Mobility control via passports

Samuel Hym

Mobility control via passports

Samuel Hym

PPS, Université Paris Diderot (Paris 7) & CNRS

April 6, 2007

Abstract

Dπ is a simple distributed extension of the π-calculus in which agents are explicitly located, and may use an explicit migration construct to move between locations.

In this paper we introduce passports to control those migrations; in order to gain access to a location agents are now expected to show some credentials, granted by the destination location. Passports are tied to specific locations, from which migration is permitted. We describe a type system for these passports, which includes a novel use of dependent types, and prove that well-typing enforces the desired behaviour in migrating processes.

Passports allow locations to control incoming processes. This induces a major modification to the possible observations which can be made of agent-based systems. Using the type system we describe these observations, and use them to build a *loyal* notion of observational equivalence for this setting. Finally we provide a complete proof technique in the form of a bisimilarity for establishing equivalences between systems.

Keywords: control of agent migrations, distributed computation, process calculus, typed observational equivalence

1 Introduction

Dπ [HR02] is a process calculus designed to reason about distribution of computation. It is built as a simple extension of the π-calculus in which agents are explicitly located without nesting so that a system might look like:

$$l_1[\![c\,!\,\langle b\rangle\, P_1]\!] \mid l_2[\![P_2]\!] \mid (\mathsf{new}\, a : \mathsf{E})(l_3[\![P_3]\!] \mid l_1[\![c\,?\,(x : \mathsf{T})\, P_4]\!])$$

where the l_i are location names and the P_i are processes located in one of those locations. Here, P_1 and P_4 are placed in the same location l_1, even if they are scattered in the term. Channels also are distributed: one channel is anchored in exactly one location: two processes must be in the same location to communicate. In our example, the system can evolve into

$$l_1[\![P_1]\!] \mid l_2[\![P_2]\!] \mid (\mathsf{new}\, a : \mathsf{E})(l_3[\![P_3]\!] \mid l_1[\![P_4\{^b\!/\!_x\}]\!])$$

when P_1 and P_4 communicate. This makes Dπ a streamlined distributed version of the π-calculus, which allows to concentrate our attention on agent migrations.

Dπ agents can ask their migration from their current location, say k, to the location l via the primitive

$$\mathsf{goto}_p\, l$$

The p, added by the present work, is a *passport* which must match the actual migration attempted, from k to l. The intuition of those passports is really close to passwords, requested whenever trying to enter a location and therefore allowing that location to control which processes should be granted access.

Some other approaches to control migrations have been investigated in process calculi. In Ambients-related calculi, the migrations are particularly hard to control so many works tried to address this problem: in Safe Ambients ([LS00]), the destination location must grant access to incoming ambients by using a *co-capability*. These co-capabilities have been enriched in [MH06] with *passwords*: the password used to migrate is syntactically checked when the migration is to be granted. This idea of passwords was pursued in the NBA calculus ([BCMS02]) which combines it with another choice to control behaviours of ambients: communications across boundaries are allowed so that the troublesome open primitive from the original Mobile Ambients is not necessary to obtain an expressive calculus. This second approach was also used in different hierarchical calculi like Seal [CN02] or Kell [SS04].

In non-hierarchical calculi, we have a better handle over migrating behaviours so that more powerful techniques can be employed, for instance leveraging type systems. In [HMR03], access to a location is a capability tied to that location via its type. In [HRY05], access requires the knowledge of a *port* which also governs subsequent resource accesses by typing the migrating processes, using for this complex process types developed in particular accesses. This approach is strongly constraining processes and requires higher order actions. In [MV05], access to locations and resources is conditioned by policies based on the history of migrations of the agent.

The present work provides a first-order theory that could serve as a foundation for a fine-grained control of comparable power to [HRY05]. The only location of the history taken into account to grant access is the origin of the migrating process: we will define a simple setting in which it is possible to describe "trust sub-networks" such as an intranet. Furthermore, the origin of a process seems easier to assert realistically than its full history. The simplicity of this setting can also be seen in the choice of regular names for passports: thanks to this, the calculus is homogeneous.

In the present work, we associate rights to the names of the passports via typing: for instance, the type $l \mapsto k$ is attached to some passport granting access to k from l. The typing system will therefore have to include dependent types to describe the link between passports and the locations they are attached to. But this approach to tie rights to types provides type-based tools and techniques to reason about security properties.

In particular, we will investigate the notion of typed observational equivalence inherited from [BS98]. The founding intuition of observational equivalences is to distinguish two systems only when it is possible to observe a difference between them through a series of interactions. In a typed observational equivalence where types represents permissions, the *barbs* the observer is allowed to see are conditioned by the permissions he managed to get access to.

Since permissions are represented by types, a normal type environment is used to describe the observer's rights.

Control of migrations has a great impact on the set of possible observations: since all interactions are performed over located channels, permissions to access these locations, *i.e.* passports, are mandatory to observe anything if the observer abides by the rules. We will therefore introduce an intuitive typed congruence that takes into account the migration rights of such a *loyal* observer. As usual, the closure of the equivalence over all admissible contexts makes this equivalence intractable. So we will provide an alternative coinductive definition for this equivalence as a bisimilarity based on *actions* which identify the possible interactions between the system and its observer. This alternative definition reveals a difficulty arising from dependent types: as an artefact of dependencies, some name scopes must be opened even when the name itself is not revealed to the observer.

Outline The rest of this paper is structured as follows. The calculus modified with passports is presented in Section 2, in particular with its complex type system including subtyping order. Then an observational equivalence that characterises how systems can use passports to protect themselves from observers is defined in Section 3. A proof technique, in the form of a bisimilarity, is provided in Section 4 to alleviate the complexity of the observational equivalence defined in Section 3; we also sketch in this section why this proof technique is complete. Finally, we finish with some concluding remarks and perspectives.

2 Typed Dπ with passports

We present here a stripped-down version of the Dπ-calculus to focus on migration control. In particular, this version does not include recursive processes, that were thoroughly dealt with in [HH07]. An account of the complete calculus is presented in [Hym06]. Most of it is inherited from previous works, like [HR02], so we will insist mostly on the differences. Let us start by looking at passports on examples.

2.1 Overview of passports

Dπ describes distributed computation as explicitly located processes that may may migrate between locations. To allow those locations to control incoming processes, we devise a system in which they can deliver *passports* which will be required from processes trying to enter the location. So the construct $\mathsf{goto}\, k$ now becomes $\mathsf{goto}_p\, k$, where p is the actual passport invoked to authorise access to k.

The control we propose is finer that a mere control of the name of the passport an incoming process use: a location is also allowed to control that the origin of the incoming process corresponds to the passport. Passports can then be delivered for specific communications. In the case of a simple client-server situation, the client can deliver a passport only for the response coming from the server. It might be formalised as:

$$\begin{aligned} &cl[\![\mathsf{newpass}\, pass\, \mathsf{from}\, sv\, \mathsf{in} \\ &\qquad \mathsf{goto}_{p_{sv}}\, sv.\ req\,!\,\langle cl, (quest, res, pass)\rangle \mid \ldots res\,?\,(x)\, P]\!] \end{aligned} \tag{1}$$

where the client *cl* generates a passport *pass* specific to the server *sv* before going there (using the passport p_{sv}) and requesting some computation while waiting for the result on the channel *res* in *cl*. The corresponding server might look like:

$$sv[\![*req\,?\,(x_{cl},(x_{quest},x_{res},x_{pass}):\mathsf{T})\cdots\mathsf{goto}_{x_{pass}}\,x_{cl}.\;x_{res}\,!\,\langle r\rangle]\!] \qquad (2)$$

This example suggests a smooth enforcement of the origin control: a public server might want to accept every request, whatever the originating location. The passports corresponding to such a policy, like p_{sv} in the example, can be created by

$$\mathsf{newpass}\,p_{sv}\,\mathsf{from}\,\star$$

where $\star$ stands for *anywhere*.

Since passports allow a location to choose the locations it is accepting processes from, a location can thus express the trust it is putting in its surrounding locations. So we can set up a situation where some locations $l_1,\ldots,l_n$ form a sub-network and trust each other: to achieve this the location l_i would provide a passport p_{l_i} granting access to every process coming from one of the locations l_j but it would deny access to any process coming from another location. It would look like:

$$N = (\mathsf{new}\,p_{l_1}:l_2,\ldots,l_n\mapsto l_1)\cdots(\mathsf{new}\,p_{l_n}:l_1,\ldots,l_{n-1}\mapsto l_n)\,l_1[\![P_1]\!]\,|\cdots|\,l_n[\![P_n]\!]$$

where every process coming from any other location than one of the l_j would be simply denied access if it tried to use a passport p_{l_i}. This is indicated by the *type annotation*, for instance $l_2,\ldots,l_n\mapsto l_1$ for the passport p_{l_1}.

Building on this example, we can add a gateway location *gw* through which every process coming from outside would have to go:

$$(\mathsf{new}\,p^{gw}_{l_i}:gw\mapsto l_i)\cdots N\,|\,gw[\![P_{gw}]\!]$$

where the optional passports $p^{gw}_{l_i}$ would define the access policy of the inner locations for the processes which are incoming from the gateway. For instance, when modelling a firewalled sub-network, the passports $p^{gw}_{l_i}$ would be revealed only to processes originally created inside l_i: these processes would naturally be allowed to enter back into the sub-network to bring the result of their investigation.

Let us now describe formally the calculus with passports.

2.2 Syntax & semantics

Processes are described using *names* (usually written $a,b,\ldots$, reserving c,d for *channel* names, k,l for *locations* and p,q for *passports*) and *variables* (usually written $x,y,\ldots$). When both names and variables can be used, we will talk of *identifiers* and write them $u,v,\ldots$. We will write $\tilde{u}$ for a set of identifiers and $\vec{u}$ for a tuple. We will also write $\tilde{u}^\star$ when either $\tilde{u}$ or $\star$ is expected. Finally, we will use capital letters when tuples are allowed so V can represent $(v_1,(v_2,v_3),v_4)$ or any other *value*, composed of identifiers and X any *pattern*, composed of variables.

