Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 194 (2007) 23–38
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Modelling the Security of Smart Cards by Hard and Soft Types for Higher-Order Mobile Embedded Resources [1]

Mikkel Bundgaard and Thomas Hildebrandt[2]
Jens Chr. Godskesen[3]

IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark
{mikkelbu, hilde, jcg}@itu.dk

Abstract

We provide a type system inspired by affine intuitionistic logic for the calculus of Higher-Order Mobile Embedded Resources (Homer), resulting in the first process calculus combining affine linear (non-copyable) and non-linear (copyable) higher-order mobile processes, nested locations, and local names. The type system guarantees that linear resources are neither copied nor embedded in non-linear resources during computation.
We exemplify the use of the calculus by modelling a simplistic 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. A purely linear calculus would not be able to express that embedded software processes may be copied. Conversely, a purely non-linear calculus would not be able to express that mobile hardware processes cannot be copied.

Keywords: Higher-order process passing, linear types, copyable and non-copyable resources, nested locations, security, smart cards

1 Introduction

Following the seminal work on Mobile Ambients [10], several process calculi, including variations of Mobile Ambients [18,4], the Seal calculus [11], and the Homer calculus [15], 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* (i.e. non-copyable devices such as laptops, PDAs, and

[1] This research is supported by the Danish Research Agency grants no: 274-06-0415 (CosmoBiz), no: 272-05-0258 (Mobile Security) and no: 2059-03-0031 (BPL).

[2] Programming, Logic and Semantics Group.

[3] Computational Logic and Algorithms Group.

doi:10.1016/j.entcs.2007.09.011

smart cards), as well as mobile *computations* (i.e. copyable software agents and migrating processes). Nested explicit locations are typically used to represent administrative domains, firewalls, physical boundaries of mobile devices, boundaries of software messages and processes such as encryption, 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.

In the present paper we argue that mobile computing hardware devices are intrinsically *linear*, while mobile computations in software are intrinsically *non-linear*: A hardware device cannot easily be copied, and the security of a system often depends on this fact, for instance for smart cards. Contrarily, software must usually be explicitly protected against copying, e.g. by enclosing it in a tamper-proof (and non-copyable) hardware device.

The Mobile Ambients calculus and its descendants combine linear mobile processes (the ambients) and non-linear, copyable messages (values). These features make the calculi suitable for modelling mobile computing. But since none of the calculi allow general duplication of processes inside ambients, it becomes difficult to represent copyable, mobile computations. On the other hand, the more recent higher-order process calculi, such as the Seal calculus [11] and Homer [15], 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.

Somewhat surprisingly, we found no calculus combining linear and non-linear mobile embedded processes and local names. In the present paper we thus present an extension, inspired by affine intuitionistic linear logic, of the type and effect system for Homer presented in [13]. The extension allows us to distinguish between affine linear and non-linear uses of variables (as in the linear lambda calculus) and to type the names of locations (akin to reference types) and thereby to restrict the content of locations to be either linear or non-linear. We define non-linear to be a subtype of affine linear, which enable non-linear (software) locations to be embedded in linear (hardware) locations. This also ensures 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: If a linear resource could be embedded within a non-linear resource it could be copied by copying the embedding resource. In Homer it is possible to reference nested resources using composite location paths. To type these paths we introduce composite reference types guaranteeing that linear locations within non-linear locations are never referenced.

We claim that the calculus captures the intrinsic copyability features of mobile computing hardware devices and software processes as outlined above. We justify this claim by giving in Sec. 5 an example of a simplistic e-cash system, the security of which depends on the non-copyability of smart cards and the ATM itself. Dually, the copyability of software processes, in this case encrypted messages, constitutes an important security threat. We do not claim that the example prove the security of a realistic smart card system, but that it shows that a realistic model for both mobile computing and computations should allow for both linear and non-linear mobile processes.

The type system has consequences for the treatment of infinite behaviours. In most (untyped) higher-order calculi (HOπ [19], λ-calculus, CHOCS [23], Homer [15]) 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 [15] 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.

