some sense "explains" the (behavioural) meaning of another model

1.4 A Tower of Models

The importance of relating models stems from a desire to really *use* these models in practice, either for implementation, or in order to make predictions about some underlying system. If the "model of" relationship means "explains" or "approximates" or "describes", then the exact nature of this relationship will determine for what this model may be used. Similarly, it often makes sense to relate not just models to entities of the real world, but also to relate one model to another (where these models might be described in different domain-specific modelling languages, for example). This gives rise to the notion of a "tower of models", as proposed by Milner [84]. The metaphor extends to encompass the approximate dimensions of the tower — many layers of models, each explaining the one immediately below it gives a tower that is tall and thin, whereas having many parallel models of the same system gives a tower that is short and broad. Moving from the base of the tower upwards somehow captures increasing degrees of abstraction, while moving down from the top of the tower gives additional (concrete) detail. What lies at the extrema? At the bottom of the tower might be an executable implementation of some system, or simply the real-world entity that is of interest. At the very top might be a completely abstract (partial) specification of some key behaviours. At every layer in between is any number of informatic models.

The vision of many different models explaining the same system is not a new one — the practice of *stepwise refinement* has been common in software engineering for many years [104], in which an abstract specification is refined through many steps towards an eventual implementation.

We can even consider more pedestrian software development activities in the context of a tower of models. The use of UML (Unified Modelling Language) [96] is common in many software development methodologies, and the construction of many parallel views of the same underlying system using different modelling mechanisms is common. This does not quite satisfy our definition, as the explanatory power of these models extends only to *syntax*, ignoring most considerations of *semantics*. Where semantic interpretations do exist, there is a lack of a meaningful meta-model that explains the semantics of lower-level semantic models, and therefore our tower is distinctly lop-sided.

So to properly realise the vision of a tower of models, we require explanatory power for both the syntax and semantics of models. We will argue that bigraphical reactive systems are therefore an appropriate host for towers of models, and that the individual models are best described in terms

of domain-specific modelling languages (specified in terms of bigraphical reactive systems). One technical contribution provided within this thesis (in Chapter 4) is a mechanism for the construction of *vertical refinement mappings* between bigraphical reactive systems — i.e., formal mappings that relate different domain-specific modelling languages to one another. In Chapter 5, we provide a more tractable means of analysing models constructed in this manner. Finally, in Chapter 6 we provide a means by which to systematically construct such languages. Of these contributions we will say more later; for now it is sufficient to say that the unifying theme of this thesis is enabling the construction of towers of (domain-specific) modelling languages as bigraphical reactive systems.

1.5 Ubiquitous Computing

Of much interest in recent years is the promise of *ubiquitous computing* [103]. Ubiquitous (or *pervasive*) computing has the stated intent of subsuming the traditional desktop computing model by integrating computing into the fabric of everyday life. Such systems are meant to be so tightly integrated and seamless as to "disappear" into the background. Such varied and diverse computing environments offer new challenges to those who might hope to describe, analyse, and understand such systems. Ubiquitous computing seems well-suited to a domain-specific modelling approach, in that any given (possibly ad hoc) configuration of computing elements constitutes a different domain, and applications written over this structure need to be able to describe both computation as well as high-level reconfiguration of the system [103].

The approaches that we propose in this thesis have already been applied to problems from the ubiquitous computing domain, under the guise of *context-aware systems* [97]. Birkedal et al. [20] explored the possibility of encoding context-aware computing in terms of bigraphical reactive systems, with some success. However, they discovered certain limitations at that time, some of which we hope to address in the present work.

1.6 Problem Formulation

This thesis is primarily concerned with the investigation of domain-specific modelling approaches within bigraphical reactive systems, and attempts to answer the question: "How can we move modelling approaches closer to

their respective domains?", with the side-condition that we are restricting our attention mostly to the techniques available within bigraphical reactive systems. Why restrict ourselves in such a manner? The answer is found mostly in the previous work on bigraphical reactive systems, in which they have demonstrated excellent promise in acting as a host for domain-specific modelling languages, as well as generalising a number of other approaches. Given the entire design space of approaches to the instantiation of domain-specific modelling languages, bigraphical reactive systems are almost uniquely promising in this regard. We review bigraphical reactive systems and enumerate much of the previous work in this area in Chapter 2.

1.7 The State of the Art

Having restricted ourselves primarily to bigraphical reactive systems, it remains interesting and useful to review other current approaches to the same problem, which we attempt here. We include here only high-level approaches — related work for each discrete strand of work is included within the respective chapters that follow. We distinguish between two broad classes of mechanisms for describing DSMLs — those that take a primarily *syntax-directed* approach, in that the main intention is to describe the syntactic structure of models, and those that are additionally concerned with describing the *semantics* of DSMLs, such that the meaning of a given model may be fully understood with respect to the language. The former is more common, whereas the latter approach is really our primary focus.

1.7.1 Syntax-Directed Approaches

There exist a large number of approaches to the construction of *meta-models* that describe the syntactic structure of well-formed models. These may take the form of grammars for textual languages, graph grammars for graph-based models, or schemas for structured formats such as XML. The actual interpretation of these models is generally left implicit, or is outside the scope of the meta-model. For example, an entity-relationship model [29] describes the static structure of a database, in terms of the tables, columns, and data-formats that may be present in a well-formed instance of that model. The meta-model for entity-relationship models (i.e., the model that describes the structure of well-formed entity-relationship models) does not attempt to describe the connection between a given model and its eventual

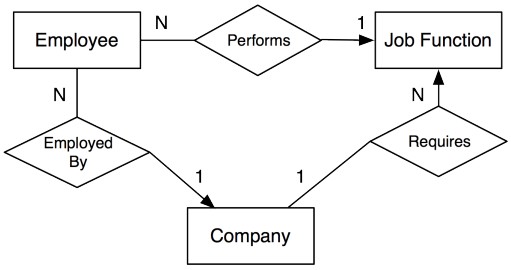


Figure 1.1: An example Entity-Relationship Diagram

behaviour, or indeed its interpretation with respect to the structure it is modelling. This interpretation is left informal and implicit, or in some cases is given as a secondary artifact that provides mappings from models to other formal objects (e.g., relational algebra terms).

Entity Relationship Diagrams

An entity-relationship diagram (e.g., Fig. 1.1) defines a model in terms of a (static) set of relations between entities. The interpretation of these models is external to the meta-model, in that there is no sense in which the model captures how a model evolves over time, nor the dynamic behaviour that manifests. While these kinds of meta-models do define a family of models, the formal interpretation of these models with respect to some underlying semantics (or indeed the relationship of one model to another) is considered an external, secondary concern.

Ontology languages

Ontology languages define a set of relationships over a set of objects of interest, and are therefore closely related to entity-relationship diagrams. The sense in which an ontology language is a "domain-specific" language is because the modeller defines a set of primitives and relationships, and these may be combined within individual models in order to describe a particular ontology. Ontology languages tend to be used to describe *static* models, and in this sense differ substantially from the kinds of domain-specific modelling languages that we propound. However, recent interest within ontology languages has focused on *modular* construction [31], and in this respect they are more similar to the bigraphical reactive systems DSMLs that we propose in Chapter 6, in that modular construction of DSMLs is one of the stated aims of that work.

XML Schema

XML schemas define a family of well-formed XML documents in terms of their structure. XML documents do not, however, capture anything about the dynamic semantics of documents — in the sense of capturing how they may evolve over time in response to the environment. XML documents lack any kind of dynamic semantics — the schema defines the syntax of well-formed documents, but the actual interpretation of these documents is left entirely unspecified, to be determined by the tools that will accept these documents as input. Hildebrandt et al. [56] proposed a connection between *Reactive XML* and bigraphical reactive systems, having observed that an XML document equipped with a set of reaction rules that selectively rewrite portions of documents correspond closely with a bigraphical reactive system. Consequently, an XML schema defines only well-formed documents, while additional structures (e.g., reaction rules) are required in order to imbue a given document with actual behaviour that we may reason about or use for analysis.

1.7.2 Semantics-Directed Approaches

In contrast to the various approaches to meta-modelling and domain-specific languages that focus solely upon describing the syntactic structure of well-formed models, there have been more recent attempts to give domain-specific modelling languages in a manner that also permits one to describe the semantics of the language. These approaches are considerably less common, largely owing to the additional complexity that it invokes. It is much easier to provide well-understood mechanisms for describing syntax (e.g., grammars or schemas), while leaving the interpretation of these syntactic constructs implicit, or merely defined by the tools that manipulate them. Our bigraphical approach to domain-specific modelling languages includes a uniform mechanism for describing both the syntax and the semantics of a modelling language, which constitutes an advance beyond that which is currently available.

Embeddings in General-Purpose Languages

One common mechanism for providing a domain-specific modelling language is to embed it within a general-purpose programming language. Languages such as Lisp, Scala, ML, and Haskell have all been used as domain-specific modelling language hosts. The means of defining such a language is

to design a set of combinators that build up data-structures that represent the model as a value within the language, relying upon the syntactic conventions of the language to permit models to be expressed. These models may be imbued with a semantics by providing functions that operate upon them, written in the same host language. The fact that this approach moves syntactic definitions and semantic interpretations closer to one another can only be seen as a positive step.

The challenge with such embeddings is that the semantics of the host language itself are not necessarily formal, written, or well-understood. If one wishes to investigate the formal interpretation of a given model, one cannot necessarily do so directly, as it will often require a deeper understanding of the host language than is available. Similarly, the mismatch between modelling languages (that attempt to describe or explain), versus programming languages (that specify computations) can lead to awkwardness, and an inability to directly express that which might otherwise be desirable. Anyone attempting systematic study of such embedded modelling languages is working with respect to the fixed semantics of some host language, and attempting to reconcile the semantics of a modelling language with these is not always straightforward (or indeed, always possible).

Such language embeddings also lack the compositional properties that we seek. The kinds of host languages that are employed rarely provide any easy mechanism for the modular construction of domain-specific languages, and the ad hoc construction methods that one might employ to facilitate modular construction are unlikely to be amenable to formal analysis at a semantic level.

Xtext

Xtext [41] is a language construction toolkit, designed to speed the implementation of parsers, analyses, and editors for various programming languages and modelling languages. It is integrated into the Eclipse ecosystem, and semantics are implemented by giving mappings to Java semantics. As a result, the Xtext approach is really a specialisation of other approaches to embedding modelling languages into general-purpose languages, although the particular implementation details are made more elegant by the framework. Similarly, because it relies on traditional specification techniques for grammars (which are not always amenable to composition), and the lack of any formal meta-theory that would permit us to relate different languages, Xtext falls short of the requirements for a tower of models.

Modular Meta-Theory

A recent approach to reuse in the programming languages community has been attempts to build programming language meta-theory is reusable and modular. Such approaches have generally focused upon the construction of reusable meta-theory within theorem provers, such that formalisations of languages such as MiniML may be achieved in a compositional manner [37]. The somewhat orthogonal goals of programming languages and modelling languages become significant here, in that we hold that the direct accessibility of the semantics of a modelling language are important. The kinds of reasoning that one might wish to perform concerning the semantics of a programming language are markedly different to those that one would like to perform upon modelling language semantics, particularly when considering specialisation to a domain.

The semantics of general-purpose programming languages tend to be more complex, but also more restricted — they are, after all, a model of computation in some general sense. A modelling language (particularly one specialised to a given domain) need not be a general model of computation, and therefore the semantics of the language are likely to be simpler, but also possibly less well-defined. It remains to be seen whether the techniques developed in [37] would transfer to a domain-specific modelling language setting, and whether the kinds of sophisticated meta-theory that would enable are necessary for such languages, nor whether these techniques would permit domain-experts to actually construct modelling languages, given the requirement that one can manually verify that the encoded semantics correspond to the intended semantics appropriate for that domain.

We argue that the ability to easily define and inspect the semantics of arbitrary modelling languages depends more directly upon the ability to construct domain objects and define their semantics by rewriting, and a sophisticated external meta-theory defined in a theorem proving environment such as Coq [18] seems well outside the reach of many domain experts (who are not necessarily experts in programming languages). Consequently, the techniques from programming language meta-theory do not appear directly reusable in a domain-specific modelling language setting, given the somewhat orthogonal goals.

1.8 Contributions

The contributions of this thesis may be grouped into a number of discrete areas that together advance the state of the art in the construction of bigraphical domain-specific modelling languages. We briefly summarise these areas:

- **Defining Languages:** In Chapter 3 and Chapter 6 we see new mechanisms for defining bigraphical languages in ways that promote reuse, and which make bigraphical reactive systems more generally accessible. In particular the presentation found in Chapter 3 obviates the need for domain-experts and those familiar with process calculi to work directly with the unfamiliar categorical structure of bigraphs, and provides a straightforward logic for defining sorting predicates that is guaranteed to give rise to well-behaved sortings. The modular construction mechanisms proposed in Chapter 6 enable a standardised means of communicating the definition of a bigraphical language, and to reuse components that may be specialised to a given domain.

- **Tool support:** In Chapter 3 and Chapter 5 we see advances in the ability to execute and use bigraphical domain-specific modelling languages. The introduction of a bigraphical abstract machine and a fairness condition for bigraphical reactive systems provides opportunities to bridge the gap between theory and applicability, while the result that reaction in sorted and unsorted categories considerably improves our ability to implement sortings within bigraphical tools. The prototype tool described in Chapter 5 represents a concrete contribution to the applicability of bigraphical reactive systems, as does the tractable heuristic for computing reaction rule interference.

- **Relating Languages:** The introduction of vertical refinement mechanisms for bigraphs, as well as sufficient conditions to make it easier to construct such refinements in Chapter 4, and the further development of this work in Chapter 6 helps to make real the vision of a tower of models. This work has already permitted others to begin relating different BRS models to one another, and therefore represents a contribution to the state of the art.

We will say more of the contributions of this thesis in Chapter 7, as well as detailing some of the further research questions which it has raised.

1.9 Structure of the Thesis

This thesis is composed of three chapters of new material, and four papers that have been published previously (or submitted for publication) elsewhere. As a result, there exist several strands of work within this thesis; however, they are all unified under the goal of enabling the construction of domain-specific modelling languages in bigraphs. Each of the chapters that consist of a stand-alone publication (Chapters 3, 4, 5, and 6) include new introductory material that attempts to give context and relevant background to the work, and relate it to the stated goals of this thesis.

The remainder of the thesis is structured as follows:

- Chapter 2 gives a general introduction to process calculi in general, and bigraphs specifically. This is designed to be a comprehensible introduction for a general computer science audience, and introduces key concepts from the bigraphs theory, as well as introducing much of the notation that will be used throughout the remaining technical sections of the thesis.
- Chapter 3 introduces a complete meta-calculus presentation of bigraphs, and gives a sorting logic that is guaranteed to give rise to deconstructable predicates of the form required to ensure that the sorted category has RPOs, following the work of Debois [35]. This chapter also introduces a bigraphical abstract machine (BAM) that characterises the semantics of bigraph reaction in a way that is amenable to further analysis, including a fairness condition for bigraphical reactive systems.
- Chapter 4 develops ideas around vertical refinement relations that permit mappings to be constructed between different domain-specific modelling languages. The vertical refinement mechanism we propose is related to that given by Reeves and Streader [94, 95], and is a first contribution to making real the promise of a "tower of models". This chapter is derived from a paper published in the 2011 International Refinement workshop (REFINE'11).
- Chapter 5 introduces a bigraph tool (called *BigMC*) that represents a concrete step towards making bigraphical reactive systems usable as a host for domain-specific modelling languages. This chapter is drawn primarily from a paper published in the ACM Symposium on Applied Computing 2012. A central contribution is a tractable heuristic for computing interference or causation between reaction rules.

- Chapter 6 introduces a mechanism for the modular construction of bigraphical reactive systems, further developing the concepts introduced in Chapters 3 and 4. This represents a systematic means by which one can consider composition of domain-specific modelling languages described as bigraphical reactive systems.
- Chapter 7 offers a summary of the contributions of the thesis, and identifies promising future research directions within this area.

Chapter 2

Bigraphs

This thesis is primarily concerned with the theory and application of bigraphs, and as it is composed of individual publications, each chapter presents bigraphical reactive systems in a manner appropriate to the result being communicated in that particular work. This chapter should be considered the canonical introduction to bigraphical reactive systems for the purposes of this work, in that it subsumes the much shorter introductions given in the individual chapters. I will assume no knowledge, as it is my sincere hope that this chapter may stand alone as a comprehensible introduction to bigraphs. In introducing the technical aspects of bigraphs, it seems only appropriate to describe the historical developments that led to their development. For the expert reader, this material will be familiar; however, my intent in presenting a detailed history of the origins of bigraphs is in part to answer the oft-asked question: "why use bigraphs?", which is a reasonable question given the perceived complexity of the theory. I hope to communicate that bigraphs are an elegant solution for a problem to which many solutions have been posited (and found to be wanting). The sources of information for the historical and technical aspects of this chapter are numerous, and in-text citations will be given where appropriate. However, the authoritative definition of bigraphs should be considered Robin Milner's papers and technical reports published with various collaborators, and his 2009 book *The Space and Motion of Communicating Agents* [83]. Any contributions that are my own will be clearly identified; however, most of this content is simply a distillation of the excellent resources I have been able to draw upon in the course of learning about bigraphs.

2.1 History

Systematic attempts to describe the behaviour of systems that are "computational" in their nature pre-date the invention of the digital computer by several decades. The development of models of computation mark the birth of computer science as a discipline distinct from mathematics, through the pioneering work of Gödel, Church, Turing, and others working on symbolic logics and the foundations of mathematics in the 1930s. Systems that attempt to describe *computation* (and most modern programming languages) stem from this tradition. However, the real benefit of *calculi* — i.e., formal systems that have fixed rules — was in their use as abstractions of other systems. The process of building abstractions that permit greater understanding has been the cornerstone of computer science since its inception.

The development of models of computation during these formative years focused primarily on *sequential* computation, where computation is described in terms of sequences of steps that are performed one after another. A particularly pervasive model of computation is *automata*, a class of abstract machines of which the *finite state machine* is perhaps the most ubiquitous. Around the time Turing was developing the Turing machine, Petri proposed *Petri nets* as a means of describing chemical reactions. Petri nets were later given an algebraic treatment in Petri's PhD thesis [93]. A Petri net is a model of *concurrent* computation, in that different steps can potentially be performed at the same time. The connection to chemical reactions is obvious — in the real world, many different chemical reactions could be taking place at once, rather than each reaction occurring sequentially. This is a key assumption underlying the development of bigraphs (and many of the other formalisms we will see in this chapter) — *concurrency is a good model of real systems*. Concurrency theory languished for some time, with renewed interest beginning in the 1970s with the development of the Actor model [3] in which the primary components of a system are *actors* that send and receive messages.

The use of *interrupts* on digital computers to give the appearance of multiple programs executing at once, along with the rise of *time-sharing* systems that could serve multiple users at once, meant that the practice of concurrency was evolving alongside the theoretical developments in the area. Concurrent programming idioms such as semaphores [38], locks, mutexes, and later processes and threads, gave rise to a host of complex concurrent systems, the behaviour of which was not always well-understood.

2.1.1 Process Calculi

A shift came with the work of Robin Milner between 1973 and 1980 in the development of the *Calculus of Communicating Systems* (CCS) [74]. As the name suggests, CCS is concerned with the *communication* behaviour of systems. By abstracting away actual computation, and instead focusing on the communication that takes place, we get a very different view of a system. There is a classical example used — the vending machine and the robot. If we describe a vending machine in terms of the inputs and outputs it can perform (e.g., it can accept input in terms of a coin being inserted and drink-selection buttons being pressed, and supplies output by dispensing a drink), then we have abstracted away from the circuits and software that make the vending machine function and have instead arrived at a more abstract concept of the functionality that the vending machine supplies.

The shift that focusing on communication behaviour induces is significant, yet subtle. A sequential program or machine that accepts input at the start of execution will effectively run until completion (or it might fail in the process) without further intervention from any external party. A concurrent *process* of the sort that we describe in CCS (such as our vending machine) will only engage in behaviour if there is some other party willing to engage in the corresponding communication behaviour. The vending machine on its own will never produce any output — it must instead be placed into some *environment* or *context* that supplies that corresponding behaviour. There needs to be a user to insert a coin and press a drink-selection button, and perhaps to receive the drink that is dispensed. Without an environment, the vending machine is stymied.

CCS provides a concrete syntax with which to describe concurrent processes, as well as the act of placing a given process into an environment:

$$P ::= 0 \mid x.P \mid \bar{x}.P \mid P + P' \mid P \mid P'$$

where 0 is the *nil* (or "inaction") process that has no behaviour, $x.P$ accepts an input on a *channel* named x, and then proceeds a P, $\bar{x}.P$ supplies output on the channel named x, $P + P'$ is the *choice* operator that behaves either as P or as P', and $P \mid P'$ is the *parallel composition* of two processes, such that they are permitted to run together and synchronise where inputs and outputs on channels agree. CCS offers a number of other constructs, but these are omitted for simplicity.

Using this syntax we can describe our vending machine model:

$$VM \stackrel{\text{def}}{=} \mathsf{coin}.(\mathsf{button1}.\overline{\mathsf{drink1}}.VM + \mathsf{button2}.\overline{\mathsf{drink2}}.VM)$$

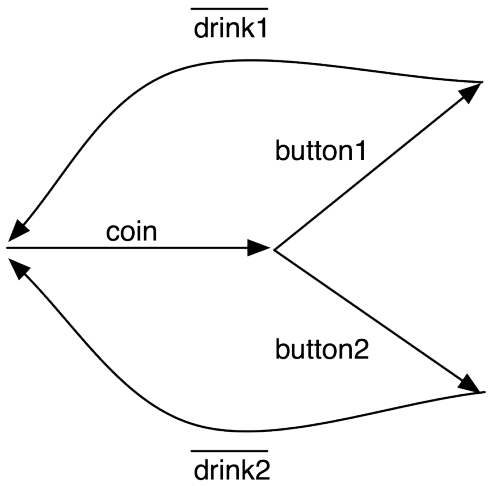


Figure 2.1: A labelled transition system for the CCS vending machine model VM

The process VM accepts input on a channel named coin (which we use to model the insertion of a coin into the machine), and then either accepts input on the channel button1, after which it will output on the channel drink1 and then behaves like VM again from the start, or it accepts input on channel button2, after which it outputs drink2 and then returns to the start.

The concept of *channels* is central to these kinds of process calculi. A channel is uniquely identified by its name, and provides instantaneous, guaranteed communication between the end-points of that channel. Extensions of CCS permit modelling channels with delayed or lossy transmission, but the basic abstraction proves sufficient for modelling many scenarios. It is also worth noting that the process communication is *synchronous*, such that the sender and receiver must agree on the next action. If a process wishes to send on channel x before there is another process willing to receive on channel x, then that process is blocked from further progress until such a time as there is a receiver willing to engage in the corresponding action.

We can visualise this process using a *labelled transition system* (or LTS), as in Fig. 2.1. In a labelled transition system, the transitions are labelled with *actions*, which in this case are the CCS input or output actions. The labelled transition system diagram is a useful aid for understanding the behaviour of these kinds of systems, but it is also a demonstration of the importance of good visual representations, to which we will return later.

We have defined our vending machine, but what of its environment? We

can design a robot that will interact with the vending machine:

$$Robot \stackrel{\text{def}}{=} \overline{\mathsf{coin}}.\overline{\mathsf{button2}}.\mathsf{drink2}$$

The *Robot* process will offer a coin and offer to press button2, and then accept drink2 as input. We can place the *Robot* in the same environment with the VM process using parallel composition:

$$VM \mid Robot$$

The semantic interpretation of a CCS process may be expressed as a kind of *term rewriting* in which we reduce a process by some fixed set of rules for rewriting the syntax of the terms. In the case of CCS, this is one of the rules:

$$x.P \mid \bar{x}.Q \rightarrow P \mid Q$$

This means that wherever we have two processes in parallel where the first action of one is to perform input on some channel x, and the other is offering output on that same x, that communication may be performed (and discarded), and we proceed with whatever work each process has remaining to do — i.e., P and Q respectively, in this case. This notion of giving the semantics of some formal system in terms of rewriting will become significant as we begin to explore bigraphs in the rest of this chapter.

2.1.2 Behavioural Theories

Underlying CCS (and the many other process calculi developed around the same time — e.g., Communicating Sequential Processes (CSP) [60] and the Algebra of Communicating Processes (ACP) [16]) is a robust *behavioural theory* that permits us to use the models we construct within the formalism to make predictions or guarantees about the behaviour of the system. In CCS, the labelled transition system is used to give meaning to the processes we construct (this is often referred to as a *labelled transition system semantics*). From the LTS, we can derive notions of *equivalence* between processes. One of the best-known equivalences is *bisimulation*, which is a binary relation on transition systems. One system is said to *simulate* another if for each action that one system can perform in a given state, the other system can take the same step. Two systems are *bisimilar* if each system can simulate the other. This gives rise to a notion of *observational equivalence*, such that no observer could tell the two systems apart by observing their behaviour. Another powerful equivalence is process *refinement*, which

was primarily developed in the context of CSP [25]. Refinement insists that one (more concrete) system exhibits a subset of the behaviour of another (more abstract) system. This gives rise to the process of *stepwise* refinement, in which an abstract specification is refined through multiple steps towards a more concrete implementation.

The desire to analyse process models has given rise to a host of tools and techniques for making concrete guarantees about the behaviour of processes. This has permitted their use as specifications for software systems, network and security protocols, and processes. The concurrent semantics present in hardware description languages like VHDL [87] and Verilog [100] have been construed within the setting of process calculi [13, 72]. Similarly, areas such as business process management and automation have made extensive use of process calculi in order to reason about processes. There has been an extensive proliferation of process calculi derived from CCS, CSP, and ACP, all tailored to specific domains.

2.1.3 π-calculus and Mobility

While CCS provided a strong foundation from which to consider the systematic study of communicating and concurrent systems, it has an inherent limitation. A given process is essentially *static* — its definition is fixed, and it communicates over a fixed set of channels. This is sufficient for modelling many kinds of systems, but it is not a good representation of other kinds of systems that have since become common. The primary mismatch stems from the fact that it is often useful and desirable to have a *dynamic* set of channels and permit processes to evolve (in terms of the network topology and the processes themselves moving) as they behave. This hints at a notion of *location* — where the "location" of a process is effectively given by the set of channels to which it has access. This also gives rise to a natural notion of *mobility*, such that the set of channels to which a channel has access (and therefore its location) should be able to change as the process behaves within an environment. This provides considerably more flexibility and expressivity in the models that we can describe. It was the desire to include mobility that motivated Milner, Parrow & Walker to propose the π-calculus [85], which is a calculus of *mobile* processes.

The main development results from the fact that π-calculus promotes channels to first-class values, such that names (of channels) can be communicated over channels. This allows the topology of the process to evolve and change as it evolves. π-calculus also includes *replication*, such that a process may dynamically spawn copies of some other process. The syntax

of a simplified variant of π-calculus is given as:

$$P \stackrel{\text{def}}{=} 0 \mid x(y).P \mid \bar{x}\langle y\rangle.P \mid P \mid P' \mid !P \mid (\nu x)P$$

where 0 is once again the *nil* process with no behaviour, $x(y).P$ accepts input of a value y on the channel x and then proceeds as P, $\bar{x}\langle y\rangle.P$ outputs the value y on the channel x and then proceeds as P, $P \mid P'$ is parallel composition, $!P$ creates a new copy of the process P, and $(\nu x)P$ creates a new *local name* (channel) x that is visible only within the process P and then proceeds as P.

We can demonstrate the additional flexibility of the π-calculus with a small example:

$$(\nu a)(\bar{b}\langle a\rangle.a(x).0) \mid b(c).\bar{c}\langle c\rangle.0$$

This process create a new (local) channel called a, and then outputs it on a global channel b. The second part of this process accepts input on channel b, and calls the communicated value c. It then uses this channel c which it just received to send the name c (i.e., it has a local reference c to the channel a). The first process then accepts input on a, and calls the communicated value in x.

There are many variants of π-calculus, including the Fusion calculus [89], *asynchronous* π-calculus [24], and many others. There are also a number of different types of bisimulation for π-calculus, each of which gives a different notion of equivalence and suited to different tasks. The π-calculus is the basis for much of the work on *session types* [45] (and *multi-party* session types [63]) in which the participating components of some kind of transaction (e.g., a travel agency system querying various airline systems to find the best-priced ticket) are assigned *types* that are effectively π-calculus terms, that are used to describe and analyse local system behaviours in terms of global behaviours, and vice-versa. Real-world systems such as those constructed in terms of *services* can benefit from the theoretical foundations of various tools for describing system behaviours in terms of calculi derived from the π-calculus (e.g., [86]).

2.1.4 Chemical Abstract Machine

The *Chemical Abstract Machine* (cham) was proposed by Berry & Boudol [17] based upon earlier work on using a *chemical solution* metaphor to describe concurrent systems that exhibit interact and communicate. Cham described the state of a system in terms of a solution of floating molecules

that could be reconfigured by *reaction rules*. Cham also permitted the formation of *membranes* that denoted some limited area in which reactions could occur, providing the first hints of notions of *spaces* in a concurrency setting. Augmented with various rules for stirring, heating, cooling, and cleaning the solutions (which correspond to various operations on terms), cham is able to represent the semantics of CCS, as well as several other calculi. The work on cham is significant for being the first of a new generation of concurrency formalisms that looked to provide appropriate metaphors for describing complex interactions, integrating notions such as reaction and locations.

2.1.5 Action Calculi

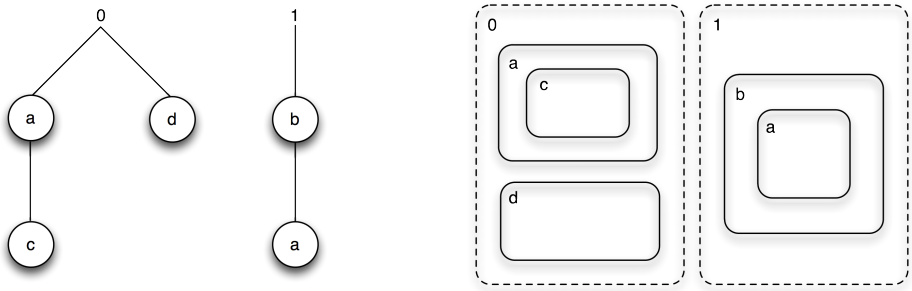
Action calculi represented another attempt to further generalise and unify models of interaction. Many of the ideas present in bigraphs were originally developed in the setting of action calculi, including the construction of terms from atoms, the visual syntax (action graphs) [76], rewriting semantics [99], and the original work on deriving bisimulation congruences [70]. Action calculi should be seen as a direct precursor to the development of bigraphs, in that bigraphs represent the extension and evolution of action calculi to encompass more general ideas, and the further development to allow solutions to the challenges presented by ubiquitous computing.

2.1.6 Mobile Ambients

Cardelli & Gordon proposed *mobile ambients* [28] in 1998 as a means of capturing mobility in concurrent systems, particularly with the emergence of the commercial internet, and optimism surrounding the possibilities for mobile agents and large-scale distributed computing in this setting. Ambients are bounded regions that can serve to capture various natural notions of containment — e.g., the contents of a room, individual files within a folder, a family tree, and so on. The dynamics of ambients are introduced through a small set of operations, expressed in the manner of a process calculus:

$$P \stackrel{\text{def}}{=} 0 \mid n[P] \mid in\ x.P \mid out\ x.P \mid open\ x.P \mid P \mid P'$$

where $n[P]$ is an ambient named n with contents P, $in\ x.P$ moves the ambient to move into the sibling ambient x and then proceed as P, $out\ x.P$ moves the ambient outside x and then proceeds as P, $open\ x.P$ removes an



(a) Normal tree diagram (b) Nested tree diagram

Figure 2.2: Place graphs rendered as both a normal tree diagram and using nesting to represent the parent-child relationships

ambient x, placing its contents as siblings, and $P \mid P'$ places two ambients side-by-side.

Mobile ambients are able to recover various other calculi — including the π-calculus — by simulating name-passing using operations on ambients. Having seen mobile ambients and the Chemical abstract machine, we are beginning to converge on the common elements that have led to the development of bigraphs.

2.2 Anatomy of a Bigraph

We have seen that through the historical development of process calculi, various common themes have emerged. It is worth noticing a few common elements that appear in all of the calculi we have seen thus far. They all include *names*, which might be used to identify channels or other elements (as in mobile ambients, for example). All of the calculi presented also include some kind of regular syntactic structure, and some of the calculi use this to create a notion of *locality*, like the membranes of cham, or ambients. Bigraphs are a distillation of this commonality. A bigraph represents orthogonal notions of *locality* and *connectivity* through the use of two separate graph structures. We will introduce these separately, and then show how they are combined to form a bigraph.

2.2.1 The Place Graph

The *place graph* consists of a forest of *trees*, each of which has a *root*. The place graph is used to represent notions of locality or containment. Trees are usually drawn as in Fig. 2.2a, with the root at the top, and the leaf nodes at the bottom. Within bigraphs, the convention is to draw the trees of the place graph as nested nodes instead, as in Fig. 2.2b, where the place graph roots are distinguished from nodes by dashed outlines. Both of these figures represent the same graph; however, whereas parent-child relationships in the normal tree representation are given as (directed) edges between nodes, we use containment of one node within another to mean that the inner nodes are children of the outer (containing) node.

The roots of the place graph are sometimes referred to as *regions* — we will only refer to them as such. Regions are ordered with respect to one another, and are indexed $0 \ldots n$; however, the sibling nodes within the trees are *unordered* with respect to one another.

To nodes of the place graph we assign *controls* — these are identifiers for nodes drawn from a set that is called a *signature*, usually denoted as Σ. The signature for the place graph given in Fig. 2.2 is $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$.

The final type of object that may appear in place graph is a *hole* (sometimes referred to as a *site*), which are depicted in Fig. 2.3 as shaded boxes. Holes are indexed sequentially from $0, \ldots, m$, and no index may appear more than once. The meaning of a hole will become clearer as we begin to discuss composition of bigraphs. For the moment, the name "hole" is appropriately descriptive — in the sense that it is a place holder into which another place graph may be inserted.

The size and shape of the nodes we draw in the graphical presentation of place graphs is not significant. Sometimes nodes with different controls are drawn using particular shapes, but this is simply an aid to the reader, and has no particular significance. Also, because sibling nodes are unordered with respect to one another, their placement within a parent is not significant. Regions are ordered, and by convention are drawn horizontally from left to right (using dashed outlines to distinguish them from place graph nodes).

We introduce some terminology by which to refer to certain properties of a place graph. The number of regions is referred to as the *outer width* of the place graph, and the number of holes as its *inner width*. A place graph with outer width 1 is called *prime*, whereas a place graph with outer width 2 or greater is called *wide*.

The containment of a place graph can be used in many different ways

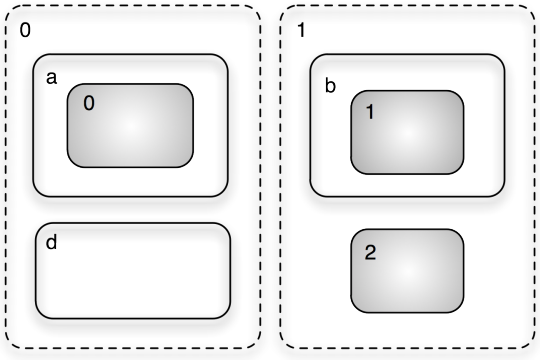


Figure 2.3: A place graph with inner width 3 and outer width 2

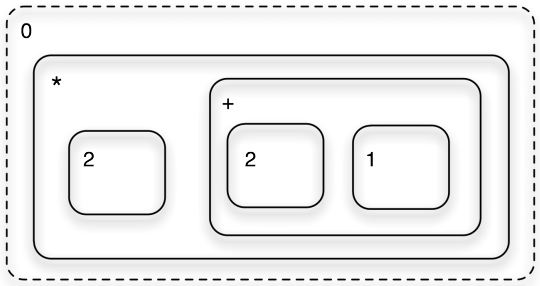


Figure 2.4: A prime place graph representing the abstract syntax of $2*(2+1)$

— one could use controls like Building, Floor, Room, and Person to describe the state of a multi-story building with many rooms, some of which contain people. The connection to the process calculi we have seen thus far might appear less obvious, but place graphs as presented here are well-suited to capturing a number of properties. The concept of an *abstract syntax tree* is familiar in programming languages, and indeed all of the calculi we have presented thus far are representable as abstract syntax trees. Given that a place graph consists of trees, we can encode an abstract syntax tree directly. Supposing we chose to use the controls 1, 2, $+$, and $*$, we could represent a term like $2 * (2 + 1)$ using the place graph given in Fig. 2.4. We will return to this connection later.

We arrive at a precise mathematical definition. Where m is the inner width and n is the outer width, a place graph is a 3-tuple:

Definition 1 (Place Graph)**.**

$$(V, prnt, ctrl) : m \to n$$

where:

- V is a set of nodes
- $prnt : m \uplus V \to V \uplus n$ is the *parent map* that records the parent of any place graph hole or node. We will use the notation $m \uplus V$ throughout to mean $\{0, \ldots, m\} \cup V$, where the two sets are assumed (or known) to be disjoint.
- $ctrl : V \to \Sigma$ assigns controls (from the signature Σ) to the nodes of the place graph.
- The notation $m \to n$ allows us to record the *interfaces* of the place graph, where m is referred to as the *inner interface* of the place graph and n is the *outer interface*.

We have two notions of composition for place graphs, which we will define in terms of our definition of a place graph.

Composition

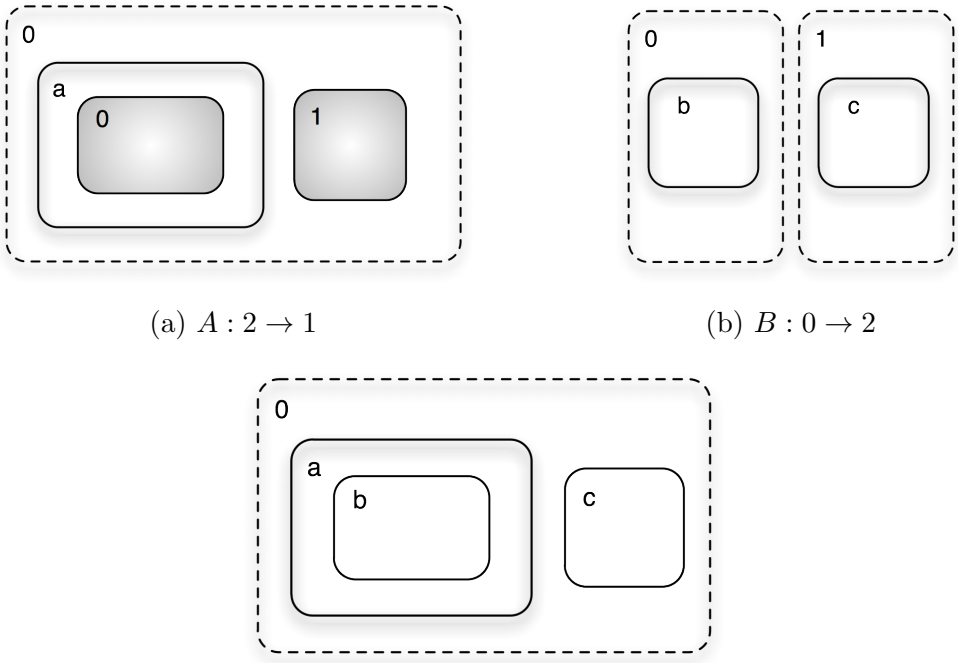
The first form of place graph composition is usually given as $A \circ B$, where A and B are place graphs. The intuition behind the kind of composition is that we take each of the regions of B, and place them into the holes of A, such that region 0 of B is inserted into hole 0 of A, region 1 of B is inserted into hole 1 of A, and so on. For this to work, the inner interface of A and the outer interface of B must agree — i.e., the inner width of A must be the same as the outer width of B, such that there are enough regions of B to fill all the holes of A. We demonstrate this kind of composition in Fig. 2.5. We can now define place graph composition formally:

Definition 2 (Place Graph Composition)**.** For $A = (V_A, prnt_A, ctrl_A) : m \to n$ and $B = (V_B, prnt_B, ctrl_B) : l \to m$,

$$A \circ B \stackrel{\text{def}}{=} (V_A \uplus V_B, prnt, ctrl_A \uplus ctrl_B) : l \to n$$

where for some place (hole or node) $p \in l \uplus V_A \uplus V_B$, the parent map is defined:

$$prnt(p) = \begin{cases} prnt_B(p) & \text{if } p \in l \uplus V_B \text{ and } prnt_B(p) \in V_B \\ prnt_A(j) & \text{if } p \in k \uplus V_B \text{ and } prnt_B(p) = j \in m \\ prnt_A(p) & \text{if } p \in V_A \end{cases}$$



(a) $A : 2 \to 1$

(b) $B : 0 \to 2$

(c) $A \circ B : 0 \to 1$

Figure 2.5: Place graph composition of $A \circ B$

The requirement for the more complex definition of *prnt* involves the need to update the parent map such that the immediate children of regions of B become children of the parents of the holes they are inserted into, and the regions and holes are removed altogether.

Tensor Product

While the insertion of one place graph into another might be considered the "normal" notion of composition, there is another useful way of combining place graphs that is the *tensor product*, which is denoted as $A \otimes B$. The tensor product is the juxtaposition of regions, as demonstrated in Fig. 2.6 — it effectively just involves placing the regions side-by-side — however, we arrive at a formal definition for tensor product on place graphs:

Definition 3 (Place Graph Tensor Product)**.** For disjoint place graphs $A = (V_A, prnt_A, ctrl_A) : j \to m$ and $B = (V_B, prnt_B, ctrl_B) : l \to n$,

$$A \otimes B \stackrel{\text{def}}{=} (V_A \uplus V_B, ctrl_A \uplus ctrl_B, prnt_A \uplus prnt'_B) : (j + l) \to (m + n)$$

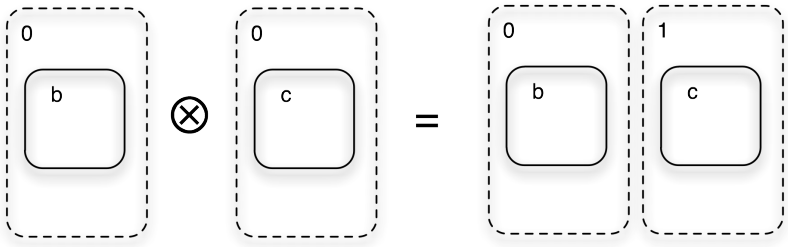


Figure 2.6: Demonstrating the tensor product $A \otimes B$

where $prnt'_B(j+i) = m+j$ wherever $prnt_B(i) = j$.

2.2.2 The Link Graph

What, then, of link graphs? We've seen that the place graph is capable of representing notions like containment and locality — now we see how the link graph is used to capture relationships and *connectivity*.

