An Asynchronous, Distributed Implementation of Mobile Ambients

Cédric Fournet
fournet@microsoft.com
Microsoft Research

Jean-Jacques Lévy
Jean-Jacques.Levy@inria.fr
INRIA Rocquencourt

Alan Schmitt
Alan.Schmitt@inria.fr
INRIA Rocquencourt

Preliminary Draft, March 2000

Abstract

We present a first distributed implementation of the Cardelli-Gordon's ambient calculus. We use Jocaml as an implementation language and we present a formal translation of Ambients in the distributed Join Calculus, the process calculus associated with Jocaml. We prove the correctness of the translation.

The operational semantics of Ambients requires that complex migration steps be performed atomically. As a result, direct implementations rely on the serialization of migrations via a centralized control. In contrast, our implementation is fully asynchronous and allows concurrent reduction steps. It relies on a message-passing protocol between an ambient and its parent. Only the actions involving an ambient migrating or being opened are blocked during a reduction step.

The proof of correctness of this implementation is two staged. First, an extended ambient calculus with a refined semantics is introduced, and the two semantics are related using coupled-simulations. Then, a bisimulation is shown to exist between processes in the extended ambient calculus and their distributed join calculus translations. In combination, these results imply that our translation does not affect a variety of global observation predicates. In the final section, the actual implementation in Jocaml is discussed. As a whole, our paper gives an insight in the implementability of ambients.

In the appendices, the semantics of ambients and the join calculus is recalled, with an overview of both calculi.

Contents

1 Introduction

This paper presents a highly concurrent distributed implementation of the Cardelli-Gordons's calculus of Mobile Ambients [5] in Jocaml [13]. The ambient calculus is a simple and very esthetic model for distributed mobile computing. However, until now, it did not have a distributed implementation. Such an implementation may seem easy to build, especially with a language with distribution and strong migration (Jocaml), but we encountered several difficulties and design choices.

Ambients are nested. Their dynamics is defined by three atomic steps: an ambient may move into a sibling ambient (IN), it may move out of its parent ambient (OUT), or it may open one of its child ambients (OPEN). Each atomic migration step may involve several ambients, possibly on different sites. For instance, the source and destination ambients participate to an IN-step; similarly the source and parent ambients take part to an OUT-step; the target ambient participates to an OPEN-step. Each atomic step of the ambients calculus can be decomposed in two parts: checking whether the local structure of the ambient tree enables the step, and actually performing the migration.

The first part imposes some distributed synchronization. One may use a global synchronous primitive existing at the operating system or networking level, but such a solution is not much portable, nor does exist in large scale networks. A first range of solutions can be designed by considering locks and critical sections in order to serialize the implementation of atomic steps. For instance, the two ambients participating to a reduction step are temporary locked, and the locks are released at the end of the step. However this solution cannot be symmetric, in the same way as there is no symmetric distributed solution to the Dining Philosophers problem. Some ambients have to be distinguished, for instance, one ambient could be the synchronizer of all ambients. Naturally, the nested structure of ambients can be used, for instance each ambient controls the synchronization of its direct subambients. In both cases, one has to be careful to avoid deadlocks or too much serialization. This solution would be similar to Cardelli's centralized implementation of an earlier variant of the ambient calculus in Java™ [2, 3]. One advantage of a serialized solution is the ease of the correctness proof of the implementation. On the negative side, each attempt to perform a step takes several locks higher up in the ambient hierarchy; these locks may be located at remote sites, leading to long delays before these locks are released for other local steps. Moreover, due to the mobility discipline of the ambient calculus, an ambient that migrates from one point to another in the ambient hierarchy has to travel through an ambient enclosing both the origin and the destination, thus inducing global bottlenecks.

A different set of solutions is fully asynchronous. Atomic steps of ambients are decomposed into several elementary steps, each involving only local synchronization. In this approach, each ambient step is implemented as a run of a protocol involving several messages. Concurrency is higher, as only the moving ambient might not be available for other reduction steps. For instance, our solution never blocks steps involving parents of a moving ambient. Our solution even allows ambients to move out of a moving ambient; or to complete an IN-move towards a moving ambient; but it does not allow ambients to initiate such IN-moves. The implementation of migration towards a mobile target may be problematic, but can be handled independently of the implementation of ambient synchronization, e.g., using a forwarding mechanism. In our case, we simply rely on the strong migration primitive of Jocaml. On the negative side, the correctness proof is more involved.

In this paper, we present an asynchronous distributed algorithm for implementing ambients, we make it precise as a translation into the Join Calculus—the process calculus which is a model of Jocaml[10], and we refine this translation into a distributed implementation of ambients written in Jocaml. The algorithm provides an insight into the implementability of ambients. The Jocaml prototype is a first, lightweight, distributed implementation of ambients. The translation is proved correct in two stages: first we use barbed coupled-similarity for the correctness of the algorithm, then we use an hybrid barbed bisimulation for the actual translation to the join calculus. Technically, the first stage is a first application of coupled-simulations [18] in a reduction-based, asynchronous setting; it relies on the introduction of an auxiliary ambient calculus extended with transient states; it does not depend of the target language. The second stage is a challenging application of the decreasing diagram technique [17]. In combination, these results imply that the translation does not affect a variety of global observation predicates.