In the full paper [7], we extend the work in [15] to the linear and non-linearly typed calculus by providing a barbed bisimulation congruence, weak and strong labelled bisimulations, and prove that Howe's method extends to this richer typed setting and thus provides a technique for contextual reasoning about linear and non-linear mobile embedded resources. We show that the labelled bisimulation congruences are sound with respect to the barbed bisimulation congruences, and also complete in the strong case.

The structure of the paper is as follows. In Sec. 2 we present the Homer calculus, and in Sec. 3 we give it transition semantics. The type system for Homer is presented in Sec. 4, and we provide the Smart Card example in Sec. 5. In Sec. 6 we conclude and propose future work.

Related work

The interplay between linearity and non-linearity has been studied thoroughly in variations of Intuitionistic Linear Logic 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 [24], in recent languages for quantum computing with classical control [22], and for controlling the use of names (and thus mobility) in the π-calculus [17]. We found no prior studies of linear and non-linear mobile processes combined with nested locations and local names.

The Homer calculus extends Plain CHOCS, but shares ideas with recent calculi for mobility such as the Seal calculus [11], the M-calculus [20] (and its recent successor the Kell calculus [21]), 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 [3]) which ensure unity of location names (used for deterministic routing). A type system for Seal calculus is presented in [11], 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 [9]. In [8] 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 a composite channel is also suggested, but the idea is not pursued.

Linear types have been studied in great detail in the π-calculus [17,16] by Kobayashi et al. Recently Berger, Honda, and Yoshida [1,2] have investigated the connection between sequential functional computation and typed π-calculus. For a higher-order π-calculus Yoshida and Hennessy [27,26] 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 [12] for the calculus of Mobile Resources [14], the predecessor of Homer, but the theory was never developed. Homer was originally presented in [15], together with an adaptation of Howe's method to prove that late contextual bisimulation is a sound characterisation of barbed bisimulation congruence. In [13] the results were extended to prove that input-early strong bisimulation congruence is both a sound and complete characterisation of barbed bisimulation congruence. Homer has also been examined in the setting of bigraphs [5] and its expressivity have been studied in an encoding of the π-calculus [6].

2 Homer

In this section we present the syntax of Homer. The only difference from [13] 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, $\mathbf{a}$ of *abstractions* ranged over by a, and $\mathbf{c}$ of *concretions* ranged over by c are defined by the grammar:

$$p ::= \mathbf{0} \mid x \mid \varphi e \mid p \parallel p' \mid (n)p \mid !p \ , \qquad e ::= a \mid b \ ,$$

$$a ::= (x)p \ , \qquad b ::= \langle p' : \tilde{n} \rangle p \ , \qquad c ::= b \mid (n)c \ ,$$

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. Intuitively, this set of names can be thought of as the names *allocated* by the resource. The annotation is used to control dynamic scope extension when a resource is moved. The annotation is needed because one can define a context that tests if a name is free in a mobile resource. Without the annotations any two processes that do not contain the same names during their computation would be distinguishable (for a full description of this problem and its solution see [13,15]). The type system presented in Sec. 4 guarantees that this set contains all names appearing in the process p'.

The process constructors are the standard constructors from concurrent process calculi: the inactive process, $\mathbf{0}$, process variables, x, action prefixing, φe, parallel composition, $\|$, restriction of the name n in p, $(n)p$, and replication of p, $!p$. Homer is defined as a simple extension of the higher-order calculus Plain CHOCS [23] 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 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 reductions (ignoring the type annotations)

