Typing Linear and Non-Linear Higher-Order Mobile Embedded Resources with Local Names

Mikkel Bundgaard
Thomas Hildebrandt
Jens Chr. Godskesen

IT University Technical Report Series TR-2007-97

ISSN 1600–6100 May 2007

ISSN 1600–6100

ISBN 978-87-7949-155-7

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web www.itu.dk

Typing Linear and Non-Linear Higher-Order Mobile Embedded Resources with Local Names

Mikkel Bundgaard Thomas Hildebrandt
Jens Chr. Godskesen
IT University of Copenhagen*
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark
{mikkelbu, hilde, jcg}@itu.dk

Abstract

We provide the first process calculus combining (affine) linear and non-linear higher-order mobile processes, nested locations, and local names. We do so by extending the type and effect system of Homer, a calculus of non-linear Higher-Order Mobile Embedded Resources, with a distinction between affine linear and non-linear locations (akin to reference types) and uses of variables (as in the linear lambda calculus). The type system guarantees that linear resources are neither copied nor embedded in non-linear resources during computation. We introduce composite reference types to guarantee that no path ever points to a linear location nested within a non-linear location. We extend the LTS semantics and adapt Howe's method to the typed setting, providing a bisimulation congruence. We apply the bisimulation to prove that scope extension across linear location boundaries is sound. Finally, we use the calculus to model an e-cash Smart Card system, the security of which depends on the interplay between (linear) mobile hardware, embedded (non-linear) mobile processes, and local names.

1 Introduction

The interplay between linearity and non-linearity has been studied thoroughly in variations of Intuitionistic Linear Logic [Bar96] and the corresponding denotational models, term models, and substructural type systems. The models and type systems have been used to describe and reason about co-existing linear and non-linear resources in functional programming, e.g. for memory management and references to system resources [TW99], in recent languages for quantum computing with classical control [SV06], and for controlling the use of names (and thus mobility) in the π-calculus [KPT99].

Following the seminal work on Mobile Ambients [CG00], several process calculi, variations of Mobile Ambients, the Seal calculus [CVZN05], and the Homer calculus [HGB04], have been proposed that combine (1) mobile processes, (2) nested explicit locations and (3) local names. These models are motivated by scenarios in global ubiquitous computing: Mobile processes are employed to represent both mobile *computing devices* (e.g. laptops, PDAs, and smart cards), as well as mobile *computations* (e.g. software agents and migrating processes). Nested explicit locations represent administrative domains, physical boundaries of mobile devices, boundaries of software processes such as sand-boxes and locations of data in memory. Finally, local names are used to represent, private keys and scope of references to locations in memory.

*This research is supported by the Danish Research Agency grant no: 274-06-0415 (Computer Supported Mobile Adaptive Business Processes - CosmoBiz) and grant no: 272-05-0258 (Mobile Security).

In the present paper, we initiate the study of linearity and non-linearity in the context of these three key features. Our motivations are two-fold: Firstly, we claim that mobile computing devices are intrinsically linear. A hardware device cannot easily be copied, and the security of a system may depend on this fact, for instance for smart cards. Mobile computations in software are intrinsically non-linear (software must usually be explicitly protected against copying, e.g. by enclosing it in a tamper-proof hardware device). We justify this claim in Section 8, by giving an example of an e-cash system, the security of which depends on the non-copyability of hardware devices; dually the copyability of software processes constitute an important security threat. The example indicates that a realistic model encompassing both mobile computing as well as mobile computations should allow for both linear and non-linear mobile processes.

Secondly, our previous work on providing semantics for the Higher-Order Mobile Embedded Resources (Homer) calculus [HGB04, GH05] and the related work on the Seal calculus [CVZN05] have shown that the combination of mobile communicating processes, explicit locations, and local names reveals a crucial difference between linear and non-linear mobile processes, which in itself justifies a study of the combination of linearity and non-linearity in this setting. When processes can communicate names across location boundaries, one needs a strategy for scope extension. If mobile processes are linear (or affine linear) as in Mobile Ambients, scope extension across location boundaries commutes with mobility, as expressed by the structural congruence rule

$$m[(n)p] \equiv (n)m[p] \text{ for names } n \neq m \ .$$

This is *not* the case if mobile processes are non-linear: Cloning the location m in the left hand process of the rule yields the process $m[(n)p] \parallel m[(n)p]$ which in general will be different from the process $(n)(m[p] \parallel m[p])$ resulting from cloning the location m in the right hand process.

Therefor in Homer and Seal, the scope of names across location boundaries is instead extended by *need* at the time of the communication. However, this choice makes it possible for any name n to construct a context $C_n[\ ]$ that exploits the dynamic scope extension and cloning to test, if the name n is free in a mobile process (see e.g. [HGB04]). Consequently, for two resources to be related in a congruence, they must at all times contain the same set of free names. Since names in process calculi normally get discarded during computation, this implies that any congruence would distinguish too many processes. In [GH05] we proved that a coarser and more interesting bisimulation congruence can be obtained by typing mobile embedded resources with an explicit effect type, identifying the set of names *allocated* by the resource (reminiscent of a type and effect system [ANN99]).

Inspired by affine intuitionistic linear logic, we extend in the present paper the type and effect system for Homer given in [GH05] to distinguish between affine linear and non-linear uses of variables (as in the linear lambda calculus) as well as typing the names of locations (akin to reference types), thereby restricting the content of locations to be either linear or non-linear. By making non-linear a subtype of affine linear and allowing non-linear locations to be embedded in linear locations, we ensure that non-linear resources can be used as affine linear resources. On the other hand, the type system guarantees that linear resources are never copied nor embedded in non-linear resources. We introduce composite reference types guaranteeing that linear locations within non-linear locations are never referenced.

The type system has consequences for the treatment of infinite behaviours. In most (untyped) higher-order calculi (HOπ [San92], λ-calculus [Bar84], CHOCS [Tho93], Homer [HGB04]) one can encode infinite behaviour by process passing (and process duplication). Constructors such as the Y-combinator, replication, or general recursion is then taken not as primitives, but rather as derived constructions. However, the encoding of recursion in Homer [HGB04] depends on the ability to copy resources. Thus, we can only encode recursion (and hence replication) of non-linear resources. Since replication does make sense for linear resources, allowing the availability of an arbitrary number of the same resource, we introduce this as a primitive constructor in the calculus.

Finally, we prove that Howe's method, used in [HGB04] for proving that the weak and strong

labelled bisimulations given for Homer are congruences, extends to this richer setting and thus provides a technique for contextual reasoning about linear and non-linear mobile embedded resources. Furthermore, we prove that the strong labelled bisimulation is sound and complete characterisation of barbed bisimulation congruence, again extending the results of [GH05]. We employ the strong bisimulation congruence to show that the scope of a local name can indeed be extended freely across affine linear locations.

To recap, the main contributions of this paper are thus:

- a type system for copyable and non-copyable resources extending the prior type/effect system for Homer and introducing composite reference types,
- introduction of replication as a primitive to allow replicated hardware,
- extension of Howe's method to prove the labelled bisimulations to be a congruence in this setting,
- proof that the strong labelled bisimulation is sound and complete characterisation of barbed bisimulation congruence,
- use of the strong bisimulation congruence to show that the scope of a local name can be extended freely across affine linear locations,
- a concrete example depending on all of the key features of the calculus.

Structure of the paper The structure of the paper is as follows. In Section 2 we present the Homer calculus, and in Section 3 and 4 we give it reaction and labelled transition semantics. The type system for Homer is presented in Section 5. In Section 6 we provide Homer with a weak and strong behavioural congruence, barbed bisimulation congruences, and in Section 7 we prove that input-early delay context bisimulations are congruences and sound characterisations of the barbed bisimulation congruences. In Section 7 we also prove that scope extension across linear location boundaries is safe. We provide the Smart Card example in Section 8, and we conclude and propose directions for future research in Section 9.

Related Work: We found no prior studies of linear and non-linear mobile processes combined with nested locations and local names. The Mobile Ambients calculus does combine affine linear mobile processes (the ambients) and non-linear messages (values), but replication of ambients does not allow one to model non-linear mobile computations, i.e. duplication of ambients after they have been moved or the contained processes have been partly executed. Thus, while the Mobile Ambients calculus is very suitable for modelling mobile computing (i.e. non-copyable devices), it does not support a direct representation of copyable mobile computations. The more recent higher-order process calculi, such as the Seal calculus [CVZN05] and Homer [HGB04], have explicitly introduced copyable mobile resources in the context of nested locations. But by assuming that *all* resources are copyable these calculi in turn become unrealistic as models of non-copyable mobile computing devices.

The Homer calculus extends Plain CHOCS, but shares ideas with recent calculi for mobility such as the Seal calculus [CVZN05], the M-calculus [SS03] (and its recent successor the Kell calculus [SS04]), in particular the ability to represent copyable (non-linear), objectively mobile anonymous resources in nested named locations. Type systems have been introduced for the M-calculus (and the Kell calculus [BS03]) which ensure unity of location names (used for deterministic routing). A type system for Seal calculus is presented in [CVZN05], the type system both type active processes and locations, thus enabling one to declare the type of processes that can enter and exit a location.

The composite address paths in Homer are in some respects similar to the composite channel names found in the π-calculus with polyadic synchronisation [CM03]. In [Car05] a type system

for polyadic synchronisation is given, based on Milner's type system for the polyadic π-calculus. Composite channel names which are typed with the type of the first (or last) element in the channel is also suggested, but the idea is not pursued.

Linear types have been studied in great detail in the π-calculus [KPT99, Kob02] by Kobayashi et al. Recently Berger, Honda, and Yoshida [BHY01, BHY05] have investigated the connection between sequential functional computation and typed π-calculus. For a higher-order π-calculus Yoshida and Hennessy [YH04, Yos04] have examined a type system which captures the effect of mobile processes by typing each process with an interface which describes the resources which the process can access.

A first version of the type system for linear and non-linear resources was proposed in [GH04] for the Mobile Resources calculus [GHS02], the predecessor of Homer, but the theory was never developed. Homer was originally presented in [HGB04], obtaining persistent allocation of names by an explicit process constructor, together with an adaptation of Howe's method to prove that late contextual bisimulation is a sound characterisation of barbed bisimulation congruence. In [GH05] the explicit name allocation constructor was replaced by explicit type annotations and the technical results were extended to prove that input-early strong congruence is both a sound and complete characterisation of barbed bisimulation congruence. Homer has also been examined in the setting of bigraphs [BH06] and its expressivity have been studied in an encoding of the π-calculus [BHG06].

2 Homer

In this section we provide the syntax of Homer. The only difference from [GH05] is that we have extended the syntax with replication.

We assume an infinite set of *names* $\mathcal{N}$ ranged over by m and n, and let $\tilde{n}$ range over finite sets of names. We let δ range over non-empty finite sequences of names, referred to as *paths* and let $\overline{\delta}$ denote *co-paths*. Paths and co-paths are used to reference arbitrarily deeply nested resources. We let φ range over δ and $\overline{\delta}$ and define $\overline{\overline{\delta}} = \delta$. We assume an infinite set of *process variables* $\mathcal{V}$ ranged over by x and y. The sets $\mathbf{p}$ of *process expressions* ranged over by p, q, and r, $\mathbf{a}$ of *abstractions* ranged over by a, and $\mathbf{c}$ of *concretions* ranged over by c are defined by the grammar:

$$\begin{array}{llll} p ::= \mathbf{0} \mid x \mid \varphi e \mid p \parallel p' \mid (n)p \mid !p \ , & & e ::= a \mid b \ , \\ a ::= (x)p \ , & b ::= \langle p' : \tilde{n} \rangle p \ , & c ::= b \mid (n)c \ , \end{array}$$

where b ranges over unrestricted concretions. We let $\mathbf{t}$, ranged over by t, denote the set $\mathbf{p} \cup \mathbf{a} \cup \mathbf{c}$. Whenever e denotes an abstraction we let $\overline{e}$ denote a concretion, and vice versa. The process p' in the concretion (referred to as the resource) is annotated by a finite set of names. As explained in the introduction, the type system presented in Section 5 guarantees that this set contains all names appearing free in the process p'.

Homer is defined as a simple extension of the higher-order calculus Plain CHOCS [Tho93] to allow for *active* processes at named locations denoted by prefixes of the form $n\langle p' : \tilde{n} \rangle p$ and a corresponding prefix denoted by $\overline{n}(x)q$ for moving the process at the location named n and substituting it in for the variable x in q. When the active process is moved, the location disappears (as in Seal) and the residual process is activated. The two prefixes complement the usual prefixes for *passive* process passing in CHOCS denoted by $\overline{n}\langle p' : \tilde{n} \rangle p$ and $n(x)q$. By active and passive we mean that the process p' in the prefix $n\langle p' : \tilde{n} \rangle p$ may perform internal reactions as well as interactions with processes outside the location, whereas the process p' in $\overline{n}\langle p' : \tilde{n} \rangle p$, as in CHOCS, can neither react nor interact with other processes. Interactions with embedded resources are obtained by the use of name *paths*, allowing a process to pass another process to (or to move) an arbitrarily deeply nested active embedded resource. For instance, we have the reactions (ignoring the type annotations)

$$n\langle m(x)q \rangle \parallel \overline{nm}\langle p \rangle p' \searrow n\langle q[{}^{p}/_{x}] \rangle \parallel p' \tag{1}$$

and

$$n\langle m\langle p\rangle p'\rangle \parallel \overline{nm}(x)q \searrow n\langle p'\rangle \parallel q[^{p}/_{x}] \ , \tag{2}$$

where nm is the name path consisting of the name n followed by the name m.

As usual, (x) bind the variable x and (n) bind the name n. We define the notions of free and bound names ($fn(t)$ and $bn(t)$) and variables of t ($fv(t)$ and $bv(t)$) as standard with the sole exception that $fn(\langle p' : \tilde{n}\rangle p) = \tilde{n} \cup fn(p)$, i.e. the annotation determines the free names of a resource. We will call a process without free variables *closed*, and let $\mathbf{t}_c$ and $\mathbf{p}_c$ denote the classes of closed terms and processes, respectively. We will throughout the paper consider terms up to α-equivalence, written $p \equiv_\alpha q$, and we will write $\mathbf{t}_{/\alpha}$ and $\mathbf{p}_{/\alpha}$ for the set of α-equivalence classes of terms and processes, respectively. We will also extend this notion to the sets of closed processes and terms.

We use standard shorthands and often elide $\mathbf{0}$ in a process, e.g. writing $\langle p : \tilde{n}\rangle$ instead of $\langle p : \tilde{n}\rangle\mathbf{0}$. For a set of names $\tilde{n} = \{n_1, \ldots, n_k\}$ we will write $(\tilde{n})t$ for $(n_1)\cdots(n_k)t$. We will also write n for the singleton set $\{n\}$ and when convenient let δ and $\overline{\delta}$ denote the set of names in the path. For any two sets s and s' we will write ss' for the union of s and s' under the assumption that $s \cap s' = \emptyset$.

We define the application between an abstraction and a concretion. The application depends on the type annotation of a resource. Since type annotations can "be wrong" in the untyped calculus the application can update type annotations with a "wrong" annotation. However in the typed setting, presented in the following sections application behave as intended.

Definition 1 (application and substitution)**.** Given a concretion $c = (\tilde{m})\langle p : \tilde{n}\rangle p'$ and an abstraction $a = (x)p''$ we define their application as follows whenever $\tilde{m} \cap fn(p'') = \emptyset$

$$c \cdot a = (\tilde{m})(p' \parallel p''[^{p:\tilde{n}}/_{x}]) \quad \text{and} \quad a \cdot c = (\tilde{m})(p''[^{p:\tilde{n}}/_{x}] \parallel p')$$

where $p''[^{p:\tilde{n}}/_{x}]$ is defined inductively in the structure of p''.

$$\begin{array}{lcll}
\mathbf{0}[^{p:\tilde{n}}/_{x}] & = & \mathbf{0} & \\
x[^{p:\tilde{n}}/_{x}] & = & p & \\
x'[^{p:\tilde{n}}/_{x}] & = & x' & \text{if } x \neq x' \\
(q \parallel q')[^{p:\tilde{n}}/_{x}] & = & q[^{p:\tilde{n}}/_{x}] \parallel q'[^{p:\tilde{n}}/_{x}] & \\
((n)q)[^{p:\tilde{n}}/_{x}] & = & (n)(q[^{p:\tilde{n}}/_{x}]) & \text{if } n \notin \tilde{n} \\
(!q)[^{p:\tilde{n}}/_{x}] & = & !(q[^{p:\tilde{n}}/_{x}]) & \\
(\varphi e)[^{p:\tilde{n}}/_{x}] & = & \varphi e[^{p:\tilde{n}}/_{x}] & \\
(\langle q : \tilde{m}'\rangle q')[^{p:\tilde{n}}/_{x}] & = & \langle q[^{p:\tilde{n}}/_{x}] : \tilde{m}' \cup \tilde{n}\rangle q'[^{p:\tilde{n}}/_{x}] & \text{if } x \in fv(q) \\
(\langle q : \tilde{m}'\rangle q')[^{p:\tilde{n}}/_{x}] & = & \langle q : \tilde{m}'\rangle q'[^{p:\tilde{n}}/_{x}] & \text{if } x \notin fv(q) \\
((x')q)[^{p:\tilde{n}}/_{x}] & = & (x')(q[^{p:\tilde{n}}/_{x}]) & \text{if } x \neq x'
\end{array}$$

To allow for more succinct presentations of the reaction and labelled transition semantics we close concretions and abstractions under some of the process operators, i.e. restriction, parallel composition, and resources. Hence, whenever $c = (\tilde{n})\langle p_1 : \tilde{n}_1\rangle p$ and assuming $\tilde{n} \cap (fn(p') \cup n \cup \delta) = \emptyset$ (using α-conversion if needed) we let $c \parallel p'$ denote $(\tilde{n})\langle p_1 : \tilde{n}_1\rangle(p \parallel p')$, we let $\delta\langle c : \tilde{n}'\rangle p'$ denote $(\tilde{n})\langle p_1 : \tilde{n}_1\rangle\delta\langle p : \tilde{n}'\tilde{n}\rangle p'$, and we let $(n)c$ denote $(n\tilde{n})\langle p_1 : \tilde{n}_1\rangle p$, if $n \in \tilde{n}_1$ and otherwise it denotes $(\tilde{n})\langle p_1 : \tilde{n}_1\rangle(n)p$. Similarly, whenever $a = (x)p$ and assuming $x \notin fv(p')$ (using α-conversion if needed) we let $a \parallel p'$ denote $(x)(p \parallel p')$, $(n)a$ for $(x)(n)p$, and we let $\delta\langle a : \tilde{n}\rangle p'$ denote $(x)\delta\langle p : \tilde{n}\rangle p'$. These shorthands are applied to "lift" the concretion/abstraction inside the path context in the reaction semantics, and in the rules (*nesting*), (*rest*), (*par*), (*par′*), and (*repl1*) in the labelled transition semantics.

