Mobility control via passports

Samuel Hym

Samuel.Hym@ens-lyon.fr

PPS, Université Paris Diderot & CNRS

Université de Lyon, LIP, ENS Lyon & Université Claude Bernard Lyon 1 & INRIA & CNRS

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 major modifications to the 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.

Key words: process calculus, control of agent migrations, distributed computation, observational equivalence

1 Introduction

Dπ [1] 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 trigger 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. Those passports are permits, 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. A very generic idea has been developed in [2]: every location is composed of two layers, a *membrane* and the actual body. Every migration attempted must then go through the membrane in which some controls are performed to implement an access policy. A few such policies were considered for instance in [3]. In the setting we propose here, we allow normal processes to control the access policies themselves by delivering passports when needed: the only part of the control of migrations that appear in the syntax of processes is precisely those passports.

In Ambients-related calculi, the migrations are particularly hard to control so many works tried to address this problem: in Safe Ambients [4], the destination location must grant access to incoming ambients by using a *co-capability*. These co-capabilities have been enriched in [5] with *passwords*: the password used to migrate is syntactically checked at *runtime* when the migration is to be granted. This idea of passwords was pursued in the NBA calculus [6] 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 can be removed without impeding the expressiveness of the calculus. This second approach was also used in different hierarchical calculi like Seal [7] or Kell [8]. Another way to control mobility is to rely on type systems. In Mobile Ambients, [9] introduces a type system to characterise which ambients might end up in which ambients and which ambients might be opened inside another ambient. To avoid the complexity of dependent types, this work uses *groups*: when a name is generated, it is attributed a group; every mobility control is then performed at the level of

groups. [10] developed this control with a setting in which processes are allowed to move between ambients and the type system indicates in which groups of ambients a process or an ambient might end up. Our approach, on the other hand, allows processes running inside a location to control the processes trying to enter.

Hierarchy brings obviously a lot of complexity to the migration schemes so there are some type systems implementing finer policies in non-hierarchical calculi. For instance, KLAIM has been equipped in [11] with a type system to control access to resources, including code migration. Contrary to our setting, in that work location policies control where the code goes to, and not comes from, and the policy is fixed when the location is generated. The present work has been more inspired by [12]. In that work, access to a location is a capability tied to that location via its type: access is either always granted or always denied depending on the type used when the location is generated. Of course, even when access is granted, the location name can then be transmitted without giving access; nevertheless, this setting lacks flexibility. In the present work, we refine that approach to be able to grant access selectively, depending on the origin location and to authorise such access migrations *dynamically*, namely after the generation of the location itself. That is why *passport* names are added to the calculus to bear those authorisations. We chose to use regular names to preserve the *homogeneity* of the calculus: in particular, they can be exchanged over channels and their scopes are dealt with in precisely the same way as any other name, including for their extrusions. Types are then used to tie rights to the names of the passports: 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. Since a passport must grant access to only *one* location, the one which delivered that passport, using "groups" to try and avoid dependent types would fall back on defining one group per location and so it would only reduce the expressiveness of the language. Fortunately, those dependent types bring little extra complexity to the type system itself and to the proofs of its properties, including subject reduction. What is more, this approach to tie rights to types provides type-based tools and techniques to reason about security properties. We also argue that relying on names to bear access rights gives a good handle to control those rights.

Other type systems have been used to control mobility in Dπ-based calculi. In [13], access requires the knowledge of a *port* which also governs subsequent resource accesses by typing the migrating processes, using for this complex behavioural process types developed in particular in [14]. This approach is strongly constraining processes and requires higher order actions. The present work provides a first-order theory that aims at becoming a foundation for a fine-grained control of comparable power to [13]: while the passport types

developed hereafter correspond to a simple mobility control, they should leave room to extensions to control resource accesses.

In [15], access to locations and resources is conditioned by policies based on the history of migrations of the agent. In the present work, 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 setting we propose here relies on a simple view of trust: when a location l expresses its trust into another one k (through a passport valid from k), it also decides to trust k not to relay any dangerous process from another location.

In the following, we will investigate the notion of typed observational equivalence inherited from [16]. 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 it 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. We argue that relying on names to bear access rights also gives a clean equivalence theory, in which the rights granted to the observer are easily expressed. 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.

An extended abstract of the present work has previously been published at CONCUR [17].

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 [18]. An account of the complete calculus is presented in [19]. Let us start by looking at passports on examples.

2.1 Overview of passports

Dπ describes distributed computation as explicitly located processes that 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 than 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:

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

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:

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

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:

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

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:

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

where the optional passports $p_{l_i}^{gw}$ 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 $p_{l_i}^{gw}$ 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:

Fig. 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, 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 using p will be granted access from any location in $\tilde{u}$;
  - *universal passports*, that allow migration from any location (for instance when describing the behaviour of a public server accepting requests from anywhere) 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 is the only way to model all the possible situations (the child location granting access to processes from the

Fig. 2 Reduction semantics

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

$$\text{(R-NEWLOC)}\quad l[\![\mathsf{newloc}\, k, (\vec{c}), (\vec{p}), (\vec{q}) : \textstyle\sum x : \mathrm{LOC}.\ \mathsf{T}\ \mathsf{with}\ P_k\ \mathsf{in}\ P]\!] \longrightarrow (\mathsf{new}\left\langle k, ((\vec{c}), (\vec{p}), (\vec{q})) : \mathsf{T}\{{}^{l}/_{x}\}\right\rangle)(k[\![P_k]\!] \mid l[\![P]\!])$$

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

$$\text{(R-COMM)}\quad l[\![a\,!\,\langle V\rangle\ P_1]\!] \mid l[\![a\,?\,(X : \mathsf{T})\ P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2\{{}^{V}/_{X}\}]\!]$$

$$\text{(R-IF-V)}\quad l[\![\mathsf{if}\ a = a\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2]\!] \longrightarrow l[\![P_1]\!]$$

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

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

$$\text{(R-SPLIT)}\quad l[\![P_1 \mid P_2]\!] \longrightarrow l[\![P_1]\!] \mid l[\![P_2]\!]$$

$$\text{(R-REP)}\quad l[\![*P]\!] \longrightarrow l[\![P]\!] \mid l[\![*P]\!]$$

$$\text{(R-C-PAR)}\quad \frac{M_1 \longrightarrow M_1'}{M_1 \mid M_2 \longrightarrow M_1' \mid M_2} \qquad \text{(R-C-NEW)}\quad \frac{M_1 \longrightarrow M_1'}{(\mathsf{new}\, a : \mathsf{E})\, M_1 \longrightarrow (\mathsf{new}\, a : \mathsf{E})\, M_1'}$$

$$\text{(R-STRUCT)}\quad \frac{M_1 \equiv M_2 \longrightarrow M_2' \equiv M_1'}{M_1 \longrightarrow M_1'}$$

mother location; or vice versa; and any other variation). Indeed, 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.

The new reduction rules are fairly unsurprising since passports are added to the calculus as regular identifiers. 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 the inherited (R-NEWCHAN) rule, for instance, when channels are generated their type C becomes the explic-

itly located type $\mathsf{C}_{@l}$; similarly, when passports are actually generated in (R-NEWPASS), they are tied to the location to which they will grant access, by getting the type $\tilde{k} \mapsto l$ (type for a passport allowing migrations 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 sequentialises 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 orthogonal to passports types; see [18] 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

Fig. 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
$\tilde{u} \mapsto v$	Passport from $\tilde{u}$ to v
$\star \mapsto v$	Universal passport 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
$\sum x : \textsc{loc}.\ \mathsf{T}$	Dependent sum
$\mathsf{L} ::=$	*Types to declare locations*
$\sum x : \textsc{loc}.\ \sum y : \textsc{loc}.\ (\mathsf{C}_{1@}y, \ldots), (\tilde{u}_1^\star \mapsto y, \ldots), (\tilde{v}_1^\star \mapsto x, \ldots)$	

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 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 k) 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:

Fig. 4 Expansion of values

$$\langle u : \mathsf{C} \rangle_{@w} = u : \mathsf{C}_{@w}$$

$$\langle u : \mathsf{C}_{@w_0} \rangle_{@w} = u : \mathsf{C}_{@w_0}$$

$$\langle u : \text{LOC} \rangle_{@w} = u : \text{LOC}$$

$$\langle ((\vec{u}), V) : \sum \vec{x} : \vec{\text{LOC}}.\ \mathsf{T} \rangle_{@w} = u_1 : \text{LOC}, \ldots, \left\langle V : \mathsf{T}\{\vec{u}/\vec{x}\} \right\rangle_{@w}$$

$$\langle (V_1, \ldots, V_n) : (\mathsf{T}_1, \ldots, \mathsf{T}_n) \rangle_{@w} = \langle V_1 : \mathsf{T}_1 \rangle_{@w}, \ldots, \langle V_n : \mathsf{T}_n \rangle_{@w}$$

$$\langle u : \tilde{v}^\star \mapsto v' \rangle_{@w} = u : \tilde{v}^\star \mapsto v'$$

$$\langle u : \ \hookrightarrow v' \rangle_{@w} = u : w \mapsto v'$$

in the types of the form

$$\mathsf{L} = \sum x : \text{LOC}.\ \sum y : \text{LOC}.\ (\mathsf{C}_{1@}y, \ldots), (\tilde{u}_1^\star \mapsto y, \ldots), (\tilde{v}_1^\star \mapsto x, \ldots)$$

the 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 $\sum x : \text{LOC}.\ \mathsf{T}$, 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* $\left\langle k, ((\vec{c}), (\vec{p}), (\vec{q})) : \mathsf{T}\{l/x\} \right\rangle$.