The paper is organized as follows. In section 2, we present the asynchronous algorithm and we show a formal translation from ambient processes to join calculus processes. In section 3, we discuss the correctness of the translation in terms of observations. In section 4, we focus on the operational correspondence between a process and its translation; to this end, we refine the ambient calculus to express additional transient states induced by the translation. In section 5, we state our main technical results and give an idea of their proofs. In section 6, we describe more practical aspects of the Jocaml implementation. We conclude in section 7.

In appendix A, we recall the operational semantics of the distributed join calculus and of the calculus of Mobile Ambients, and we give an overview of both calculi. (Additional motivations, details, and examples can be found in [10] and in [5], respectively.) In appendices B.1 and B.2, we provide the proofs for section 5.

2 From ambients to the join calculus

We describe the asynchronous algorithm, then we specify it as a translation from ambients to the join calculus. We begin with a fragment of the ambient calculus given by the grammar $P ::= \; a[\,P\,] \;\big|\; \mathtt{in}\; a.P \;\big|\; \mathtt{out}\; a.P \;\big|\; P \mid P' \;\big|\; \mathbf{0}$. In a second stage, we incorporate OPEN steps and other ambient constructs.

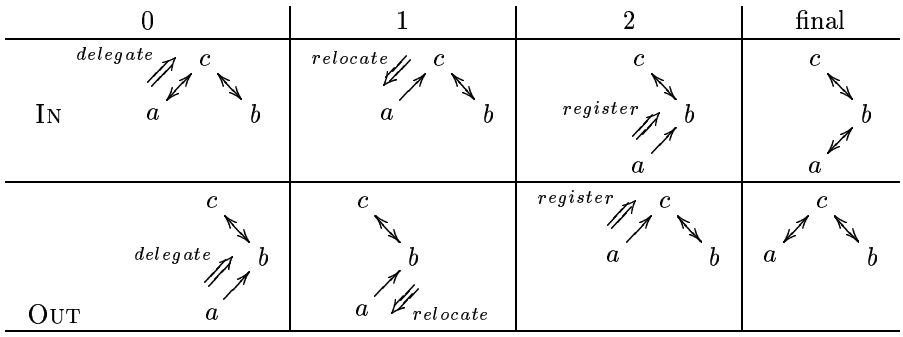
2.1 An asynchronous algorithm

The dynamic tree structure of ambients is represented by a doubly linked tree. Each node in the tree implements an ambient: each node contains non-ambient processes such as $\mathtt{in}\; b.P$ or $\mathtt{out}\; a.c[Q]$ running in parallel; each node also hosts an *ambient manager* that controls the steps performed in this ambient and in its direct subambients. Different nodes may be running at different physical sites, so their ambient managers should use only asynchronous messages to communicate with one another. Since several ambients may have the same name, each node is also associated with a unique identifier. (Informally, we still refer to ambients by name, rather than unique identifier.)

Each ambient points to its subambients and to its parent ambient. The down links are used for controlling subambients, the up link is used for proposing new actions. The parent of the moving ambient for an IN-step knows the destination ambient; the parent also knows the destination ambient—its own parent—for an

OUT-step; it controls the opened ambient for an OPEN-step. Hence, the decision to perform a step will always be taken by the parent of the affected ambient.

Ambient moves of a in and out of b correspond to three successive steps, depicted below. Single arrows represent current links; double arrows represent messages in transit.



We detail the dynamics of an IN-step, e.g., $c[\, a[\, \mathtt{in}\, b.Q\,]\ \mid\ b[\, 0\,]\,] \rightarrow c[\, b[\, a[\, Q\,]\,]\,]$.

0-step: initially, a delegates the migration request IN b to its current parent (here c); to this end, it uses its current up link to send a message to c saying that a is willing to move into an ambient named b.

1-step: the enclosing ambient c matches a's request with a's and b's down links. Atomically, a's request and the down link to a are erased, and a relocation message is sent to a; this message contains the address of b, so that a will be able to relocate to b, and also a descriptor of a's successful action, so that a can complete this step by triggering its guarded process.

2-step: the moving ambient a receives c's relocation message, relocates to b's site, and updates its up link to point to b. It also sends a message to b that eventually registers a as a subambient of b.

The 1-step may preempt other actions delegated by a to its former parent c. Such actions should now be delegated to its new parent b. For that purpose, a's ambient manager keeps a log of the pending actions delegated in 0-steps, and, as it completes one of these action in a 2-step, it re-delegates all other actions towards its new parent. The log cannot be maintained by the parent, because delegation messages may arrive long after a's departure. Moreover, in the case an ambient moves back into a former parent, former delegation messages may still arrive, and should not be confused with fresh ones. Such stale messages must be deleted. This is not directly possible in an asynchronous world, but equivalently each migration results in a modification of the unique identifier of the moving ambient, each delegation message is tagged with this identifier, and the parent discards every message with an old identifier.

An OUT-step of a out of b corresponds to the same series of three steps. The main different is in step 1, as the enclosing ambient b matches a's request with a's down link and its own name b, and passes its own up link to c in the relocation message sent back to a.