3 Reaction Semantics

As customary, we provide Homer with a reaction semantics in the chemical abstract machine style [BB90] using structural congruence, evaluation and path contexts, and reaction rules.

Table 1: Structural congruence

$p \parallel \mathbf{0} \equiv p$	$(p \parallel p') \parallel p'' \equiv p \parallel (p' \parallel p'')$	$p \parallel q \equiv q \parallel p$
$(n)p \parallel q \equiv (n)(p \parallel q),\ \text{if } n \notin fn(q)$		$p \equiv q,\ \text{if } p \equiv_\alpha q$
$(n)(m)p \equiv (m)(n)p$	$(n)p \equiv p,\ \text{if } n \notin fn(p)$	$!p \equiv p \parallel !p$

Table 2: Reaction rule

(*react*)	$\overline{\gamma\delta}e \parallel \mathcal{D}_\gamma(\delta\overline{e}) \searrow e \cdot \mathcal{D}_\gamma(\overline{e}), \quad bn(\mathcal{D}_\gamma) \cap \delta = \emptyset$

We define *structural congruence*, as expected, as the least congruence relation on $\mathbf{p}_{/\alpha}$ that satisfies the rules in Table 1. The rules express the usual commutative monoid requirements of parallel composition and the inactive process, the rules of scope of local names, and the definition of replication. The rules does *not* contain the scope extension mentioned in the introduction, $m\langle(n)p : \tilde{n}\rangle q \equiv (n)m\langle p : \tilde{n}n\rangle q$ for $n \notin m \cup fn(q)$.

We define a (general) *context* as a term in which we have replaced one terminal $\mathbf{0}$ with the symbol $(-)$, denoting a *hole* in which we can place a process. We will let $\mathcal{C}$ range over (general) contexts. As Homer permits reactions arbitrarily deep in the location hierarchy and also permits reactions between a process and an arbitrarily deeply nested sub-resource, we define the concepts of evaluation and path contexts to succinctly state the reaction semantics. Both kinds of contexts are a restricted kind of a general context. An *evaluation context* $\mathcal{E}$ is a context with no free variables and whose hole is not guarded by a prefix, nor does it occur as the object of a send prefix. We define evaluation contexts by the following grammar

$$\mathcal{E} ::= (-) \mid \mathcal{E} \parallel p \mid (n)\mathcal{E} \mid \delta\langle\mathcal{E} : \tilde{n}\rangle p \ , \text{ for } p \in \mathbf{p}_c \ .$$

Note that the evaluation context $\delta\langle\mathcal{E} : \tilde{n}\rangle p$ enables internal reactions of active resources. To permit reaction between a process and an arbitrarily deeply nested sub-resource, we define a restricted set of evaluation contexts, i.e. a family of *path contexts* $\mathcal{D}_\gamma$ indexed by a path address $\gamma \in \mathcal{N}^*$ which indicates the path under which the context's hole is found. A path context $\mathcal{D}_\gamma$ is defined inductively by $\mathcal{D}_\epsilon = (-)$ (where ϵ denotes an empty sequence of names) and

$$\mathcal{D}_{\delta\gamma} ::= \delta\langle(\tilde{m})(\mathcal{D}_\gamma \parallel p) : \tilde{n}'\rangle p' \ , \text{ for } p, p' \in \mathbf{p}_c \ ,$$

such that $\gamma \cap \tilde{m} = \emptyset$ and none of the names in $\tilde{m}$ are already bound in $\mathcal{D}_\gamma$, i.e. we assume all names binding the hole of a context are unique. We let $\mathcal{D}_\gamma$ range over path contexts. If the indexed path in $\mathcal{D}_\gamma$ is not important we write $\mathcal{D}$ instead.

Finally, we define $\searrow$ as the least binary relation on $\mathbf{p}_{c/\alpha}$ satisfying the (parametrised) reaction rule in Table 2 and closed under structural congruence and evaluation contexts. By the latter we mean that $p \searrow p'$ implies $\mathcal{E}(p) \searrow \mathcal{E}(p')$ for all evaluation contexts $\mathcal{E}$. The reaction rule captures succinctly both kind of reactions mentioned in the introduction. The sending of a passive resource down into the location hierarchy $\mathcal{D}_\gamma$, where it is received by the abstraction $\overline{e}$ and substitute in for the process variable in the body of the abstraction. And the taking of a computing resource from the location hierarchy $\mathcal{D}_\gamma$ to the abstraction e, and again substituting it in for the process variable in the body of the abstraction.

4 Transition Semantics

In this section we provide Homer with a labelled transition semantics. As in the previous section, the only difference from [GH05] is that we have extended the semantics with rules for replication.

Table 3: Transition rules

$$(prefix)\ \frac{}{\varphi e \xrightarrow{\varphi} e} \qquad (sync)\ \frac{p \xrightarrow{\varphi} e \quad p' \xrightarrow{\overline{\varphi}} \bar{e}}{p \parallel p' \xrightarrow{\tau} e \cdot \bar{e}} \qquad (nesting)\ \frac{p \xrightarrow{\pi} t}{\delta\langle p : \tilde{n}\rangle p' \xrightarrow{\delta \cdot \pi} \delta\langle t : \tilde{n}\rangle p'}$$

$$(rest)\ \frac{p \xrightarrow{\pi} t}{(n)p \xrightarrow{\pi} (n)t},\ n \notin fn(\pi) \qquad (par)\ \frac{p \xrightarrow{\pi} t}{p \parallel p' \xrightarrow{\pi} t \parallel p'} \qquad (par')\ \frac{p' \xrightarrow{\pi} t}{p \parallel p' \xrightarrow{\pi} p \parallel t}$$

$$(repl1)\ \frac{p \xrightarrow{\pi} t}{!p \xrightarrow{\pi} t \parallel !p} \qquad (repl2)\ \frac{p \xrightarrow{\varphi} a \quad p \xrightarrow{\overline{\varphi}} c}{!p \xrightarrow{\tau} (a \cdot c) \parallel !p}$$

We let π range over the set Π of labels, formally defined as $\pi ::= \tau \mid \varphi$, (recall $\varphi ::= \delta \mid \overline{\delta}$). The set of free names in π, $fn(\pi)$, is $fn(\delta)$ whenever $\pi = \delta$ or $\pi = \bar{\delta}$ and $\emptyset$ otherwise. The rules in Table 3 then define a labelled transition system

$$(\mathbf{t}_{c/\alpha}, \longrightarrow \subseteq \mathbf{p}_{c/\alpha} \times \Pi \times \mathbf{t}_{c/\alpha})$$

for α-equivalence classes of closed processes.

The rules conservatively extend the rules for Plain CHOCS. Note that the rule (*sync*) covers the two different kind of interactions: the active and passive resource movement as described in the previous section, and that the rule (*nesting*) permits arbitrarily deeply nested active resources to be moved, receive resources, and perform internal computation steps. To allow these three kinds of actions we use an operation $\delta \cdot (\_)$ for extending location paths, defined by:

$$\delta \cdot \tau = \tau, \quad \delta \cdot \delta' = \delta\delta' \ .$$

Note that the operation is not defined for $\overline{\delta}$ since $\overline{\delta}$ is directed "downward" and thus not visible outside the resource. Since $\delta \cdot \tau = \tau$, the nesting rule implies that $\delta\langle p : \tilde{n}\rangle p' \xrightarrow{\tau} \delta\langle t : \tilde{n}\rangle p'$, if $p \xrightarrow{\tau} t$.

As an example of using the rules (and shorthands for concretions and abstractions) the reaction (1) in Section 2 can be derived from $m(x)q \xrightarrow{m} (x)q$, so $n\langle m(x)q : \tilde{n}\rangle \xrightarrow{nm} (x)n\langle q : \tilde{n}\rangle$. Combining with $\overline{nm}\langle p : \tilde{n}'\rangle p' \xrightarrow{\overline{nm}} \langle p : \tilde{n}'\rangle p'$ we obtain

$$n\langle m(x)q : \tilde{n}\rangle \parallel \overline{nm}\langle p : \tilde{n}'\rangle p' \xrightarrow{\tau} (x)n\langle q : \tilde{n}\rangle \cdot \langle p : \tilde{n}'\rangle p' \ .$$

By Definition 1 we get $(x)n\langle q : \tilde{n}\rangle \cdot \langle p : \tilde{n}'\rangle p' = n\langle q[^{p:\tilde{n}'}/_x] : \tilde{n} \cup \tilde{n}'\rangle \parallel p'$ (if $x \in fv(q)$). Similarly for the reduction (2) we have that $m\langle p : \tilde{n}'\rangle p' \xrightarrow{m} \langle p : \tilde{n}'\rangle p'$, so $n\langle m\langle p : \tilde{n}'\rangle p' : \tilde{n}''\rangle \xrightarrow{nm} \langle p : \tilde{n}'\rangle n\langle p' : \tilde{n}''\rangle$. So combining with $\overline{nm}(x)q \xrightarrow{\overline{nm}} (x)q$ we obtain

$$n\langle m\langle p : \tilde{n}'\rangle p' : \tilde{n}''\rangle \parallel \overline{nm}(x)q \xrightarrow{\tau} \langle p : \tilde{n}'\rangle n\langle p' : \tilde{n}''\rangle \cdot (x)q \ ,$$

which by Definition 1 is the process $n\langle p' : \tilde{n}''\rangle \parallel q[^{p:\tilde{n}'}/_x]$.

In order to prove that the reaction relation and the labelled transition relation coincides (when restricted to τ-transitions), we need that transitions are preserved by structural congruence.

Proposition 1. *If $p \equiv q$ and $q \xrightarrow{\pi} t'$ then for some t'' we have $p \xrightarrow{\pi} t''$ and $t'' \equiv t'$.*

We can then prove that our two semantics coincides when restricted to τ-transitions and reactions, respectively.

Theorem 2 (semantics coincide)**.** *$p \searrow p'$ if and only if $p \xrightarrow{\tau} \equiv p'$.*

Proof.
($\Rightarrow$) From $p \searrow p'$ we can prove that this implies $p \equiv \mathcal{E}(\overline{\gamma\delta}e \parallel \mathcal{D}_\gamma(\delta\overline{e}))$ and $p' \equiv \mathcal{E}(e \cdot \mathcal{D}_\gamma(\overline{e}))$ and $bn(\mathcal{D}_\gamma) \cap \delta = \emptyset$ from the definition of the reaction relation. So from Proposition 63 we know that there exists some p'' such that $p \xrightarrow{\tau} p''$ and $p'' \equiv \mathcal{E}(e \cdot \mathcal{D}_\gamma(\overline{e}))$. So since $p'' \equiv p'$ we have that $p \xrightarrow{\tau} \equiv p'$ as desired.
($\Leftarrow$) We prove that $p \xrightarrow{\tau} p'$ implies $p \searrow p'$ by using Proposition 56 which tells us that $p \equiv \mathcal{E}(\overline{\gamma\delta}e \parallel \mathcal{D}_\gamma(\delta\overline{e}))$ and $p' \equiv \mathcal{E}(e \cdot \mathcal{D}_\gamma(\overline{e}))$ and $bn(\mathcal{D}_\gamma) \cap \delta = \emptyset$ and the result follows from the definition of the reaction relation. □

5 The Type System

We are now ready to present the extension of the type and effect system given for Homer in [GH05] to allow a distinction between affine linear and non-linear (unrestricted) resources.

We will assume a set $\mathcal{S} = \{\mathtt{aff}, \mathtt{un}\}$, of *affine* and *unrestricted* (i.e. non-linear) *sorts*, and let S range over sorts. Furthermore, we will assume the subtyping relation $\leq$ on $\mathcal{S}$ such that $\mathtt{un} < \mathtt{aff}$, which corresponds with our intuition that an unrestricted process can be used instead of an affine process. Or concretely, as exemplified by the model of a Smart Card system in Section 8, that software can be embedded in, and used as, hardware, but not the other way around.

Process types consist of two parts written as $S\ \tilde{n}$. The first part, the sort S, records if the process is affine linear or non-linear. The second part, $\tilde{n}$, was introduced by the type system in [GH05] and can be regarded as an effect that captures the names used or allocated by the process. The type system guarantees that this set is a superset of the free names in the process. Besides process types, we also define concretion and abstraction types. The *concretion type* $\langle S \rangle S'\ \tilde{n}'$ types a concretion $(\tilde{m}')\langle p : \tilde{m} \rangle p'$ in which the transferred process p has sort S and where the entire concretion has the sort S' and effect $\tilde{n}'$. The *abstraction type* $S \twoheadrightarrow S'\ \tilde{n}$ types an abstraction $(x)p$ that itself has sort S' and effect $\tilde{n}$ and accepts a process of sort S. Note that since we do not pass abstractions as values, we do not need to distinguish between the effect of abstractions and processes by annotating the effect of the abstraction on the arrow as it is standard in effect systems [ANN99]. We will only consider abstraction and concretion types where $S \leq S'$, and this requirement is enforced by the typing rules.

Definition 2 (types)**.** We define three kinds of types, *process types* T_p, *concretion types* T_c, and *abstraction types* T_a, by the following grammar

$$T ::= T_p \mid T_c \mid T_a$$
$$T_p ::= S\ \tilde{n}\ , \quad T_c ::= \langle S \rangle T_p\ , \quad T_a ::= S \twoheadrightarrow T_p$$

We define parallel composition, $\parallel$, of two process types with the same sort, $S\ \tilde{n}$ and $S\ \tilde{n}'$, as $S\ \tilde{n} \cup \tilde{n}'$. For $n \notin \tilde{n}$ we write $(S\ \tilde{n})n$ for the process type $S\ \tilde{n}n$ and $(\langle S \rangle S'\ \tilde{n})n$ for the concretion type $\langle S \rangle S'\ \tilde{n}n$, i.e. the *disjoint* extension of the name set with a name n.

We write $T \cup \tilde{n}''$ for the (not necessarily disjoint) name extension of the type T defined inductively as follows

$$\begin{array}{lcl} (S\ \tilde{n}) \cup \tilde{n}'' & = & S\ \tilde{n} \cup \tilde{n}'' \\ (\langle S \rangle T_p) \cup \tilde{n}'' & = & \langle S \rangle T_p \cup \tilde{n}'' \\ (S \twoheadrightarrow T_p) \cup \tilde{n}'' & = & S \twoheadrightarrow T_p \cup \tilde{n}'' \end{array}$$

The above name extension is used to propagate effects when resources are moved. We extend the subtyping relation $\leq$ to types, so $S\ \tilde{n} \leq S'\ \tilde{n}$ if $S \leq S'$, $\langle S \rangle T_p \leq \langle S \rangle T_p'$ if $T_p \leq T_p'$, and $S \twoheadrightarrow T_p \leq S \twoheadrightarrow T_p'$ if $T_p \leq T_p'$. We define type environments as expected as a finite mapping from names and variables to their sort.

Definition 3 (type environment)**.** A type environment Γ is a finite partial function $\Gamma : \mathcal{N} \uplus \mathcal{V} \rightharpoonup \mathcal{S}$ from names and variables to sorts. We will write $dom_n(\Gamma)$ and $dom_v(\Gamma)$ for respectively names

Table 4: Combination of type environments

$$(\textit{empty})\ \frac{}{\emptyset \odot \emptyset = \emptyset} \qquad (\textit{var-un})\ \frac{\Gamma \odot \Gamma' = \Gamma''}{(\Gamma, x : \mathtt{un}) \odot (\Gamma', x : \mathtt{un}) = \Gamma'', x : \mathtt{un}}$$

$$(\textit{var-l})\ \frac{\Gamma \odot \Gamma' = \Gamma''}{(\Gamma, x : S) \odot \Gamma' = \Gamma'', x : S} \qquad (\textit{var-r})\ \frac{\Gamma \odot \Gamma' = \Gamma''}{\Gamma \odot (\Gamma', x : S) = \Gamma'', x : S}$$

$$(\textit{name})\ \frac{\Gamma \odot \Gamma' = \Gamma''}{(\Gamma, n : S) \odot (\Gamma', n : S) = \Gamma'', n : S}$$

$$(\textit{name-l})\ \frac{\Gamma \odot \Gamma' = \Gamma''}{(\Gamma, n : S) \odot \Gamma' = \Gamma'', n : S} \qquad (\textit{name-r})\ \frac{\Gamma \odot \Gamma' = \Gamma''}{\Gamma \odot (\Gamma', n : S) = \Gamma'', n : S}$$

Table 5: Typing address paths

$$\frac{}{\Gamma, n : S \vdash n : S\ \mathtt{Ref}\ S} \qquad \frac{\Gamma, n : S \vdash \delta : S''\ \mathtt{Ref}\ S'}{\Gamma, n : S \vdash \delta n : S''\ \mathtt{Ref}\ S}(S \leq S') \qquad \frac{\Gamma \vdash \delta : S\ \mathtt{Ref}\ S'}{\Gamma \vdash \overline{\delta} : S\ \mathtt{Ref}\ S'}$$

and variables in the domain of Γ, and let $dom(\Gamma) = dom_n(\Gamma) \cup dom_v(\Gamma)$. If $n \notin dom_n(\Gamma)$ we write $\Gamma, n : S$ for the extension of Γ with the mapping from n to S, and similarly for variables. We will write $\Gamma \setminus n$ for the removal of the mapping of n from Γ, and similarly for variables, and we will extend this notion to sets as well. We will let Δ range over environments with no variable mappings.

To present our typing rules we need to be able to combine two environments in a way that, as usual for linear type systems, constrain the presence of linearly used variables. We define the combination Γ'' of two type environments Γ and Γ', denoted $\Gamma \odot \Gamma' = \Gamma''$, using the rules in Table 4. The rules enforce that for $\Gamma \odot \Gamma' = \Gamma''$, any name occurring in Γ'' can either occur in Γ, Γ', or in both (if it has the same sort). The same is the case for unrestricted variables, whereas the same affine linear variable cannot in occur in both Γ and Γ'. This underlines, that our type system is concerned with linear use of *processes* and not of names, as in [KPT99].