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

and

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

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 ($\mathit{fn}(t)$ and $\mathit{bn}(t)$) and variables of t ($\mathit{fv}(t)$ and $\mathit{bv}(t)$) as standard with the sole exception that $\mathit{fn}(\langle p' : \tilde{n}\rangle p) = \tilde{n} \cup \mathit{fn}(p)$, ie. 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, 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 will say that a relation R on processes is a *congruence* if $\mathbf{0}\ R\ \mathbf{0}$ and $x\ R\ x$, and $p\ R\ p'$ and $q\ R\ q'$ implies $p \parallel q\ R\ p' \parallel q'$, $(n)p\ R\ (n)p'$, $\varphi(x)p\ R\ \varphi(x)p'$, and $\varphi\langle q : \tilde{n}\rangle p\ R\ \varphi\langle q' : \tilde{n}\rangle p'$. We then define *structural congruence* $\equiv$ as the least relation on $\mathbf{p}_{/\alpha}$ that is a congruence and that satisfies the (usual) monoid rules for $(\|, \mathbf{0})$ and scope extension as for the π-calculus.

We define the application between an abstraction and a concretion as usual, except that the substitution updates the type annotation in locations.

Definition 2.1 (application and substitution) *For a concretion $c = (\tilde{m})\langle p : \tilde{n}\rangle p'$ and an abstraction $a = (x)p''$, where $\tilde{m} \cap fn(p'') = \emptyset$, we define their application by*

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

where the capture free substitution $p''[^{p:\tilde{n}}/_x]$ is defined inductively in the structure of p'' as usual, except that in the case for concretions, the annotation of the sub-resource is updated, if the variable appears free in the sub-resource. That is, if $x \in fv(q)$ then $(\langle q : \tilde{m}'\rangle q')[^{p:\tilde{n}}/_x] = \langle q[^{p:\tilde{n}}/_x] : \tilde{m}' \cup \tilde{n}\rangle q'[^{p:\tilde{n}}/_x]$. If $x \notin fv(q)$ then $(\langle q : \tilde{m}'\rangle q')[^{p:\tilde{n}}/_x] = \langle q : \tilde{m}'\rangle q'[^{p:\tilde{n}}/_x]$.

Note that the substitution discards the type when a variable is reached (see Appendix A for the full definition of application and substitution).

3 Transition semantics

In this section we provide Homer with a labelled transition semantics. As in the previous section, the only difference from [13] is that we have extended the semantics with rules for replication.

We let π range over the set Π of labels, 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 $\pi = \overline{\delta}$ and $\emptyset$ otherwise. The rules in Table 1 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.

To allow for a more succinct presentation of the transitions of nested active resources we close concretions and abstractions under process operators. 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 $(n)c$ denote $(n\tilde{n})\langle p_1 : \tilde{n}_1\rangle p$, if $n \in \tilde{n}_1$ and otherwise it denotes $(\tilde{n})\langle p_1 : \tilde{n}_1\rangle(n)p$, and we let $\delta\langle c : \tilde{n}'\rangle p'$ denote $(\tilde{n})\langle p_1 : \tilde{n}_1\rangle\delta\langle p : \tilde{n}'\tilde{n}\rangle p'$. 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 in the rules (*nesting*), (*rest*), (*par*), (*par′*) and (*repl1*).

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

Table 1
Transition rules.

$$(\mathit{prefix})\ \frac{}{\varphi e \xrightarrow{\varphi} e} \qquad (\mathit{nesting})\ \frac{p \xrightarrow{\pi} t}{\delta\langle p : \tilde{n}\rangle p' \xrightarrow{\delta \bullet \pi} \delta\langle t : \tilde{n}\rangle p'}$$

$$(\mathit{rest})\ \frac{p \xrightarrow{\pi} t}{(n)p \xrightarrow{\pi} (n)t}(n \notin \mathit{fn}(\pi)) \qquad (\mathit{sync})\ \frac{p \xrightarrow{\varphi} e \quad p' \xrightarrow{\overline{\varphi}} \bar{e}}{p \parallel p' \xrightarrow{\tau} e \cdot \bar{e}}$$

$$(\mathit{par})\ \frac{p \xrightarrow{\pi} t}{p \parallel p' \xrightarrow{\pi} t \parallel p'} \qquad (\mathit{par'})\ \frac{p' \xrightarrow{\pi} t}{p \parallel p' \xrightarrow{\pi} p \parallel t}$$

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

thus not visible outside the resource. Since $\delta \bullet \tau = \tau$, 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 reduction (1) in the previous section 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 Def. 2.1 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 $x \in \mathit{fv}(q)$).