The *link graph* is a *hypergraph*, which means that each edge connects zero or more objects. The links are undirected, such that connecting A to B also connects B to A in a symmetrical manner. We also have a set of nodes, which are uniquely identified — we will use $v_0, \ldots, v_n$ to refer to them in this presentation. Each node has some number of *ports*. Ports serve as the connection points between the nodes and the links of the link graph.

In the same way that we had inner and outer interfaces for place graphs (respectively the number of holes and the number of regions in the place graph), we have inner and outer interfaces for link graphs, too. However, the link graph faces will be sets of *names*, which we call the *inner* and *outer* names, respectively. These will be the connection points at which one link graph may be joined to another by link graph composition.

Fig. 2.7 demonstrates a link graph over a set of nodes $V = \{v_0, v_1, v_2, v_3\}$. By convention, the inner interface $\{x, y\}$ of the link graph is drawn pointing downwards, and the outer interface $\{a, b\}$ is drawn facing upwards. The elements of the link graph are the ports (indicated by small circles on the boundaries of nodes), the edges (which are often not shown, but which we have drawn as small rectangles for clarity), the outer and inner names (drawn at the top and bottom of the diagram, respectively), and the *links* that are the lines connecting all of these elements together. Ports are indexed, and ordered with respect to one another, such that by convention we usually draw them left-to-right around a node boundary, such that the

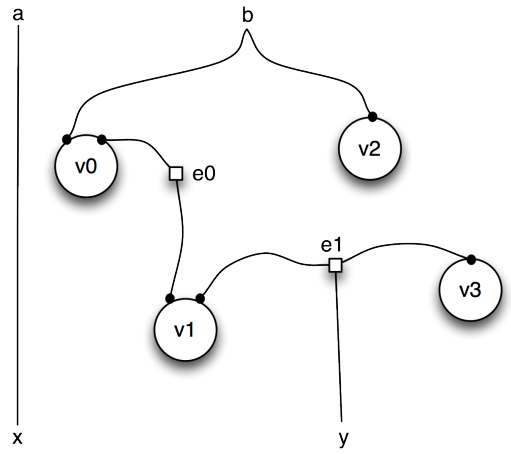


Figure 2.7: A link graph with interfaces $\{x, y\} \to \{a, b\}$

0^{th} port is the left-most one. Where this may be ambiguous owing to the node shape, we can write small indices next to the ports for clarity. Finally, it is worth noting that nodes in the link graph do not have any parent-child relationships — that is a property of the place graph — so the placement of nodes in the link graph is not significant.

We now give a formal definition of a link graph which captures all of these elements. We first extend the definition of a control to be both a name and an *arity*, which is a finite ordinal representing the number of ports of a node assigned that control. We use the notation $\mathsf{a} : 2$ to mean a control a with arity 2. We also define the function $ar(-)$ that gives the arity of a given control (though we will occasionally abuse this notation by applying it to nodes, the meaning of which is to return the arity of the control assigned to that node).

Definition 4 (Link Graph)**.**

$$(V, E, ctrl, link) : X \to Y$$

where V is a set of nodes, E is a set of edges, $ctrl : V \to \Sigma$ is a control map, $link : P \uplus X \to E \uplus Y$ is the *link map*, X is the set of inner names, and Y is the set of outer names. P is defined to be the set of ports, which is of the form $P = \{(v, i) : v \in V \wedge i \in ar(ctrl(v))\}$, i.e., a port is represented as pair consisting of a node (from V) and an index.

The link map is a function, such that each element of the domain may only appear once (i.e., map to one thing). A given inner name may only

connect to an edge or to an outer name. A port may connect to an edge or an outer name. Neither edges nor ports may not be connected directly to one another, and edges may not connect to outer names. This may seem restrictive, but this definition is sufficiently expressive for most applications, and it has the important property of preserving the compositionality of link graphs. We can return to Fig. 2.7, and give its definition as a link graph:

$$(\{v_0, v_1, v_2, v_3\}, \{e_0, e_1\}, link) : \{x, y\} \to \{a, b\}$$

where $link \stackrel{\text{def}}{=}$

$$\begin{aligned} x &\to a, \\ (v_0, 0) &\to b, \\ (v_0, 1) &\to e_0, \\ (v_1, 0) &\to e_0, \\ (v_1, 1) &\to e_1, \\ (v_2, 0) &\to b, \\ (v_3, 0) &\to e_1, \\ y &\to e_1 \end{aligned}$$

Link Graph Composition

As with the place graph, we can define the composition of link graphs. We follow Milner's definition [83]:

Definition 5 (Link Graph Composition)**.** For two link graphs

$$A = (V_A, E_A, ctrl_A, link_A) : X_A \to Y_A$$

and

$$B = (V_B, E_B, ctrl_B, link_B) : X_B \to Y_B$$

the composition is defined:

$$B \circ A \stackrel{\text{def}}{=} (V_A \uplus V_B, E_A \uplus E_B, ctrl_A \uplus ctrl_B, link') : X_A \to Y_B$$

where $link'$ is defined for some $p \in X_A \uplus P_A \uplus P_B$, where P_A and P_B are the ports of A and B, respectively:

$$link(p) \stackrel{\text{def}}{=} \begin{cases} link_A(p) & \text{if } p \in X_A \uplus P_A \text{ and } link_A(p) \in E_A \\ link_B(y) & \text{if } p \in X_A \uplus P_A \text{ and } link_A(p) = y \in Y_A \\ link_B(p) & \text{if } p \in P_B \end{cases}$$

Link Graph Tensor Product

As with the tensor product for place graphs, we can define the juxtaposition of disjoint link graphs as follows:

Definition 6 (Link Graph Tensor Product)**.** For disjoint link graphs $A = (V_A, E_A, ctrl_A, link_A) : X_A \to Y_A$ and $B = (V_B, E_B, ctrl_B, link_B) : X_B \to Y_B$,

$$A \otimes B \stackrel{\text{def}}{=} (V_A \uplus V_B, E_A \uplus E_B, ctrl_A \uplus ctrl_B, link_A \uplus link_B) : X_A \uplus X_B \to Y_A \uplus Y_B$$

2.3 Bigraphs

Having defined the link graph and the place graph independently, we can combine them to arrive at a definition of a bigraph.

Definition 7 (Bigraph)**.**

$$(V, E, ctrl, prnt, link) : \langle m, X \rangle \to \langle n, Y \rangle$$

where V is a set of nodes, E is a set of edges, $ctrl : V \to \Sigma$ is a control map, $prnt : m \uplus V \to V \uplus n$ is a parent map, $link : P \uplus X \to Y \uplus E$, and $P \stackrel{\text{def}}{=} \{(v, i) : v \in V \wedge i \in ar(ctrl(v))\}$. The inner interface of the bigraph is $\langle m, X \rangle$ where m is the inner width of the place graph and X is the set of inner names, while the outer interface of the bigraph is $\langle n, Y \rangle$, where n is the outer width of the place graph and Y is the set of outer names.

These definitions are almost identical to those given separately in the place and link graphs.

The graphical syntax for a bigraph follows from that introduced separately for the place and link graphs as well. We allow the links of the link graph to cross node and region boundaries. To avoid confusion we do not draw edges as we did in the link graph example; edges are only part of the link graph, and it is therefore somewhat deceptive to include them in a presentation that would make them appear to be part of the place graph structure.

We see a completed bigraph in Fig. 2.8 — hopefully it is possible to distinguish the place and link graph elements, and verify that the composition of these as we have described it yields a coherent bigraph structure.

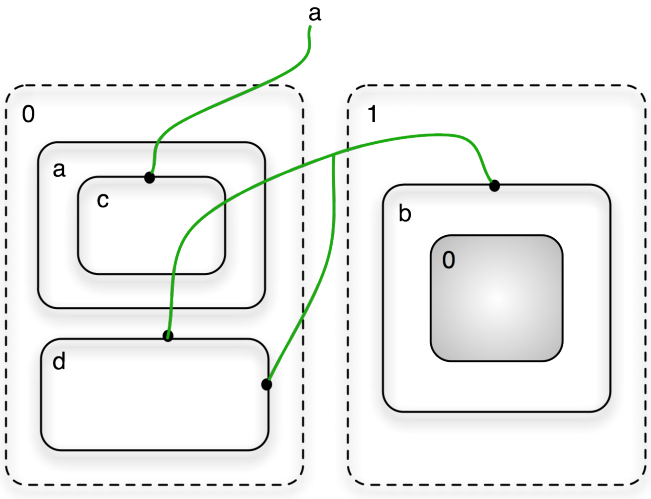


Figure 2.8: An example bigraph with interfaces $\langle 1, \emptyset \rangle \to \langle 2, \{a\} \rangle$

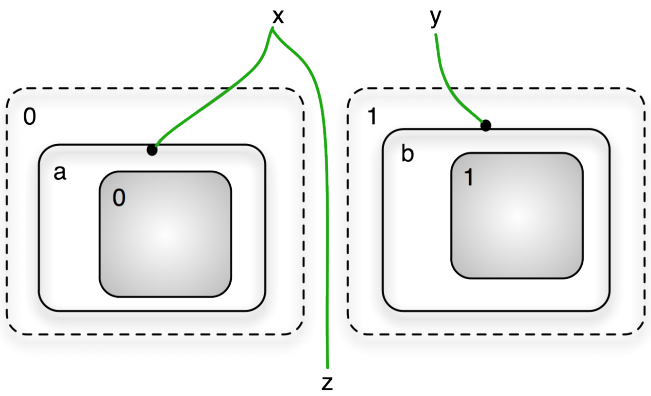


Figure 2.9: A bigraph $B : \langle 2, \{z\} \rangle \to \langle 2, \{x, y\} \rangle$

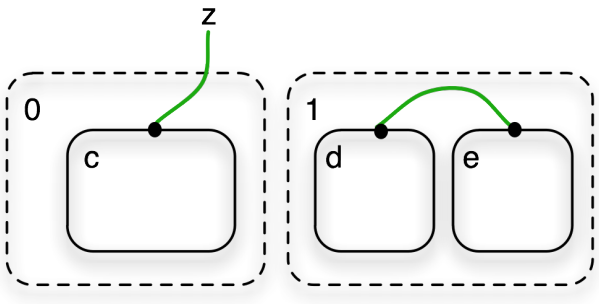


Figure 2.10: A bigraph $A : \langle 0, \emptyset \rangle \to \langle 2, \{z\} \rangle$

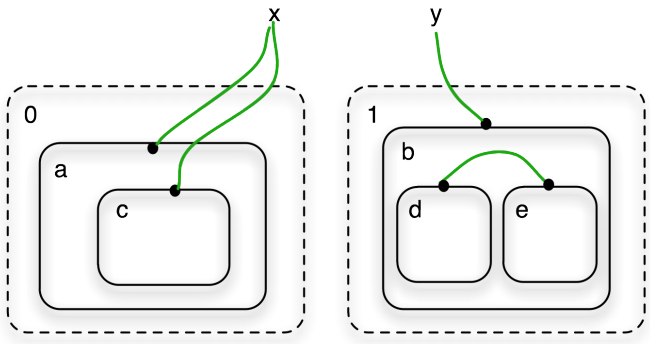


Figure 2.11: Bigraph composition $B \circ A : \langle 0, \emptyset \rangle \to \langle 2, \{x, y\} \rangle$

2.3.1 Bigraph Composition

The composition of two bigraphs is precisely defined in terms of the separate link and place graph compositions as already presented, with the same requirement that the inner and outer interfaces agree. We see an example of bigraph composition in Fig. 2.11, in which the bigraph A (Fig. 2.10) is inserted in to the context B (Fig. 2.9). We now proceed to a formal definition:

Definition 8 (Bigraph Composition)**.** For bigraphs $A = (P_A, L_A) : \langle m_A, X_A \rangle \to \langle n_A, Y_A \rangle$ and $B = (P_B, L_B) : \langle m_B, X_B \rangle \to \langle n_B, Y_B \rangle$:

$$B \circ A \stackrel{\text{def}}{=} (P_B \circ P_A, L_B \circ L_A) : \langle m_A, X_A \rangle \to \langle n_B, Y_B \rangle$$

We introduce some terminology here — where A and B are bigraphs, when composing $B \circ A$, we may refer to B as a *context* for A. An inner

interface $\langle 0, \emptyset \rangle$ (i.e., the inner interface of a bigraph with no holes and no inner names) is sometimes referred to as ϵ. A bigraph that has interfaces of the shape $\epsilon \to \langle n, Y \rangle$ is called *ground*, and is also sometimes referred to as a *process* or *agent*.

2.4 Matching and Reaction

We've now seen bigraphs in terms of their static structure; they could be mistaken for some slightly-exotic graph data structure. However, this is not the real power of bigraphs — the fact that the static structure is directly useful for representing various situations and systems in the real world is only a sign that the underlying representation is sufficiently general. The real power of bigraphs becomes apparent when we start to consider *dynamics* — i.e., how a given system evolves over time. In bigraphs terms we express dynamics as *reaction*, with sets of reaction rules that give possible ways in which a system might be reconfigured.

A reaction rule may be thought of as having the form $R \to R'$, where R is known as the *redex* and R' as the *reactum*. The intuitive definition of reaction is fairly straightforward: if we can find an instance of the redex somewhere in a bigraph B, then we may replace that redex with the reactum to obtain some new bigraph B'.

Taken together, a bigraph signature and a set of reaction rules is called a *bigraphical reactive system* (or *BRS*). A bigraphical reactive system should be thought of as a kind of language definition — the signature describes the syntax, and the reaction describe the semantics. We will make precise the nature of bigraphical reactive systems in Section 2.6.

We introduce some notation here: We use the notation $B \to B'$ to mean that for some bigraph B, there is a reaction possible that, when applied, would result in B'. Occasionally arrows will be labelled with the specific reaction rule that can be applied; however, this is non-standard. We also use $B \to^* B'$ to mean B' is reachable from B by the application of zero or more reactions in sequence, and $B \to^+ B'$ to mean a sequence of one or more reactions. The negation is used too — $B \not\to B'$ means that there is no single step of reaction leading to B' from B.

We have given an intuitive definition for bigraph reaction, but we wish to define the semantics of bigraph reaction precisely. The presence of an instance of a given redex inside a particular bigraph is determined by the process of *matching*. Formally, a match for a redex R within a bigraph G

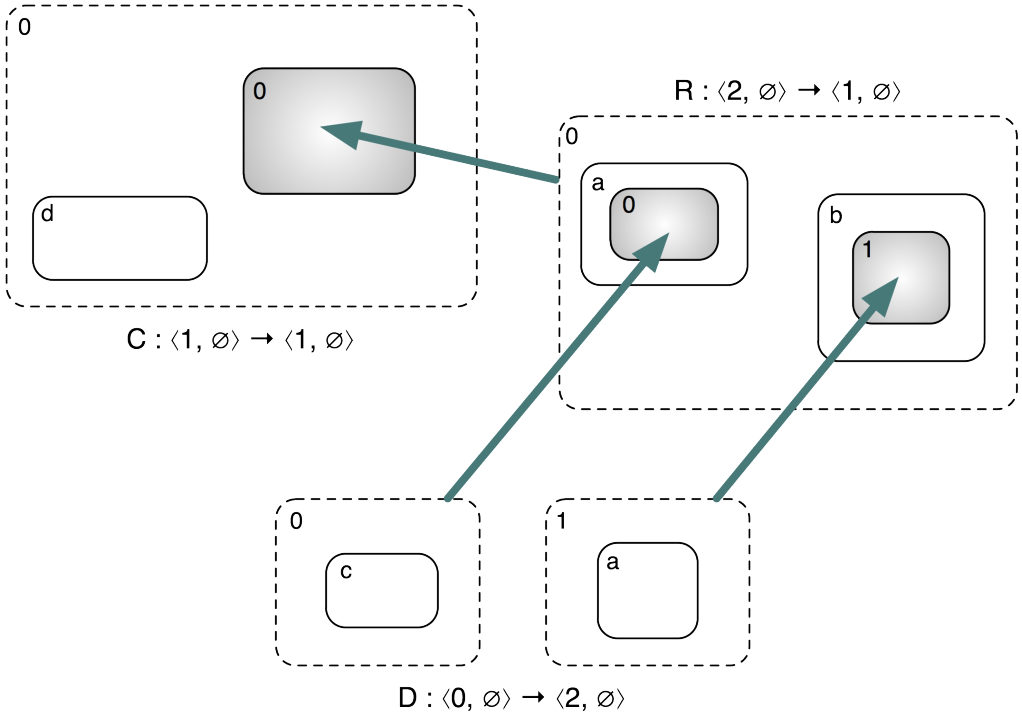


Figure 2.12: The decomposition of a bigraph G into a match for R, with context C and parameters D

is any solution to the equation:

$$G = C \circ (R \otimes id_I) \circ D \qquad \text{where C is } \textit{active}$$

Where C is a context, R is the redex, and D is a vector of parameters. The notion of (de-) composition used here is the normal bigraphical one. This characterisation of matching basically states that in order to find an instance of R in G, we just need to be able to "pull apart" G into some context that contains the match, and instance of the redex we are looking for, and some parameters that are found within the holes of the redex. The term $\otimes id_I$ for some interface I ensures that parameters are treated correctly, though we will often omit this from the definition of matching where it may be assumed. We illustrate this decomposition process in Fig. 2.12.

However, we so far ignored the side-condition "where C is active". The notion of *activity* is added as a mechanism by which to control exactly where reaction may occur — specifically, that reaction may only occur within active contexts. What makes a context active? To controls we add a

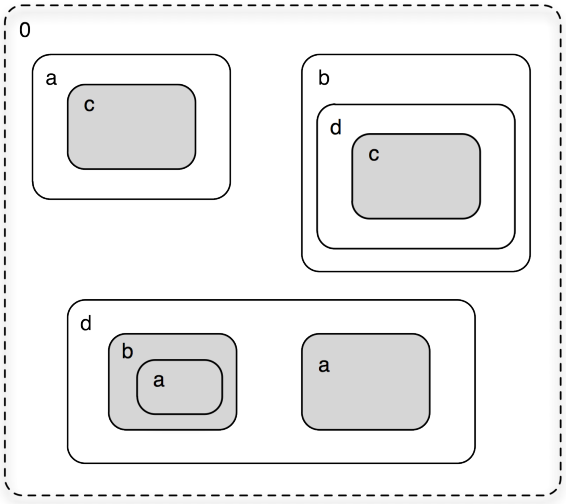


Figure 2.13: Where b is active, and a, c, and d are passive, the shaded regions indicate inactive locations where reaction cannot occur.

notion of activity, by labelling some controls *active* and others *passive*. To determine whether a given context is active, we must examine all the place-graph ancestors of a given hole, which must all be active. If every ancestor of a hole in the place graph is active, then that bigraph constitutes an active context. Regions are active by definition, so reaction may always take place at the top-level. We demonstrate activity in Fig. 2.13 by shading the place graph (this is just for illustrative purposes, and is not a part of the standard visual presentation of bigraphs) — where a and b are active and c is passive, only those unshaded place graph nodes could be potential locations for a reaction to occur. Reaction cannot occur within the shaded regions. These areas can participate in reactions by being part of a match that is rooted within an active context (or as parameters), but they may not be the root of a match. However, it is always possible that some reaction rule may lift these portions of the bigraph into an active context (the top-level, for example), so the fact that a given part of a bigraph cannot be the location of a reaction at one moment in time does not entail that it should remain so forever.

So it is possible to think of a match as finding an embedding of one bigraph expressing a pattern (the redex) into another (the agent). Following Højsgaard [61], we use the notation $R \hookrightarrow G$ to mean that we can find an embedding of R into G. The negation ($R \not\hookrightarrow G$) is also used.

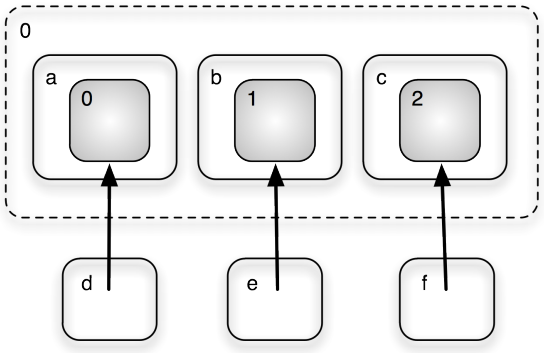


Figure 2.14: Matched parameters for a redex $R : \langle 3, \emptyset \rangle \to \langle 1, \emptyset \rangle$

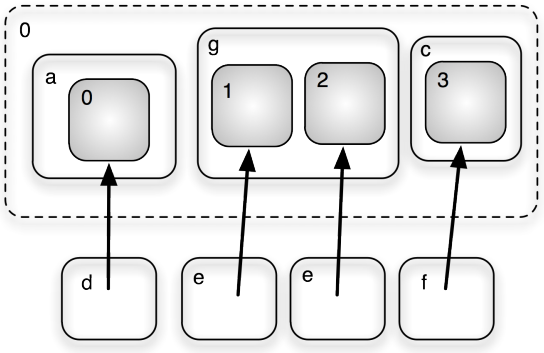


Figure 2.15: Instantiation of parameters for a reactum $R' : \langle 4, \emptyset \rangle \to \langle 1, \emptyset \rangle$

Matching (and embedding) can therefore be thought of as a precondition of reaction, as having successfully identified a match, we can use the decomposition to substitute the redex for the reactum, yielding a new agent. This leads us to a definition of reaction based upon matching:

Definition 9 (Reaction)**.** For any reaction rule $R \to R'$, and agent G:

$$G = C \circ (R \otimes id_I) \circ D \to C \circ (R' \otimes id_I) \circ D = G' \qquad \text{where } C \text{ is active}$$

Which is to say that having found a context C and vector of parameters D to place around the redex, we can replace the redex R with the reactum R', placing the parameters back into the reactum by composition in order to fill the holes, and placing the resulting reactum into the same (unaltered) context.

2.4.1 Parameter Instantiation

We have obscured an important detail here — namely, what of reactums that discard the contents of holes, or which create copies of holes? Bigraphical composition is not well-defined for such reactums, and therefore a reactum with four holes (duplicating one of them) cannot be a context for a vector of matched parameters such as $d_0 \otimes d_1 \otimes d_2$ through composition, as their inner and outer interfaces disagree. We remedy this with an *instantiation map*. The instantiation map is a transformation on a bigraph (the parameters) that ensures that the width of the parameters, and the order, is appropriate for the reactum. Given the redex in Fig. 2.14 and reactum in Fig. 2.15, we can see that there are three holes in the redex, and four in the reactum, as hole 1 is cloned in the reactum. Consequently, the instantiation map from holes of the reactum to holes of the redex for this rule is given as:

$$\eta \stackrel{\text{def}}{=} \{0 \to 0, 1 \to 1, 2 \to 1, 3 \to 2\}$$

If we consider the parameters of the redex $D = d_0 \otimes d_1 \otimes d_2 : \langle 0, \emptyset \rangle \to \langle 3, \emptyset \rangle$, then the application of the instantiation map yields $d_0 \otimes d_1 \otimes d_1 \otimes d_2 : \langle 0, \emptyset \rangle \to \langle 4, \emptyset \rangle$, which is of the appropriate shape to be inserted into the reactum. Consequently, we slightly alter the definition of reaction, and of reaction rules. A reaction rule is a 3-tuple (R, R', η), but we continue to write $R \to R'$ where the instantiation map is unambiguous (i.e., where it is just the identity map from holes of the redex to holes of the reactum). For the definition of reaction, we give:

Definition 10 (Reaction with instantiation)**.** For any reaction rule $R \to R'$, and agent G:

$$G = C \circ (R \otimes id_I) \circ D \to C \circ (R' \otimes id_I) \circ \eta(D) = G'$$

2.4.2 Terminology

It is important to notice that reaction rules are themselves bigraphs — namely, the redex and reactum are both bigraphs. If the redex and reactum are ground (i.e., they have no holes or inner names), then we refer to the rule as a *ground reaction rule*. If the redex and reactum contain holes or inner names, we call it a *parametric reaction rule*. A rule with a prime redex and reactum is called a *prime reaction rule*, while a rule with a wide redex and reactum is called a *wide reaction rule*.

2.4.3 Structural Constraints

For a reaction rule to be considered well-formed, it must obey a number of properties. Most of these properties follow immediately from the definition of reaction, but we restate them here directly to avoid confusion. When considering reaction only in the setting of ground agents, reaction rules that are well-formed obey the following properties:

- The redex and the reactum must have the same outer interface — i.e., the same outer width and set of outer names.
- The reactum cannot contain holes that are not also in the redex.
- The reactum cannot contain inner names that are not also in the redex.

2.4.4 Wide Matching

The existence of wide reaction rules introduces several interesting considerations. What, exactly, is the difference between the matching of the prime redex in Fig. 2.16 and the wide redex in Fig. 2.17? The difference lies in the insistence on finding both a and b as siblings (i.e., within the same parent node or region) in the first case, and simply finding an a-node and a b node separately *anywhere* in the bigraph (provided they are non-overlapping). We exemplify the difference in Fig. 2.18 and Fig. 2.19, with the wide and prime matches indicated. Wide rules of this kind can be used to construct rules that match patterns like "two people anywhere in the building, each connected to some name x", whereas a prime redex would express a property like "two people in the same room, each connected to some name x".

2.5 A Bigraph Term Representation

We have thus far seen two ways of representing a bigraph: as a 5-tuple with interfaces, or using a graphical notation. The 5-tuple description of a bigraph in terms of its constituent nodes, edges, control map, parent map, and link map, is precise, but not particularly human-friendly. The graphical notation is also precise, but it is not particularly compact, nor is it particularly machine-friendly (although there exists a graphical editor based upon Eclipse called Big Red [42] for the construction of bigraphical

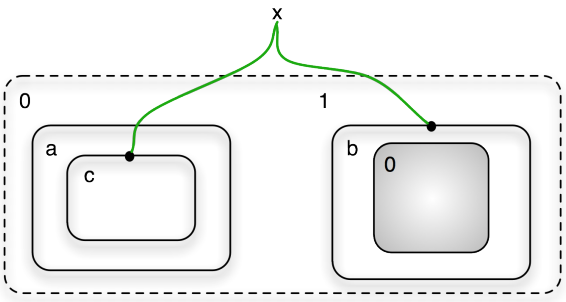


Figure 2.16: A prime redex

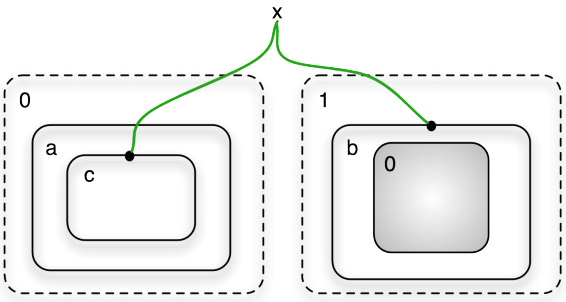


Figure 2.17: A wide redex

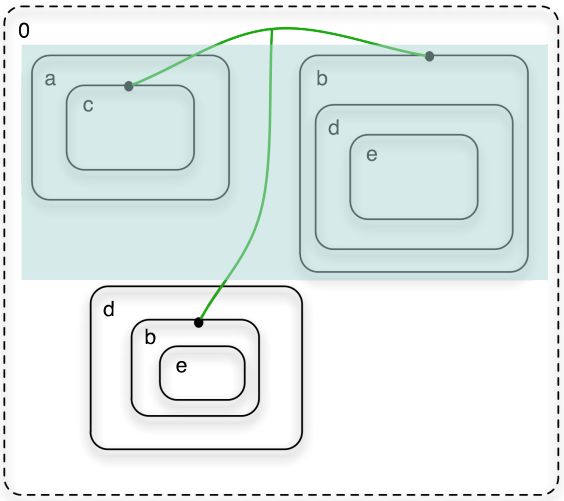


Figure 2.18: There is only one match for the prime redex given in Fig. 2.16 (shaded)

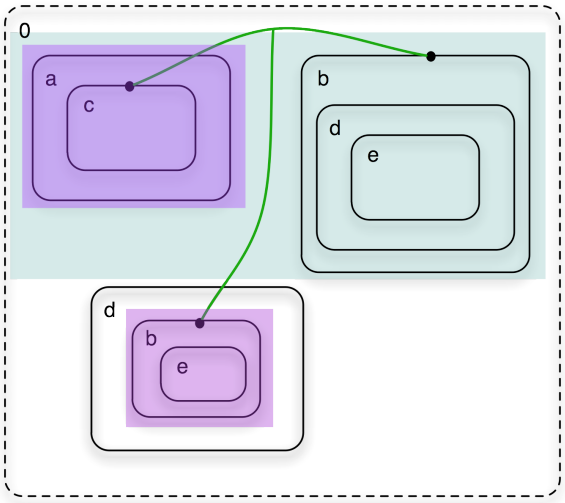


Figure 2.19: In addition to the match found for the prime redex, the wide redex given in Fig. 2.17 has an additional match in which the matched regions are not siblings (shaded)

reactive systems). Consequently, we present a third way to describe a bigraph — a textual *term representation* in the style of the process calculi we have already seen. This term representation is not complete (i.e., there are bigraphs that are not expressible in this calculus), though a complete such term representation is given in Chapter 6. The intention here is to give the general flavour of bigraph term languages, several of which exist in the wild, though they are all largely similar.

We first present a grammar for our term language, and then explain the mapping from these constructs to the graphical notation:

Definition 11 (A Bigraph term language)**.**

$$P \stackrel{\text{def}}{=} 0 \mid \mathsf{nil} \mid \mathsf{c}\langle\bar{x}\rangle.P \mid P \mid P' \mid \boxed{i} \mid P \,||\, P'$$

where $\bar{x}$ is a (possibly-empty) ordered list of names.

This definition requires some explanation: 0 is referred to as the *null* bigraph, which is used as the terminator for the composition of regions. nil is the nil process, which represents an empty bigraph. $\mathsf{c}\langle\bar{x}\rangle.P$ is *prefixing*, which is to say that there exists some node with control c, ports $x_0 \ldots x_n \in \bar{x}$, and children P. $P \mid P'$ is juxtaposition of sibling bigraphs under the same

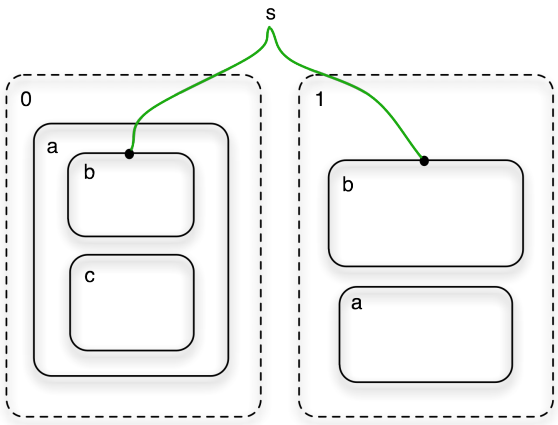


Figure 2.20: The bigraph $\mathsf{a}.(\mathsf{b}\langle s\rangle \mid \mathsf{c}) \,||\, \mathsf{b}\langle s\rangle \mid \mathsf{a}$

parent, such that P and P' are siblings. The notation $\boxed{i}$ is a hole, indexed by i. Finally, $P \,||\, P'$ introduces regions, which should be interpreted as being indexed $0 \ldots n$ from left to right.

For example, the process

$$\mathsf{a}\langle\rangle.(\mathsf{b}\langle s\rangle.\mathsf{nil} \mid \mathsf{c}\langle\rangle.\mathsf{nil}) \,||\, \mathsf{b}\langle s\rangle.\mathsf{nil} \mid \mathsf{a}\langle\rangle.\mathsf{nil} \,||\, 0$$

is illustrated in the graphical notation in Fig. 2.20. Where a name set is empty or where nil and null processes are unambiguous, we omit them, so the same process could be rewritten:

$$\mathsf{a}.(\mathsf{b}\langle s\rangle \mid \mathsf{c}) \,||\, \mathsf{b}\langle s\rangle \mid \mathsf{a}$$

Which is equivalent. It becomes clear that this representation is a very compact means of expressing bigraphs. The main part missing from this term language is a means of expressing edges and inner names — such a mechanism (and formal translation to the graphical structure) is given in Chapter 3. It is worth noting that this term language is very similar to the input language for the BigMC bigraphical model checker tool, described in Chapter 5.

2.5.1 BRS Contexts

We have seen that through composition, one bigraph may be considered a *context* for another. The implications of this are several.

We recall that in Section 2.1.1, when discussing our vending machine example, we made reference to an environment with which that vending machine must interact in order to make progress (or indeed, to do anything at all). There must be some environment willing to supply coins, push buttons, and accept drinks from the machine. Based on the semantics of CCS we described, we can derive a minimal context for the vending machine, which is effectively some environment willing to provide the corresponding actions at each step:

$$Ctx_{VM} \stackrel{\text{def}}{=} \overline{\mathsf{coin}}.(\overline{\mathsf{button1}}.\mathsf{drink1}.Ctx_{VM} + \overline{\mathsf{button2}}.\mathsf{drink2}.Ctx_{VM})$$

When composed with our vending machine, this context permits the vending machine to perform all of its actions infinitely.

So what is a context for a bigraphical reactive system? For some bigraph B, a context turns out to be any bigraphical context C into which we could place B that would enable reaction (by analogy to the CCS example, this qualifies as "doing something"). For example, if we had the rule:

$$\mathcal{R} = \mathsf{a}.\boxed{0} \mid \mathsf{c}.\boxed{1} \rightarrow \mathsf{b}.\boxed{0} \mid \mathsf{c}.\boxed{1}$$

and an agent:

$$B = \mathsf{a}.\mathsf{b}$$

then it is obvious that the rule $\mathcal{R}$ is not presently enabled with respect to B (i.e., the redex of $\mathcal{R}$ does not match in B). We could place B into a context C such as:

$$C = \boxed{0} \mid \mathsf{c}.\mathsf{d}$$

Which would give

$$C \circ B = \mathsf{a}.\mathsf{b} \mid \mathsf{c}.\mathsf{d}$$

for which $\mathcal{R}$ is enabled. We have therefore found a context C that enables reaction in B. This context, however, is not *minimal*, in the sense that there is a "smaller" context that would also enable reaction (in this case, $\boxed{0} \mid \mathsf{c}.nil$). For a full discussion of the role of minimal contexts in enabling reaction, see [83].

2.6 Categorical Foundations

The formulation of bigraphs we have given thus far is independent of the elegant mathematical foundations upon which Milner's original presentation

of bigraphs rests. We will therefore return now to the *categorical* foundations — i.e., the formulation of bigraphs in terms of category theory. This will also permit us to draw several distinctions which would otherwise be awkward to express, but which become considerably simpler in this setting. We do not aim to describe category theory here, though we will briefly revisit the fundamentals. Interested readers should refer to a standard text on category theory such as that of Awodey [8], upon which much of this introductory material is based.

Category theory makes distinctions between two types of mathematical constructions: *objects* and *arrows* (or *morphisms* in some presentations), which are the constituents of a *category*.

Definition 12 (Category)**.** Following [8], we define a category as consisting of:

- Objects $A, B, C, \ldots$
- Arrows $f, g, h, \ldots$
- An arrow has a *domain* and a *codomain*, for which we write $f : A \to B$ to mean A is the domain and B is the codomain of the arrow f.
- Composition is defined for arrows where the domain and codomain agree, such that for arrows $f : A \to B$ and $g : B \to C$, we have $g \circ f : A \to C$
- Objects have *identity* arrows, such that for any object A, there is an arrow $1_A : A \to A$ that is its identity.
- Composition is *associative*, such that for $f : A \to B$, $g : B \to C$, and $h : C \to D$, it is always the case that $h \circ (g \circ f) = (h \circ g) \circ f$.
- Identity arrows are the unit of composition, such that for $f : A \to B$: $f \circ 1_A = f = 1_B \circ f$.

2.6.1 Bigraphical Categories

The notation used for arrows such as $f : A \to B$ is reminiscent of that used in bigraphs, and this is no accident. In our bigraphical categories, the objects are interfaces, and the bigraphs themselves are the arrows. The definition of bigraph composition that we have already seen gives us composition of arrows that is associative, and it is easy to contrive bigraphs that are identity arrows for any given interface.

We introduce a distinction here between *abstract* and *concrete* bigraphs. The bigraphs we have seen thus far are all concrete bigraphs. The main difference subsists in whether nodes and edges are identified in some sense, i.e., whether we can distinguish two bigraphs that are structurally identical, but for which the node and edge sets are distinct. These identifying elements of a bigraph are called its *support*. We define this precisely:

Definition 13 (Support)**.** The *support* $|B|$ of any particular (concrete) bigraph $B = (V, E, ctrl, prnt, link) : \langle m, X \rangle \to \langle n, Y \rangle$ is:

$$|B| \stackrel{\text{def}}{=} V \uplus E$$

Having support for bigraphs gives rise to a natural notion of *support translation* and *support equivalence*. We avoid making these precise here, as full details are available in [83]. Rather, we simply give the intuition behind the concepts. A support translation is a pair of bijections on nodes and edges between two bigraphs, which gives rise to support equivalence where such a translation exists. Support equivalence for two bigraphs F and G is written $F \simeq G$, and is approximately the condition that two bigraphs are structurally the same up to the specific choice of the nodes and edges that constitute their support. The control, parent, and link maps are the same, but the specific node and edge identities are different. If they were precisely the same, then $F = G$ would be appropriate.

We also draw a distinction here with *lean* bigraphs, which are those without any *idle* edges. An edge is idle if there are no inner names or ports mapped to it. Two bigraphs F and G are *lean-support equivalent* (for which we write $F \eqsim G$) when they are support equivalent if their idle edges are not considered.

The distinction between abstract and concrete bigraphs becomes significant when we come to consider the kinds of bigraphical categories we can construct, but it is important to note that moving between each setting is simple. To move from a concrete bigraph to an abstract one, we simply discard its support (i.e., we forget all of its node and edge identifiers). To move from an abstract bigraph to a concrete one, we just need to provide its support by adding distinguishing identifiers for nodes and edges. This exercise (and indeed the distinction between concrete and abstract bigraphs) is mostly a mathematical one, i.e., it is not possible to distinguish a concrete bigraph from an abstract one by simply looking at its graphical representation, and therefore this distinction is primarily of importance for those who wish to study the foundations of bigraphs, rather than those who simply wish to employ them within a particular modelling task.

Bigraph categories divide generally into two varieties, the details of which we will give only in summary: for concrete bigraphs, we work mostly in *s-categories*, while for abstract bigraphs we work mostly in *symmetric partial monoidal* (spm) categories. The primary difference between each of these, and between these and "regular" categories, is the handling of partiality with respect to composition and tensor product. Specifically, an s-category has support and permits composition of arrows to be undefined even where the domain and codomain of those arrows agree. This is primarily to permit composition and tensor product only for those bigraphs that agree upon inner and outer interfaces, *and* which have disjoint support — otherwise composition could violate the structural conditions (e.g., potentially create cycles in the place graph, or malformed links in the link graph). An spm-category has a partial tensor product $I \otimes J$, with the condition that if $I \otimes J$ is defined, then so is $J \otimes I$ (hence *symmetric*). Milner [83] provides a much more rigorous treatment of these categorical structures, and as always should be considered the primary source for such information.

We introduce notation for our bigraphical categories: When discussing an s-category of (concrete) bigraphs over a signature Σ, we write $\grave{}\mathbf{BG}(\Sigma)$ to mean the (infinite) category consisting of all possible bigraphs over that signature as arrows, and all possible interfaces (of width $0 \ldots \infty$ and with name sets drawn from the powerset of some infinite set of possible names) as objects. For spm-categories of (abstract) bigraphs, we use the notation $\mathbf{BG}(\Sigma)$.

2.6.2 Functors

A functor $F : A \to B$ is a categorical construct that maps objects and arrows of one category to objects and arrows of another, in a way that respects identities (i.e., $F_{1_A} = 1_{F(A)}$), reflects domains and codomains of arrows (i.e., $F(f : A \to B) = F(f) : F(A) \to F(B)$), and is compositional (i.e., $F(g \circ f) = F(g) \circ F(f)$) [8]. Functors are standard in category theory, and therefore they become a useful tool with which to express relationships between different bigraphical categories. We will see an example of this in Chapter 4, but we also recall Milner's definition of the lean-support quotient functor:

$$[\![\cdot]\!] : \grave{}\mathbf{BG}(\Sigma) \to \mathbf{BG}(\Sigma)$$

That maps a category of concrete bigraphs to a category of abstract bigraphs (by "forgetting" the support and ignoring idle edges). Similarly,

these categories may be equipped with a *width functor width* : $BG(\Sigma) \to NAT$, where NAT is the category of natural numbers.

2.6.3 Bigraphical Reactive System Categories

Having arrived at a notation for expressing an entire category of bigraphs that share a common signature, we can also introduce notation for such a category augmented with a set of reaction rules (i.e., a bigraphical reactive system). For this we use the notation $\mathbf{BG}(\Sigma, \mathcal{R})$, where Σ is a signature and $\mathcal{R}$ is a set of reaction rules. We use the same convention for distinguishing s-categories from spm-categories.

2.7 Sortings

We have seen that a signature Σ gives rise to a bigraphical reactive system $\mathbf{BG}(\Sigma, \mathcal{R})$ that contains all possible bigraphs that may be constructed over that signature. Assuming we wished to encode arbitrary terms of the λ-calculus [12] within bigraphs, we could choose an appropriate signature. We give a grammar for an untyped λ-calculus as follows:

$$E ::= \lambda x.E \mid EE' \mid x$$

where x may range over some infinite set of names. Terms of the form $\lambda x.E$ are called λ-abstractions, and so we will use the control Abs to represent them. EE' is the application of E to E', so we will represent expressions of this form with the control App. Finally, variables may appear anywhere in expressions, so we will use the control Var to represent these. We could give a signature for our BRS as follows:

$$\Sigma_\lambda \stackrel{\text{def}}{=} \{\mathsf{Abs} : 1, \mathsf{App} : 0, \mathsf{Var} : 1, \mathsf{L} : 0, \mathsf{R} : 0\}$$

Which would allow us to represent a term such as $\lambda x.\lambda y.xy$ as:

$$\mathsf{Abs}\langle x\rangle.\mathsf{Abs}\langle y\rangle.\mathsf{App}.(\mathsf{L}.\mathsf{Var}\langle x\rangle \mid \mathsf{R}.\mathsf{Var}\langle y\rangle)$$

The use of L and R allows us to distinguish the left and right operands to application despite the unordered nature of place graph siblings (other strategies for this exist, but we have chosen a particularly simple one). This is clearly an agent of our bigraphical reactive system, as it uses only controls drawn from the signature we have chosen. However, what of this agent?

$$\mathsf{Var}\langle z\rangle \mid \mathsf{Abs}\langle y\rangle.\mathsf{App}.(\mathsf{L}.\mathsf{App}.\mathsf{R}.\mathsf{Var}\langle x\rangle \mid \mathsf{L}.\mathsf{Abs}\langle x\rangle)$$

This is clearly still an agent of our BRS, as it uses only appropriate controls drawn from the signature, but it is obviously not what we want. This bigraph has no corresponding well-formed λ-calculus term, and it does not conform to the grammar we specified for λ-calculus terms. It also references variables outside the scope of their binders, so both the place graph (representing the abstract syntax of terms) and the link graph (representing name binding) are malformed with respect to the system we are trying to model.