We also need typing of address paths: $\Gamma \vdash \varphi : S\ \mathtt{Ref}\ S'$, as defined by the rules in Table 5. The type $S\ \mathtt{Ref}\ S'$ is read as *a reference via S to S'*. The rules ensure that the sorts of the names in an address path typed $S\ \mathtt{Ref}\ S'$ form a non-strictly descending chain, ensuring that an affine resource cannot be referenced inside an unrestricted resource, and that the first name of the address path has sort S and the last name of the path has sort S'. For instance, letting $\Gamma = m : \mathtt{aff}, n : \mathtt{un}$, we can derive $\Gamma \vdash mm : \mathtt{aff}\ \mathtt{Ref}\ \mathtt{aff}$ and $\Gamma \vdash mmn : \mathtt{aff}\ \mathtt{Ref}\ \mathtt{un}$, but we cannot derive neither $\Gamma \vdash nm : \mathtt{un}\ \mathtt{Ref}\ \mathtt{aff}$ nor $\Gamma \vdash mnm : \mathtt{aff}\ \mathtt{Ref}\ \mathtt{aff}$.

Proposition 3. *Writing φ_i for the i'th element of the path φ, and $length(\varphi)$ for the length of the path φ, we have $\Gamma \vdash \varphi : S\ \mathtt{Ref}\ S'$ if and only if $dom_n(\Gamma) \supseteq \varphi$, $\Gamma(\varphi_1) = S$, $\Gamma(\varphi_{length(\varphi)}) = S'$, and $\forall i, j$ with $1 \leq i \leq j \leq length(\varphi)$ implies $\Gamma(\varphi_j) \leq \Gamma(\varphi_i)$.*

We define the typing of processes, abstractions, concretions, and contexts using the rules in Table 6. The type system conservatively generalises the prior type (effect) system for Homer [GH05], which we can obtain by taking $\mathcal{S}$ to be a singleton set, making it possible to delete all references to sorts from abstraction and concretion types, and completely remove side-conditions and environments. We only explain some of the rules, the rest should be self-explanatory. The (*conc*) rule allows us to type a basic concretion, if the extruded process has a sub-sort of the residual process. We can type an abstraction with (*abs*), if we can type the body of the abstraction under an extended environment where x is given a sub-sort of the sort of the abstraction. The rule

Table 6: Typing rules for affine linear and non-linear Homer

$$(\textit{variable})\ \frac{}{\Gamma, \tilde{n} : \tilde{S}, x : S \vdash x : S\ \tilde{n}} \qquad (\textit{inactive})\ \frac{}{\Gamma, \tilde{n} : \tilde{S} \vdash \mathbf{0} : \mathtt{un}\ \tilde{n}}$$

$$(\textit{parallel})\ \frac{\Gamma \vdash p : S\ \tilde{n} \quad \Gamma' \vdash p' : S\ \tilde{n}'}{\Gamma \odot \Gamma' \vdash p \parallel p' : S\ \tilde{n} \cup \tilde{n}'} \qquad (\textit{rest})\ \frac{\Gamma, n : S \vdash p : T_p n}{\Gamma \vdash (n)p : T_p}$$

$$(\textit{rest-conc})\ \frac{\Gamma, n : S \vdash (\tilde{m})\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c n}{\Gamma \vdash (\tilde{m}n)\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c} \qquad (\textit{subsump})\ \frac{\Gamma \vdash p : T_p}{\Gamma \vdash p : T'_p}(T_p \leq T'_p)$$

$$(\textit{repl})\ \frac{\Gamma \vdash p : T_p}{\Gamma \vdash\ !p : T_p}(\forall x \in fv(p).\ \Gamma(x) = \mathtt{un}) \qquad (\textit{abs})\ \frac{\Gamma, x : S' \vdash p : S\ \tilde{n}}{\Gamma \vdash (x)p : S' \twoheadrightarrow S\ \tilde{n}}(S' \leq S)$$

$$(\textit{conc})\ \frac{\Gamma \vdash p : S\ \tilde{n} \quad \Gamma' \vdash p' : S'\ \tilde{n}'}{\Gamma \odot \Gamma' \vdash \langle p : \tilde{n}\rangle p' : \langle S\rangle S'\ \tilde{n} \cup \tilde{n}'}(S \leq S')$$

$$(\textit{pre-abs})\ \frac{\Gamma \vdash a : S' \twoheadrightarrow S\ \tilde{n} \quad \Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'}{\Gamma \vdash \varphi a : S\ \tilde{n} \cup \varphi}(S'' \leq S)$$

$$(\textit{pre-conc})\ \frac{\Gamma \vdash b : \langle S'\rangle S\ \tilde{n} \quad \Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'}{\Gamma \vdash \varphi b : S\ \tilde{n} \cup \varphi}(S'' \leq S)$$

$$(\textit{context})\ \frac{}{\Gamma, \Gamma' \vdash (-)_{(\Gamma, S\ \tilde{n})} : S\ \tilde{n}\tilde{n}'}(\tilde{n}' \subseteq dom_n(\Gamma') \text{ and } \tilde{n} \subseteq dom_n(\Gamma))$$

(*pre-abs*) allows us to form a process from an abstraction as long as the sort of the received process is the sort that the abstraction expects from the address path. The (*subsump*) rule corresponds to the usual subsumption rule in type systems with subtyping. The side-condition in the rule (*repl*) ensures us that all variables in Γ that occur free in p are unrestricted, however Γ may contain affine variables which do not occur free in p. The reason why we allow this is to ensure that typing is invariant under structural congruence. If we did not allow Γ to contain affine variables at all, then the structural equivalence $!p \equiv !p \parallel \mathbf{0}$ would introduce problems, as a typing of the process on the right-hand side could contain affine variables in the typing environment, whereas on the left-hand side it could not.

In Section 3 we defined a hole as the symbol $(-)$ for which we could substitute in an arbitrary process. In the typed setting we need more control of which kind of processes that can be placed in a hole, we therefore augment holes to carry annotation to constrain the set of processes. We write $(-)_{(\Gamma, T_p)}$ for a hole in which we can place a process p which can be typed as $\Gamma \vdash p : T_p$. We will use the annotation of the hole to index contexts, i.e. we will let $\mathcal{C}_{(\Gamma, T_p)}$ range over general contexts, where the hole is annotated with (Γ, T_p). We extend this notation to evaluation contexts and path contexts, written $\mathcal{E}_{(\Gamma, T_p)}$ and $\mathcal{D}_{(\Gamma, T_p), \gamma}$ respectively, as before we will leave out the indexed path in a path context, if it is not important. As a technical detail (due to the side-condition of the rule (*repl*)), we define the free variables of a hole as $fv((-)_{(\Gamma, T_p)}) = dom_v(\Gamma)$.

The typing rules for processes employ the path types to make sure that the resource provider and receiver agrees on what is being communicated, combining ideas of reference types, which constrain the types of the referenced resources, and types for process calculi, which constrain the types of objects being communicated on channels. Thus for a typed address path $\Gamma \vdash \varphi : S\ \mathtt{Ref}\ S'$ both the resource provider and receiver agree on that the communicated process has sort S' (this constraint can be weakened by subsumption for the provider's part, and narrowing for the receiver's

part). The sort S of the outermost name in the address path is used in the side-conditions of the rules (*pre-conc*) and (*pre-abs*) to ensure that any process using the path has a super-sort of S, which means that affine names can never occur in paths inside unrestricted resources. For instance, if n is affine and m is unrestricted then in the process $nm\langle p : \tilde{n}'\rangle q : S\ \tilde{n}$ the resource p is unrestricted, but the typing rules enforce that $S = \mathtt{aff}$, meaning that the entire process is typed as affine. This is a restricted use of linear resources, but it fits well with the scenario of linear, mobile computing devices containing non-linear mobile computations: A mobile computing device can never be contained in, or manipulated by, a software process. We leave for future work to study more relaxed type systems for linear resources that may be contained in, or manipulated by, non-linear resources. This makes sense if one, as in [TW99], consider linear *software* resources which (in a type-safe environment) should be guaranteed not to be copied (i.e. for reasons of memory management).

We prove the standard properties about the type system: strengthening of unused names and variables, weakening of the typing environment, invariance of typing under structural congruence, that composition of a well-typed process with a well-typed context gives rise to a well-typed process, a substitution lemma, properties of well-typed terms, and subject reduction for the reaction relation and the labelled transition relation. We prove the following results by induction on the typing derivation.

Proposition 4 (weakening)**.** *Let l range over names and variables. If $\Gamma \vdash t : T$ and $l \notin dom(\Gamma)$ then $\Gamma, l : S \vdash t : T$.*

We can strengthen the environment with names and variables that does not occur free in the process. Note that the strengthening of names also effects the type of the process.

Proposition 5 (strengthening, names)**.** *Assume $n \notin fn(t)$ and $\Gamma, n : S \vdash t : T$ then $\Gamma \vdash t : T \setminus n$.*

Proposition 6 (strengthening, variables)**.** *Assume $x \notin fv(t)$ and $\Gamma, x : S \vdash t : T$ then $\Gamma \vdash t : T$.*

We can also add names from the type environment to the effect of a term.

Proposition 7. *If $\Gamma \vdash t : T$ and $n : S' \in \Gamma$ then $\Gamma \vdash t : T \cup n$.*

As expected in a type system with subtyping we have narrowing of variables.

Proposition 8 (narrowing of variables)**.** *If $\Gamma, x : S \vdash f : T$ and $S' \leq S$ then $\Gamma, x : S' \vdash f : T$.*

Note that we in general cannot have narrowing (or widening) of names, as this can make address paths ill-typed, i.e. the ordering can be destroyed, if we allow to change the type of a name. As expected the typing is invariant under structural congruence.

Proposition 9 (structural congruence and typing)**.** *If $t \equiv t'$ then $\Gamma \vdash t : T$ if and only if $\Gamma \vdash t' : T$.*

Proof. By induction on the derivation of $t \equiv t$. □

The subsumption rule between process types induces subsumption between concretion and abstraction types as well.

Proposition 10. *If $\Gamma \vdash t : T$ and $T \leq T'$ then $\Gamma \vdash t : T'$.*

The combination of a well-typed process and a well-typed context preserves well-typedness.

Proposition 11 (general context application)**.** *If $\Gamma' \vdash \mathcal{C}_{(\Gamma, T_p)} : S'\ \tilde{n}'$ and $\Gamma \vdash p : T_p$ then $\Gamma' \vdash \mathcal{C}_{(\Gamma, T_p)}(p) : S'\ \tilde{n}'$.*

As corollaries of Proposition 11 we have that the application of a well-typed context and a well-typed process (of the appropriate type) results in a well-typed process.

Corollary 12. *If* $\Gamma' \vdash \mathcal{E}_{(\Gamma,T_p)} : S'\ \tilde{n}'$ *and* $\Gamma \vdash p : T_p$ *then* $\Gamma' \vdash \mathcal{E}_{(\Gamma,T_p)}(p) : S'\ \tilde{n}'$.

Corollary 13. *If* $\Gamma' \vdash \mathcal{D}_{(\Gamma,T_p)} : S'\ \tilde{n}'$ *and* $\Gamma \vdash p : T_p$ *then* $\Gamma' \vdash \mathcal{D}_{(\Gamma,T_p)}(p) : S'\ \tilde{n}'$.

Lemma 14 (substitution lemma)**.** *Let* $\Delta \vdash p : S\ \tilde{n}$ *be a closed process and let* $\Gamma', x : S \vdash t' : T'$ *be a term with* $\Delta \odot \Gamma'$ *defined then* $\Delta \odot \Gamma' \vdash t'[^{p:\tilde{n}}/_x] : T''$*, where* $T'' = T' \cup \tilde{n}$ *if* $x \in fv(t')$ *and* $T'' = T'$ *otherwise.*

Proof. Proof by induction on the derivation of $\Gamma', x : S \vdash t' : T'$. We will only consider the case for (*conc*), since the rule has two premises and since it (possibly) involves updating the type annotation of a resource. The other cases are easier.

- (*conc*) So we know that $\Gamma', x : S = \Gamma'' \odot \Gamma'''$, $t' = \langle p'' : \tilde{n}'' \rangle p'$, and $T' = \langle S'' \rangle S'\ \tilde{n}'' \cup \tilde{n}'$. By inversion we know $\Gamma'' \vdash p'' : S''\ \tilde{n}''$ and $\Gamma''' \vdash p' : S'\ \tilde{n}'$ and $S'' \leq S'$. There are several sub-cases depending on where x occurs (if x occurs at all) and the sort of x. We consider only the sub-cases where x is affine linear. We can prove the cases where x is non-linear in same manner. Below we use the following results: weakening (Proposition 4); strengthening of variables (Proposition 6); a corollary of Lemma 16(1), i.e. for $\Gamma \vdash t : T$ if $x \notin dom_v(\Gamma)$ then $x \notin fv(t)$; and if $x \notin fv(t)$ then $t[^{p:\tilde{n}}/_x] = t$ which we prove by structural induction.

  Since x is affine linear it can occur in either Γ'' or Γ''', but not in both.

  - $x \in dom_v(\Gamma'')$. We have two sub-cases depending on whether x occurs free in p''
    - $x \notin fv(p'')$. So by strengthening we obtain $\Gamma'' \setminus x \vdash p'' : S''\ \tilde{n}''$ and since $x \notin dom_v(\Gamma''')$ we have $\Gamma''' \vdash p'[^{p:\tilde{n}}/_x] : S'\ \tilde{n}'$. So by the rule (*conc*) we obtain $(\Gamma'' \odot \Gamma''') \setminus x \vdash \langle p'' : \tilde{n}'' \rangle p'[^{p:\tilde{n}}/_x] : \langle S'' \rangle S'\ \tilde{n}'' \cup \tilde{n}'$ and the result follows by weakening with Δ.
    - $x \in fv(p'')$. So by the induction hypothesis $(\Delta \odot \Gamma'') \setminus x \vdash p''[^{p:\tilde{n}}/_x] : S''\ \tilde{n}'' \cup \tilde{n}$, and we have $\Gamma''' \vdash p'[^{p:\tilde{n}}/_x] : S'\ \tilde{n}'$ as above. So again by (*conc*) we obtain $\Delta \odot \Gamma' \vdash \langle p''[^{p:\tilde{n}}/_x] : \tilde{n}'' \cup \tilde{n} \rangle p'[^{p:\tilde{n}}/_x] : \langle S'' \rangle S'\ \tilde{n}'' \cup \tilde{n} \cup \tilde{n}'$.
  - $x \in dom_v(\Gamma''')$. So by the induction hypothesis $(\Delta \odot \Gamma''') \setminus x \vdash p'[^{p:\tilde{n}}/_x] : T''$ and $T'' = S'\ \tilde{n}' \cup \tilde{n}$ if $x \in fv(p')$ and $T'' = S'\ \tilde{n}'$, otherwise. In both cases the result follows as above.

□

Proposition 15 (well-typed application)**.** *If* $\Gamma \vdash a : S'' \twoheadrightarrow S'\ \tilde{n}''$ *is an closed abstraction and* $\Gamma' \vdash c : \langle S'' \rangle S'\ \tilde{n}'$ *is a closed concretion with* $c \cdot a$ *and* $\Gamma \odot \Gamma'$ *defined then* $\Gamma \odot \Gamma' \vdash c \cdot a : S'\ \tilde{n}'' \cup \tilde{n}'$ *is a closed process.*

Proof. Follows from Lemma 14 and the definition of application and substitution (Definition 1). □

Our type system ensures us that well-typed terms satisfies several properties, below we state the main properties. The properties imply that the annotation of resources contains the free names of the resource, that affine terms cannot be contained in unrestricted terms, and that affine terms cannot be duplicated.

Lemma 16 (properties of well-typed terms)**.** *Writing* $n(T)$ *for the names and* $s(T)$ *for the sort of the type* T*, defined as* $\tilde{n}$ *and* S*, if* T *is of the form* $S\ \tilde{n}$*,* $S' \twoheadrightarrow S\ \tilde{n}$*, or* $\langle S' \rangle S\ \tilde{n}$*. If* $\Gamma \vdash t : T$ *then*

1. $fn(t) \subseteq n(T) \subseteq dom_n(\Gamma)$ *and* $fv(t) \subseteq dom_v(\Gamma)$.
2. *If* $x : \texttt{aff} \in \Gamma$ *then* x *occurs free at most once in* t.
3. *If* $x : \texttt{aff} \in \Gamma$ *and* $x \in fv(t)$ *then* $s(T) = \texttt{aff}$.

4. If $s(T) = \mathtt{un}$ *then for every sub-derivation* $\Gamma' \vdash t' : T'$ *we have* $s(T') = \mathtt{un}$.

Proof (sketch). We can prove the first three items by induction on the typing derivation. For the last item note that for every rule in Table 6, if the conclusion of the rule has sort S then each of the premises have a sort S' with $S' \leq S$. □

Theorem 17 (subject reduction, reaction relation)**.** *If* $\Gamma \vdash p : S\ \tilde{n}$ *and* $p \searrow p'$ *then* $\Gamma \vdash p' : S\ \tilde{n}$.

Proof (sketch). By considering the reaction rule in Table 2 and using the previous results about placing a process in an evaluation context (Corollary 12), invariance under $\equiv$ (Proposition 9), and well-typed application (Proposition 15). □

Theorem 18 (subject reduction, labelled transition relation)**.** *Suppose* $\Gamma \vdash p : S\ \tilde{n}$ *and* $p \xrightarrow{\pi} t$ *then one of the following cases hold.*

- $\pi = \tau$, $t = p'$ *and* $\Gamma \vdash p' : S\ \tilde{n}$.
- $\pi = \varphi$, $t = a$ *and* $\Gamma \vdash a : S' \twoheadrightarrow S\ \tilde{n}$ *and* $\Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'$ *for some* S' *and* S'' *with* $S'' \leq S$.
- $\pi = \varphi$, $t = c$ *and* $\Gamma \vdash c : \langle S' \rangle S\ \tilde{n}$ *and* $\Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'$ *for some* S' *and* S'' *with* $S'' \leq S$.