2.2 A simple translation

The compositional translation $[\![\ \cdot\ ]\!]_e$ appears in Figure 1. It uses a (meta) function $AM_{a,e}(\,\cdot\,)$ implementing the ambient manager. Overall, the tree of nested ambients

$$
\begin{aligned}
[\![a[\,P\,]]\!]_e &= \texttt{def}\ AM_{a,e}(P)\ \texttt{in}\ \mathbf{0} \\
[\![\texttt{in}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.in(a,\kappa) \\
[\![\texttt{out}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.out(a,\kappa) \\
[\![P \mid Q]\!]_e &= [\![P]\!]_e \mid [\![Q]\!]_e \\
[\![\mathbf{0}]\!]_e &= \mathbf{0}
\end{aligned}
$$

where the ambient manager $AM_{a,e}(P)$ is defined as:

$$
\begin{aligned}
D_0 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid in(b,\kappa) \triangleright s(a,i,e,l \cup \{\text{In } b\ \kappa\}) \mid e.sub_{in}(i,b,\kappa) \\
&\wedge\ s(a,i,e,l) \mid out(b,\kappa) \triangleright s(a,i,e,l \cup \{\text{Out } b\ \kappa\}) \mid e.sub_{out}(i,b,\kappa) \\
D_1 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid amb(k,c,e_c) \mid sub_{in}(k,b,\kappa) \triangleright \\
&\quad\ s(a,i,e,l) \mid amb(j,b,e_b) \mid e_c.reloc(e_b,\kappa) \\
&\wedge\ s(a,i,e,l) \mid amb(j,b,e_b) \mid sub_{out}(j,a,\kappa) \triangleright s(a,i,e,l) \mid e_b.reloc(e,\kappa) \\
D_2 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid reloc(e',\kappa) \triangleright \texttt{go}(e'.here); (I_{a,e_h,e'} \mid \kappa() \mid Flush(l,in,out,\kappa)) \\
D &\stackrel{\text{def}}{=} D_0 \wedge D_1 \wedge D_2 \\
\\
I_{a,e_h,e} &\stackrel{\text{def}}{=} \texttt{def uid}\ i\ \texttt{in}\ s(a,i,e,\emptyset) \mid e.amb(i,a,e_h) \\
\\
AM_{a,e}(P) &\stackrel{\text{def}}{=} here\left[D\ :\ I_{a,e_h,e} \mid [\![P]\!]_{e_h}\right]
\end{aligned}
$$

with the record notation $e_h \stackrel{\text{def}}{=} \left\{ \begin{array}{l} here = here, amb = amb, sub_{in} = sub_{in}, \\ sub_{out} = sub_{out}, reloc = reloc, in = in, out = out \end{array} \right\}$.

Figure 1: Translation from In/Out ambient processes to the join calculus

is mapped to an isomorphic tree of nested locations. Each ambient is mapped to a join calculus location containing the definition of the channel names that form the ambient interface, and containing processes that represent the ambient state. The definition D is composed of three groups of rules D_0, D_1, and D_2 that respectively implement 0, 1, and 2-steps of the algorithm.

To represent the data structure used in the algorithm of section 2.1, an ambient is represented by an interface e, which is a record that contains fields *here*, *amb*, sub_{in}, sub_{out}, *reloc*, *in*, and *out*. The *here*-field is the name of the location hosting the ambient, whereas the other fields are channel names used to interact with this ambient. The translation is parameterized by the interface e of the current enclosing ambient. A down link to a subambient named b with interface e_b and unique identifier (uid) j is represented as a message $amb(j,b,e_b)$. For every ambient, the up link to its parent ambient is represented by the parent interface e, which is stored in the state message $s(a,i,e,l)$. In addition, the state message contains the name a and the current uid i of the ambient, and the log l of In and Out actions that have been delegated to the parent ambient using e.

We resume our study of an In-action, considering the role of each message in the translation of $c[\ a[\,\texttt{in}\ b.Q\,]\ \mid\ b[\,0\,]\,]$. Initially, the translation defines a continuation κ for Q and issues a message $in(b,\kappa)$ in a, which is a subjective migration request into an ambient named b.

The first step consists of delegating the request to the parent ambient. Using the first rule of a's D_0, the messages $s(a,i,e,l)$ and $in(b,\kappa)$ are consumed, the request is

$$
\begin{array}{rcl}
[\![\texttt{open}\ a.P]\!]_e & = & \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.open(a,\kappa) \\
[\![\langle n \rangle]\!]_e & = & e.snd(n) \\
[\![(n).P]\!]_e & = & \texttt{def}\ \kappa(n) \triangleright [\![P]\!]_e\ \texttt{in}\ e.rcv(\kappa) \\
[\![!P]\!]_e & = & \texttt{def}\ \kappa() \triangleright [\![P]\!]_e \mid \kappa()\ \texttt{in}\ \kappa() \\
[\![\nu a.P]\!]_e & = & \texttt{def}\ \texttt{fresh}\ a\ \texttt{in}\ [\![P]\!]_e
\end{array}
$$

with additional rules in the definition of $AM_{a,e}(P)$:

$$
\begin{array}{rcl}
D_1' & \stackrel{\text{def}}{=} & s(a,i,e,l) \mid amb(j,b,e_b) \mid open(b,\kappa) \triangleright s(a,i,e,l) \mid e_b.opening(\kappa) \\
D_2' & \stackrel{\text{def}}{=} & s(a,i,e,l) \mid opening(\kappa) \triangleright f(e) \mid \kappa() \mid Flush(l, e.in, e.out, \kappa) \\
D_C & \stackrel{\text{def}}{=} & s(a,i,e,l) \mid rcv(\kappa) \mid snd(n) \triangleright s(a,i,e,l) \mid \kappa(n) \\
D_F & \stackrel{\text{def}}{=} & f(e) \mid in(b,\kappa) \triangleright f(e) \mid e.in(b,\kappa) \\
 & \wedge & f(e) \mid out(b,\kappa) \triangleright f(e) \mid e.out(b,\kappa) \\
 & \wedge & f(e) \mid open(b,\kappa) \triangleright f(e) \mid e.open(b,\kappa) \\
 & \wedge & f(e) \mid amb(j,b,e_b) \triangleright f(e) \mid e.amb(j,b,e_b) \\
 & \wedge & f(e) \mid sub_{in}(k,b,\kappa) \triangleright f(e) \mid e.sub_{in}(k,b,\kappa) \\
 & \wedge & f(e) \mid sub_{out}(k,b,\kappa) \triangleright f(e) \mid e.sub_{out}(k,b,\kappa) \\
 & \wedge & f(e) \mid rcv(\kappa) \triangleright f(e) \mid e.rcv(\kappa) \\
 & \wedge & f(e) \mid snd(b) \triangleright f(e) \mid e.snd(b) \\
 & & \\
D & \stackrel{\text{def}}{=} & D_0 \wedge D_1 \wedge D_1' \wedge D_2 \wedge D_2' \wedge D_C \wedge D_F
\end{array}
$$

with the extended record notation

$$
e_h \stackrel{\text{def}}{=} \left\{ \begin{array}{l} here = here, amb = amb, sub_{in} = sub_{in}, sub_{out} = sub_{out}, open = open, \\ reloc = reloc, in = in, out = out, opening = opening, rcv = rcv, snd = snd \end{array} \right\}
$$

Figure 2: Additional clauses for the full translation

recorded in a's log as an entry $\text{In}\ b\ \kappa$, and the request is forwarded to the enclosing ambient c described by the interface e. The ambient a remains active, with new state $s(a,i,e,l \cup \{\text{In}\ b\ \kappa\})$. In parallel, the message $e.sub_{in}(i,b,\kappa)$ is a subambient move request sent to c, with the explicit identifier i of the requester a.

The second step is performed by c's ambient manager. The message $sub_{in}(i,b,\kappa)$ may be consumed using the first rule of D_1. The rule also requires that both the ambient that issued the request and another destination ambient with name b be actually present. This step removes the down link for the moving ambient—the message $amb(i,a,e_a)$—, thus blocking other actions competing for the same message, whereas the destination ambient remains available for concurrent steps. A relocation message $e_a.reloc(e_b,\kappa)$ is emitted, signalling to the requesting ambient a that it must migrate to the ambient with interface e_b, with continuation κ.

The third step, using a's rule D_2, consumes the message on $reloc$ and the current state message, performs a join calculus migration to the location of the destination ambient, then resumes its activities using the new interface e_b. To this end, the process I_{a,e_a,e_b} restores an active state: it generates a fresh uid i', issues a local state message $s(a,i',e_b,\emptyset)$ representing the up link, and sends to the new parent a message $e_b.amb(i',a,e_a)$ representing the down link. (Since no down link will

ever mention the previous uid i, previous delegation messages mentionning i will never match a rule of D_1. In our implementation, these stale messages are actually discarded.) In addition, the message $\kappa()$ trigger the continuation. Finally, the process $Flush(l \cup \{\text{IN } b\ \kappa\}), in, out, \kappa)$ restarts any preempted actions appearing in the log. As defined in appendix A, this process emits a message $in(d, \kappa')$ or $out(d, \kappa')$ in a for every entry IN $d\ \kappa'$ or OUT $d\ \kappa'$ appearing the log l. These entries correspond to actions preempted by the migration; they will be delegated to a new parent through other iterations of 0-steps of the algorithm.

Similarly, an OUT-step is performed according to the algorithm by using the second rule of D_0 of the moving ambient, the second rule of D_1 of the enclosing ambient, and finally the rule D_2 of the moving ambient.

2.3 Dealing with other ambient constructs

The translation of Figure 2 generalizes the translation above to the full ambient calculus. For each additional construct, we add a clause to the compositional translation $[\![\cdot]\!]_e$. We also upgrade $AM_{a,e}(\cdot)$, and use a larger environment e with extra fields for *open*, *opening*, *rcv*, and *snd*. New constructs to consider are:

Values and Scopes. Ambient names are mapped to identical names in the join calculus. The two calculi rely on similar lexical scope disciplines, with scope extrusion performed by structural equivalence (rule SCOPE in join, rules R1 and R2 in ambients). Thus, it suffices to translate the creation of local ambient names $\nu a.P$ into the creation of binders `fresh` a in join-calculus definitions.

Communication. Ambient communication is implemented by supplementing every ambient manager with a rule D_C that binds two channels *snd* and *rcv* and synchronizes message outputs and message requests. This encoding is much like the encoding of pi-calculus channels into the join calculus (see [8]).

Replication. Each replicated process $!P$ is coded using a standard recursive encoding of infinite loops in the join calculus.

Open. Ambient processes may dissolve ambient boundaries using the `open` a capabilities. In contrast, join calculus names are statically bound to their defining location, and location boundaries never disappear. We thus lose the one-to-one mapping from ambients to locations, and distinguish two states for each AM location: either the ambient is still running—and the message s is present, or it has been opened—and henceforth messages sent to its interface are forwarded to the ambient that performed the open. The indirection is achieved through the use of message f. This leads to complications in the proofs, as one must prove that these opened locations do not interfere with the rest of the translation.