The syntax of Dπ is given in Figure 1. Our contributions are:

Figure 1 Syntax for the Dπ-calculus

M ::=	*Systems*
$l[\![P]\!]$	Located process
$M_1 \mid M_2$	Parallel composition
$(\mathsf{new}\, a : \mathsf{E})\, M$	Name scope
$\mathbf{0}$	Inactive system

P ::=	*Processes*
$u\,!\,\langle V \rangle\, P$	Writing on channel
$u\,?\,(X : \mathsf{T})\, P$	Reading on channel
$\mathsf{if}\ u_1 = u_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$	Condition
$\mathsf{goto}_v\, u.\ P$	Migration
$\mathsf{newchan}\, c : \mathsf{C}\ \mathsf{in}\ P$	Channel generation
$\mathsf{newloc}\, l, (\vec{c}), (\vec{p}), (\vec{q}) : \mathsf{L}\ \mathsf{with}\ P_l\ \mathsf{in}\ P$	Location generation
$\mathsf{newpass}\, p\ \mathsf{from}\ \tilde{u}^{\star}\ \mathsf{in}\ P$	Passport generation
$P_1 \mid P_2$	Parallel composition
$*P$	Replication
stop	Termination

- The migration construct $\mathsf{goto}_v\, u$ now mentions the *passport v* to get access to the location u.

- The new construct to generate passports, $\mathsf{newpass}$, provides two kinds of origin control:

  – passports that allow migration from a given set of originating locations $\tilde{u}$ are created by $\mathsf{newpass}\, p\, \mathsf{from}\, \tilde{u}$; thus a location can express its trust in the sub-network $\tilde{u}$: every process from any location in $\tilde{u}$ will be granted access;

  – *universal passports*, that allow migration from any location (for instance to describe the behaviour of a public server) are created by $\mathsf{newpass}\, p\, \mathsf{from}\, \star$.

  Of course, the location a passport grants access to is the location where the passport is generated: that is the only way to allow locations to control incoming processes.

- The construct to generate new locations, newloc, is enriched: passports to access the new location (*child*) or the location where the construct is called (*mother*) can be generated on the fly. This solves a potential deadlock situation: if passports to access the child were always created from inside the child itself, some passports granting access from the child would be needed to export them...

Let us consider now the semantics associated with the calculus, defined by the rules given in Figure 2 using the standard structural congruence. The first three rules concern particularities of the present work, the other ones are inherited. For instance (R-COMM) is a really standard communication reduction rule between a sender and a receiver: the substitution of the pattern X by the actual content of the message V is triggered.

Figure 2 Reduction semantics

$$\textsc{(r-goto)}\quad l[\![\mathsf{goto}_p\, k.\ P]\!] \longrightarrow k[\![P]\!]$$

$$\textsc{(r-newloc)}\quad l[\![\mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \textstyle\sum x : \textsc{loc}.\ \mathsf{T}\ \mathsf{with}\ P_k\ \mathsf{in}\ P]\!]$$
$$\longrightarrow (\mathsf{new}\langle k, ((\vec{c}), (\vec{p}), (\vec{q})) : \mathsf{T}\{^l/_x\}\rangle)(k[\![P_k]\!] \mid l[\![P]\!])$$

$$\textsc{(r-newpass)}\quad l[\![\mathsf{newpass}\, p\ \mathsf{from}\ \tilde{k}^\star\ \mathsf{in}\ P]\!] \longrightarrow (\mathsf{new}\, p : \tilde{k}^\star \mapsto l)\, l[\![P]\!]$$

$$\textsc{(r-comm)}\quad l[\![a\,!\,\langle V\rangle\, P_1]\!] \mid l[\![a\,?\,(X : \mathsf{T})\, P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2\{^V/_X\}]\!]$$

$$\textsc{(r-if-v)}\quad l[\![\mathsf{if}\ a = a\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2]\!] \longrightarrow l[\![P_1]\!]$$

$$\textsc{(r-if-f)}\quad l[\![\mathsf{if}\ a_1 = a_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2]\!] \longrightarrow l[\![P_2]\!] \quad \text{when } a_1 \neq a_2$$

$$\textsc{(r-newchan)}\quad l[\![\mathsf{newchan}\, c : \mathsf{C}\ \mathsf{in}\ P]\!] \longrightarrow (\mathsf{new}\, c : \mathsf{C}@l)\, l[\![P]\!]$$

$$\textsc{(r-split)}\quad l[\![P_1 \mid P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2]\!]$$

$$\textsc{(r-rep)}\quad l[\![*P]\!] \longrightarrow l[\![P]\!] \mid l[\![*P]\!]$$

$$\textsc{(r-c-par)}\quad \frac{M_1 \longrightarrow M_1'}{M_1 \mid M_2 \longrightarrow M_1' \mid M_2} \qquad \textsc{(r-c-new)}\quad \frac{M_1 \longrightarrow M_1'}{(\mathsf{new}\, a : \mathsf{E})\, M_1 \longrightarrow (\mathsf{new}\, a : \mathsf{E})\, M_1'}$$

$$\textsc{(r-struct)}\quad \frac{M_1 \equiv M_2 \longrightarrow M_2' \equiv M_1'}{M_1 \longrightarrow M_1'}$$

The new reduction rules are fairly unsurprising since passports are homogeneously added to the calculus. In the reduction rule for the migration (r-goto), the passport involved is simply ignored: the verification of the passport will be performed using *types*. In the reduction rules of the constructs generating names, types are instantiated in a similar way to what is usually done for channels: when passports are actually generated in (r-newpass), they are tied to the location to which they will grant access, by getting the type $\tilde{k} \mapsto l$ (from $\tilde{k}$ to l). As for (r-newloc), the main operation taking place is the generation of a set of news, one for each of the names actually generated. Again, this operation involves modifications in the type annotations. These are taken care of via the expansion $\langle \cdots \rangle$ which sequentializes the value $(k, ((\vec{c}), (\vec{p}), (\vec{q})))$ into a list of names while attaching some type to every name. Since this relies heavily on the type system, it will be explained with it.

2.3 Type system for the language

Using the standard approach, the type system for Dπ is built by attaching types to identifiers. Following the approach used in previous works on Dπ, types describe *permissions*. This view is highly relevant for passports: the type $l \mapsto k$ we can attach to a passport indicates the authorisation to migrate from l to k. The typechecking performed on processes and systems is then based on a set of hypotheses, which associate types to names, and verifies that every use of identifiers is allowed, *i.e.* done according to the permissions tied to types. Again, we provide here only a simple presentation of the set of types to focus on passports. In particular, we got rid of the recursive types which are completely

Figure 3 Syntax of pre-types

$\mathsf{C} ::=$	*Local channel types*
$\textsc{r}\langle \mathsf{T}_1 \rangle$	Right to read values of type T_1
$\textsc{w}\langle \mathsf{T}_2 \rangle$	Right to write values of type T_2
$\textsc{rw}\langle \mathsf{T}_1, \mathsf{T}_2 \rangle$	Intersection of the two previous types
$\mathsf{E} ::=$	*Identifiers types*
$\textsc{loc}$	Location
$\mathsf{C}@u$	Channel in location u
$\tilde{u}^\star \mapsto v$	Passport from $\tilde{u}^\star$ to v
$\mathsf{T} ::=$	*Transmissible values types*
E	Identifier
C	Local channel
$\hookrightarrow v$	Local passport to v
$(\mathsf{T}_1, \ldots, \mathsf{T}_n)$	Tuple
$\sum \vec{x} : \vec{\textsc{loc}}.\ \mathsf{T}$	Dependent sum
$\mathsf{L} ::=$	*Types to declare locations*

$$\sum x : \textsc{loc}.\ \sum y : \textsc{loc}.\ (\mathsf{C}_1@y, \ldots), (\tilde{u}_1^\star \mapsto y, \ldots), (\tilde{v}_1^\star \mapsto x, \ldots)$$

orthogonal to passports types; see [HH07] for a detailed account of recursive types.

2.3.1 Types for identifiers and values

The definition of the types that can be associated with identifiers is stratified: the types for channels mention the types of the values exchanged over it. So the types for identifiers and for values are defined at the same time: they are summed up in Figure 3. In fact, the syntax given in this figure describes *pre-types*. We will explain shortly which pre-types are actual types.

Two major modifications are made in types. Firstly, we introduce new types for passports:

- $\tilde{u} \mapsto v$ will be the type of a passport to access v from one of the locations in $\tilde{u}$ and $\star \mapsto v$ of a universal passport to v;
- when passports to v are communicated inside a location l to be used to migrate from l, we call them *local passports* and we can give them the facility type $\hookrightarrow v$.

Secondly, we add a dependent sum type for values that are transmitted over channels: since the type for a passport mentions the names of the source and target locations, the dependent sum provides a way to send those names (locations and passport), packed together. This way, a location l and a local passport p to migrate to l might form a pair (l, p) of dependent type $\sum x : \textsc{loc}.\ \hookrightarrow x$. The type T in the server of our first example (2) would be such a dependent sum, with a tuple as the second value.