The expansion is formally described in Figure 4: $\langle V : \mathsf{T} \rangle_{@w}$ 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 corresponds 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 $_{@w}$). 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.

The fact that dependent sums can bind variables in a type means that we will always consider types up to α-renaming of these variables. That binding property of variables in types plays 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, $\text{R}\langle\text{R}\langle\rangle_{@}x\rangle_{@}k$ 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 com-

munication. 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 $\text{RW}\langle \mathsf{T}^r, \mathsf{T}^w \rangle$. 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:

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

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

The rule for dependent sums is even simpler:

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

The other subtyping rules are standard (we condensed all the rules for channel types into one for conciseness):

$$\text{(SR-LOC)} \quad \text{LOC} <: \text{LOC}$$

$$\text{(SR-TUPLE)} \quad \frac{\mathsf{T}_i <: \mathsf{T}'_i \text{ pour tout } i}{(\vec{\mathsf{T}}) <: (\vec{\mathsf{T}'})} \qquad \text{(SR-LOC-CHANNEL)} \quad \frac{\mathsf{C}_1 <: \mathsf{C}_2}{\mathsf{C}_{1@u} <: \mathsf{C}_{2@u}}$$

$$\text{(SR-CHANNEL)} \quad \frac{\mathsf{T}_2^w <: \mathsf{T}_1^w <: \mathsf{T}_1^r <: \mathsf{T}_2^r}{\begin{array}{c}\text{RW}\langle \mathsf{T}_1^r, \mathsf{T}_1^w\rangle <: \text{RW}\langle \mathsf{T}_2^r, \mathsf{T}_2^w\rangle \\ \text{RW}\langle \mathsf{T}_1^r, \mathsf{T}_1^w\rangle <: \text{W}\langle \mathsf{T}_2^w\rangle \\ \text{RW}\langle \mathsf{T}_1^r, \mathsf{T}_1^w\rangle <: \text{R}\langle \mathsf{T}_2^r\rangle \\ \text{W}\langle \mathsf{T}_1^w\rangle <: \text{W}\langle \mathsf{T}_2^w\rangle \\ \text{R}\langle \mathsf{T}_1^r\rangle <: \text{R}\langle \mathsf{T}_2^r\rangle\end{array}}$$

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

Definition 1 (Types) *A pre-type* T *is a* type *when:*

- *for every occurrence of a type of the form* $\text{RW}\langle \mathsf{T}^r, \mathsf{T}^w\rangle$ *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. In particular, we preserve the partial meets property. This property bears on $\downarrow$-compatible types: we say that any two types T_1 and T_2 are $\downarrow$-compatible, written $\mathsf{T}_1 \downarrow \mathsf{T}_2$, when they share a common subtype. So the property of partial meets can be stated as:

Theorem 2 (Partial meets) *Any two* $\downarrow$*-compatible types have a meet.*

Proof: The property of partial meets is well-known for the standard type system in Dπ (see [1]). Let us then consider passport types: 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. Dependent sums are also easy to deal with since the dependency can only bear on variables of type LOC. □

Fig. 5 Well-formed environments

$$\text{(E-EMPTY)}\quad \vdash \mathbf{env}$$

$$\text{(E-LOC)}\quad \frac{\Gamma \vdash \mathbf{env}}{\Gamma, u : \textsc{loc} \vdash \mathbf{env}} \downarrow(\Gamma(u) \cup \{\textsc{loc}\})$$

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

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

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 associating 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:

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

  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 this means that the set of all those types must have a meet which sums up all the rights granted on u.

Fig. 6 Typing values

$$\text{(V-ID)}\ \frac{\Gamma, u : \mathsf{E}, \Gamma' \vdash \mathbf{env}}{\Gamma, u : \mathsf{E}, \Gamma' \vdash u : \mathsf{E}} \qquad \frac{\Gamma \vdash u : \mathsf{E}_1 \quad \Gamma \vdash u : \mathsf{E}_2}{\Gamma \vdash u : \mathsf{E}_3}\ \mathsf{E}_1 \sqcap \mathsf{E}_2 <: \mathsf{E}_3\ \text{(V-INF)}$$

$$\text{(V-LOCALIZATION)}\ \frac{\Gamma \vdash w : \textsc{loc} \quad \Gamma \vdash u : \mathsf{E}}{\Gamma \vdash_w u : \mathsf{E}}$$

$$\text{(V-LOCATED-CHAN)}\ \frac{\Gamma \vdash w : \textsc{loc} \quad \Gamma \vdash u : \mathsf{C}@w}{\Gamma \vdash_w u : \mathsf{C}} \qquad \frac{\Gamma \vdash u : w \mapsto v}{\Gamma \vdash_w u : \hookrightarrow v}\ \text{(V-LOCATED-PASS)}$$

$$\text{(V-TUPLE)}\ \frac{\Gamma \vdash_w V_i : \mathsf{T}_i}{\Gamma \vdash_w (\vec{V}) : (\vec{\mathsf{T}})} \qquad \frac{\Gamma \vdash_w u_i : \textsc{loc} \quad \Gamma \vdash_w V : \mathsf{T}\{\vec{u}/\vec{x}\}}{\Gamma \vdash_w ((\vec{u}), V) : \sum \vec{x} : \vec{\textsc{loc}}.\ \mathsf{T}}\ \text{(V-DEP)}$$

The verification is similar for the other identifier types (see Figure 5).

- Some inference rules then correspond to judgements 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

$$l : \textsc{loc}, \vec{l} : \textsc{loc}, p : l_1, l_2 \mapsto l, p : l_3, l_4 \mapsto l \vdash p : l_1, l_3 \mapsto l$$

  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 judgements 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 (see Figure 6).
- 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:

$$\Gamma \vdash_u P$$

Fig. 7 Typing processes

$$\text{(T-W)}\ \frac{\Gamma \vdash_w u : \mathrm{W}\langle \mathsf{T} \rangle \qquad \Gamma \vdash_w V : \mathsf{T} \qquad \Gamma \vdash_w P}{\Gamma \vdash_w u\,!\,\langle V \rangle\, P}$$

$$\text{(T-R)}\ \frac{\Gamma \vdash_w u : \mathrm{R}\langle \mathsf{T} \rangle \qquad \Gamma; \langle X : \mathsf{T} \rangle\, @w \vdash_w P}{\Gamma \vdash_w u\,?\,(X : \mathsf{T})\, P}$$

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

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

$$\text{(T-NEWCHAN)}\ \frac{\Gamma; c : \mathsf{C}@w \vdash_w P}{\Gamma \vdash_w \mathsf{newchan}\, c : \mathsf{C}\, \mathsf{in}\, P}$$

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

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

$$\text{(T-PAR)}\ \frac{\Gamma \vdash_w P_1 \qquad \Gamma \vdash_w P_2}{\Gamma \vdash_w P_1 \mid P_2}$$

$$\text{(T-REP)}\ \frac{\Gamma \vdash_w P}{\Gamma \vdash_w *P}$$

$$\text{(T-STOP)}\ \frac{\Gamma \vdash \mathbf{env}}{\Gamma \vdash_w \mathsf{stop}}$$

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.

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

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

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.

Notice that, in (T-NEWPASS), the environment in which P is checked is obtained by a combination of Γ and p written using ";" instead of ",": we use, in the following, semi-colons to state that the domains of definition of the two combined environments are completely disjoint.