3 Correctness of the translation

The distributed synchronization algorithm significantly departs from the operational semantics of ambients; besides, the translation of nested ambients may yield arbitrarily large terms with numerous instances of the algorithm running in parallel. This makes the correctness of the distributed implementation problematic. On the other hand, both calculi have a reduction-based semantics, which can be equipped

with standard notions of observables. This provides a precise setting for establishing correctness on the translation, rather than on an abstraction of the algorithm. (Of course, there are still minor discrepancies between the translation and the actual code in Jocaml; see section 6.)

We first define a syntactic notion of observation. For each calculus, we use a family of predicates on processes indexed by names, written $P \downarrow_b$.

- In the ambient calculus, $P \downarrow_b$ when b is free in P and $P \equiv \nu\widetilde{v}.\, b[\, Q\,] \mid R$.
- In the join calculus, $P \downarrow_b$ when b is free in P and $P \equiv \alpha[D' : b(\widetilde{v}) \mid P'] \wedge D$.

Next, we express the correctness of the translation in terms of the following predicates, for both ambient and join processes:

- A process P has a *weak barb* on b (written $P \Downarrow_b$) when $P \to^* P' \downarrow_b$.
- A process P *diverges* (written $P \Uparrow$) when P has an infinite series of steps.
- A process P has a *fair-must barb* on b (written $\Box P \Downarrow_b$) when for all P' such that $P \to^* P'$, we have $P' \Downarrow_b$.

In combination, these predicates give a precise contents to the informal notion of correctness: "the translation should neither suppress existing behaviors, nor introduce additional behaviors." The minimal notion of correctness for an implementation is the reflection of weak barbs, which rules out spurious behaviors; the converse direction states that the implementation does not discard potential behaviors; for instance, it rules out deadlocks, or even an empty translation. The preservation of convergence is of pragmatic importance. In addition, correctness for fair-must tests relates infinite computations, and rules out implementations with restrictive scheduling policies.

Since top-level ambients are not translated into messages on free names, the observation of translated ambients requires some special care. To this end, we supplement the top-level translation with a mechanism that can test for a particular barb $\Downarrow_b$ in the source calculus. For a given ambient interface e, we let D and Q be defined by $[\![\, a[\, \mathbf{0}\,]\,]\!]_e = \mathtt{def}\ \mathit{here}[D : Q]\ \mathtt{in}\ \mathbf{0}$, and let e_0 be the record obtained by substituting the empty word for $e_0.\mathit{here}$ in the interface e_h defined by D. We use the definition $D_t \stackrel{\text{def}}{=} s(a,i,e,l) \mid \mathit{amb}(j,b,e_b) \mid t(b) \triangleright s(a,i,e,l) \mid \mathit{amb}(j,b,e_b) \mid \mathit{yes}()$ and write $[\![\, \cdot \,]\!]^b$ for the translation that maps every process P to the process $[D \wedge D_t : Q \mid t(b) \mid [\![P]\!]_{e_0}]$. Without loss of generality, we always assume that the names in a, t, e, yes do not clash with free names of P, and that the location and channel names of the translation do not clash with any name of P.

We are now ready to state that all the derived observations discussed above are preserved and reflected by the translation:

Theorem 3.1 *For every ambient process P and name b, we have*

1. *$P \Downarrow_b$ if and only if $[\![P]\!]^b \Downarrow_{yes}$;*
2. *$P \Uparrow$ if and only if $[\![P]\!]^b \Uparrow$;*
3. *$\Box P \Downarrow_b$ if and only if $\Box[\![P]\!]^b \Downarrow_{yes}$.*

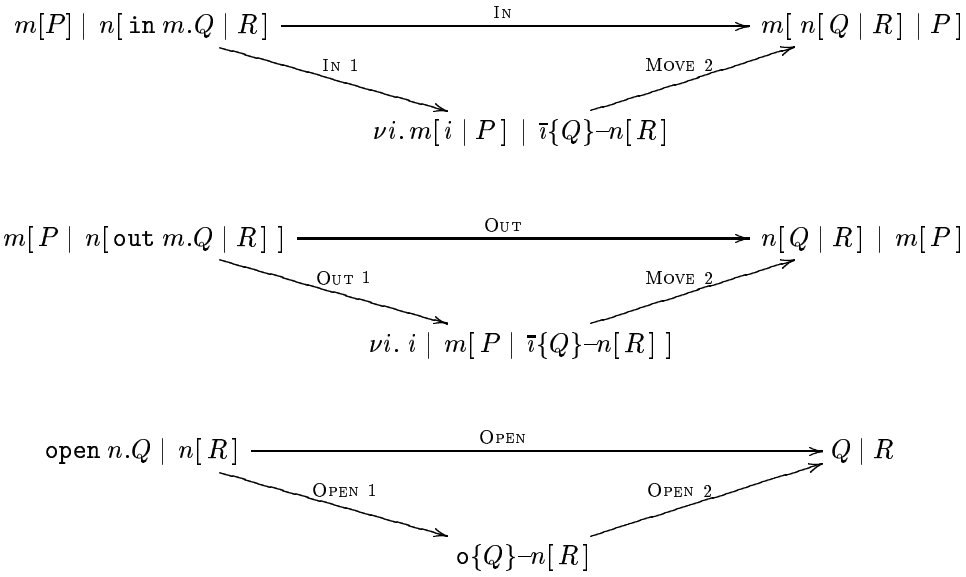
While correctness is naturally expressed in terms of observables along the reduction traces of processes, its proof is challenging. In particular, a direct approach clearly leads to intractable inductions on both the syntax of the source process and on the series of reductions. The next two sections present our proof strategy; a proof of theorem 3.1 appears at the end of appendix B.2.