Those dependent types are also used to describe the tie between the locations and the passports in the newloc construct, as described by the *declarative*

Figure 4 Expansion of values

$$
\begin{aligned}
&\langle u : \mathsf{C} \rangle\,{\scriptstyle @}w = u : \mathsf{C}{\scriptstyle @}w\\
&\langle u : \mathsf{C}{\scriptstyle @}w_0 \rangle\,{\scriptstyle @}w = u : \mathsf{C}{\scriptstyle @}w_0\\
&\langle u : \textsc{loc} \rangle\,{\scriptstyle @}w = u : \textsc{loc}\\
&\langle ((\vec{u}), V) : \textstyle\sum \vec{x} : \vec{\textsc{loc}}.\ \mathsf{T} \rangle\,{\scriptstyle @}w = u_1 : \textsc{loc}, \ldots, \langle V : \mathsf{T}\{{}^{\vec{u}}\!/\!_{\vec{x}}\} \rangle\,{\scriptstyle @}w\\
&\langle (V_1, \ldots, V_n) : (\mathsf{T}_1, \ldots, \mathsf{T}_n) \rangle\,{\scriptstyle @}w = \langle V_1 : \mathsf{T}_1 \rangle\,{\scriptstyle @}w, \ldots, \langle V_n : \mathsf{T}_n \rangle\,{\scriptstyle @}w\\
&\langle u : \tilde{v}^{\star} \mapsto v' \rangle\,{\scriptstyle @}w = u : \tilde{v}^{\star} \mapsto v'\\
&\langle u :\ \hookrightarrow v' \rangle\,{\scriptstyle @}w = u : w \mapsto v'
\end{aligned}
$$

types L. As mentioned before, the system

$$l[\![\mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \mathsf{L}\, \mathsf{with}\, P_k\, \mathsf{in}\, P]\!]$$

will generate a new location k together with a set of passports $\vec{p}$ to access k (for instance from l), a set of passports $\vec{q}$ to access l (for instance from l) and, finally, a set of new channels $\vec{c}$ located inside k. The tie between those passports and the locations they grant access to are encoded in dependencies: in the types of the form L, the type variable x will be bound to the name of the *mother* location (l in our example), and y to the *child* location (here k). As the reduction rule (R-NEWLOC) indicates, when the type L used in the newloc construct is of the form $\sum x : \textsc{loc}.\ \mathsf{T}$, the actual binding of x to l is performed by a simple substitution $\{{}^{l}\!/\!_{x}\}$ on T. On the other hand, the binding of y to k results from the *expansion* $\langle k, ((\vec{c}), (\vec{p}), (\vec{q})) : \mathsf{T}\{{}^{l}\!/\!_{x}\} \rangle$.

The expansion is formally described in Figure 4: $\langle V : \mathsf{T} \rangle\,{\scriptstyle @}w$ intuitively corresponds to the reception of the value V at the type T in the location w. More specifically it plays two roles: it takes into account the location where the expansion is performed to transform every local channel type and every local passport type into their located equivalents; and it extracts from T the type that correspond to every identifier appearing in the value V. This is done by generating a list of *identifier* : *type*. The rules in Figure 4 show that, when the type does not contain any local channel or local passport, the expansion is independent of the location where it is performed. In that case, we will use the *unlocated* expansion (without the location ${\scriptstyle @}w$). That is why, in the rule (R-NEWLOC) where the specific form of the types L prohibits local channel types and local passport types, the expansion is used unlocated to resolve the newloc into a list of news.

Dependent sums and the fact that they can bind a variable in a type play a very important role in our setting: they allow to avoid unbound identifiers in the type of the values exchanged over some channel. So, for instance, $\textsc{r}\langle \textsc{r}\langle {\scriptstyle @} \rangle x \rangle{\scriptstyle @}k$ will not be a valid type for a channel even if it is in the syntax of pre-types. Thanks to dependent sums, we can thus avoid substitution of variables in types when they are instantiated after a communication. This constraint of closure of the scopes of all identifiers appearing inside value types is one of the two properties that differentiate pre-types from types. The second constraint bears on types of the form $\textsc{rw}\langle \mathsf{T}^r, \mathsf{T}^w \rangle$ When values are sent at type T^w and received at type T^r, the types T^w and T^r must be related: considering only the permissions conveyed via types, all the received rights must have been sent. We therefore formalise this relationship between types as a *subtyping* order.

2.3.2 Subtyping

We inherit from previous works on Dπ a subtyping order. This order is extended in a fairly natural way on passport (pre-)types: for instance, a universal passport to access l allows to come from anywhere so should be a subtype of any passport to l. The following inference rules sum up subtyping for passports:

$$\text{(SR-PASS)}\ \frac{\tilde{u}' \subseteq \tilde{u}}{\tilde{u} \mapsto v <: \tilde{u}' \mapsto v} \qquad \text{(SR-PASS-*)}\ \ \star \mapsto v <: \tilde{u}^\star \mapsto v$$

$$\text{(SR-LOCAL-PASS)}\ \ \hookrightarrow v <: \ \hookrightarrow v$$

The rule for dependent sums is even simpler:

$$\text{(SR-DEP)}\ \frac{\mathsf{T}_1 <: \mathsf{T}_2}{\sum \vec{x} : \vec{\textsc{loc}}.\ \mathsf{T}_1 <: \sum \vec{x} : \vec{\textsc{loc}}.\ \mathsf{T}_2}$$

We refer the reader to previous works (in particular [HH07]) for a complete presentation of subtyping in Dπ.

Now that subtyping is properly defined, we can formally define *types*:

Definition 2.1 (Types)**.** *A pre-type* T *is a* type *when:*

- *for every occurrence of a type of the form* $\textsc{rw}\langle \mathsf{T}^r, \mathsf{T}^w\rangle$ *it contains,* $\mathsf{T}^w <: \mathsf{T}^r$*;*
- *for every occurrence of a type of the form* C *it contains, every identifier appearing in* C *is bound by a dependent sum.*

The new subtyping rules have only a small impact over the theory of types, *i.e.* the structure of the ordered set of types, for the calculus. Indeed, since any two passport types can have a common subtype only when they grant access to the same location, we see easily that they must have a greatest lower bound, *i.e.* a *meet*, as soon as they have a common subtype. We write the fact that T_1 and T_2 share a common subtype as $\mathsf{T}_1 \downarrow \mathsf{T}_2$ and we say that they are $\downarrow$-compatible. So the property of partial meets, preserved in the present setting, can be stated as:

Theorem 2.1 (Partial meets)**.** *Any two* $\downarrow$*-compatible types have a meet.*

We can now build a type system for processes and systems upon this structured set of types for identifiers and values.

2.3.3 Typechecking processes and systems

As usual, the type system relies on *type environments*, written Γ, Φ, Ω, which are lists of hypotheses, *i.e.* associations of types to identifiers, for instance $l : \textsc{loc}, k : \textsc{loc}, p : \star \mapsto k, \ldots$ Based on those environments, the typechecking of systems is layered as follows.

- The consistency of environments must be checked: when a given identifier is given two different types, we must verify that those two types are $\downarrow$-compatible. This ensures that we cannot end up associated both a location type and a passport type to a single identifier, for instance.

This is performed using inference rules the conclusions of which look like: "$\Gamma \vdash_{\mathbf{env}}$" to mean that the environment Γ is *well-formed*. The rule for passports simply reads:

$$\text{(E-PASS)}\quad \frac{\Gamma \vdash \mathbf{env} \quad w : \text{LOC} \in \Gamma \quad \forall w_i \in \tilde{w}, w_i : \text{LOC} \in \Gamma}{\Gamma, u : \tilde{w}^\star \mapsto w \vdash \mathbf{env}} \quad \downarrow(\Gamma(u) \cup \{\tilde{w}^\star \mapsto w\})$$

which means that the extension of the well-formed environment Γ with hypothesis $u : \tilde{w}^\star \mapsto w$ does not break the consistency of the environment whenever:

- w and all the identifiers in $\tilde{w}$ already appear as locations in Γ;
- all the types that are already associated with u in Γ, written $\Gamma(u)$, must be $\downarrow$-compatible with $\tilde{w}^\star \mapsto w$; by Theorem 2.1 this means that the set of all those types must have a meet which sums up all the rights granted on u.

The verification is similar for the other identifier types and is therefore given only in appendix (see Figure 7).

- Some inference rules then correspond to judgments of the form $\Gamma \vdash u : \mathsf{E}$ meaning that the type E can be associated to u under the set of hypotheses Γ. In particular, subtyping has to be taken into account for this, for instance to prove

$$l : \text{LOC}, \vec{l} : \text{LOC}, p : l_1, l_2 \mapsto l, p : l_3, l_4 \mapsto l \vdash p : l_1, l_3 \mapsto l$$

i.e. where the two types associated with p in the environment have to be combined to conclude that p authorises migrations from both l_1 and l_3. Another set of rules builds on them to obtain statements about values. The major difference is the fact that those judgements are *localised*: in the previous environment, it is possible to conclude that p has the local passport type $\hookrightarrow l$ only when the value will be used in one of the locations $l_1, \ldots, l_4$. Consequently the judgments take the form

$$\Gamma \vdash_u V : \mathsf{T}$$

where u is the location in which the value V can be given the type T under the hypotheses Γ. As for the well-formedness of environments, the corresponding rules are fairly straightforward and are then given only in appendix (see Figure 8).

- Typechecking of processes must be performed according to the location where the process is running: the process $a\,!\,\langle\rangle$ can be correct only when it is launched in the location of the channel a. Consequently, as for the typechecking of values, the statements for typechecking of processes take the following form:

$$\Gamma \vdash_u P$$

which intuitively states that running the process P in location u will require at most the permissions contained in Γ.

Let us first consider the typing rules for the use and the creation of passports.

$$\text{(T-GOTO)}\quad \frac{\Gamma \vdash u : w \mapsto v \qquad \Gamma \vdash_v P}{\Gamma \vdash_w \mathsf{goto}_u\, v.\, P}$$

$$\text{(T-NEWPASS)}\quad \frac{\Gamma \vdash \tilde{u} : \widetilde{\text{LOC}} \qquad \Gamma; p : \tilde{u}^\star \mapsto w \vdash_w P}{\Gamma \vdash_w \mathsf{newpass}\, p\, \mathsf{from}\, \tilde{u}^\star\, \mathsf{in}\, P}$$

These rules should be self-explanatory: to migrate between its "current" location and a location v, a process must own an appropriate passport. Since the rules for typing identifiers can use subtyping, the hypothesis $\Gamma \vdash u : w \mapsto v$ in (T-GOTO) is simply stating that the passport u must be valid to enter v from any set of locations containing w. The rule (T-NEWPASS) is even simpler: it simply forges the passport type by using the current location so that the creation of passports to enter a given location can happen only from inside that location. This simple property ensures that a location has an actual control over the set of passports that grant access to it.