Similarly for the reduction (2) we have that $m\langle p : \tilde{n}'\rangle p' \xrightarrow{m} \langle p : \tilde{n}'\rangle p'$, so

$$n\langle m\langle p : \tilde{n}'\rangle p' : \tilde{n}''\rangle \xrightarrow{nm} \langle p : \tilde{n}'\rangle n\langle p' : \tilde{n}''\rangle \ .$$

Combining this transition 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 Def. 2.1 is the process $n\langle p' : \tilde{n}''\rangle \parallel q[{}^{p:\tilde{n}'}/_{x}]$.

4 Type system

We are now ready to present the extension of the type and effect system given for Homer in [13] to allow a distinction between affine linear and non-linear resources.

We will assume a set $\mathcal{S} = \{\texttt{aff}, \texttt{un}\}$, 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 $\texttt{un} < \texttt{aff}$, 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 the e-cash system in Sec. 5 below, 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 [13] and can be regarded as an effect that captures the names used or allocated by the process, as described in Sec. 2. 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* $S \multimap S'\ \tilde{n}$ types an abstraction $(x)p$ that itself has sort S' and effect $\tilde{n}$ and accepts a process of sort S. We will only consider abstraction and concretion types where $S \leq S'$, and this is ensured by the typing rules.

Definition 4.1 (types) *We define three kinds of types,* process types T_p*,* concretion types T_c*, and* abstraction types T_a*, by the following grammar*

$$T ::= T_p \mid T_c \mid T_a$$

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

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 $\langle S\rangle S'\ \tilde{n}n$. We write $T \cup \tilde{n}''$ for the (not necessarily disjoint) name extension of the type T defined by

$$\begin{aligned}(S\ \tilde{n}) \cup \tilde{n}'' &= S\ \tilde{n} \cup \tilde{n}''\\ (\langle S\rangle T_p) \cup \tilde{n}'' &= \langle S\rangle T_p \cup \tilde{n}''\\ (S \multimap T_p) \cup \tilde{n}'' &= S \multimap T_p \cup \tilde{n}''\ .\end{aligned}$$

Type environments Γ assign sorts to names and variables.

Definition 4.2 (type environment) *A type environment* Γ *is a 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 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 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. Letting l range over both names and variables, we define the partial combination Γ'' of two type environments Γ and Γ', denoted $\Gamma \odot \Gamma' = \Gamma''$, by $\Gamma \cup \Gamma'$ if $\{x \mid \Gamma(x) = \mathtt{aff}\} \cap \{x' \mid \Gamma'(x') = \mathtt{aff}\} = \emptyset$, and if $l \in dom(\Gamma) \cap dom(\Gamma')$ implies $\Gamma(l) = \Gamma'(l)$, otherwise the combination is undefined. The requirements 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 [17].

We also need typing of *address paths*: $\Gamma \vdash \varphi : S\ \mathtt{Ref}\ S'$, as defined by the rules

Table 2
Typing address paths.

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

Table 3
Typing rules for affine linear and non-linear Homer

$$(\mathit{variable})\ \frac{}{\Gamma, x : S \vdash x : S\ \tilde{n}}(\tilde{n} \subseteq \mathit{dom}_n(\Gamma)) \qquad (\mathit{inactive})\ \frac{}{\Gamma \vdash \mathbf{0} : \texttt{un}\ \tilde{n}}(\tilde{n} \subseteq \mathit{dom}_n(\Gamma))$$

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

$$(\mathit{rest\text{-}conc})\ \frac{\Gamma, n : S \vdash (\tilde{m})\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c n}{\Gamma \vdash (\tilde{m}n)\langle p : \tilde{m}n\tilde{n}\rangle p' : T_c} \qquad (\mathit{embed})\ \frac{\Gamma \vdash p : \texttt{un}\ \tilde{n}}{\Gamma \vdash p : \texttt{aff}\ \tilde{n}}$$