Proof. Proof by induction on the derivation of $p \xrightarrow{\pi} t$ using the results of Appendix A to handle the typing of shorthands. Note that the rules in Table 6 are not completely syntax-directed, since we can apply the rule (*subsump*) to any process. However, this only introduces minor complications (for brevity we will not state the inversion lemma of the typing relation) We will only consider a few of the cases

- (*prefix*) Suppose $\Gamma \vdash p : S\ \tilde{n}$ and the transition $p \xrightarrow{\pi} t$ is derived from (*prefix*). So we know that $p = \varphi e$, $\pi = \varphi$, and $t = e$. We have two sub-cases depending on whether e is an abstraction or a concretion. We present only the first sub-case as they are similar.

  So we have $\Gamma \vdash \varphi a : S\ \tilde{n}$ for some a. By inversion we know that this can only be derived using (*pre-abs*) possibly followed by the rule (*subsump*). For generality we assume that (*subsump*) has been used. So we have $\Gamma \vdash \varphi a : S'''\ \tilde{n}$ for some S''' with $S''' \leq S$. By the rule (*pre-abs*) we have $\Gamma \vdash a : S' \twoheadrightarrow S'''\ \tilde{n}'$ for some S' ad $\tilde{n}'$ with $\tilde{n}' \cup \varphi = \tilde{n}$ and $\Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'$ and $S'' \leq S'''$. The desired result, $\Gamma \vdash a : S' \twoheadrightarrow S\ \tilde{n}$ follows from application of Proposition 7 and Proposition 10.

- (*par*) Assume $\Gamma \vdash p : S\ \tilde{n}$ and the transition $p \xrightarrow{\pi} t$ is derived using (*par*). We know $p = p'' \parallel p'$ and $t = t'' \parallel p'$ and that $p'' \xrightarrow{\pi} t''$. As above, for generality we assume that the rule (*subsump*) has been applied in the derivation. So we have $\Gamma \vdash p'' \parallel p' : S'''\ \tilde{n}$ and $S''' \leq S$. So by inversion we know $\Gamma_1 \vdash p'' : S'''\ \tilde{n}_1$ and $\Gamma_2 \vdash p' : S'''\ \tilde{n}_2$ and $\Gamma = \Gamma_1 \odot \Gamma_2$ and $\tilde{n} = \tilde{n}_1 \cup \tilde{n}_2$. We have three sub-cases to consider, and we will present two of them (when $\pi = \tau$ and t'' is a process p''', and when $\pi = \varphi$ and t'' is an abstraction a''').

  - ($\pi = \tau$) By induction hypothesis we know that $\Gamma_1 \vdash p''' : S'''\ \tilde{n}_1$ and the desired result, $\Gamma \vdash p''' \parallel p' : S\ \tilde{n}$, follows from (*parallel*) and (*subsump*).
  - ($\pi = \varphi$) By induction hypothesis we know that $\Gamma_1 \vdash a''' : S' \twoheadrightarrow S'''\ \tilde{n}_1$ and $\Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'$ with $S'' \leq S'''$. So by Proposition 43 we know that $\Gamma \vdash a''' \parallel p' : S' \twoheadrightarrow S'''\ \tilde{n}$, so by Proposition 10 the desired result follows and the conditions on sorts are satisfied.

□

5.1 A Type Checking Algorithm

In this section we examine a type checking algorithm for the declarative rules given in Table 6. The two main difficulties are to eliminate the subsumption rule (*subsump*) and to eliminate the non-deterministic partitioning of the type environment. We solve both problems by using standard methods: generalise the axioms in the typing system and by keeping track of which affine variables a term uses, following the approach of [Wal04].

However, before we can present the type checking algorithm we need to change the syntax of Homer slightly, by adding an explicit annotation on restrictions. The reason for this is that we cannot guess the sort of the restricted name from the restriction alone, we need to know in which contexts the name is used[1]. So with this change in syntax we need to change the two typing rules regarding restriction (we will subscript this new typing judgement with an A to resolve any ambiguity).

$$(\textit{rest})\ \frac{\Gamma, n : S \vdash_A p : T_p n}{\Gamma \vdash_A (n : S)p : T_p} \qquad (\textit{rest-conc})\ \frac{\Gamma, n : S \vdash_A (\tilde{m} : \tilde{S})\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c n}{\Gamma \vdash_A (n : S)(\tilde{m} : \tilde{S})\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c}$$

We define a relation between terms of the syntax without explicit annotations on restrictions and the syntax with explicit annotations on restrictions, and we write this relation as t `ann` t'. Two terms are related, t `ann` t', if t and t' are syntactical equivalent except for the explicit annotations in t'. If we can type a term t using the rules in Table 6 then there exists an explicitly annotated term t' related to t which can be typed with the new typing judgement.

Proposition 19. *$\Gamma \vdash t : T$ if and only if there exists a t' such that t `ann` t' and $\Gamma \vdash_A t' : T$.*

For the typing of address path we don't need to change anything as the rules already are syntax-directed and terminates.

Lemma 20. *The typing of address path given in Table 5 is both syntax-directed, deterministic, and terminates.*

Following the standard approach to remove the non-deterministic splitting of the type environment we define a new judgement $\Gamma \vdash_M t : T;\ \Gamma'$. Compared to the existing judgements we have added an additional component, the *output* type environment. Intuitively one can read the judgement as "under the type environment Γ we can type t as T (thereby consuming a portion of Γ), and Γ' is the unused portion of Γ". Since we only constrain the usage of affine variables, these are the only elements of Γ which can be consumed. So when type checking a rule with several premises, e.g. (*parallel*) we pass the entire type environment to the left branch to type check it, and the output type environment from this type checking is then used to type check the right branch.

To define the typing rules we need to calculate the subtraction of a mapping from a type environment, written $\Gamma - l : S$, formally defined as follows

$$\begin{array}{lcll} (\Gamma, l : S) - l : S & = & \Gamma & \\ \Gamma - l : S & = & \Gamma & l \notin \mathit{dom}(\Gamma)\ . \end{array}$$

The first clause is used for subtracting non-linear variables, unused affine variables, and names. The second clause is only for used affine variables, as they are the only bindings which can be consumed during type checking.

We can now define the typing of terms using the rules in Table 7. The type rules differ from the rules in Table 6 in several respects. As described in the beginning of this section, we have removed the subsumption rule and the non-determinism in partitioning the type environment. We have

[1] In a concrete implementation we could do without the annotations and instead use sort variables. The instantiation of these sort variables could then be chosen lazily when the name used. Constraint Handling Rules [Frü98] would be an obvious candidate to implement this approach.

Table 7: Algorithmic Typing rules for affine linear and non-linear Homer

$$(\textit{unrest})\ \frac{}{\Gamma, \tilde{n} : \tilde{S}, x : \mathtt{un} \vdash_M x : S\ \tilde{n};\ \Gamma, \tilde{n} : \tilde{S}, x : \mathtt{un}} \qquad (\textit{aff})\ \frac{}{\Gamma, \tilde{n} : \tilde{S}, x : \mathtt{aff} \vdash_M x : \mathtt{aff}\ \tilde{n};\ \Gamma, \tilde{n} : \tilde{S}}$$

$$(\textit{parallel})\ \frac{\Gamma \vdash_M p : S\ \tilde{n};\ \Gamma' \quad \Gamma' \vdash_M p' : S\ \tilde{n};\ \Gamma''}{\Gamma \vdash_M p \parallel p' : S\ \tilde{n};\ \Gamma''} \qquad (\textit{inactive})\ \frac{}{\Gamma, \tilde{n} : \tilde{S} \vdash_M \mathbf{0} : S\ \tilde{n};\ \Gamma, \tilde{n} : \tilde{S}}$$

$$(\textit{repl})\ \frac{\Gamma \vdash_M p : T_p;\ \Gamma'}{\Gamma \vdash_M\ !p : T_p;\ \Gamma'}(\forall x \in \mathit{fv}(p).\ \Gamma(x) = \mathtt{un}) \qquad (\textit{rest})\ \frac{\Gamma, n : S \vdash_M p : T_p n;\ \Gamma'}{\Gamma \vdash_M (n : S)p : T_p;\ \Gamma' - n : S}$$

$$(\textit{abs})\ \frac{\Gamma, x : S' \vdash_M p : S\ \tilde{n};\ \Gamma'}{\Gamma \vdash_M (x)p : S' \twoheadrightarrow S\ \tilde{n};\ \Gamma' - x : S'}(S' \leq S)$$

$$(\textit{conc})\ \frac{\Gamma \vdash_M p : S\ \tilde{n};\ \Gamma' \quad \Gamma' \vdash_M p' : S'\ \tilde{n}';\ \Gamma''}{\Gamma \vdash_M \langle p : \tilde{n}\rangle p' : \langle S\rangle S'\ \tilde{n}';\ \Gamma''}(S \leq S' \text{ and } \tilde{n} \subseteq \tilde{n}')$$

$$(\textit{pre-abs})\ \frac{\Gamma \vdash_M a : S' \twoheadrightarrow S\ \tilde{n}\varphi;\ \Gamma' \quad \Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'}{\Gamma \vdash_M \varphi a : S\ \tilde{n}\varphi;\ \Gamma'}(S'' \leq S)$$

$$(\textit{rest-conc})\ \frac{\Gamma, n : S \vdash_M (\tilde{m} : \tilde{S})\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c n;\ \Gamma'}{\Gamma \vdash_M (n : S)(\tilde{m} : \tilde{S})\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c;\ \Gamma' - n : S}$$

$$(\textit{pre-conc})\ \frac{\Gamma \vdash_M b : \langle S'\rangle S\ \tilde{n}\varphi;\ \Gamma' \quad \Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'}{\Gamma \vdash_M \varphi b : S\ \tilde{n}\varphi;\ \Gamma'}(S'' \leq S)$$

also removed the non-determinism in splitting up the effect in several of the rules. For instance in the rule (*parallel*) we type check each branch with the full effect $\tilde{n}$ instead of splitting up the effect in two (not necessarily disjoint) parts. We also type check the right-hand branch with the full effect in the rule (*conc*). Similar, in the two rules (*pre-abs*) and (*pre-conc*) we type check the abstraction and concretion with the full effect, and not with a possible smaller effect, as done in Table 6. These changes, however, will not affect the set of well-typed terms. Note that we in the rules (*pre-abs*) and (*pre-conc*) have chosen to type check the address paths with the initial type environment Γ, we could just as well have chosen to use the output type environment Γ', as the environments contain the same set of name bindings (see Lemma 22 below).

In order to match the subsumption rule in the declarative type system we must prove that we still can perform subsumption using the new type system.

Proposition 21. *If* $\Gamma \vdash_M t : T;\ \Gamma'$ *and* $T \leq T'$ *then* $\Gamma \vdash_M t : T';\ \Gamma'$.

Let $n(\Gamma)$ denote the name mappings, $v_{\mathtt{un}}(\Gamma)$ the mappings of unrestricted variables, and $v_{\mathtt{aff}}(\Gamma)$ the mappings of affine variable in Γ, then we have that for a well-typed term $\Gamma \vdash_M t : T;\ \Gamma'$ that Γ and Γ' coincide on the mappings of names and unrestricted variables, but that Γ can contain some mappings of affine variables which do not occur in Γ'.

Lemma 22. *If* $\Gamma \vdash_M t : T;\ \Gamma'$ *then* $n(\Gamma') = n(\Gamma)$ *and* $v_{\mathtt{un}}(\Gamma') = v_{\mathtt{un}}(\Gamma)$, *and* $v_{\mathtt{aff}}(\Gamma') \subseteq v_{\mathtt{aff}}(\Gamma)$.

Furthermore, we have that affine variables which appear free in the term do not occur in the output type environment.

Lemma 23. *Let* $\Gamma \vdash_M t : T;\ \Gamma'$ *and* $x : \mathtt{aff} \in \Gamma$. *If* $x \in \mathit{fv}(t)$ *then* $x : \mathtt{aff} \notin \Gamma'$.

We prove the same results for the algorithmic type system as for the declarative system: weakening and strengthening of names and variables, and the addition of names to the effect of the term. We prove all the following four results by induction on the typing derivation.

Proposition 24. *Let l range over names and variables. If $\Gamma \vdash_M t : T;\ \Gamma'$ and $l \notin dom(\Gamma)$ then $\Gamma, l : S \vdash_M t : T;\ \Gamma', l : S$.*

Proposition 25. *Assume $n \notin fn(t)$ and $\Gamma, n : S \vdash_M t : T;\ \Gamma', n : S$ then $\Gamma \vdash_M t : T \setminus n;\ \Gamma'$.*

Proposition 26. *Assume $x \notin fv(t)$ and $\Gamma, x : S \vdash_M t : T;\ \Gamma', x : S$ then $\Gamma \vdash_M t : T;\ \Gamma'$.*

Proposition 27. *If $\Gamma \vdash_M t : T;\ \Gamma'$ and $n : S' \in \Gamma$ then $\Gamma \vdash_M t : T \cup n;\ \Gamma'$.*

Using some of the previous results we prove our algorithmic typing relation sound and complete with respect to the declarative presentation in Table 6.

Theorem 28 (soundness and completeness)**.** *$\Gamma \vdash_A t : T$ if and only if $\Gamma \vdash_M t : T;\ \Gamma'$.*

Proof. In both directions we prove it by induction over the typing derivation, and using the results about weakening and strengthening of type environments, the addition of names to the effect, Proposition 21 to match the subsumption rule, and Lemma 22 relating the input and output type environment in the algorithmic type checking.
($\Rightarrow$) Proof by induction on the derivation of $\Gamma \vdash_A t : T$. We only present some of the cases

- (*parallel*) Assume that we have $\Gamma \odot \Gamma' \vdash_A p \parallel p' : S\ \tilde{n} \cup \tilde{n}'$, and it is derived from $\Gamma \vdash_A p : S\ \tilde{n}$ and $\Gamma' \vdash_A p' : S\ \tilde{n}'$, so by using the induction hypothesis twice we have

  $$\Gamma \vdash_M p : S\ \tilde{n};\ \Gamma'' \quad \text{and} \quad \Gamma' \vdash_M p' : S\ \tilde{n}';\ \Gamma''' \ .$$

  Let $\tilde{n}'' = \tilde{n} \cup \tilde{n}'$. We weaken the first judgement with the names and variables occurring in $\Gamma' \setminus \Gamma$ and add $\tilde{n}'$ to the effect, hence obtaining $\Gamma \odot \Gamma' \vdash_M p : S\ \tilde{n}'';\ \Gamma'' \odot \Gamma'$, and we weaken the second judgement with the names and variables occurring in $\Gamma'' \setminus \Gamma'$ and add $\tilde{n}$ to the effect, hence obtaining $\Gamma' \odot \Gamma'' \vdash_M p' : S\ \tilde{n}'';\ \Gamma''' \odot \Gamma''$, so we end up with $\Gamma \odot \Gamma' \vdash_M p \parallel p' : S\ \tilde{n}'';\ \Gamma''' \odot \Gamma''$ as desired.

- (*subsump*) We have that $\Gamma \vdash_A p : T'_p$ is derived from $\Gamma \vdash_A p : T_p$ and $T_p \leq T'_p$, so by the induction hypothesis we have $\Gamma \vdash_M p : T_p;\ \Gamma'$ and the result follows from Proposition 21 (algorithmic subsumption).

($\Leftarrow$) Proof by induction on the derivation of $\Gamma \vdash_M t : T;\ \Gamma'$. Again, we only present some of the cases

- (*parallel*) Assume that we have $\Gamma \vdash_M p \parallel p' : S\ \tilde{n};\ \Gamma''$, and it is derived from $\Gamma \vdash_M p : S\ \tilde{n};\ \Gamma'$ and $\Gamma' \vdash_M p' : S\ \tilde{n};\ \Gamma''$, so by using the induction hypothesis twice we have

  $$\Gamma \vdash_A p : S\ \tilde{n} \quad \text{and} \quad \Gamma' \vdash_A p' : S\ \tilde{n} \ .$$

  We cannot in general combine Γ and Γ' as both might contain the same affine variables. However, we can obtain Γ''' by strengthening of Γ, where we remove all affine variables which do not occur free in p, hence giving us $\Gamma''' \vdash_A p : S\ \tilde{n}$. The combination $\Gamma''' \odot \Gamma'$ is defined, since an affine variable x can only occur in Γ', if it occurs in Γ and if it does not occur free in p. So we obtain the desired result $\Gamma \vdash_A p \parallel p' : S\ \tilde{n}$, as $\Gamma''' \odot \Gamma' = \Gamma$.

- (*pre-abs*) Assume that we have $\Gamma \vdash_M \varphi a : S\ \tilde{n}\varphi;\ \Gamma'$ and that it is derived from $\Gamma \vdash_M a : S' \twoheadrightarrow S\ \tilde{n}\varphi;\ \Gamma'$ and $\Gamma \vdash \varphi : S''$ `Ref` S' and we have $S'' \leq S$. By the induction hypothesis we have $\Gamma \vdash_A a : S' \twoheadrightarrow S\ \tilde{n}\varphi$, so the result follows $\Gamma \vdash_A \varphi a : S\ \tilde{n}\varphi$.

□

6 Barbed Bisimulation Congruence

In this section we define strong and weak *barbed bisimulation congruences* based on the reaction semantics and observations. In the following section we define input-early delay context bisimulations which in the weak case is a sound characterisation, and in the strong case is both a sound and complete characterisation.

As in previous work [HGB04, GH05] we define weak and strong barbs as the observation of a resource at top level. Let $\searrow^*$ be the transitive and reflexive closure of $\searrow$.

Definition 4 (barbs)**.** We say that the process p has a *strong barb* n, written $p \downarrow n$ if $p \equiv (\tilde{n})(n\langle p' : \tilde{n}'\rangle p'' \parallel p''')$ and $n \notin \tilde{n}$. We say that the process p has a *weak barb* n, written $p \Downarrow n$ if there exists p' such that $p \searrow^* p'$ and $p' \downarrow n$.

We have previously in [HGB04] shown that this choice of barbs is robust. We say that a well-typed process $\Gamma \vdash p : T_p$ has a strong barb, if the underlying process p has a strong barb, and similarly for weak barbs. We have a correspondence between barbs and transitions.

Proposition 29. *$p \downarrow n$ if and only if $p \xrightarrow{n} c$ for some c.*

We define binary relations between typed terms with respect to the same environment and type, and we furthermore require that the relations are closed under extension of the effect, weakening of environment, and under subsumption of the type.