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.

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

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

$$\sum x : \textsc{loc}.\ \sum y : \textsc{loc}.\ (\mathsf{C}_{1@}y,\ldots),(\tilde{u}_1^{\star} \mapsto y,\ldots),(\tilde{v}_1^{\star} \mapsto x,\ldots)$$

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

Fig. 8 Bracket extension of environments

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

identifiers are identical: we want to merge all the permissions the process owns over those two identifiers. This is performed using $[\cdot]_{u_1=u_2}$ 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 $u_1 : \mathrm{R}\langle\rangle_@l$ and $u_2 : \mathrm{W}\langle\rangle_@l$, $[\Gamma]_{u_1=u_2}$ will contain the four

$$u_1 : \mathrm{R}\langle\rangle_@l \qquad u_2 : \mathrm{R}\langle\rangle_@l \qquad u_2 : \mathrm{W}\langle\rangle_@l \qquad u_1 : \mathrm{W}\langle\rangle_@l$$

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 8. Notice that, when the environment contains a hypothesis $u_1 : \mathsf{C}_@u_2$, 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:

$$\Gamma = p : l_0 \mapsto l, x_l : \mathrm{LOC}, x_p : l_0 \mapsto x_l$$

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

Fig. 9 Typing systems

$$(\text{T-NIL})\ \frac{\Gamma \vdash \mathbf{env}}{\Gamma \vdash \mathbf{0}} \qquad (\text{T-NEW})\ \frac{\Gamma; a : \mathsf{E} \vdash M}{\Gamma \vdash (\mathsf{new}\, a : \mathsf{E})\, M}$$

$$(\text{T-PROC})\ \frac{\Gamma \vdash l : \text{LOC} \qquad \Gamma \vdash_l P}{\Gamma \vdash l[\![P]\!]} \qquad (\text{T-S-PAR})\ \frac{\Gamma \vdash M_1 \qquad \Gamma \vdash M_2}{\Gamma \vdash M_1 \,|\, M_2}$$

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, see Figure 7.

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

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

Again, these rules are completely inherited, see Figure 9.

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

Proof: The form this proof takes is completely routine (see [1]) and is based on substitution lemmas. Only the use of bracket extensions makes this proof more complex than usual, mostly because of the interactions between the substitutions and the extensions. Let us look only at the case when

$$l[\![a\,!\,\langle V\rangle\, P_1]\!] \,|\, l[\![a\,?\,(X : \mathsf{T})\, P_2]\!]$$

evolves by a communication. The interested reader will find in [19] a fully detailed account of this proof of subject reduction.

We want to prove $\Gamma \vdash_l P_2\{^V/_X\}$ out of the proof of $\Gamma; \langle X : \mathsf{T}\rangle \circledcirc l \vdash_l P_2$. Note that, when proving this substitution lemma by induction, potential tests in P_2 would end up building hypothesis of the form $[\Gamma']_{u_1=u_2} \vdash_u P_2'$ which breaks the structure of the most natural induction hypothesis. So the substitution lemma takes the following very general form:

Lemma 4 (Substitution in processes) *Let Γ_1 and Γ_2 be two environments, X be a pattern and V a value such that, for every $v : \mathsf{E}$ in Γ_1, $\Gamma_2 \vdash v\{^V/_X\} : \mathsf{E}\{^V/_X\}$ is provable. Then any provable judgement $\Gamma_1 \vdash_u P$ implies that the judgement $\Gamma_2 \vdash_{u\{^V/_X\}} P\{^V/_X\}$ is also provable.*

That lemma relies on the fact that the types of values exchanged over a channel are closed: by Definition 1, every identifier appearing in a type of the form C must be bound. The absence of nested name dependency is critical to make the loose connection between Γ_1 and Γ_2 sufficient.

The lemma is proved by induction on the proof of $\Gamma_1 \vdash_u P$. We look here only at the two most interesting cases. When P is of the form $v\,?\,(X : \mathsf{T})\,P'$, the absence of free variable in T allows to lift the hypothesis on Γ_1 and Γ_2 to $\Gamma_1;\langle X : \mathsf{T}\rangle_{@u}$ and $\Gamma_2;\langle X : \mathsf{T}\rangle_{@(u\{^V/_X\})}$ and consequently apply the induction hypothesis to conclude.

When P is a test of the form $\mathsf{if}\ u_1 = u_2\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2$, the reasoning requires some extra care. This is done in the following two stages.

- We first notice that $[\Gamma_1]_{u_1=u_2}$ must be well-formed when $[\Gamma_2]_{u_1\{^V/_X\}=u_2\{^V/_X\}}$ also is well-formed: the only possible reason for $[\Gamma_1]_{u_1=u_2}$ to be ill-formed is the existence of two incompatible types associated with a unique name. By considering the various cases, we can easily show that this incompatibility would also be present in $[\Gamma_2]_{u_1\{^V/_X\}=u_2\{^V/_X\}}$. Let us look for instance at the case where the incompatibility is generated by two hypotheses $w : \mathsf{C}_1@l_1$ and $w : \mathsf{C}_2@l_2$ in $[\Gamma_1]_{u_1=u_2}$, where $l_1 \neq l_2$. This implies that $(w : \mathsf{C}_1@l_1)\{^V/_X\}$ and $(w : \mathsf{C}_2@l_2)\{^V/_X\}$ could be inferred in $[\Gamma_2]_{u_1\{^V/_X\}=u_2\{^V/_X\}}$ if it were a well-formed environment. Since the substitution $\{^V/_X\}$ leaves names unmodified, those incompatible judgements entail that $[\Gamma_2]_{u_1\{^V/_X\}=u_2\{^V/_X\}}$ is ill-formed.
- Secondly, we lift the induction hypothesis over Γ_1 and Γ_2 to $[\Gamma_1]_{u_1=u_2}$ and $[\Gamma_2]_{u_1\{^V/_X\}=u_2\{^V/_X\}}$ by a simple case analysis of the origin of every hypothesis in $[\Gamma_1]_{u_1=u_2}$.

□

The property of type safety is more relevant in the present work. Let us specialise here the property to bear on passport types only, the result being well-known for the rest of the calculus. Following the approach developed in [20], we define *erroneous reduction* of a system M, written $M \xrightarrow{\text{err}}_\Gamma$, by the following set of rules:

$$
\begin{aligned}
l[\![\mathsf{goto}_p\, k.\ P]\!] &\xrightarrow{\text{err}}_\Gamma \quad \text{if } \Gamma \not\vdash_l p : l \mapsto k \\
M_1 \mid M_2 &\xrightarrow{\text{err}}_\Gamma \quad \text{if } M_1 \xrightarrow{\text{err}}_\Gamma \text{ or } M_2 \xrightarrow{\text{err}}_\Gamma \\
(\mathsf{new}\, a : \mathsf{E})\, M &\xrightarrow{\text{err}}_\Gamma \quad \text{if } M \xrightarrow{\text{err}}_{\Gamma;a:\mathsf{E}}
\end{aligned}
$$

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

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 [21], 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 it got access to, including its passports, using a *type environment* written Ω. This type environment thus describes the observations that can be performed, in a similar way to the knowledge-indexed relations defined in [12]. Following [12], we will 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 $\Gamma <: \Omega$ where $<:$ is extended from types to environment in the following way: we will say that Γ is a subtype of Γ' as soon as every typing judgement that can be inferred in Γ' can also be inferred in Γ.

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 $\Omega \vDash M\ \mathcal{S}\ N$ 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 6 (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 $M \longrightarrow^{*} \equiv (\mathsf{new}\, \vec{a} : \vec{\mathsf{E}})(M' \mid l[\![c\,!\,\langle V \rangle\, P]\!])$.*

Note that the only control performed in this definition is whether the observer is able to reach the location where the interaction takes place: since our mobility control happens only when *entering* a location, it will always be possible to report the observation in the observer's home location.

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 a sequence of passports $p : \star \mapsto l_1$, $p_1 : l_1 \mapsto l_2$, $\ldots$, $p_n : l_n \mapsto l$ 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 7 (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 $w \in \mathcal{R}_\Gamma$.*

The intuition of this notion can be given by this simple property the proof of which is immediate:

Proposition 8 (Prevention of burglaries) *If Γ' is a loyal extension of Γ, $\mathcal{R}_{\Gamma;\Gamma'} \cap \mathsf{dom}(\Gamma) = \mathcal{R}_\Gamma$.*

Finally, we define the loyal contextuality of a relation $\mathcal{S}$.