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

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

$$(\mathit{pre\text{-}abs})\ \frac{\Gamma \vdash a : S' \rightarrowtriangle S\ \tilde{n} \quad \Gamma \vdash \varphi : S'' \;\texttt{Ref}\; S'}{\Gamma \vdash \varphi a : S\ \tilde{n} \cup \varphi}(S'' \leq S)$$

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

in Table 2. The type $S\ \texttt{Ref}\ S'$ is read as *a reference via* S *to* S'. The rules ensure that the sorts of the names in an address path typed $S\ \texttt{Ref}\ S'$ 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 $\Gamma = m : \texttt{aff}, n : \texttt{un}$, we can derive $\Gamma \vdash mm : \texttt{aff Ref aff}$ and $\Gamma \vdash mmn : \texttt{aff Ref un}$, but we cannot derive neither $\Gamma \vdash nm : \texttt{un Ref aff}$ nor $\Gamma \vdash mnm : \texttt{aff Ref aff}$.

We define the typing of processes, abstractions, and concretions using the rules in Table 3. The type system conservatively generalises the prior type (effect) system for Homer [13], which we can obtain by removing the (*embed*) rule and 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 (*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 rule (*embed*) corresponds to the usual subsumption rule in type systems with subtyping, concretely it allows us to treat unrestricted processes as affine processes. 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 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 of the address path in the path type is used in the side-conditions of the rules (*pre-conc*) and (*pre-abs*) to ensure that any process using a 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 have implemented a typing algorithm by eliminating the rule (*embed*) and following the approach for linear type systems [25]. The typing algorithm requires that we annotate name restriction with a sort, as we cannot infer the correct sort from the restriction. See the full paper for this algorithm [7].

We can prove the standard properties about the type system: strengthening of unused names and variables, invariance under structural congruence etc. Again, we refer to the full paper for these results [7] and only present the main results here. As expected in a type system with subtyping we have narrowing of variables.

Proposition 4.3 (narrowing of variables) *If* $\Gamma, x : S \vdash t : T$ *and* $S' \leq S$ *then* $\Gamma, x : S' \vdash t : T$.

Note that we in general cannot use narrowing (or widening) for 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.

Lemma 4.4 (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.*

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 4.5 (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 $S\ \tilde{n}$, $S' \mathrel{-\!\!\triangleright} S\ \tilde{n}$, or $\langle S'\rangle S\ \tilde{n}$. If $\Gamma \vdash t : T$ then*

- *$fn(t) \subseteq n(T) \subseteq dom_n(\Gamma)$ and $fv(t) \subseteq dom_v(\Gamma)$.*
- *If $x : \mathtt{aff} \in \Gamma$ then x occurs free at most once in t.*
- *If $x : \mathtt{aff} \in \Gamma$ and $x \in fv(t)$ then $s(T) = \mathtt{aff}$.*
- *If $s(T) = \mathtt{un}$ then for every sub-derivation $\Gamma' \vdash t' : T'$ we have $s(T') = \mathtt{un}$.*

Theorem 4.6 (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 $\Gamma \vdash a : S' \mathrel{-\!\!\triangleright} S\ \tilde{n}$ and $\Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'$ for some S' and $S'' \leq S$.*
- *$\pi = \varphi$, $t = c$ and $\Gamma \vdash c : \langle S'\rangle S\ \tilde{n}$ and $\Gamma \vdash \varphi : S''\ \mathtt{Ref}\ S'$ for some S' and $S'' \leq S$.*

5 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 a process (or message) communicated on the public channel e. 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:

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

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. Hence we get the following sequence of transitions

$$\begin{aligned} atm \xrightarrow{\tau}\equiv\ & (k)\big((n')(v\langle e\langle k\langle n'\langle \mathbf{0} : \emptyset\rangle : \{n'\}\rangle : \{k,n'\}\rangle : \{e,k,n'\}\rangle \parallel \\ & \qquad \overline{vekn'}(x)\overline{c}\langle \mathbf{0} : \emptyset\rangle) \parallel cash_k\big) \\ \xrightarrow{\tau}\equiv\ & (k)\,(p \parallel cash_k) \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle \ , \end{aligned}$$

where $p \ =_{\mathit{def}} \ (n')\,(v\langle e\langle k\langle \mathbf{0} : \{n'\}\rangle : \{k, n'\}\rangle : \{e, k, n'\}\rangle)$ is a slot containing a ”used” smart card, i.e. where the nonce has been removed.