This problem is frequently encountered, and it effectively results from signatures being insufficiently expressive to exclude particular "bad" bigraphs from our bigraphical reactive system. This is where *sorting* is used. A word on terminology: terms such as "type", "kind", and "sort" are commonly used in certain contexts to identify a hierarchy of distinguished classes and meta-classes. The use of "sorting" in this context (while deceptively similar in purpose) comes not from meta-meta-types, but rather from many-sorted algebras, in a slightly-unfortunate instance of namespace collision between related fields.

A sorting is a mechanism for limiting the bigraphs of a bigraphical reactive system to only those "good" bigraphs that are well-formed with respect to some external set of constraints. Several forms of sortings exist, but we will briefly introduce the *predicate sortings* of Debois [35] here.

A predicate sorting — as the name suggests — introduces a predicate ϕ on bigraphs that distinguishes good bigraphs from bad ones. Such a predicate then gives rise to a functor mapping bigraphs of a *sorted* category to those of an *unsorted* category. The functor expresses the sorting (and in some sense *is* the sorting), as it partitions our BRS into those agents which have an image in the functor (i.e., for some functor $F : A \to B$ for which $F(a) = b$, b is said to be the F-image of a) and those that do not.

Sorting predicates are commonly added to either the notation for the BRS category, in the style $\mathbf{BG}(\Sigma, \mathcal{R}, \phi)$, or to the signature in the manner $\mathbf{BG}(\Theta, \mathcal{R})$ where $\Theta = (\Sigma, \phi)$.

What form would a sorting for our λ-calculus example take? In Chapter 3 we provide a mechanism for describing bigraphical sortings precisely, and in a manner that is guaranteed to preserve desirable properties of the sorted category (e.g., the presence of RPOs is not always guaranteed). For now, we can give informally the flavour of a sorting for λ-calculus:

- Var nodes always have zero children.
- Abs nodes have exactly one child that is either App, Var, or Abs.

- App nodes always have two children, L and R.
- L and R nodes must only have parent App.
- If some Var is linked to some Abs, then that Abs is a place graph ancestor of the Var.

The first four rules are structural constraints to ensure that terms are well-formed, while the last constraint is a *scope condition* that ensures that uses of bound names appear only within the scope of their binders. These conditions would admit our first (well-formed) λ-calculus example, and successfully reject the latter. Similar scope conditions exist for other calculi, including the π-calculus (which we give in Chapter 3). These kinds of scope conditions are closely related to both *binding bigraphs* [32] and *local bigraphs* [82], both of which were proposed by Milner (and developed further by others) as a means of controlling structural conditions associated with name binding and scope in general. Debois [35] provides an excellent summary of variations of bigraphs, and of those which may be recovered using sortings.

2.8 Representing Languages

One fascinating aspect of bigraphical reactive systems as an approach to the construction of models is that we end up defining not just one model, but an entire *family* of models. By defining a bigraph signature (and possibly a sorting), we define the syntax of our models, in terms of defining the constructors that may appear, and where they may appear with respect to one another. When we give a set of reaction rules, we are defining the dynamic semantics for a modelling language, the terms of which are determined by the signature and the sorting. We end up defining a modelling language that is specialised to a particular domain, usually without expending especially more effort than would have been required to define just one model. This allows many different models that share syntax and semantics to be constructed within a unified domain-specific modelling language.

The other benefit of this approach is that tool support for bigraphs permits instantiation of these tools for any given language defined in terms of bigraphical reactive systems. Any editor that can manipulate bigraphs immediately becomes an editor for the specific domain-specific modelling language that we have defined, and so it is with analysis or verification tools, simulators, or visualisation tools. This represents a considerable saving of

labour compared to the approaches that would require these tools to be built for every instance language individually.

2.9 Computation

The representation of *computation* within bigraphs is a recurring theme. While modelling languages and process calculi commonly abstract away computation, the varied goals of bigraphs mean that it is sometimes desirable to include actual computation within models. The first suggestions of such a mechanism were provided by Milner in his work on *growth* [83], which considered systems that behave in an "unfolding" manner. This work was continued by Debois [36] in his work *Computation in the Informatic Jungle*, and we briefly summarise this work here.

When considering computation within models, we are really talking about embedding one language within another — in this case, embedding a language concerned with computation within one that is appropriate to our modelling task. The primary challenge inherent in such an approach is that constructs of one language may require multiple steps in order to end in a consistent state. For example, Debois uses the example of a door that is age-controlled. We annotate users with their age, represented in unary form (e.g., using S to mean successor and Z to mean zero, the number 3 would be $\mathsf{S.S.S.Z}$). We can encode within our system consisting of doors and age-annotated users a sub-language that allows us to compute conditions over unary numbers — e.g., we can insist that the user's age be greater than 18. Given two unary numbers A and B encoded as bigraphs, computing $A < B$ is relatively simple. We use the controls Left and Right to record which operand is which, and give the rules as follows:

$$\mathsf{Left.S.}\boxed{0} \mid \mathsf{Right.S.}\boxed{1} \rightarrow \mathsf{Left.}\boxed{0} \mid \mathsf{Right.}\boxed{1}$$

$$\mathsf{Left.Z} \mid \mathsf{Right.S.}\boxed{0} \rightarrow \mathsf{True}$$

$$\mathsf{Left.}\boxed{0} \mid \mathsf{Right.Z} \rightarrow \mathsf{False}$$

These three rules, taken together, permit us to compute $A < B$ for arbitrary A and B. For example, to compute $3 < 4$:

$$\mathsf{Left.S.S.S.Z} \mid \mathsf{Right.S.S.S.S.Z}$$

$$\mathsf{Left.S.S.Z} \mid \mathsf{Right.S.S.S.Z}$$

$$\mathsf{Left.S.Z} \mid \mathsf{Right.S.S.Z}$$

Left.Z | Right.S.Z

True

We can see that $3 < 4$ evaluates to the control True after five steps of reaction, which is fine in this case. In Chapter 6 we will begin to consider how computation can be integrated into a domain-specific modelling language.

2.10 Tools and Techniques

Tool support for bigraph models is still in its early days, but more and more tools are becoming available and usable. A recent summary of tools is given by Højsgaard [61]; instead, we focus on two recent tools.

2.10.1 Big Red

Big Red [42] is a graphical editor for bigraphs, based upon Eclipse. It permits editing of signatures (including custom shapes for different controls), agents, and reaction rules. It also interfaces with the BigMC tool (see Chapter 5), to permit execution of bigraphical reactive systems, and checking of properties of these.

2.10.2 BigMC

BigMC [92] is a prototype tool for bigraphs that supports checking of properties expressed as matching. It can be used to execute bigraphical reactive systems, and inspect reachable states within such systems. BigMC is described in detail in Chapter 5.

2.11 Common Idioms for Modelling

We devote this section to capturing a number of common patterns that may prove useful when attempting to construct models or modelling languages within bigraphical reactive systems. Where possible, the first known instance of this pattern in the literature is given — in other instances, the technique may be novel to this thesis.

2.11.1 Sequences and Lists

Sequential representations of data are ubiquitous in programs, and therefore we provide two approaches to the structuring of such data within bigraphs, and two corresponding ways writing reaction rules over them.

Nested Lists

We can encode lists in a manner analogous to cons cells [11], which are pervasive in functional programming languages. We use the nested link structure to encode the list structure, and introduce a passive control Cons, of arity 0. Therefore a list consisting of Apple, Orange, and Banana would be encoded as follows:

$$\mathsf{Cons}.(\mathsf{Apple} \mid \mathsf{Cons}.(\mathsf{Orange} \mid \mathsf{Cons}.(\mathsf{Banana} \mid \mathsf{Cons})))$$

This representation uses the place graph in the manner of a binary tree, using the control assigned to the children to distinguish the tail of the list from the data represented in that position. We could write two reaction rules that together will function as a *flatten* operation, taking the list and promoting all of its elements to the top-level, destroying the list in the process:

$$\mathsf{Cons}.(\boxed{0} \mid \mathsf{Cons}.\boxed{1}) \rightarrow \boxed{0} \mid \mathsf{Cons}.\boxed{1}$$

$$\mathsf{Cons} \rightarrow \mathit{nil}$$

This allows a destructive traversal of the list, and manipulation of the list elements at each step.

Linked Lists

Linked lists are a common data structure in many programming idioms, and we can use the link graph structure to encode them in a straightforward manner. We introduce a passive control List with arity 2. One port will be an identifying link to the predecessor, and the other port will be the successor link. We also introduce a passive Null control of arity 1 to indicate the end of a list. We can therefore encode a linked list as follows:

$$\mathsf{List}\langle x0, x1\rangle.\mathsf{Apple} \mid \mathsf{List}\langle x1, x2\rangle.\mathsf{Orange} \mid \mathsf{List}\langle x2, x3\rangle.\mathsf{Banana} \mid \mathsf{Null}\langle x3\rangle$$

This style of list allows a simple in-place traversal using the link graph once again — we use a passive control Pointer of arity 1 to point to the current place in the list:

$$\mathsf{Pointer}\langle x0\rangle \mid$$

$$\mathsf{List}\langle x0, x1\rangle.\mathsf{Apple} \mid \mathsf{List}\langle x1, x2\rangle.\mathsf{Orange} \mid \mathsf{List}\langle x2, x3\rangle.\mathsf{Banana} \mid \mathsf{Null}\langle x3\rangle$$

We can then write the same *flatten* function in a non-destructive manner:

$$\mathsf{Pointer}\langle h\rangle \mid \mathsf{List}\langle h, t\rangle.\boxed{0} \to \boxed{0} \mid \mathsf{Pointer}\langle t\rangle \mid \mathsf{List}\langle h, t\rangle$$

2.11.2 Corecursion

It can be desirable to include corecursive (i.e., dual to recursive, where each successive step moves further from the base case) structures within bigraphical reactive systems. A concrete example would be a process calculus term such as the following:

$$\mathsf{MACHINE} := \mathsf{x}?.\mathsf{y}!.\mathsf{MACHINE}$$

which is the process that inputs on channel x, outputs on channel y, and then behaves once again like $\mathsf{MACHINE}$. The way to encode such a process is to use a special token, and to encode the corecursion into the reaction rules. Assuming a signature consisting of the passive controls send, recv, and $\mathsf{MACHINE}$, we construct a reaction rule as follows:

$$\mathsf{MACHINE} \to \mathsf{recv}\langle x\rangle.\mathsf{send}\langle y\rangle.\mathsf{MACHINE}$$

and then we construct the initial system as simply:

$$\mathsf{MACHINE}$$

This corresponds to a simplified version of the condition on *unfolding* seeds, as introduced by Debois [36].

Chapter 3

BMC & BAM: A Declaratively-Sorted Bigraphical Meta-Calculus & Abstract Machine

Context

This chapter is derived from the forthcoming IT University of Copenhagen technical report BMC & BAM: A Declaratively-Sorted Bigraphical Meta-Calculus & Abstract Machine *by Thomas Hildebrandt and Gian Perrone, 2012. A version of this work is also (as of writing) under review for ESOP 2013, the results of which will be known prior to the eventual publication of this thesis.*

Abstract

We present a declaratively-sorted bigraphical meta-calculus (BMC) and a general bigraphical abstract machine (BAM) which can be instantiated to any instance calculus of BMC and both fair and stochastic evaluation strategies. The BMC meta-calculus thereby enables the familiar presentation and manipulation of a broad class of sorted bigraphical reactive systems as simple process calculus terms which can be executed and simulated using BAM, without the need for any knowledge of category theory. The declarative sorting mechanism is given by a new XPath-like bigraph logic, and guarantees the existence of a corresponding closure-sorted, pure bigraphical reactive system as defined by Debois et al., for which a labelled bisimulation congruence can be derived. The logic allows for intuitive declaration of the sorting and binding constraints for all the process models previously described as closure-sorted bigraphical reactive systems, and in particular for the pi-calculus. Finally, we prove that any so-called *maximal* execution strategy for BAM gives rise to a fair execution strategy.

3.1 Introduction

Through the historical development of process calculi and formalisms for describing the behaviour of communicating systems, the proliferation of similar-but-incompatible languages and calculi gave rise to a need for *unifying* mechanisms. Several efforts in this direction have emerged, including action calculi [75], Fusion systems [43], Kell calculus [98], and Psi-calculi [14]. In any unifying formalism, we believe a design goal needs to be the minimisation of the number of primitive constructs required, in order to limit the complexity of the formalism.

Bigraphs have been successful as a meta-model in capturing the syntax and semantics of many different formalisms (e.g., λ-calculus [82], CCS [81], the mobile ambient calculus [64], several variants of the π-calculus [64, 26, 39], Fusion Calculus [50], and Petri Nets [71]), and in being used directly as a modelling formalism in diverse domains such as business processes [56, 105], context-aware computing [20], and systems biology [68, 33]. However, the promise of bigraphs as a unifying framework for the study, design, and analysis of process languages has yet to be realised fully.

We claim that one major barrier to the adoption and use of bigraphs has been the focus on presenting bigraphical reactive systems (BRSs) as categorical, graph-rewriting systems, rather than as process languages. We

aim to demonstrate within this paper that declaratively-sorted bigraphical reactive systems [80] lend themselves to the design of a small, simple meta-calculus that is sufficiently powerful and general to uniformly represent a wide variety of nominal calculi, and eliminates the need to understand the graph representation and underlying category theory before being able to use BRSs to build and analyse process languages.

Concurrent with the development and proliferation of nominal calculi for describing interaction, the pragmatic benefits of *executing* systems described in this manner have become increasingly clear. Formulations of the semantics of many different calculi as *abstract machines* have appeared (e.g., the Pict abstract machine [19] and the Fusion machine [44]). Consequently, we propose a meta-abstract machine that may be instantiated as a graph-based abstract machine for any bigraphical reactive system (and therefore for any instance calculus). The ability to uniformly derive such abstract machines from declarative language specifications is a significant improvement in providing unifying mechanisms for these kinds of calculi.

In summary, we present three major contributions within this paper: A meta-process calculus for nominal calculi that characterises a large class of closure-sorted bigraphical reactive systems, a sorting logic that always gives rise to well-behaved sorting predicates, and a bigraphical abstract machine over bigraphical reactive systems that may be described in terms of the meta-calculus, and can be instantiated as a fair abstract machine for any instance calculus. Together, these three elements advance the theory and applicability of bigraphs, and advance the stated goals of the bigraphical reactive systems formalism by permitting further systematic study of nominal calculi in this setting.

3.2 A Bigraphical Meta-Calculus: BMC

As described in the introduction, we will in this paper present Milner's pure bigraphical reactive systems sorted by the general closure sortings introduced by Debois et al. [35] as instances of a meta-process calculus *BMC* (bigraphical meta-calculus). To describe processes, BMC includes only the core operations of nominal concurrent process calculi: parallel composition, control prefixing, name restriction, and renaming. We will use variations of the mobile ambient calculus [28] and the π-calculus [85] as running examples.

Example 1. We give the terms

$$\mathsf{recv}\langle x, y\rangle.\mathsf{send}\langle y, z\rangle \mid (\nu a)(\mathsf{send}\langle x, a\rangle.\mathsf{recv}\langle a, b\rangle)$$

and

$$\mathsf{amb}\langle x\rangle.\mathsf{in}\langle y\rangle.\mathbf{nil} \mid \mathsf{amb}\langle y\rangle.\mathbf{nil}$$

as examples of terms from a BMC instantiation of a version of the π-calculus and an instantiation of the ambient calculus (respectively), both given below. Note that there are no built-in name binders in the core calculus. The scope of the names y and b in the example term will be restricted by the sorting logic defined in Sec. 3.2.2.

We will now in Sec. 3.2.1-3.2.4 describe how an instance of the meta-calculus is defined by giving a *(bigraph) signature* (defining the allowed control prefixes), a *sorting* (e.g. defining the grammar and scope rules), and a set of *reaction rules* (defining the operational semantics). We then show in Sec. 3.3 that for any specific instance of the meta-calculus, the syntax trees for multi-hole contexts in the calculus will then correspond to so-called closure sorted, pure bigraphs for the same signature. Moreover, the operational semantics of the calculus will correspond to the semantics of the bigraphical reactive system.

3.2.1 Pure Signatures and BMC Process Terms

An instance of a bigraphical reactive system, and thus our meta-calculus, is defined by giving a pure bigraph signature [77] defining the set of controls that can appear as prefixes, the number of name parameters for each type of control (the arity), and whether a control is active or passive. Active and passive controls respectively allow and disallow reactions (rewriting) under a prefix with that control.

Definition 14 (Pure bigraph signature)**.** A pure *bigraph signature* is a tuple (Σ, ar, Σ_a), where Σ is a set of *controls*, $ar : \Sigma \to \omega$ is a function assigning a finite ordinal (the arity) to each control, and $\Sigma_a \subseteq \Sigma$ is the subset of *active* controls. In order to compactly represent signatures, for a signature where $\Sigma = \{\mathsf{c_1}, \ldots, \mathsf{c_n}\}$, with $\Sigma_a \subseteq \Sigma$ and $ar(\mathsf{c_i}) = ar_i$, we will write:

$$(\{\mathsf{c_1} : ar_1, \ldots, \mathsf{c_n} : ar_n\}, \Sigma_a)$$

We will often refer to a signature (Σ, ar, Σ_a) by simply the set of controls Σ.

Example 2. The signature $\Sigma_{\mathbf{Amb_{io}}} \stackrel{\text{def}}{=} (\{\mathsf{in} : 1, \mathsf{out} : 1, \mathsf{amb} : 1\}, \{\mathsf{amb}\})$ describes a simple variant of the ambient calculus also considered in [83] and employed in our example term above. It features only in and out commands given as controls with a single name, and the ambient amb control which also has a single name and is active. The signature $\Sigma_{\mathbf{Pi}_{sr}} = (\{\mathsf{send} : 2, \mathsf{recv} : 2\}, \emptyset)$ describes a very simple version of the π-calculus [85], $\mathbf{Pi}_{sr}$, which includes only input and output controls, and which has no active controls.

We now define the core process terms of the meta-calculus relative to a signature. We assume a global set of names N ranged over by a, b, c, x, y, z.

Definition 15 (Meta-calculus process terms)**.** For a signature Σ, the processes terms p are defined by the grammar

$p ::=$	nil	Nil process
	$\mathsf{c}\langle\bar{x}\rangle.p$	Prefix
	$p \mid p$	Parallel process
	$(\nu x)p$	Restriction

for $\mathsf{c} \in \Sigma$, $\bar{x} \in N^*$ and $|\bar{x}| = ar(\mathsf{c})$.

However, as already indicated in the first example above, the core grammar above only restricts the possible controls used as prefixes. It does not allow any grammatical constraints on the nesting of controls, nor any distinction between binding and non-binding names within prefixes. The usual approach would be to define grammatical constraints using a BNF grammar that imposes further constraints on the above grammar, and introduce distinctions between the names for controls as in Fusion, Psi, and binding bigraphs. However, an alternative solution has been proposed for bigraphical reactive systems in the form of *sortings* that identify the subset of *well-sorted* terms. Below we introduce a new logic permitting sortings to be expressed declaratively.

3.2.2 Declarative Sortings

We consider sortings in terms of *predicate sortings* [35], in which some deconstructable (i.e., closed under sub-terms) predicate is used to determine whether a given term is in included in a given instance calculus.

A new contribution of the present paper is a logic for the construction of predicates over terms of our meta-process calculus, that is shown (in Sec. 3.3.2) to be guaranteed to give rise to appropriate deconstructable

predicates. This logic is sufficiently expressive to recover all the sortings proposed in the literature that were identified by Debois as being replaceable with closure sortings [35]. Most significantly, we can recover a definition of *local* bigraphs [82], which allows one to restrict terms to only those well-formed π-calculus terms that obey the normal scope restriction conditions. The specific scope condition for **Pi** is:

$$\phi_\pi \stackrel{\text{def}}{=} \mathsf{recv}(u) \wedge u \neq v \wedge (u@2 = v@1 \vee u@2 = v@2) \implies u \mathbin{//} v$$

Intuitively, the predicate states that for any recv control u, if the second parameter ($u@2$) is linked to the first ($v@1$) or second parameter ($v@2$) of a distinct control v then that control must be nested within the u control ($u \mathbin{//} v$).

The following definition defines the syntax of the logic formally.

Definition 16 (Sorting Logic)**.** A formula ϕ is defined relative to a bigraph signature Σ, and is of the form:

$$\begin{aligned}
F &\stackrel{\text{def}}{=} && U^+ \implies U^- \mid F \vee F \mid F \wedge F \\
U &\stackrel{\text{def}}{=} && v = u \mid v \neq u \mid v \mathbin{/} u \mid v \mathbin{//} u \mid v \not\succ u \mid v \not/\!\!/ u \mid \mathsf{c}(u) \mid \neg\mathsf{c}(u) \\
U^+ &\stackrel{\text{def}}{=} && v@i = u@j \mid U^+ \wedge U^+ \mid U^+ \vee U^+ \mid \mathit{local}(v@i) \mid U \\
U^- &\stackrel{\text{def}}{=} && v@i \neq u@j \mid U^- \wedge U^- \mid U^- \vee U^- \mid \neg\mathit{local}(v@i) \mid U
\end{aligned}$$

where $\mathsf{c} \in \Sigma$, $i, j \in \mathbb{N}$, and $u, v \in \mathbf{Var}$, an infinite set of meta-variables.

Note the separation of the logic into positive (U^+) and negative (U^-) terms — this is key to preserving the decomposability property of predicates that we establish in Section 3.3.2. Positive terms only appear under implication, which by simplification (i.e., $A \implies B \equiv \neg A \vee B$) means that formulae consist only of negative terms. It is also possible to write terms without implication, using $\mathit{true} \implies U^-$ (obtaining true by $x = x$).

We have chosen to use syntax inspired by XPath [15], as the properties of terms we wish to capture are in some respects similar. We will give a formal interpretation of these logical connectives in Section 3.3.1, but for now we will give the intuition behind the operators. The variables like u and v range over all the prefixes of a term, which when combined with the various operators permit us to express structural conditions. To check whether a particular prefix v has a control c, we use the syntax $\mathsf{c}(v)$. The formula

v / u is true iff the prefix assigned to u occurs directly under the prefix assigned to v. The v // u operator is similar, but it insists instead that u appear *anywhere* under the prefix assigned to v, i.e. at arbitrary depth. Given that prefixes are assigned to meta-variables, we can also consider statements about the names that appear in these prefixes — $v@i = u@j$ allows us to check whether the ith name of the prefix assigned to v is shared with the jth name of u. The connectives $v \not/ u$ and $v \not// u$ are the negation of v / u and v // u, respectively. The final non-standard logical constructs are $local(v@i)$ and $\neg local(v@i)$, which insist (respectively) that the ith name of the prefix assigned to v be or not be bound to a name restriction. For example, $local(v@1)$ applied to $(\nu x)(\mathsf{send}\langle x, y\rangle)$, will be true, while $local(v@2)$ for the same term would be false.

We can now extend signatures with a sorting given declaratively by a formula of the sorting logic.

Definition 17 (Sorted BRS Signature)**.** A *sorted* BRS signature is defined as

$$\Theta \stackrel{\text{def}}{=} (\Sigma, \phi)$$

where Σ is a pure bigraph signature, and ϕ is a sorting predicate. We use $\mathbf{BMC}(\Theta)$ to mean the set of all BMC processes of a sorted signature Θ.

3.2.3 BMC Contexts and Composition

To describe reaction rules (and characterise bigraphs) we extend our terms to *contexts*. Technically, the terms of our meta-calculus are divided in two kinds: *Prime* processes and *wide* processes. A prime process (processes of the sort we have seen already) can be inserted into a single-hole context. A wide process is essentially a list of prime processes, referred to as *regions* (separated by the wide parallel operator $\|$), which may be inserted as the contents of a multi-hole context. We extend the syntax to multi-hole process contexts by adding to p the construct $\boxed{i}$, a *hole* with index i, and the construct y/x, a *link* from the name x to y, where x is referred to as an *inner* name. Intuitively, a prime process p can be placed within a hole $\boxed{i}$ of a context and the link y/x in the context will rename any occurrence of x in p to the name y.

Definition 18 (BMC contexts)**.** For a signature Σ define the (wide and

prime) Σ-*contexts* r by extending the grammar of process terms as follows

$$\begin{array}{rll} r ::= & \mathbf{null} & \text{Null process} \\ & r \parallel r & \text{Wide parallel} \\ & p & \text{Prime} \\ & (\nu x)r & \text{Restriction} \\ & {}^{y}\!/_{x} & \text{Linking} \\ \\ p ::= & \ldots \mid \boxed{i} & \text{Prime contexts} \end{array}$$

for $x, y \in \mathsf{N}$ and $i \in \omega$. A (wide or prime) context is well-formed if 1) any two holes in the term have different indices, 2) the set of all indices of holes is an ordinal $\{0, 1, \ldots m-1\}$ for some $m \in \omega$ (referred to as the *inner* width) and 3) for any two links ${}^{y}\!/_{x}$ and ${}^{y'}\!/_{x'}$ appearing in the context we have that the inner names are distinct, i.e. $x \neq x'$. We refer to the set of all inner names appearing in links of a context as the *inner* name set of the context.

Definition 19 (Free names)**.** We define the free names of a (wide/prime) context inductively

$$\mathsf{fn}(r) \stackrel{\text{def}}{=} \begin{cases} \mathsf{fn}(r') \setminus \{z\} & \text{if } r = (\nu z)r' \\ \mathsf{fn}(r') \cup \mathsf{fn}(r'') & \text{if } r = r' \parallel r'' \text{ or } r = r' \mid r'' \\ \bar{x} \cup \mathsf{fn}(p) & \text{if } r = \mathsf{c}\langle \bar{x} \rangle.p \\ \{y\} & \text{if } r = {}^{y}\!/_{x} \\ \emptyset & \text{otherwise} \end{cases}$$

Example 3. The process term $p = \mathsf{amb}\langle x \rangle.\mathsf{in}\langle y \rangle$ of $\mathbf{Amb_{io}}$ can be inserted in the context $r = \boxed{0} \mid \mathsf{amb}\langle z \rangle \parallel {}^{z}\!/_{y}$, resulting in the process

$$r \circ p = \mathsf{amb}\langle x \rangle.\mathsf{in}\langle z \rangle.\mathbf{nil} \mid \mathsf{amb}\langle z \rangle.\mathbf{nil} \parallel {}^{z}\!/_{y} \ .$$

As usual, process and context terms are equipped with a structural congruence $\equiv$. For the BMC calculus, the structural congruence makes prime parallel composition associative and commutative, wide parallel composition associative, and the nil process **nil** and null process **null** respectively the identity for prime and wide parallel composition. Moreover, it allows scope extension and alpha conversion of local names.

The formulation of the structural congruence is prepared for the definition of a normal form for process contexts given below, in which all restrictions are extended to the encompass the entire term. The normal form can

be obtained by applying only α-conversion and relation $\rightarrow_{\equiv}$ in the direction of the arrow. It depends crucially on the last axiom, allowing restrictions of names to be lifted outside control prefixes, which is not usually allowed in the π-calculus (but is needed in the ambient calculus e.g., to allow scopes of local names to be extended outside ambients). A consequence of this axiom, as we will see below, is that treatment of replication (and in general duplication) of processes with local names needs special care.

Definition 20. Structural congruence $\equiv$ is the least congruence on expressions equating expressions up to α-conversion such that

$$(p_0 \mid p_1) \mid p_2 \equiv p_0 \mid (p_1 \mid p_2) \qquad (r_0 \parallel r_1) \parallel r_2 \equiv r_0 \parallel (r_1 \parallel r_2) \qquad p \mid p' \equiv p' \mid p$$

and including the following relation (where $x \notin \mathsf{fn}(r') \cup \mathsf{fn}(p') \cup \bar{z}$):

$$r \parallel \mathbf{null} \rightarrow_{\equiv} r \qquad (\nu x)r \parallel r' \rightarrow_{\equiv} (\nu x)(r \parallel r') \qquad (\nu x)p \mid p' \rightarrow_{\equiv} (\nu x)(p \mid p')$$

$$p' \mid (\nu x)p \rightarrow_{\equiv} (\nu x)(p' \mid p) \qquad r' \parallel (\nu x)r \rightarrow_{\equiv} (\nu x)(r' \parallel r) \qquad {}^{y}/_{z} \parallel r \rightarrow_{\equiv} r \parallel {}^{y}/_{z}$$

$$\mathbf{null} \parallel r \rightarrow_{\equiv} r \qquad p \mid \mathbf{nil} \rightarrow_{\equiv} p \qquad \mathbf{nil} \mid p \rightarrow_{\equiv} p \qquad \mathsf{c}\langle\bar{z}\rangle.(\nu x)p \rightarrow_{\equiv} (\nu x)\mathsf{c}\langle\bar{z}\rangle.p$$

Notation: Associativity allows us to leave out parentheses for prime and wide parallel composition, so we introduce two notationally convenient forms for representing processes. For a prime process $p_0 \mid \ldots \mid p_n$, we write $\Pi_{i\in n}\, p_i$. For a wide process $r_0 \parallel \cdots \parallel r_n$, we write $\mathbf{\Pi}_{i\in n}\, r_i$. Owing to the commutativity of $\mid$, we consider prime parallel processes unordered, as is usual in process calculi. We rely on the identities $\Pi_{i\in 0}\, p_i = \mathbf{nil}$ and $\mathbf{\Pi}_{i\in 0}\, r_i = \mathbf{null}$. As is usual, we will often leave out trailing nil processes, writing $\mathsf{c}_{\bar{x}}$ for $\mathsf{c}_{\bar{x}}.\mathbf{nil}$. We will also write ${}^{\bar{y}}/_{\bar{z}}$ for $\mathbf{\Pi}_{i\in n}\, {}^{y_i}/_{z_i}$.

Definition 21 (Normal form)**.** For a signature Σ we define the (wide and prime) Σ-*context expressions* $\mathbf{r}$ in normal form by the grammar

		wide concrete normal form contexts
$\mathbf{r}$	$::=$	$(\nu\bar{x})(\mathbf{\Pi}_{i\in n}\, \mathbf{p}_i \parallel {}^{\bar{y}}/_{\bar{z}})$
		prime concrete normal form contexts
$\mathbf{p}$	$::=$	$\Pi_{i\in n}\, \mathsf{c}\langle\bar{x}\rangle.\mathbf{p}_i \mid \Pi_{j\in m}\, \boxed{i_j}$

for $\mathsf{c} \in \Sigma$, $\bar{x} \in N^*$, $|\bar{x}| = ar(\mathsf{c})$ and $i_j \in \omega$. We refer to n, the number of top level prime processes in $\mathbf{r}$ above, as the *width* of wide context r. We write $r : m \rightarrow n$ for a wide context with m holes and width n.

Proposition 1 (Normal form representation)**.** Every wide context r is structurally congruent to a wide context $\mathbf{r} = (\nu\bar{x})(\mathbf{\Pi}_{i\in n}\,\mathbf{p}_i \parallel {}^{\bar{y}}/_{\bar{z}})$ on normal form, which is unique up to alpha-conversion and reordering of the names in $\bar{x}$, reordering of the linkings in ${}^{\bar{y}}/_{\bar{z}}$ and reordering of prime parallel compositions.

A wide context $r' : m \to n'$ can be inserted into a wide context $r : n' \to n$, forming the composite context $r \circ r' : m \to n$. Intuitively, the composition apply all linkings in r as substitutions/renamings on the free names of r', place the top level prime processes of r' in the corresponding holes of r, extending the scope of the local names of r' to embrace the entire process (possibly alpha-converting the local names of r'), and finally promote the (renamed) linkings of r' to be linkings of $r \circ r'$ as well as the linkings of r that does not have the same inner name as any linking in r'. This intuition is made formal in following definition.

Definition 22 (Composition of normal form contexts)**.** For two contexts $r' : m \to n'$ and $r : n' \to n$ we define $r \circ r' : m \to n$ as

$$r \circ r' = (\nu\bar{x}\bar{x}')(r''[p'_0\{\bar{z} \mapsto \bar{y}\}, \ldots, p'_{n-1}\{\bar{z} \mapsto \bar{y}\}] \parallel {}^{\bar{y}'\{\bar{z}\mapsto\bar{y}\}}/_{\bar{z}'} \parallel {}^{\bar{y}\setminus\{y_i|z_i\in\bar{z}'\}}/_{\bar{z}\setminus\bar{z}'}) \ ,$$

if $r = (\nu\bar{x})(r'' \parallel {}^{\bar{y}}/_{\bar{z}})$, $r' = (\nu\bar{x}')(\mathbf{\Pi}_{i\in n'}\,p'_i \parallel {}^{\bar{y}'}/_{\bar{z}'})$, $r'' = \mathbf{\Pi}_{i\in n}\,p_i$ and $\bar{x}' \cap (\mathsf{fn}(r'') \cup \bar{x} \cup \bar{z}) = \emptyset$.

In Section 3.3 below, we show that a context $r : m \to n$ with normal form $(\nu\bar{x})(\mathbf{\Pi}_{i\in n}\,\mathbf{p}_i \parallel {}^{\bar{y}}/_{\bar{z}})$ precisely corresponds to an abstract bigraph $\mathbf{brs}(r) : \langle m, \bar{z}\rangle \to \langle n, \mathsf{fn}(r)\rangle$ and the composition $r \circ r' : m \to n$ defined above corresponds to the bigraph composition $(\mathbf{brs}(r) \parallel \mathbf{Id}_{\mathsf{fn}(r')\setminus\bar{x}}) \circ (\mathbf{brs}(r') \parallel \mathbf{Id}_{\bar{z}\setminus\bar{z}'}) : \langle m, \bar{z} \cup \bar{z}'\rangle \to \langle n, \mathsf{fn}(r) \cup (\mathsf{fn}(r')\setminus\bar{x})\rangle$. Bigraph connoisseurs will note that the composition of terms is more liberal than the composition of bigraphs, since bigraphs are explicitly typed with outer and inner names, while terms are only typed by their width and number of holes.

3.2.4 BMC Reaction rules

The dynamics of a meta-calculus instance (and in bigraphical reactive systems) is defined by giving a set of *reaction rules* which defines how to rewrite portions of the process.

Definition 23 (Reaction Rule)**.** A reaction rule is expressed as a triple

$$(r_L : m \to n, r_R : m' \to n, \eta : (\mathcal{N} \times \mathcal{N}) \times (\mathcal{N} \times r))$$

where r_L is a BMC context to be matched (called the *redex*), r_R is a BMC context with which the matched redex will be replaced (called the *reactum*), and η is the *instantiation map* that maps occurrences of holes and names in the reactum to the matched parameters in the redex. Where the instantiation map is unambiguous (e.g., where it is just the identity map), we may omit it and simply write rules as $r_L \to r_R$, or $(r_L \to r_R, \eta)$ to aid readability where η is significant.

We express reaction on a process r leading to the process r' as a decomposition

$$r = r_C \circ r_L \circ D \to r_C \circ r_R \circ \eta(D) = r'$$

where r_C is an active *context* and $D = \mathbf{\Pi}_{i \in n}\, d_i$ is a wide parallel composition of prime processes with no name restrictions (i.e. an ordered list of matched parameters). Notice that the definition of η must ensure that the inner with of the reactum agrees with the instantiation of the parameters, such that for $D : 0 \to m$ (where m is the inner width of the redex), we have $\eta(D) : 0 \to m'$, where m' is the inner width of the reactum.

Example 4. The reaction rules for $\mathbf{Amb_{io}}$ are given by

$$\mathsf{amb}\langle x\rangle.(\mathsf{in}\langle y\rangle.\boxed{0} \mid \boxed{1}) \mid \mathsf{amb}\langle y\rangle.\boxed{2} \to \mathsf{amb}\langle y\rangle.(\mathsf{amb}\langle x\rangle.(\boxed{0} \mid \boxed{1}) \mid \boxed{2})$$

$$\mathsf{amb}\langle y\rangle.(\mathsf{amb}\langle x\rangle.(\mathsf{out}\langle y\rangle.\boxed{0} \mid \boxed{1}) \mid \boxed{2}) \to \mathsf{amb}\langle x\rangle.(\boxed{0} \mid \boxed{1}) \mid \mathsf{amb}\langle y\rangle.\boxed{2}$$

The single reaction rule for $\mathbf{Pi_{sr}}$ is given by

$$\mathsf{send}\langle x, y\rangle.\boxed{0} \mid \mathsf{recv}\langle x, z\rangle.\boxed{1} \to \boxed{0} \mid \boxed{1} \parallel {}^{y}/_{z}$$

To illustrate the use of the instantiation map and duplication of subprocesses during reactions, we consider extending $\mathbf{Pi_{sr}}$ with replication restricted to input actions (as in the variant of the π-calculus considered in [19]).

The most straightforward approach is to extend the signature to

$$\Sigma_{\mathbf{Pi}} \stackrel{\text{def}}{=} (\{\mathsf{send} : 2, \mathsf{recv} : 2, \mathsf{rep} : 0\}, \emptyset)$$

and add the following reaction rule to cover the semantics of replication:

$$\mathsf{send}\langle x, y\rangle.\boxed{0} \mid \mathsf{rep}.\mathsf{recv}\langle x, z\rangle.\boxed{1} \to \boxed{0} \mid \boxed{1} \mid \mathsf{rep}.\mathsf{recv}\langle x, z\rangle.\boxed{2} \parallel {}^{y}/_{z'}, \eta)$$

where $\eta = (\{0 \to 0, 1 \to 1, 2 \to 1\}, (\mathbf{null}, {}^{z'}/_{z}, \mathbf{null}))$.

The application of an instantiation map $(\bar{\eta_p}, \bar{\eta_l})$ to a set of parameters D is as follows:

$$\prod_{i \in |D|} (\boxed{i} \parallel \bar{\eta}_{l_i}) \circ \bar{\eta}_{p_i}(D_i)$$

The instantiation map η for the second rule is significant because of the mismatch between the number of holes in the redex and the reactum, as well as the need to rename parameters in the replicated sub-term to avoid accidental name capture. Consequently, the instantiation map consists of a map from holes of the reactum to holes of the redex, and an n-tuple of renamings to be applied to the parameters, where n is both the inner width of the reactum, and the outer width of the instantiated parameters.

The fact that replication is only allowed for input guarded processes can be expressed by a sorting constraint:

$$\phi_{rep} \stackrel{\text{def}}{=} (\mathsf{rep}(x) \wedge x \mathbin{/} y) \implies \mathsf{recv}(y)$$

The above BMC instance does however suffer from the a problem, in that local names can float out of the replication. This can however be solved by introducing a new control for local name definition within replication and adding rules for creating fresh names after a replica has been created.

3.3 From BMC to BRS (and back again)

In this section we show that BMC terms correspond to closure sorted, pure bigraphs in a precise sense. First we recall the definition of a pure bigraph.

Definition 24 (Pure bigraphs)**.** A bigraph is a 5-tuple:

$$(V, E, ctrl, prnt, link) : \langle m, \bar{x} \rangle \to \langle n, \bar{y} \rangle$$

where $ctrl : V \to \mathcal{K}$, $prnt : m \uplus V \to V \uplus n$, $link : \bar{x} \uplus P \to E \uplus \bar{y}$, and $P \stackrel{\text{def}}{=} \{(v, i) : i \in ar(ctrl(v))\}$ is the set of *ports*. The parent map *prnt* must be acyclic, i.e., if $prnt^i(v) = v$ for any $v \in V$, then $i = 0$ (where $prnt^i$ is i applications of *prnt*).

In this definition, *ctrl* is the *control map* that associates nodes with controls drawn from a signature. The parent map *prnt* defines the nested tree structure of the place graph, while the link map *link* associates the indexed *ports* of nodes and the *inner names* with (hyper-)edges (given by the set E) and *outer names*.

We now define the bigraph corresponding to a normal form term. Intuitively, the set of vertices will be paths of integers indicating the path to a control. We first define loc_r that returns for a non-empty path ρ the corresponding process term at that path:

- $loc_{\mathbf{r}}(k\rho) = loc_{\mathbf{p}_k}(\rho)$ if $\mathbf{r} = (\nu \bar{z})(\mathbf{\Pi}_{i \in |\mathbf{r}|}\, \mathbf{p}_i \parallel {}^{\bar{y}}/_{\bar{x}})$
- $loc_{\mathbf{p}}(k\rho) = loc_{\mathbf{p}_k}(\rho)$ if $\mathbf{p} = \Pi_{i \in |\mathbf{p}|}\, \mathsf{c}_i\langle \bar{x}_i \rangle.\mathbf{p}_i \mid \Pi_{j \in m}\, \boxed{i_j}$
- $loc_{\mathbf{p}}(k) = p_k$ if $\mathbf{p} = \Pi_{i \in |\mathbf{p}|}\, p_i \mid \Pi_{j \in m}\, \boxed{i_j}$

Given a $\mathbf{r} = (\nu \bar{z})(\mathbf{\Pi}_{i \in n}\, \mathbf{p}_i \parallel {}^{\bar{y}}/_{\bar{x}})$ we define the control map $ctrl_{\mathbf{r}}$ as a partial map from $\mathcal{N}^+$ as follows.

$$ctrl_{\mathbf{r}}(\rho) = \begin{cases} \mathsf{c} & \text{where } loc_{\mathbf{r}}(\rho) = \mathsf{c}\langle \bar{x} \rangle.p, \\ undefined & \text{otherwise} \end{cases}$$

The parent map $prnt$ is on vertices the 1-step prefix order and on hole indices $k \in m$ the parent map is defined by identifying the unique hole either under a root or using the loc map:

- $prnt_{\mathbf{r}}(\rho i) = \rho$ for $\rho \in \mathcal{N}^+$ and $i \in \mathcal{N}$.
- $prnt_{\mathbf{r}}(k) = i$, if $\mathbf{p}_i = \Pi_{i \in n'}\, \mathsf{c_i}\langle \bar{x}_i \rangle.\mathbf{p}'_i \mid \Pi_{j \in m}\, \boxed{i}_j$ and $\exists j \in m.i_j = k$
- $prnt_{\mathbf{r}}(k) = \rho$, if $loc_{\mathbf{r}}(\rho) = \mathsf{c}\langle \bar{x} \rangle.\mathbf{p}$ and $\mathbf{p} = \Pi_{i \in n'}\, \mathsf{c_i}\langle \bar{x}_i \rangle.\mathbf{p}'_i \mid \Pi_{j \in m}\, \boxed{i_j}$ and $\exists j \in m.i_j = k$

Finally we define the link map for $\mathbf{r} = (\nu \bar{x})(\mathbf{\Pi}_{i \in n}\, \mathbf{p}_i \parallel {}^{\bar{y}}/_{\bar{z}})$ as follows

- $link_{\mathbf{r}}(\rho j, i) = x_i$ if $loc_{\mathbf{r}}(\rho j) = \mathsf{c}\langle \bar{x} \rangle.\mathbf{p}'$ and $\bar{x} = x_1 \ldots x_{ar(\mathsf{c_j})}$
- $link_{\mathbf{r}}(z_j) = y_j$ if $\mathbf{r} = \mathbf{\Pi}_{i \in n}\, \mathbf{p}_i \parallel \mathbf{\Pi}_{j \in |\bar{z}|}\, {}^{y_j}/_{z_j}$

We can now define the bigraph corresponding to a BMC context as follows.