Definition 9 (Loyally contextual relation) *A relation $\mathcal{S}$ is said* loyally contextual *only when:*

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

Definition 10 (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.*

The contexts considered in this congruence can launch processes in every *reachable* location (to allow more contexts to be used) while barbs can only be observed in *directly reachable* (to get the simplest notion of observations). Note though that the congruence obtained does not depend on this choice: it is simple to see that we could use *reachable barbs* or *directly reachable contexts* and still end up defining the same equivalence (indeed, reachable contexts can be obtained from directly reachable ones with explicit migrations; and reachable barbs are only a matter of using a context to detect the barb and report it in a directly reachable location). Our choice then corresponds to the simplest possible barbs with the greatest possible number of contexts.

4 Loyal bisimilarity

The definition given for the loyal barbed congruence is justified by intuitions but it is highly intractable: every proof of equivalence indeed requires a quantification over all contexts. 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 represent 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 (see [1]): 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

Fig. 10 Labelled transition system, significant rules

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

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

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

$$\text{(LTS-COMM)} \quad \frac{\begin{array}{c}\Omega_M \rhd M \xrightarrow{(\Phi)a!V} \Omega'_M \rhd M' \\ \Omega_N \rhd N \xrightarrow{(\Phi)a?V} \Omega'_N \rhd N'\end{array}}{\begin{array}{l}\Omega \rhd M \mid N \xrightarrow{\tau} \Omega \rhd (\mathsf{new}\,\Phi)\, M' \mid N' \\ \Omega \rhd N \mid M \xrightarrow{\tau} \Omega \rhd (\mathsf{new}\,\Phi)\, N' \mid M'\end{array}}$$

$$\text{(LTS-C-PAR)} \quad \frac{\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'}{\begin{array}{l}\Omega \rhd M \mid N \xrightarrow{\mu} \Omega' \rhd M' \mid N \\ \Omega \rhd N \mid M \xrightarrow{\mu} \Omega' \rhd N \mid M'\end{array}}$$

$$\text{(LTS-C-NEW)} \quad \frac{\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'}{\Omega \rhd (\mathsf{new}\, a : \mathsf{E})\, M \xrightarrow{\mu} \Omega' \rhd (\mathsf{new}\, a : \mathsf{E})\, M'} \; a \notin \mathsf{n}(\mu)$$

$$\text{(LTS-OPEN)} \quad \frac{\Omega \rhd M \xrightarrow{(\Phi)a!V} \Omega' \rhd M' \qquad b \neq a}{\Omega \rhd (\mathsf{new}\, b : \mathsf{E})\, M \xrightarrow{(b:\mathsf{E};\Phi)a!V} \Omega' \rhd M' \quad b \in \mathsf{fn}(V) \cup \mathsf{fn}(\Phi)}$$

$$\text{(LTS-WEAK)} \quad \frac{\Omega; \Omega_e \rhd M \xrightarrow{(\Phi)a?V} \Omega' \rhd M' \quad \mathsf{dom}(\Omega_e) \cap (\{a\} \cup \mathsf{fn}(M)) = \emptyset}{\Omega \rhd M \xrightarrow{(\Omega_e;\Phi)a?V} \Omega' \rhd M' \quad \Omega_e \text{ is a loyal extension of } \Omega}$$

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 it is not interacting with the system. We present in Figure 10 the rules of the LTS, omitting (LTS-NEWLOC), (LTS-NEWPASS), (LTS-IF-V), (LTS-IF-F), (LTS-NEWCHAN), (LTS-SPLIT) and (LTS-REP) which are τ transitions and are directly modelled after the corresponding reduction rule as (LTS-GOTO) is modelled after (R-GOTO).

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 ($\Omega \vdash_l a : \mathrm{R}\langle \mathsf{T} \rangle$). 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. Formally, this type is defined by:

$$\Omega^r(a) = \begin{cases} \mathsf{T}_r \text{ if } \bigsqcap\{\mathsf{E} \mid (a : \mathsf{E}) \in \Omega\} = \mathrm{RW}\langle \mathsf{T}_r, \mathsf{T}_w \rangle @ l \\ \mathsf{T} \quad \text{if } \bigsqcap\{\mathsf{E} \mid (a : \mathsf{E}) \in \Omega\} = \mathrm{R}\langle \mathsf{T} \rangle @ l \end{cases}$$

We know that we are in one of those two cases because $\Omega \vdash_l a : \mathrm{R}\langle \mathsf{T} \rangle$.

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 $\xrightarrow{:!:}$. 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 criterion used in defining barbs, that rule is constrained in the same way as the loyal contextuality (Definition 9).

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

Theorem 11 (Coincidence of semantics) *The reduction semantics and the semantics extracted from the LTS coincide in the sense of the following two properties:*

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

Proof: That fairly routine proof is decomposed in the following way.

(1) For the first implication, it is enough to notice that a system on which (LTS-COMM) is applicable is structurally congruent to a system on which (R-COMM) is applicable.
(2) For the second implication the structural congruence and (R-COMM) require a bit of work.
   - For the structural congruence, we simply prove: $\Omega_1 \rhd M_1 \xrightarrow{\mu} \Omega_2 \rhd M_2$ and $M_1 \equiv M^1$ implies that $\Omega_1 \rhd M^1 \xrightarrow{\bar{\mu}} \Omega_2 \rhd M^2$ and $M_1 \equiv M^2$ where $\bar{\mu}$ is μ up to some permutation in Φ when $\mu = (\Phi)c!V$. This is done by looking at the justification of $M_1 \equiv M^1$.
   - For (R-COMM), the fact that $\Omega \rhd M$ is well-formed implies that $M = l[\![c\,!\,\langle V\rangle\, P]\!] \,|\, l[\![c\,?\,(X : \mathsf{T})\, Q]\!]$ is well-typed in some environment Γ. It is then easy to see that the configurations $\Gamma \rhd l[\![c\,!\,\langle V\rangle\, P]\!]$ and $\Gamma \rhd l[\![c\,?\,(X : \mathsf{T})\, Q]\!]$ can perform respectively an output and an input action.

□

4.2 Bisimilarity equivalence

With the LTS defined above, we would like to define an equivalence $\mathcal{R}$ as a standard bisimulation: when $\Omega \vDash M \,\mathcal{R}\, N$ and $\Omega \rhd M \xrightarrow{\mu} \Omega' \rhd M'$ then there must exist some N' such that $\Omega \rhd N \xRightarrow{\hat{\mu}} \Omega' \rhd N'$ and $\Omega' \vDash M' \,\mathcal{R}\, N'$ where $\xRightarrow{\hat{\tau}} = \xrightarrow{\tau}{}^*$ and $\xRightarrow{\hat{\mu}} = \xrightarrow{\tau}{}^* \xrightarrow{\mu} \xrightarrow{\tau}{}^*$ 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 $\xRightarrow{\hat{\mu}}$, whatever μ may be.

Unfortunately 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 $\mathrm{RW}\langle \sum x, y : \mathrm{L}\vec{\mathrm{O}}\mathrm{C}.\ x \mapsto y \rangle_{@} l)$ and consider the following two systems:

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

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

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):

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

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 12 (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 $\tilde{a}$.

The two previous systems, with empty annotations, can then perform the following actions:

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

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

Definition 13 (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 [12]): the control of migrations hinders tracking the knowledge of the observer (apart from the passports, note that we must keep track of annotations because they hide some names from the observer).

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 9 is the fact that contexts of the form $[\cdot] \,|\, l[\![P]\!]$ 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 $a \in \tilde{a}$.

Definition 14 (Annotated loyal barbed congruence) *We call* annotated loyal barbed congruence*, written* $\cong^{al}$*, the biggest symmetric loyally contextual annotated relation that preserves barbs and is closed over reductions.*

Note that the conditions which define $\cong^{al}$ 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:

Lemma 15 (Bisimilarity is closed on loyal extensions) *If* $\Omega \vDash M \ {}_{\tilde{a}_M}\!\approx^{al}_{\tilde{a}_N} N$ *and* Ω' *is a loyal extension of* Ω *such that for every hypothesis* $a : \mathsf{E}$ *in* Ω', a *is fresh, then* $\Omega; \Omega' \vDash M \ {}_{\tilde{a}_M}\!\approx^{al}_{\tilde{a}_N} N$.