Definition 5. We define a binary relation $\mathcal{R}$ on $\mathbf{t}_{/\alpha}$ between typed terms $\Gamma \vdash t : T$ and $\Gamma \vdash t' : T$ of the same type and wrt. the same environment and write this as $\Gamma \vdash t \,\mathcal{R}\, t' : T$. We say that $\mathcal{R}$ is *well-typed* if $\Gamma \vdash t \,\mathcal{R}\, t' : T$ implies:

1. $\Gamma \vdash t \,\mathcal{R}\, t' : T \cup n$, if $n \in dom_n(\Gamma)$,
2. $\Gamma, \Gamma' \vdash t \,\mathcal{R}\, t' : T$,
3. $\Gamma \vdash t \,\mathcal{R}\, t' : T'$ for $T \leq T'$.

We will throughout the paper assume that all relations on typed terms are well-typed. We define the restriction of a binary relation $\mathcal{R}$ to closed processes by $\mathcal{R}_c = \mathcal{R} \cap \mathbf{p}_{c/\alpha} \times \mathbf{p}_{c/\alpha}$.

Definition 6. A *weak barbed simulation* is a well typed binary relation $\mathcal{R}$ on $\mathbf{p}_{c/\alpha}$ such that whenever $\Delta \vdash p \,\mathcal{R}\, q : S\ \tilde{n}$,

- if $p \downarrow n$, then $q \Downarrow n$
- if $p \searrow p'$, then there exists q' such that $q \searrow^* q'$ and $\Delta \vdash p' \,\mathcal{R}\, q' : S\ \tilde{n}$.

$\mathcal{R}$ is a *weak barbed bisimulation* if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are weak barbed simulations. *Weak barbed bisimulation congruence* $\approx_b$ is the largest congruence such that $(\approx_b)_c$ is a weak barbed bisimulation.

We define strong barbed simulation similar to above by replacing $\searrow^*$ with $\searrow$ and $q \Downarrow n$ with $q \downarrow n$, and define strong barbed bisimulation congruence $\sim_b$ accordingly.

Proposition 30. *$\approx_b$ and $\sim_b$ are equivalence relations.*

7 Bisimulation Congruence

In this section we provide Homer with weak and strong input-early (delay) context bisimulations. We prove that both weak and strong context bisimulations are sound characterisations of weak and strong barbed bisimulation, and furthermore that strong context bisimulation is a complete characterisation.

We define *delay* transitions as usual by $p \stackrel{\tau}{\Longrightarrow} p$ and $p \stackrel{\pi}{\Longrightarrow} p'$, if $p \stackrel{\tau}{\longrightarrow}^{*} \stackrel{\pi}{\longrightarrow} p'$. We extend a relation $\mathcal{R}$ on closed typed processes to typed concretions by $\Delta \vdash c \; \mathcal{R}^{\square} \; c' : \langle S'' \rangle S' \; \tilde{n}'$, if for all typed abstractions $\Delta' \vdash a : S'' \twoheadrightarrow S' \; \tilde{n}''$ with $\Delta \odot \Delta'$ defined we have $\Delta \odot \Delta' \vdash c \cdot a \; \mathcal{R} \; c' \cdot a : S' \; \tilde{n}'' \cup \tilde{n}'$. We then define typed input-early delay context (bi)simulation as follows. Subject reduction ensures transitions between well-typed terms.

Definition 7. An *input-early delay context simulation* is a well typed binary relation $\mathcal{R}$ on $\mathbf{p}_{c/\alpha}$ such that $\Delta \vdash p \; \mathcal{R} \; q : S \; \tilde{n}$ implies

- if $p \stackrel{\tau}{\longrightarrow} p'$ then $\exists q'.\ q \stackrel{\tau}{\Longrightarrow} q'$ such that $\Delta \vdash p' \; \mathcal{R} \; q' : S \; \tilde{n}$.
- if $p \stackrel{\overline{\delta}}{\longrightarrow} a$ and $\Delta \vdash \overline{\delta} : S'' \; \mathtt{Ref} \; S'$ then $\forall c \in \mathbf{c}_{c/\alpha}$ with $\Delta' \vdash c : \langle S' \rangle S \; \tilde{n}'$ and $\Delta \odot \Delta'$ defined, there $\exists a'.\ q \stackrel{\overline{\delta}}{\Longrightarrow} a'$ such that $\Delta \odot \Delta' \vdash c \cdot a \; \mathcal{R} \; c \cdot a' : S \; \tilde{n} \cup \tilde{n}'$.
- if $p \stackrel{\delta}{\longrightarrow} a$ and $\Delta \vdash \delta : S'' \; \mathtt{Ref} \; S'$ then $\forall c \in \mathbf{c}_{c/\alpha}$ with $\Delta' \vdash c : \langle S' \rangle S''' \; \tilde{n}''$ and $\forall \mathcal{D}_{(\Delta, S \; \tilde{n})}.\ \Delta'' \vdash \mathcal{D}_{(\Delta, S \; \tilde{n})} : S''' \; \tilde{n}'$ with $\Delta' \odot \Delta''$ defined there exists an $a'.\ q \stackrel{\delta}{\Longrightarrow} a'$ such that $\Delta' \odot \Delta'' \vdash c \cdot \mathcal{D}_{(\Delta, S \; \tilde{n})}(a) \; \mathcal{R} \; c \cdot \mathcal{D}_{(\Delta, S \; \tilde{n})}(a') : S''' \; \tilde{n}' \cup \tilde{n}''$.
- if $p \stackrel{\overline{\delta}}{\longrightarrow} c$ and $\Delta \vdash \overline{\delta} : S'' \; \mathtt{Ref} \; S'$ then $\exists c'.\ q \stackrel{\overline{\delta}}{\Longrightarrow} c'$ such that $\Delta \vdash c \; \mathcal{R}^{\square} \; c' : \langle S' \rangle S \; n$.
- if $p \stackrel{\delta}{\longrightarrow} c$ and $\Delta \vdash \delta : S'' \; \mathtt{Ref} \; S'$ then $\exists c'.\ q \stackrel{\delta}{\Longrightarrow} c'$ such that $\forall \mathcal{D}_{(\Delta, S \; \tilde{n})}.\ \Delta' \vdash \mathcal{D}_{(\Delta, S \; \tilde{n})} : S''' \; \tilde{n}'$ we have $\Delta' \vdash \mathcal{D}_{(\Delta, S \; \tilde{n})}(c) \; \mathcal{R}^{\square} \; \mathcal{D}_{(\Delta, S \; \tilde{n})}(c') : \langle S' \rangle S''' \; \tilde{n}'$.

$\mathcal{R}$ is an input-early delay context bisimulation if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are input-early delay context simulations. Let $\approx$ denote the largest input-early delay context bisimulation. We define input-early strong context simulation by replacing $\Longrightarrow$ with $\longrightarrow$ and let $\sim$ denote the largest input-early strong context bisimulation.

Note that we employ the shorthands introduced in Section 2 when placing abstractions and concretions in path contexts in the third and fifth condition above. For $\Gamma \vdash p : S \; \tilde{n}$ and $\Gamma', x : S \vdash q : T_p$ with $\Gamma \odot \Gamma'$ defined we define substitution on typed terms by $\Gamma \vdash q : T_p[^{p:\tilde{n}}/_x] = \Gamma \vdash q[^{p:\tilde{n}}/_x] : T'_p$, where $T'_p = T_p \cup \tilde{n}$ if $x \in fv((\,)q)$ and $T'_p = T_p$ otherwise.

Definition 8. A binary relation $\mathcal{R}$ on $p_{/\alpha}$ is *substitutive* if $\Gamma \vdash p \; \mathcal{R} \; p' : S \; \tilde{n}$ and $\Gamma', x : S \vdash p'' \; \mathcal{R} \; p''' : T_p$ with $\Gamma \odot \Gamma'$ defined implies $\Gamma \odot \Gamma' \vdash p'' : T_p[^{p:\tilde{n}}/_x] \; \mathcal{R} \; p''' : T_p[^{p':\tilde{n}}/_x]$. We also say that $\mathcal{R}$ is *constructor compatible* if

- $\Gamma \vdash \mathbf{0} \; \mathcal{R} \; \mathbf{0} : S \; \tilde{n}$ if $\tilde{n} \subseteq dom_n(\Gamma)$,
- $\Gamma, x : S \vdash x \; \mathcal{R} \; x : S \; \tilde{n}$ if $\tilde{n} \subseteq dom_n(\Gamma)$,
- $\Gamma \vdash p \; \mathcal{R} \; p' : S \; \tilde{n}$ and $\Gamma_1 \vdash p_1 \; \mathcal{R} \; p'_1 : S \; \tilde{n}'$ with $\Gamma \odot \Gamma_1$ defined implies $\Gamma \odot \Gamma_1 \vdash p \parallel p_1 \; \mathcal{R} \; p' \parallel p'_1 : S \; \tilde{n} \cup \tilde{n}'$,
- $\Gamma \vdash p \; \mathcal{R} \; p' : S \; \tilde{n}$ and $\Gamma_1 \vdash p_1 \; \mathcal{R} \; p'_1 : S' \; \tilde{n}'$ with $\Gamma \odot \Gamma_1$ defined, and $\Gamma \odot \Gamma_1 \vdash \varphi : S'' \; \mathtt{Ref} \; S'''$ with $S \leq S'$, $S \leq S'''$, and $S'' \leq S'$ implies $\Gamma \odot \Gamma_1 \vdash \varphi\langle p : \tilde{n} \rangle p_1 \; \mathcal{R} \; \varphi\langle p' : \tilde{n} \rangle p'_1 : S' \; \tilde{n} \cup \tilde{n}' \cup \varphi$,
- $\Gamma, x : S' \vdash p \; \mathcal{R} \; p' : S \; \tilde{n}$ and $\Gamma \vdash \varphi : S''' \; \mathtt{Ref} \; S''$ with $S'' \leq S'$ and $S''' \leq S$ implies $\Gamma \vdash \varphi(x)p \; \mathcal{R} \; \varphi(x)p' : S \; \tilde{n} \cup \varphi$,
- $\Gamma, n : S \vdash p \; \mathcal{R} \; p' : T_p n$ implies $\Gamma \vdash (n)p \; \mathcal{R} \; (n)p' : T_p$,
- $\Gamma \vdash p \; \mathcal{R} \; p' : T_p$ and $\forall x \in fv(p).\ \Gamma(x) \leq \mathtt{un}$ implies $\Gamma \vdash !p \; \mathcal{R} \; !p' : T_p$.

Table 8: Howe relation for typed processes and concretions

$$\frac{\Gamma \vdash \mathbf{0}\ \mathcal{R}^{\circ}\ p : T_p}{\Gamma \vdash \mathbf{0}\ \mathcal{R}^{\bullet}\ p : T_p} \qquad \frac{\Gamma \vdash x\ \mathcal{R}^{\circ}\ p : T_p}{\Gamma \vdash x\ \mathcal{R}^{\bullet}\ p : T_p} \qquad \frac{\Gamma \vdash p_1\ \mathcal{R}^{\bullet}\ p_1' : T_p \quad \Gamma' \vdash p_2\ \mathcal{R}^{\bullet}\ p_2' : T_p' \quad \Gamma \odot \Gamma' \vdash p_1' \parallel p_2'\ \mathcal{R}^{\circ}\ p : T_p \parallel T_p'}{\Gamma \odot \Gamma' \vdash p_1 \parallel p_2\ \mathcal{R}^{\bullet}\ p : T_p \parallel T_p'}$$

$$\frac{\Gamma \vdash p_1\ \mathcal{R}^{\bullet}\ p_1' : S\ \tilde{n} \quad \Gamma' \vdash p_2\ \mathcal{R}^{\bullet}\ p_2' : S'\ \tilde{n}' \quad \Gamma \odot \Gamma' \vdash \varphi : S''\ \mathtt{Ref}\ S \quad \Gamma \odot \Gamma' \vdash \varphi\langle p_1' : \tilde{n}\rangle p_2'\ \mathcal{R}^{\circ}\ p'' : S'\ \tilde{n} \cup \tilde{n}' \cup \varphi}{\Gamma \odot \Gamma' \vdash \varphi\langle p_1 : \tilde{n}\rangle p_2\ \mathcal{R}^{\bullet}\ p'' : S'\ \tilde{n} \cup \tilde{n}' \cup \varphi}$$

$$\frac{\Gamma \vdash p_1\ \mathcal{R}^{\bullet}\ p_1' : S\ \tilde{n} \quad \Gamma' \vdash p_2\ \mathcal{R}^{\bullet}\ p_2' : S'\ \tilde{n}' \quad \Gamma \odot \Gamma' \vdash \langle p_1' : \tilde{n}\rangle p_2'\ \mathcal{R}^{\square}\ p'' : \langle S\rangle S'\ \tilde{n} \cup \tilde{n}'}{\Gamma \odot \Gamma' \vdash \langle p_1 : \tilde{n}\rangle p_2\ \mathcal{R}^{\bullet}\ p'' : \langle S\rangle S'\ \tilde{n} \cup \tilde{n}'}$$

$$\frac{\Gamma, x : S' \vdash p\ \mathcal{R}^{\bullet}\ p' : S\ \tilde{n} \quad \Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S' \quad \Gamma \vdash \varphi(x)p'\ \mathcal{R}^{\circ}\ p'' : S\ \tilde{n} \cup \varphi}{\Gamma \vdash \varphi(x)p\ \mathcal{R}^{\bullet}\ p'' : S\ \tilde{n} \cup \varphi}$$

$$\frac{\Gamma, n : S \vdash p\ \mathcal{R}^{\bullet}\ p' : T_p n \quad \Gamma \vdash (n)p'\ \mathcal{R}^{\circ}\ p'' : T_p}{\Gamma \vdash (n)p\ \mathcal{R}^{\bullet}\ p'' : T_p}$$

$$\frac{\Gamma, n : S \vdash c\ \mathcal{R}^{\bullet}\ c' : T_c n \quad \Gamma \vdash (n)c'\ \mathcal{R}^{\square}\ c'' : T_c}{\Gamma \vdash (n)c\ \mathcal{R}^{\bullet}\ c'' : T_c} \qquad \frac{\Gamma \vdash p\ \mathcal{R}^{\bullet}\ p' : T_p \quad \Gamma \vdash !p'\ \mathcal{R}^{\circ}\ p'' : T_p}{\Gamma \vdash !p\ \mathcal{R}^{\bullet}\ p'' : T_p}$$

A well typed relation $\mathcal{R}$ on typed processes $\mathbf{p}_{/\alpha}$ is a *congruence* if it is substitutive (closed under substitution by related processes) and constructor compatible (closed under all process constructors).

The extension of a binary relation $\mathcal{R}$ to open terms p and q where $fv(p) = fv(q) = \{x_1, \ldots, x_k\}$ is done as usual, defining $\Gamma, x_1 : S_1, \ldots, x_k : S_k \vdash p\ \mathcal{R}^{\circ}\ q : T_p$, if for all well-typed closed processes $(1 \leq i \leq k)\ \Delta_i \vdash p_i : S_i\ \tilde{n}_i$ it holds that

$$\Gamma \odot \Delta_1 \odot \cdots \odot \Delta_k \vdash p'\ \mathcal{R}\ q' : T_p \cup \tilde{n}_1 \cup \cdots \cup \tilde{n}_k$$

for $p' = p[{}^{p_1:\tilde{n}_1}/_{x_1}] \cdots [{}^{p_k:\tilde{n}_k}/_{x_k}]$ and $q' = q[{}^{p_1:\tilde{n}_1}/_{x_1}] \cdots [{}^{p_k:\tilde{n}_k}/_{x_k}]$. We extend $\mathcal{R}^{\circ}$ to typed concretions by $\Gamma \vdash c\ \mathcal{R}^{\square}\ c' : \langle S''\rangle S'\ \tilde{n}'$, if for all typed abstractions $\Gamma' \vdash a : S'' \twoheadrightarrow S'\ \tilde{n}''$ with $\Gamma \odot \Gamma'$ defined we have $\Gamma \odot \Gamma' \vdash c \cdot a\ \mathcal{R}^{\circ}\ c' \cdot a : S'\ \tilde{n}'' \cup \tilde{n}'$.

In [GH05] we showed how to extend Howe's method to prove that input-early delay and strong context bisimulations are congruences for Homer. We will outline in the following section that this method can be further extended to affine and unrestricted typed processes with replication.

7.1 Howe's method

The key ingredient in Howe's method is to extend the bisimulation $\approx$ inductively to a constructor compatible relation $\approx^{\bullet}$, sometimes referred to as the *Howe relation*, and prove it to coincide with the bisimulation. In [GH05] we showed how to extend Howe's method to higher-order nested embedded resources by defining the Howe relation for concretions $\approx^{\blacksquare}$ and path contexts $\approx^{\blacktriangle}$. Below we show that the proof extends to linear and non-linearly typed processes with replication.

First we define the *Howe-relation* $\mathcal{R}^{\blacksquare}$ on typed processes and concretions $\mathbf{p}_{/\alpha} \cup \mathbf{c}_{/\alpha}$, relative to a binary relation $\mathcal{R}$ on $\mathbf{p}_{c/\alpha}$ and as the least relation satisfying the rules in Table 8. Let $\mathcal{R}^{\bullet}{=}\mathcal{R}^{\blacksquare} \cap \mathbf{p}_{/\alpha} \times \mathbf{p}_{/\alpha}$, i.e. the relation restricted to (possibly open) processes. Below we will let $\equiv$ denote the structural congruence relation, except for the unfolding of replication.

The first step in proving that the Howe relation is a bisimulation is to prove the following properties.