Definition 25 (Bigraph for BMC normal form term)**.** For a normal form context $\mathbf{r} = (\nu \bar{z})(\mathbf{\Pi}_{i \in n}\, \mathbf{p}_i) \parallel {}^{\bar{y}}/_{\bar{x}} : m \to n$ the corresponding concrete bigraph is the 5-tuple:

$$\mathbf{brs}(\mathbf{r}) = (dom(ctrl_{\mathbf{r}}), \bar{z}, ctrl_{\mathbf{r}}, prnt_{\mathbf{r}}, link_{\mathbf{r}}) : \langle m, \bar{x} \rangle \to \langle n, \mathsf{fn}(r) \rangle$$

Conversely, we can construct a BMC term from a pure bigraph as follows.

Definition 26 (BMC term corresponding to bigraph)**.** For a well-formed concrete bigraph $B = (V, E, ctrl, prnt, link) : \langle m, \bar{x} \rangle \to \langle n, \bar{y} \rangle$, we define a translation $\mathbf{bmc}(B)$ to a well-formed meta-calculus term:

$$\mathbf{bmc}(B) \stackrel{\text{def}}{=} (\nu E)(\prod_{i \in n} term(i) \parallel \prod_{i \in |\bar{x}|} {}^{link(x_i)}/_{x_i}) : m \to n$$

where

$$term(i) \stackrel{\text{def}}{=} \prod_{j \in prnt^{-1}(i)} \mathsf{c}_j(ns(j)).term(j) \mid hl(i)$$

with $\mathsf{c}_j = ctrl(j)$ and $ns(v) \stackrel{\text{def}}{=} link((v, 0)) \ldots link((v, ar(ctrl(v))$, and $hl(v) \stackrel{\text{def}}{=} \Pi_{i \in m} \boxed{i}$ where $prnt(i) = v$.

Finally, we can show that mapping a BMC term to bigraphs and back yields the same term up to structural congruence.

Proposition 2. For a normal form context $\mathbf{r} = (\nu \bar{z})(\mathbf{\Pi}_{i \in n} \mathbf{p}_i) \parallel {}^{\bar{y}}/_{\bar{x}} : m \to n$ it holds that $\mathbf{bmc}(\mathbf{brs}(\mathbf{r})) \equiv \mathbf{r}$.

Note that it the converse is not true, since mapping a bigraph to terms may throw away information about names in the interfaces.

3.3.1 Declarative Sortings

We can now give a formal interpretation of formulae in the sorting logic.

Definition 27 (Sorting logic interpretation)**.** We give a formal interpretation of formulae in terms of some bigraph $(V, E, ctrl, prnt, link) : \langle m, X \rangle \to$

$\langle n, Y\rangle$ relative to an assignment of variables to nodes $e : \mathbf{Var} \to V$:

$$
\begin{aligned}
[\![U^+ \implies U^-]\!]_e &\stackrel{\text{def}}{=} [\![U^+]\!]_e \implies [\![U^-]\!]_e \\
[\![F \wedge F']\!]_e &\stackrel{\text{def}}{=} [\![F]\!]_e \wedge [\![F']\!]_e \\
[\![F \vee F']\!]_e &\stackrel{\text{def}}{=} [\![F]\!]_e \vee [\![F']\!]_e \\
[\![v = u]\!]_e &\stackrel{\text{def}}{=} e(v) = e(u) \\
[\![v \neq u]\!]_e &\stackrel{\text{def}}{=} e(v) \neq e(u) \\
[\![v \mathbin{/} u]\!]_e &\stackrel{\text{def}}{=} prnt(e(u)) = e(v) \\
[\![v \mathbin{//} u]\!]_e &\stackrel{\text{def}}{=} prnt^+(e(u)) = e(v) \\
[\![v \not\mathbin{/} u]\!]_e &\stackrel{\text{def}}{=} prnt(e(u)) \neq e(v) \\
[\![v \not\mathbin{//} u]\!]_e &\stackrel{\text{def}}{=} prnt^+(e(u)) \neq e(v) \\
[\![v@i \neq u@j]\!]_e &\stackrel{\text{def}}{=} link(e(v), i) \neq link(e(u), j) \\
[\![\mathsf{c}(v)]\!]_e &\stackrel{\text{def}}{=} ctrl(e(v)) = \mathsf{c} \\
[\![\neg\mathsf{c}(v)]\!]_e &\stackrel{\text{def}}{=} ctrl(e(v)) \neq \mathsf{c} \\
[\![\neg local(v@i)]\!]_e &\stackrel{\text{def}}{=} \neg\exists z \in E : link(e(v), i) = z \\
[\![U \wedge U']\!]_e &\stackrel{\text{def}}{=} [\![U]\!]_e \wedge [\![U']\!]_e \\
[\![U \vee U']\!]_e &\stackrel{\text{def}}{=} [\![U]\!]_e \vee [\![U']\!]_e
\end{aligned}
$$

A bigraph $(V, E, ctrl, prnt, link) : \langle m, X\rangle \to \langle n, Y\rangle$ is determined to be well-sorted with respect to a formula F by universal quantification over all possible assignments of meta-variables to nodes: $\forall e : \mathbf{Var} \to V.\ [\![F]\!]_e$

3.3.2 Sorting Predicate Decomposability

We wish to characterise in a precise way the effect of sorting on the underlying categorical structure of bigraphs. For this we depend heavily upon the results in Debois' PhD thesis [35], and readers hoping for a detailed exposition of sorting in bigraphs should consult that work. We recall here the specific definitions that permit us to apply so-called *closure sortings* in this setting.

Definition 28 ([35] Decomposable and Deconstructable Predicates)**.** A predicate ϕ is *decomposable* iff $\phi(f \circ g)$ implies $\phi(f)$ and $\phi(g)$. A predicate ϕ that is decomposable and which reflects identities is called *deconstructable*.

Definition 29 (ϕ-respecting bigraphical reactive system)**.** Following [35], we say that any BRS $\mathbf{R}$ is ϕ-respecting iff every rule $(L, R, \eta) \in \mathbf{R}$ has both $\phi(L)$ and $\phi(R)$.

Definition 30 ([35] Sortings)**.** A sorting of some category C is a functor $F : X \to C$ that is faithful and surjective on objects. X is then the *sorted category*. We may abuse notation and elide the functor altogether, referring to the sorting as $X \to C$ alone.

Notation: We write $C(\cdot, c)$ to mean the class of morphisms of a category C that has codomain c, and $C(c, \cdot)$ for the set of morphisms of C that has domain c. We have chosen here to follow the non-standard notation of [35], although some readers may recognise these same constructions by the notation $Ob(C \downarrow c)$.

Definition 31 ([35])**.** For some category C, predicate ϕ, object c of C, and sets $F \subseteq C(\cdot, c)$ and $G \subseteq C(c, \cdot)$:

$$G \perp_{\phi,C,c} F \stackrel{\text{def}}{=} \forall f \in F, g \in G.\phi(g \circ f)$$

We elide the subscripts on $G \perp F$ where the intended category, predicate, and subscript are clear.

Definition 32 ([35] Closure Sorting)**.** Let C be a BRS category and ϕ be a deconstructable predicate on C. We define the closure sorting $\mathfrak{C}_\phi \to C$. The category $\mathfrak{C}_\phi$ has objects $(F, G)_c$ where $F \subseteq C(\cdot, c)$, $G \subseteq C(c, \cdot)$, $G \perp F$, and $1_c \in F \cap G$. We also define F and G to be *maximal* in the sense that there does not exist any additional $h \notin F \cup G$ that could be added to either of these sets without violating $G \perp F$. The morphisms of $\mathfrak{C}_\phi$ are $k : (F, G)_c \to (H, J)_d$ where $k : c \to d$ is a morphism of C and both

$$f \in F \implies k \circ f \in H \qquad \text{and} \qquad j \in J \implies j \circ k \in G$$

These definitions are almost verbatim from [35]. We observe for a given morphism $k : (F, G)_c \to (H, J)_d$, J can be thought of as the set of acceptable contexts for k, while F is the set of bigraphs for which k may be a context.

We recover three desirable properties of closure sortings; however, we merely assert these properties, as full characterisations and proofs are available in the work of both Debois [35] and Jensen [66]. Debois [35] establishes that for a category C with RPOs, and a deconstructable predicate ϕ, the sorted category $\mathfrak{C}_\phi$ has RPOs, and $\mathfrak{C}_p \to C$ transfers RPOs and has semantic correspondence. Informally, semantic correspondence is the property

that for any reaction of the reactive system of the sorted category, we can find a corresponding reaction in the unsorted category. We use this theorem directly to establish our ability to approximate the computation of sorting in a pragmatic manner, which makes implementation of the sorting logic and modular construction language a practical reality. Namely, we wish to show an equivalence between reaction in the sorted and unsorted categories, which will allow us to attempt reaction in the unsorted setting, and then check that the result still satisfies the predicate.

Theorem 1. We consider the two transition relations, $\xrightarrow{l}_{\mathfrak{C}}$ and $\xrightarrow{l}_{C}$, which indicate reaction in the sorted and unsorted categories, respectively. For a sorting $\mathfrak{C}_\phi \to C$ approximating a predicate ϕ, there is an equivalence:

$$f \xrightarrow{g}_{\mathfrak{C}} h \equiv f' \xrightarrow{g'}_{C} h'$$

where $\phi(h'), \mathfrak{C}(f) = f', \mathfrak{C}(g) = g',$ *and* $\mathfrak{C}(h) = h'$

Proof. Immediate from the semantic correspondence theorem [35]. □

Theorem 2. Any well-formed formula $\phi = \forall e : \mathbf{Var} \to V.\psi$ in our sorting logic is a deconstructable predicate that is decomposable such that $\phi(P \circ Q)$ implies $\phi(P)$ and $\phi(Q)$.

Proof. For the identity bigraph 1, there are no nodes over which to quantify, and so our predicate is satisfied immediately (and identities are respected). For some bigraph $R = (V, E, ctrl, prnt, link) : \langle m, X\rangle \to \langle n, Y\rangle$ for which $\phi(R)$, we consider the decomposition $R = P \circ Q$, and label the node-sets $V = V_P \uplus V_Q$. What then could cause $\neg\phi(P)$ or $\neg\phi(Q)$? Recalling that terms of our logic simplify to the fragment of the logic containing only negative terms (i.e., U and U^- from Def. 16), we proceed by cases over the constructs of our sorting logic.

Case ($\psi = v \,/\, u$,$\psi = v \,//\, u$)**.** For contradiction, assume $\neg\phi(P)$. We get $\exists u, v \in V_P : \neg\psi$, but we already know $V_P \subseteq V$ and $\forall u, v \in V : \psi$, and so we are done.

Case ($\psi = v \not\prec u$,$\psi = v \not\nearrow u$)**.** Decomposition cannot introduce new parent relations, so a node will not become a child or descendent of another if it was not already so prior to decomposition.

Case ($\psi = v@m \neq u@n$)**.** Decomposition cannot introduce further aliasing between names, only (potentially) differentiate them. Therefore given that ψ is already true for every u and v in V, these ports cannot reference a shared name in the decomposition that was not already present in $link_R$.

Case ($\psi = \neg local(v@i)$)**.** Decomposition cannot add a new edge or new link, and $link_P \subseteq link_R$.

Case ($\psi = \mathsf{c}(x), \neg\mathsf{c}(x)$)**.** Immediate, as $ctrl_P \subseteq ctrl$, and control assignments are not altered by decomposition.

Case ($\psi = F \wedge F', F \vee F'$)**.** By induction on the structure of sub-terms.

Having exhausted all cases, we have shown that any well-formed formula is deconstructable. □ □

3.4 Bigraphical Abstract Machine

In this section, we define the Bigraphical Abstract Machine (BAM), a general, graph-based abstract machine for bigraphical reactive systems, which can be instantiated for any instance BRS. By leveraging the mapping from terms to bigraphs given in Section 3.3, BAM can be applied to the bigraphical meta-calculus, and thus provides a graph-based abstract machine for any instance calculus. Moreover, the machine supports a general choice of reaction strategy, which can be instantiated to provide a fair abstract machine for any calculus, and which also extends to provide a stochastic abstract machine if reaction rules are extended with rates. To simplify the exposition we restrict attention to reaction rules in which no two holes are siblings.

BAM is based on a characterisation of matches of reaction rule redexes as particular graph embeddings.

Definition 33 (Redex Embedding)**.** For a ground bigraph $P : (0, \emptyset) \to (n', X)$ in BRS with active controls Σ_a and a reaction rule $R = (r_L, r_R : \langle m, Y\rangle \to \langle n, Z\rangle, \eta)$, define a *redex embedding* $R \hookrightarrow_{(e_p, e_l)} P$ to be given by two maps, the *place embedding* $e_p : n \cup V_{r_L} \to n' \cup V_P$, P (i.e., from the roots and nodes of the bigraph corresponding to the redex context r_L to the roots and nodes of the bigraph P), and the *link embedding* $e_l : Z \cup E_{r_L} \to X \cup E_P$ (i.e., from names and edges of the redex to those in the process), satisfying the following constraints

1. e_p is injective and e_l is injective on edges and maps edges to edges
2. $v \in e_p(m)$ and $v' = prnt^*(v) \in V_P$ implies $ctrl_P(v') \in \Sigma_a$
3. $v \in V_{r_L}$ implies $prnt(e_p(v)) = e_p(prnt(v))$, $ctrl(v) = ctrl(e_p(v))$, and $link(e_p(v)@i) = e_l(link(v@i))$

4. $|children(e_p(v))| > |children(v)|$ implies $\exists i \in m.prnt_{r_L}(i) = v$
5. $x \in E_{r_L}$ implies $e_l(x) \in scope(e_p(m))$

where $e_p(m) = \{e_p(i) \mid i \in m\}$, i.e., the image of the roots of the match, and $scope(U)$ for a set U of nodes and roots of P is the set of edges that are only linked to nodes that are elements in or descendants of an element in U. For a subset of nodes $V' \subseteq V_P$ of the place graph let $V'\downarrow = \{v \in V_P \mid \exists k.prnt^k(v) \in V'\}$. We define the *redex parameters* as $D_{(e_p,e_l)} = \mathbf{\Pi}_{i\in m}\,\mathbf{bmc}(d_i)$ and $d_i = (V', \emptyset, ctrl_{P|V'}, prnt_{P|V'} \cup \{(v, \{1\}) \mid prnt(v) = e_p(i)\}, link')$ for $V' = \{e_p(i)\}\downarrow \setminus e_p(V_{r_L})$ and $link'((v,i) = link_P((v,i))$ if $link_P((v,i)) \in X \cup (E \setminus e_l(E_{r_L}))$ and $link'((v,i) = e \in E_{r_L}$ if $e_l(e) = link_P((v,i))$

We define $\mathbf{Emb}(\Theta, \mathcal{R})$ to be the set of all such embeddings for a sorted signature Θ and a set of reaction rules $\mathcal{R}$.

Proposition 3 (Redex Embedding and BRS Matching correspondence)**.** For a ground bigraph $P : (0, \emptyset) \to (n', X)$ and a reaction rule $R = (r_L, r_R : \langle m, Y\rangle \to \langle n, Z\rangle, \eta)$, there is a *match* $P \equiv \mathbf{brs}(C \circ r_L \circ D)$ if and only if there is a redex embedding $R \hookrightarrow_{(e_p,e_l)} P$ such that $D = D_{(e_p,e_l)}$, and $C = (V', E_P \setminus e_l(E_{r_L}), ctrl_{P|V'}, prnt_{P|V'} \cup \{(i, e_p(i)) \mid i \in n\}, link')$ for $V' = V_P \setminus e_p(V_{r_L})\downarrow$ and $link'(v,i) = link_P(v,i)$ if $v \in V'$ and $i \in ar(vtrl_P(v))$.

Example 5 (Redex Embedding for $\mathbf{Pi}_{sr}$)**.** Given a ground bigraph

$$B = \mathbf{brs}((\nu a)(\mathsf{send}\langle a, b\rangle \mid \mathsf{recv}\langle a, c\rangle \mid \mathsf{recv}\langle a, d\rangle))$$

belonging to $\mathbf{Pi}_{sr}$, and a redex $r_L = \mathsf{send}\langle x, y\rangle.\boxed{0} \mid \mathsf{recv}\langle x, z\rangle.\boxed{1}$, we can exemplify one redex embedding. The redex bigraph $\mathbf{brs}(r_L) : \langle 2, \emptyset\rangle \to \langle 1, \{x, y, z\}\rangle$ is given by:

- $V_{r_L} = \{u_0, u_1\}$
- $E_{r_L} = \emptyset$
- $ctrl_{r_L} = \{u_0 \to \mathsf{send}, u_1 \to \mathsf{recv}\}$
- $prnt_{r_L} = \{\boxed{0} \to u_0, \boxed{1} \to u_1, u_0 \to 0, u_1 \to 0\}$
- $link_{r_L} = \{((u_0, 1), x), ((u_0, 2), y), ((u_1, 1), x), ((u_1, 2), z)\}$

The bigraph $P : \langle 0, \emptyset\rangle \to \langle 1, \{b, c, d\}\rangle$ is given by:

- $V_P = \{v_0, v_1, v_2\}$

- $E_P = \{a\}$
- $ctrl_P = \{v_0 \to \mathsf{send}, v_1 \to \mathsf{recv}, v_2 \to \mathsf{recv}\}$
- $prnt_P = \{v_0 \to 0, v_1 \to 0, v_2 \to 0\}$
- $link_P = \{((v_0, 1), a), ((v_0, 2), b), ((v_1, 1), a), ((v_1, 2), c), ((v_2, 1), a), ((v_2, 2), d)\}$

One redex embedding is $r_L \hookrightarrow_{(e_p,e_l)} P$, where $e_p = \{0 \to 0, u_0 \to v_0, u_1 \to v_2\}$ and $e_l = \{x \to a, y \to b, z \to d\}$.

Definition 34 (Redex Embedding Reaction)**.** For a ground bigraph $P : (0, \emptyset) \to (n', X)$ and a redex embedding $R \hookrightarrow_{(e_p,e_l)} P$ for a reaction rule $R = (r_L, r_R : \langle m, Y\rangle \to \langle n, Z\rangle, \eta)$, define the *redex embedding reaction* $P \to_{(e_p,e_l)} P'$ where

- $V_{P'} = V_P \setminus e_p(V_{r_L})\!\downarrow \cup V_{R'}$
- $E_{P'} = E_P \setminus e_l(E_{r_L}) \cup E_{R'}$
- $ctrl_{P'} = ctrl_{P|V_{P'}} \cup ctrl_{R'}$
- $prnt_{P'} = prnt_{P|V_{P'}} \cup prnt_{R'} \setminus n \cup \{(v, e_p(i)) \mid i \in n \text{ and } prnt_{R'}(v) = i\}$
- $link_{P'}(v, i) = link_P(v, i)$ if $v \in V_P \setminus e_p(V_{r_L})\downarrow$ and $link_{P'}(v, i) = link_{R'}(v, i)$ if $v \in V_{R'}$

where $R' \equiv \mathbf{brs}(r_R \circ \eta(D_{(e_p,e_l)}))$ and the nodes of the place graph in R' is chosen to be fresh with respect to P, i.e., $V_{R'} \cap V_P = \emptyset$

From Prop. 3, Def. 34 and the definition of bigraph composition in [77] we can now prove correspondence between redex embedding reactions and BRS reactions.

Theorem 3 (Soundness and Completeness)**.** For a ground bigraph $P : \langle 0, \emptyset\rangle \to \langle n', X\rangle$ and a reaction rule $R = (r_L, r_R : \langle m, Y\rangle \to \langle n, Z\rangle, \eta)$ then $P \equiv \mathbf{brs}(C \circ r_L \circ D)$ if and only if there exists a redex embedding $R \hookrightarrow_{(e_p,e_l)} P$ such that $P \to_{(e_p,e_l)} C \circ r_R \circ \eta(D)$.

We define well-sorted redex reactions just as for standard bigraph reactions in Sec. 3.2.4.

Definition 35 (Well-sorted Redex Embedding Reaction and Redex Embeddings)**.** For a redex embedding reaction $P \rightarrow_{(e_p,e_l)} P'$ and a sorting predicate ϕ, we define a well-sorted redex embedding reaction to be that for which $\phi(P)$ implies $\phi(P')$. We say that an embedding $R \hookrightarrow_{(e_p,e_l)} P$ is well-sorted if the redex embedding reaction $P \rightarrow_{(e_p,e_l)} P'$ is well-sorted. Let $E(P) = \{e \mid r_L \hookrightarrow_e P \wedge P \rightarrow_e P' \wedge \phi(P) \implies \phi(P')\}$ be the set of all well-sorted embeddings.

The redex embedding reactions preserve node identities for all nodes that are not included in the redex (or its parameters) and introduces fresh node identities for the reactum (and parameters). This allows us to define strong and weak fairness for infinite execution sequences in the standard way (and in particular consistent with Bidinger & Compagnoni [19]).

Definition 36 (Fairness)**.** For a ground bigraph P_0, we say that an infinite well-sorted reaction sequence $P_0 \rightarrow_{e_1} P_1 \rightarrow_{e_2} P_2 \ldots$ is *strongly fair* if for any strictly increasing sequence $(u_i)_{i\in\omega}$ we have $\bigcap_{i\in\omega} E(P_{u_i}) = \emptyset$. It is *weakly fair* if for any $k \in \omega$ we have $\bigcap_{i\geq k} E(P_i) = \emptyset$.

As a consequence of the definition of reaction rules, weak and strong fairness always coincide in our setting (as is also the case for the Pict abstract machine in [19]), so we will simply refer to strongly/weakly fair sequences as fair.

In order to identify fair strategies for our abstract machine, we define the standard notion of concurrent redex embeddings as the embeddings for which the redexes instantiated with parameters do not overlap.

Definition 37 (Concurrent Redex Embeddings)**.** For a ground bigraph P, any two redex embeddings $R \hookrightarrow_{(e_p,e_l)} P$ and $R' \hookrightarrow_{(e'_p,e'_l)} P$ are defined to be *concurrent* iff $Cod(e_p)\downarrow \cap Cod(e'_p)\downarrow = \emptyset$ We write $(e_p, e_l) \mid\mid (e'_p, e'_l)$ to mean these embeddings are concurrent.

We note that two concurrent redex embedding reactions are concurrently enabled and can be performed in any order leading to the same end process.

Lemma 1 (Application of Concurrent Embeddings is Commutative)**.** For any two redex embeddings $R \hookrightarrow_e P$ and $R' \hookrightarrow_{e'} P$, having $e \mid\mid e'$, then $P \rightarrow_e P' \rightarrow_{e'} P''$ and $P \rightarrow_{e'} Q' \rightarrow_e Q''$ implies $P'' \equiv Q''$.

We now define a strategy as a function that for any ground process in a BRS returns a subset of concurrently enabled redex embeddings.

Definition 38 (Strategy)**.** Define a *strategy* to be a mapping $\mathbf{S} : \mathbf{BMC}(\Theta) \to \mathcal{P}(\mathbf{Emb}(\Theta, \mathcal{R}))$, such that $\mathbf{S}(P) \subseteq E(P)$, for all $e \neq e' \in \mathbf{S}(P)$ we have $e \mid\mid e'$, and $E(P) \neq \emptyset \implies \mathbf{S}(P) \neq \emptyset$.
We say that a strategy is *maximal* if for any $e \in E(P) \setminus \mathbf{S}(P)$ and $e' \in \mathbf{S}(P)$ we have $e \not\mid e'$.

Definition 39 (BAM)**.** For a sorted BRS signature $\Theta = (\Sigma, \phi)$ with reaction rules $\mathcal{R}$, we define the operation of $\mathbf{BAM}(\Theta, \mathcal{R}, \mathbf{S}, p) \to \mathbf{BAM}(\Theta, \mathcal{R}, \mathbf{S}, p')$ where $\mathbf{S}$ is the strategy and p a ground process, and p' is the result of applying all the matches in $\mathbf{S}(p)$ to p (in any order).

Proposition 4. Any maximal strategy gives rise to only fair execution sequences.

3.4.1 Stochastic Simulation with BAM

We note that since the machine at each step computes the set of all possible reactions, it is a relatively simple matter to support stochastic simulation, by augmenting reaction rules with *rates*, that determine the frequency of a given rule firing. We now sketch a stochastic version of BAM, relying on the embeddings-based approach of Højsgaard & Krivine [62] for scalable simulations. Our goal is not necessarily to provide an efficient or scalable mechanism for stochastic simulation using bigraphs (which has been well-studied in [62]); rather, we aim to demonstrate that as a piece of theoretical machinery, BAM may act in a unifying capacity for the further study of different execution models for bigraphs.

We reiterate here several of the definitions from [62], expressed in the setting of BAM. The details of the exact simulation procedure (Gillespie's algorithm), and the connection to the stochastic bigraphs of Krivine et al. [68] are also given in that work.

Definition 40 (Stochastic Reaction Rules)**.** To each reaction rule R, we assign a rate constant ϱ_R, such that the frequency of reaction will be proportional to this rate constant, divided by the number of concurrently-enabled instances of that rule.

Definition 41 (Reaction Rule Activity)**.** For a given reaction rule R and a rate ϱ_R, we can compute the *activity* α_R for that reaction rule, based on the number of embeddings of that redex present in a given process p:

$$\alpha_R(p) \stackrel{\text{def}}{=} |E_R(p)| \times \varrho_R$$

where E_R gives the set of concurrently-enabled embeddings of R into p.

Definition 42 (System Activity)**.** The total activity of a system is given as α with respect to a process p and a set of reaction rules $\mathcal{R}$:

$$\alpha_{\mathcal{R}}(p) \stackrel{\text{def}}{=} \sum_{R \in \mathcal{R}} \alpha_R(p)$$

For a given reaction rule $R \in \mathcal{R}$, the probability of it being selected is given by:

$$\frac{\alpha_R(p)}{\alpha_{\mathcal{R}}(p)}$$

Then for a given match $m \in E_R(p)$, the probability is given by:

$$\frac{1}{|E_R(p)|}$$

These two values, taken together, are used to compute a Monte Carlo step for a given time increment. Such an execution strategy would therefore conform to the definition of stochastic execution for bigraphs in terms of embeddings given by Højsgaard & Krivine [62].

3.5 Related Work

In one of his earliest works on bigraphs [77], Milner proposed a term language for bigraphs with normal forms. We have made minor deviations from Milner's syntax, but these changes have no effect on the expressivity of the language, while making the representation of terms in calculi such as the π-calculus more compact and familiar.

Milner's encoding of the π-calculus [83] relied upon the introduction of *binding* bigraphs [78] as a variant of bigraphs, in order to accurately represent the scope conditions for names in the π-calculus. Introducing additional constructs into the theory has disadvantages — especially as further variations upon bigraphs emerge — and so we aim to avoid this in our approach. By using declarative specifications of predicate sortings, we can recover (for example) the π-calculus scope condition within the usual bigraphs setting, without needing to introduce new constructs that extend the formalism.

Our approach to bigraphical sortings relies heavily upon the *closure sorting* proposed by Birkedal et al. [21] and developed further in Debois' PhD thesis [35]. This current work is a continuation of the work on closure sortings, in the hope that we can advance the state of the art in this area sufficiently to make practical tool support a realistic possibility.

Conforti et al. [30] proposed *BiLog*, a spatial logic for expressing properties over arbitrary bigraphs. BiLog is very general, and it is not the case that any well-formed formula in BiLog will necessarily result in a deconstructable predicate (and therefore a well-behaved sorting implementing this predicate). We have taken a different approach in defining a minimal logic for describing predicates for sortings that will always result in a deconstructable predicate, and which is at least sufficient to implement those sortings from the literature identified by Debois as being replaceable with closure sortings [35].

Bacci and Grohmann [9] characterised the decidability of Debois' closure sortings, and proposed *match sortings* as a tractable subset. The sorting logic we have presented and match sortings both permit recovery of *local bigraphs* [82], and excluding sortings that specify interfaces, our sorting logic can be used to specify the same kind of properties (i.e., forbidding the presence of a particular match anywhere within a term).

For the design of the bigraphical abstract machine (BAM), we have departed from previous efforts to describe abstract machines for families of nominal calculi (including the Fusion machine [44] and the Pict abstract machine [19]) by defining a general graph-based abstract machine that can be specialised to any instance calculus.

The characterisation of matching as embeddings of redexes into processes in BAM is similar to that given by Højsgaard [61] to enable scalable simulation of bigraphical languages. The similarities provide some promise that the simulation implementation based upon embeddings in that work could be related as an instance of BAM.

3.6 Conclusion and Future Work

We have given a core bigraphical meta-calculus (BMC) along with a new XPath-like bigraph logic employed to declare sortings when defining instances of BMC. We have demonstrated that the logic intuitively captures both grammatical constraints, e.g., limiting replication in the π-calculus to only input guarded processes (as in [19]), as well as the scope condition for the binding input parameter in the π-calculus. By describing the scope condition as part of the sorting, BMC can be defined as a very simple core process calculus, which we have shown corresponds in a precise way to Milner's pure bigraphs. Moreover, we proved that any property described in the logic is decomposable (as defined by Debois et al.). This means that the meta-calculus corresponds to a closure-sorted bigraphical reactive system,

for which it was demonstrated in [35] that a labelled transition bisimulation congruence can be derived, using the general theory of RPOs for bigraphical reactive systems.

We have also shown that it is not necessary to make the name and sorting constraints explicit in interfaces. It is possible to work with the un-typed process calculus terms, and to simply restrict reactions to those that preserve the sorting constraint. We then provided a general bigraphical abstract machine (BAM), which can be instantiated with any instance calculus of BMC. We prove that any maximal evaluation strategy for BAM gives rise to a fair execution strategy.

The BMC meta-calculus thereby enables the familiar presentation and manipulation of a broad class of sorted bigraphical reactive systems as simple process calculus terms which can be executed and simulated using BAM. In particular, our work provides a new fair abstract machine for the π-calculus and mobile ambient calculus, which can be used without requiring any knowledge of the categorical foundations of bigraphical reactive systems. The results therefore significantly advances both the theory and applicability of bigraphs.

We consider it interesting future work to attempt to fully recover stochastic bigraphs within the context of BAM, and to explore the representation of a number of other nominal calculi previously described as bigraphical reactive systems in this setting, so as to demonstrate more fully the general applicability of this approach for the study and analysis of nominal calculi.

Acknowledgements

We would like to thank Marino Miculan, Søren Debois, and Taus Brock-Nannestad for their helpful comments at various stages of this work. This work was funded in part by the Danish Research Agency (grant no.: 2106-080046) and the IT University of Copenhagen (the Jingling Genies project).

Chapter 4

Bigraphical Refinement

Context

> *This chapter is derived from the paper* Bigraphical Refinement *by Gian Perrone, Søren Debois, and Thomas Hildebrandt which was accepted for publication in the Proceedings of the 2011 International Workshop on Refinement (Refine'11), EPTCS 55, 2011, pp. 20-36.*

The version that appears in this thesis has been reformatted, and features an extension from the original in that it permits refinements to be constructed up to the *stuttering closure* of a set of traces. This is a more liberal notion of refinement than that proposed in the Refine'11 workshop, though it represents only a small change from the definition of refinement given in that work. The definition of stuttering is included as an addendum to the original version of the paper in Section 4.7, so as to avoid confusion between the version appearing in this thesis and the published version of the paper.

Refinement is a central concept within formal software development practice, and *vertical* refinement is a special form of refinement that relates entire different languages. This seems a useful and central concept in advancing the vision of a "tower of models", in that we require some formal means of specifying the relationships between the different languages that constitute the pieces of a given tower. While bisimulation is well-defined for bigraphs, refinement had not been explored previously. Vertical refinement has certain pragmatic benefits — it's a much weaker property (e.g., it does not require symmetry in the relation), and therefore it seems easier to construct relations of this type.

As an overall contribution to the theory and applicability of bigraphical reactive systems, the notion of bigraphical refinement appears to have enabled the further use of bigraphs already, and seems likely to continue to do so. The notion of refinement proposed in the Refine'11 workshop paper has been employed to relate different bigraphical formalisations of Petri nets to one another [102], and to relate different models of indoor environments [54]. As such, it appears that the sufficient condition on abstraction functors introduced in this work is a useful contribution, as these very conditions have been used in the cited works to establish that a safe refinement exists—giving credence to the claim made in the workshop paper that this is a considerably easier property to establish than the more general "weak preservation of reaction" condition.

Abstract

We propose a mechanism for the vertical refinement of bigraphical reactive systems, based upon a mechanism for limiting observations and utilising the underlying categorical structure of bigraphs. We present a motivating example to demonstrate that the proposed notion of refinement is sensible with respect to the theory of bigraphical reactive systems; and we propose a sufficient condition for guaranteeing the existence of a safety-preserving vertical refinement. We postulate the existence of a complimentary notion of horizontal refinement for bigraphical agents, and finally we discuss the connection of this work to the general refinement of Reeves and Streader.

4.1 Introduction

Refinement is the process of gradually developing a specification towards a suitable implementation, through a series of steps in which more concrete entities are shown to be as acceptable as the more abstract entities preceding it in the chain of refinement steps, based upon what may be observed of these entities. The utility of this method has been demonstrated through many years of application in academic and industrial settings. In this paper we attempt to bring these well-studied benefits to a new class of systems — namely, bigraphical reactive systems. We focus primarily on *vertical refinement* [23], where the aim is to relate models constructed with respect to different semantics.

A *bigraphical reactive system* [83, 81] (BRS) is a model construction paradigm proposed by Milner and colleagues that aims to enable modelling of interactive systems within a cohesive theoretical framework. While the primary long-term focus of bigraphs is on models of ubiquitous and context-aware systems [20], they have demonstrated value in other areas such as biological applications [68, 33, 34] and business processes [56, 105]. Bigraphical reactive systems also capture the syntactic and semantic structure of many formalisms associated with process modelling, providing a unifying meta-calculus within which to relate many of these well-developed theories. Already encodings into various bigraphical reactive systems have been demonstrated for amongst others the λ-calculus [82], CCS [81], the Mobile Ambients calculus [64], several variants of the π-calculus [64, 26, 39], Fusion Calculus [50] and Petri Nets [71].

Bigraphical reactive systems consist of two graphs (hence the name *bi*graph) modelling the orthogonal notions of *locality* and *connectivity* which

together capture the static structure of a system, and a set of *reaction rules* that may selectively rewrite portions of the bigraph in order to capture the dynamic behaviour of that system. We will introduce bigraphs and bigraphical reactive systems (assuming no prior knowledge) in Section 4.2.

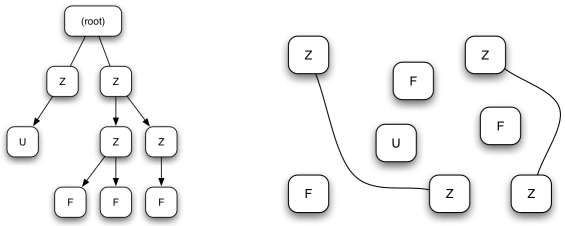
The usual notion of "observation" in a BRS is derived from the above notion of dynamic behaviour: a BRS gives rise to an LTS, the labels of which are simply the least context enabling reaction. The present effort towards refinement takes this connection between static structure and dynamic behaviour to heart, and attempts to short-circuit the LTS in favour of a more directly structural mechanism of refinement. This makes sense uniquely for bigraphs exactly because of the close correspondence between structure and dynamics. The primary contribution of this paper is to introduce such a mechanism as a small step towards bringing the well-established benefits of refinement to models constructed within the bigraph formalism. Additionally, we give a sufficient condition for an abstraction functor (Section 4.4) to give rise to a safe refinement, and show that this notion of refinement corresponds with (and indeed, in part is an instance of) the general refinement of Reeves and Streader [94, 95].

4.1.1 Structure of the paper

The remainder of this paper is structured as follows: We review bigraphs (assuming no prior knowledge) in Section 4.2. In Section 4.3 we introduce a running example that will be used to illustrate all of the concepts presented. In Section 4.4 we present our definition of vertical refinement for bigraphical reactive systems and show that the proposed refinement preserves safety properties with respect to the abstraction functor upon which it is parametrised. Additionally, we present a sufficient condition for an abstraction functor to give rise to a safe refinement. Finally, in Section 4.5 we discuss a candidate horizontal refinement mechanism for bigraphical agents, derived from the general refinement of Reeves and Streader [94, 95], and discuss the connection of this work to general refinement.

4.2 Bigraphical Reactive Systems

Bigraphical reactive systems is a graphical formalism emphasising the orthogonal notions of *locality* and *connectivity*. A BRS is a category of bigraphs and a set of reaction rules that may be applied to rewrite these bigraphs. We provide here a short, informal introduction to the anatomy



(a) Place Graph (b) Link Graph

Figure 4.1: The constituent place (Fig. 4.1a) and link (Fig. 4.1b) graphs that form a particular bigraph.

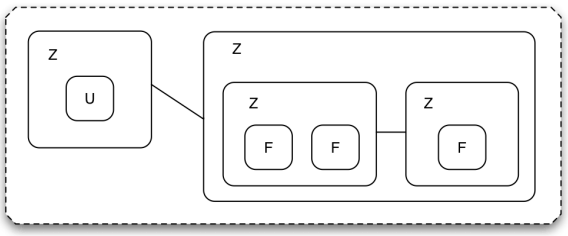


Figure 4.2: The bigraph resulting from the combination of the place and link graphs in Fig. 4.1a and Fig. 4.1b. This bigraph is an agent of the BRS_{notify} example BRS with signature $\Sigma = \{\mathsf{Z}, \mathsf{U}, \mathsf{F}, \mathsf{N}\}$ that we will introduce in Section 4.3.

of a BRS without assuming any prior knowledge. For a complete treatment of bigraphs and BRSs, readers are referred to [83, 81].

4.2.1 Static Structure

The most basic construction within the static fragment of bigraphical reactive systems is the *node*. This follows from normal definition of a node within graph theory. To nodes we assign *controls*, which are drawn from a *signature* Σ, the set of controls. We sometimes use a convenient shorthand such that we may refer to a node as being an "X node", by which we really mean a node that has been assigned the control X. Nodes may be nested to arbitrary depth to form a tree that is known as the *place graph* (Fig. 4.1a). We represent this nesting by containment, as shown in Fig. 4.2. We distinguish between controls of two kinds: *active* and *passive* ones; we

shall see later how active controls admit dynamic behaviour beneath them whereas passive controls do not. Every tree of nodes is contained by a *region* (the dotted border in Fig. 4.2). Bigraphs permit multiple regions (a place forest).

To controls (and therefore nodes) we assign a fixed *arity*, which defines the number of *ports* that a given node possesses. A port is a connection point on a node; it must always be connected to other such connection points by the *link graph*. The link graph (Fig. 4.1b) is an undirected hypergraph over the ports of the nodes of the place graph. A single (hyper) edge may connect arbitrarily many ports on different nodes.

Within the place graph, in addition to regions and nodes, there may also exist *holes* (known as *sites* in some bigraphs literature), which are expressed visually as shaded grey nodes (as in Fig. 4.3a). A hole is a location into which a region of another bigraph may be inserted by composition. It may be helpful to think of bigraphs with holes as "contexts" and those without as "processes" or "terms".

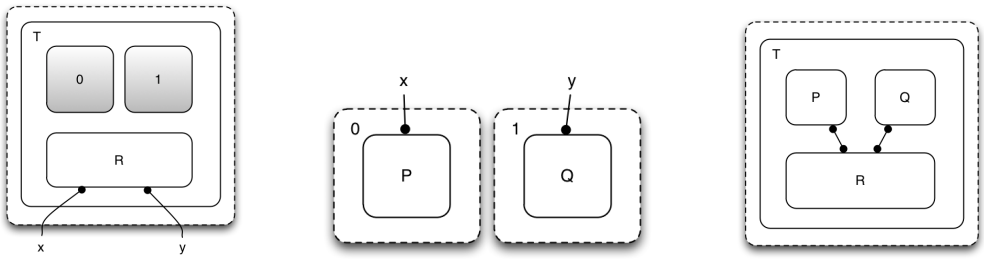
Present also within Fig. 4.3 are *names* that represent (named) points at which links of the link graph may be fused to form a single (hyper) edge. In the intuition of contexts and terms, names of bigraphs roughly correspond to unstructured names, as in the π-calculus. By convention, *outer names* are drawn upwards, and *inner names* are drawn downwards. Outer names are analogous in the link graph to regions in the place graph, while inner names are analogous to holes. Through composition of link graphs, sets of inner and outer names that agree are matched and joined.

Definition 43 (Interface)**.** An interface is a pair $\langle j, X\rangle$ where $0 \leq j$, indicating the number of holes or regions, and X is a set of (inner or outer) names.

Definition 44 (Bigraph)**.** A bigraph is a 5-tuple:

$$(V, E, ctrl, prnt, link) : \langle k, X\rangle \to \langle m, Y\rangle$$

Here V is the set of nodes, E is the set of hyperedges, *ctrl* is the *control map* that assigns controls (and therefore arities) to nodes, *prnt* is the *parent map* that defines the tree structure in the place graph and *link* is a link map that defines the link structure. The inner interface $\langle k, X\rangle$ indicates that the bigraph has k holes, and a set of inner names X. The outer interface $\langle m, Y\rangle$ indicates that the bigraph has m regions and a set of outer names Y.



(a) $A : \langle 2, \{x, y\}\rangle \to \langle 1, \emptyset\rangle$ (b) $B : \langle 0, \emptyset\rangle \to \langle 2, \{x, y\}\rangle$ (c) $A \circ B : \langle 0, \emptyset\rangle \to \langle 1, \emptyset\rangle$

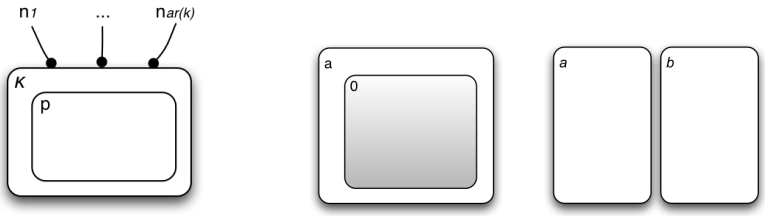
Figure 4.3: The composition of two bigraphs A and B with their respective interfaces

Definition 45 (Composition)**.** Bigraphs are composed separately in the place and the link graphs. The interfaces of the bigraphs must be compatible in order for composition to be defined, i.e., the sets of names and the number of regions/holes must be the same. Fig. 4.3 illustrates the composition $A \circ B$ of bigraphs A and B. In the place graph, we insert contents of the left-most region of B into hole 0 of A, and the contents of the right-most region of B into hole 1 of A. Regions are numbered left-to-right: we insert the contents of region 0 into hole 0 etc. In the link graph, links are spliced together where there is name agreement between the inner and outer names of the bigraphs being composed. We may refer to A in this case as being a *context* into which B is inserted.