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

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

which picks up the e-cash process by $\overline{v}(x)$ and creates two copies. Then (again letting $p \ =_{\mathit{def}} \ (n')\,(v\langle e\langle k\langle \mathbf{0} : \{n'\}\rangle : \{k, n'\}\rangle : \{e, k, n'\}\rangle)))$ security will break down

$$\begin{aligned} \mathit{atm} \parallel \mathit{thief} \ \xrightarrow{\tau}\equiv \quad & (k)\big(v\langle \mathit{crypt}_{e,k} : \{e, k\}\rangle \parallel v\langle \mathit{crypt}_{e,k} : \{e, k\}\rangle \parallel \mathit{cash}_k\big) \\ \xrightarrow{\tau}{}^{*}\equiv \quad & (k)\,(p \parallel p \parallel \mathit{cash}_k) \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle \ \ . \end{aligned}$$

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

We then show, that we cannot type the system $\mathit{atm} \parallel \mathit{thief}$, if the slot v is linear, as this makes it impossible to copy the content of the slot, i.e. the smart card.

Proposition 5.2 *For any* $\Delta, v : \mathtt{aff}$*,* $\tilde{n}$ *and sort* S *it is not possible to derive* $\Delta, v : \mathtt{aff} \vdash \mathit{atm} \parallel \mathit{thief} : S\ \tilde{n}$*.*

Proof (Sketch) Assume that it is possible to derive $\Delta, v : \mathtt{aff} \vdash \mathit{atm} \parallel \mathit{thief} : S\ \tilde{n}$, by inspecting the derivation, and without loss of generality, is must 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 4.5 (that x occurs free at most once). □

Note that the encrypted nonce is unrestricted. The security would be broken, if we repeatedly had used the same secret name n as challenge for the card, i.e. swapping the local name (n) and the replication in the definition of the control process, defining cash_k as $(n)!\big(\overline{ve}\langle n\langle \mathbf{0} : \emptyset\rangle : \{n\}\rangle \overline{vekn}(x)\overline{c}\langle \mathbf{0} : \emptyset\rangle\big)$, A thief which interrupts the ATM just after the name n has been send (and encrypted at the card) and which copies the encrypted content of the card could be defined by

$$\mathit{thief}' = \overline{ve}(x)(v\langle e\langle x : \emptyset\rangle : \{e\}\rangle \parallel ve(x')v\langle e\langle x : \emptyset\rangle : \{e\}\rangle) \ \ ,$$

where the right-hand side of the parallel composition receives and discards the challenge message the second time it is send by the ATM, and provides a card with the copied encrypted content. Letting

$$\begin{aligned} p &=_{def} \ !\big(\overline{ve}\langle n\langle \mathbf{0} : \emptyset\rangle : \{n\}\rangle \overline{vekn}(x)\overline{c}\langle \mathbf{0} : \emptyset\rangle\big)\ , \\ q &=_{def} \ v\langle e\langle k\langle \mathbf{0} : \{n\}\rangle : \{k,n\}\rangle : \{e,k,n\}\rangle,\ \text{and} \\ q' &=_{def} \ v\langle e\langle k\langle n\langle \mathbf{0} : \emptyset\rangle : \{n\}\rangle : \{k,n\}\rangle : \{e,k,n\}\rangle \end{aligned}$$