Proof: Ω' is a loyal extension introducing only fresh names for Ω. So an observer knowing $\Omega; \Omega'$ cannot observe any action Ω could not. This means that every action of the configuration $\Omega; \Omega' \triangleright_{\tilde{a}_M} M$ is also an action of $\Omega \triangleright_{\tilde{a}_M} M$. The result follows easily. □

Theorem 16 (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* $\Omega \vDash M \mid l[\![O]\!] \ {}_{\tilde{a}_M}\!\approx^{al}_{\tilde{a}_N} N \mid l[\![O]\!]$.

Proof: To get this result we 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 structured in this way: 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). Let us consider an action performed by the system containing M. and let us look first at the case where M is alone to perform the action.

M alone This must then be of the form:

$$\Omega \rhd (\mathsf{new}\, \Phi_M)(M \mid \prod_i l_i[\![O_i]\!]) \xrightarrow{\mu} \Omega' \rhd (\mathsf{new}\, \Phi_{M'})(M' \mid \prod_i l_i[\![O_i]\!])$$

If this is τ transition, it can be performed regardless of the knowledge of the observer and of the annotations. So

$$\Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd_{\tilde{a}^i_M} M \longrightarrow \Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd_{\tilde{a}^i_M} M'$$

and N can perform an equivalent reduction because it is bisimilar.

If μ is the output $\xrightarrow{(\tilde{b})c!V}$, the action must be proved by a transition of the LTS of the form:

$$\Omega \rhd (\mathsf{new}\, \Phi_M)(M \mid \textstyle\prod_i l_i[\![O_i]\!])$$
$$\xrightarrow{(\Phi'_M)c!V} \Omega, \langle V : \Omega^r(c) \rangle \ {}_{@}l \rhd (\mathsf{new}\, \Phi_{M'})(M' \mid \textstyle\prod_i l_i[\![O_i]\!])$$

the proof of which contains the proof of the transition of M alone, namely:

$$\Omega \rhd M \xrightarrow{(\Phi''_M)c!V} \Omega, \langle V : \Omega^r(c) \rangle \ {}_{@}l \rhd M'$$

We can prove that any transition of a configuration $\Omega \rhd M$ can also be performed by $\Omega' \rhd M$ where $\Omega' <: \Omega$ by an induction on the proof of the transition.

This induces that:

$$\Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd M \xrightarrow{(\Phi''_M)c!V} \Omega_O; p_1 : \star \mapsto l_1, \ldots, \langle V : \Omega^r(c)\rangle \circledcirc l \rhd M'$$

which corresponds to some action:

$$\Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd_{\tilde{a}^i_M} M \xrightarrow{(\tilde{b}')c!V} \Omega_O; p_1 : \star \mapsto l_1, \ldots, \left\langle V : \Omega'^r(c)\right\rangle \circledcirc l \rhd_{\tilde{a}^i_{M'}} M'$$

that must be matched by some action:

$$\Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd_{\tilde{a}^i_N} N \xRightarrow{(\tilde{b}')c!V} \Omega_O; p_1 : \star \mapsto l_1, \ldots, \left\langle V : \Omega'^r(c)\right\rangle \circledcirc l \rhd_{\tilde{a}^i_{N'}} N'$$

with

$$\Omega_O; p_1 : \star \mapsto l_1, \ldots, \left\langle V : \Omega'^r(c)\right\rangle \circledcirc l \vDash M' \; {}_{\tilde{a}^i_{M'}}\!\approx^{al}_{\tilde{a}^i_{N'}} N'$$

The action performed by N must be proved by a transition

$$\Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd N \xRightarrow{(\Phi''_N)c!V} \Omega_O; p_1 : \star \mapsto l_1, \ldots, \left\langle V : \Omega'^r(c)\right\rangle \circledcirc l \rhd N'$$

Naturally, we would like to deduce that $\Omega \rhd N$ can also perform this transition. Since $\Omega \rhd M$ can perform a similar transition, we know that all the required rights for the output on the channel c to be visible are available in Ω: namely we can prove $\Omega \vdash_l c : \mathrm{R}\langle \mathsf{T}\rangle$ and $l \in \mathcal{R}_\Omega$. We say that Ω *enables* the output on the channel c. Because that output is enabled, we can prove, by an induction on the proof of the transition of $\Omega_O; p_1 : \star \mapsto l_1, \ldots \rhd N$, that $\Omega \rhd N$ can perform the same transition. We conclude consecutively that:

$$\begin{aligned}
& \Omega \rhd N \xRightarrow{(\Phi''_N)c!V} \Omega, \langle V : \Omega^r(c)\rangle \circledcirc l \rhd N' \\
\text{so}\quad & \Omega \rhd (\mathsf{new}\,\Phi_N)(N \mid \textstyle\prod_i l_i[\![O_i]\!]) \\
& \quad \xRightarrow{(\Phi'_N)c!V} \Omega, \langle V : \Omega^r(c)\rangle \circledcirc l \rhd (\mathsf{new}\,\Phi_{N'})(N' \mid \textstyle\prod_i l_i[\![O_i]\!]) \\
\text{and}\quad & \Omega \rhd_{\tilde{a}_N} (\mathsf{new}\,\Phi_N)(N \mid \textstyle\prod_i l_i[\![O_i]\!]) \\
& \quad \xRightarrow{(\tilde{b}'')c!V} \Omega, \langle V : \Omega^r(c)\rangle \circledcirc l \rhd_{\tilde{a}_{N'}} (\mathsf{new}\,\Phi_{N'})(N' \mid \textstyle\prod_i l_i[\![O_i]\!])
\end{aligned}$$

For this action to be the expected one, namely $(\tilde{b})c!V$, we must check that $\tilde{b} = \tilde{b}''$.

$$\begin{aligned}
\tilde{b}'' &= (\mathsf{dom}(\Phi'_N) \cup \tilde{a}_N) \cap \mathsf{fn}(V) \\
&= \left((\mathsf{dom}(\Phi'_N) \cup \tilde{a}_N) \cap \mathsf{fn}(V) \cap \mathcal{N}_O\right) \cup \left(((\mathsf{dom}(\Phi'_N) \cup \tilde{a}_N) \cap \mathsf{fn}(V)) \setminus \mathcal{N}_O\right)
\end{aligned}$$

Since

$$(\mathsf{dom}(\Phi'_N) \cup \tilde{a}_N) \cap \mathcal{N}_O = (\mathsf{dom}(\Phi'_M) \cup \tilde{a}_M) \cap \mathcal{N}_O$$

only the second part of the $\tilde{b}''$ is still to be checked. Note that

$$\mathsf{dom}(\Phi'_N) \cap \mathsf{fn}(V) = (\mathsf{dom}(\Phi_N) \cup \mathsf{dom}(\Phi''_N)) \cap \mathsf{fn}(V)$$

since all the names that are mentioned in V must be opened.

$$\begin{aligned}&((\mathsf{dom}(\Phi'_N) \cup \tilde{a}_N) \cap \mathsf{fn}(V)) \setminus \mathcal{N}_O \\ &\quad = \Big((\mathsf{dom}(\Phi_N) \cup \mathsf{dom}(\Phi''_N) \cup \tilde{a}_N) \setminus \mathcal{N}_O\Big) \cap \mathsf{fn}(V) \\ &\quad = \Big(\tilde{a}^i_N \cup (\mathsf{dom}(\Phi''_N) \setminus \mathcal{N}_O)\Big) \cap \mathsf{fn}(V)\end{aligned}$$

Since, by definition, $\mathcal{N}_O$ is the set of names known to the observer, it cannot contain a name of $\mathsf{dom}(\Phi''_N)$. So this second part is exactly

$$(\tilde{a}^i_N \cup \mathsf{dom}(\Phi''_N)) \cap \mathsf{fn}(V) = \tilde{b}' = (\tilde{a}^i_M \cup \mathsf{dom}(\Phi''_M)) \cap \mathsf{fn}(V)$$

which finishes to prove that $\tilde{b}'' = \tilde{b}$, *i.e.* the system containing M and the one containing N can perform the same action.

To completely prove that we stay in the relation $\mathcal{S}$, we still need to check that all the induction hypotheses are preserved after this action. For this we define quite naturally the resulting environment Ω'_O as $\Omega_O, \langle V : \Omega^r(c)\rangle \circledcirc l$ since that's all the knowledge the observer got in the action. This new Ω'_O can still be split into the hypotheses it contains over names appearing in $\mathsf{dom}(\Phi_{M'})$ and a subtype environment of $\Omega, \langle V : \Omega^r(c)\rangle \circledcirc l$. We also define, similarly, $\mathcal{N}'_O$ as $\mathcal{N}_O \cup \mathsf{fn}(V)$. Finally the equalities concerning the annotations ($\tilde{a}_{M'}$, $\tilde{a}_{N'}$ but also $\tilde{a}^i_{M'}$ and $\tilde{a}^i_{N'}$) and the other sets of names can be checked following reasoning similarly to the one we gave for $\tilde{b}$ and $\tilde{b}''$.