Definition 46 (Tensor Product)**.** There exists an additional way in which to combine bigraphs, namely the *tensor product* $A \otimes B$, where A and B are bigraphs. Where A and B do not share any inner or outer names, this just involves juxtaposing their place graphs, taking the union of their names, and increasing the indices of holes in B to make them unique with respect to A. This definition obscures some technical details. It is recommended that readers interested in following the proofs in Section 4.4.1 refer to [83] for a precise definition.

4.2.2 Notation

We introduce a rudimentary term language for representing bigraphs that should be familiar to most readers accustomed to the notation for process



(a) $\kappa(n_1, \ldots, n_{ar(\kappa)}).p$ (b) $a.\boxed{0}$ (c) $a.\mathsf{nil} \mid b.\mathsf{nil}$

Figure 4.4: Example bigraph terms with their associated graphical representation

algebras. The present language is not complete, i.e., it cannot express every bigraph, but it can express the ones we will use in examples. It is a subset of a complete such language [79]. We will use this term language in conjunction with the graphical representation used in Fig. 4.2.

Definition 47 (Bigraph Term Language)**.**

$$p ::= \kappa(n_1, \ldots, n_{ar(\kappa)}).p \mid p \mid p \mid \boxed{i} \mid \mathsf{nil}$$

Where $\kappa \in \Sigma$.

The term language requires some explanation — $\kappa(n1, \ldots, n_{ar(\kappa)}).p$ is *prefixing* (Fig. 4.4a), indicating a node assigned the control κ. The arity of κ is given by $ar(\kappa)$. The sequence $n_1, \ldots, n_{ar(\kappa)}$ are the ports of the node. Finally, the suffix p is the term that is nested inside this node. $p \mid p$ is *juxtaposition* of terms (Fig. 4.4c), placing them as siblings within the place graph. $\boxed{i}$ is a hole (Fig. 4.4b), indexed by some integer $0 \leq i$. Finally, nil is the nil terminator which is simply the empty bigraph.

4.2.3 Dynamics

Having introduced the basic structure of bigraphs, the static portion of a BRS, we now introduce the *reactive* portion of a BRS that imbues a system with dynamic behaviour. This relies on *reaction rules* that define rewriting that may be applied to a bigraph. A reaction rule (R, R', η) consists of a *redex* R, a *reactum* R' and an *instantiation map* η, where the redex is a bigraph to be matched and the reactum is the bigraph with which the matched portion of the bigraph should be replaced. The instantiation map

indicates how parameters matched by holes in the redex should manifest in the reactum after matching. Where the instantiation map is unambiguous (e.g., it is the identity map), we may just write $R \to R'$.

Definition 48 (Reaction)**.** Matching of a particular reaction rule (R, R', η) against a particular bigraph G and rewriting it into some other bigraph G' proceeds by decomposition of the bigraph into a *context* C, a *match* R (the redex), and a set of *parameters* d (for portions of the bigraph that are matched by holes in the redex). This decomposition is then reassembled with the reactum R' replacing the matched portion of G, with select parts of d substituted into the holes of R', forming the resulting bigraph G'.

$$G = C \circ R.d \to C \circ R'.\eta(d) = G'$$

We require further that the context C be *active*, that is, that every control above holes of C are active (see CCS example below).

We have suppressed details of the handling of names here by using the notation "$R.d$"; we have also suppressed details in the phrase "with select parts of d" and not explained the use of the map η. We refer the reader to [83] or [81] for details. The present paper can be read without understanding these details, as reaction in our examples always take the form of the following special case:

$$a = C \circ R \circ d \to C \circ R' \circ d\ .$$

Definition 49 (Bigraphical Reactive System)**.** We use the notation $BG(\Sigma, \mathcal{R})$ to denote a bigraphical reactive system with a *signature* Σ (the set of constituent controls), and a set of reaction rules $\mathcal{R}$. More formally, $BG(\Sigma, \mathcal{R})$ is an *spm category* [83] in which the objects are interfaces and the arrows are bigraphs (which we refer to as *agents* of $BG(\Sigma, \mathcal{R})$), equipped with a set of reaction rules $\mathcal{R}$.

As an example, we introduce a very simple calculus in the style of the Calculus of Communicating Systems (CCS) [74], where we first give an encoding of the terms as bigraphs, and then define a reaction rule that imbues these terms with dynamic behaviour. Interested readers are referred to [83] for a real encoding of CCS.

Our calculus defines sequencing ($t.P$), parallel composition ($t \mid t$), and sending and receiving on a named channel ("$x!$" and "$y?$", respectively, where x and y are channel names). The encoding of these constructs into the bigraphical term language in Definition 47 is straightforward — these

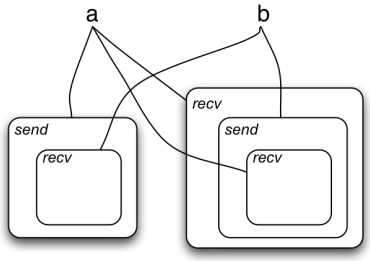


Figure 4.5: The process $\mathsf{send}(a).\mathsf{recv}(b).\mathsf{nil} \mid \mathsf{recv}(a).\mathsf{send}(b).\mathsf{recv}(a).\mathsf{nil}$

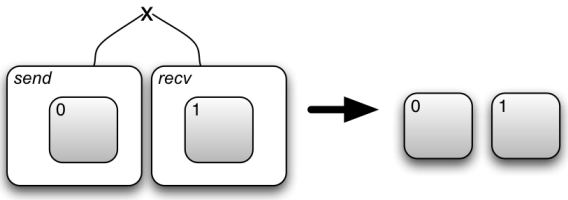


Figure 4.6: The R_{CCS} reaction rule

primitives are already defined in terms of the bigraphical term language, except for "send" and "receive" which we straightforwardly encode as nodes with controls send and recv, each with arity 1. Fig. 4.5 gives a graphical representation of the process $\mathsf{send}(a).\mathsf{recv}(b).\mathsf{nil} \mid \mathsf{recv}(a).\mathsf{send}(b).\mathsf{recv}(a).\mathsf{nil}$. According to our encoding, sequencing is represented by prefixing, parallel composition by juxtaposition, actions (such as send and recv) by *passive* controls, and channels by outer names. This is by no means the only encoding possible, but this technique is one of the most straightforward.

Having developed the encoding of our calculus within bigraphs, we can give a reaction rule R_{CCS} that will (through repeated rewriting) reduce the term as far as possible based upon agreement between parallel processes as to which action should be taken next:

$$R_{CCS} \stackrel{\text{def}}{=} \mathsf{recv}(x).\boxed{0} \mid \mathsf{send}(x).\boxed{1} \rightarrow \boxed{0} \mid \boxed{1}$$

This rule is presented graphically in Fig. 4.6. It essentially "peels off" the outer layers of the terms where a send and a recv action are linked to the same channel name, rewriting the entire bigraph to the juxtaposition of whatever was nested inside those send and recv controls (i.e. the parts of the bigraph matched by the holes in the redex). As an example, the CCS

reaction $a!.b? \mid a?.c! \to b? \mid c!$ becomes the bigraphical reaction

$$\mathsf{send}(a).\mathsf{recv}(b).\mathsf{nil} \mid \mathsf{recv}(a).\mathsf{send}(c).\mathsf{nil} \to \mathsf{recv}(b).\mathsf{nil} \mid \mathsf{send}(c).\mathsf{nil}$$

4.3 Example

Aside from their role as a meta-calculus for the study of process modelling formalisms, bigraphical reactive systems are intended to provide a basis upon which to construct models of the kinds of context-aware and ubiquitous systems that are becoming increasingly popular. Consequently, we introduce an example based on modelling a context-aware social network notification system, such that a user is notified whenever a friend is in the same physical location.

We will give this example without using the link-graph part of bigraphs to keep it simple. We emphasise that the example generalises to a more interesting one in which connectivity counts — where notification is dependent not only on physical co-location but also on whether or not users and friends are virtually connected through their laptops and phones.

We will subsequently extend this to a system in which not all friends, but rather only particular designated "special friends", trigger notifications, and show that (and in what sense) the latter system is a refinement of the former.

The example system captures the dynamics of some physical environment (consisting of discrete zones within which we can detect the presence of a user by some mechanism that is outside the scope of this model) in which a user's friends move from zone to zone. When one of the user's friends is present in the same zone as the user, a notification is given, modelled by adding a "notification" node to the zone.

4.3.1 The abstract system: BRS_{notify}

We first define controls Z (Zone), U (User), F (Friend), N (Notification) and S (Special friend marker). Every control has arity 0 and every control is active; altogether we have a signature

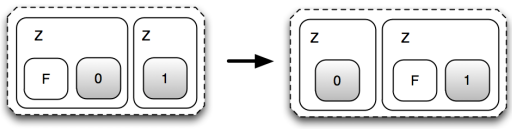
$$\Sigma_N = \mathsf{Z}, \mathsf{U}, \mathsf{F}, \mathsf{N}$$

The bigraphs of our systems are thus arbitrary trees over these controls. We shall of course be interested only in those where Z are inner nodes and the remaining controls are leaves.

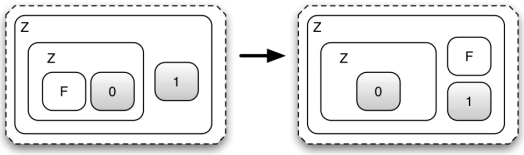
With these particular bigraphs in mind, we give reaction rules reconfiguring a bigraph by allowing nodes with control F — friends — to move between nested zones as follows. These rules are illustrated graphically in Fig. 4.7.

$$\begin{aligned} M_1 = \mathsf{Z}.(\mathsf{F} \mid \boxed{0}) \mid \mathsf{Z}.\boxed{1} \quad &\rightarrow \quad \mathsf{Z}.\boxed{0} \mid \mathsf{Z}.(\mathsf{F} \mid \boxed{1}) \\ M_2 = \mathsf{Z}.(\mathsf{Z}.(\mathsf{F} \mid \boxed{0}) \mid \boxed{1}) \quad &\rightarrow \quad \mathsf{Z}.(\mathsf{Z}.\boxed{0} \mid \mathsf{F} \mid \boxed{1}) \\ M_3 = \mathsf{Z}.(\mathsf{Z}.\boxed{0} \mid \mathsf{F} \mid \boxed{1}) \quad &\rightarrow \quad \mathsf{Z}.(\mathsf{Z}.(\mathsf{F} \mid \boxed{0}) \mid \boxed{1}) \end{aligned}$$

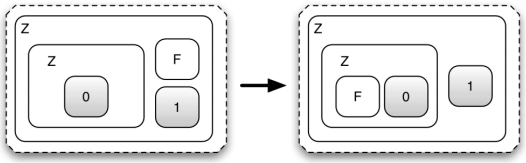
Reaction rules are here given on the form "$R \rightarrow R'$" rather than the more precise (R, R', η); recall from the above introduction to bigraphs that we use the former form whenever η is inconsequential (in this case, it is the identity map).



(a) M_1



(b) M_2



(c) M_3

Figure 4.7: Reaction rules M_1, M_2 and M_3 that allow *friend* nodes to move between *zones*.

We extend the movement rules M with an additional rule R_1 for notifications to be issued when a U (user) and F (friend) node exist within the

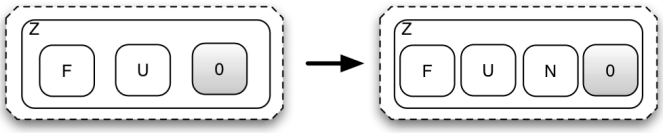


Figure 4.8: Reaction rule R_1

same zone. This reaction rule is illustrated in Fig. 4.8.

$$\begin{aligned} \Sigma_N &= \Sigma_M \cup \{\mathsf{U}, \mathsf{N}\} \\ R_1 &= \mathsf{Z}.(\mathsf{U} \mid \mathsf{F} \mid \boxed{0}) \qquad \rightarrow \qquad \mathsf{Z}.(\mathsf{U} \mid \mathsf{F} \mid \mathsf{N} \mid \boxed{0}) \end{aligned}$$

Let BRS_{notify} be the bigraphical reactive system formed by the addition of the reaction rule R_1 to the set of movement rules M:

$$BRS_{notify} = BG(\Sigma_N, M \cup \{R_1\})$$

4.3.2 The concrete system: $BRS_{selective}$

We now create a second bigraphical reactive system, this one refining (both intuitively and in a sense to be made precise) the system BRS_{notify} just introduced. In this new system, instead of simply notifying whenever *any* friend is present in the same zone as the user, we wish only to issue a notification in the presence of a particular designated friend, distinguished by the presence of an S (special friend marker) inside the friend node in question. Consequently, we define the set of controls Σ_S for $BRS_{selective}$ to include (in addition to the controls of Σ_N) the S control. The modified reaction rule R_2 is presented graphically in Fig. 4.9.

$$\begin{aligned} \Sigma_S &= \Sigma_N \cup \{\mathsf{S}\} \\ R_2 &= \mathsf{Z}.(\mathsf{U} \mid \mathsf{F}.\mathsf{S} \mid \boxed{0}) \rightarrow \mathsf{Z}.(\mathsf{U} \mid \mathsf{F}.\mathsf{S} \mid \mathsf{N} \mid \boxed{0}) \\ BRS_{selective} &= BG(\Sigma_S, M \cup \{R_2\}) \end{aligned}$$

At an intuitive level, this BRS refines the one of the previous sub-section. In the following section, we shall define exactly in what sense this is the case.

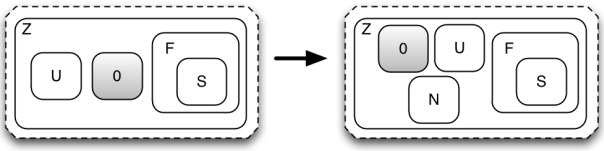


Figure 4.9: Reaction rule R_2

4.4 Vertical BRS Refinement

We recall the distinction here between *horizontal* and *vertical* refinement. Vertical refinement is concerned with moving between differing levels of abstraction, or indeed completely independent modelling languages, whereas horizontal refinement instead aims to relate models specified at the same fundamental level of abstraction, and within the same modelling setting. When we refer to the refinement of a BRS, we mean a vertical refinement, indeed, this is the only meaningful interpretation, as a BRS is the category consisting of (infinitely) many actual agents of the same general shape. We will later return (briefly) to what it would mean for an *agent* to be refined, that is, to a horizontal refinement between two agents of the same BRS (each of which would be bigraphs, representing — for example — two CCS processes).

To summarise the distinction between horizontal and vertical refinement in the setting of BRSs: In the former case, we are talking about what we can observe of all such agents, whereas in the latter we are referring to what we can observe of the behaviour of a single agent. In the present section, we consider vertical refinement; we comment on horizontal refinement in the subsequent section.

4.4.1 Safe refinements

First, what observations can you make of bigraphical agents? While the notion of a trace is familiar within refinement literature, within bigraphical reactive systems it is unclear exactly what might correspond to an *action* within the usual definition of a trace. Consequently, we formulate a trace of a BRS such that each element of the trace is a bigraphical agent (i.e., a bigraph of that BRS). Therefore the notion of trace is not one of a system exhibiting behaviour in the form of some observable actions, rather, it is the entire state of the model as it changes over time such that every element of the trace is a bigraph, related to the next element of the trace by the

application of some reaction rule. While this may seem very crude at first glance, it is important to remember that the dynamic behaviour of a bigraph is derived from reaction rules and the structure in a perhaps more direct manner than in many other calculi. As such, it makes sense to consider the abstract specification to comprise, by itself, an entire observation — cf. the structure of agents of BRS_{notify} above.

If an observation is a complete agent of the abstract specification, what then is an observation of an agent of the concrete system? We leave that to the system constructor, merely insisting that the observations one makes of concrete implementation agents must somehow be a function of their structure. Thus, observations of concrete agents are given by a structure-preserving map from concrete agents to abstract ones. In the parlance of category theory, this is called a "functor", a functor that we shall in this instance call an *abstraction functor*.

Definition 50 (Trace, observation)**.** For a given BRS A, a *trace* is a (possibly infinite) sequence of bigraphs (agents) $\langle a_1, a_2, \ldots \rangle$, such that for each a_i and a_{i+1} in the sequence there is a reaction $a_i \to a_{i+1}$. If $s = \langle s_1, \ldots, s_n \rangle$ and $t = \langle t_1, \ldots \rangle$ are traces and $s_n \to t_1$, we may form the *composite trace* $s; t = \langle s_1, \ldots, s_n, t_1, \ldots \rangle$. In this case we say that t is an *extension* of s. We write $Tr(A)$ for the set of all traces of a given BRS A. If $F : A \to A'$ is a functor and $\langle a_1, a_2, \ldots \rangle \in Tr(A)$ is a trace of A, we apply F pointwise to obtain a sequence $F(t) = \langle F(a_1), F(a_2), \ldots \rangle$.

We note two important properties:

1. $Tr(x)$ is by definition prefix-closed; that is, for any trace $t \in Tr(x)$, every prefix t' of t is also in $Tr(x)$.

2. Even though $F : A \to A'$ takes sequences of A to sequences of A', it does not necessarily take *traces* of A to traces of A'.

Of course, not just any functor will do: to have a refinement, the dynamic behaviour of the concrete implementation must be allowed by the dynamic behaviour the abstract specification allows on its agents, the observations. Altogether, our notion of refinement follows from the usual trace equality, however, because a BRS tends to permit too much observation, our bigraphical notion of refinement requires as a side condition that there exist an abstraction functor $F : C \to A$ such that for any trace $\langle c_0, c_1, \ldots \rangle$, F gives rise to a trace $\langle F(c_0), F(c_1), \ldots \rangle$. We present vertical refinement as the conjunction of two constituent definitions, separating the preservation of orthogonal safety and liveness properties through refinement.

Definition 51 (Safe Vertical Refinement)**.**

$$A \overset{\text{safe}}{\sqsubseteq}_F C \quad \overset{\text{def}}{=} \quad F(Tr(C)) \subseteq Tr(A)$$

This definition satisfies the "reduction of non-determinism" role of refinement, in that it is always valid to simply pick one alternative and implement it in C when presented with non-deterministic choice in A.

Lemma 2. Safe Vertical Refinement is transitive and reflexive for the identity functor.

Proof. Reflexivity is trivial. Suppose $A \overset{\text{safe}}{\sqsubseteq}_F C$ and $C \overset{\text{safe}}{\sqsubseteq}_G D$. Then $FG(Tr(D)) \subseteq F(Tr(C)) \subseteq Tr(A)$. □

We proceed to illustrate safe refinement using the two BRSs above, then give a sufficient condition for an abstraction functor to yield a safe refinement.

Recall our claim that $BRS_{selective}$, which issues notifications upon co-location with "special friends" is a refinement of BRS_{notify}, which does so upon co-location with any friend. The latter employs an additional control S. This indicates that our abstraction functor must (at the very least) ensure that all nodes of control S must be hidden, renamed or removed so as to ensure that the codomain of F is BRS_{notify} (i.e. that F can transform any agent of $BRS_{selective}$ into a valid agent of BRS_{notify}).

By this reasoning, we arrive at an abstraction functor "pattern" that is likely applicable to many other BRSs. We call this the *hiding functor*. Its essential function is to simply hide, for a given signature Σ, all nodes that have been assigned controls from some particular set of controls H. This includes joining any children of nodes that will be hidden to parents that will remain visible after the application of the hiding functor. For our example, the hiding set $H = \{\mathsf{S}\}$ (i.e. the designated "special" friend control).

Definition 52 (Hiding Functor)**.** We define an abstraction functor $F_{\Sigma,H} : BG(\Sigma) \to BG(\Sigma \setminus H)$ for hiding, parametrised by Σ, the signature of the "implementation" BRS, and H, a set of controls to be hidden. On *objects*, this functor is the identity. On *arrows*, its action is $F_{\Sigma,H}((V, E, prnt, ctrl, link)) \overset{\text{def}}{=} (V', E, ctrl', prnt', link)$, where

- $V' = \{v \in V | ctrl(v) \notin H\}$
- $ctrl' = ctrl \restriction V'$, and

– $prnt'(l) \stackrel{\text{def}}{=} \begin{cases} prnt(l) & \text{where } ctrl(prnt(l)) \notin H \\ prnt'(prnt(l)) & \text{otherwise} \end{cases}$

This "hiding functor" is an abstraction functor for our example system. Recalling the definition of a bigraphical agent (and therefore of an arrow in the category BRS_{notify} or $BRS_{selective}$) given in Definition 44, the purpose of this hiding functor is to exclude any nodes that have a control that is in the set of hidden controls H, exclude these controls from the control map $ctrl$, and recursively recreate the parent map $prnt$ such that any children of a node with a control in H is attached to its most immediate place-graph ancestor that is not marked with a control in H. We call the abstraction functor for our example notification system A_{friend}, which is defined as the hiding functor above, instantiated with $H = \{\mathsf{S}\}$.

While the hiding functor has the flavour of a forgetful functor — it dispenses with structure — it cannot reasonably be called so as it is not faithful. Many distinct configurations (e.g. special-friend controls) will map to the same bigraph. This is a technical distinction only; we use "hiding" in no special sense, except as a name for abstraction functors of this general shape.

It is easy to prove that with A_{friend} as abstraction functor, $BRS_{selective}$ is indeed a safe refinement of BRS_{notify}. However, instead of proving so directly, we shall instead provide a general theorem about abstraction functors: When they preserve reaction, and in particular, when they preserve just reaction *rules*, they give rise to safe refinement.

Theorem 4. Let $F : C \to A$ be an abstraction functor. If F preserves reaction, that is, if $c \to c'$ implies $F(c) \to F(c')$, then $A \stackrel{\text{safe}}{\sqsubseteq}_F C$.

Proof. Immediate from Definition 51 of safe refinement. □

From this theorem it becomes apparent that an abstraction functor may be any functor at all that obeys this property.

The terminology deceives, here: The guarantee that the concrete system has *no more* behaviour than the abstract one is in fact upheld by the abstraction functor *preserving* behaviour.

Of course, proving that a functor preserves reaction need not at all be easy. Fortunately, we can exploit the connection between static structure and dynamic behaviour of bigraphs: a functor which preserves the reaction *rules*, structurally, will also preserve (dynamic) reaction, and will thus be a safe refinement.

Theorem 5 (Safe Abstraction Functors)**.** Let $A = BG(\Sigma, \mathcal{R})$ and $C = BG(\Sigma', \mathcal{R}')$ be BRSs. A functor $F : C \to A$ yields a safe vertical refinement $A \overset{\text{safe}}{\sqsubseteq}_F C$ *if* it satisfies the following conditions.

1. It preserves and respects tensor.
2. It preserves active contexts.
3. It preserves reaction rules: For any reaction rule $(R, R', \eta) \in \mathcal{R}'$ (a) the F-image $(F(R), F(R'), \eta)$ is a rule in $\mathcal{R}$; and (b) for any parameter d of that rule, $\overline{\eta}(F(d)) = F(\overline{\eta}(d))$.

Proof. Suppose $c_1, \ldots, c_n$ is a trace of C. It is sufficient to prove that for each $i < n$, there is a reaction $F(c_i) \to F(c_{i+1})$. We know that $c_i \to c_{i+1}$, so there is some reaction rule $(R, R', \eta) \in \mathcal{R}'$, context E of C, and some set of names Z s.t.

$$c_i = E \circ (R \otimes 1_Z) \circ d \qquad \to \qquad E \circ (R' \otimes 1_Z) \circ \overline{\eta}(d) = c'_i$$

Where $\overline{\eta}(d)$ is the instantiation of parameters (see [83] for details). But then, because $(F(R), F(R'), \eta)$ is a rule of $\mathcal{R}$, we compute and find $a_i = F(c_i) = F(E \circ (R \otimes 1_Z) \circ d) = F(E) \circ (F(R) \otimes 1_{F(\mathsf{Z})}) \circ F(d) \to F(E) \circ (F(R') \otimes 1_{F(\mathsf{Z})}) \circ \overline{\eta}(F(d)) = F(E) \circ (F(R') \otimes 1_{F(\mathsf{Z})}) \circ F(\overline{\eta}(d)) = F(E \circ (R' \otimes 1_Z) \circ \overline{\eta}(d)) = F(c'_i) = a'_i$ □

We remark that the three conditions of this Theorem appear to be good candidates for a definition of a morphism of *parametric* reactive systems, as suggested in [36].

It is straightforward to verify that for our example BRSs, $BRS_{selective}$ and BRS_{notify}, the hiding functor does in fact satisfy the three conditions of this Theorem. Thus we have the following corollary:

Corollary 1. $BRS_{selective}$ is a sound refinement of BRS_{notify} with respect to the abstraction functor A_{friend}, that is,

$$BRS_{notify} \overset{\text{safe}}{\sqsubseteq}_{A_{friend}} BRS_{selective}$$

The $\overset{\text{safe}}{\sqsubseteq}$ relation captures safety properties of the system being refined (i.e. it does not permit a refined model any undesirable extra behaviour, provided that the abstraction functor does not hide any "undesirable" behaviour). However, it does not guarantee that the system does anything at all (i.e. an empty trace is a safe refinement of any system). To guarantee that some additional liveness properties are preserved by refinement, it is necessary to extend our definition.

4.4.2 Live refinements

In order to guarantee that a given concrete system actually exhibits any of the desirable behaviour of the abstract system that it refines, we must define a notion of liveness. Whereas in a process algebraic setting it might be possible to rely on the presence of a particular output (or all possible outputs) to define "desired" observable behaviour, within a bigraphical setting the lack of any primitive notions of "input" or "output" (it is up to the system designer to define what these concepts mean with respect to a particular model) means that it is necessary to explicitly choose such "desirable" behaviours.

In the absence of an intrinsic notion of desirable behaviour, we further parametrise our notion of liveness, already parametric in terms of the abstraction functor F, on the admissible traces. This parametrisation on the notion of admissibility is akin to those used in [57, 53].

Definition 53 (Live Vertical Refinement)**.** Let $F : C \to A$ be an abstraction functor, let $\mathbf{C} \subseteq Tr(C)$ be the *admissible traces* for C, and let similarly $\mathbf{A} \subseteq Tr(A)$, the admissible traces of A. We then say that $(C, \mathbf{C})$ is a *live* refinement of $(A, \mathbf{A})$ iff for every trace s of $Tr(C)$, whenever $F(s)$ has an extension t' to an admissible trace $F(s); t' \in \mathbf{A}$, then there exists an extension s' of s to an admissible trace $s; s' \in \mathbf{A}$ with $F(s') = F(t')$. In this case we write:

$$(A, \mathbf{A}) \overset{\text{live}}{\sqsubseteq}_F (C, \mathbf{C}) \ .$$

If we wish to take the admissible traces $\mathbf{A}$ of the abstract system A as canonical, we can define $\mathbf{C}$ as those traces whose F-images are admissible. The reflection of extensions to admissible traces is reminiscent of the use of open maps in [57, 58].

Lemma 3. Live Vertical Refinement is transitive.

Proof. Suppose $(A, \mathbf{A}) \overset{\text{live}}{\sqsubseteq}_F (C, \mathbf{C})$ and $(C, \mathbf{C}) \overset{\text{live}}{\sqsubseteq}_G (D, \mathbf{D})$, and suppose $FG(t); u' \in \mathbf{A}$. Then u' has a pre-image s' with $G(t); s' \in \mathbf{C}$; but then s' has a pre-image t' with $t; t' \in \mathbf{D}$. □

Let us provide a suitable set of admissible traces for our running example, BRS_{notify}. For this BRS, the obvious notion of admissibility (think "successful") is when notification has occurred. So we define the set of admissible traces as simply those finite traces in which the user has been

notified, that is, in which the final agent contains the notification control next to the user and his friend:

$$\mathbf{S}_{notified} \stackrel{\text{def}}{=} \{\langle a_1, \ldots, a_n\rangle \in Tr(BRS_{notify}) \mid \exists C.\ a_n = C \circ (\mathsf{U} \mid \mathsf{F} \mid \mathsf{N})\}$$

For $BRS_{selective}$, we transfer the notion of admissibility:

$$\mathbf{S}_{selective} \stackrel{\text{def}}{=} \{t \in Tr(BRS_{notify}) \mid F(t) \in \mathbf{S}_{notified}\}$$

The selective system $BRS_{selective}$ under these notions of admissibility is in fact *not* a live refinement of the original one BRS_{notify}. One might think so: After all, one can extend a trace to admissibility simply by moving the special friend next to the user. Unfortunately, there need not be a special friend, and even if there were, the abstract system might extend to admissibility by moving a (non-special) friend next to the user. We will now show this in detail, thus proving of the following proposition:

Proposition 5. $(BRS_{notify}, \mathbf{S}_{notify}) \not\overset{\text{live}}{\sqsubseteq}_{A_{friend}} (BRS_{selective}, \mathbf{S}_{selective})$.

Proof. Consider an agent $\mathsf{Z}.(\mathsf{U} \mid \mathsf{F})$ of $BRS_{selective}$. Applying A_{friend} we find simply $A_{friend}(\mathsf{Z}.(\mathsf{U} \mid \mathsf{F})) = \mathsf{Z}.(\mathsf{U} \mid \mathsf{F})$, which succeeds after just one reaction

$$\mathsf{Z}.(\mathsf{U} \mid \mathsf{F}) \to \mathsf{Z}.(\mathsf{U} \mid \mathsf{F} \mid \mathsf{N})$$

by reaction rule R_1. Now, if we actually had a live refinement, we should be able to match this reaction in $BRS_{selective}$. A simple inspection of the rules however prove that this is not possible. □

This is, however, not a show-stopper, rather it is a welcome demonstration of the utility of such a vertical refinement mechanism. We could remedy this situation by introducing into $BRS_{selective}$ an additional reaction rule that spontaneously adds the designated friend marker S to any friend F. However, this seems to contradict the intuition of the model, so in this instance it is perhaps better to leave $BRS_{selective}$ unmodified and accept that there are (known) conditions under which this BRS cannot progress to a successful state.

Having defined our two separate (live and safe) refinement relations, we can complete the definition of safe and live vertical refinement:

Definition 54 (Safe and Live Vertical Refinement)**.**

$$(A, \mathbf{A}) \sqsubseteq_F (C, \mathbf{C}) \stackrel{\text{def}}{=} A \overset{\text{safe}}{\sqsubseteq}_F C \ \wedge\ (A, \mathbf{A}) \overset{\text{live}}{\sqsubseteq}_F (C, \mathbf{C})$$

4.5 Discussion & related work

Having introduced our notion of vertical BRS refinement and shown the conditions under which it is safe and live with respect to the chosen abstraction functor, we now discuss potential approaches to horizontal refinement and related work. As it happens, both topics take us to the general refinement of Reeves and Streader [94, 95].

General horizontal refinement recognises three components to refinement: *entities* E, i.e., the specifications and implementations being refined; *contexts* Ξ, which are the environment within which the entities interact; and a *user*, which defines the possible observations $O(-)$ that can be made of an entity within a particular context. Refinement is then the relation

$$A \sqsubseteq_{\Xi,O} C \stackrel{\text{def}}{=} \forall x \in \Xi . O([C]_x) \subseteq O([A]_x) \ ,$$

where Ξ is the set of contexts, O is a map assigning observations to entities in contexts, and $[-]_x$ inserts an entity into context x.

Interestingly, our proposed notion of bigraphical *vertical* refinement falls under the umbrella of general *horizontal* refinement. Entities would be BRSs (like BRS_{notify} and $BRS_{selective}$); contexts Ξ would be just the trivial context, which leaves the entity unchanged. Finally, the observation map O is in our case simply $Tr(-)$, the map that takes a BRS to the traces observable of it. We do not think this is a coincidence. It seems intuitive that horizontal refinement of an entire class of agents would correspond to vertical refinement.

What about general *vertical* refinement, then? The definition of vertical refinement within the general refinement framework [95] relies upon a notion of *layers*, representing a level of abstraction in terms of (E_L, Ξ_L, O_L), where E_L is a set of entities, Ξ_L is a set of contexts and O_L is an observation function. Vertical refinement is then defined in terms of a Galois-connection that interprets high-level entities as low-level ones and vice versa.

The analogy of this notion with our use of an abstraction functor $F : C \to A$ should be apparent: If we could find that functor F to be one of an adjoint pair, we would be in an analogous situation. Unfortunately, it remains unclear if such an adjunction would retain the intuition behind the Galois-connection of general vertical refinement: morphisms (i.e., bigraphs) do not measure approximation; they represent the agents under investigation. In particular, the hiding functors used for the example in the present paper do not appear to be part of adjoint pairs.

Leaving vertical refinement behind, what is then a good notion of horizontal refinement for bigraphs? Returning to general horizontal refinement,

bigraphs actually do come with a notion of entity, context, and observation, namely agents (roughly, bigraphs with no holes/inner names), bigraph contexts (bigraphs with holes/inner names), and an LTS (given a BRS). We have in the present paper by-passed the LTS as the notion of observation, following the bigraphical connection by structure and dynamics to its extreme conclusion, using the structure of the abstract specification as the observations.

For horizontal refinement, this approach appears not sensible: We would after all be relating agents of the same BRS. Important examples (like CCS-process refinement) cannot be expressed within this particular approach, which should guide the development of other horizontal refinement strategies for bigraphical agents. One obvious choice seems now to be the LTS intrinsic to BRSs. We have yet to pursue this option; we caution that while BRS LTSs have been successful in recovering semantics of various process algebras and other models of concurrency, it has been less successful in providing useful semantics for pervasive systems, one of our key interests.

However, even leaving the question of suitable observations open, we would likely find a notion inside general horizontal refinement by taking

$$a \sqsubseteq_O c \stackrel{\text{def}}{=} \forall x \in \Xi . O(x \circ c) \subseteq O(x \circ a) \ ,$$

where a and c are agents of some BRS B; Ξ is the set of contexts of that BRS, and O is some notion of the semantics of agents of B, perhaps traces of the LTS of B, or perhaps some other notion. Indeed, early indications are that this approach would be promising in recovering (for example) CCS process refinement, contingent upon an appropriate notion of observation.

4.5.1 Related Work

Restricting the set of controls admissible under a certain control, or requiring a control to be present is well-studied in bigraphs (e.g., [22, 83, 81, 88]). However, that study has invariably focused on ensuring that the bigraphical LTS theory is retained under such additional constraints, and are thus only superficially related to the present paper.

Goldsmith & Creese [48] explore an approach to refinement within bigraphs (and particularly within Spygraphs, a specialisation of bigraphs). They observe the ease with which one may derive an LTS for a BRS that is labelled exclusively by the trivial context id (equivalent to a τ action in a process algebraic setting). These kinds of contextual labels are not helpful for analysis, as they capture no behaviour. Similarly, the LTS semantics of

bigraphs share the same intentionality inherent in the graphical presentation. While Goldsmith & Creese suggest (to good effect in a CSP setting) that it may be appropriate to perform hiding at a process-level before considering a transition into bigraphs, this would seem inappropriate for many modelling situations (e.g., those which have no convenient term or process representation). While the transformation on bigraphical reactive systems proposed by that work may give rise to a refinement that is appropriate for some situations, we aim instead in this present work to work directly within the structure of bigraphs so as to ensure generality. As bigraphs attempt to be both a modelling formalism and a general meta-calculus for existing process calculi, it seems appropriate that the notion of refinement we introduce should be similarly general, in the hope that we may recover calculus-specific notions of refinement within this general setting.

4.6 Conclusion

We have presented a vertical refinement mechanism for bigraphical reactive systems that adds refinement to the toolbox of model builders working within a bigraphical setting. The addition of a sufficient condition for safe abstraction functors, and the accompanying observation that it is the *preservation* of behaviour with respect to reaction that guarantees that a refinement exhibits no undesirable behaviour, provides a firm foundation from which to explore the limits and utility of this kind of vertical refinement.

We have pointed out a clear connection to the existing work on generalising refinement across many modelling formalisms, and therefore it seems appropriate (given the application of BRSs as a *meta-calculus*) that our notion of vertical refinement is also in some sense general. We leave for future work the exploration of further mechanisms for horizontal refinement within a bigraphical setting, noting that such a notion would very likely fall within the model of general refinement, and thus likely generalise well to other modelling formalisms encoded within bigraphical reactive systems.

4.7 Addendum: Stuttering

We consider an extension to a more liberal notion of refinement by including *stuttering* steps, in the style of Abadi & Lamport [1]. This relaxes the requirement that the concrete system behave like the abstract system in

lock-step, but rather permits the concrete system to take finitely many *stuttering* steps before reaching a state agreeing with the abstract system, provided that each of the intermediate steps are mapped to "no operations" by the abstraction functor.

Definition 55 (Stuttering Equivalence and Stuttering Closure)**.** Following Abadi & Lamport [1], we consider consecutive repeated actions in a sequence to be "stuttering", and we define for any sequence s the stutter-free form $\sharp s$, which is s with all maximal finite stuttering subsequences removed (i.e., consecutive repeated elements removed). We also define $s \simeq s'$ to mean $\sharp s = \sharp s'$ where s and s' are sequences that are equivalent up to stuttering. If S is a set of sequences, we write $S^\star$ for the *stuttering closure* of S, defined as the least set containing S and satisfying:

$$S^\star = \{s \mid \exists s' \in S.s \simeq s'\}$$

While we permit infinite traces, in constructing our abstraction functors, we should be careful to only permit *finitely* many (consecutive) stuttering steps, though they may occur infinitely often. The distinction is subtle but important, such that $\sharp s$ may remove infinitely many stuttering subsequences, but should not be used to remove subsequences of infinite length. A process that stutters infinitely before reaching an acceptable state defeats the intuition of what a safe refinement should be, and we should therefore be wary of processes that exhibit this kind of stuttering behaviour.

We can now revise our definition of safe vertical refinement, by giving a new definition that permits stuttering steps. Note that the version of refinement given in Definition 51 that does not permit stuttering is trivially subsumed by this definition, as $Tr(C) \subseteq Tr(C)^\star$. We preserve both definitions, referring to the stuttering variant as *stuttering-safe* vertical refinement, as it may not be desirable in all instances to permit stuttering.

Definition 56 (Stuttering-Safe Vertical Refinement)**.**

$$A \overset{\text{safe}}{\sqsubseteq}_F C \quad \overset{\text{def}}{=} \quad F(Tr(C)) \subseteq Tr(A)^\star$$

Notice that the concrete system C may safely refine A by not terminating. For example, suppose A, C have the same underlying category of bigraphs but different reaction semantics, such that $Tr(A) = \{\epsilon, \langle a\rangle, \langle a, b\rangle\}$ whereas $Tr(C) = \{\epsilon, \langle a\rangle, \langle a, a\rangle, \ldots\}$. Then the identity functor 1 on the shared underlying category is a stuttering-safe refinement, i.e., $A \overset{\text{safe}}{\sqsubseteq}_1 C$.

This notion of refinement is, of course, transitive.

Lemma 4. Safe Vertical Refinement is 1) transitive and 2) reflexive for the identity functor.

Proof. Reflexivity follows from $S \subseteq S^\star$. For transitivity, suppose $A \overset{\text{safe}}{\sqsubseteq}_F C$ and $C \overset{\text{safe}}{\sqsubseteq}_G D$. Then, because F, G lifts to sequences pointwise, $FG(\mathit{Tr}(D)) \subseteq F(\mathit{Tr}(C)^\star)$. Now consider a sequence s' of $\mathit{Tr}(C)^\star$. By definition, s' arises by inserting finitely many stuttering steps into some trace $s \in \mathit{Tr}(C)$. But then $F(s) \in \mathit{Tr}(A)^\star$, so also $F(s') \in \mathit{Tr}(A)^\star$. Hence $F(\mathit{Tr}(C)^\star) \subseteq \mathit{Tr}(A)^\star$ and we are done. □

We can now show that an equivalent theorem about safe abstraction functors can be proved for stuttering-safe vertical refinement too.

Theorem 6. Let $F : C \to A$ be an abstraction functor. If F weakly preserves reaction, that is, if $c \to c'$ implies $F(c) \to F(c')$ or $F(c) = F(c')$, then $A \overset{\text{safe}}{\sqsubseteq}_F C$.

Proof. We must prove that in this case $F(\mathit{Tr}(C)) \subseteq \mathit{Tr}(A)^\star$. So consider some trace $s \in \mathit{Tr}(C)$; we proceed by induction on s. The base cases of empty or singleton traces are trivial, so suppose $s = s'; \langle x, y \rangle$. By induction, $F(s'; \langle x \rangle) \in \mathit{Tr}(A)^\star$. Because $s'; \langle x, y \rangle$ is a trace, $x \to y$ in C. But then either $F(x) \to F(y)$ or $F(x) = F(y)$; in both cases also $F(s'; \langle x, y \rangle) \in \mathit{Tr}(A)^\star$. □

We similarly revise our definitions for live refinement, such that stuttering steps may be included in live refinements too:

Definition 57 (Stuttering-Live Vertical Refinement)**.** Let $F : C \to A$ be an abstraction functor, let $\mathbf{C} \subseteq \mathit{Tr}(C)$ be the *admissible traces* for C, and let similarly $\mathbf{A} \subseteq \mathit{Tr}(A)$, the admissible traces of A. We then say that $(C, \mathbf{C})$ is a *stuttering-live* refinement of $(A, \mathbf{A})$ with respect to a functor $F : C \to A$ iff for every trace s of $\mathit{Tr}(C)$, whenever $F(s) = t$ has an extension t' to an admissible trace $t; t' \in \mathbf{A}^\star$, then there exists an extension s' of s with $s; s' \in \mathbf{C}^\star$ and $F(s') = t'$. In this case we write:

$$(A, \mathbf{A}) \overset{\text{live}}{\sqsubseteq}_F (C, \mathbf{C}) \,.$$

Lemma 5. Live Vertical Refinement is 1) transitive and 2) reflexive with respect to the identity functor.

Proof. Reflexivity is trivial. For transitivity, suppose $(A, \mathbf{A}) \overset{\text{live}}{\sqsubseteq}_F (C, \mathbf{C})$ and $(C, \mathbf{C}) \overset{\text{live}}{\sqsubseteq}_G (D, \mathbf{D})$, and suppose $(FG(t))u' \in \mathbf{A}^\star$. Then u' has a pre-image s' with $G(t); s' \in \mathbf{C}^\star$. But then s' has a pre-image t' with $t; t' \in \mathbf{D}^\star$. □

4.7.1 Summary

The value of stuttering in vertical refinement is that it permits markedly different systems to be related by vertical refinement relations. Particularly when we begin to consider calculational BRSs [36], such that we may wish to include calculation (requiring many steps) in the concrete system. For this reason and others, stuttering seems a significant step in improving the flexibility of vertical refinement, and therefore contributes further to the vision of a tower of models related by vertical refinement relations.