4 A calculus of Ambients extended with transient states

In order to prove theorem 3.1, we introduce an intermediate calculus of ambients with constructs that materialize the key transient states of the algorithm of section 2.1, and we equip this calculus with a reduction semantics in direct correspondence with the algorithm. For instance, atomic IN steps are decomposed into series of 1- and 2-steps. (However, 0-steps are not represented, inasmuch as requests are always eventually delegated to the current parent.) In the next section, we rely on the extended calculus to establish correctness as the composition of two equivalences. First, we will use *coupled simulations* [18] to relate the two semantics for ambients; then, we will use bisimulations to relate ambients equipped with the extended semantics to their join calculus translations.

The grammar for the extended calculus appears in Figure 3. It has new processes representing ambients which are committed to move or to be opened, as the result of their father's 1-step—we call such transient ambients *stubs*—and also new processes marking the future position of migrating ambients—we call such precursors *scions*. Pairs of stubs and scions are syntactically connected by a marker i.

The extended operational semantics appears in Figure 4. (The term $X^{=}a[P]$ stands for either $Xa[P]$ or $a[P]$.) It is a reduction-based semantics with auxiliary labels for stubs and scions. Each of the reduction steps IN, OUT, and OPEN is decomposed in two steps, as depicted below. Initial steps $\rightarrow_1$ introduce stubs and scions; completion steps $\rightarrow_2$ consume them.



P	::=		extended ambient process
		$\ldots$	*all the constructors of Figure 5* on page 19
		$\mid \quad Xa[P]$	stub
		$\mid \quad i$	scion
		$\mid \quad \nu i.P$	marker restriction
X	::=		state extension
		$\overline{i}\{P\}$–	the stub is committed to move to i
		$\mid \quad \mathtt{o}\{P\}$–	the stub is being opened

Well-formed conditions: stubs and scions may occur only in extended evaluation contexts; restricted markers i must be used linearly (exactly one stub and one scion).

Figure 3: Syntax for an ambient calculus extended with transient states

Extended evaluation contexts $E(\cdot)$ are defined by the grammar

$$E(\cdot) ::= \cdot \quad \mid \quad P \mid E(\cdot) \quad \mid \quad E(\cdot) \mid P \quad \mid \quad X^{=}a[E(\cdot)] \quad \mid \quad \nu a.E(\cdot) \quad \mid \quad \nu i.E(\cdot)$$

Structural equivalence $\equiv$ is the smallest equivalence relation on processes that is closed by application of extended evaluation contexts and by α-conversion, and that satisfies the axioms P0, P1, P2, C0, C1, R1, R2 of figure 6 and:

$$\text{R2X} \ \frac{a \neq n \quad a \text{ not free in } X}{Xn[\nu a.P] \equiv \nu a.Xn[P]}$$

Labeled transitions $\xrightarrow{\alpha}$ is the smallest family of relations closed by application of restriction-free extended evaluation contexts and such that

$$\textsc{Stub} \ \ \overline{i}\{Q\}\text{–}n[R] \xrightarrow{\overline{i}.n[Q|R]} \mathbf{0} \qquad \textsc{Scion} \ \ i \xrightarrow{i.P} P \qquad \textsc{Scope} \ \ \frac{P \xrightarrow{\alpha} P' \quad v \text{ not free in } \alpha}{\nu v.P \xrightarrow{\alpha} \nu v.P'}$$

Reduction steps $\rightarrow$ are defined as in Figure 6 with extended evaluation contexts. *Initial steps* $\rightarrow_1$*, Completion steps* $\rightarrow_2$*, and Other steps* $\rightarrow_C$ are the smallest relations closed by structural equivalence, by application of extended evaluation contexts, and such that:

$$\textsc{In 1} \ \ \begin{array}{ll} & m[P] \mid n[\mathtt{in}\, m.Q \mid R] \\ \rightarrow_1 & \nu i.\ m[i \mid P] \mid \overline{i}\{Q\}\text{–}n[R] \end{array} \qquad \textsc{Out 1} \ \ \begin{array}{ll} & X^{=}m\big[n[\mathtt{out}\, m.Q \mid R] \mid P\big] \\ \rightarrow_1 & \nu i.\ i \mid X^{=}m\big[\overline{i}\{Q\}\text{–}n[R] \mid P\big] \end{array}$$

$$\textsc{Move 2} \ \ \frac{P \xrightarrow{\overline{i}.S} P' \quad Q \xrightarrow{i.S} Q'}{\nu i.(P \mid Q) \rightarrow_2 P' \mid Q'}$$

$$\textsc{Open 1} \ \ \mathtt{open}\, n.Q \mid n[R] \rightarrow_1 \mathtt{o}\{Q\}\text{–}n[R] \qquad \textsc{Open 2} \ \ \mathtt{o}\{Q\}\text{–}n[R] \rightarrow_2 Q \mid R$$

$$\textsc{Recv} \ \ \langle a \rangle \mid (x).P \rightarrow_C P\{^{a}/_{x}\} \qquad \textsc{Repl} \ \ !P \rightarrow_C P \mid !P$$