If the system containing M performs an input $(\Phi)a?V$ instead of an output, the reasoning is far easier since M alone can perform exactly the same action (the Φ does not have to be modified) and all annotations are left unchanged.

$\prod_i l_i[\![O_i]\!]$ alone In that case, it is easy to see that both systems will be able to perform the same action. The only difficulty to take into account is the fact that some O_{i_0} might generate a new location name and change the general structure of the system into some

$$\prod_i l_i[\![O_i]\!] \,|((\mathsf{new}\,\Phi)\, k[\![O_k]\!] \,|\, l_{i_0}[\![O'_{i_0}]\!]) \,|\, \prod_i l_i[\![O_i]\!]$$

To regain the induction structure, we use the fact that $\mathcal{S}$ is to be a bisimulation up to structural congruence, using the structural congruence to join Φ and Φ_M, respectively Φ_N. Besides, Φ is fully added to the knowledge of the observer: Ω'_O is $\Omega_O; \Phi$, to be able to type $\prod_i l_i[\![O_i]\!] \,|\, k[\![O_k]\!] \,|\, l_{i_0}[\![O'_{i_0}]\!] \,|\, \prod_i l_i[\![O_i]\!]$. Finally, a new passport p_k must be added to the obtain the hypothesis

$$\Omega'_O; p_1 : \star \mapsto l_1, \ldots, p_k : \star \mapsto k \vDash M \;{}_{\tilde{a}^i_M}\!\approx^{al}_{\tilde{a}^i_N}\; N$$

Since p_k, k and all of Φ is fresh, it is easy to see that this follows from the hypothesis $\Omega_O; p_1 : \star \mapsto l_1, \ldots \vDash M\ {}_{\tilde{a}^i_M}{\approx^{al}}_{\tilde{a}^i_N}\ N$. The other hypotheses are automatically preserved.

The other technical case to check is when the action performed is an output or an input. The reasoning for these cases is similar and simpler than the one when M performs the action.

Communication between M and $\prod_i l_i[\![O_i]\!]$ This supposes that M is performing an output and $\prod_i l_i[\![O_i]\!]$ an input or vice versa. Note that M and N are known to be bisimilar for an observer having explicit access to all the locations l_i, in particular the one where the communication takes place. Thanks to this, the reasoning is similar to the previous cases. □

Lemma 17 (Bisimilarity is closed on restrictions) *If $\Omega; a : \mathsf{E} \vDash M\ {}_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N}\ N$ and both configurations $\Omega \rhd (\mathit{new}\, a : \mathsf{E})\, M$ and $\Omega \rhd (\mathit{new}\, a : \mathsf{E})\, N$ are well-formed, then $\Omega \vDash (\mathit{new}\, a : \mathsf{E})\, M\ {}_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N}\ (\mathit{new}\, a : \mathsf{E})\, N$.*

Proof: To prove this lemma, let us consider the relation $\mathcal{S}$ defined by:

$$\begin{cases} \Omega \vDash (\mathsf{new}\, a : \mathsf{E})\, M\ {}_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N}\ (\mathsf{new}\, a : \mathsf{E})\, N \\ \Omega \vDash (\mathsf{new}\, a : \mathsf{E})\, M\ {}_{\tilde{a}_M}\mathcal{S}_{\tilde{a}_N, a}\ N \\ \Omega \vDash M\ {}_{\tilde{a}_M, a}\mathcal{S}_{\tilde{a}_N}\ (\mathsf{new}\, a : \mathsf{E})\, N \\ \Omega \vDash M\ {}_{\tilde{a}_M, a}\mathcal{S}_{\tilde{a}_N, a}\ N \end{cases}$$

$$\text{if} \quad \begin{cases} a \notin \tilde{a}_M \quad \text{et} \quad a \notin \tilde{a}_N \\ \Omega; a : \mathsf{E} \vDash M\ {}_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_N}\ N \\ \Omega \rhd_{\tilde{a}_M} (\mathsf{new}\, a : \mathsf{E})\, M \quad \text{and} \quad \Omega \rhd_{\tilde{a}_N} (\mathsf{new}\, a : \mathsf{E})\, N \quad \text{are well-formed} \end{cases}$$

and the relation $\mathcal{R} = \approx^{al} \cup \mathcal{S}$. The proof that this relation is a bisimulation is fairly easy by an analysis of the different possible cases for the scope of the name a: open (in which case we must be in $\approx^{al}$), or one of the two ways to be closed (restriction or annotation) in both M and N. This analysis must be done only for outputs: on inputs no modification of the scope of a can happen. □

We have now the tools to obtain:

Theorem 18 $\approx^{al} \subseteq \cong^{al}$

Proof: A barb on a channel c corresponds exactly to an action $(\cdot)c!\cdot$ so $\approx^{al}$ must preserve barbs. The fact that it is closed on reductions follows from Theorem 11. And its loyal annotated contextuality is given by Theorem 16 and Lemmas 15 and 17. □

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* written $\cong^{p}$, 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:

$$\begin{aligned} w_1 : \text{LOC}, \ldots, w_m : \text{LOC}, \\ u_1 : \tilde{w}_{i_1} \mapsto w_{i_1}, \ldots, u_n : \tilde{w}_{i_n} \mapsto w_{i_n}, \\ v_1 : \mathsf{C}_{1@}w_{j_1}, \ldots, v_o : \mathsf{C}_{o@}w_{j_o} \end{aligned}$$

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 $w_{j_k} \neq w_{j_{k'}}$.

So this normal form has the following structure: all the locations are defined first because types can depend only on location identifiers so that all the locations can be listed first; and every identifier is attributed exactly one type per location identifier to which it is attached since the well-formedness of environments ensures that any two types associated with a given identifier must be weakly $\downarrow$-compatible (remember that when typechecking processes, a given channel or passport can be attached to more than one location variable). The existence of such a normal form follows from the property of partial meets (Theorem 2) which ensures that all the types associated with a given identifier sum up to their meet.

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 19 (Reification of environments) *To an environment* Γ *of the following (normal) form*

$$\begin{aligned} w_1 : \text{LOC}, \ldots, w_m : \text{LOC}, \\ u_1 : \tilde{w}_{i_1} \mapsto w_{i_1}, \ldots, u_n : \tilde{w}_{i_n} \mapsto w_{i_n}, \\ v_1 : \mathsf{C}_{1@}w_{j_1}, \ldots, v_o : \mathsf{C}_{o@}w_{j_o} \end{aligned}$$

we associate the value

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

of type

$$\mathsf{T}_\Gamma = \sum x_1, \ldots, x_m : \tilde{\text{LOC}}.\ \tilde{x}^\star_{i_1} \mapsto x_{i_1}, \ldots, \tilde{x}^\star_{i_n} \mapsto x_{i_n}, \mathsf{C}_{1@}x_{j_1}, \ldots, \mathsf{C}_{o@}x_{j_o}$$

Proposition 20 (Soundness of the reification) *For any well-formed environment* Γ *and any location* w *defined in* Γ, $\Gamma \vdash_w V_\Gamma : \mathsf{T}_\Gamma$.

To prove that the parallel congruence is included in the bisimilarity, we want to prove that it is a bisimulation. For this, let us consider an element in the parallel congruence, say $\Omega \vDash M \ {}_{\tilde{a}_M}\!\cong^p_{\tilde{a}_N} N$, and let us suppose that $\Omega \rhd_{\tilde{a}_M} M$ can perform an output action $(\tilde{b})c!U$. Let us define some context that can detect that very output action and that, when that action has been performed, outputs on some fresh channel ω the value $V_{\Omega,\langle U:\Omega^r(c)\rangle}$, namely the reification of the knowledge of the observer after the action. Such a context might 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 could be distinguished. 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.

Let us detail a bit more the step 2: the context has received some value U' that got substituted for some pattern X and it must check that U' is U.