Table 9: Howe relation for typed path contexts

$$\frac{\Gamma'', \tilde{m} : \tilde{S} \vdash p\ R^{\bullet}\ p' : S\ \tilde{m}\tilde{n}'' \quad \Gamma, \tilde{m}' : \tilde{S}' \vdash \mathcal{D}_{(\Gamma,T_p),\gamma}\ R^{\blacktriangle}\ \mathcal{D}'_{(\Gamma,T_p),\gamma} : S\ \tilde{m}'\tilde{n} \quad \Gamma' \vdash q\ R^{\bullet}\ q' : S'\ \tilde{n}' \quad \Gamma''' \vdash \delta : S''\ \texttt{Ref}\ S}{\Gamma''' \vdash \delta\langle(\tilde{m}'')(\mathcal{D}_{(\Gamma,T_p),\gamma} \parallel p) : \tilde{n}'''\rangle q\ R^{\blacktriangle}\ \delta\langle(\tilde{m}'')(\mathcal{D}'_{(\Gamma,T_p),\gamma} \parallel p') : \tilde{n}'''\rangle q' : S'\ \tilde{n}''''},$$

$\Gamma'', \tilde{m} : \tilde{S} \odot \Gamma, \tilde{m}' : \tilde{S}'$ defined, $\tilde{m}'' = \tilde{m} \cup \tilde{m}'$, $\gamma \cap \tilde{m}'' = \emptyset$, $\Gamma \odot \Gamma' \odot \Gamma'' = \Gamma'''$. $\tilde{n}''' = \tilde{n} \cup \tilde{n}''$, and $\tilde{n}'''' = \tilde{n} \cup \tilde{n}'' \cup \tilde{n}' \cup \delta$.

$$\frac{}{\Gamma, \Gamma' \vdash (-)_{(\Gamma,S\ \tilde{n})}\ R^{\blacktriangle}\ (-)_{(\Gamma,S\ \tilde{n})} : S\ \tilde{n}\tilde{n}'}(\tilde{n}' \subseteq dom_n(\Gamma') \text{ and } \tilde{n} \subseteq dom_n(\Gamma))$$

Proposition 31. *Let $\mathcal{R}$ be an equivalence relation on $\mathbf{p}_{c/\alpha}$ between typed processes then*

1. *$\mathcal{R}^{\blacksquare}$ is reflexive.*
2. *$\mathcal{R}^{\blacksquare}\ \mathcal{R}^{\square} \subseteq \mathcal{R}^{\blacksquare}$.*
3. *$\mathcal{R}^{\square} \subseteq \mathcal{R}^{\blacksquare}$.*
4. *$\mathcal{R}^{\bullet}$ is constructor compatible*
5. *$\mathcal{R}^{\bullet -1} \subseteq \mathcal{R}^{\bullet *}$*
6. *$\mathcal{R}^{\bullet}$ is substitutive*
7. *$\mathcal{R}^{\bullet *}$ is symmetric*

The next step is to prove the following simulation property.

Lemma 32. *For closed processes $\Delta \vdash p : S\ \tilde{n}$ and $\Delta \vdash q : S\ \tilde{n}$ it holds (up to α-equivalence) that $\Delta \vdash p \approx^{\bullet} q : S\ \tilde{n}$ implies*

- *if $p \xrightarrow{\tau} p'$ then $\exists q'.\ q \stackrel{\tau}{\Longrightarrow} q'$ such that $\Delta \vdash p' \equiv\approx^{\bullet} q' : S\ \tilde{n}$*
- *if $p \xrightarrow{\overline{\delta}} a$ and $\Delta \vdash \overline{\delta} : S''\ \texttt{Ref}\ S'$ then $\forall \Delta' \vdash c \approx^{\blacksquare} c' : \langle S'\rangle S\ \tilde{n}'$ with $\Delta \odot \Delta'$ defined, there $\exists a'.\ q \stackrel{\overline{\delta}}{\Longrightarrow} a'$ such that $\Delta \odot \Delta' \vdash c \cdot a \equiv\approx^{\bullet} c' \cdot a' : S\ \tilde{n} \cup \tilde{n}'$*
- *if $p \xrightarrow{\delta} a$ and $\Delta \vdash \delta : S''\ \texttt{Ref}\ S'$ then $\forall \Delta' \vdash c \approx^{\blacksquare} c' : \langle S'\rangle S\ \tilde{n}'$ and $\forall \Delta'' \vdash \mathcal{D}_{(\Delta,S\ \tilde{n})} \approx^{\blacktriangle} \mathcal{D}'_{(\Delta,S\ \tilde{n})} : S'''\ \tilde{n}''$ with $\Delta' \odot \Delta''$ defined there $\exists a'.\ q \stackrel{\overline{\delta}}{\Longrightarrow} a'$ and $\Delta' \odot \Delta'' \vdash c \cdot \mathcal{D}_{(\Delta,S\ \tilde{n})}(a) \equiv\approx^{\bullet} c \cdot \mathcal{D}'_{(\Delta,S\ \tilde{n})}(a') : S'''\ \tilde{n}' \cup \tilde{n}''$.*
- *if $p \xrightarrow{\overline{\delta}} c$ and $\Delta \vdash \overline{\delta} : S''\ \texttt{Ref}\ S'$ then $\exists c'.\ q \stackrel{\overline{\delta}}{\Longrightarrow} c'$ such that $\Delta \vdash c \approx^{\blacksquare} c' : \langle S'\rangle S\ n$*
- *if $p \xrightarrow{\delta} c$ and $\Delta \vdash \delta : S''\ \texttt{Ref}\ S'$ then $\exists c'.\ q \stackrel{\delta}{\Longrightarrow} c'$ such that $\forall \Delta' \vdash \mathcal{D}_{(\Delta,S\ \tilde{n})} \approx^{\blacktriangle} \mathcal{D}'_{(\Delta,S\ \tilde{n})} : S'''\ \tilde{n}'$ we have $\Delta' \vdash \mathcal{D}_{(\Delta,S\ \tilde{n})}(c) \approx^{\blacksquare} \mathcal{D}'_{(\Delta,S\ \tilde{n})}(c') : \langle S'\rangle S'''\ \tilde{n}'$.*

From the lemma above it follows that the transitive closure of $\approx^{\bullet}$ is an input-early delay context bisimulation. From Proposition 31 it then follows that the Howe relation and the bisimulation extended to open terms coincides.

Theorem 33. *$\approx^{\circ}$ and $\sim^{\circ}$ are congruences.*

From Proposition 29 (correspondence between barbs and certain transitions) and Theorem 2 (correspondence between reactions and τ-transitions) we have that $\approx$ is a weak barbed bisimulation, and similar that $\sim$ is a strong barbed bisimulation.

Lemma 34. *$\approx$ is a weak barbed bisimulation, and $\sim$ is a strong barbed bisimulation.*

From Lemma 34 and Thm. 33 we have that the open extension of $\approx$ is sound with respect to weak barbed bisimulation congruence, and that the open extension of $\sim$ is sound with respect to strong barbed bisimulation congruence.

Theorem 35 (soundness)**.** *$\approx^{\circ} \subseteq \approx_b$ and $\sim^{\circ} \subseteq \sim_b$.*

As in [GH05] we can define defining testing contexts for the different kinds of labels, hence proving that strong input-early contextual bisimulation is complete with respect to barbed bisimulation congruence restricted to closed terms.

Proposition 36. $(\sim_b)_c \subseteq \sim$.

Proof. We prove that the relation $\Delta \vdash p\ (\sim_b)_c\ q : S\ \tilde{n}$ is a strong input-early context bisimulation up-to structural congruence (see Proposition 39 for validity of this proof technique). We only present one of the five cases that define a strong input-early context bisimulation (Definition 7). The first case (the case for τ-transitions) follows from Theorem 2 (that τ-transitions and reactions match). Below we present the third case (the case for the reception of a passive resource), the remaining three cases are similar to this case.

- Case: $(p \xrightarrow{\delta} a)$. We have that $p \xrightarrow{\delta} a$ and $\Delta \vdash \delta : S''\ \mathtt{Ref}\ S'$ and we need to prove that for all concretions $c \in \mathbf{c}_{c/\alpha}$ with $\Delta' \vdash c : \langle S' \rangle S'''\ \tilde{n}''$ and for all path-contexts $\mathcal{D}_{(\Delta, S\ \tilde{n})}$ with $\Delta'' \vdash \mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma} : S'''\ \tilde{n}'$ and $\Delta' \odot \Delta''$ defined there exists an abstraction a' such that $q \xrightarrow{\delta} a'$ and $\Delta' \odot \Delta'' \vdash c \cdot \mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}(a)\ (\sim_b)_c\ c \cdot \mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}(a') : S'''\ \tilde{n}' \cup \tilde{n}''$.

  Assume that c is of the form $(\tilde{m})\langle r : \tilde{m}' \rangle r'$ and that $\mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}$ have the form $\mathcal{D}'_\gamma\ \left((-)_{(\Delta,S\ \tilde{n})} \parallel p'\right)$ for some path context $\mathcal{D}'_\gamma$ and some closed process p'. The case is easier if the path-context $\mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}$ is just a hole, so we do not present that case. For clarity we omit the type annotation of $\mathcal{D}'_\gamma$.

  Without loss of generality we assume that the last name in δ is some n i.e. $\delta = \gamma'' n$. So we know that $\Delta(n) = S'$. Let j be an integer greater than the sum of all address path lengths in resources in c and in $\mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}$. We write n^j for the address path consisting of the concatenation of j copies of n. For a fresh name m we define the following context

$$C = (\tilde{m})\left(\overline{\gamma n^j}\langle r : \tilde{m}' \rangle \overline{\gamma n^j}(x)(r' \parallel m\langle \mathbf{0} : \emptyset \rangle)\right) \parallel \mathcal{D}'_\gamma\ \left((-)_{(\Delta,S\ \tilde{n})} \parallel n^j(x)\overline{\delta}\langle x : \emptyset \rangle n^j \langle \mathbf{0} : \emptyset \rangle p'\right)$$

  (for clarity we leave out the type annotation of C). Note that C is well-typed in the extended environment $\Delta' \odot \Delta'', m : S'''$. We now have the following reductions

$$\begin{array}{rcll} C(p) & \searrow & (\tilde{m})\left(\overline{\gamma n^j}(x)(r' \parallel m\langle \mathbf{0} : \emptyset \rangle) \parallel \mathcal{D}''_\gamma\left(p \parallel \overline{\delta}\langle r : \tilde{m}' \rangle n^j \langle \mathbf{0} : \emptyset \rangle p'\right)\right) & \\ & \searrow & (\tilde{m})\left(\overline{\gamma n^j}(x)(r' \parallel m\langle \mathbf{0} : \emptyset \rangle) \parallel \mathcal{D}''_\gamma\left(p'' \parallel n^j \langle \mathbf{0} : \emptyset \rangle p'\right)\right) & (*) \\ & \searrow & (\tilde{m})\left(r' \parallel m\langle \mathbf{0} : \emptyset \rangle \parallel \mathcal{D}''_\gamma\left(p'' \parallel p'\right)\right) & \\ & \equiv & c \cdot \mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}(a) \parallel m\langle \mathbf{0} : \emptyset \rangle = p''' \parallel m\langle \mathbf{0} : \emptyset \rangle\ , & \end{array}$$

  where the only difference between $\mathcal{D}'$ and $\mathcal{D}''$ is that we have added the names in $\tilde{m}'$ to the type annotations in the locations constituting the path down to the hole. Note that a synchronisation happens between p and $\overline{\delta}\langle r : \tilde{m}' \rangle n^j \langle \mathbf{0} : \emptyset \rangle p'$ in the reaction marked with (*), thus capturing the transition of p.

  From $\Delta \vdash p\ (\sim_b)_c\ q : S\ \tilde{n}$ it follows that $\Delta' \odot \Delta'', m : S''' \vdash C(p)\ (\sim_b)_c\ C(q) : S'''\ (\tilde{n}' \cup \tilde{n}'')m$, so we have that

$$C(q) \searrow\searrow\searrow q''' \parallel m\langle \mathbf{0} : \emptyset \rangle\ ,$$

  where $q''' = c \cdot \mathcal{D}_{(\Delta,S\ \tilde{n}),\gamma}(a')$ for some $q \xrightarrow{\delta} a'$ and

$$\Delta' \odot \Delta'', m : S''' \vdash p''' \parallel m\langle \mathbf{0} : \emptyset \rangle\ (\sim_b)_c\ q''' \parallel m\langle \mathbf{0} : \emptyset \rangle : S'''\ (\tilde{n}' \cup \tilde{n}'')m\ .$$

  We can now use the context $(-) \parallel m(x)$ to remove the resource $m\langle \mathbf{0} : \emptyset \rangle$ (again omitting the type annotation of the hole). So we get

$$\Delta' \odot \Delta'', m : S''' \vdash p'''\ (\sim_b)_c\ q''' : S'''\ (\tilde{n}' \cup \tilde{n}'')m\ .$$

Finally, using the context $(m)(-)$ and $\equiv$ we obtain

$$\Delta' \odot \Delta'' \vdash p''' \equiv (\sim_b)_c \equiv q''' : S''' \ \tilde{n}' \cup \tilde{n}'' \ .$$

□

The result extends to open processes since $\sim_b$ is substitutive.

Corollary 37 (completeness)**.** $\sim_b \subseteq \sim^\circ$.

Combining the results of Corollary 37 and Theorem 35 we obtain that the open extension of strong input-early contextual bisimulation is a sound and complete characterisation of strong barbed bisimulation congruence.

Theorem 38 (sound and complete characterisation)**.** $\sim^\circ = \sim_b$.

For the same reasons as in [GH05] we conjecture that input-early delay contextual bisimulation is not complete with respect to weak barbed bisimulation.

7.2 Scope Extension Across Affine Location

In this section we use the bisimulation congruence to prove that the scope of a local name can be extended across the boundary of an affine location. In order to simplify the proof we will again use a standard enhancement of the bisimulation proof method [San98], bisimulation up-to $\equiv$. For brevity, we will not write up the definition of bisimulation up-to $\equiv$.

Proposition 39. *If $\mathcal{R}$ is a strong bisimulation up-to $\equiv$ then $\mathcal{R} \subseteq \sim$.*

Theorem 40. *For $m \neq n$, $\Delta, n : \mathtt{aff} \vdash (m)n\langle p : \tilde{m}m\rangle \sim n\langle (m)p : \tilde{m}\rangle : \mathtt{aff}\ \tilde{m} \cup n$.*

Proof. We will write $p >^*_m q$ for two processes p and q which are syntactical equivalent except that p can contain *some* m annotations on affine resources which does not occur in q. Furthermore, we will write $q = p \setminus \{m\}$ for two processes p and q which are syntactical equivalent except for *all* m annotations in p and these annotations must only occur in affine resources. Let $\mathcal{R}$ denote the following well-typed relation

$$\{(\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}((m)(p'[^{p:\tilde{m}m}/_x])),\ \mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}(q'[^{(m)q:\tilde{m}}/_x]))\}$$

for all processes $p >^*_m q$ and $q' = p' \setminus \{m\}$ with $\Delta \vdash (m)p'[^{p:\tilde{m}m}/_x] : \mathtt{aff}\ \tilde{n}$ and $\Delta \vdash q'[^{(m)q:\tilde{m}}/_x] : \mathtt{aff}\ \tilde{n}$ with x affine and any context $\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}$ with $\Delta' \vdash \mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})} : \mathtt{aff}\ \tilde{n}'$ where the hole does not occur under a replication. We then prove that $\mathcal{R}$ is a strong context bisimulation up-to $\equiv$.

The structure of the proof is as follows. Assume we have two processes related by $\mathcal{R}$, i.e. $p'' \ \mathcal{R}\ p'''$ and p'' can perform a transition, $p'' \xrightarrow{\pi} t$, then by the results in Appendix B.1 we can deduce that p'' and t have a certain structure (up-to $\equiv$). From the structure of p'' we can deduce that p''' has a certain (similar) structure (again up-to $\equiv$), as p'' and p''' only differ on the location of the restriction of m and on m annotations on affine resources. From the structure of p''' we can deduce that p''' can match the transition and end up in a term t' which is related to t by $\mathcal{R}$ (note that "related to" means something different depending on the particular case in the bisimulation) using the results in Appendix B.2. The proof is *very* long and tedious and we refer to Appendix C for some of the cases. □

The result in Theorem 40 suggests a normal form for affine and non-linear Homer, where we extend the scope of local names outside affine locations. We will leave this, and related, directions of research for future work.

8 An e-cash Smart Card application

In this section we provide a simple model of an e-cash system that illustrates the combination of linear and non-linear mobile resources, nested locations, and local names. Consider first a process defined by

$$crypt_{e,k} = e(x)e\langle k\langle x : \emptyset\rangle : \{k\}\rangle \ .$$

The process is able to receive a resource on the name e, which is then placed inside a location named k nested in a location named e. If k is cryptographic key, one can think of the process as being able to perform a single encryption of the received process. This can be utilised in a simple e-cash system consisting of an ATM that is able to provide a coin $\overline{c}\langle \mathbf{0} : \emptyset\rangle$, if the process in the location v can encrypt a nonce n with the private key k:

$$\begin{array}{lcl} atm & = & (k)\big(v\langle crypt_{e,k} : \{e,k\}\rangle \parallel cash_k\big) \\ cash_k & = & !(n)\ \big(\overline{ve}\langle n\langle \mathbf{0} : \emptyset\rangle : \{n\}\rangle\overline{vekn}(x)\overline{c}\langle \mathbf{0} : \emptyset\rangle\big) \ . \end{array}$$

In the control process $cash_k$ of the ATM a nonce process $n\langle \mathbf{0} : \emptyset\rangle$ is sent to the location e inside the process in the location v. Subsequently, a process is retrieved from the sub location $vekn$. If this succeeds, it must be the case that the process inside the location v has embedded the nonce in the location k, and the ATM then emits a coin. The control process can potentially be executed any number of times. The intended behaviour is however, that only one coin will ever be delivered, since the method on the card can only encrypt once.

Alas, if the process in the slot v can be copied, the security is broken. A e-cash copying thief may be defined by

$$thief \quad = \quad \overline{v}(x)\,(v\langle x : \emptyset\rangle \parallel v\langle x : \emptyset\rangle) \ ,$$

which picks up the e-cash process by $\overline{v}(x)$ and creates two copies. Then

$$\begin{array}{rcl} atm \parallel thief & \xrightarrow{\tau}\equiv & (k)\big(v\langle crypt_{e,k} : \{e,k\}\rangle \parallel v\langle crypt_{e,k} : \{e,k\}\rangle \parallel cash_k\big) \\ & \xrightarrow{\tau}{}^{*}\equiv & (k)cash_k \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle \end{array}$$

The type system presented in the previous section allows us to type the location v as affine linear. Thereby, we can model that the process in location v is intended as being embedded in a non-copyable smart card (and also ensure that the entire system cannot be copied either). First, we show that the system is well-typed.

Lemma 41. *Let* $\Delta = e : \mathtt{un}, c : \mathtt{aff}, v : \mathtt{aff}$*, then* $\Delta \vdash atm : \mathtt{aff}\ \{e, c, v\}$.

We then show, that we cannot type the system $atm \parallel thief$, if the slot v is linear.