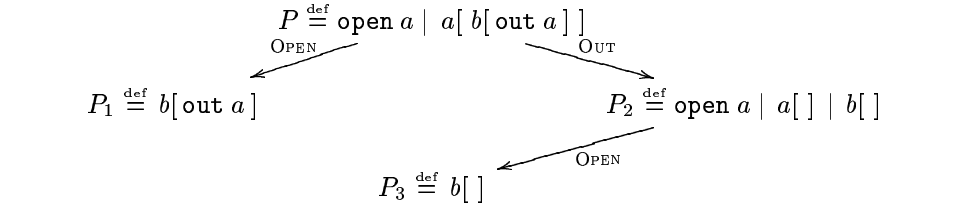
Figure 4: Semantics for an ambient calculus extended with transient states

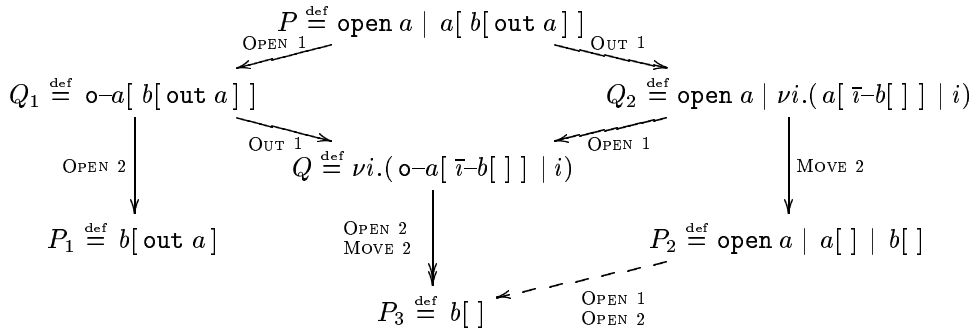
The reduction rules for RECV and REPL are those of the original semantics, except that we write $\rightarrow_C$ instead of $\rightarrow$. Overall, we obtain a reduction system for extended ambient processes with steps $\rightarrow_{12C} \stackrel{\text{def}}{=} \rightarrow_1 \cup \rightarrow_2 \cup \rightarrow_C$. For the sake of comparison, we also extend the original reduction semantics and the observation predicates from ambients to extended ambients. The definition of observation remains unchanged, that is, the predicates $\downarrow_b$ is insensitive to extended constructs.

Initially, stubs and scions are neighbors, but they may drift apart as the result of other steps before performing the matching completion step, so an auxiliary labeled transition system is used to match stubs and scions. The completion of a deferred migration is thus rendered as a global communication step between processes residing in two different ambient contexts, and syntactically linked by the names $\overline{i}$ and i. This may seem difficult to implement, but actually scions i have no operational contents; they represent the passive target for a strong migration. (In conjunction with labeled transitions, rule MOVE 2 demands that all scope extrusions be performed before completion.)

To illustrate the extended calculus, we give below the reductions for a process P with a critical pair. In P, either a is opened first, and b retains its $\mathtt{out}\ a$ action ($P \rightarrow P_1$), or b exits a first, and b is left empty ($P \rightarrow\!\!\rightarrow P_3$).

The first diagram represents the steps for the original semantics; the second diagram represents the steps for the extended semantics. Processes P, P_1, P_2, P_3 are regular ambient processes; processes Q, Q_1, Q_2 are transient processes reflecting intermediate states of our algorithm.





5 Coupled simulations and operational correspondence

We continue our study of correctness in terms of equivalences based upon weak barbs. These equivalences are essential to obtain a modular proof. As a side benefit, they also provide a finer account of correctness. (See [7] for a discussion of equivalences

and encodings in process calculi.) Instead of equivalences, we actually often use relations ranging over different domains, equipped with different notions of reduction steps and observations.

Definition 5.1 (Barbed Bisimulations) *A relation $\mathcal{R} \in \mathcal{P} \times \mathcal{P}'$ is a weak barbed simulation when, for all $P\ \mathcal{R}\ Q$, we have*

1. *if $P \to^* P'$, then there exists Q' such that $Q \to^* Q'$ and $P'\ \mathcal{R}\ Q'$;*
2. *if $P \Downarrow_b$, then $Q \Downarrow_b$.*

$\mathcal{R}$ is a barbed bisimulation when $\mathcal{R}$ and its converse $\mathcal{R}^{-1}$ are barbed simulations.

Bisimulations come with effective proof techniques that consider only a few steps at a time, rather than whole execution traces. Unfortunately, barbed bisimilarity $\approx$—the largest barbed bisimulation closed by application of evaluation contexts—is too discriminating for our protocol. Transient processes such as Q_1 in the example above account for a partially-committed internal choice: Q_1 may reduce to P_1 and P_3, but not to P_2. In fact, P_1, P_2, and P_3 are separated by $\approx$ ($P_2 \not\approx P_3$ is easily established using the simple context $(\cdot) \mid a[\ c[\ ]\ ]$), hence the extended process Q_1 is not bisimilar to *any* P, P_i of the example. (Conversely, we have here $Q_2 \approx P_2$ and $Q \approx P_3$, but for other processes with reductions within stubs, this may not be the case.)