As we already mentioned, passports can also be generated at the same time as a location: without the possibility to set up passports at the same time as the child location, processes coming from the mother would be denied access and vice versa. We want to be able to express all the possible links: whether the mother location gives access to the processes coming from its child location, etc.

$$\text{(T-NEWLOC)}\quad \frac{\begin{array}{c}\Gamma;\langle(k,((\vec{c}),(\vec{p}),(\vec{q}))) : \mathsf{T}\{^{w}/_{x}\}\rangle \vdash_k P_k \\ \Gamma;\langle(k,((\vec{c}),(\vec{p}),(\vec{q}))) : \mathsf{T}\{^{w}/_{x}\}\rangle \vdash_w P\end{array}}{\Gamma \vdash_w \mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \sum x : \text{LOC}.\ \mathsf{T}\, \mathsf{with}\, P_k\, \mathsf{in}\, P}$$

This typing rule uses a small "trick" with the dependent sum. Recall that the type $\sum x : \text{LOC}.\ \mathsf{T}$ appearing in the newloc construct is of the form

$$\sum x : \text{LOC}.\ \sum y : \text{LOC}.\ (\mathsf{C}_1{}_{@}y, \ldots), (\tilde{u}_1^\star \mapsto y, \ldots), (\tilde{v}_1^\star \mapsto x, \ldots)$$

The set of names $\vec{p}$ are passports to y, the second dependent variable, which will be associated to the new location k by virtue of the expansion of the dependent sum. But the passports $\vec{q}$ should grant access to the location w where the construct is invoked, which is why the reduction rule instantiates on the fly the variable x by the name of the location. The first dependent sum serves thus only the role of binder for that variable.

Finally, let us look at another rule of interest:

$$\text{(T-IF)}\quad \frac{\Gamma \vdash_w P_2 \qquad [\Gamma]_{u_1=u_2} \vdash_w P_1 \text{ when } [\Gamma]_{u_1=u_2} \vdash \mathbf{env}}{\Gamma \vdash_w \mathsf{if}\ u_1 = u_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}$$

This rule simply states that all the permissions available after a failed comparison between two identifiers u_1 and u_2 are only the available permissions before the test. But in the other case, the process "knows" that the two identifiers are identical: we want to merge all the permissions the process owns over those two identifiers. This is performed using $[\cdot]_{u_1=u_2}$

Figure 5 Bracket extension of environments

$$\begin{array}{rcl}
[\Gamma, u_i : \mathsf{E}]_{u_1=u_2} & = & [\Gamma]_{u_1=u_2}, u_1 : \mathsf{E}, u_2 : \mathsf{E} \\
[\Gamma, v : \mathsf{C}\text{@}u_i]_{u_1=u_2} & = & [\Gamma]_{u_1=u_2}, v : \mathsf{C}\text{@}u_1, v : \mathsf{C}\text{@}u_2 \quad \text{if } v \notin \{u_1, u_2\} \\
[\Gamma, v : \tilde{w} \mapsto u_i]_{u_2=u_1} & = & [\Gamma]_{u_2=u_1}, v : \tilde{w} \mapsto u_1, u : \tilde{w} \mapsto u_2 \quad \text{if } (\tilde{w} \cup \{v\}) \cap \{u_1, u_2\} = \emptyset \\
[\Gamma, v : u_i, \tilde{w} \mapsto w]_{u_2=u_1} & = & [\Gamma]_{u_2=u_1}, v : u_1, u_2, \tilde{w} \mapsto w \quad \text{if } \{v, w\} \cap \{u_1, u_2\} = \emptyset \\
[\Gamma, v : u_i, \tilde{w} \mapsto u_j]_{u_2=u_1} & = & [\Gamma]_{u_2=u_1}, v : u_1, u_2, \tilde{w} \mapsto u_1, v : u_1, u_2, \tilde{w} \mapsto u_2 \quad \text{if } v \notin \{u_1, u_2\} \\
[\Gamma, v : \star \mapsto u_i]_{u_2=u_1} & = & [\Gamma]_{u_2=u_1}, v : \star \mapsto u_1, v : \star \mapsto u_2 \quad \text{if } v \notin \{u_1, u_2\} \\
[\Gamma, v : \mathsf{E}]_{u_1=u_2} & = & [\Gamma]_{u_1=u_2}, v : \mathsf{E} \quad \text{otherwise}
\end{array}$$

which modifies the environment to duplicate any statement about u_1 into a statement about u_2 and vice versa. So, for instance, if Γ contains the two hypotheses $u_1 : \textsc{r}\langle\rangle\text{@}l$ and $u_2 : \textsc{w}\langle\rangle\text{@}l$, $[\Gamma]_{u_1=u_2}$ will contain the four

$$u_1 : \textsc{r}\langle\rangle\text{@}l \qquad u_2 : \textsc{r}\langle\rangle\text{@}l \qquad u_2 : \textsc{w}\langle\rangle\text{@}l \qquad u_1 : \textsc{w}\langle\rangle\text{@}l$$

The operation is slightly more complex when the u_i can appear in the set $\tilde{w}$ in a passport type $\tilde{w} \mapsto w$. The full set of cases to define the bracket extension of environments is given in Figure 5. Notice that, when the environment contains a hypothesis $u_1 : \mathsf{C}\text{@}u_2$, all substitutions are not performed. But that case can fail: if u_2 appears to be a location while u_1 is a channel, the test of their equality will always turn out to be false. That is why the rule (T-IF) will enforce the typechecking of the process P_1 only when the environment $[\Gamma]_{u_1=u_2}$ is actually well-formed: two identifiers with incompatible types cannot be equal so P_1 will never be triggered in that case.

The test of compatibility raises a problem. Consider the following environment:

$$\Gamma = p : l_0 \mapsto l, x_l : \textsc{loc}, x_p : l_0 \mapsto x_l$$

and the test that x_p is equal to p. $[\Gamma]_{p=x_p}$ will contain $p : l_0 \mapsto l$ and $p : l_0 \mapsto x_l$. With the definition of $\downarrow$-compatibility we gave (the fact that they share a subtype), the environment would not be well-formed because subtyping over passport types enforces the equality of the destinations (l and x_l). So we use a *weak* $\downarrow$*-compatibility*: two passport types with different destinations are *weakly* $\downarrow$*-compatible* when at most one of the destinations is a name, all the other ones must be variables. Indeed the cases of weak $\downarrow$-compatibility will be encountered in well-formed environments only because of the bracket extension, *i.e.* only in the *true* branch of a test: the variables (and the eventual name) have been compared and found equal. So those passport types can be considered compatible since we know that they will share the same destination. The weak $\downarrow$-compatibility is also defined in the same way for channel types.

The other rules of typechecking for processes are pretty much standard; for the sake of completeness we provide the full set in appendix, Figure 9.

- Finally, systems can be typechecked. The most typical rule simply states:

$$\text{(T-PROC)} \quad \frac{\Gamma \vdash l : \textsc{loc} \qquad \Gamma \vdash_l P}{\Gamma \vdash l[\![P]\!]}$$

  Again, these rules are completely inherited (see Figure 10 for the complete set).

2.3.4 Properties of the type system

The complex type system explained in the previous sections gives two standard properties: subject reduction and type safety.

Theorem 2.2 (Subject reduction). *$\Gamma \vdash M$ and $M \longrightarrow^* N$ imply $\Gamma \vdash N$.*

The use of bracket extensions makes the proof of that standard theorem more complex, mostly because of the interactions between the substitutions and the extensions. The interested reader will find in [Hym06] a detailed account of this proof.

The property of type safety is more relevant in the present work. Let us state here only the important part as far as passports are concerned. Following the approach developed in [WF94], we define *erroneous reduction* of a system M, written $M \xrightarrow{\text{err}}_\Gamma$, by a set of rules. The rule about passports states:

$$l[\![\mathsf{goto}_p\, k.\ P]\!] \xrightarrow{\text{err}}_\Gamma \qquad \text{if } \Gamma \not\vdash_l p : l \mapsto k$$

in any context. The erroneous reduction expresses then the fact that an appropriate passport is mandatory to migrate, and the type safety property will allow to conclude that no process in a well-typed system will ever try to enter a location without proper right to do so.

Theorem 2.3 (Type safety). *$\Gamma \vdash M$ implies $M \not\xrightarrow{\text{err}}_\Gamma$.*

The proof of this theorem follows directly from the typechecking rules.

3 Loyal observational equivalence

The main goal of passports is to allow a location to control the processes it accepts. Naturally, this implies that the *observable* behaviour of a system depends on the actual authorisations the *observer* is granted. Let us then define an equivalence that takes passports into account drawing inspiration from [MS92], namely an equivalence in which the observer does not "cheat" and sees only what the system allows it to see.

For this, we will describe explicitly the knowledge of the observer, *i.e.* the rights he got access to, including his passports, using a *type environment* written Ω. Following [HMR03], we will therefore consider *configurations*, written $\Omega \rhd M$, when the system M is facing an observer knowing Ω. That configuration will be meaningful only when Ω and M agree in some sense: if M is the system $a[\![P]\!]$, Ω should not associate a passport type to a. To avoid such conflicts, we will restrict our attention to *well-formed configurations*, *i.e.* configurations $\Omega \rhd M$ for which there exists some environment Γ such that $\Gamma \vdash M$ and $\Gamma <: \Omega$ ($<:$ is extended from types to environment in the natural way).

The relations will also mention the observer's knowledge by relating configurations. Since we will mostly insist on equivalence relations where two systems are impossible to distinguish for the same observer, we will write $\Omega \vDash M \mathrel{\mathcal{S}} N$ when $(\Omega \rhd M)\mathcal{S}(\Omega \rhd N)$.