we have the following transitions

$$\begin{aligned} \mathit{atm} \parallel \mathit{thief}' \xrightarrow{\tau}{}^{*}\equiv &\ \{k,n\}\big(v\langle \mathbf{0} : \{e,k,n\}\rangle \parallel \overline{vekn}(x)\overline{c}\langle \mathbf{0} : \emptyset\rangle \parallel p \parallel q' \parallel ve(x')q'\big) \\ \xrightarrow{\tau}{}^{*}\equiv &\ \{k,n\}\big(v\langle \mathbf{0} : \{e,k,n\}\rangle \parallel p \parallel q \parallel q\big) \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle \parallel \overline{c}\langle \mathbf{0} : \emptyset\rangle\ \ . \end{aligned}$$

This security threat would not show in a purely linear calculus. We leave for future work to apply the bisimulation congruence presented in [7] to prove that the typed *atm* is indeed secure in any context.

6 Conclusions and future work

We have successfully extended the prior type and effect system for Homer to provide the first process calculus combining affine linear and non-linear nested mobile embedded processes with local names. 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 Mobile Ambients and 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 the IT-University of Copenhagen.

References

[1] Berger, M., K. Honda and N. Yoshida, *Sequentiality and the π-calculus*, in: S. Abramsky, editor, *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications (TLCA'01)*, Lecture Notes in Computer Science **2044** (2001), pp. 29–45.

[2] Berger, M., K. Honda and N. Yoshida, *Genericity and the π-calculus*, Acta Informatica **42** (2005), pp. 83–141.

[3] Bidinger, P. and J.-B. Stefani, *The Kell calculus: Operational semantics and type system*, in: E. Najm, U. Nestmann and P. Stevens, editors, *Proceedings of the 5th IFIP International Conference on Formal Methods for Object-Based Distributed Systems (FMOODS'03)*, Lecture Notes in Computer Science **2884** (2003), pp. 109–123.

[4] Bugliesi, M., G. Castagna and S. Crafa, *Access control for mobile agents: The calculus of boxed ambients*, ACM Transactions on Programming Languages and Systems (TOPLAS) **26** (2004), pp. 57–124.

[5] Bundgaard, M. and T. Hildebrandt, *Bigraphical semantics of higher-order mobile embedded resources with local names*, in: A. Rensink, R. Heckel and B. König, editors, *Proceedings of the Graph Transformation for Verification and Concurrency workshop (GT-VC'05)*, Electronic Notes in Theoretical Computer Science **154** (2006), pp. 7–29.

[6] Bundgaard, M., T. Hildebrandt and J. C. Godskesen, *A CPS encoding of name-passing in higher-order mobile embedded resources*, Theoretical Computer Science **356** (2006), pp. 422–439.

[7] Bundgaard, M., T. Hildebrandt and J. C. Godskesen, *Typing linear and non-linear higher-order mobile embedded resources with local names*, Technical Report TR-2007-97, IT University of Copenhagen (2007), available from `http://www.itu.dk/~mikkelbu/typedHomer.pdf`.

[8] Carbone, M., "Trust and Mobility," Ph.D. thesis, BRICS (2005).

[9] Carbone, M. and S. Maffeis, *On the expressive power of polyadic synchronisation in π-calculus*, Nordic Journal of Computing **10** (2003), pp. 70–98.

[10] Cardelli, L. and A. D. Gordon, *Mobile ambients*, Theoretical Computer Science **240** (2000), pp. 177–213.

[11] Castagna, G., J. Vitek and F. Z. Nardelli, *The Seal calculus*, Journal of Information and Computation **201** (2005), pp. 1–54.

[12] Godskesen, J. C. and T. Hildebrandt, *Copyability types for mobile computing resources* (2004), presented at the International Workshop on Formal Methods and Security, Nanjing, China.

[13] Godskesen, J. C. and T. 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)*, Lecture Notes in Computer Science **3821** (2005), pp. 140–151.

[14] Godskesen, J. C., T. Hildebrandt and V. Sassone, *A calculus of mobile resources*, in: L. Brim, P. Jancar, M. Kretínský and A. Kucera, editors, *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, Lecture Notes in Computer Science **2421** (2002), pp. 272–287.