Chapter 5

A Verification Environment for Bigraphs

Context

> *This chapter is derived from the paper* A Model Checker for Bigraphs *by Gian Perrone, Søren Debois, and Thomas Hildebrandt which was accepted for publication in the proceedings of the ACM Symposium on Applied Computing (Software Verification and Tools Track) 2012.*
>
> *The version that appears in this work has been reformatted, and augmented with some new content, including the initial performance data, and a more precise characterisation of the reachability checking algorithm that makes use of the causation analysis, both of which were suggested by the anonymous reviewers. This version is presently under review for inclusion in the* Innovations in Systems and Software Engineering *(ISSE) journal, the results of which should be known prior to the eventual publication of this thesis.*

Embarking upon bigraphs research for the first time, I was struck by the lack of an easy means to execute bigraphs. The BPLTool [46] was no longer maintained, and suffered from being of the first generation of bigraph tools — It did not make it easy to take a bigraph model and really *use* it. BigMC was an attempt to remedy this. Growing out of an (abortive) attempt to construct a refinement checker for the methods described in Chapter 4, the implementation took place during summer of 2011, and we quickly

discovered that there was an opportunity to compute an interference (or causation) relation between bigraph reaction rules. Since the publication of this paper, Espen Højsgaard's PhD thesis [61] was published, which gave a more theoretically precise characterisation of the ways in which reaction rules can interact. However, the method presented in this chapter remains (to date) the only method for computing reaction rule interference that has been implemented, and which is known to be tractable and efficient. For this reason, it seems possible that it may be implementable to good effect in other bigraph tools. In reading this chapter, the causation computation should be considered one of the primary contributions, as it does (in my view) represent a concrete step towards making the execution of bigraphs more efficient, and therefore making the formalism itself more generally applicable.

Also, since the publication of this work, BigMC has seen some use by others wishing to experiment with bigraphs. This use has seen the tool improve in quality and performance considerably, and has seen it integrated with the Big Red [42] graphical editor for bigraphs, developed at the IT University of Copenhagen.

Abstract

We present the BigMC tool for bigraphical reactive systems that may be instantiated as a verification tool for any formalism or domain-specific modelling language encoded as a bigraphical reactive system. We introduce the syntax and use of BigMC, and exemplify its use with two small examples: a textbook "philosophers" example, and an example motivated by a ubiquitous computing application. We give a tractable heuristic with which to approximate interference between reaction rules, and prove this analysis to be safe. We provide a mechanism for state reachability checking of bigraphical reactive systems, based upon properties expressed in terms of matching, and describe a checking algorithm that makes use of the causation heuristic.

5.1 Introduction

BigMC began as a tool for performing reachability analysis on bigraphical [83, 81] models of systems and models constructed within languages encoded in bigraphs; however, it has grown to provide a general way of executing bigraphs according to a variety of strategies. The default strategy is a breadth-first exploration of the state space to permit checking of properties, but the tool has matured to permit user-defined state space exploration strategies. We provide an overview of the motivation for the development of BigMC as well as detailing the implementation of the tool. We assume no prior knowledge of bigraphs, and a brief introduction to bigraphs as well as references to introductory material are given in the sections that follow. It is important to note that bigraphs serve as a *meta-calculus* in which to embed existing formalisms (e.g., several variants of π-calculus [64, 26, 39], the λ-calculus [82], the Mobile Ambient calculus [64], Petri Nets [71] and CCS [81]) as well as encoding your own domain-specific modelling language. As a result, a verification tool for bigraphs effectively permits the instantiation of a verification tool for any such formalism that can be encoded within bigraphs, and also gives rise to a *specification language* that can express safety properties over these models.

The primary contributions of this work are as follows:

- A verification tool for models, languages and correctness properties described as bigraphs that may be instantiated as a reachability checker or simulator for many different domain-specific and general purpose languages encoded as bigraphs.

- A tractable, safe approximation of interference analysis over reaction rules, which represents an advancement of the state of the art in bigraph theory.
- A novel mechanism for reachability analysis on bigraphical reactive systems (BRSs) based upon the expression of correctness properties as matching.
- We build on the preliminary description of the analysis method [92], by detailing an algorithm that uses the interference analysis, providing the potential to check safety properties of some systems with infinitely many states.

5.1.1 Motivation

Bigraphs were proposed by Milner and colleagues as a modelling formalism and meta-calculus [83]. One of the primary challenges that bigraphs were designed to address was the proliferation of so-called *ubiquitous computing* applications that has been repeatedly predicted. The challenge is that we must aim to understand the complexity of these systems thoroughly before they "disappear" into the computing fabric of every day life [5]. If bigraphs are a means to understand the complexity of these emerging systems, then a verification environment for bigraphs that permits reachability analysis and verification of properties is surely a good first step towards mastering this complexity.

5.1.2 Reachability Analysis

BigMC implements explicit-state checking of properties expressed as matching. From a description of a model, it finds all possible future configurations of this model and checks the specification against them. This limits its utility for checking any model with infinite behaviour. Infinite behaviour in this context could mean having finitely many reaction rules that by every application lead to a previously-unexplored state (i.e., an infinite-state model), or by having cycles (i.e., where some finite set of states may be visited infinitely often). BigMC can check properties of models of the latter kind reliably, and in some cases those of the former (described in Section 5.5). Both the properties being checked and the models themselves are specific instantiations of bigraphical reactive systems.

One of primary benefits of the tool is the ability to provide a *counter-example* in the event that the specification is shown not to hold. In our case,

this means showing the system configuration that violates the specification, and the path through the transition system by which this configuration was reached.

5.1.3 Simulating Bigraphs

BigMC has many uses, including as a teaching tool for bigraphs, and as a straightforward means to execute a bigraphical reactive system. Using it in this mode simply requires not specifying any properties to check, and enabling the so-called *local* checking mode, for which the transition system is not stored. This can be further leveraged to enable a random walk through the states of a model, by disabling the exhaustive search of the state space and instead choosing one of several possible successor states. We have enabled experimental support for extensions to BigMC that would permit other means of choosing the order in which to export states during BRS execution to be implemented, including stochastic bigraphs [68], for example. At present, choosing between alternative execution strategies requires direct modification of a specific part of the source code, though we hope to remedy this and provide runtime options for various execution strategies in the future.

5.1.4 Related Work

There exist several tools for operating on various kinds of bigraphs already; however, all but one of these (BigraphER) do not enable any kind of reachability analysis.

BPLTool [46] was one of the first implementations of bigraph matching, with an emphasis on the correctness of the implementation. While it implemented matching, there was no notion of checking or analysis of properties of systems [46].

DBtk [10] is a tool for manipulating *directed bigraphs* [49], which are a variant of bigraphs with a directed link structure. It implements matching for directed bigraphs, and enables the generation of relative pushouts (RPOs) which are the constructions that permit the bisimulation congruence result within the theory of bigraphs [83]. It does not extend to checking properties of bigraphical models, however.

BigraphER [27] is a tool that has appeared recently, supporting *bigraphs with sharing*, which is an extension of bigraphs in which the place graph is permitted to be a DAG, rather than a tree as in pure bigraphs. BigraphER supports checking of BiLog formulae, although the limited platform support

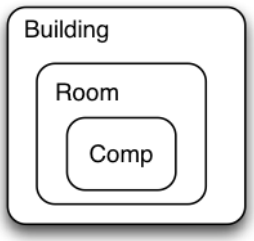


Figure 5.1: Place graph prefixing example. Containment of one node inside another indicates that the inner node is a child of the outer node in the place graph tree structure.

at present has prevented us from performing a direct comparison of the functionality that it provides.

The *Stochastic Bigraphical Abstract Machine* (SBAM) [61] is a prototype tool for the scalable simulation of stochastic bigraphs [68], specifically tailored towards systems biology applications. Owing to the emphasis on *scalable* simulation, only *ground* reaction rules (i.e., non-parametric rules) are considered, and SBAM is therefore not suitable for reachability analysis over the full generality of bigraphical reactive systems. It would be interesting future work to consider ways in which the BigMC tool that we present could integrate with SBAM to enable scalable simulation of the subset of BigMC models which correspond to those for which SBAM is well-suited.

5.1.5 Structure

The rest of this work is structured as follows: In Section 5.2 we give a brief introduction to bigraphs and introduce the term language with which they are expressed in the BigMC tool. Section 5.3 introduces two example problems to which the tool has been applied. Section 5.4 gives the naive reachability analysis algorithm implemented by the tool. Section 5.5 describes a static analysis on reaction rules to limit the state space that must be explored, and gives a proof of correctness before extending the reachability checking algorithm to make use of the analysis. Finally, Section 5.7 provides a brief summary, and introduces possible directions for future work.

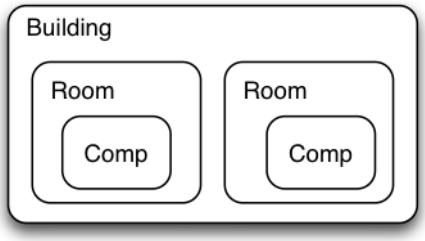


Figure 5.2: Place graph sibling juxtaposition example. Nodes contained by the same parent are siblings in the place graph tree structure.

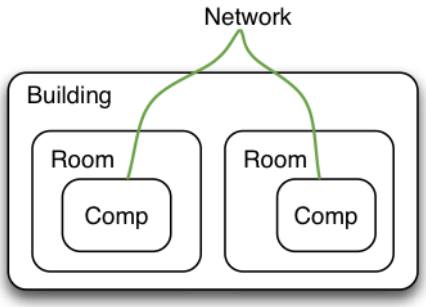


Figure 5.3: Link graph example, showing that links (the heavy lines) can cross node boundaries to link ports to names.

5.2 Bigraphs

BigMC implements *pure* bigraphs, excluding shared *local names* [83]. We present here a brief introduction to bigraphs, eliding some details that are not directly relevant to the verification task at hand. For a detailed introduction to bigraphs and bigraphical reactive systems, the reader is referred to [83]. Bigraphs have been found to be very general as a *meta-calculus* in which to define diverse calculi and models of concurrent systems. Consequently, a verification environment for bigraphs allows one to specify a language (encoded as a bigraphical reactive system) and then check properties (encoded as a set of bigraph) of models constructed with respect to this language. This distinction is one of the motivating factors for using bigraphs, in that they retain general expressiveness while still being comprehensible as a direct modelling formalism.

5.2.1 Static Structure

A bigraph consist of two graphs; a *place* forest that captures notions of locality or containment, and a *link* hyper-graph that models connectivity or associations. The place graph consists of a set of *nodes*, each of which is assigned a *control* drawn from the *signature* of the bigraph. A control is some distinguishing name and an *arity* that determines the number of connecting *ports* that are available at that node to be associated with links in the link graph. A control is also marked either *active* or *passive*, which determines whether reaction may take place within a given context. Bigraphs have a formal visual presentation as well as graph and term representations. We will primarily present the term representation here, as this is the basis for the input language for BigMC.

We write nodes using their control names, which may be any valid identifier. Terms are nested by *prefixing*, such that for some control name a and some term T, $a.T$ is a place graph node with control a containing the term T. For example, `Building.Room.Comp` (Fig. 5.1) is a `Building` node containing a `Room` node, which in turn contains a `Comp` node. Sibling nodes are juxtaposed (as in Fig. 5.2) using the parallel operator, e.g.:

`Building.(Room.Comp | Room.Comp)`

is a `Building` node containing two `Room` nodes, each of which contains a `Comp` node. Prefixing binds tighter than juxtaposition. We described the place graph as being a *forest* of trees. The roots of these trees (referred to as a *regions*) are ordered with respect to one another, while their children are unordered. New regions are introduced using the double parallel operator `||`, which unlike `|` is not commutative, e.g., `A.B || C | D.E` has two regions that may not be reordered, but `C | D.E` and `D.E | C` are equivalent. Bigraphs with more than one region are called *wide*, while non-wide terms are called *prime*.

Link graphs are presented here as a set of *outer names* that are globally visible. A node with the control a with an arity of 3 can link to names x, y and z with the syntax `a[x, y, z]`. Because the link graph is a *hypergraph*, multiple different ports may connect to a single name. Two nodes linked to a common name has the effect of linking those two ports together. We demonstrate this by imbuing the *Comp* nodes from our previous examples with a single port that allows them to be connected together by some common name — perhaps indicating connectivity to some network, for example. Links can cross node boundaries (as in Fig. 5.3), as they are completely independent of the place graph:

```
Building.
   (Room.Comp[Network] | Room.Comp[Network])
```

5.2.2 Dynamics

Dynamic behaviour is introduced using *reaction rules* that can be expressed in the form $R \to R'$, where R is a *redex* to be found inside some active context, and R' is a *reactum* with which the matched portion of the bigraph should be replaced. Reaction rules may contain *holes* (also known as *sites* in some literature) which are place graph elements which can match *parameters*. Place graph holes are represented with the notation `$n`, where $n \geq 0$. An active context is one for which the controls of all the place graph ancestors of the holes are marked active.

We can describe reaction as the decomposition of some bigraph G into a context C, a redex R and some parameters d, rewriting it to some G' thus:

$G \equiv C \circ (R \otimes 1_X) \circ d \rightarrowtriangle C \circ (R' \otimes 1_X) \circ \eta(d) \equiv G'$

Where η is some instantiation of the parameters, and R' is the reactum of the matched rule. The notation $R \otimes 1_X$ is used to handle outer names of the parameters (see [83] for details). Composition $C \circ a$ can be thought of as the insertion of some agent a into the context C, which contains one or more holes. Continuing our building example, if we wished to match some redex matching any two sibling rooms `Room.$1 | Room.$2` within our model, then one appropriate decomposition of the model might be:

$$C = \texttt{Building.\$0}$$
$$R = \texttt{Room.\$0 | Room.\$1}$$
$$d = \texttt{Comp[Network] || Comp[Network]}$$

Such that composing all these parts together (inserting R into hole 0 in C, and inserting the regions of d into the holes of R, in left-to-right order) will permit us to recover our original model exactly (modulo handling of names and commutativity of `|`, which we obscure in this example).

5.2.3 Term Language

The full grammar for BigMC bigraph terms is given as:

```
M ::= E; M | E;
E ::= %passive k : n
E ::= %active k : n
```

```
E ::= %rule N T -> T
E ::= %property N P
E ::= T -> T | T
T ::= K.T | T | T | T || T | $n | K | nil
K ::= k[L] | k
L ::= N , L | N
N ::= [a-zA-Z][a-zA-Z0-9]* | -
```

Where k is drawn from the set of controls (described by the same regular expression as N without the − character) and n ranges over the natural numbers. The keyword `nil` indicates the empty bigraph, and is used to terminate nesting of prefixes, though we will omit it where it is unambiguous (e.g., `a.b.c.nil` can be written `a.b.c`). The tool will reject rules with only `nil` on the left-hand side (as it cannot meaningfully execute them).

Controls must be pre-defined by the declaration of the bigraph *signature* with the %`active` and %`passive` commands, which define the arity of a given control as well as whether it is active or passive. Our term language is not complete (it lacks inner names and edges), but we believe that the algorithms employed will extend to a complete term representation. The inclusion of the "−" character in the syntactic category defining what may appear in links is a convenience. It acts as a sort of wild-card for links, the effect of which is to generate some anonymous name that may not be referenced elsewhere. This could just as well be avoided, by simply forcing the user to invent new names for every occurrence.

5.2.4 Properties

Properties are expressed as combinations of two built-in predicates: `matches` and `terminal`. The former describes some redex that we must find (or insist that we not find) in every possible agent of a given system as it behaves. For example, we can write a property: %`property no_a` !`matches`(`a`.$`0`); which states that we must never find a match for the redex a.$0, otherwise a violation will be reported. The `terminal`() predicate is true iff there are no possible further states reachable by a step of reaction from the current one. There also exists the boolean operators *and*, *or* and *not* to combine predicates. The full grammar for defining properties is given by:

```
P ::= matches(T) | terminal()
P ::= P && P | P || P | !P
```

5.3 Examples

5.3.1 Dining Philosophers

We introduce a simple formulation of the dining philosophers problem [60]. We encode a table with five philosophers, and in the rules implement a particular strategy (known to be vulnerable to deadlocks), in which a given philosopher may pick up the fork to the left and then the fork to the right so as to have two forks with which to eat. The forks are then placed back on the table in the same order — left first, then right. Place graph nesting is used to represent holding forks, and the link graph is used to capture the adjacency of forks and philosophers.

```
# Fork, with a fork id
%passive F : 1;
# Philosopher (left fork, right fork)
%passive P : 2;

%rule pickup_left
  P[lf,rf] || F[lf] -> P[lf,rf].F[lf] || nil;
%rule pickup_right
  P[lf,rf].F[lf] || F[rf] ->
    P[lf,rf].(F[lf] | F[rf]) || nil;
%rule drop_left
  P[lf,rf].(F[lf] | F[rf]) ->
    F[lf] | P[lf,rf].F[rf];
%rule drop_right
  P[lf,rf].F[rf] -> P[lf,rf] | F[rf];

# The philosopher's table
   F[F1] | P[F1,F2] |
   F[F2] | P[F2,F3] |
   F[F3] | P[F3,F4] |
   F[F4] | P[F4,F5] |
   F[F5] | P[F5,F1];

%property deadlock_free !terminal();
```

Using BigMC we find the expected violation — the situation where every philosopher has picked up the fork to the left, and none can proceed any

further. We encode the *deadlock freedom* property as the lack of any *terminal* node in the transition system, i.e., the property that the system is never in a state where it cannot proceed (by a step of reaction) to some other state. The terminal (and therefore deadlock) state is:

$$\mathtt{P[F1,F2].F[F1]} \mid \mathtt{P[F2,F3].F[F2]} \mid \mathtt{P[F3,F4].F[F3]} \mid \mathtt{P[F4,F5].F[F4]} \mid \mathtt{P[F5,F1].F[F5]}$$

What we have defined here is a simple *domain-specific modelling language* for describing dining philosophers. The signature of controls is generic, as are the reaction rules that give the semantics of the language. Only the specific table would need to change if we wanted to model other situations with different numbers of philosophers, for example. In Section 5.6 we use this example with different numbers of philosophers to measure the performance of the BigMC tool.

5.3.2 Example: Built environment

Our application involves *secure rooms* that contain computers, which may contain *secret tokens*. We have `Alice` and `Bob`, each of whom carries a mobile phone which can connect to computers, or which can transfer secure tokens between one another. `Alice` can legitimately gain access to the secure rooms. However, when carrying a phone, she is capable of connecting to the computer system, taking the secure token, and then communicating it to `Bob`, who should not have access to the token. The safety property we check is therefore that `Bob` can never possess a secure token.

```
%active Building : 0;
%active SRoom : 0;
%active Alice : 0;
%active Bob : 0;
%active Comp : 1;
%passive Phone : 1;
%passive Tok : 0;

%rule call Phone[-].Tok || Phone[-].$0 ->
   Phone[PhoneLink].Tok || Phone[PhoneLink].$0;
%rule transf_tok
   Alice.Phone[x].Tok || Bob.Phone[x].$0 ->
   Alice.Phone[-] || Bob.Phone[-].(Tok | $0);
```

```
%rule comp_connect
   SRoom.(Comp[x].Tok | Alice.Phone[y]) ->
      SRoom.(Comp[x].Tok | Alice.Phone[x]);
%rule transfer_comp Comp[x].Tok || Phone[x] ->
   Comp[x] || Phone[-].Tok;
%rule enter_room Alice.$0 | SRoom.$1 | $2 ->
   SRoom.($1 | Alice.$0) | $2;
%rule leave_room SRoom.(Alice.$0 | $1) | $2 ->
   SRoom.$1 | Alice.$0 | $2;

# The building model
Building.(
   SRoom.Comp[C1].Tok |
   SRoom.Comp[C2].Tok |
   Alice.Phone[-]
) | Bob.Phone[-];

%property secure
           !matches(Bob.Phone[x].(Tok | $0));
```

This system permits `Bob` to obtain a token from `Alice` once she downloads it onto her phone, and then communicates it to `Bob`. If we instead replace the rules `enter_room` and `leave_room` with:

```
%rule s_enter_room
   Alice.Phone[x].$0 | SRoom.$1 | $2 ->
   SRoom.($1 | Alice) | $2;
%rule s_leave_room SRoom.(Alice | $1) | $2 ->
   SRoom.$1 | Alice.Phone[-] | $2;
```

`Alice`'s phone is removed from her custody while entering the secure room, and returned to her upon exiting the room. With this substitution, we can exhaustively check the model and ensure that `Bob` cannot possibly obtain a secure token under any circumstances.

5.3.3 Example: CCS

We give one final example that represents one of the stated short-term goals of the bigraphs formalism: the analysis and representation of process calculi [83]. We give a simple encoding of the finite fragment of the Calculus

of Communicating Systems (CCS) [74], and demonstrate that a straightforward encoding permits us to usefully represent CCS terms in such a way that they may be executed directly.

For our chosen fragment of CCS, terms are given over the signature $P ::= 0 \mid c.P \mid \overline{c}.P \mid P \mid P'$, where 0 is the null process, $c.P$ is the process that accepts input on the channel c and then proceeds as P, $\overline{c}.P$ outputs on channel c and then proceeds as P, and $P \mid P$ is the *parallel composition* of two processes.

We give a translation to BigMC terms as follows, which is a simplified form of the encoding given by Milner in [83]:

$$\begin{aligned} [\![0]\!] &\stackrel{\text{def}}{=} && \texttt{nil} \\ [\![c.P]\!] &\stackrel{\text{def}}{=} && \texttt{recv[c].}[\![P]\!] \\ [\![\overline{c}.P]\!] &\stackrel{\text{def}}{=} && \texttt{send[c].}[\![P]\!] \\ [\![P \mid P']\!] &\stackrel{\text{def}}{=} && [\![P]\!] \mid [\![P']\!] \end{aligned}$$

For a given CCS term such as $\overline{x}.\overline{y}.0 \mid x.0 \mid y.0$, we end up with the BigMC agent:

```
send[x].send[y] | recv[x] | recv[y]
```

Where both `send` and `recv` are passive and of arity 1. We can give the reaction rule that enables CCS synchronisations to take place:

```
%rule ccs send[c].$0 | recv[c].$1 -> $0 | $1;
```

For which BigMC reports the discovery of the sequence of states:

```
send[x].send[y] | recv[x] | recv[y]
send[y] | recv[y]
nil
```

5.4 Reachability Checking

The task of reachability analysis over a bigraphical reactive system equates to finding some sequence of reactions $a \rightarrowtriangle^* a'$ such that $\neg P(a')$ for some property P. If no such sequence of reactions exists, then the model does not violate the property. In practice, we need to handle the state explosion. We propose to do this (in part) by avoiding the consideration of some sequences of reactions that can never lead to violation of P, as well as

reducing the number of matches we need to recompute where the match can never succeed. We present this analysis in Section 5.5.

The checking procedure operates upon a *work queue*. The initial user-supplied model is added to the work queue as the only element. We construct a "reaction graph" $G = (V, E)$. Owing to the well-defined meaning of "transition system" within bigraphs, we resist using that term, though the concept is similar; specifically, the reaction graph in this instance is labelled with reaction rules, while a bigraphical LTS is labelled with minimal contexts [83].

Algorithm 1

1. Remove the first element w of the work queue Q.
2. For each reaction rule, find all matches $m_1 ... m_n$ in w.
3. Apply each match $m_1 ... m_n$ to w to form the new set of bigraphs B.
4. For each $b \in B$, check whether b is already in the reaction graph G.
   - if $b \in G$, add an edge from w to b labelled with the match that was applied to obtain b from w.
   - if $b \notin G$, insert b into G and add an edge from w to b. Add b to the tail of the work queue Q.
5. Check each property $p \in P$ against w.
   - If $p(w)$ is true, continue.
   - If $p(w)$ is false, report a violation. A counter-example trace from the starting state to the violating state is generated by traversing the inverse of G from w back to the root. Cycles are resolved by considered incoming edges to be sorted based on the order in which they were discovered, and always taking the smallest (i.e., the first discovered) edge.
6. Repeat the procedure for the next item in the work queue, terminating successfully if the work queue is empty.

There is no guarantee that this algorithm will terminate — indeed, some models exhibit infinite behaviour. Instead, we introduce a user-configured maximum step count, which limits the number of iterations performed within this algorithm to at most some fixed constant. This can be used

to partially explore a model, or merely to bound the computation for some finite subset of an otherwise infinite-state model. We will revise this algorithm in the next section, after introducing a static analysis on reaction rules. However, this version of the algorithm is the one implemented in the main branch of BigMC, and it is the performance of this algorithm that we characterise in Section 5.6.

5.5 Static Analysis of Rules

We present a static analysis for reaction rules that permits us to avoid checking some agents entirely, and possibly check some infinitely large state spaces by excluding statically some matches that can always be shown to be free of safety property violations. Because bigraphs are a meta-calculus, and BigMC therefore instantiates to a reachability checker for many different languages, we limit our analysis to the more general problem of pruning entire branches of the transition to avoid checking them altogether, rather than relying on any *a priori* knowledge of the semantics of a particular calculus to avoid (for example) checking of equivalent interleavings.

We define *causation* on reaction rules, saying $\overline{R}$ *causes* $\overline{S}$ if the application of some reaction rule $\overline{R}$ to an agent a could cause the redex of some reaction rule $\overline{S}$ to be matched in some a', where a' is the result of applying $\overline{R}$ to a. Deciding this exactly for all agents need not be easy, however. Instead, we present a *conservative approximation* of this causation predicate that is guaranteed to preserve non-causation, which we give as a binary relation $\overline{R} \rightsquigarrow \overline{S}$.

It is sufficient to examine place graph *paths* in order to safely approximate causation. A rule $\overline{R}$ will only cause another rule $\overline{S}$ if the reactum of the former overlaps the redex of the latter in some way. The contents of these terms are known; however, they will be nested inside unknown contexts, and may have unknown parameters. Therefore we can only consider these redexes and reactums in isolation, and we cannot rely on the contents of contexts or parameters. We can consider the cases in which these overlaps can occur in terms of intersections of paths.

Definition 58 (Paths, Prefixes, Suffixes and Subsequences)**.** For any well-formed term t, $P(t)$ is the set of complete paths in the place graph of t from

the roots terminated by either *nil* or a hole.

$$\begin{aligned}\mathrm{Pref}(t) &\stackrel{\mathrm{def}}{=} \{p : pq \in P(t)\}\\ \mathrm{Suff}(t) &\stackrel{\mathrm{def}}{=} \{qr : pqr \in P(t) \wedge r \in \{\langle nil\rangle, \langle \$n\rangle\}\}\\ \mathrm{Subs}(t) &\stackrel{\mathrm{def}}{=} \{q : pqr \in P(t)\}\end{aligned}$$

Definition 59 (Orthogonality of terms)**.** Determining that two parametric terms R and R' do not overlap by inspection of their place-graph paths is defined as:

$$\begin{aligned}R \perp' R' \stackrel{\mathrm{def}}{=} &\neg\exists p.nil \in P(R) : p.nil \in \mathrm{Suff}(R') \wedge\\ &\neg\exists p \in \mathrm{Pref}(R) : p\$n \in \mathrm{Suff}(R') \wedge\\ &\neg\exists p\$n \in P(R) : p \in \mathrm{Subs}(R') \wedge\\ &\neg\exists p\$n \in P(R) : p\$n \in \mathrm{Suff}(R')\\ R \perp R' \text{ iff } &R \perp' R' \wedge R' \perp' R\end{aligned}$$

Definition 60 (Causation)**.** For any two reaction rules $\overline{R} = R \to R'$ and $\overline{S} = S \to S'$, $\overline{R} \rightsquigarrow \overline{S}$ iff $R' \not\perp S$, where R' and S are prime and non-trivial (not empty or consisting only of holes).

This definition guarantees non-causation, as any way in which a redex and reactum may overlap has been considered in terms of the existence of overlapping place graph paths. For example, a rule $\overline{R} = \texttt{a.b} \to \texttt{a.b.c}$ could potentially cause another rule $\overline{S} = \texttt{b.c} \to \texttt{d.e}$, as there exists a path $\langle b, c\rangle$ of the redex S that is a suffix of the path $\langle a, b, c\rangle$ of the reactum R'. This is sufficient to demonstrate that under some circumstances, $\overline{R} \rightsquigarrow \overline{S}$, as $R' \not\perp S$. Notice, however, that $S' \perp R$, and therefore $\overline{S} \not\rightsquigarrow \overline{R}$.

Definition 61 (Occurrence [83])**.** We define $R \hookrightarrow R'$ iff $\exists C, D.R' = C \circ R \circ D$, where R is a redex, R' is a reactum, C is a context and D is a (possibly wide) parameter.

Lemma 6. $\overline{R} \rightsquigarrow \overline{S} \implies \overline{R}$ causes $\overline{S}$

Proof. Assume $\overline{R} \not\rightsquigarrow \overline{S}$ for some reaction rules R and S for which $\exists a, a', b.a \rightarrowtriangle_{\overline{R}} a' \rightarrowtriangle_{\overline{S}} b$, and which for no c it is $a \not\rightarrowtriangle_{\overline{S}} c$. We can decompose a' thus: $a' = C \circ R' \circ d = D \circ S \circ e$ by the rules of bigraph matching. We then consider the ways that $R' \circ d$ and $S \circ e$ could be related by $\hookrightarrow$:

Case. $R' \circ d \not\rightarrow S \circ e$ and $S \circ e \not\rightarrow R' \circ d$, we can show the matches to be disjoint. If there is some top-most node of $R' \circ d$, then we can extend this path upward towards the root such that we intersect with some top-most node of $S \circ e$, or vice versa. If there is no such path, then because they are both prime, the matches are disjoint and $\overline{S}$ must have already been applicable to the agent a prior to the application of $\overline{R}$. Therefore $\overline{R}$ did not cause $\overline{S}$.

Case. Where $R' \circ d \hookrightarrow S \circ e$, we know that $P(R' \circ d) \subseteq \text{Suff}(S \circ e)$. Because $a \not\rightarrowtriangle_{\overline{S}} c$, $S \not\rightarrow d$, and therefore we can proceed by considering only the possible intersections of paths in $P(R')$ and $P(S)$:

1. R' is nested inside S such that there are no holes to consider, therefore $\exists p.nil \in P(R') : p.nil \in \text{Suff}(S)$.
2. R' is nested entirely inside $S \circ e$, and therefore $\exists p \in \text{Pref}(R') : p\$n \in \text{Suff}(S)$.
3. $R' \circ d$ is nested entirely inside S, and therefore $\exists p\$n \in P(R') : p \in \text{Subs}(S)$.
4. $R' \circ d$ is nested entirely inside $S \circ e$, and therefore $\exists p\$n \in P(R') : p\$n \in \text{Suff}(S)$.

These four cases are exhaustive, as there are no additional ways in which $R' \circ d$ could be embedded within $S \circ e$. These rules precisely negate our definition of orthogonality, and any one of these conditions is sufficient to demonstrate that $R' \not\perp S$. However, by the definition of $\overline{R} \not\rightsquigarrow \overline{S}$, we know that $R' \perp S$ and we therefore have a contradiction.

Case. Where $S \circ e \hookrightarrow R' \circ d$, because $R' \perp S$ is symmetric, we can discharge this case by the same reasoning as the previous case where R' is substituted for S and vice versa.

Case. Where $R' \circ d \hookrightarrow S \circ e$ and $S \circ e \hookrightarrow R'$, $R' \circ d = S \circ e$, and therefore either of the previous two cases may be applied to discharge this case.

Having discharged these four cases, we can conclude that there is no such agent a for which we could have $\overline{R} \not\rightsquigarrow \overline{S}$ where R causes S, therefore $\overline{R} \rightsquigarrow \overline{S} \implies \overline{R}$ causes $\overline{S}$. □

Theorem 7. For every agent a and any property M that is prime, non-empty and non-trivial, we show by induction on the length of the reaction sequence that if $M \not\rightarrow a$ and $\forall \overline{R} \in \mathcal{R}.\neg(\overline{R} \rightsquigarrow^* M)$, then $\forall a \twoheadrightarrow^* a'.M \not\rightarrow a'$, where $\rightsquigarrow^*$ is the transitive closure of the $\rightsquigarrow$ relation.

Proof. Assume some sequence $a_m \rightarrowtriangle \ldots \rightarrowtriangle a_{n-1}$ for which $M \not\rightsquigarrow a_m \wedge \cdots \wedge M \not\rightsquigarrow a_{n-1}$. We extend this sequence with some reaction $a_{n-1} \rightarrowtriangle_{\overline{R}} a_n$. We aim to show that $M \not\rightsquigarrow a_n$:

For $n = 0$, this is trivial. For $n > 0$, assume that $M \hookrightarrow a_n$. From Lemma 1 we know that where $\overline{R} \not\rightsquigarrow M$, we can only find a reaction sequence $a_{n-1} \rightarrowtriangle_{\overline{R}} a_n \rightarrowtriangle_{M \to M} b$ iff there was already a possible reaction $a_{n-1} \rightarrowtriangle_{M \to M} b$ present for a_{n-1}, and therefore iff $M \hookrightarrow a_{n-1}$ (such that we could find a decomposition $a_{n-1} = C \circ M \circ d$ to enable this reaction).

Having $M \not\rightsquigarrow a_0$ and $M \hookrightarrow a_n$ iff $M \hookrightarrow a_{n-1}$, by induction on the length of the reaction sequence we can conclude that $\forall a \rightarrowtriangle^* a'.M \not\rightsquigarrow a'$. □

We can potentially use our causation graph for two optimisations. When considering matching, for some agent a where reaction rules $\overline{R_1}$ and $\overline{R_2}$ apply, to the resulting agents a_1 and a_2 we need only match rules (potentially) caused by $\overline{R_1}$ or $\overline{R_2}$. Similarly, where we have some property $\neg M$ and a (possibly infinite) sequence of reaction $a \rightarrowtriangle^*_{\overline{R} \in \mathcal{R}} a'$, where $\neg\exists \overline{R} \in \mathcal{R}.\overline{R} \rightsquigarrow^* M$, we need not check this sequence of reaction any further, as it can never lead to a violation of the property $\neg M$. The latter is the optimisation that permits checking of some models with infinitely many configurations, provided we can exclude all such infinite branches.

The reachability analysis algorithm that uses this causation analysis is given as follows for an agent b, a work queue $W = \langle b \rangle$, single property p, a set of reaction rules $\mathcal{R}$, and a reaction graph $G = (V, E)$, where $E \subseteq V \times V \times \mathbb{P}(\mathcal{R})$, which initially contains only the node b, and an edge $(b, b, \mathcal{R})$.

Algorithm 2

1. Take w where $W = w; W'$
2. Check the property p against w.
   - If $p(w)$ is true, continue.
   - If $p(w)$ is false, report a violation and stop.
3. For the incident edges of w in G, compute the set of "live" reaction rules $\mathcal{L}$ by:
$$\mathcal{L} \stackrel{\text{def}}{=} \{\overline{R} \mid (v_0, w, \mathbb{R}) \in E \wedge \overline{R} \in \mathbb{R}\}$$
4. If $\neg\exists \overline{R} \in \mathcal{L}.R \rightsquigarrow^* \neg p$, return to 1 with $W = W'$.
5. For each $\overline{R} \in \mathcal{L}$, compute a set of matches for $\overline{R}$ in w.

6. For each match m, apply $\overline{R}$ to w to obtain w', updating the following:
   - To G, add the node w'.
   - To G, add the edge $(w, w', \mathcal{L} \cup \{\overline{S} \in \mathcal{R} \mid \overline{R} \rightsquigarrow \overline{S}\})$.
   - Update the work queue, so that $W = W'; w'$
7. Repeat the procedure from Step 1 for the next item in the work queue, terminating successfully if the work queue is empty.

This version of the checking algorithm is implemented in an experimental branch of BigMC; however, the implementation is not sufficiently robust at present to enable a direct performance comparison. We expect that this algorithm could be implemented in several of the existing bigraph tools, including BigraphER [27] and SBAM [61].

As an example of the kinds of system for which the static analysis is be effective, consider active controls `a`, `b`, and `c`:

```
rule ra a.$0 -> a.a.$0;
rule rb b -> c;
%property notc !matches(c);
```

Given an initial system `a | b`, despite `a` exhibiting infinite behaviour, we can avoid non-termination by inferring that rule `ra` can never cause `matches(c)` by the analysis given above.

5.6 Performance

We present a preliminary characterisation of the performance of the BigMC tool in Fig. 5.4 and Fig. 5.5. The trials were performed using BigMC version 20120207 on a Macbook Air at 1.86GHz with 2GB of RAM. The plotted data were collected across 3 repeated trials for each condition to obviate the effects of transient system load. Each condition was a model of n philosophers, for $n \in 3..10$ executed with the deadlock-free property, with the reaction rules being held constant across all trials.

Fig. 5.4 gives an account of the time performance of BigMC across the number of states that were explored in the course of the checking procedure. Checking terminates when a violation is found, and therefore the figures reflect the measured user-space time (using the MacOS X `time` system utility) from initialisation until the violation was found, reported, and BigMC terminated. We can see that the time to find the violation is superlinear in

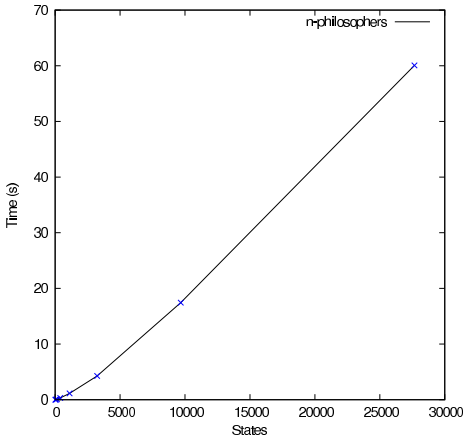


Figure 5.4: Performance of BigMC v20120207 on n-philosophers, for 3..10, where the y-axis is the time before the first counter-example was reported, and the x-axis is the number of states that were explored before finding the counter-example

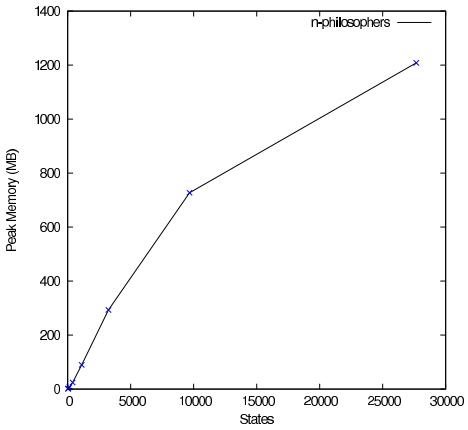


Figure 5.5: Peak memory usage in Megabytes of BigMC v20120207 on n-philosophers, for 3..10, where the y-axis is the peak measured memory usage first counter-example was reported, and the x-axis is the number of states that were explored before finding the counter-example

the number of states, although not dramatically so. Collecting timing information beyond approximately this number of states became impractical, as the effect of physical memory exhaustion became insurmountable (which in practical terms corresponded to non-termination).

Fig. 5.5 demonstrates the peak memory usage of BigMC while executing the same checking operations. Measurements of peak memory usage were made on separate trials from time performance to avoid measurement effects. Data were collected using the Mac OS X system `ps -orss` command with an inter-sample period of 100ms. The peak memory usage appears to scale in a sublinear manner under the measured conditions. This is due to the peak memory requirement being when the largest number of elements are being actively computed in the work queue. For a given application, while the size of individual states increases for larger numbers of philosophers, the number of new states that must be stored at each step does not increase linearly, hence the sublinear scaling of memory usage on this particular application. It is unclear as to whether this would be replicable for all applications.

5.7 Conclusion

We have presented a tool for execution and reachability analysis of bigraphs that instantiates to a verification environment for any given domain-specific modelling language encoded as bigraphs. The tool is GPL licensed, and is available from `http://bigraph.org/bigmc`. While the examples presented in this work are small owing to the early stage of this work, we believe it demonstrates the promise of such an approach, particularly in extending the reach of a domain-specific modelling languages to problem domains that require tool support for systematic analysis. The static analysis of reaction rules that we have presented greatly improves the tractability of checking of these kinds of models, and provides a good basis from which to explore further optimisations.We intend to investigate better heuristics for rule causation analysis, as well as attempting to apply techniques from model checking (such as excluding equivalent concurrent interleavings from consideration by partial-order methods [47]) to this new domain. Finally, we hope to consider the effect of using our causation analysis in other tools.

Chapter 6

Scaffolding the Tower: Modular Construction of Bigraphical Reactive Systems

Abstract

It has been more than 10 years since Milner introduced the bigraphical reactive systems meta-model with the aim of capturing the space and motion of ubiquitous computing systems within a 'tower of models'. In the following years, subsequent research has continually evolved and expanded the theory. With this work, we aim to reify the vision of a tower of models by providing an exposition of several significant developments within bigraphs, as well as presenting a number of novel unifying contributions towards enabling the modular construction of modelling languages related by vertical refinement. We show that modular construction is a natural fit for vertical refinement between bigraphical reactive systems.

6.1 Introduction

The development of *bigraphical reactive systems* (BRSs) by Milner and colleagues [83] has long held the promise of enabling a unified approach to the instantiation of rigorous *domain-specific modelling languages* (DSMLs), i.e., formal languages supporting rigorous analysis, for which the modelling constructs are specialised to some domain, instantiated within a "tower" of (informatic) models [84]. Indeed, the general technique of deriving a bisimulation congruence for any BRS [65] and the ability of bigraphs to easily

describe the syntax and semantics of diverse modelling languages for concurrent and mobile systems, as well as the fundamental physical structures of space and connectivity, makes them a promising candidate as a generic host for DSMLs and their analysis, for both computational and physical systems, and their combination as cyber-physical systems [69].

In claiming to enable *modular* construction of DSMLs, we really mean the act of specialising and combining general patterns to some particular domain. The fact that we can make certain guarantees about the preservation of properties through this specialisation and combination is of benefit, and as a result is a departure from existing approaches to the construction of DSMLs.

Bigraphs have been successful as a meta-model in capturing the syntax and semantics of many different formalisms (e.g., λ-calculus [82], CCS [81], the Mobile Ambients calculus [64], several variants of the π-calculus [64, 26, 39], Fusion Calculus [50], and Petri Nets [71]), and in being used directly as a modelling formalism in diverse domains such as business processes [56, 105], context-aware computing [20], and systems biology [68, 33, 34]. However, the promise of bigraphs as a unifying framework for the ad hoc instantiation of DSMLs has not materialised. The main contribution of this work is to present a mechanism and language for combining smaller bigraphical reactive systems to form larger ones, This language provides modularity properties in the construction of DSMLs, and promotes reuse of language components, which represents a significant step towards the instantiation of domain-specific modelling languages within a bigraphical meta-model.

We consider the modular construction of domain-specific languages in terms of the BMC meta-calculus that we proposed previously [59], as we believe this considerably simplifies the presentation of the modular construction process. We believe that these modular construction operations on DSMLs have correlates in a categorical setting, which may lead to further insights and more powerful analytical techniques, but of this we will say more in Sec. 6.8.

A refinement is a particular kind of behaviour-preserving map that permits one to model systems at different levels of abstraction, or to capture implementation details in lower-level models. Vertical refinement relations are those that relate entire languages to one another, and as a result are appropriate for capturing the kinds of relationships between the domain-specific modelling languages we will construct. We use previous work on bigraphical refinement [91] to assist the construction of "well-behaved" modular language designs.