Proposition 42. *For any* $\Delta, v : \mathtt{aff}$*,* $\tilde{n}$ *and sort* S *it is not possible to derive* $\Delta, v : \mathtt{aff} \vdash atm \parallel thief : S\ \tilde{n}$.

Proof. Assume that it is possible to derive $\Delta, v : \mathtt{aff} \vdash atm \parallel thief : S\ \tilde{n}$, by inspecting the derivation, and without loss of generality, it must then also be possible to derive $\Delta, v : \mathtt{aff}, x : \mathtt{aff} \vdash v\langle x : \emptyset\rangle \parallel v\langle x : \emptyset\rangle : S\ \tilde{n}$, but this contradicts Lemma 16 (that x occurs free at most once). □

We leave for future work to apply the bisimulation congruence to prove that the typed *atm* is indeed secure in any context. Note that the encrypted nonce is unrestricted. If we accidentally had defined the $cash_k$ control process as $(n)!\ \big(\overline{ve}\langle n\langle \mathbf{0} : \emptyset\rangle : \{n\}\rangle\overline{vekn}(x)\overline{c}\langle \mathbf{0} : \emptyset\rangle\big)$, i.e. swapping the local name (n) and the replication and thus repeatedly using the same secret name n as challenge for the card, the security would be broken. A thief copying the encrypted content of the card could be defined by:

$$thief \quad = \quad \overline{ve}(x)(v\langle e\langle x : \emptyset\rangle : \{e\}\rangle \parallel v\langle e\langle x : \emptyset\rangle : \{e\}\rangle) \ .$$

This security threat would not show in a purely linear calculus.

9 Conclusions and Further Work

We have successfully extended the prior type and effect system and labelled bisimulation congruence for Homer to provide the first process calculus combining affine linear and non-linear nested mobile embedded processes with local names. We have demonstrated that the bisimulation congruence allows us to prove scope extension to be safe across affine linear locations, which in previous work has been shown to be unsafe for non-linear locations. By a concrete e-cash Smart Card system we have exemplified that the calculus captures the difference between mobile *computing* hardware and embedded mobile software *computations*, which is crucial for the security of pervasive and ubiquitous computing.

We believe that the type system presented for Homer in the present paper can be adapted to other calculi combining mobile embedded resources with local names, as for instance the Seal calculus. We expect to investigate other variations and applications of linear types and more expressive type systems for Homer within the research projects for Mobile Security and Computer Supported Mobile Adaptive Business Processes (CosmoBiz) at ITU. In particular, we plan to investigate more complex type systems, expressing not only copyability capabilities, but also *access* capabilities to resources like the ones presented in [CVZN05] and [YH04], and *shapes*, e.g. for controlling the number of sub-resources within a resource or more generally a schema for processes. We also plan investigate whether part of the approach presented in [Yos04] can be applied in the setting of Homer, even though Homer does not contain name passing and hence name substitution. In this paper we have only touched the surface of assigning types to composite addresses, an area primarily left untouched in the existing literature. So far only a few type systems exist [Car05] for calculi with composite addresses. We plan to examine this as future work.

The proof of Theorem 40 required that we consider general processes in constructing the bisimulation relation, due to the universal quantification over contexts in the definition of context bisimulation. We plan to investigate whether enhancements to the bisimulation proof method such as bisimulation up-to context can be adapted to calculi with affine linear and non-linear active process mobility and explicit, nested locations.

Acknowledgements: We wish to thank the anonymous referees of a previous version of the paper for helpful comments.

References

[ANN99] Torben Amtoft, Flemming Nielson, and Hanne Riis Nielson. *Type and Effect Systems.* Imperial College Press, 1999.

[Bar84] Henk Barendregt. *The Lambda Calculus - its Syntax and Semantics.* North-Holland Publishing Co., 1984.

[Bar96] Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, LFCS, Department of Computer Science, University of Edinburgh, 1996.

[BB90] Gerard Berry and Gérard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL'90)*, pages 81–94. ACM Press, 1990.

[BH06] Mikkel Bundgaard and Thomas Hildebrandt. Bigraphical semantics of higher-order mobile embedded resources with local names. In Arend Rensink, Reiko Heckel, and Barbara König, editors, *Proceedings of the Graph Transformation for Verification and Concurrency workshop (GT-VC'05)*, volume 154 of *Electronic Notes in Theoretical Computer Science*, pages 7–29. Elsevier, 2006.

[BHG06] Mikkel Bundgaard, Thomas Hildebrandt, and Jens Chr. Godskesen. A CPS encoding of name-passing in higher-order mobile embedded resources. *Theoretical Computer Science*, 356(3):422–439, 2006.

[BHY01] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In Samson Abramsky, editor, *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*, volume 2044 of *Lecture Notes in Computer Science*, pages 29–45. Springer Verlag, 2001.

[BHY05] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the π-calculus. *Acta Informatica*, 42(2–3):83–141, 2005.

[BS03] Philippe Bidinger and Jean-Bernard Stefani. The Kell calculus: Operational semantics and type system. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *Proceedings of the 5th IFIP International Conference on Formal Methods for Object-Based Distributed Systems (FMOODS'03)*, volume 2884 of *Lecture Notes in Computer Science*, pages 109–123. Springer Verlag, 2003.

[Car05] Marco Carbone. *Trust and Mobility.* PhD thesis, BRICS, 2005.

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

[CM03] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[CVZN05] Giuseppe Castagna, Jan Vitek, and Fracesco Zappa Nardelli. The Seal calculus. *Journal of Information and Computation*, 201(1):1–54, 2005.

[Frü98] Thom W. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1–3):95–138, 1998.

[GH04] Jens Chr. Godskesen and Thomas Hildebrandt. Copyability types for mobile computing resources. Presented at the International Workshop on Formal Methods and Security, Nanjing, China, 2004.

[GH05] Jens Chr. Godskesen and Thomas Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. In *Proceedings of the 25th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 140–151. Springer Verlag, 2005.

[GHS02] Jens Christian Godskesen, Thomas Hildebrandt, and Vladimiro Sassone. A calculus of mobile resources. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 272–287. Springer Verlag, 2002.

[HGB04] Thomas Hildebrandt, Jens Chr. Godskesen, and Mikkel Bundgaard. Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources. Technical Report TR-2004-52, IT University of Copenhagen, 2004.

[Kob02] Naoki Kobayashi. Type systems for concurrent programs. In *Proceedings of UNU/IIST 10th Anniversary Colloquium*, 2002.

[KPT99] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(5):914–947, 1999.

[San92] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

[San98] Davide Sangiorgi. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

[SS03] Alan Schmitt and Jean-Bernard Stefani. The M-calculus: A higher-order distributed process calculus. In *Proceedings of the 30th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL'03)*, pages 50–61. ACM Press, 2003.

[SS04] Alan Schmitt and Jean-Bernard Stefani. The Kell calculus: A family of higher-order distributed process calculi. In Corrado Priami and Paola Quaglia, editors, *Proceedings of the International Workshop on Global Computing Workshop (GC'04)*, volume 3267 of *Lecture Notes in Computer Science*, pages 146–178. Springer Verlag, 2004.

[SV06] Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Journal of Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

[Tho93] Bent Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[TW99] David N. Turner and Philip Wadler. Operational interpretations of linear logic. *Theoretical Computer Science*, 227(1–2):231–248, 1999.

[Wal04] David Walker. Substructural type systems. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 3–43. MIT Press, 2004.

[YH04] Nobuko Yoshida and Matthew Hennessy. Assigning types to processes. *Journal of Information and Computation*, 174(2):143–179, 2004.

[Yos04] Nobuko Yoshida. Channel dependent types for higher-order mobile processes (extended abstract). In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL'04)*, pages 147–160. ACM Press, 2004.

A Results about Shorthands and Typing

In this section we list some results about the typing of the closure of concretions and abstractions under process operators. The results are primarily used in the proof of subject reduction of the labelled transition system. We prove all the propositions by unfolding the definition and using the typing rules.

Proposition 43. *If* $\Gamma'' \vdash a : S'' \twoheadrightarrow S'\ \tilde{n}''$ *and* $\Gamma' \vdash p' : S'\ \tilde{n}'$ *and* $\Gamma'' \odot \Gamma'$ *defined then* $\Gamma'' \odot \Gamma' \vdash a \parallel p' : S'' \twoheadrightarrow S'\ \tilde{n}'' \cup \tilde{n}'$.

Proposition 44. *If* $\Gamma'' \vdash c : \langle S'' \rangle S'\ \tilde{n}''$ *and* $\Gamma' \vdash p' : S'\ \tilde{n}'$ *and* $\Gamma'' \odot \Gamma'$ *defined then* $\Gamma'' \odot \Gamma' \vdash c \parallel p' : \langle S'' \rangle S'\ \tilde{n}'' \cup \tilde{n}'$.

Proposition 45. *If* $\Gamma, n : S \vdash a : S'' \twoheadrightarrow S'\ \tilde{n}n$ *then* $\Gamma \vdash (n)a : S'' \twoheadrightarrow S'\ \tilde{n}$.

Proposition 46. *If* $\Gamma, n : S \vdash c : \langle S'' \rangle S'\ \tilde{n}n$ *then* $\Gamma \vdash (n)c : \langle S'' \rangle S'\ \tilde{n}$.

Proposition 47. *If* $\Gamma'' \vdash a : S'' \twoheadrightarrow T'\ \tilde{n}''$ *and* $\Gamma' \vdash p' : S'\ \tilde{n}'$ *with* $\Gamma'' \odot \Gamma'$ *defined and* $\Gamma'' \odot \Gamma' \vdash \delta : T\ \texttt{Ref}\ T'$ *and* $T \leq S'$ *then* $\Gamma'' \odot \Gamma' \vdash \delta\langle a : \tilde{n}'' \rangle p' : S'' \twoheadrightarrow S'\ \tilde{n}'' \cup \delta \cup \tilde{n}'$.

Proposition 48. *If* $\Gamma'' \vdash c : \langle S''' \rangle T'\ \tilde{n}''$ *and* $\Gamma' \vdash p' : S'\ \tilde{n}'$ *with* $\Gamma'' \odot \Gamma'$ *defined and* $\Gamma'' \odot \Gamma' \vdash \delta : T\ \texttt{Ref}\ T'$ *and* $T \leq S'$ *then* $\Gamma'' \odot \Gamma' \vdash \delta\langle c : \tilde{n}'' \rangle p' : \langle S''' \rangle S'\ \tilde{n}'' \cup \delta \cup \tilde{n}'$.

The above results are also used in the following two propositions for typing an abstraction or concretion placed inside a path context, which we use in the subject reduction of the reaction semantics.

Proposition 49. *If* $\Gamma' \vdash \mathcal{D}_{(\Gamma, T_p)} : S'\ \tilde{n}'$ *and* $\Gamma \vdash a : S'' \twoheadrightarrow T_p$ *then* $\Gamma' \vdash \mathcal{D}_{(\Gamma, T_p)}(a) : S'' \twoheadrightarrow S'\ \tilde{n}'$.

Proposition 50. *If* $\Gamma' \vdash \mathcal{D}_{(\Gamma, T_p)} : S'\ \tilde{n}'$ *and* $\Gamma \vdash c : \langle S'' \rangle T_p$ *then* $\Gamma' \vdash \mathcal{D}_{(\Gamma, T_p)}(c) : \langle S'' \rangle S'\ \tilde{n}'$.

B Results Relating Structure and Transitions

In this section we present some auxiliary results relating transitions and the structure of terms involved in the transition. All of the results are used in the proof of Theorem 40.

B.1 From Transition to Structure

First, we characterise the structure of a term involved in a transition, from the label of the transition and the resulting kind of term (process, abstraction, or concretion). We state the results for untyped terms for simplicity, and since we can lift them to the typed setting straightforwardly.

Proposition 51. *If* $p \xrightarrow{\overline{\delta}} a$ *then* $p \equiv (\tilde{n})(\overline{\delta}a' \parallel p'')$ *and* $a \equiv (\tilde{n})(a' \parallel p'')$ *for some abstraction* a' *and process* p'' *and* $\tilde{n} \cap \overline{\delta} = \emptyset$.

Proposition 52. *If* $p \xrightarrow{\overline{\delta}} c$ *then* $p \equiv (\tilde{n})(\overline{\delta}c' \parallel p'')$ *and* $c \equiv (\tilde{n})(c' \parallel p'')$ *for some concretion* c' *and process* p'' *and* $\tilde{n} \cap \overline{\delta} = \emptyset$.

Proposition 53. *If* $p \xrightarrow{\delta} a$ *then* $p \equiv (\tilde{n})(\mathcal{D}_\gamma(\delta' a') \parallel p'')$ *and* $a \equiv (\tilde{n})(\mathcal{D}_\gamma(a') \parallel p'')$ *for some abstraction* a' *and process* p'' *and* $\delta = \gamma\delta'$, $\tilde{n} \cap \delta = \emptyset$, *and* $bn(\mathcal{D}_\gamma) \cap \delta' = \emptyset$.

Proposition 54. *If* $p \xrightarrow{\delta} c$ *then* $p \equiv (\tilde{n})(\mathcal{D}_\gamma(\delta' c') \parallel p'')$ *and* $c \equiv (\tilde{n})(\mathcal{D}_\gamma(c') \parallel p'')$ *for some abstraction* c' *and process* p'' *and* $\delta = \gamma\delta'$, $\tilde{n} \cap \delta = \emptyset$, *and* $bn(\mathcal{D}_\gamma) \cap \delta' = \emptyset$.

Lemma 55. *If* $\tilde{n}_1$ *is fresh for all names (both free and bound) in* $(\tilde{n}_2)(\overline{e} \parallel p_2)$ *and* $\tilde{n}_2$ *is fresh for all names in* $(\tilde{n}_1)(e \parallel p_1)$ *then* $(\tilde{n}_1)(e \parallel p_1) \cdot (\tilde{n}_2)(\overline{e} \parallel p_2) \equiv (\tilde{n}_1\tilde{n}_2)(e \cdot \overline{e} \parallel (p_1 \parallel p_2))$.

Proposition 56. *If* $p \xrightarrow{\tau} p'$ *then* $p \equiv \mathcal{E}(\overline{\gamma\delta}e \parallel \mathcal{D}_\gamma(\delta\overline{e}))$ *and* $p' \equiv \mathcal{E}(e \cdot \mathcal{D}_\gamma(\overline{e}))$ *and* $bn(\mathcal{D}_\gamma) \cap \delta = \emptyset$.

B.2 From Structure to Transition

The following results allow us to reason in the other direction, i.e. if a process have a certain structure then it is possible to perform a transition ending up in a term of a certain structure.

Proposition 57. *If* $p \equiv (\tilde{n})(\overline{\delta}c' \parallel p''')$ *and* $\tilde{n} \cap \overline{\delta} = \emptyset$ *then there exists some* c *such that* $p \xrightarrow{\overline{\delta}} c$ *and* $c \equiv (\tilde{n})(c' \parallel p''')$.

Proposition 58. *If* $p \equiv (\tilde{n})(\overline{\delta}a' \parallel p''')$ *and* $\tilde{n} \cap \overline{\delta} = \emptyset$ *then there exists some* a *such that* $p \xrightarrow{\overline{\delta}} a$ *and* $a \equiv (\tilde{n})(a' \parallel p''')$.

Lemma 59. *If* $p \xrightarrow{\delta} t$ *and* $bn(\mathcal{D}_\gamma) \cap \delta = \emptyset$ *then* $\mathcal{D}_\gamma(p) \xrightarrow{\gamma\delta} \mathcal{D}_\gamma(t)$.

Proposition 60. *If* $p \equiv (\tilde{n})(\mathcal{D}_\gamma(\delta'c') \parallel p'')$ *and* $\tilde{n} \cap \gamma\delta' = \emptyset$ *and* $bn(\mathcal{D}_\gamma) \cap \delta' = \emptyset$ *then there exists some* c *such that* $p \xrightarrow{\gamma\delta'} c$ *and* $c \equiv (\tilde{n})(\mathcal{D}_\gamma(c') \parallel p'')$.

Proposition 61. *If* $p \equiv (\tilde{n})(\mathcal{D}_\gamma(\delta'a') \parallel p'')$ *and* $\tilde{n} \cap \gamma\delta' = \emptyset$ *and* $bn(\mathcal{D}_\gamma) \cap \delta' = \emptyset$ *then there exists some* a *such that* $p \xrightarrow{\gamma\delta'} a$ *and* $a \equiv (\tilde{n})(\mathcal{D}_\gamma(a') \parallel p'')$.

Lemma 62. *If* $p \xrightarrow{\tau} p'$ *then* $\mathcal{E}(p) \xrightarrow{\tau} \mathcal{E}(p')$.

Proposition 63. *If* $p \equiv \mathcal{E}(\overline{\gamma\delta}e \parallel \mathcal{D}_\gamma(\delta\overline{e}))$ *and* $bn(\mathcal{D}_\gamma) \cap \delta = \emptyset$ *then there exists some* p' *such that* $p \xrightarrow{\tau} p'$ *and* $p' \equiv \mathcal{E}(e \cdot \mathcal{D}_\gamma(\overline{e}))$.

C Sketch of Proof of Theorem 40

In this section we present some of the cases for proving that the well-typed relation $\mathcal{R}$ of Theorem 40 is a strong context bisimulation up-to $\equiv$. First, we recall $\mathcal{R}$

$$\{(\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}((m)(p'[^{p:\tilde{m}m}/_x])),\ \mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}(q'[^{(m)q:\tilde{m}}/_x]))\}$$

for all processes $p >^*_m q$ and $q' = p' \setminus \{m\}$ with $\Delta \vdash (m)p'[^{p:\tilde{m}m}/_x] : \mathtt{aff}\ \tilde{n}$ and $\Delta \vdash q'[^{(m)q:\tilde{m}}/_x] : \mathtt{aff}\ \tilde{n}$ with x affine and any context $\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}$ with $\Delta' \vdash \mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})} : \mathtt{aff}\ \tilde{n}'$ where the hole does not occur under a replication.

We have, for brevity, decided to present only two of the cases of proving that $\mathcal{R}$ is a strong context bisimulation up-to $\equiv$ (and only in one direction, as the work done in the opposite direction is quite similar): when the left-hand side process p'' does a δ-transition to some abstraction a, and when it does a $\overline{\delta}$-transition to some concretion c (i.e. the cases 3 and 4 in the bisimulation). In the following let p'' and q'' be related by $\mathcal{R}$, i.e. $\Delta' \vdash p'' \mathcal{R} q'' : \mathtt{aff}\ \tilde{n}'$.