To obtain an observational equivalence, let us first define the basic observations. They must be interactions with the studied system, *i.e.* communications over some channels. Since channels are located, this will be possible only when the observer is granted access to their location. To actually allow the system to "choose" which locations should be reachable, we decided to place the observer into a *fresh* location. This implies that the only *directly reachable* locations are the destinations of the *universal passports* in Ω. So we define *barbs* thus:

Definition 3.1 (Barbs)**.** *M shows a barb on c to Ω, written $\Omega \rhd M \Downarrow c$, whenever there exist a location l and a passport p such that:*

- *$\Omega \vdash p : \star \mapsto l$;*
- *$\Omega \vdash c : \mathrm{R}\langle \mathsf{T}\rangle_{@}l$, for some type T;*
- *there exist some P, M' and $(\vec{a} : \vec{\mathsf{E}})$ with $c, l \notin \vec{a}$ and such that $M \longrightarrow^{*}\equiv (\mathsf{new}\, \vec{a} : \vec{\mathsf{E}})(M' \mid l[\![c\,!\,\langle V\rangle\, P]\!])$.*

Some observer knowing Ω will be able to distinguish two systems as soon as they show different sets of barbs. To get an equivalence out of this simple property, the observer is usually allowed to test the system by putting it in any context in order to eventually obtain a distinguishing barb. In our setting, we should consider only *loyal* contexts, *i.e.* contexts which use only rights available to the observer: they should not try to launch code in *unreachable* locations and access channels without the corresponding permissions. We formally define a location l as *reachable* knowing Ω when there exist $p : \star \mapsto l_1$, $p_1 : l_1 \mapsto l_2$, $\ldots$, $p_n : l_n \mapsto l$ in Ω. We will write $\mathcal{R}_\Omega$ for the set of such reachable locations. Then a context of the form $[\cdot] \mid l[\![P]\!]$ is *loyal* only when l is reachable and P is well-typed in Ω. The observer must also be *loyal* when introducing new names (for instance to be used in P):

Definition 3.2 (Loyal extension)**.** *Γ' is a* loyal extension *of Γ when:*

- *$\Gamma; \Gamma'$ is a well-formed environment;*
- *for every $u : \tilde{v}^{\star} \mapsto w$ and $u : \mathsf{C}_{@}w$ in Γ' when w appears in Γ, then $w \in \mathcal{R}_\Gamma$.*

Finally, we define the loyal contextuality of a relation $\mathcal{S}$. The loyal barbed congruence follows from this notion.

Definition 3.3 (Loyally contextual relation)**.** *A relation $\mathcal{S}$ is said* loyally contextual *only when:*

- *If $\Omega \vDash M \mathrel{\mathcal{S}} N$ and Ω' is a loyal extension of Ω such that for every hypothesis $a : \mathsf{E}$ in Ω', a is fresh, then $\Omega; \Omega' \vDash M \mathrel{\mathcal{S}} N$.*
- *If $\Omega \vDash M \mathrel{\mathcal{S}} N$, $k \in \mathcal{R}_\Omega$ and $\Omega \vdash k[\![P]\!]$ then $\Omega \vDash M \mid k[\![P]\!] \mathrel{\mathcal{S}} N \mid k[\![P]\!]$.*
- *If $\Omega; a : \mathsf{E} \vDash M \mathrel{\mathcal{S}} N$ and both configurations $\Omega \rhd (\mathsf{new}\, a : \mathsf{E})\, M$ and $\Omega \rhd (\mathsf{new}\, a : \mathsf{E})\, N$ are well-formed, then $\Omega \vDash (\mathsf{new}\, a : \mathsf{E})\, M \mathrel{\mathcal{S}} (\mathsf{new}\, a : \mathsf{E})\, N$.*

Definition 3.4 (Loyal barbed congruence)**.** *We call* loyal barbed congruence, *written $\cong^{l}$, the biggest symmetric loyally contextual relation that preserves barbs and is closed over reductions.*

Figure 6 Labelled transition system, significant rules

$$\textsc{(lts-goto)} \quad \Omega \rhd l[\![\mathsf{goto}_p\, k.\ P]\!] \xrightarrow{\tau} \Omega \rhd k[\![P]\!]$$

$$\textsc{(lts-w)} \quad \frac{l \in \mathcal{R}_\Omega \qquad \Omega \vdash_l a : \mathrm{R}\langle \mathsf{T} \rangle \quad \text{where } \mathsf{T} = \Omega^r(a)}{\Omega \rhd l[\![a\,!\,\langle V \rangle\, P]\!] \xrightarrow{a!V} \Omega, \langle V : \mathsf{T} \rangle \rhd l[\![P]\!]}$$

$$\textsc{(lts-r)} \quad \frac{l \in \mathcal{R}_\Omega \qquad \Omega \vdash_l a : \mathrm{W}\langle \mathsf{T}' \rangle \qquad \Omega \vdash_l V : \mathsf{T}'}{\Omega \rhd l[\![a\,?\,(X : \mathsf{T})\, P]\!] \xrightarrow{a?V} \Omega \rhd l[\![P\{{}^V\!/x\}]\!]}$$

$$\textsc{(lts-comm)} \quad \frac{\begin{array}{c} \Omega_M \rhd M \xrightarrow{(\Phi)a!V} \Omega'_M \rhd M' \\ \Omega_N \rhd N \xrightarrow{(\Phi)a?V} \Omega'_N \rhd N' \end{array}}{\begin{array}{c} \Omega \rhd M \,|\, N \xrightarrow{\tau} \Omega \rhd (\mathsf{new}\,\Phi)\, M' \,|\, N' \\ \Omega \rhd N \,|\, M \xrightarrow{\tau} \Omega \rhd (\mathsf{new}\,\Phi)\, N' \,|\, M' \end{array}}$$

$$\textsc{(lts-c-par)} \quad \frac{\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'}{\begin{array}{c} \Omega \rhd M \,|\, N \xrightarrow{\mu} \Omega' \rhd M' \,|\, N \\ \Omega \rhd N \,|\, M \xrightarrow{\mu} \Omega' \rhd N \,|\, M' \end{array}}$$

$$\textsc{(lts-c-new)} \quad \frac{\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'}{\Omega \rhd (\mathsf{new}\, a : \mathsf{E})\, M \xrightarrow{\mu} \Omega' \rhd (\mathsf{new}\, a : \mathsf{E})\, M'} \quad a \notin \mathsf{n}(\mu)$$

$$\textsc{(lts-open)} \quad \frac{\Omega \rhd M \xrightarrow{(\Phi)a!V} \Omega' \rhd M'}{\Omega \rhd (\mathsf{new}\, b : \mathsf{E})\, M \xrightarrow{(b:\mathsf{E};\Phi)a!V} \Omega' \rhd M'} \quad \begin{array}{l} b \neq a \\ b \in \mathsf{fn}(V) \cup \mathsf{fn}(\Phi) \end{array}$$

$$\textsc{(lts-weak)} \quad \frac{\Omega; \Omega_e \rhd M \xrightarrow{(\Phi)a?V} \Omega' \rhd M'}{\Omega \rhd M \xrightarrow{(\Omega_e;\Phi)a?V} \Omega' \rhd M'} \quad \begin{array}{l} \mathsf{dom}(\Omega_e) \cap (\{a\} \cup \mathsf{fn}(M)) = \emptyset \\ \Omega_e \text{ is a loyal extension of } \Omega \end{array}$$

4 Loyal bisimilarity

The definition given for the loyal barbed congruence is justified by intuitions but it is complex and highly intractable. So we also propose a complete proof technique for this equivalence: a bisimilarity. The idea of the bisimilarity is to provide an alternative but equivalent definition of the semantics using a *Labelled Transition System* (LTS) where the labels represents the possible interactions between the system and its environment. Then two systems can be distinguished if, after some preliminary interactions, one can perform a transition the other cannot.

4.1 Labelled transitions system

The way the LTS is built is completely standard: we associate the label τ to every internal reduction a system can perform, to indicate that the environment is not involved. This means that every reduction rule of Figure 2 but (R-COMM) becomes a transition labelled by τ. For instance, let us mention the rule (LTS-GOTO):

$$\Omega \rhd l[\![\mathsf{goto}_p\, k.\ P]\!] \xrightarrow{\tau} \Omega \rhd k[\![P]\!]$$

Note that, since the interactions we are characterising are between some system M and an observer knowing Ω, we define transitions on *configurations*. Also note that the knowledge of the observer is left untouched in a τ transition since he is not interacting with the system. We present in Figure 6 only some significant rules: all the other ones are τ transitions that can be directly derived from the reduction semantics.

Let us explain the major rules of the LTS and start with (LTS-W). The conditions of this rule are similar to the ones for barbs. Indeed an observer knowing Ω will be able to interact with a system outputting a message V on a channel a in a location l only when l is reachable ($l \in \mathcal{R}_\Omega$) and when the observer can input on that channel ($\Omega \vdash_l a : \text{R}\langle\mathsf{T}\rangle$). The knowledge of the observer will consequently be enriched by the message: Ω becomes $\Omega,\langle V : \mathsf{T}\rangle$ along that transition. In this expression, the type T indicates all the rights the observer learns, calculated using the *meet* of the types associated with the channel. Suppose for instance that the meet of all the types associated with a in Ω is $\text{RW}\langle\mathsf{T}_1,\mathsf{T}_2\rangle@l$; then T_1 sums up all the rights that can be obtained by inputting on a. We denote that type T_1 as $\Omega^r(a)$ in (LTS-W).

The rule (LTS-R) is symmetrical: the observer also needs to have access to the location where the interaction takes place and to be allowed to actually send the message that the observed system will receive. Of course, the knowledge of the observer is not increased by the message since it is its author.

As usual, the rules (R-COMM) and (LTS-COMM) have little in common: in the reduction semantics, the possibility to interact for two processes that are not syntactically close is guaranteed via the structural congruence; in the LTS, this is replaced by the fact that a system containing a process about to send a message will be able to perform an output transition $\xrightarrow{\cdot!\cdot}$. But in the typed LTS we present here, the type environment representing the knowledge of the observer must allow the input and the output to be performed. To ensure this, note that (LTS-COMM) does not specify the type environment in which both the output and the input are possible: Ω_M and Ω_N are simply two environments in which the transitions can be proved. And, as soon as two such environments exist, we can conclude that the two subsystems can communicate. We can also notice that (LTS-COMM) must close, as usual, the scopes of the names, scopes that must be opened by (LTS-OPEN) to permit the communication: here this implies that the output labels have to contain the types associated with all the names; this is why we reuse the notation for type environments (Φ) in labels.