6.1.1 Contributions

In defining the core of a bigraphical term language in his 2001 technical report on bigraphs [77], Milner speculated (page 46): "The rest of such a language should allow a user to specify a signature and a set of reaction rules. It may also take advantage of some [extensions to bigraphs], in particular the ability to provide names with scope". The main contribution of the present work is to propose such a language that will allow us to specify domain-specific modelling languages as bigraphical reactive systems in a principled manner. Similarly, we begin to show that with an appropriate notion of vertical refinement, it becomes practical to begin to build towers of models that permit many formally-related domain-specific languages to exist within a common framework, such that models on higher levels describe or explain those on lower levels.

6.1.2 Structure

The rest of this work is structured as follows: In Section 6.2, we recall the essential definitions of the Bigraphical Meta-Calculus (BMC) that we proposed previously [59]. In Section 6.3, we introduce a language for the modular construction of bigraphical reactive systems, motivated by several small examples of modules and their composition. In Section 6.5, we show that the same mechanisms introduced in Section 6.3 for describing modular composition are also an appropriate for describing vertical refinement relations between different languages described as bigraphical reactive systems, permitting a formal relationship between languages to be described. Finally, in Section 6.8 we offer a summary of the present work, and a number of directions for future work.

6.2 Recalling BMC

We have previously presented BMC (Bigraphical Meta-Calculus) as a meta-process language that is complete with respect to bigraphical reactive systems. We briefly recall BMC here, though full details are available elsewhere [59]. BMC corresponds exactly to bigraphical reactive systems, but provides a convenient and familiar process calculus-style representation of bigraphs, which obviates mostly the need to manipulate the underlying categorical structure of bigraphs.

Definition 62 (BMC Signature)**.** A pure *bigraph signature* is a tuple (Σ, ar, Σ_a), where Σ is a set of *controls*, $ar : \Sigma \to \omega$ is a function assigning a finite ordinal (the arity) to each control, and $\Sigma_a \subseteq \Sigma$ is the subset of *active* controls. We will often refer to a signature (Σ, ar, Σ_a) by simply the set of controls Σ. In order to compactly represent signatures, for a signature where $\Sigma = \{\mathsf{c_1}, \ldots, \mathsf{c_n}\}$, with $\Sigma_a \subseteq \Sigma$ and $ar(\mathsf{c_i}) = ar_i$, we will write:

$$(\{\mathsf{c_1} : ar_1, \ldots, \mathsf{c_n} : ar_n\}, \Sigma_a)$$

As a notational convenience, we will sometimes use the notation $\mathsf{c}_a : n$ to indicate an active control c with arity n.

Definition 63 (BMC Terms)**.** The set of ground (process) BMC terms for a given instance calculus are given by the following grammar, with respect to a signature Σ:

$p ::=$	nil	Nil process
	$\mathsf{c}\langle\bar{x}\rangle.p$	Prefix
	$p \mid p$	Parallel process
	$(\nu x)p$	Restriction

for $\mathsf{c} \in \Sigma$, $\bar{x} \in N^*$ and $|\bar{x}| = ar(\mathsf{c})$.

Contexts for processes are those terms into which other processes may be inserted. These are also used to describe *parametric* reaction rules.

Definition 64 (BMC Contexts)**.** For a signature Σ we define the (wide and prime) Σ-*contexts* r by extending the grammar of process terms as follows

$r ::=$	$\mathbf{null}$	Null process
	$r \parallel r$	Wide parallel
	p	Prime
	$(\nu x)r$	Restriction
	$^{y}/_{x}$	Linking
$p ::=$	$\ldots \mid \boxed{i}$	Prime contexts

for $x, y \in \mathsf{N}$ and $i \in \omega$.

Selective rewriting of portions of a term (which corresponds to a bigraph) is described by a set of *reaction rules*.

Definition 65 (BMC Reaction Rules)**.** A reaction rule is expressed as a triple

$$(r_L : m \to n, r_R : m' \to n, \eta : (\mathbb{N} \times \mathbb{N}) \times (\mathbb{N} \times r))$$

where r_L is a BMC context to be matched (called the *redex*), r_R is a BMC context with which the matched redex will be replaced (called the *reactum*), and η is the *instantiation map* that maps occurrences of holes and names in the reactum to the matched parameters in the redex. Where the instantiation map is unambiguous (e.g., where it is just the identity map), we may omit it and simply write rules as $r_L \to r_R$, or $(r_L \to r_R, \eta)$ to aid readability where η is significant.

In order to exclude certain terms of a given signature Σ, we can construct *sorting* predicates that act as structural conditions on terms. A full account of this logic (and its interpretation) is given in [59]. However, it is sufficient to say for now that it may be used to specify which terms should be excluded.

Definition 66 (Bigraph Sorting Logic)**.**

$$\begin{aligned}
F &\stackrel{\text{def}}{=} && U^+ \implies U^- \mid F \vee F \mid F \wedge F \\
U &\stackrel{\text{def}}{=} && v = u \mid v \neq u \mid v \,/\, u \mid v \,//\, u \mid v \not\succ u \mid v \not\nearrow u \mid \mathsf{c}(u) \mid \neg\mathsf{c}(u) \\
U^+ &\stackrel{\text{def}}{=} && v@i = u@j \mid U^+ \wedge U^+ \mid U^+ \vee U^+ \mid \mathit{local}(v@i) \mid U \\
U^- &\stackrel{\text{def}}{=} && v@i \neq u@j \mid U^- \wedge U^- \mid U^- \vee U^- \mid \neg\mathit{local}(v@i) \mid U
\end{aligned}$$

where $\mathsf{c} \in \Sigma$, $i, j \in \mathbb{N}$, and $u, v \in \mathbf{Var}$, an infinite set of meta-variables.

Briefly, formulae in this logic are implicitly universally quantified over all possible assignments of nodes to meta-variables. The nodes in this case are a bigraphical concept, but correspond to the prefixes (e.g., $\mathsf{c}\langle\bar{x}\rangle.p$). The formula $u \,/\, v$ specifies that some prefix v appears directly underneath a prefix u, while $u \,//\, v$ specifies that v may appear anywhere underneath the prefix u at arbitrary depth. The connectives $u \not\succ v$ and $u \not\nearrow v$ are the negative forms of the same connectives, respectively. The predicate $\mathsf{c}(u)$ is true iff the control assigned to the prefix u is c. The predicate $v@i = u@j$ specifies that the name in the ith position of the prefix assigned to v is the same as the name in the jth position of the name assigned to u. Finally, $\mathit{local}(v@i)$ specifies that the ith name of v exists under a name restriction such as (νz). The logic specifies that positive statements appear only on

the left hand side of implication, such that the formula simplify to contain only negative terms (e.g., $\neg \mathsf{c}(u)$). A formal interpretation of the terms of the logic is given in [59]. Having defined the logic for sorting predicates, we can now define sorted signatures.

Definition 67 (Sorted Signature)**.** A sorted bigraph signature A^ϕ consists of a signature A and a sorting predicate ϕ, such that:

$$A^\phi \stackrel{\text{def}}{=} (A, \phi)$$

Definition 68 (Language Terms)**.** As a notational convenience, we write $\mathbf{BMC}(A^\phi)$ for the set of all well-sorted terms of a given sorted signature A^ϕ.

6.3 A Language from Atoms

We consider the construction of a domain-specific modelling language in terms of *atoms* — i.e., the simplest (irreducible) building blocks of a BMC instance calculus (and therefore of a bigraphical reactive system). There exist three basic types of atoms: *singleton signatures*, *reactions*, and *sortings*. We use these atoms to construct general "patterns" for language features that may be combined and specialised in a principled manner, such that we reduce the effort involved in constructing (and reasoning about) instance calculi.

A domain-specific language module is defined as a pair, consisting of a (sorted or unsorted) signature, and a set of *signature-dependent* reaction rules. The former component gives the static structure of terms, while the latter imbues them with dynamics. The fact that the rules are *signature-dependent* is significant, and will be explained fully in Section 6.3.2.

We can envision a library of predefined language components that distill common functionality and general patterns in a way that permits reuse. We will initially present the constructs of our modular construction mechanism, and then exemplify it by building generic, reusable language components that may be specialised for a particular domain.

We first introduce the syntax of our language for modular construction of instance calculi, and will devote the next several sections to explaining the meaning of these constructs:

Definition 69 (Modular Construction Terms)**.**

$L \stackrel{\text{def}}{=} \mathsf{c} : \mathbb{N}$	Passive Singleton Signature
c_a	Active
ϕ_Σ	Sorting Predicate
$L \circledcirc L$	Composition with predicate disjunction
$L \circledwedge L$	Composition with predicate conjunction
$L \setminus L$	Restriction
$L + [\![\mathsf{R}]\!]_\Sigma$	Reaction addition
$L - [\![\mathsf{R}]\!]_\Sigma$	Reaction subtraction
$L[\mathsf{c} := \mathsf{d}]$	Control substitution
$L[\mathsf{c} ::= \mathsf{d}\langle \mathbb{N}^* \rangle]$	Substitution with remapping

Where R denotes reaction rules as defined in BMC, $\mathsf{c}, \mathsf{d} \in \mathbf{K}$, which is an infinite set of controls, $\Sigma \subset \mathbf{K}$, and ϕ ranges over sorting predicates specified in the bigraph sorting logic of BMC.

Instance calculi are defined in terms of pairs, consisting of a signature and a set of *signature-dependent* reaction rules. We will define the constructs given above in terms of their interpretation as language pairs of this shape, and then eventually (in Section 6.4) define the mapping from these language pairs to bigraphical reactive systems.

Definition 70 (BMC Language)**.** A language over the sorted signature A^ϕ with a set of signature-dependent reaction rules $\mathcal{R}$ is defined by:

$$(\!| A^\phi, \mathcal{R} |\!)$$

Readers familiar with bigraphical reactive systems will note the similarity of this definition to the normal definition of bigraphical reactive systems given by Milner [83], with the main difference subsisting in the inclusion of a larger set of reaction rules in $\mathcal{R}$, including rules that may never become part of an eventual BRS. We formalise the mapping to bigraphical reactive systems in Section 6.4.

6.3.1 Singleton Signatures

As BMC terms (and bigraphical reactive systems) are defined with respect to a signature, we provide mechanisms for constructing signatures.

Definition 71 (Singleton Signature)**.** A *singleton signature* is of the form:

$$\mathsf{c} : ar_{\mathsf{c}}$$

where $\mathsf{c} \in \mathbf{K}$, $ar_{\mathsf{c}} \in \mathbb{N}$.

Definition 72 (Singleton Signature Interpretation)**.** A (passive) singleton signature $\mathsf{c} : ar_{\mathsf{c}}$ is promoted to a language by the following mappings:

$$\mathsf{c} : ar_{\mathsf{c}} \stackrel{\text{def}}{=} (\!|(\{\mathsf{c} : ar_{\mathsf{c}}\}, \emptyset), \emptyset|\!)$$

Introducing a control as a single signature and marking that control as *active* are performed separately, such that a control may be marked as active prior to even being introduced into a signature! This may seem counter-intuitive, but it turns out to provide the appropriate commutativity property for signature composition.

Definition 73 (Activity Interpretation)**.** For a given control c, we mark it active using the notation c_a, which is interpreted as:

$$\mathsf{c}_a \stackrel{\text{def}}{=} (\!|(\emptyset, \{\mathsf{c}\}), \emptyset|\!)$$

Notice that we revert to the normal BMC notion of a signature once we promote singleton signatures to languages — the ability to have single controls be treated as signatures is merely a notational convenience. Sorting predicates, on the other hand, are always defined with respect to a particular signature, and so must be treated more carefully.

Definition 74 (Sorting Predicate Interpretation)**.** A sorting predicate ϕ that depends on some signature Σ is promoted to a language by mapping to the pair:

$$[\![\phi_\Sigma]\!] \stackrel{\text{def}}{=} (\!|\Sigma^\phi, \emptyset|\!)$$

where Σ is the *smallest* such signature containing all of the controls used in ϕ.

Before we can introduce a notion of language composition, we require one auxiliary definition that captures the property that two signatures agree in some sense. We call this property *signature consistency*.

Definition 75 (Signature Consistency)**.** For two signatures $A = (\Sigma, \Sigma_a)$ and $B = (\Sigma', \Sigma'_a)$, we say A and B are *consistent* iff:

$$\forall \mathsf{c} \in \Sigma \cap \Sigma' : ar_A(\mathsf{c}) = ar_B(\mathsf{c}) \wedge \mathsf{c} \in \Sigma_a \iff \mathsf{c} \in \Sigma'_a$$

where $ar_A : \Sigma \to \mathbb{N}$ gives the arity of controls in A, and $ar_B : \Sigma' \to \mathbb{N}$ gives the arity of controls in B.

Signature consistency is effectively the condition that two signatures agree upon the arity and activity of controls that appear in both signatures—we will say more of this when we come to substitution of controls in Section 6.3.4. Having defined consistency, we can define the trivial composition of two unsorted signatures.

Definition 76 (Signature Composition)**.** For two signatures $A = (\Sigma, \Sigma_a)$ and $B = (\Sigma', \Sigma'_a)$:

$$A \cup B \stackrel{\text{def}}{=} (\Sigma \cup \Sigma', \Sigma_a \cup \Sigma'_a) \qquad \text{where } A \text{ and } B \text{ consistent}$$

Proposition 6. Where the signatures are consistent, signature composition is commutative and associative.

We now extend signature composition to *sorted* signatures by introducing two different notions of composition. The need for two different notions is motivated by the different ways of combining two sorting predicates, in that we can either take the conjunction or the disjunction of the two sorting predicates.

Definition 77 (Sorted Signature composition)**.** For two sorted signatures $A^\phi = (\Sigma, \phi)$ and $B^\psi = (\Sigma', \psi)$:

$$A^\phi \circledvee B^\psi \stackrel{\text{def}}{=} (\Sigma \cup \Sigma', \phi \vee \psi)$$

$$A^\phi \circledwedge B^\psi \stackrel{\text{def}}{=} (\Sigma \cup \Sigma', \phi \wedge \psi)$$

where Σ and Σ' are consistent. Unsorted signatures are assumed to be sorted by the trivial "true" predicate $\top$ (which admits all terms of the language) so that the same compositions may be applied to both sorted and unsorted signatures.

Definition 78 (Language Composition)**.** For two languages $(\!|A^\phi, \mathcal{R}_1|\!)$ and $(\!|B^\psi, \mathcal{R}_2|\!)$, we define two notions of composition:

$$(\!|A^\phi, \mathcal{R}_1|\!) \circledvee (\!|B^\psi, \mathcal{R}_2|\!) \stackrel{\text{def}}{=} (\!|A^\phi \circledvee B^\psi, \mathcal{R}_1 \cup \mathcal{R}_2|\!)$$

$$(\!|A^\phi, \mathcal{R}_1|\!) \circledwedge (\!|B^\psi, \mathcal{R}_2|\!) \stackrel{\text{def}}{=} (\!|A^\phi \circledwedge B^\psi, \mathcal{R}_1 \cup \mathcal{R}_2|\!)$$

We note here that we do not explicitly forbid the composition of languages that will result in a language without any terms—though such a language is unlikely to be particularly useful. Such a composition would result where the conjunction of sorting predicates excluded all non-empty

terms from being well-sorted, yielding a language equivalent to $(\!|\emptyset, \emptyset|\!)$. Preventing such compositions would unnecessarily complicate the definition of composition, as well as requiring a decision procedure on sorting predicates, so it has been omitted.

Having defined our two notions of language composition, we can use the same mechanisms to build signatures from singleton signatures (e.g., $\mathsf{c} : 1 \mathbin{⩓} \mathsf{d} : 2 = (\!|\{\mathsf{c} : 1, \mathsf{d} : 2\}, \emptyset|\!)$), and apply sorting predicates to unsorted signatures. Note for example that a sorting predicate ϕ_Σ, interpreted as $(\!|\Sigma^\phi, \emptyset|\!)$ can be composed with some non-empty signature A for which $\Sigma \subseteq A$, to obtain $(\!|A^\top \mathbin{⩓} \Sigma^\phi, \emptyset \cup \emptyset|\!)$, which simplifies to $(\!|A^\phi, \emptyset|\!)$.

6.3.2 Reaction

We have thus far considered only the terms of languages constructed from (sorted) signatures. We now turn our attention to the semantics of languages constructed in this manner, by considering the inclusion of reaction rules. In contrast with the normal bigraphical reactive systems method (in which a bigraphical category is equipped with a set of reaction rules), through modular construction of languages, reaction rules constructed with respect to a set of controls may be added to a system constructed around a signature that lacks those controls, or one in which the arity and activity of controls is not consistent with respect to the controls included in the reaction rules. Consequently, we permit any reaction rule to be added to any system, but these remain *signature-dependent*, and in a manner we will make precise in Section 6.4, do not become fully-fledged reaction rules until the signature of the language that contains them is sufficient.

Definition 79 (Signature-Dependent Reaction Rule)**.** A signature-dependent reaction rule consists of a reaction rule $\mathsf{R} = (R, R', \eta)$ that depends upon the presence of at least the signature Σ, and is given as:

$$\mathsf{R}_\Sigma$$

with the condition that $R, R' \in \mathbf{BMC}(\Sigma)$, and $\forall \Sigma'. R, R' \in \mathbf{BMC}(\Sigma') \implies \Sigma \subseteq \Sigma'$.

Definition 80 (Reaction Rule Equality)**.** We define two reaction rules R_Σ and $\mathsf{R}'_{\Sigma'}$ to be equal iff $\mathsf{R} = \mathsf{R}'$ and Σ and Σ' are consistent.

A signature-dependent reaction rule R_Σ may be promoted to a language by $(\!|\Sigma, \{\mathsf{R}\}|\!)$. This ability to promote reaction rules to languages is not

sufficient, however. It may in some instances be desirable to add a rule to a language and have it remain "unapplied", and therefore we provide a mechanism for the composition of languages with signature-dependent reaction rules.

Definition 81 (Reaction addition)**.** For a language $L = (\!|A^\phi, \mathcal{R}|\!)$ and a reaction rule R_Σ, reaction addition is defined:

$$L + \mathsf{R}_\Sigma \stackrel{\text{def}}{=} (\!|A^\phi, \mathcal{R} \cup \{\mathsf{R}_\Sigma\}|\!)$$

Definition 82 (Reaction subtraction)**.** For a language $L = (\!|A^\phi, \mathcal{R}|\!)$ and a reaction rule R_Σ, reaction subtraction is defined:

$$L - \mathsf{R}_\Sigma \stackrel{\text{def}}{=} (\!|A^\phi, \mathcal{R} \setminus \{\mathsf{R}_\Sigma\}|\!)$$

6.3.3 Restriction

It is often desirable to take a particular definition and specialise it by restricting the language in some way, such that unused or unwanted constructs in a language may be removed. We first define restriction on signatures, and then extend the definition to languages.

Definition 83 (Signature Restriction)**.** For two signatures $A = (\Sigma, \Sigma_a)$ and $B = (\Sigma', \Sigma'_a)$, we define signature restriction:

$$A \setminus B \stackrel{\text{def}}{=} (\Sigma \setminus \Sigma', \Sigma_a \setminus \Sigma'_a)$$

where A and B are consistent.

Extending the definition of signature restriction on unsorted signatures to sorted signatures is not entirely intuitive, as the restriction of one sorting predicate by another is not straightforward.

Definition 84 (Sorted Signature Restriction)**.** For two sorted signatures $A^\phi = (\Sigma, \Sigma_a)$ and $B^\psi = (\Sigma', \Sigma'_a)$, restriction is defined:

$$A^\phi \setminus B^\psi \stackrel{\text{def}}{=} (\Sigma \setminus \Sigma', \Sigma_a \setminus \Sigma'_a)^\phi$$

where A and B are consistent.

What happens to the sorting of B^ψ in this case? It is ignored, in the sense that the controls of B are no longer in A, and therefore the sorting is no longer well-defined with respect to $A \setminus B$. It may in some instances be desirable to "restrict" a sorting in the sense that the result will be $A^{\phi \vee \neg \psi}$, which is valid, but not always possible, as negation on arbitrary sorting predicates is not always defined.

Definition 85 (Language restriction)**.** For two languages $L_1 = (\!|A^\phi, \mathcal{R}_1|\!)$ and $L_2 = (\!|B^\psi, \mathcal{R}_2|\!)$, restriction is defined:

$$L_1 \setminus L_2 \stackrel{\text{def}}{=} (\!|A^\phi \setminus B^\psi, \mathcal{R}_1 \setminus \mathcal{R}_2|\!)$$

where A and B are consistent.

6.3.4 Substitution

We now introduce a mechanism for performing substitutions on signatures, sortings, languages, and terms, such that a language may be specialised by choosing particular controls to substitute prior to composition, controlling under what circumstances controls are merged or kept disjoint through composition.

Definition 86 (Control Substitution)**.** For a given signature $A = (\Sigma, \Sigma_a)$, substitution of the control d for the control c is defined:

$$A[\mathsf{c} := \mathsf{d}] = (\Sigma[{}^{d}\!/_{c}], \Sigma_a[{}^{d}\!/_{c}])$$

where $\mathsf{d} \notin \Sigma$

Definition 87 (Control Substitution on Sortings)**.** For a sorting predicate ϕ, we define $\phi[\mathsf{c} := \mathsf{d}]$ by induction on the structure of terms:

$$\phi[\mathsf{c} := \mathsf{d}] \stackrel{\text{def}}{=} \begin{cases} \mathsf{d}(v) & \text{if } \phi = \mathsf{c}(v) \\ \neg \mathsf{d}(v) & \text{if } \phi = \neg \mathsf{c}(v) \\ u[\mathsf{c} := \mathsf{d}] \wedge u'[\mathsf{c} := \mathsf{d}] & \text{if } \phi = u \wedge u' \\ u[\mathsf{c} := \mathsf{d}] \vee u'[\mathsf{c} := \mathsf{d}] & \text{if } \phi = u \vee u' \\ u[\mathsf{c} := \mathsf{d}] \implies u'[\mathsf{c} := \mathsf{d}] & \text{if } \phi = u \implies u' \\ \phi & \text{otherwise} \end{cases}$$

where d does not appear in ϕ.

We can now lift substitution to sorted signatures, then to terms, and eventually to languages:

Definition 88 (Substitution on Sorted Signatures)**.** For a given sorted signature $A^\phi = (A, \phi)$, substitution of the control d for the control c is defined:

$$A^\phi[\mathsf{c} := \mathsf{d}] \stackrel{\text{def}}{=} (A[\mathsf{c} := \mathsf{d}], \phi[\mathsf{c} := \mathsf{d}])$$

Proposition 7 (Consistency Preservation)**.** Substitution preserves signature consistency.

Definition 89 (Substitution on Terms)**.** On a BMC normal-form term r, substitution of the control d for the control c is defined:

$$r[\mathsf{c} := \mathsf{d}] \stackrel{\text{def}}{=} \begin{cases} \mathsf{d}\langle\bar{x}\rangle.(p[\mathsf{c} := \mathsf{d}]) & \text{if } r = \mathsf{c}\langle\bar{x}\rangle.p \\ \mathsf{c}'\langle\bar{x}\rangle.(p[\mathsf{c} := \mathsf{d}]) & \text{if } r = \mathsf{c}'\langle\bar{x}\rangle.p \text{ where } \mathsf{c} \neq \mathsf{c}' \\ (\nu z)(r'[\mathsf{c} := \mathsf{d}]) & \text{if } r = (\nu z)r' \\ r'[\mathsf{c} := \mathsf{d}] \parallel r''[\mathsf{c} := \mathsf{d}] & \text{if } r = r' \parallel r'' \\ p[\mathsf{c} := \mathsf{d}] \mid p'[\mathsf{c} := \mathsf{d}] & \text{if } r = p \mid p' \\ r & \text{otherwise} \end{cases}$$

where d does not appear in r.

Definition 90 (Substitution on Reaction Rules)**.** On a reaction rule $\mathsf{R}_\Sigma = (R, R', \eta)_\Sigma$, substitution of the control d for the control c is defined:

$$\mathsf{R}_\Sigma[\mathsf{c} := \mathsf{d}] \stackrel{\text{def}}{=} (R[\mathsf{c} := \mathsf{d}], R'[\mathsf{c} := \mathsf{d}], \eta)_{\Sigma[\mathsf{c}:=\mathsf{d}]}$$

where $\mathsf{d} \notin \Sigma$. We permit application of substitution to a set of reaction rules $\{\mathsf{R}_{1\Sigma_1}, \ldots, \mathsf{R}_{n\Sigma_n}\}$ by $\{(\mathsf{R}_1[\mathsf{c} := \mathsf{d}])_{\Sigma_1[\mathsf{c}:=\mathsf{d}]}, \ldots, (\mathsf{R}_n[\mathsf{c} := \mathsf{d}])_{\Sigma_n[\mathsf{c}:=\mathsf{d}]}\}$.

Definition 91 (Substitution on languages)**.** For a language $L = (\!|A^\phi, \mathcal{R}|\!)$, substitution of the control d for the control c is defined:

$$L[\mathsf{c} := \mathsf{d}] \stackrel{\text{def}}{=} (\!|A^\phi[\mathsf{c} := \mathsf{d}], \mathcal{R}[\mathsf{c} := \mathsf{d}]|\!)$$

where d does not appear in A, ϕ, or $\mathcal{R}$.

Example

The operations we have seen thus far on languages are sufficient to begin to build languages by specialising generic patterns to a given domain. For example, the values true and false occur frequently in many settings:

$$M_{tf} \stackrel{\text{def}}{=} \mathsf{true} : 0 \oslash \mathsf{false} : 0$$

Similarly, the idea of being able to dereference some name to obtain a value stored elsewhere is a common pattern. We could define a language component M_{var} that captures this idea, by providing both constructors for building terms in this language, as well as providing the reaction rule (i.e., the semantics) of such terms:

$$\begin{aligned} M_{var} \stackrel{\text{def}}{=} \mathsf{var} : 1 \oslash \mathsf{value} : 1 + \\ [\![\mathsf{var}\langle x \rangle \parallel \mathsf{value}\langle x \rangle.\boxed{0} \to \boxed{0} \parallel \mathsf{value}\langle x \rangle.\boxed{0}]\!]_\Sigma \end{aligned}$$

It is often desirable to abstract away the exact mechanism by which some choice occurs within a model, and simply model a choice between many alternatives as non-deterministic choice instead. We can capture this as a general pattern that implements this behaviour too:

$$M_{choice} \stackrel{\text{def}}{=} \mathsf{choice} : 0 + [\![\mathsf{choice}.\boxed{0} \mid \boxed{1} \to \boxed{0}]\!]_\Sigma$$

The behaviour of these components need not be quite so simple. If-then-else expressions (i.e., an expression like `if` c `then` x `else` y that reduces to either x or y depending on whether the condition c is true or false) occur frequently in general-purpose programming languages, as well as in specialised modelling languages. We need not replicate this functionality for every single instance calculus! It is possible to build this as a component that is readily reusable:

$$\begin{aligned} M_{if} \stackrel{\text{def}}{=} \mathsf{if} : 0 \oslash \mathsf{if}_a : 0 \oslash \mathsf{then} : 0 \oslash M_{tf} + \\ [\![\mathsf{if}.(\mathsf{true} \mid \mathsf{then}.\boxed{0} \mid \boxed{1}) \to \boxed{0}]\!]_\Sigma + \\ [\![\mathsf{if}.(\mathsf{false} \mid \mathsf{else}.\boxed{0} \mid \boxed{1}) \to \boxed{0}]\!]_\Sigma \end{aligned}$$

Example 6. How do we begin to specialise this module to a given domain? If we were to take a simple hypothetical "rule language" for describing who should be notified in case of an emergency, such that we wish to encode some simple business rules for determining notification and responsibility,

we could use the modules we have already defined to construct a language:

$$M_{people} \stackrel{\text{def}}{=} \mathsf{Alice} : 0 \mathbin{⍓} \mathsf{Bob} : 0 \mathbin{⍓} \mathsf{Carol} : 0$$

$$\begin{aligned} L_{notify} \stackrel{\text{def}}{=} {} & \mathsf{is} : 1 \mathbin{⍓} \mathsf{or} : 0 \mathbin{⍓} \mathsf{notify} : 0 \mathbin{⍓} \mathsf{orNotify} : 0 \mathbin{⍓} M_{people} \mathbin{⍓} \\ & M_{tf} \mathbin{⍓} \\ & M_{var}[\mathsf{var} := \mathsf{is}] \mathbin{⍓} \\ & M_{choice}[\mathsf{choice} := \mathsf{or}] \mathbin{⍓} \\ & M_{if}[\mathsf{then} := \mathsf{notify}][\mathsf{else} := \mathsf{orNotify}] \end{aligned}$$

Now we can construct terms such as:

$$\begin{aligned} & \mathsf{value}\langle dayShift \rangle.\mathsf{true} \mid \\ & \mathsf{if}.(\mathsf{is}\langle dayShift \rangle \mid \\ & \qquad \mathsf{notify}.(\mathsf{or}.\mathsf{Alice} \mid \mathsf{or}.\mathsf{Bob}) \mid \\ & \mathsf{orNotify}.\mathsf{Carol}) \end{aligned}$$

This example specifies a scheduling rule that determines who to notify, depending on whether the $dayShift$ value is true or not. If it is true, then either Alice or Bob will be notified, otherwise Carol will be notified. While this is a simple example, it shows how quickly we can specialise generic constructs to a particular domain's vocabulary and purposes. This is not merely syntactic sugar — indeed, the choice of modules that are composed is a genuine composition of semantics.

The fact that this is a *domain-specific* modelling language means that it is acceptable to bake constraints (e.g., business rules) into the syntax and semantics of the language itself. We could imagine that our hypothetical shift supervisor system insists that only a single person ever be responsible for supervision during the night shift. At present we could construct a term such as:

$$\begin{aligned} & \mathsf{value}\langle dayShift \rangle.\mathsf{false} \mid \\ & \mathsf{if}.(\mathsf{is}\langle dayShift \rangle \mid \\ & \qquad \mathsf{notify}.\mathsf{Alice} \mid \\ & \mathsf{orNotify}.(\mathsf{or}.\mathsf{Carol} \mid \mathsf{or}.\mathsf{Bob})) \end{aligned}$$

If we wished to forbid this situation, we could add a sorting:

$$x \mathbin{/} y \wedge \mathsf{or}(y) \implies \mathsf{notify}(x)$$

This insists that or controls appear only underneath a notify prefix, which captures our stated constraint on the structure of terms.

6.3.5 Syntax

A word on syntax: The terms that we can construct in BMC are not necessarily well-suited to every domain in terms of aesthetics or choice of symbols; however, it is perfectly possible to compile other "surface" syntax to BMC terms. This is a simpler task than writing an actual interpreter or compiler for an arbitrary language, as the source and target semantics are precisely the same—one is merely performing a kind of direct syntactic translation in the style of a macro system. We speculate in Section 6.8 that such "surface syntax" could be construed in the setting of Banana algebra [6], or a similar syntactic framework that permits modular extension of grammars, such that the semantic transformations could be mirrored by syntactic transformations.

6.3.6 Calculational sub-languages

We now consider the class of *calculational* bigraphical reactive systems, as proposed by Debois [36]. Calculational BRSs are distinguished by the fact that their behaviour is defined in terms of *unfolding*, as opposed to the conventional *liberal* reaction of a bigraphical reactive system. Calculational BRSs have desirable properties: they are Turing-complete, unfolding is confluent, and they can be shown to only enable liberal reaction, but never disable it. In this sense, a calculational language may be embedded into a host language in such a way that the composite system is "well-behaved". Our intention here is not to give a full account of calculational bigraphical reactive systems; rather, we intend merely to demonstrate a clear relationship exists, as the inclusion of calculation within instance calculi in a principled manner is a requirement on any well-designed meta-calculus.

Definition 92 (Unfolding Seeds and Rules)**.** A parametric *unfolding seed* as defined by Debois [36] is a a control c, equipped with a set of patterns Δ_{c}, each of which is a discrete bigraph $r : \langle m, \emptyset \rangle \to \langle n, X \rangle$ that does not contain any seeds. An unfolding reaction rule is one of the form:

$$(\mathsf{c}\langle \bar{x} \rangle.\mathbf{p}, U, \eta)$$

where $\mathbf{p} \in \Delta_{\mathsf{c}}$.

Definition 93 (Unfolding BRS)**.** A BRS $\mathbf{BG}(\Sigma, \mathcal{R})$ is *unfolding* if every rule $\mathsf{R} \in \mathcal{R}$ is unfolding.

Proposition 8 (Composition Preserves Unfolding)**.** For two languages L_1 and L_2 that correspond to unfolding BRSs, $L_1 \otimes L_2$ is an unfolding BRS.

The syntactic conditions on reaction rules required for a language to be calculation (i.e., to be one that contains a sub-language obeying the constraints of calculational languages) are preserved by our notion of modular language construction and composition. A full implementation of calculational BRSs would require additional treatment of activity, which we omit here. Full details and proofs are given by Debois [36].

Example 7. We could imagine building a simple language for evaluating boolean expressions:

$$\begin{aligned} L_{bool} \stackrel{\text{def}}{=} & M_{tf} \oslash \mathsf{and} : 0 \oslash \mathsf{and}_a \oslash \mathsf{or} : 0 \oslash \mathsf{or}_a \oslash \mathsf{not} : 0 \oslash \mathsf{not}_a \\ & + [\![\, \mathsf{not.true} \to \mathsf{false} \,]\!]_\Sigma \\ & + [\![\, \mathsf{not.false} \to \mathsf{true} \,]\!]_\Sigma \\ & + [\![\, \mathsf{and.}(\mathsf{true} \mid \mathsf{true}) \to \mathsf{true} \,]\!]_\Sigma \\ & + [\![\, \mathsf{and.}(\mathsf{false} \mid \boxed{0}) \to \mathsf{true} \,]\!]_\Sigma \\ & + [\![\, \mathsf{or.}(\mathsf{false} \mid \mathsf{false}) \to \mathsf{false} \,]\!]_\Sigma \\ & + [\![\, \mathsf{or.}(\mathsf{true} \mid \boxed{0}) \to \mathsf{true} \,]\!]_\Sigma \end{aligned}$$

What if we wanted to extend this to include propositional variables? It turns out we already have a mechanism for this! The M_{var} module we constructed in Sec. 6.3.4 can already replace named variables with their definitions. Consequently, it's as simple as composing the languages:

$$L_{boolv} \stackrel{\text{def}}{=} L_{bool} \otimes M_{var}$$

Which allows us to write terms such as:

$$\mathsf{and.}(\mathsf{Var}\langle x \rangle \mid \mathsf{not.Var}\langle y \rangle)$$

When composed with a suitable bigraph containing assignments of variables to values, this will evaluate terms to boolean values. However, our composition admits too many terms. For example:

$$\mathsf{Value}\langle x \rangle.\mathsf{Value}\langle y \rangle \parallel \mathsf{not.Var}\langle x \rangle$$

This would potentially allow us to leak Value-typed controls into our logical terms, which could cause our otherwise well-behaved system to get stuck! We could add an appropriate sorting to ensure that only true and false controls appear under Value prefixes:

$$\mathsf{Value}(x) \wedge x \mathbin{/} y \implies \mathsf{true}(y) \vee \mathsf{false}(y)$$

6.4 Translation to Bigraphs

The correspondence between BMC and bigraphical reactive systems has already been demonstrated [59]. We now demonstrate how the languages defined in our modular construction mechanism correspond to bigraphical reactive systems. In short, Milner's bigraph categories are immediately usable — we merely need to "filter out" the reaction rules and sortings that are not applicable because they depend upon the presence of controls that are not present in the current signature, or because (in the case of reaction rules) they are not well-sorted.

Definition 94 (Translation to BRS)**.** For a language $L = (\!|A^{\phi}, \mathcal{R}|\!)$, we give a translation to a concrete bigraph (pre)category:

$$\grave{}\mathbf{BG}((A, \phi), \mathcal{R}')$$

where $R' = \{(R, R', \eta) \mid (R, R', \eta)_{\Sigma} \in \mathcal{R} \wedge \Sigma \subseteq A \wedge \phi(R) \wedge \phi(R')\}$.

We note that it is possible to perform the inverse translation, too, by populating a language pair from a BRS.

Definition 95 (Translation from BRS)**.** For a BRS $\grave{}\mathbf{BG}((A, \phi), \mathcal{R})$, we can give a translation to a language pair:

$$(\!|A^{\phi}, \mathcal{R}'|\!)$$

where $\mathcal{R}' = \{\mathsf{R}_A \mid \mathsf{R} \in \mathcal{R}\}$.

6.5 Refinement by Construction

Given that we have provided a number of constructs by which one may compose, transform, and specialise languages, it seems natural to ask how we might ensure that we are building the *right* language. The languages we construct from modules need not be simple, and the manifest behaviour of semantic composition may not be desirable in every case. To answer this, we propose the use of bigraphical *vertical refinement* [91], which permits the use of one language as a kind of specification for another, in the sense that we can guarantee that the latter has no more behaviour than the former when a vertical refinement exists between the two.

Vertical refinements are characterised by *abstraction functors* that map the bigraphs (and traces) of some more-concrete language to the bigraphs (and traces) of a more-abstract language. The traces of a bigraphical reactive system are defined in terms of sequences of reaction.

Definition 96 (Trace, observation [91])**.** For a given BRS A, a *trace* is a (possibly infinite) sequence of bigraphs (agents) $\langle a_1, a_2, \ldots \rangle$, such that for each a_i and a_{i+1} in the sequence there is a reaction $a_i \to a_{i+1}$. If $s = \langle s_1, \ldots, s_n \rangle$ and $t = \langle t_1, \ldots \rangle$ are traces and $s_n \to t_1$, we may form the *composite trace* $s; t = \langle s_1, \ldots, s_n, t_1, \ldots \rangle$. In this case we say that t is an *extension* of s. We write $Tr(A)$ for the set of all traces of a given BRS A. If $F : A \to A'$ is a functor and $\langle a_1, a_2, \ldots \rangle \in Tr(A)$ is a trace of A, we apply F pointwise to obtain a trace $F(t) = \langle F(a_1), F(a_2), \ldots \rangle$.

We can then define a *safe* vertical refinement as the dual of some abstraction functor $F : C \to A$, where A and C are BRSs, such that the following condition holds:

$$A \overset{\text{safe}}{\sqsubseteq}_F C \quad \overset{\text{def}}{=} \quad F(\,Tr(C)) \subseteq Tr(A)^\star$$

where $Tr(A)^\star$ is the *stuttering closure* of $Tr(A)$.

The first theorem in [91] demonstrates three conditions that ensure that a given abstraction functor $F : C \to A$ gives rise to a safe refinement on two BRSs $A = BG(\Sigma, \mathcal{R})$ and $C = BG(\Sigma', \mathcal{R}')$:

1. It preserves and respects tensor.
2. It preserves active contexts.
3. It preserves reaction rules: For any reaction rule $(R, R', \eta) \in \mathcal{R}'$ (a) the F-image $(F(R), F(R'), \eta)$ is a rule in $\mathcal{R}$; and (b) for any parameter d of that rule, $\overline{\eta}(F(d)) = F(\overline{\eta}(d))$.

It turns out that these conditions are remarkably easy to satisfy when the abstraction functor F is described in terms of the transformations on languages we have described in the previous sections. This provides a straightforward means by which to describe such abstraction functors, and (under certain conditions) to derive them automatically. This makes it considerably easier to describe abstraction functors (and therefore vertical refinements) in a communicable and reusable manner, and also makes it easier to verify that we are indeed constructing the language we intended, by using higher-level languages as specifications for the lower-level domain-specific languages that we construct.

Indeed, it turns out that for *any* transformation or sequence of transformations, if the inverse operation may be described as an abstraction functor, then a safe vertical refinement exists. For example, consider the

substitution operation $L[\mathsf{c} := \mathsf{d}]$. We can construe this operation as a functor $F : L \to L[\mathsf{c} := \mathsf{d}]$, such that the functor is the one that alters the control map of bigraphs. The inverse functor $F^{-1} : L[\mathsf{c} := \mathsf{d}] \to L$ is precisely an abstraction functor, such that $F^{-1} \circ F = Id$, which equates to the identity $L[\mathsf{c} := \mathsf{d}][\mathsf{d} := \mathsf{c}] = L$. The kinds of operations for which these implied functors are really functorial are limited only by the side-conditions for the operations — for example, the requirement that d not occur in L in the examples above. The composition of functors gives us the ability to construct arbitrary abstraction functors for these operations, with the side conditions ensuring that we are constructing sensible functors (e.g., the difference between $L[\mathsf{c} := \mathsf{d}][\mathsf{e} := \mathsf{f}]$ and $L[\mathsf{c} := \mathsf{d}][\mathsf{e} := \mathsf{d}]$ is that the condition that d not appear in L is not satisfied in the latter case).

Definition 97 (Inverses for Transformations)**.** We can define the set of inverses for various operations:

$$\begin{aligned} L \curlyvee L' &\Rightarrow L \setminus L' \\ L \curlywedge L' &\Rightarrow L \setminus L' \\ L + [\![R]\!]_\Sigma &\Rightarrow L - [\![R]\!]_\Sigma \\ L[\mathsf{c} := \mathsf{d}] &\Rightarrow L[\mathsf{d} := \mathsf{c}] \\ \mathsf{c}_a &\Rightarrow L \setminus \mathsf{c}_a \end{aligned}$$

We define those operations appearing on the right hand side of the arrows to be the *safe* transformations, in the sense that they give rise to safe refinement functors, and those on the left hand side to be the *unsafe* transformations. The latter may give rise to safe refinements, but they need not do so automatically.

We proceed to show that these safe transformations are really those that give rise to safe refinements by construction.

Theorem 8 (Refinement by construction)**.** For any safe transformation $T : L \to M$ and for any language L, we have $L \overset{\text{safe}}{\sqsubseteq}_{T^{-1}} T(L)$.

Proof.

Case $(T(L) = L \setminus L')$**.** Every (active) reaction rule in $T(L)$ is a rule of L, and every agent of $T(L)$ is an agent of L, so any reaction of $T(L)$ is a reaction of L.

Case ($T(L) = L - [\![R]\!]_\Sigma$)**.** Every reaction rule of $T(L)$ is a reaction rule of L, and every agent of $T(L)$ is an agent of L, so the second vertical refinement theorem applies immediately.

Case ($T(L) = L[\mathsf{d} := \mathsf{c}]$)**.** Since c does not occur in L, and c and d must have the same activity and arity (by signature agreement), reaction rules are preserved structurally and active contexts are preserved.

Case ($T(L) = L \setminus \mathsf{c}_a$)**.** Seeing as some c is passive in $T(L)$ and active in L, $T(L)$ will lack any reactions under c prefixes, such that there are strictly fewer reactions, and those that remain are reactions of L.