- The context must test that all the names that appear at some positions in U and that are not in $\tilde{b}$ also appear in the same positions in U'.
- It must then check that the names appearing in U' at the positions of some names among $\tilde{b}$ in U are yet unknown: for this, it checks that those names are different from every name in some set $\mathcal{N}$ containing at least all the names appearing in Ω and in M apart from the names $\tilde{a}_M$.
- Finally, it must also check that all the names appearing in U' at the positions of the $\tilde{b}$s in U are distinct from each other, except when they are the same $b \in \tilde{b}$ in U in which case it must check that they are indeed equal. This last check is mandatory to distinguish between the actions $(b_1, b_2)c!(b_1, b_1, b_2)$ and $(b_1, b_2)c!(b_1, b_2, b_2)$ for instance.

We will write this sequence of tests $X =_{\mathcal{N}} (\tilde{b})U$.

So we can define the context for that action:

Definition 21 (Output-identifying context) *To the output action $(\tilde{b})c!V$, the knowledge of the observer Ω and the set of identifiable names $\mathcal{N}$, we associate the context*

$$\mathfrak{C}^{\Omega}_{\mathcal{N}}((\tilde{b})c!U) = [\cdot] \mid \lambda[\![\delta\,!\,\langle\rangle]\!] \mid$$
$$l[\![c\,?\,(X : \Omega^r(c))$$
$$\mathsf{if}\ X =_{\mathcal{N}} (\tilde{b})U\ \mathsf{then}\ \mathsf{goto}_{\pi}\ \lambda.\ \delta\,?\,()\,\omega\,!\left\langle V_{\Omega,\langle U\{^{x_{p_1}}/_{b_1}\}\cdots\{^{x_{p_n}}/_{b_n}\}:\Omega^r(c)\rangle@l}\right\rangle\ \mathsf{else}\ \mathsf{stop}[\![$$

where:

- *the names λ, π, δ and ω are fresh;*
- *x_{p_i} is a variable appearing in the pattern X at a position of b_i in U.*

Note that $\lambda : \textsc{loc}, \pi : \star \mapsto \lambda, \delta : \textsc{rw}\langle\rangle_@\lambda, \omega : \textsc{rw}\langle \mathsf{T}_{\Omega,\langle U:\Omega^r(c)\rangle}\rangle_@\lambda$ is a loyal extension for Ω and that the context $\mathfrak{C}^{\Omega}_{\mathcal{N}}((\tilde{b})c!U$ is well-typed in Ω thus extended.

We now state the property the context was defined for:

Lemma 22 (Output actions are definable) *Let Ω be an environment, α an output action $(\tilde{b})c!U$ which is visible to Ω, M a system and $\mathcal{N}$ a set of names such that $\tilde{a}_M \cap \mathcal{N} = \emptyset$ and $(\mathsf{fn}(M) \setminus \tilde{a}_M) \subseteq \mathcal{N}$.*

$$\Omega \rhd_{\tilde{a}_M} M \xrightarrow{\alpha} \Omega \text{ after } \alpha \rhd_{\tilde{a}'_M} M'$$

implies

$$\mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)[M] \Longrightarrow (\mathsf{new}\,\Phi)(M' \mid \lambda[\![\mathsf{stop}]\!] \mid \lambda[\![\omega\,!\left\langle V_{\Omega,\langle U:\Omega^r(c)\rangle}\right\rangle]\!])$$

where $\mathsf{dom}(\Phi) = (\tilde{b} \cup \tilde{a}'_M) \setminus \tilde{a}_M$.

Conversely

$$\mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)[M] \Longrightarrow M''$$

when $\Omega\lambda\pi\omega\delta \rhd_{\tilde{a}_M} M'' \Downarrow \omega$ *and* $\Omega\lambda\pi\omega\delta \rhd_{\tilde{a}_M} M'' \not\Downarrow \delta$ *where* $\Omega\lambda\pi\omega\delta = \Omega; \lambda : \textsc{loc}; \pi : \star \mapsto \lambda; \omega : \textsc{rw}\langle \mathsf{T}_{\Omega,\langle U:\Omega^r(c)\rangle}\rangle_{@}\lambda; \delta : \textsc{rw}\langle\rangle_{@}\lambda$ *implies that M'' must be structurally congruent to* $(\mathsf{new}\,\Phi)(M' \mid \lambda[\![\mathsf{stop}]\!] \mid \lambda[\![\omega\,!\left\langle V_{\Omega,\langle U:\Omega^r(c)\rangle}\right\rangle]\!])$ *with*

$$\Omega \rhd_{\tilde{a}_M} M \stackrel{\alpha}{\Longrightarrow} \Omega,\langle U : \Omega^r(c)\rangle \rhd_{\tilde{a}'_M} M'$$

$\tilde{b} = (\mathsf{dom}(\Phi) \cup \tilde{a}_M) \cap \mathsf{fn}(V)$ *and* $\tilde{a}'_M = (\mathsf{dom}(\Phi) \cup \tilde{a}_M) \setminus \mathsf{fn}(V)$.

Proof: The first implication mostly amounts to checking that the sequence of tests $X =_{\mathcal{N}} (\tilde{b})U$ are a success when U is substituted for X.

To prove the second implication, we rely on the Theorem 11 to obtain

$$\Omega\lambda\pi\omega\delta \rhd \mathfrak{C}^{\Omega}_{\mathcal{N}}(\alpha)[M] \xrightarrow{\tau}{}^{*} \Omega\lambda\pi\omega\delta \rhd M'''$$

with $M''' \equiv M''$. Since $\Omega\lambda\pi\omega\delta \rhd_{\tilde{a}_M} M'''$ does not exhibit any barb on δ, we know that the whole context has been reduced. In particular we can extract the proof of M performing the output and we know that the sequence of tests $X =_{\mathcal{N}} (\tilde{b})U$ was successful, which concludes the proof. $\square$

The contexts characterising inputs are even simpler and look like:

$$\mathfrak{C}^{\Omega}_{\mathcal{N}}(c?V) = [\cdot] \mid \lambda[\![\delta\,!\,\langle\rangle]\!] \mid l[\![c\,!\,\langle V\rangle\,\mathsf{goto}_{\pi}\,\lambda.\,\delta\,?\,()\,\omega\,!\,\langle V_{\Omega}\rangle]\!]$$

Note that, for the input $(\Phi)c?V$, Φ is a loyal extension of Ω and since $\cong^p$ is closed on loyal extensions, there is no need to generate the names in Φ in the context.

The last property of interest to finish our proof deals with the fact that the knowledge of the environment encoded in a value can be recovered.

Lemma 23 (Scope extrusion) *If*

$$\lambda : \text{LOC}, \pi : \star \mapsto \lambda, \omega : \text{RW}\langle \mathsf{T}_\Omega \rangle_@ \lambda \vDash$$
$$(\mathit{new}\,\Phi_M)(M \mid \lambda[\![\omega\,!\,\langle V_\Omega \rangle]\!]) \;{}_{\tilde{a}_M}{\cong^p}_{\tilde{a}_N}\; (\mathit{new}\,\Phi_N)(N \mid \lambda[\![\omega\,!\,\langle V_\Omega \rangle]\!])$$

and the names λ, π and ω are fresh for M and N, then

$$\Omega \vDash M \;{}_{\tilde{a}'_M}{\cong^p}_{\tilde{a}'_N}\; N$$

where $\tilde{a}'_M = (\tilde{a}_M \cup \mathsf{dom}(\Phi_M)) \setminus \mathsf{dom}(\Omega)$ *and* $\tilde{a}'_N = (\tilde{a}_N \cup \mathsf{dom}(\Phi_N)) \setminus \mathsf{dom}(\Omega)$.

Proof: Of course, this proof is done by considering the relation $\mathcal{S}$:

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

and proving that it is included in $\cong^p$ by checking that it verifies all the defining conditions of $\cong^p$.

The intuition of that proof is basically that a context can be added to the system $(\mathsf{new}\,\Phi_M)(M \mid \lambda[\![\omega\,!\,\langle V_\Omega \rangle]\!])$, recover the value V_Ω and perform the expected test. For instance, to check that $\mathcal{S}$ is barb-preserving, we use systems of the form

$$\lambda[\![\omega\,?\,(X : \mathsf{T}_\Omega)\,\mathsf{goto}_{x_p}\,x_l.\; x_c\,?\,(Y : \Omega^r(c))\,\mathsf{goto}_\pi\,\lambda.\; \delta\,!\,\langle\rangle]\!]$$

where x_c, x_l and x_p are the variables of the pattern X corresponding to a channel on which a barb must be detected, the location of that channel and a passport leading to that location. (We use x_a to denote the variable of X corresponding to a in V_Ω). Putting this in parallel to a system will exhibit a barb on δ if and only if there was a barb on c.