Finally, let us look at the rule (LTS-WEAK). Exactly as (LTS-W) which closely matches the criterium used in defining barbs, that rule is constrained in the same way as the loyal contextuality (Definition 3.3).

Since the LTS defines some semantics for the calculus, we want to make sure that the semantics coincide with the reduction semantics:

Theorem 4.1 (Coincidence of semantics)**.** *The reduction semantics and the semantics extracted from the LTS coincide in the sense of the following two properties:*

- $\Omega \triangleright M \xrightarrow{\tau} \Omega \triangleright M'$ *implies that* $M \longrightarrow M'$*;*
- $M \longrightarrow M'$ *implies that there exists some* $M'' \equiv M'$ *with* $\Omega \triangleright M \xrightarrow{\tau} \Omega \triangleright M''$*.*

4.2 Bisimilarity equivalence

With the LTS defined above, we would like to define an equivalence $\mathcal{R}$ as a standard bisimulation: when $\Omega \vDash M \mathrel{\mathcal{R}} N$ and $\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'$ then there must exist some N' such that $\Omega \rhd N \overset{\hat{\mu}}{\Longrightarrow} \Omega' \rhd N'$ and $\Omega' \vDash M' \mathrel{\mathcal{R}} N'$ where $\overset{\hat{\tau}}{\Longrightarrow} = \xrightarrow{\tau}^* $ and $\overset{\hat{\mu}}{\Longrightarrow} = \xrightarrow{\tau}^* \xrightarrow{\mu} \xrightarrow{\tau}^*$ when $\mu \neq \tau$. Remark that the use of the same Ω' for M and N is not constraining: the knowledge of the observer is modified in exactly the same way along the transitions $\xrightarrow{\mu}$ and $\overset{\hat{\mu}}{\Longrightarrow}$, whatever μ may be.

But this definition cannot be used right away in our case, because of dependent types. Let us consider a case where the discrepancy appears. Suppose some channel c in l on which a passport can be transmitted (so c is of type $\textsc{rw}\langle \sum x, y : \vec{\textsc{loc}}.\ x \mapsto y\rangle_{@}l)$ and consider the following two systems:

$$(\mathsf{new}\, k' : \textsc{loc}) \quad (\mathsf{new}\, p : k, k' \mapsto l) \quad l[\![c\,!\,\langle (k,l),(p)\rangle\; d\,!\,\langle k'\rangle]\!] \qquad (1)$$

$$(\mathsf{new}\, k' : \textsc{loc}) \quad (\mathsf{new}\, p : k \mapsto l) \quad l[\![c\,!\,\langle (k,l),(p)\rangle\; d\,!\,\langle k'\rangle]\!] \qquad (2)$$

The only difference is the fact that the passport p can be used also from the new location k' in the first system. Since the observer receives p at the type $k \mapsto l$ in both cases, it should not be able to make the difference. But, they can perform the following transitions with *distinct labels* (for simplicity, we ignore the type annotations in the labels):

$$\begin{array}{lrl}
\Omega \rhd (1) & \xrightarrow{(k',p)c!((k,l),(p))} & \Omega, p : k \mapsto l \rhd l[\![d\,!\,\langle k'\rangle]\!] \\
 & \xrightarrow{d!k'} & \Omega, p : k \mapsto l, k' : \textsc{loc} \rhd l[\![\mathsf{stop}]\!] \\
\Omega \rhd (2) & \xrightarrow{(p)c!((k,l),(p))} & \Omega, p : k \mapsto l \rhd (\mathsf{new}\, k' : \textsc{loc})\, l[\![d\,!\,\langle k'\rangle]\!] \\
 & \xrightarrow{(k')d!k'} & \Omega, p : k \mapsto l, k' : \textsc{loc} \rhd l[\![\mathsf{stop}]\!]
\end{array}$$

namely not opening the scope of k' in the same transition. To avoid this problem, we annotate configurations with a set of names whose scopes have been opened because of type dependencies, not because they were revealed. The labels are modified accordingly to mention only the names that are actually revealed.

Definition 4.1 (Actions)**.** *The* annotated configuration $\Omega \rhd_{\tilde{a}} M$ *can perform the* action μ *and become* $\Omega' \rhd_{\tilde{a}'} M'$ *when:*

- *if μ is τ or $(\Phi)a?V$: the transition $\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'$ is provable in the LTS and $\tilde{a} = \tilde{a}'$;*
- *if μ is $(\tilde{b})a!V$: the transition $\Omega \rhd M \xrightarrow{(\Phi)a!V} \Omega' \rhd M'$ is provable in the LTS, $\tilde{b} = \mathsf{fn}(V) \cap (\mathsf{dom}(\Phi) \cup \tilde{a})$ and $\tilde{a}' = (\mathsf{dom}(\Phi) \cup \tilde{a}) \setminus \mathsf{fn}(V)$.*

So the definition of actions enforces that the names $\tilde{b}$ mentioned in an output action are indeed revealed to the observer, since they must appear in the message (*i.e.* in the set $\mathsf{fn}(V)$), whether their scopes is opened by this action (so appearing in $\mathsf{dom}(\Phi)$) or kept hidden in the annotation $\tilde{a}$.

So, starting the two previous systems with empty annotations, they can perform the following actions:

$$\begin{array}{lrl}
\Omega \rhd_{\emptyset} (1) & \xrightarrow{(p)c!((k,l),(p))} & \Omega, p : k \mapsto l \rhd_{\{k'\}} l[\![d\,!\,\langle k'\rangle]\!] \\
 & \xrightarrow{(k')d!k'} & \Omega, p : k \mapsto l, k' : \textsc{loc} \rhd_{\emptyset} l[\![\mathsf{stop}]\!] \\
\Omega \rhd_{\emptyset} (2) & \xrightarrow{(p)c!((k,l),(p))} & \Omega, p : k \mapsto l \rhd_{\emptyset} (\mathsf{new}\, k' : \textsc{loc})\, l[\![d\,!\,\langle k'\rangle]\!] \\
 & \xrightarrow{(k')d!k'} & \Omega, p : k \mapsto l, k' : \textsc{loc} \rhd_{\emptyset} l[\![\mathsf{stop}]\!]
\end{array}$$

Using those annotated configuration, it becomes possible to define a meaningful equivalence as a bisimilarity.

Definition 4.2 (Loyal bisimilarity)**.** *The* loyal bisimilarity*, written* $\approx^{al}$*, is the largest bisimulation defined in the standard way over* actions *of annotated configurations.*

4.3 Equivalences coincidence

Since the bisimilarity has been introduced as a proof technique, we have to prove that, under some conditions, the two equivalences coincide. The proof of that property is significantly more complex than its equivalent in the literature (see for instance [HMR03]). We will describe here only the main steps of the proof; it is fully detailed in [Hym06].

4.3.1 Dealing with annotations

The first difference to take into account is the fact that the bisimilarity is defined over *annotated* configurations contrary to the barbed congruence. This is bypassed by simply considering *annotated typed relations*, written

$$\Omega \vDash M \ {}_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N} \ N$$

Using those annotated relations, we can define again a notion of contextuality: the only difference with the definition 3.3 is the fact that contexts of the form $[\cdot] \mid l[\![P]\!]$ can be used with the hypothesis $\Omega \vDash M \ {}_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N} \ N$ only when none of the free names of that context are in the annotations $\tilde{a}_M$ and $\tilde{a}_N$. This is a mere consequence of the fact that the names in the annotations are still hidden to the observer.

To obtain an annotated barbed congruence, we should also define the barbs of annotated configurations. But the possible interactions of $\Omega \rhd_{\tilde{a}} M$ are exactly the interactions of $\Omega \rhd M$ since the names $\tilde{a}$ are hidden to the observer so $\Omega \rhd M$ will never show a barb on some name a when $a \in \tilde{a}$. This annotated contextuality therefore immediately induces an *annotated loyal barbed congruence*, written $\cong^{al}$.

By definition, $\cong^{al}$ is thus the biggest relation verifying some conditions which never modify the annotations. Because of this, we directly obtain that $\cong^{l}$ is equal to ${}_{\emptyset}{\cong^{al}}_{\emptyset}$. Now we can prove that $\cong^{al}$ and $\approx^{al}$ coincide by proving both inclusions.

4.3.2 The bisimilarity is included in the barbed congruence

The proof of this inclusion is mainly the proof of the fact that the bisimilarity is contextual. This is naturally done by checking all three items defining contextuality, the major property to check being:

Theorem 4.2 (Bisimilarity is closed on parallel contexts)**.** *If* $\Omega \vDash M \ {}_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N} \ N$*,* $l \in \mathcal{R}_\Omega$*,* $\Omega \vdash l[\![O]\!]$ *and* $\mathsf{fn}(O) \cap (\tilde{a}_M \cup \tilde{a}_N) = \emptyset$ *then* $\Omega \vDash M \mid l[\![O]\!] \ {}_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N} \ N \mid l[\![O]\!]$*.*

Sketch of proof 1. To get this result we simply build a relation and prove that it is a bisimulation which induces the fact that it is included in the biggest bisimulation, $\approx^{al}$. Because that relation must be closed on reductions, we will consider a relation $\mathcal{S}$ in which systems have a very general form:

$$\Omega \vDash (\mathsf{new}\,\Phi_M)(M \mid \prod_i l_i[\![O_i]\!]) \;{}_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N}\; (\mathsf{new}\,\Phi_N)(N \mid \prod_i l_i[\![O_i]\!])$$

The main difficulty to tackle is the fact that, along reductions, the knowledge of the observer, initially completely located in Ω (because $l[\![O]\!]$ is well-typed in Ω), is split between Ω and $\prod_i l_i[\![O_i]\!]$. In particular, a part of the environments Φ and annotations $\tilde{a}$ should be included in the general knowledge of the observer since they might have been communicated to the processes O_i. A precise account of this knowledge is kept to preserve the full-strength of the initial hypothesis of bisimilarity between M and N. So we will impose the following conditions on the relation $\mathcal{S}$.