□

This theorem coincides with the intuition that refinement effectively restricts the behaviours of the concrete system to a subset of the behaviours of the abstract system, and therefore those operations that in some sense restrict behaviour (and which therefore have inverses that add behaviour) are the "safe" ones. To this we add a corollary:

Corollary 2 (Composition preserves transformation safety)**.** For any safe transformations F and G, FG is also safe.

Proof. Immediate from the transitivity of safe vertical refinement. □

6.5.1 Relaxing Consistency

We have so far considered mostly the kinds of operations that have obvious inverses, and which therefore make it very easy to construct vertical refinement relations, in the sense that the available transformations are trivially structure-preserving. These kinds of operations, while general and useful, are insufficient for describing certain transformations that we might like to be able to express. The most important operation of this kind is substitution in which we may relax the condition that the languages have consistent signatures, such that we can modify the arity or activity of a given control.

Definition 98 (Substitution without consistency)**.** For a given signature $A = (\Sigma, \Sigma_a)$, substitution of the control d of arity ar_d and activity α_d for the control c of arity ar_c and activity α_c is defined:

$$A[\mathsf{c} ::= \mathsf{d}\langle\bar{p}\rangle] \stackrel{\text{def}}{=} ((\Sigma \setminus \{\mathsf{c} : ar_\mathsf{c}\}) \cup \{\mathsf{d} : ar_\mathsf{d}\}, \Sigma'_a)$$

where $\bar{p} \in \{1 \ldots ar_\mathsf{c}\}^*$, $|\bar{p}| = ar_\mathsf{d}$, and $\Sigma'_a = (\Sigma_a \setminus \{\mathsf{c}\}) \cup \{\mathsf{d}\}$ if α_d, or $(\Sigma_a \setminus \{\mathsf{c}\})$ otherwise.

We note that $\bar{p}$ induces a mapping from ports of d controls to ports of c controls. Consequently, we define this form of substitution on terms as in Def. 89, but replacing the case for prefixes with:

$$\mathsf{d}\langle\bar{p}(\bar{x})\rangle.(q[\mathsf{c} ::= \mathsf{d}\langle\bar{p}\rangle]) \qquad \text{if } r = \mathsf{c}\langle\bar{x}\rangle.q$$

This result may similarly be extended to reaction rules and languages, but notably not to sorting predicates—there does not appear to be any sensible "default" behaviour that one could offer in terms of transforming sorting predicates in the presence of such substitution, and therefore we leave sorting predicates unmodified in the presence of this substitution, and instead offer a proposition.

Proposition 9. For a sorting predicate ϕ, and a sorted signature A^ϕ where d does not appear in A or in ϕ, $\mathbf{BMC}(A^\phi[\mathsf{c} ::= \mathsf{d}\langle\bar{p}\rangle]) \subseteq \mathbf{BMC}(A[\mathsf{c} ::= \mathsf{d}\langle\bar{p}\rangle])$.

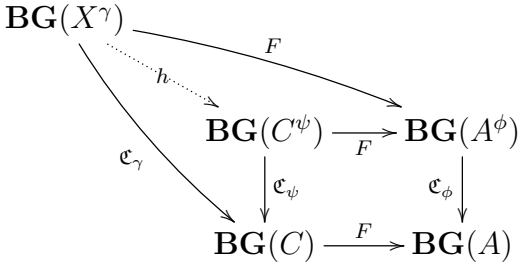
This is approximately the condition that, at worst, the substitution of one control for another will degenerate to the unsorted case, for instances where the sorting includes a specific requirement like $\mathsf{c}(x) \implies \ldots$, such that the complete absence of any c controls effectively reduces the sorting to $\mathsf{false} \implies \ldots$ (which admits all terms of $\mathbf{BMC}(A)$). Naturally, it would be just as easy to contrive a sorting and a substitution that would cause the sorting to become equivalent to false (i.e., a language with no terms).

This form of substitution is clearly just a more general instance of the consistency-preserving substitution introduced previously (i.e., they are equivalent where $\bar{p}$ is the identity map, and $ar_\mathsf{c} = ar_\mathsf{d}$). However, this substitution may arbitrarily cause sortings to no longer hold, and very few guarantees may be made about its behaviour in the general case. Knowing this, why include it? It turns out to be valuable for relating common calculi, such as making clear the link between CCS and the π-calculus. Safe vertical refinement relations may still exist when this form of substitution is used—they just need not always be easy to find.

To this end, we offer a conjecture, consisting of one sufficient condition by which to establish the existence of a safe refinement in the presence of an arbitrary sorting predicate.

Conjecture 1 (Safety without consistency)**.** Given two signatures A and C, where A is equipped with a sorting predicate ϕ, for some abstraction functor $F : \mathbf{BG}(C) \to \mathbf{BG}(A)$, what is the most permissive sorting ψ for C that we could construe that could still give rise to a safe refinement? It

turns out to be precisely that which causes $\mathfrak{C}_\psi$ to be a weak pullback along F in the following commuting diagram:



We note that for some arbitrary sorting γ, h is trivial to deduce where $X \subseteq C$ and $\gamma \implies \psi$. Such a weak pullback need not exist, and a safe refinement may still be found, but establishing this condition in order to establish whether the proposed sorting predicate is the right one may in some instances be easier.

6.6 Modular Construction of Actor Languages

The Actor Model [3, 55] is a popular model of concurrent computation that has been integrated into programming languages such as Scala [52], Erlang [7], and Haskell [40]. Actors contain local state, and communicate with other named actors to which they hold a reference by means of *messages*. Message ordering is not guaranteed (though this can be implemented), and terminated actors simply discard messages they receive.

Actors provide an interesting example system because they are used to build so-called *open systems*, in the sense that the full topology of the system is not necessarily known at design time. A compositional model-building approach seems appropriate, given that individual actors may have disparate semantics, and many different kinds of values can be passed around as messages. Also, given the tendency for other systems with their own semantics to integrate actors as a concurrency model makes them of interest as a demonstration of our modular construction mechanism. Composition of actor systems has previously been studied by Agha et al. [2, 4].

We define the semantics of actors independently of the internal logic of the actors themselves—the "actor language" used to determine the internal state of an actor and which messages that actor will send (and to whom)—will be added later by composition.

Definition 99 (Actor System)**.**

$$\begin{aligned}\mathbf{Act} \stackrel{\text{def}}{=} & \mathsf{Actor} : 1 \mathbin{⍟} \mathsf{Actor}_a \mathbin{⍟} \\ & \mathsf{ActorRef} : 1 \mathbin{⍟} \\ & \mathsf{Message} : 0 \mathbin{⍟} \\ & \mathsf{Drain} : 0 + \\ & [\![\mathsf{Actor}\langle x\rangle.(\mathsf{ActorRef}\langle y\rangle.(\mathsf{Message}.\boxed{0} \mid \boxed{1}) \mid \boxed{2}) \parallel \mathsf{Actor}\langle y\rangle.\boxed{3} \rightarrow \\ & \qquad \mathsf{Actor}\langle x\rangle.(\mathsf{ActorRef}\langle y\rangle.\boxed{1} \mid \boxed{2}) \parallel \mathsf{Actor}\langle y\rangle.(\mathsf{Message}.\boxed{0} \mid \boxed{3})]\!]_\Sigma + \\ & [\![\mathsf{Actor}\langle x\rangle.(\mathsf{Drain} \mid \boxed{0}) \rightarrow \mathsf{Actor}\langle x\rangle.\mathsf{Drain}]\!]_\Sigma \mathbin{⍟} \\ & x \mathbin{/} y \wedge x \mathbin{/} z \wedge y \neq z \wedge \mathsf{ActorRef}(y) \wedge \mathsf{ActorRef}(z) \implies y@1 \neq z@1\end{aligned}$$

The control Actor is used to identify actors — it is marked active because reaction should be able to take place at any time within an actor. An ActorRef is a reference to an actor that another actor may hold (or pass around as a message). It also acts as a mailbox for the remote actor, such that a Message, when placed into an ActorRef at the top-level of an Actor will be transmitted to that actor. Messages arrive at the top-level of an actor, at which point it is up to the particular actor to deal with them. The final control is Drain, which marks an actor as being inactive or failed, such that any messages sent to it may be discarded immediately without further action. The sorting we add ensures that an actor never has two ActorRefs for the same remote actor — this ensures that there is only one mailbox per local/remote actor pair.

We now consider extending the actor system with semantics for individual actors, as well as a set of values to send within messages. We construct as an example system an equipment rental service, such that one actor is configured as the "rental depot", and any number of other actors may borrow and return items at will:

$$
\begin{aligned}
\mathbf{Depot} \stackrel{\text{def}}{=} & \mathsf{Item} : 0 \mathbin{⍓} \\
& \mathsf{Request} : 1 \mathbin{⍓} \\
& \mathsf{Return} : 0 \mathbin{⍓} \\
& \mathsf{Depot} : 0 + \\
& [\![\mathsf{Actor}\langle x\rangle.(\mathsf{Depot} \mid \mathsf{Message.Return}.\boxed{0} \mid \boxed{1}) \rightarrow \\
& \quad \mathsf{Actor}\langle x\rangle.(\mathsf{Depot} \mid \boxed{0} \mid \boxed{1})]\!]_{\Sigma} + \\
& [\![\mathsf{Actor}\langle x\rangle.(\mathsf{Depot} \mid \mathsf{Message.Request}\langle y\rangle \mid \mathsf{Item} \mid \mathsf{ActorRef}\langle y\rangle.\boxed{0} \mid \boxed{1}) \rightarrow \\
& \quad \mathsf{Actor}\langle x\rangle.(\mathsf{Depot} \mid \mathsf{ActorRef}\langle y\rangle.(\boxed{0} \mid \mathsf{Message.Item}) \mid \boxed{1})]\!]_{\Sigma} \mathbin{⍓} \\
& \mathbf{Act}
\end{aligned}
$$

It is now possible to define a trivial client actor that borrows and returns items continuously.

$$
\begin{aligned}
\mathbf{Client} \stackrel{\text{def}}{=} & \mathsf{Item} : 0 \mathbin{⍓} \\
& \mathsf{Request} : 1 \mathbin{⍓} \\
& \mathsf{Return} : 0 \mathbin{⍓} \\
& \mathsf{Client} : 0 + \\
& [\![\mathsf{Actor}\langle x\rangle.(\mathsf{Client} \mid \mathsf{Item} \mid \mathsf{ActorRef}\langle y\rangle.\boxed{0}) \rightarrow \\
& \quad \mathsf{Actor}\langle x\rangle.(\mathsf{Client} \mid \mathsf{ActorRef}\langle y\rangle.(\mathsf{Message.Return.Item} \mid \boxed{0}))]\!]_{\Sigma} + \\
& [\![\mathsf{Actor}\langle x\rangle.(\mathsf{Client} \mid \mathsf{Message}.\boxed{0} \mid \boxed{1}) \rightarrow \\
& \quad \mathsf{Actor}\langle x\rangle(\mathsf{Client} \mid \boxed{0} \mid \boxed{1})]\!]_{\Sigma} + \\
& [\![\mathsf{Actor}\langle x\rangle.(\mathsf{Client} \mid \mathsf{ActorRef}\langle y\rangle) \rightarrow \\
& \quad \mathsf{Actor}\langle x\rangle.(\mathsf{Client} \mid \mathsf{ActorRef}\langle y\rangle.\mathsf{Message.Request}\langle x\rangle)]\!]_{\Sigma} \mathbin{⍓} \\
& \mathbf{Act}
\end{aligned}
$$

The composition **Depot** ⍓ **Client** gives us a system appropriate for constructing models of arbitrarily many depots and clients. What if we wished to add another type of client? For example, we could construct another language by adding a **Thief** that never returns items:

$$\mathbf{Thief} \stackrel{\text{def}}{=} \mathbf{Depot} \mathbin{⍓} \mathbf{Client} \mathbin{⍓} (\mathbf{Client}[\mathsf{Client} := \mathsf{Thief} : 0] \setminus \mathsf{Return} : 0)$$

We note that the latter system is trivially a safe vertical refinement of the former (the abstraction functor maps Thief controls to Client controls),

though it is unlikely to be a live refinement in the presence of any sensible choice of admissible traces. The ability to compose new clients (with different semantics) gives the general flavour of how one might consider these kinds of open systems problems using our notions of composition. It would not require a particularly more complex model of actors to capture the semantics of real-world actor systems such as *Akka* [51], which would provide avenues for integrating both abstract and concrete models of actor systems within a single "tower" of models.

While we have chosen a particularly simple internal language for the actors in this case (e.g., Request and Return commands), a more realistic scenario would likely embed some kind of computational sub-language per-actor, culminating in a system with markedly different semantics for different actors.

6.7 Related Work

Birkedal et al. [20] noted that there are two ways in which to consider the composition of *parametric* reaction rules: merge the rules first and then ground them, or ground the rules and merge the resulting (possibly-infinite) sets of ground rules. They observe that in general the result of the former is a superset of the latter, and therefore choose the former approach. We follow their example and choose to first merge rules and then ground them when considering parametric reaction rules. Similarly, the plato-graphical models of that work provides one of the first examples of principled composition of languages, although in that instance there was no intention to consider all general possible compositions, rather it was specialised for the particular use case. The composition in that work can be considered one of the inspirations of the present work.

The *Banana algebra* of Andersen & Brabrand [6] is a kind of syntactic-equivalent to the kinds of semantic composition and extension we have presented. That work provides mechanism for the modular construction and transformation of descriptions of the syntax of languages, but does not extend to considering the semantics of such languages, and is therefore somewhat orthogonal (though complimentary) to the work we have presented. The two approaches employ many of the same kinds of operations, owing to the similar goals of the task. There may be an interesting avenue for future work in attempting to relate these approaches more closely, such that the "surface" syntax of languages and the semantics could be described in a unified manner.

In his work on closure sortings, Debois [35] briefly considered composition of sortings. He noted in that context that because sortings are expressed as functors $F : \mathfrak{C}_\phi \to C$, and functors are by their nature composable, there is a relatively natural notion of composition suggested by this. We consider this interesting future work to consider the role of composition of sorting functors in deriving other ways to combine sortings.

Pereira et al. [90] proposed *BiAgents*, which is an extension of bigraphs that includes certain kinds of agent-based systems as first-class entities within bigraphs. We considered this work as an example of the kinds of systems that should be recoverable within our modular construction mechanism, such that even relatively complex bigraphical extensions may be represented in pure bigraphs (with declarative sortings), rather than requiring extensions to the underlying theory, with all the consequences that such extensions entail (e.g., loss of generality, lack of transferability of results). We have yet to recover BiAgents fully within pure bigraphs, although we believe it to be possible (as a calculational sub-BRS).

6.8 Conclusion

We have presented a number of operations on atoms that permit the definition and specialisation of bigraphical reactive systems in a modular fashion. In addition to this, we have demonstrated how existing work on vertical refinement of bigraphical reactive systems transfers to this setting, and how the same mechanisms can be used to construct abstraction functors. Specifically, we show that there is a subset of "safe" transformations that always give rise to a safe vertical refinement. We also proposed a sufficient condition for safe vertical refinements in the presence of arbitrary sortings. Finally, we have exemplified this modular construction approach on a simplified actor model, demonstrating that this approach may be used to extend the reach of bigraphical systems to modelling "open systems" in a natural way.

It remains unclear what limitations exist in the modular construction approach we have proposed. It is sufficient to say that it appears promising, although only further experimentation will determine whether this is immediately usable for a wide range of applications, or whether further extension will be required to encompass other modelling domains. We believe that construing the modular construction mechanisms presented thus far in terms of the forthcoming novel categorical presentation of bigraphs by Miculan & Peressotti [73] could yield further insights into the full promise

of this approach. We consider that giving a formal semantics for our modular construction calculus in that setting will permit more of the categorical tools available in that setting to be used to reason about language equivalence and potentially give rise to stronger conditions for "well-behaved" modular composition.

6.8.1 Future Work

We believe there are several important aspects to consider as future extensions of this work. Fully characterising the connection to Debois' work on calculational BRSs [36] should permit sensible ways of including "normal" computation within DSMLs constructed in this way.

We consider the potential role of *activity* in limiting the general applicability of the modular DSML construction approach. Activity as defined in bigraphical reactive systems is somewhat inflexible. While it is appropriate for the encoding of process calculi and many other languages, we can imagine situations in which a finer-grained notion of what constitutes an active context in which reaction may take place could be valuable. One such example might be the encoding of biological systems, where finer-grained notion of activity can permit constraints of the system to be captured more directly, and in a modular fashion more fitting to the general DSML instantiation approach. We propose that our existing sorting logic could be adapted to this purpose, such that a language is defined as a (sorted) signature, a set of reaction rules, and a predicate for each reaction rule on contexts that determines whether it is "active" or not (i.e., whether it represents a valid execution context for that rule). The investigation of whether this represents a viable generalisation of bigraphical reactive systems represents interesting future work.

Acknowledgements

This work was funded in part by the Danish Research Agency (grant no.: 2106-080046) and the IT University of Copenhagen (the Jingling Genies project).

Chapter 7

Conclusion

We have now presented the primary technical contributions of this thesis. In this chapter we summarise the results and contributions of the thesis, and discuss future directions for research into the instantiation of domain-specific modelling languages as bigraphical instance calculi.

7.1 Summary

Chapter 1 introduced the general challenge of constructing towers of models, and laid out a domain-specific modelling language approach as a means of realising this vision. We reviewed a number of other approaches to meta-modelling and the construction of domain-specific modelling languages, and outlined the contributions of this thesis.

Chapter 2 we saw a general introduction to the history and motivation behind bigraphical reactive systems, including the genesis from models that abstract away all but the communication behaviour of systems to obtain a view of a system that is amenable to analysis. We saw the formal underpinnings of bigraphical reactive systems, including the categorical structures, and a brief summary of some of the formal results in this area. The outcomes from the Bigraphical Programming Languages (BPL) project were discussed, and a number of general patterns for modelling within bigraphs were introduced.

Chapter 3 introduced the BMC meta-calculus that is a compact and intuitive meta-process calculus presentation of bigraphical reactive systems. We argued that this approach obviates the requirement that those wishing to use bigraphs to construct domain-specific modelling languages understand the formal mathematical underpinnings. We introduced a simple

logic for the construction of sorting predicates in the style of Debois [35]. This logic is a considerable improvement over previous methods for defining sorting predicates (which were usually ad hoc), in that it is guaranteed that any well-formed formula is decomposable, and will therefore give rise to a closure-sorted category that still has RPOs, and to which the semantic correspondence and transfer theorems may be applied. Chapter 3 also introduced the bigraphical abstract machine, which gives a graph-rewriting abstract machine execution semantics to bigraphical reactive systems, parametrised on an execution strategy. We gave a definition of fairness for bigraphical reactive systems, and showed that any "maximal" execution strategy is fair.

Chapter 4 introduced bigraphical vertical refinement, motivated by a small example reminiscent of a context- or location-based application. A notion of what constitutes an observation within a bigraph was introduced, in the form of prefix-closed traces, and then separate definitions of safe and live vertical refinement were given. A key contribution was the inclusion of two sufficient conditions for abstraction functors to give rise to safe vertical refinements; namely, the observation that structurally preserving reaction rules is sufficient to guarantee the presence of a safe vertical refinement. The chapter concluded by exploring the connection to the work of Reeves and Streader [94, 95], and positing the existence of a complimentary notion of horizontal refinement.

Chapter 5 introduced a prototype bigraph tool for performing a number of verification and validation tasks upon bigraphical reactive systems, such that it can be instantiated as a model checker or reachability checker for a particular instance calculus, or simply used to execute and simulate models in a given language. We provided a tractable heuristic for determining when two reaction rules might interfere with one another, which permitted the tool to automatically exclude certain infinite-length traces from consideration—such a heuristic (and those based upon the work of Højsgaard [61]) will be useful in terms of the implementation of the next generation of bigraphical verification, simulation, and analysis tools.

Chapter 6 provided a language for the modular construction and definition of domain-specific modelling languages in bigraphs, in attempting to reify the vision of a tower of models. The constructs of the language permit independent language components to be defined, and then combined and specialised to a particular domain. The language also provides the ability to build bigraphical reactive systems in a way that may be easily written down in an unambiguous and declarative manner, which will provide an easier path to enabling reuse of domain-specific modelling language compo-

nents. Chapter 6 also proposed a means by which the same language could be used to specify abstraction functors for vertical refinements, providing a better unifying mechanism for languages within a tower. Finally, we speculated about the relationship to the work of Miculan & Peressotti [73], and suggested that this is fertile ground for future exploration.

7.2 Contributions

Because this thesis is composed of several different publications, we now summarise the contributions of the thesis to the theory and applicability of bigraphical reactive systems, and to the instantiation of a tower of models as domain-specific modelling languages encoded as bigraphical instance calculi.

- A novel meta-calculus presentation of bigraphical reactive systems, with a convenient normal form representation.
- A sorting logic that is guaranteed to give rise to well-behaved sorting predicates.
- That checking that for a reaction $r \to r'$, r' is well-sorted is equivalent to constructing the interfaces of a closure-sorted category, which improves the implementability of sorted bigraphical reactive systems.
- A bigraphical abstract machine that instantiates to a fair abstract for any instance calculus.
- A definition of fairness for bigraphical reactive systems, given in terms of this abstract machine.
- A novel definition of vertical refinement for bigraphical reactive systems, providing the capability to formally relate different modelling languages constructed as bigraphical instance calculi.
- A sufficient condition for ensuring that abstraction functors give rise to safe refinements.
- A notion of "observation" appropriate to bigraphical reactive system refinement.
- A new tool for the execution of bigraphical reactive system models.

- A property language based upon matching that permits the expression of properties in the language of the underlying system.
- A novel tractable heuristic method for conservatively approximating reaction rule causation, which permits several optimisations in both execution and checking of system properties.
- An initial performance characterisation of the tool.
- A modular construction language for the definition and specialisation of domain-specific modelling languages as bigraphical reactive systems.
- A means of defining abstraction functors in terms of this modular construction language.
- The inclusion of sorting predicates within the vertical refinement mechanism, including a sufficient condition to make it easier to establish whether a sorting predicate gives rise to a safe refinement.
- A proposal for the generalisation of activity within bigraphical reactive systems.
- A novel presentation of an Actor model language as a BMC instance calculus.

7.3 Discussion

Having nearly reached the end of this thesis, we ask the question: Are we closer to the goal of enabling the construction of towers of bigraphical domain-specific modelling languages? We claim that we are. A number of presentational, theoretical, and pragmatic hurdles have been overcome, all of which we would argue were necessary in order to move closer to the stated goal. Returning to the original three areas we presented in the introduction, what has been achieved?

7.3.1 Defining Languages

The BMC meta-calculus we presented in Chapter 3 seems a demonstrably nicer presentation of bigraphical reactive systems for certain audiences. The learning curve associated with bigraphs has always been regarded as steep, and existing presentations have mostly walked a fine line between being

"too categorical" for audiences unfamiliar with the nuances of category theory, and insufficiently categorical for those who advocate that approach. Where the forthcoming work of Miculan & Peressotti [73] would appear to address the latter audience, we claim that the BMC presentation can more effectively persuade the former audience of the value and utility of bigraphs.

The bigraphical sorting logic similarly represents a considerable improvement upon existing ways of presenting sortings. Given that closure sortings (and predicate sortings) appear to be an effective mechanism for capturing most of the kinds of sortings that are of interest, we can hope to limit the proliferation of incompatible means of describing sortings. The requirement that sorting predicates be deconstructable is relatively arduous to establish for an arbitrary predicate, as we discovered in attempting to formulate a useful set of primitives. This work shall hopefully need not be duplicated for a very wide class of sortings that can be described in terms of the logic we presented, as the result that any well-formed formula is a deconstructable predicate makes predicate sortings more immediately accessible to a wider audience.

The modular construction mechanism presented in Chapter 6 similarly represents a more uniform way to define and present languages. In particular, if tool support for this mechanism follows, we could hope to see some degree of standardisation in the presentation of bigraphical languages. This has the effect of making descriptions of relatively complex domain-specific or general-purpose language features reusable and communicable, in that the construction of language components from smaller components lends itself to building for reuse, as well as concise descriptions of domain-specific languages by the specialisation of well-understood generic components. We need not replicate encodings of language features in subtly-incompatible ways, when we can instead just reuse those features that are of interest, specialising them for our needs, and maintaining compatibility with the original.

7.3.2 Tool support

BigMC is arguably one of the most immediately-usable bigraph tools available at present. As an open-source tool it has seen use by various individuals hoping to apply bigraphs in some way. While other tools are emerging (which is a positive trend!), at the time of writing all alternatives suffer from being either unavailable or incomplete. BigMC will surely be surpassed by a subsequent generation of bigraph tools, but the experience and techniques that resulted from its development and use will hopefully work to inform

this next generation. In particular, providing a tractable heuristic that permits optimisation will hopefully permit the development of further optimisations in bigraph tools. Similarly, being integrated directly with BigRed [42] has permitted BigRed to be more immediately useful for executing bigraphical reactive systems, and we look forward to continued development of bigraph tools.

The observation in Chapter 3 that reaction in the unsorted and sorted categories are equivalent (without needing to construct the sorted category), while effectively a corollary to the results of Debois [35], represents a significant improvement in our ability to implement predicate sortings in practice. We are aware of current efforts to leverage the bigraphical sorting logic and this result to implement support for sortings both in BigMC and in BigRed. The immediate decomposability of well-formed formulae in the sorting logic in particular makes it considerably easier to ensure that only "well-behaved" sortings are accepted by tools.

Finally, the BAM abstract machine presented in Chapter 3 represents a serious effort in relating theory and implementation, in the sense that it provides a means by which to reason about execution strategies for bigraphical reactive systems. Being able to relate a tool implementation to BAM and therefore decide whether a particular execution strategy implemented by that tool is fair or not seems a step in the right direction, although we believe there is more to be learnt about the exact strengths and weaknesses of the BAM approach.

7.3.3 Relating Languages

We have presented two mechanisms for relating languages in this thesis: vertical refinement, and modular construction. In the sense that building one language from another by some fixed set of combinators relates them, the latter form provides a concrete means of "scaffolding the tower", so that language-designers can hopefully begin to formalise the intuitions behind different "levels" of a given tower. The fact that this happens to coincide with the construction of abstraction functors for vertical refinement is particularly convenient.

Vertical refinement is already being used "in the wild", in that other researchers have been able to take the definition as published and apply it to their particular modelling task to relate different BRSs. It seems to be early days, and a complimentary notion of horizontal refinement remains elusive, but vertical refinement relations of this approximate shape seem to be the best next concrete step towards enabling a tower of models of the

sort envisioned by Robin Milner. We are excited by the prospect that the forthcoming work of Miculan & Peressotti [73] could potentially be used to advance this work on vertical refinement.

7.4 Future Work

Having seen that we have made some progress towards the goal of enabling bigraphical reactive systems as a host for domain-specific modelling languages related by vertical refinement, there remains much to do to make this a practical reality. We devote the remainder of this section to outlining some promising research directions.

It remains a challenge to learn the limitations of BMC, BAM, the bigraphical sorting logic, and the modular construction mechanisms presented in this thesis. The best way to establish this would be to continue to recover existing calculi in this setting to ensure that it remains sensible at scale. This work will hopefully include the consideration of other variations upon bigraphs, including a full recovery of stochastic bigraphs [68].

While vertical refinement appears to be immediately useful, horizontal refinement needs to be well-motivated by particular use-cases, either by attempting to recover other well-known variants of horizontal refinement (e.g., various CSP notions of refinement) in a bigraphical setting, or by using interestingly-motivated case studies to choose an appropriate definition of horizontal refinement. The further we have investigated horizontal refinement in a bigraphical setting, the less sensible a single fixed notion of horizontal refinement has appeared. We believe that further investigation of appropriate parametrised notions of "observation" in the style of Reeves & Streader [94, 95] are an appropriate starting point for the continuation of this work.

The investigation of further optimisations and enhancements for bigraph tools in general seems an important goal, as good tool support for the kinds of DSML construction activities we advocate would appear to be essential for real adoption. Pursuing further optimisations of the sort described both in Chapter 5, as well as those previously presented by Højsgaard [61] would seem a promising avenue for the continuation of this work.

An immediately promising possibility is attempting to construe the modular construction mechanisms proposed in Chapter 6 in the categorical "Bigraphs reloaded" setting of Miculan & Peressotti [73]. This is likely to yield further insights into the theoretical limits of this approach, provide stronger

connections to vertical refinement, and better characterise the sufficient and necessary conditions for well-formed compositions.

Finally, attempting to reconcile the work on calculational bigraphs [36] with a generalised notion of activity would seem to be valuable. There may be promise in the approach of using a variant of the bigraphical sorting logic to define predicates on contexts, such that language designers and modellers can have finer-grained control over the conditions under which a reaction may occur. Such an approach would need to relax the requirement that all well-formed formulae be deconstructable (this condition would seem less important for this task), and would need to augment the logic with constructs for reasoning about holes and regions, at least.

In all, bigraphical reactive systems still appear one of the most credible approaches to managing the complexity of modelling tasks for an increasingly computational world, and we hope that this work continues in earnest.

Bibliography

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] G. Agha. A model of concurrent computation in distributed systems, 1986.

[3] G. Agha. *An overview of actor languages*, volume 21. ACM, 1986.

[4] G. Agha, I.A. Mason, S.F. Smith, and C.L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, 1997.

[5] RJ Allan. Computing grand challenges. Technical Report DL-TR-2010-008, Science and Technology Facilities Council, 2010.

[6] Jacob Andersen and Claus Brabrand. Syntactic language extension via an algebra of languages and transformations. In *Proc. 9th Workshop on Language Descriptions, Tools and Applications, LDTA '09*, March 2009.

[7] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent programming in ERLANG*, volume 2. Prentice Hall, 1996.

[8] S. Awodey. *Category theory*, volume 49. Oxford University Press, USA, 2006.

[9] G. Bacci and D. Grohmann. On the Decidability of Bigraphical Sorting. In *CALCO Young Researchers Workshop CALCO-jnr 2009*, page 1, 2009.

[10] G. Bacci, D. Grohmann, and M. Miculan. DBtk: a toolkit for directed bigraphs. *Algebra and Coalgebra in Computer Science*, 2009.

[11] H.G. Baker Jr. List processing in real time on a serial computer. *Communications of the ACM*, 21(4):280–294, 1978.

[12] H.P. Barendregt. *The lambda calculus: Its syntax and semantics*, volume 103. North Holland, 1984.

[13] C. Bayol, B. Soulas, D. Borrione, F. Corno, and P. Prinetto. A process algebra interpretation of a verification oriented overlanguage of vhdl. In *Proceedings of the conference on European design automation*, pages 506–511. IEEE Computer Society Press, 1994.

[14] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Logic In Computer Science, 2009. LICS'09. 24th Annual IEEE Symposium on*, pages 39–48. IEEE, 2009.

[15] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Siméon. XML path language (XPath) 2.0. *W3C recommendation*, 23, 2007.

[16] JA Bergstra and J.W. Klop. Algebra of communicating processes. *CWI Monograph series*, 3:89–138, 1986.

[17] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical computer science*, 96(1):217–248, 1992.

[18] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. springer, 2004.

[19] P. Bidinger and A. Compagnoni. Pict correctness revisited. *Formal Methods for Open Object-Based Distributed Systems*, pages 206–220, 2007.

[20] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical models of context-aware systems. In *Foundations of Software Science and Computation Structures*, pages 187–201. Springer, 2006.

[21] L. Birkedal, S. Debois, and T. Hildebrandt. Sortings for reactive systems. *CONCUR 2006–Concurrency Theory*, pages 248–262, 2006.

[22] L. Birkedal, S. Debois, and T. Hildebrandt. On the construction of sorted reactive systems. In *CONCUR 2008*, pages 218–232. Springer, 2008.

[23] T. Bolusset and F. Oquendo. Formal refinement of software architectures based on rewriting logic. In *International Workshop on Refinement of Critical Systems: Methods, Tools and Experience, Grenoble*, 2002.

[24] G. Boudol. Asynchrony and the π-calculus. *Rapports de recherche-INRIA*, 1992.

[25] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM (JACM)*, 31(3):560–599, 1984.

[26] M. Bundgaard and V. Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of the 8th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 1–12. ACM, 2006.

[27] M. Calder and M. Sevegnani. Process algebra for event-driven runtime verification: a case study of wireless network management. In *Integrated Formal Methods*, pages 21–23. Springer, 2012.

[28] L. Cardelli and A. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures*, pages 140–155. Springer, 1998.

[29] P.P.S. Chen. The entity-relationship modeltoward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[30] G. Conforti, D. Macedonio, and V. Sassone. Spatial logics for bigraphs. *Automata, Languages and Programming*, 2005.

[31] B. Cuenca Grau and O. Kutz. Modular ontology languages revisited. In *Proc. of the Workshop on Semantic Web for Collaborative Knowledge Acquisition*, 2007.

[32] T.C. Damgaard and L. Birkedal. Axiomatixing binding bigraphs. Technical report, Citeseer, 2005.

[33] T.C. Damgaard, V. Danos, and J. Krivine. A language for the cell. Technical Report TR-2008-116, IT University of Copenhagen, December 2008.

[34] 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.

[35] S Debois. *Sortings & Bigraphs*. PhD thesis, IT University of Copenhagen, 2008.

[36] S. Debois. Computation in the informatic jungle. Technical report, Technical Report 147, IT University of Copenhagen, December, 2011.

[37] B. Delaware, B.C.S. Oliveira, and T. Schrijvers. Meta-Theory a la Carte. *POPL 2013*, 2013. (to appear).

[38] E.W. Dijkstra. Cooperating sequential processes. Technical Report EWD-123, 1965.

[39] E. Elsborg, T. Hildebrandt, and D. Sangiorgi. Type Systems for Bigraphs. In Christos Kaklamanis and Flemming Nielson, editors, *Proceedings of the 4th International Symposium on Trustworthy Global Computing (TGC 2008)*, volume 5474 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 2009.

[40] J. Epstein, A.P. Black, and S. Peyton-Jones. Towards haskell in the cloud. In *ACM SIGPLAN Notices*, volume 46, pages 118–129. ACM, 2011.

[41] M. Eysholdt and H. Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309. ACM, 2010.

[42] A. Faithfull, G. Perrone, and T.T. Hildebrandt. Big Red: A Development Environment for Bigraphs. In *Graph Computation Models 2012 (GCM 2012)*, 2012.

[43] P. Gardner. From process calculi to process frameworks. *CONCUR 2000 (Concurrency Theory)*, pages 69–88, 2000.

[44] P. Gardner, C. Laneve, and L. Wischik. The fusion machine (extended abstract). In L. Brim, P. Jancar, and M. Kretinsky, editors, *CONCUR 2002*, volume 2421 of *Lecture Notes in Computer Science*, pages 418–433. Springer-Verlag, 2002.

[45] S. Gay, V. Vasconcelos, and A. Ravara. Session types for inter-process communication. 2003.

[46] A.J. Glenstrup, T.C. Damgaard, L. Birkedal, and E. Højsgaard. An implementation of bigraph matching. 2007.

[47] P. Godefroid. Partial-order methods for the verification of concurrent systems. *Lecture notes in computer science*, 1996.

[48] M. Goldsmith and S. Creese. Refinement-friendly bigraphs and spy-graphs. In *2010 8th IEEE International Conference on Software Engineering and Formal Methods*, pages 203–207. IEEE, 2010.

[49] D. Grohmann and M. Miculan. Directed bigraphs. *Electronic Notes in Theoretical Computer Science*, 173, 2007.

[50] D. Grohmann and M. Miculan. Reactive systems over directed bigraphs. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *Lecture Notes in Computer Science*, pages 380–394. Springer-Verlag, 2007.

[51] M.K. Gupta. *Akka Essentials*. Packt, 2012.

[52] P. Haller and M. Odersky. Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 410(2):202–220, 2009.

[53] M. Hennessy and C. Stirling. The power of the future perfect in program logics. *Information and Control*, pages 23–52, 1985.

[54] M. Henson, J. Dooley, L. Whittington, and A. Al Malaise al Ghamdi. FollowMe: A Bigraphical Approach. In *Workshop Proceedings of the 8th International Conference on Intelligent Environments*, page 434. IOS Press, 2012.

[55] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[56] T. Hildebrandt, H. Niss, and M. 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, January 2006.

[57] T.T. Hildebrandt. Categorical models for fairness and a fully abstract presheaf semantics of sccs with finite delay. In *CTCS'99*, LNCS, 1999.

[58] T.T. Hildebrandt. *Categorical Models for Concurrency: Independence, Fairness and Dataflow*. PhD thesis, BRICS, Aarhus University, 2000.

[59] T.T. Hildebrandt and G. Perrone. BMC & BAM: A Declaratively-Sorted Meta-Calculus & Abstract Machine, 2012. Draft technical report. Appears in this thesis as Chapter 3.

[60] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[61] E. Højsgaard. *Bigraphical Languages and their Simulation*. PhD thesis, IT University of Copenhagen, 2012.

[62] E. Højsgaard and J. Krivine. Towards scalable simulation of stochastic bigraphs. Technical Report TR-2011-148, IT University of Copenhagen, 2011.

[63] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *ACM SIGPLAN NOTICES*, 43(1):273, 2008.

[64] O.H. Jensen. Mobile processes in bigraphs. Available at `http://www.cl.cam.ac.uk/~rm135/Jensen-monograph.pdf`, October 2006.

[65] O.H. Jensen and R. Milner. Bigraphs and transitions. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '03)*, New York, NY, USA, 2003. ACM.

[66] O.H. Jensen and R. Milner. Bigraphs and mobile processes (revised). *Technical report, University of Cambridge Computer Laboratory*, 2004.

[67] H. Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[68] J. Krivine, R. Milner, and A. Troina. Stochastic bigraphs. *Electronic Notes in Theoretical Computer Science*, 218:73–96, 2008.

[69] E.A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[70] J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. *CONCUR 2000Concurrency Theory*, pages 243–258, 2000.

[71] J. Leifer and R. Milner. Transition systems, link graphs and petri nets. *Journal of Mathematical Structures in Computer Science*, 16(6):989–1047, 2006.

[72] KL Man, M. Boubekeur, and MP Schellekens. Process algebraic approach to systemverilog. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 86–89. IEEE, 2007.

[73] M. Miculan and M. Peressotti. Bigraphs reloaded: An abstract presentation of bigraphs in presheaf categories. To appear, 2012.

[74] R. Milner. *A calculus of communicating systems*. Springer-Verlag, 1980.

[75] R. Milner. Action calculi, or syntactic action structures. *Mathematical Foundations of Computer Science 1993*, pages 105–121, 1993.

[76] R. Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.

[77] R. Milner. Bigraphical reactive systems. *CONCUR 2001*, pages 16–35, 2001.

[78] R. Milner. Bigraphs for Petri nets. *Lectures on Concurrency and Petri Nets*, pages 161–191, 2004.

[79] R. Milner. Axioms for bigraphical structure. *Journal of Mathematical Structures in Computer Science*, 15(6):1005–1032, 2005.

[80] R. Milner. Pure bigraphs. *University of Cambridge, Tech. Rep. UCAM-CL-TR-614*, 2005.

[81] R. Milner. Pure bigraphs: Structure and dynamics. *Information and Computation*, 204(1):60–122, January 2006.

[82] R. Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). In Roberto Amadio and Iain Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency (EXPRESS 2006)*, volume 175 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2007.

[83] R. Milner. *The space and motion of communicating agents*. Cambridge University Press, 2009.

[84] R. Milner. The tower of informatic models. *From semantics to Computer Science; Essays in Memory of Gilles Kahn*, 2009.

[85] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes part 1. Technical report, ECS-LFCS-89-85, 1989.

[86] F. Montesi and M. Carbone. Programming services with correlation sets. *Service-Oriented Computing*, pages 125–141, 2011.

[87] Z. Navabi. *VHDL: Analysis and modeling of digital systems*. McGraw-Hill, Inc., 1997.

[88] S. Ó Conchúir. Kind bigraphs. In Anthony Seda, Menouer Boubekeur, Ted Hurley, Micheal Mac an Airchinnigh, Michel Schellekens, and Glenn Strong, editors, *Proceedings of the Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT 2006)*, volume 225 of *Electronic Notes in Theoretical Computer Science*, pages 361–377. Elsevier, 2009.

[89] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, pages 176–185. IEEE, 1998.

[90] E. Pereira, C. Kirsch, and R. Sengupta. Biagents-a bigraphical agent model for structure-aware computation. 2012.

[91] G. Perrone, S. Debois, and T.T. Hildebrandt. Bigraphical refinement. In *Proceedings of the 15th International Workshop on Refinement (REFINE 2011)*, pages 20–36, 2011.

[92] G. Perrone, S. Debois, and T.T. Hildebrandt. A Model Checker for Bigraphs. In *ACM Symposium on Applied Computing 2012 – Software Verification and Testing Track*. ACM, 2012.

[93] C.A. Petri. *Petri. Kommunikation mit Automaten*. PhD thesis, University of Bonn, West Germany, 1962.

[94] S. Reeves and D. Streader. General refinement, part one: interfaces, determinism and special refinement. *Electronic Notes in Theoretical Computer Science*, 214:277–307, 2008.

[95] S. Reeves and D. Streader. General refinement, part two: flexible refinement. *Electronic Notes in Theoretical Computer Science*, 214:309–329, 2008.

[96] J. Rumbaugh, I. Jacobson, and G. Booch. The unified modeling language reference manual. *JOURNAL OF OBJECT TECHNOLOGY*, 3(10).

[97] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.

[98] A. Schmitt and J.B. Stefani. The kell calculus: A family of higher-order distributed process calculi. In *Global Computing*, pages 146–178. Springer, 2005.

[99] P. Sewell. From rewrite rules to bisimulation congruences. *CONCUR'98 Concurrency Theory*, pages 269–284, 1998.

[100] D.E. Thomas and P.R. Moorby. *The Verilog® Hardware Description Language*, volume 2. Springer, 2002.

[101] W. Van der Aalst, A. ter Hofstede, and M. Weske. Business process management: A survey. *Business Process Management*, pages 1019–1019, 2003.

[102] W. Wang, G. Perrone, and T. Hildebrandt. Petri nets in bigraphs revisited. In *24th Nordic Workshop on Programming Theory*, 2012.

[103] M. Weiser. Hot topics-ubiquitous computing. *Computer*, 26(10):71–72, 1993.

[104] N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.

[105] M. Zhang, L. Shi, L. Zhu, Y. Wang, L. Feng, and G. Pu. A Bigraphical Model of WSBPEL. In *Second Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE'08)*, pages 117–120. IEEE Computer Society, 2008.

Glossary

Glossary

hypergraph a graph structure in which every edge is connected to zero or more other graph objects.

inner width a finite ordinal indicating the number of holes (or sites) in the place graph, which for a bigraph $B : \langle m, X \rangle \to \langle n, Y \rangle$ is indicated by m.

outer width a finite ordinal indicating the number of regions (or roots) of the place graph, which for a bigraph $B : \langle m, X \rangle \to \langle n, Y \rangle$ is indicated by n.

prime a bigraph or place graph with exactly one top-level region (i.e., with outer width 1).

wide a bigraph or place graph with two or more top-level regions, i.e., with outer width ≥ 2.

Index