Let us prove that $\mathcal{S}$ is closed on parallel contexts, this being the most difficult case. Consider for this the context $[\cdot] \mid k[\![P]\!]$ where $\Omega \vdash k[\![P]\!]$ and $k \in \mathcal{R}_\Omega$. We transform this context into

$$\mathfrak{C}_p = [\cdot] \mid \lambda[\![\omega\,?\,(X : \mathsf{T}_\Omega)\,\omega_p\,!\,\langle X \rangle \mid \mathsf{goto}_{x_{q_1}}\,x_{l_1}.\;\ldots\,\mathsf{goto}_{x_{q_m}}\,x_k.\; P\{^X\!/_{V_\Omega}\}]\!]$$

where $q_1, \ldots, q_m$ et $l_1, \ldots, l_m = k$ are the sequences of passports and locations proving $k \in \mathcal{R}_\Omega$.

We know then that:

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

It is easy to see that this entails:

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

To finish the proof we need to prove that it induces

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

i.e. the reductions introduced by $\mathfrak{C}_p$ can be performed without getting out of the equivalence. In fact, it is fairly easy to prove that the reduced system is bisimilar to the original one:

$$\lambda\pi\omega_p \vDash$$
$$(\mathsf{new}\,\Phi_M)(M \mid \lambda[\![\omega\,!\,\langle V_\Omega\rangle]\!] \mid \mathfrak{C}_p)\ {}_{\tilde{a}_M}{\approx^{al}}_{\tilde{a}_M}\ (\mathsf{new}\,\Phi_M)(M \mid k[\![P]\!] \mid \lambda[\![\omega\,!\,\langle V_\Omega\rangle]\!])$$

and Theorem 18 and transitivity of $\cong^p$ finish that proof. □

So we can finally reach our aim:

Theorem 24 $\cong^p \subseteq \approx^{al}$

Proof: 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 11 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 output action $\Omega \rhd_{\tilde{a}_M} M \xrightarrow{\alpha} \Omega' \rhd_{\tilde{a}'_M} M'$, Lemma 22 proves that $\mathfrak{C}^\Omega_{\mathcal{N}}(\alpha)[M]$ can reduce into some system $(\mathsf{new}\,\Phi_M)\,M' \mid \lambda[\![\mathsf{stop}]\!] \mid \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$. By contextuality and closure on reductions, $\mathfrak{C}^\Omega_{\mathcal{N}}(\alpha)[N]$ should reach an equivalent state, with a barb on ω and no barb on δ. Using the other direction of Lemma 22, that equivalent state must congruent to $(\mathsf{new}\,\Phi_N)\,N' \mid \lambda[\![\mathsf{stop}]\!] \mid \lambda[\![\omega\,!\,\langle V_{\Omega'}\rangle]\!]$ with $\Omega \rhd_{\tilde{a}_N} N \stackrel{\alpha}{\Longrightarrow} \Omega' \rhd_{\tilde{a}'_N} N'$. Now Lemma 23 finishes the proof that $\Omega' \vDash M'\ {}_{\tilde{a}'_M}{\cong^p}_{\tilde{a}'_N}\ N'$.

The reasoning is similar for input actions. □

Lemmas 18 and 24 and the correspondence between $\cong^{al}$ and $\cong^l$ directly entail the expected result:

Theorem 25 (Full abstraction of $\approx^{al}$ for $\cong^l$) *$\Omega \vDash M \cong^l N$ if and only if $\Omega \vDash M\ {}_{\emptyset}{\approx^{al}}_{\emptyset}\ N$*

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 [13] and [22]. 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.

It would also be interesting to refine passports to stricter notions of trust, where other locations are prevented from relaying processes for instance.

Acknowledgement The author would like to thank Matthew Hennessy for numerous helpful discussions and comments and the anonymous referees for their careful proof-reading and remarks.

References

[1] M. Hennessy, A Distributed Pi-calculus, Cambridge University Press, 2007.

[2] G. Boudol, A generic membrane model (note), in: C. Priami, P. Quaglia (Eds.), Global Computing, Vol. 3267 of Lecture Notes in Computer Science, Springer, 2004, pp. 208–222.

[3] D. Gorla, M. Hennessy, V. Sassone, Security policies as membranes in systems for global computing, Logical Methods in Computer Science 1 (3).
URL `http://dx.doi.org/10.2168/LMCS-1(3:2)2005`

[4] F. Levi, D. Sangiorgi, Controlling interference in ambients, in: 27th Annual Symposium on Principles of Programming Languages (POPL) (Boston, MA), ACM, 2000, pp. 352–364.

[5] M. Merro, M. Hennessy, A bisimulation-based semantic theory of Safe Ambients, ACM Transactions on Programming Languages and Systems 28 (2) (2006) 290–330.

[6] M. Bugliesi, S. Crafa, M. Merro, V. Sassone, Communication interference in mobile boxed ambients, in: M. Agrawal, A. Seth (Eds.), FSTTCS, Vol. 2556 of Lecture Notes in Computer Science, Springer, 2002, pp. 71–84.
URL http://link.springer.de/link/service/series/0558/bibs/2556/25560071.htm

[7] G. Castagna, F. Z. Nardelli, The Seal calculus revisited: Contextual equivalence and bisimilarity, in: M. Agrawal, A. Seth (Eds.), FSTTCS, Vol. 2556 of Lecture Notes in Computer Science, Springer, 2002, pp. 85–96.
URL http://link.springer.de/link/service/series/0558/bibs/2556/25560085.htm

[8] A. Schmitt, J.-B. Stefani, The Kell calculus: A family of higher-order distributed process calculi, in: C. Priami, P. Quaglia (Eds.), Global Computing, Vol. 3267 of Lecture Notes in Computer Science, Springer, 2004, pp. 146–178.
URL http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3267&spage=146

[9] L. Cardelli, G. Ghelli, A. D. Gordon, Ambient groups and mobility types, in: J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, T. Ito (Eds.), IFIP TCS, Vol. 1872 of Lecture Notes in Computer Science, Springer, 2000, pp. 333–347.
URL http://link.springer.de/link/service/series/0558/bibs/1872/18720333.htm

[10] M. Coppo, M. Dezani-Ciancaglini, E. Giovannetti, Types for Ambient and Process Mobility, Mathematical Structures in Computer ScienceTo appear.

[11] R. D. Nicola, G. L. Ferrari, R. Pugliese, Types as specifications of access policies, in: J. Vitek, C. D. Jensen (Eds.), Secure Internet Programming, Vol. 1603 of Lecture Notes in Computer Science, Springer, 1999, pp. 118–146.

[12] M. Hennessy, M. Merro, J. Rathke, Towards a behavioural theory of access and mobility control in distributed systems, Theoretical Computer Science 322 (2003) 615–669.

[13] M. Hennessy, J. Rathke, N. Yoshida, SafeDpi: a language for controlling mobile code., Acta Informatica 42 (4-5) (2005) 227–290.

[14] N. Yoshida, Channel dependent types for higher-order mobile processes (Sep. 2004).
URL http://citeseer.ist.psu.edu/684491.html;http://www.doc.ic.ac.uk/~yoshida/paper/partI.ps

[15] F. Martins, V. T. Vasconcelos, History-based access control for distributed processes., in: TGC, 2005, pp. 98–115.

[16] M. Boreale, D. Sangiorgi, Bisimulation in name-passing calculi without matching, in: Thirteenth Annual Symposium on Logic in Computer Science (LICS) (Indiana), IEEE, Computer Society Press, 1998.

[17] S. Hym, Mobility control via passports (extended abstract), in: CONCUR, Vol. 4703 of Lecture Notes in Computer Science, Springer, 2007, pp. 349–363.

[18] S. Hym, M. Hennessy, Adding recursion to Dpi, Theoretical Computer Science 373 (3) (2007) 182–212.
URL http://dx.doi.org/10.1016/j.tcs.2006.12.017

[19] S. Hym, Typage et contrôle de la mobilité, Ph.D. thesis, Université Paris Diderot – Paris 7 (Dec. 2006).

[20] A. K. Wright, M. Felleisen, A syntactic approach to type soundness, Information and Computation 115 (1) (1994) 38–94.

[21] R. Milner, D. Sangiorgi, Barbed bisimulation, in: W. Kuich (Ed.), 19th ICALP, Vol. 623 of Lecture Notes in Computer Science, Springer-Verlag, 1992, pp. 685–695.

[22] K. Crary, R. Harper, F. Pfenning, B. C. Pierce, S. Weirich, S. Zdancewic, Manifest security for distributed information, white paper (Mar. 2006).