- There exists some environment Ω_O that can be split into two parts (where X stands for both M and N):
  - a subtype environment of Ω (that can contain some knowledge about $\tilde{a}_X$ but none about the names in Φ_X);
  - a supertype environment of Φ_X.

  The way Ω_O is split into those two pieces is, in general, different for M and N.

- There exists some set of names $\mathcal{N}_O$ that contains all the names which are known to the observer. That set contains in particular all the names in Ω_O and all the names appearing in O_i.

- $\Omega_O; p_1 : \star \mapsto l_1, \ldots \vDash M \;{}_{\tilde{a}^i_M}{\approx^{al}}_{\tilde{a}^i_N}\; N$ where the passports p_i are fresh and $\tilde{a}^i_X$ are the names actually hidden to the observer ($\tilde{a}^i_X = (\mathsf{dom}(\Phi_X) \cup \tilde{a}_X) \setminus \mathcal{N}_O$). The passports p_i represent the fact that the observer has access, via the processes O_i, to the locations l_i.

- The names which are bound in $\mathsf{dom}(\Phi_X) \cup \tilde{a}_X$ and which are known to the observer (*i.e.* are in $\mathcal{N}_O$) must be the same for M and N. Formally, $(\mathsf{dom}(\Phi_M) \cup \tilde{a}_M) \cap \mathcal{N}_O = (\mathsf{dom}(\Phi_N) \cup \tilde{a}_N) \cap \mathcal{N}_O$.

- Finally, the processes O_i controlled by the observer must use only permissions he managed to get so $\Omega_O \vdash \prod_i l_i[\![O_i]\!]$.

The complete proof that $\mathcal{S}$ is indeed a bisimulation (up to structural congruence to tidy terms) is rather tiring and decomposed as usual: for every action performed by a system, we identify which parts are actually involved (M alone; $\prod_i l_i[\![O_i]\!]$ alone; or a communication between those two). The major care that must be taken in checking this property is to preserve the annotations.

The other property that requires some care is the proof that bisimilarity is closed on contexts of the form $(\mathsf{new}\,a : \mathsf{E})[\cdot]$. Again, annotations makes this more complex: the scope of the name a can be opened without revealing it to the observer, namely by adding it to the annotation. The result is then obtained by considering the two possibilities for the place where a is bound ($(\mathsf{new}\,a : \mathsf{E})$ or annotation) for both M and N.

We do not insist any further here on the proofs that $\approx^{al}$ verifies the other defining properties of $\cong^p$, which allow to conclude:

Theorem 4.3. $\approx^{al} \subseteq \cong^{p}$

4.3.3 The barbed congruence is included in the bisimilarity

The proof of the converse involves another equivalence relation to be used as an intermediary: the choice of that equivalence must facilitate both the proof of its inclusion into $\approx^{al}$ and the proof that it includes $\cong^{al}$. The relation we used is a *parallel congruence*, namely the biggest relation which is closed on parallel context and extension of the observer's environment but not on contexts of the form $(\mathsf{new}\, a : \mathsf{E})[\cdot]$. The result that the parallel barbed congruence contains the normal barbed congruence is then immediate. So we simply have to prove that the parallel congruence is included in the bisimilarity.

The guiding idea of the definition of the actions was to identify all the possible interactions between a system and its observer. So the proof of that inclusion can be based on the definition of contexts that characterise a given action of the system. Those contexts use the fact that we can put any environment Γ in a normal form looking like:

$$\begin{array}{l} w_1 : \text{LOC}, \ldots, w_m : \text{LOC}, \\ \qquad u_1 : \tilde{w}_{i_1} \mapsto w_{i_1}, \ldots, u_n : \tilde{w}_{i_n} \mapsto w_{i_n}, \\ \qquad\qquad v_1 : \mathsf{C}_1{@}w_{j_1}, \ldots, v_o : \mathsf{C}_o{@}w_{j_o} \end{array}$$

where

- the w_k are all distinct;
- $u_k = u_{k'}$ only if $k = k'$ or if $w_{i_k} \neq w_{i_{k'}}$;
- $v_k = v_{k'}$ only if $k = k'$ or if $w_{j_k} \neq w_{j_{k'}}$.

This follows from two facts: types can depend only on location identifiers so that all the locations can be listed first; the well-formedness of environments ensures that any two types associated with a given identifier must be weakly $\downarrow$-compatible.

This normal form of environments is relevant for the contexts that characterise the actions of a system because they provide a way to encode every environment into a value of the calculus.

Definition 4.4 (Reification of environments)**.** *To an environment of the following (normal) form*

$$\begin{array}{l} w_1 : \text{LOC}, \ldots, w_m : \text{LOC}, \\ \qquad u_1 : \tilde{w}_{i_1} \mapsto w_{i_1}, \ldots, u_n : \tilde{w}_{i_n} \mapsto w_{i_n}, \\ \qquad\qquad v_1 : \mathsf{C}_1{@}w_{j_1}, \ldots, v_o : \mathsf{C}_o{@}w_{j_o} \end{array}$$

we associate the value

$$V_\Gamma = ((w_1, \ldots, w_m), (u_1, \ldots, u_n, v_1, \ldots, v_o))$$

of type

$$\mathsf{T}_\Gamma = \sum x_1, \ldots, x_m : \widetilde{\text{LOC}}.\ \tilde{x}^{\star}_{i_1} \mapsto x_{i_1}, \ldots, \tilde{x}^{\star}_{i_n} \mapsto x_{i_n}, \mathsf{C}_1{@}x_{j_1}, \ldots, \mathsf{C}_o{@}x_{j_o}$$

Proposition 4.5 (Soundness of the reification)**.** *For any well-formed environment* Γ *and any location* w *defined in* Γ*,* $\Gamma \vdash_w V_\Gamma : \mathsf{T}_\Gamma$*.*

Thanks to this reification of environments, we can proceed as usual, namely we can define some system $\mathfrak{C}^{\Omega}_{\mathcal{N}}((\tilde{b})c!U)$ so that $M \,|\, \mathfrak{C}^{\Omega}_{\mathcal{N}}((\tilde{b})c!U)$ will be sending the value $V_{\Omega,\langle U:\Omega^r(c)\rangle}$ on some specific channel ω if and only if the system M has actually sent the message U over the channel c to $\mathfrak{C}^{\Omega}_{\mathcal{N}}((\tilde{b})c!U)$. So such a context would be of the form $l[\![O]\!]$ where l is the location of the channel c in which the action takes place. Note that the observer can launch some process in l since the action $(\tilde{b})c!U$ is visible to the observer Ω: by rule (LTS-W) this implies that l is in $\mathcal{R}_\Omega$. Then O performs the following steps.

1. It waits for a message on the channel c and, in parallel, exhibit a barb on some special channel δ.

2. It checks that the received value matches the expected U: this relies on the possibility to test the equality and inequality of names; in particular, to check that the names in $\tilde{b}$ are indeed fresh, the context is parameterised with a finite set of existing names $\mathcal{N}$ which contains all the names that are known to the observer. This test matches exactly the definition of the set $\tilde{b}$ in output actions: this set contains only the names which were hidden within the system or the annotation and which are revealed to the observer.

3. It finally cancels the barb on δ and outputs the value $V_{\Omega,\langle U:\Omega^r(c)\rangle}$ on the channel ω.

The channel δ used in the context serves only one purpose: to check that the step 2 has actually been performed: since the detected barbs always allow some preliminary τ transitions, the barb on ω is visible since the very beginning as soon as the system *can* perform the action.

By a very similar technique, it is possible to form contexts that characterise an input action, so that the following theorem can be proved:

Theorem 4.6. $\cong^p \subseteq \approx^{al}$

Sketch of proof 2. We simply prove that $\cong^p$ is a bisimulation. For this consider $\Omega \vDash M \;{}_{\tilde{a}_M}{\cong^p}_{\tilde{a}_N}\; N$. When the configuration $\Omega \rhd_{\tilde{a}_M} M$ performs a τ action to $\Omega \rhd_{\tilde{a}_M} M'$, Theorem 4.1 and the closure of $\cong^p$ on reductions allows to find a N' such that $\Omega \vDash M' \;{}_{\tilde{a}_M}{\cong^p}_{\tilde{a}_N}\; N'$.

For the action $\Omega \rhd_{\tilde{a}_M} M \xrightarrow{\alpha} \Omega' \rhd_{\tilde{a}'_M} M'$, we know that $M \,|\, \mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)$ can reduce into some system $(\mathsf{new}\,\Phi_M)\, M' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$. By contextuality and closure on reductions, $N \,|\, \mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)$ should reach an equivalent state, with a barb on ω and no barb on δ. By definition of the context $\mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)$, that equivalent state must be of the form $(\mathsf{new}\,\Phi_N)\, N' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$ with $\Omega \rhd_{\tilde{a}_N} N \overset{\alpha}{\Longrightarrow} \Omega' \rhd_{\tilde{a}'_N} N'$.

A fairly standard *scope extrusion lemma* bridges the last gap by concluding $\Omega' \vDash M' \;{}_{\tilde{a}'_M}{\cong^p}_{\tilde{a}'_N}\; N'$ from

$$\lambda, \omega, \pi \vDash (\mathsf{new}\,\Phi_M)\, M' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!] \;{}_{\tilde{a}'_M}{\cong^p}_{\tilde{a}'_N}\; (\mathsf{new}\,\Phi_N)\, N' \,|\, \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$$

where: π is a universal passport to λ, $\tilde{a}'_M$ is $(\tilde{a}_M \cup \mathsf{dom}(\Phi_M)) \setminus \mathsf{dom}(\Omega)$ and a similar formula for $\tilde{a}'_N$.