Case 3 Assume that $p'' \xrightarrow{\delta} a$ (1) and $\Delta' \vdash \delta : S''\ \mathtt{Ref}\ S'$ then we must show that for all concretions $c \in \mathbf{c}_{c/\alpha}$ with $\Delta'' \vdash c : \langle S'\rangle S'''\ \tilde{n}''$ and for all path contexts $\mathcal{D}_{(\Delta',\mathtt{aff}\ \tilde{n}')}$ with $\Delta''' \vdash \mathcal{D}_{(\Delta',\mathtt{aff}\ \tilde{n}')} : S'''\ \tilde{n}'''$ and $\Delta'' \odot \Delta'''$ defined there exists some a'' such that $q'' \xrightarrow{\overline{\delta}} a''$ and $\Delta'' \odot \Delta''' \vdash c \cdot \mathcal{D}_{(\Delta',\mathtt{aff}\ \tilde{n}')}(a) \equiv \mathcal{R} \equiv c \cdot \mathcal{D}_{(\Delta',\mathtt{aff}\ \tilde{n}')}(a'') : S'''\ \tilde{n}'' \cup \tilde{n}'''$.

From (1) and Proposition 53 we have that $p'' \equiv (\tilde{n}''')(\mathcal{D}'_\gamma(\delta'a') \parallel p''')$ and $a \equiv (\tilde{n}''')(\mathcal{D}'_\gamma(a') \parallel p''')$ and $\delta = \gamma\delta'$, $\tilde{n}''' \cap \delta = \emptyset$, and $bn(\mathcal{D}'_\gamma) \cap \delta' = \emptyset$, so $\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}((m)(p'[^{p:\tilde{m}m}/_x])) \equiv (\tilde{n}''')(\mathcal{D}'_\gamma(\delta'a') \parallel p''')$ (for simplicity we leave out the type annotation of $\mathcal{D}'_\gamma$). Recall that $\mathcal{C}$ can be an arbitrary context, except that the hole cannot occur under a replication. We proceed by considering all the different cases where the hole of $\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}$ can occur in $(\tilde{n}''')(\mathcal{D}'_\gamma(\delta'a') \parallel p''')$:

- Hole inside p'''. Then we can write p''' as $\mathcal{C}'_{(\Delta,\texttt{aff }\tilde{n})}((m)p'[^{p:\tilde{m}m}/_x])$ for some $\mathcal{C}'$, so we know that q'' is of the form $q'' \equiv (\tilde{n}''')(\mathcal{D}'_\gamma(\delta' a') \parallel \mathcal{C}'_{(\Delta,\texttt{aff }\tilde{n})}(q'[^{(m)q:\tilde{m}}/_x]))$, so by Proposition 61 there exists some a'' such that $q'' \xrightarrow{\delta} a''$ and $a'' \equiv (\tilde{n}''')(\mathcal{D}'_\gamma(a') \parallel \mathcal{C}'_{(\Delta,\texttt{aff }\tilde{n})}(q'[^{(m)q:\tilde{m}}/_x]))$. We need to prove that $\Delta'' \odot \Delta''' \vdash c \cdot \mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}(a) \equiv \mathcal{R} \equiv c \cdot \mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}(a'') : S''' \ \tilde{n}'' \cup \tilde{n}'''$, but this clearly follows from the definition of application as, the only place that the two processes differ is not touched by the application.
- Hole under a' (similar to the case above, note that we fill the hole with closed processes, so the substitution cannot affect p or q in this case).
- Hole in $\mathcal{D}'_\gamma$ but only constitute part of the path context (similar to the two cases above).
- Hole in $\mathcal{D}'_\gamma$ and δ' from either p' or p. We only present the first case, as the second case is similar. Again, there are two cases to consider depending on whether p occurs in a'. Again, we only present one of the cases: the case where p does occur in a', the other case is similar.

  We have that $p'' \equiv (\tilde{n}''')(\mathcal{D}'_\gamma(\delta' a') \parallel p''')$ and without loss of generality we can split up the path context $\mathcal{D}'_\gamma$ into the part that comes from $\mathcal{C}_{(\Delta,\texttt{aff }\tilde{n})}$ and the part that comes from $p'[^{p:\tilde{m}m}/_x]$, so there exists some $\mathcal{D}''_{\gamma'}$ such that $\mathcal{D}'_\gamma(\delta' a') \equiv \mathcal{D}''_{\gamma'}((m\tilde{m}')(\mathcal{D}'''_{\gamma''}(\delta' a') \parallel r))$ where $p'[^{p:\tilde{m}m}/_x] \equiv (\tilde{m}')(\mathcal{D}'''_{\gamma''}(\delta' a') \parallel r)$, $\gamma'\gamma'' = \gamma$, and we know that p occurs inside a'.

  So we have that $q'' \equiv (\tilde{n}''')(\mathcal{D}''_{\gamma'}((\tilde{m}')(\mathcal{D}''''_{\gamma''}(\delta' a''') \parallel r')) \parallel p''')$ and $\mathcal{D}''''_{\gamma''} = \mathcal{D}'''_{\gamma''} \setminus \{m\}$, $r' = r \setminus \{m\}$. Without loss of generality assume that $a' = (x)(r''[^{p:\tilde{m}m}/_x])$ then $a''' = (x)(r'''[^{(m)q:\tilde{m}}/_x])$ with $r''' = r'' \setminus \{m\}$. So by Proposition 61 we know that there exists some a'' such that $q'' \xrightarrow{\delta} a''$ and $a'' \equiv (\tilde{n}''')(\mathcal{D}''_{\gamma'}((\tilde{m}')(\mathcal{D}''''_{\gamma''}(a''') \parallel r')) \parallel p''')$.

  It remains to prove that $\Delta'' \odot \Delta''' \vdash c \cdot \mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}(a) \equiv \mathcal{R} \equiv c \cdot \mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}(a'') : S''' \ \tilde{n}'' \cup \tilde{n}'''$, Note that the local names and the residual of the concretion, the path context $\mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}$, and the bodies of the abstractions a and a'' down to the hole of $\mathcal{D}''_{\gamma'}$, by the definition of application, together constitutes a common context for both sides of the relation. Also note that the application will not affect, p and q and their relationship, as we don't substitute into p and q, since they are closed processes. The processes r and r' remain unchanged in the application, $\mathcal{D}''''_{\gamma''}$ and $\mathcal{D}'''_{\gamma''}$ can have some of their annotations updated, but not with the local name m and hence remain in relation. Finally, r''' and r'' can change due to the substitution, but since we substitute in the same process in both processes, and since the process does not contain the local name m, the process are still related after the substitution.
- Finally, we need to handle the case where the entire path context $\mathcal{D}'_\gamma$ comes from either p', p, or both. Either the top-location in $\mathcal{D}'_\gamma$ comes p' or from p.
  - Top-location in $\mathcal{D}'_\gamma$ is from p. Then we know that x occurs unrestricted in p', so we can rewrite q'' to exactly the same form as p'' up-to m-annotations on affine locations. So $q'' \equiv (\tilde{n}''')(\mathcal{D}''_\gamma(\delta' a''') \parallel p'''')$, where $\mathcal{D}'_\gamma >^*_m \mathcal{D}''_\gamma$, $a' >^*_m a'''$, and $p''' >^*_m p''''$. So by Proposition 61 we know that there exists some a'' such that $q'' \xrightarrow{\delta} a''$ and $a'' \equiv (\tilde{n}''')(\mathcal{D}''_\gamma(a''') \parallel p'''')$. We need to prove that the resulting abstractions are related, i.e. $\Delta'' \odot \Delta''' \vdash c \cdot \mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}(a) \equiv \mathcal{R} \equiv c \cdot \mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}(a'') : S''' \ \tilde{n}'' \cup \tilde{n}'''$. But clearly the residual of the concretion and the path context $\mathcal{D}_{(\Delta',\texttt{aff }\tilde{n}')}$ can be matched by the context in the relation, and since we substitute in the same process in a and a'' they are still related by $>^*_m$ (up-to $\equiv$) after the substitution. So the resulting processes are still in the relation.
  - Top-location in $\mathcal{D}'_\gamma$ is from p'. So the restriction of m is not necessarily in the same location in p'' and q''. In p'' we know that the restriction is on top-level, but in q'' the restriction can be inside one or more locations. The case is similar to the case above, where the hole is in $\mathcal{D}'_\gamma$ and δ' either comes from p or p'.

Case 4 Assume that $p'' \xrightarrow{\overline{\delta}} c$ (2) and $\Delta' \vdash \overline{\delta} : S''\ \mathtt{Ref}\ S'$ then we must show that there exists some c'' such that $q'' \xrightarrow{\overline{\delta}} c''$ and for all a with $\Delta'' \vdash a : S' \twoheadrightarrow \mathtt{aff}\ \tilde{n}''$ and $\Delta' \odot \Delta''$ defined we have $\Delta' \odot \Delta'' \vdash c \cdot a \equiv \mathcal{R} \equiv c'' \cdot a : \mathtt{aff}\ \tilde{n}' \cup \tilde{n}''$.

Again from (2) and Proposition 52 we have that $p'' \equiv (\tilde{n}''')(\overline{\delta}c' \parallel p''')$ and $c \equiv (\tilde{n}''')(c' \parallel p''')$ and $\tilde{n}''' \cap \delta = \emptyset$, so $\mathcal{C}_{(\Delta,\mathtt{aff}\ \tilde{n})}((m)(p'[^{p:\tilde{m}m}/_x])) \equiv (\tilde{n}''')(\overline{\delta}c' \parallel p''')$. Again we consider all the possible cases.

- Hole in p''' (similar to the same case in Case 3).
- Hole in c'. There are two sub-cases, either the hole is in the process to be output or in the residual process of the concretion, we only present the first case, as the second case is simpler. Let c' be $\langle \mathcal{C}'_{(\Delta,\mathtt{aff}\ \tilde{n})}((m)(p'[^{p:\tilde{m}m}/_x])) : \tilde{n}''''\rangle q'''$ for some $\mathcal{C}'$ and q''', so we know that $q'' \equiv (\tilde{n}''')(\overline{\delta}\langle \mathcal{C}'_{(\Delta,\mathtt{aff}\ \tilde{n})}(q'[^{(m)q:\tilde{m}}/_x]) : \tilde{n}''''\rangle q''' \parallel p''')$. By Proposition 57 we know that there exists some c'' such that $q'' \xrightarrow{\overline{\delta}} c''$ and $c'' \equiv (\tilde{n}''')(\langle \mathcal{C}'_{(\Delta,\mathtt{aff}\ \tilde{n})}(q'[^{(m)q:\tilde{m}}/_x]) : \tilde{n}''''\rangle q''' \parallel p''')$, hence it remains to prove that $\Delta' \odot \Delta'' \vdash c \cdot a \equiv \mathcal{R} \equiv c'' \cdot a : \mathtt{aff}\ \tilde{n}' \cup \tilde{n}''$.

  Without loss of generality we take $a = (x')p''''$ and we know that x' is an affine variable. By the definition of application and our shorthands, we have that the same local names are lifted in both sides, and that the residual process in c and c'' are the same. Furthermore, the body of the abstraction a (after the substitution) is the same down to the (possible) location of x'. All of this implies that the processes are the same down to the (possible) location of x', i.e. the processes have the form

  $$(\tilde{n}_1)((\tilde{n}_2)(q''' \parallel p''') \parallel p''''[^{r:\tilde{n}''''}/_{x'}])\ ,$$

  where $\tilde{n}_1 = \tilde{n}''' \cap \tilde{n}''''$ and $\tilde{n}''' = \tilde{n}_1\tilde{n}_2$ (i.e. $\tilde{n}_1$ is the local names that are lifted) and r is either $\mathcal{C}'_{(\Delta,\mathtt{aff}\ \tilde{n})}((m)(p'[^{p:\tilde{m}m}/_x]))$ or $\mathcal{C}'_{(\Delta,\mathtt{aff}\ \tilde{n})}(q'[^{(m)q:\tilde{m}}/_x])$. We consider the two cases for how the abstraction handles the input process (whether x' occurs free in p'''' or not).

  - If the abstraction does not use the variable x' then clearly the resulting processes are in the relation (the two processes are equivalent up-to $\equiv$ and hence contained in the relation).
  - If the abstraction does use the variable x' then again the processes are in the relation, since x' is affine we can construct a context

    $$\mathcal{C}''_{(\Delta,\mathtt{aff}\ \tilde{n})} =_{def} (\tilde{n}_1)((\tilde{n}_2)(q''' \parallel p''') \parallel \mathcal{C}'''_{(\Delta,\mathtt{aff}\ \tilde{n})})\ ,$$

    where $\mathcal{C}'''_{(\Delta,\mathtt{aff}\ \tilde{n})}$ is the body of the abstraction containing the context $\mathcal{C}'_{(\Delta,\mathtt{aff}\ \tilde{n})}$ in place of the variable x'. Note that the hole in the context cannot occur under replication, since the variable x' is affine, satisfying the requirement of the context in the relation.
- The only remaining cases are where $\overline{\delta}$ comes from either p or p'.
  - $\overline{\delta}$ comes from p. Then we know that x occurs unrestricted in p' (and without loss of generality we assume that $m \in \tilde{n}'''$), hence x occurs unrestricted in q', so we can rewrite q'' to exactly the same form as p'' up-to m-annotations on affine locations. So $q'' \equiv (\tilde{n}''')(\overline{\delta}c''' \parallel p'''')$, where $c' >^*_m c'''$ and $p''' >^*_m p''''$. So from Proposition 57 we know that there exists some c'' such that $q'' \xrightarrow{\overline{\delta}} c''$ and $c'' \equiv (\tilde{n}''')(c''' \parallel p'''')$. Again we need to prove that $\Delta' \odot \Delta'' \vdash c \cdot a \equiv \mathcal{R} \equiv c'' \cdot a : \mathtt{aff}\ \tilde{n}' \cup \tilde{n}''$.

    There are three cases to consider: either m is not lifted in c (and hence not in c''), m is lifted in both, m is lifted in c, but not in c''. We only present the last case as the other two cases are simpler.

Since m is lifted in c, we know from the definition of application that m is placed on top-level in $c \cdot a$. Letting $\Delta''' = \Delta' \odot \Delta''$ and $\tilde{n}'''' = \tilde{n}' \cup \tilde{n}''$, we take the empty context $\mathcal{C}'_{(\Delta''', \mathtt{aff}\ \tilde{n}'''')} =_{def} (-)_{(\Delta''', \mathtt{aff}\ \tilde{n}'''')}$ in order to place m on top-level. We will let the processes that we substitute in (i.e. p and $(m)q$ in the relation) match the entire residual of the concretion (satisfying the requirement $p >^*_m q$ of p and q in the relation).

Since m is lifted in c, but not in c'' we know that the process output from the concretion is affine, as only affine locations can differ on their annotation in c and c'' (and only on the name m). So only affine locations in the abstraction will be updated with m (satisfying the requirement $q' = p' \setminus \{m\}$ of p' and q' in the relation). So the processes are still in the relation.

– $\overline{\delta}$ comes from p'. Then c' can either come from p, p', or from both.

  * c' comes entirely from p'. So the hole of $\mathcal{C}_{(\Delta, \mathtt{aff}\ \tilde{n})}$ must occur unguarded and on top-level (and without loss of generality we assume that $m \in \tilde{n}'''$). Hence $q'' \equiv (\tilde{n}''' \setminus m)(\overline{\delta} c''' \parallel p'''')$ where $c''' = c' \setminus \{m\}$ and $p''' \equiv r \parallel r'[^{p:\tilde{m}m}/_x]$ and $p'''' \equiv r \parallel r''[^{(m)q:\tilde{m}}/_x]$ with $r'' = r' \setminus \{m\}$ (as p''' and p'''' are built partly from the context and partly from $p'[^{p:\tilde{m}m}/_x]$ and $q'[^{(m)q:\tilde{m}}/_x]$). By Proposition 57 we have that there exists some c'' such that $q'' \xrightarrow{\overline{\delta}} c''$ and $c'' \equiv (\tilde{n}''' \setminus m)(c''' \parallel p'''')$.
    Then there are two sub-cases depending on whether m is lifted in c. The first sub-case (m is not lifted) is similar to the previous sub-case above where m is not lifted in c and hence not in c''. The second sub-case (m is lifted, but only in c and not in c'', as c'' does not contain m-annotations) is also similar to the case that we presented above.
  * c' comes from both p' and p. p can either be in the output process or in the residual process. We only present the first case, as the other case is very similar to previous cases. Without loss of generality we assume that $m \in \tilde{n}'''$. So we have that $c' \equiv \langle r[^{p:\tilde{m}m}/_x] : \tilde{n}''''\rangle r''$ and $q'' \equiv (\tilde{n}''' \setminus m)(\overline{\delta} c''' \parallel p'''')$ where $c''' \equiv \langle r'[^{(m)q:\tilde{m}}/_x] : \tilde{n}'''' \setminus m\rangle r'''$ and $p'''' = p''' \setminus \{m\}$, $r' = r \setminus \{m\}$, $r''' = r'' \setminus \{m\}$.
    By Proposition 57 we have that there exists some c'' such that $q'' \xrightarrow{\overline{\delta}} c''$ and $c'' \equiv (\tilde{n}''' \setminus m)(c''' \parallel p'''')$. We need to prove that $\Delta' \odot \Delta'' \vdash c \cdot a \equiv \mathcal{R} \equiv c' \cdot a : \mathtt{aff}\ \tilde{n}' \cup \tilde{n}''$.
    Again we have two sub-cases depending on m is lifted (note that in both cases the process output is affine, as p is affine, and the restriction of m in c'' is inside the output process, and hence cannot be lifted by c'').
    · m is lifted. So by the definition of application we will place m on top-level in $c \cdot a$, and the sub-case proceeds as the case above.
    · m is not lifted. This implies that c'' does not contain m at all, so the restriction of m in c'' does not bind any occurrences of m, so the output processes will be equivalent except for the "useless" restriction of m in the output process of c'' (in c the restriction will be in the residual process, as it is not lifted).
    So the two applications $c \cdot a$ and $c'' \cdot a$ will (possibly) place the restriction of m in two different places, but since the restriction in c'' does not bind any occurrences we can use $\equiv$ to remove the restriction. Hence the processes are in the relation, if we use a substitution that simply discards the process to be substitute in (and using a context to capture all of the process surrounding the restriction of m in the residual of c).