[15] Hildebrandt, T., J. C. Godskesen and M. Bundgaard, *Bisimulation congruences for Homer — a calculus of higher order mobile embedded resources*, Technical Report TR-2004-52, IT University of Copenhagen (2004).

[16] Kobayashi, N., *Type systems for concurrent programs* (2002), in *Proceedings of UNU/IIST 10th Anniversary Colloquium.*

[17] Kobayashi, N., B. C. Pierce and D. N. Turner, *Linearity and the pi-calculus*, ACM Transactions on Programming Languages and Systems (TOPLAS) **21** (1999), pp. 914–947.

[18] Levi, F. and D. Sangiorgi, *Mobile safe ambients*, ACM Transactions on Programming Languages and Systems (TOPLAS) **25** (2003), pp. 1–69.

[19] Sangiorgi, D., "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms," Ph.D. thesis, Department of Computer Science, University of Edinburgh (1992).

[20] Schmitt, A. and J.-B. 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)* (2003), pp. 50–61.

[21] Schmitt, A. and J.-B. Stefani, *The Kell calculus: A family of higher-order distributed process calculi*, in: C. Priami and P. Quaglia, editors, *Proceedings of the International Workshop on Global Computing Workshop (GC'04)*, Lecture Notes in Computer Science **3267** (2004), pp. 146–178.

[22] Selinger, P. and B. Valiron, *A lambda calculus for quantum computation with classical control*, Journal of Mathematical Structures in Computer Science **16** (2006), pp. 527–552.

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

[24] Turner, D. N. and P. Wadler, *Operational interpretations of linear logic*, Theoretical Computer Science **227** (1999), pp. 231–248.

[25] Walker, D., *Substructural type systems*, in: B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, MIT Press, 2004 pp. 3–43.

[26] Yoshida, N., *Channel dependent types for higher-order mobile processes (extended abstract)*, in: N. D. Jones and X. Leroy, editors, *Proceedings of the 31st ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL'04)* (2004), pp. 147–160.

[27] Yoshida, N. and M. Hennessy, *Assigning types to processes*, Journal of Information and Computation **174** (2004), pp. 143–179.

A Application and substitution

Definition A.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 $\tilde{m} \cap fn(p'') = \emptyset$*

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

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

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

B Results

We will write φ_i for the i'th element of the path φ, and *length*(φ) for the length of the path φ.

Proposition B.1 *$dom_n(\Gamma) \supseteq \varphi$ and $\forall i, j$ with $1 \leq i \leq j \leq length(\varphi)$ we have $\Gamma(\varphi_j) \leq \Gamma(\varphi_i)$ and $\Gamma(\varphi_1) = S$ and $\Gamma(\varphi_{length(\varphi)}) = S'$ iff $\Gamma \vdash \varphi : S\ \mathtt{Ref}\ S'$.*

We combine both weakening propositions in one, and let l range over names and variables.

Proposition B.2 (weakening) *If $\Gamma \vdash t : T$ and $l \notin dom(\Gamma)$ then $\Gamma, l : S \vdash t : T$.*

Proposition B.3 (strengthening, names) *Assume $n \notin fn(t)$ and $\Gamma, n : S \vdash t : T$ then $\Gamma \vdash t : T \setminus n$.*

Proposition B.4 (strengthening, variables) *Assume $x \notin fv(t)$ and $\Gamma, x : S \vdash t : T$ then $\Gamma \vdash t : T$.*

Proposition B.5 *If $\Gamma \vdash t : T$ and $n : S' \in \Gamma$ then $\Gamma \vdash t : T \cup n$.*

Proposition B.6 (structural congruence and typing) *If $f \equiv f'$ then $\Gamma \vdash t : T$ iff $\Gamma \vdash f' : T$.*

Proposition B.7 (well-typed application) *If* $\Gamma \vdash a : S'' \multimap S'\ \tilde{n}''$ *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.*