The results stated above directly entails the expected result:

Theorem 4.7 (Full abstraction of $\approx^{al}$ for $\cong^{l}$)**.** *$\Omega \models M \cong^{l} N$ if and only if $\Omega \models M\ {}_{\emptyset}{\approx^{al}}_{\emptyset}\ N$*

5 Conclusion & perspectives

This work presents a new approach to control the migrations of agents in the context of distributed computation, using simple passports that should correspond to the origin location of the migrating agent. We have developed the full theory of this idea, with a loyal barbed congruence that takes those passports into account to distinguish between systems. We have also provided a complete proof technique for this equivalence as a bisimilarity.

This work provides a solid ground on which to investigate subtler notions of security like the ones presented in [HRY05] and [CHP^{+}06]. We already started to study more complex passports in which resources that can be accessed after the migration depend on the passport actually used: when a new passport is generated, its type also embed all the rights to be granted to incoming processes.

Acknowledgement The author would like to thank Matthew Hennessy for numerous helpful discussions and comments.

References

[BCMS02] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication interference in mobile boxed ambients. In Manindra Agrawal and Anil Seth, editors, *FSTTCS*, volume 2556 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2002.

[BS98] Michele Boreale and Davide Sangiorgi. Bisimulation in name-passing calculi without matching. In *Thirteenth Annual Symposium on Logic in Computer Science (LICS) (Indiana)*. IEEE, Computer Society Press, July 1998.

[CHP^{+}06] Karl Crary, Robert Harper, Frank Pfenning, Benjamin C. Pierce, Stephanie Weirich, and Stephan Zdancewic. Manifest security for distributed information. White paper, March 2006.

[CN02] Giuseppe Castagna and Francesco Zappa Nardelli. The Seal calculus revisited: Contextual equivalence and bisimilarity. In Manindra Agrawal and Anil Seth, editors, *FSTTCS*, volume 2556 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2002.

[HH07] Samuel Hym and Matthew Hennessy. Adding recursion to Dpi. *Theoretical Computer Science*, 373(3):182–212, apr 2007.

[HMR03] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322:615–669, 2003.

[HR02] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[HRY05] Matthew Hennessy, Julian Rathke, and Nobuko Yoshida. SafeDpi: a language for controlling mobile code. *Acta Informatica*, 42(4-5):227–290, 2005.

[Hym06] Samuel Hym. *Typage et contrôle de la mobilité*. PhD thesis, Université Paris Diderot – Paris 7, 2006.

[LS00] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *27th Annual Symposium on Principles of Programming Languages (POPL) (Boston, MA)*, pages 352–364. ACM, January 2000.

[MH06] Massimo Merro and Matthew Hennessy. A bisimulation-based semantic theory of Safe Ambients. *ACM Transactions on Programming Languages and Systems*, 28(2):290–330, March 2006.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[MV05] Francisco Martins and Vasco Thudichum Vasconcelos. History-based access control for distributed processes. In *TGC*, pages 98–115, 2005.

[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, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2004.

[WF94] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, November 1994.

A Complete formal definitions

We provide here figures containing the complete sets of inference rules of the type system.

Figure 7 Well-formed environments

$$(\textsc{e-empty}) \qquad \vdash \mathbf{env}$$

$$(\textsc{e-loc}) \qquad \frac{\Gamma \vdash \mathbf{env}}{\Gamma, u : \textsc{loc} \vdash \mathbf{env}} \downarrow(\Gamma(u) \cup \{\textsc{loc}\})$$

$$(\textsc{e-channel}) \qquad \frac{\Gamma \vdash \mathbf{env} \qquad w : \textsc{loc} \in \Gamma}{\Gamma, u : \mathsf{C}@w \vdash \mathbf{env}} \downarrow(\Gamma(u) \cup \{\mathsf{C}@w\})$$

$$(\textsc{e-pass}) \qquad \frac{\Gamma \vdash \mathbf{env} \qquad w : \textsc{loc} \in \Gamma \qquad \forall w_i \in \tilde{w}, w_i : \textsc{loc} \in \Gamma}{\Gamma, u : \tilde{w}^\star \mapsto w \vdash \mathbf{env}} \downarrow(\Gamma(u) \cup \{\tilde{w}^\star \mapsto w\})$$

Figure 8 Typing values

$$(\textsc{v-id}) \qquad \frac{\Gamma, u : \mathsf{E}, \Gamma' \vdash \mathbf{env}}{\Gamma, u : \mathsf{E}, \Gamma' \vdash u : \mathsf{E}} \qquad\qquad \frac{\Gamma \vdash u : \mathsf{E}_1 \quad \Gamma \vdash u : \mathsf{E}_2}{\Gamma \vdash u : \mathsf{E}_3} \; \mathsf{E}_1 \sqcap \mathsf{E}_2 <: \mathsf{E}_3 \qquad (\textsc{v-inf})$$

$$(\textsc{v-localization}) \qquad \frac{\Gamma \vdash w : \textsc{loc} \quad \Gamma \vdash u : \mathsf{E}}{\Gamma \vdash_w u : \mathsf{E}}$$

$$(\textsc{v-located-chan}) \qquad \frac{\Gamma \vdash w : \textsc{loc} \quad \Gamma \vdash u : \mathsf{C}@w}{\Gamma \vdash_w u : \mathsf{C}} \qquad\qquad \frac{\Gamma \vdash u : w \mapsto v}{\Gamma \vdash_w u : \; \hookrightarrow v} \qquad (\textsc{v-located-pass})$$

$$(\textsc{v-tuple}) \qquad \frac{\Gamma \vdash_w V_i : \mathsf{T}_i}{\Gamma \vdash_w (\vec{V}) : (\vec{\mathsf{T}})} \qquad\qquad \frac{\Gamma \vdash_w u_i : \textsc{loc} \quad \Gamma \vdash_w V : \mathsf{T}\{{}^{\vec{u}}/_{\vec{x}}\}}{\Gamma \vdash_w ((\vec{u}), V) : \sum \vec{x} : \vec{\textsc{loc}}.\, \mathsf{T}} \qquad (\textsc{v-dep})$$

Figure 9 Typing processes

$$\textsc{(t-w)}\quad \frac{\Gamma \vdash_w u : \textsc{w}\langle \mathsf{T} \rangle \qquad \Gamma \vdash_w V : \mathsf{T} \qquad \Gamma \vdash_w P}{\Gamma \vdash_w u\,!\,\langle V \rangle\, P}$$

$$\textsc{(t-r)}\quad \frac{\Gamma \vdash_w u : \textsc{r}\langle \mathsf{T} \rangle \qquad \Gamma; \langle X : \mathsf{T} \rangle\, {\scriptstyle @} w \vdash_w P}{\Gamma \vdash_w u\,?\,(X : \mathsf{T})\, P}$$

$$\textsc{(t-goto)}\quad \frac{\Gamma \vdash u : w \mapsto v \qquad \Gamma \vdash_v P}{\Gamma \vdash_w \mathsf{goto}_u\, v.\, P}$$

$$\textsc{(t-if)}\quad \frac{\Gamma \vdash_w P_2 \qquad [\Gamma]_{u_1 = u_2} \vdash_w P_1 \text{ when } [\Gamma]_{u_1 = u_2} \vdash \mathbf{env}}{\Gamma \vdash_w \mathsf{if}\ u_1 = u_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2}$$

$$\textsc{(t-newchan)}\quad \frac{\Gamma; c : \mathsf{C}{\scriptstyle @} w \vdash_w P}{\Gamma \vdash_w \mathsf{newchan}\, c : \mathsf{C}\ \mathsf{in}\ P}$$

$$\textsc{(t-newloc)}\quad \frac{\Gamma; \langle (k, ((\vec{c}), (\vec{p}), (\vec{q}))) : \mathsf{T}\{{}^{w}\!/\!_{x}\} \rangle \vdash_k P_k \qquad \Gamma; \langle (k, ((\vec{c}), (\vec{p}), (\vec{q}))) : \mathsf{T}\{{}^{w}\!/\!_{x}\} \rangle \vdash_w P}{\Gamma \vdash_w \mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \sum x : \textsc{loc}.\ \mathsf{T}\ \mathsf{with}\ P_k\ \mathsf{in}\ P}$$

$$\textsc{(t-newpass)}\quad \frac{\Gamma \vdash \tilde{u} : \widetilde{\textsc{loc}} \qquad \Gamma; p : \tilde{u}^{\star} \mapsto w \vdash_w P}{\Gamma \vdash_w \mathsf{newpass}\, p\ \mathsf{from}\ \tilde{u}^{\star}\ \mathsf{in}\ P}$$

$$\textsc{(t-par)}\quad \frac{\Gamma \vdash_w P_1 \qquad \Gamma \vdash_w P_2}{\Gamma \vdash_w P_1 \mid P_2}$$

$$\textsc{(t-rep)}\quad \frac{\Gamma \vdash_w P}{\Gamma \vdash_w *P}$$

$$\textsc{(t-stop)}\quad \frac{\Gamma \vdash \mathbf{env}}{\Gamma \vdash_w \mathsf{stop}}$$

Figure 10 Typing systems

$$\textsc{(t-nil)}\quad \frac{\Gamma \vdash \mathbf{env}}{\Gamma \vdash \mathbf{0}} \qquad\qquad \textsc{(t-new)}\quad \frac{\Gamma; a : \mathsf{E} \vdash M}{\Gamma \vdash (\mathsf{new}\, a : \mathsf{E})\, M}$$

$$\textsc{(t-proc)}\quad \frac{\Gamma \vdash l : \textsc{loc} \qquad \Gamma \vdash_l P}{\Gamma \vdash l[\![P]\!]} \qquad\qquad \textsc{(t-s-par)}\quad \frac{\Gamma \vdash M_1 \qquad \Gamma \vdash M_2}{\Gamma \vdash M_1 \mid M_2}$$