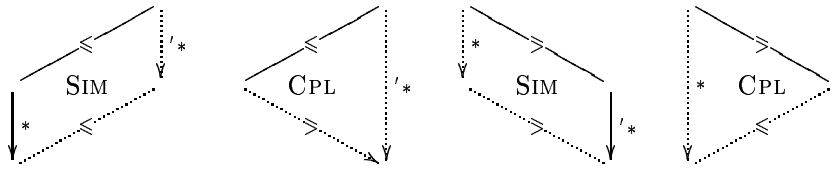
To address this issue of gradual commitment, Parrow and Sjödin proposed coarser relations called coupled simulations [18, 16]. We liberally adapt their definition to ambients:

Definition 5.2 (Coupled simulations) *The relations $\leqslant \in \mathcal{P} \times \mathcal{P}'$ and $\leqslant \in \mathcal{P}' \times \mathcal{P}$ form a pair of* barbed coupled simulations *when $\leqslant$ and $\leqslant$ are barbed simulations that meet the coupling conditions:*

1. *if $P \leqslant Q$, then for some Q' we have $Q \to^* Q'$ and $Q' \leqslant P$;*
2. *if $Q \leqslant P$, then for some P' we have $P \to^* P'$ and $P' \leqslant Q$.*

In the initial definition of coupled simulations of [18], coupling conditions are required only for pairs of stable processes. As in [7], our definition is actually the barbed counterpart of the *weakly-coupled simulations* considered in [19].

In the proofs, we will rely on diagrams for representing properties of related processes. As usual, edges in the diagrams represent relations among processes: solid edges stand for universally-quantified relations (the premises); dotted edges stand for existentially-quantified relations (the conclusions). For instance, the definition above can be restated using four diagrams:



The discrepancy between $\leqslant$ and $\geqslant$ is most useful for handling transient states such as Q_1. The coupling conditions guarantee that every transient state can be mapped both to a less advanced state and to a more advanced one. In our case, we would have $Q_1 \leqslant P$ and $P_i \leqslant Q_1$ for $i = 1, 3$.

5.1 Correctness of the asynchronous algorithm.

The first stage of our correctness argument is expressed as a coupled-simulation between ambient processes equipped with the original and the extended semantics. In the statement below, related processes have the same syntax, but live in different calculi, equipped with different reduction semantics.

Theorem 5.3 *Let $\lesssim$ be the union of $\leqslant \cap \geqslant$ for all barbed coupled simulations between ambient and extended ambient processes that are closed by application of evaluation contexts.*

For all ambient processes P, we have $P \lesssim P$.

The proof is detailed in Appendix B.1; it makes apparent some subtleties of our algorithm due to additional concurrency. After a first series of results on partial commutation properties for extended steps, the key lemmas establish that, for any ambient process P and extended process Q, if $P \rightarrow^*_{12C} Q$, then (1) for some ambient process P' we have $Q \rightarrow^*_{12C} P'$ and (2) for any such process P', we also have $P \rightarrow^* P'$ in the original semantics.

A sketch of the proof for (2) is as follows: by induction on the number of extended steps, we use partial commutation properties of 1- and 2-steps to bring forward a pair of matching steps $\rightarrow_1\rightarrow_2$, which can trivially be replaced with $\rightarrow$. We rely on an iterative procedure for selecting an adequate pair of steps. When the series begin with a 1-step, we partition the following steps into external steps (steps that involve ambients that are external to the stub created by the first step), entering steps (MOVE 2-steps traversing this stub), and internal steps. If there are internal 1-steps, we show that the first one commutes with previous steps and can be brought forward at the beginning of the series. We repeat the procedure for this new 1-step. Eventually, we obtain a 1-step followed by external steps only, followed by a matching $\rightarrow_2$ step, and we conclude by showing that this $\rightarrow_2$ can be brought in second position.

5.2 Operational correspondence.

The second stage of the proof relates ambients equipped with the extended semantics to their join calculus translations. It is simpler than the first one, in principle, but its proof is complicated because the translation makes explicit many details of the implementation that are inessential to the algorithm.

In order to express the correspondence of observations across the translation, we supplement the top-level translation $[\![\cdot]\!]^b$ of theorem 3.1 with an external choice of the ambient barb to be tested. With the same notations, we write $[\![\cdot]\!]^t$ for the translation that maps every process P to the process $[D \wedge D_t : Q \mid p(t) \mid [\![P]\!]_{e_0}]$. As before, we assume that names in a, e, p, t, and *yes* do not clash with names free in P. At any point, we can use the evaluation context $T_b(\cdot) \stackrel{\text{def}}{=} h_T[p(t) \triangleright t(b) : \mathbf{0}] \wedge (\cdot)$ to test a translated ambient barb on b by testing the plain join calculus barb $T_b(\cdot) \Downarrow_{yes}$. (We have $[\![P]\!]^b \approx T_b([\![P]\!]^t)$ in the join calculus.)

Theorem 5.4 (Correctness of the translation) *Let $\approx^{aj}$ be the largest bisimulation between extended ambient processes with reductions $\rightarrow_{12C}$ and join processes such that $Q \approx^{aj} R$ implies $Q \Downarrow_b$ iff $T_b(R) \Downarrow_{yes}$.*

For all ambient processes P, we have $P \approx^{aj} [\![P]\!]^t$.

Theorem 5.4 is a corollary of theorem B.31 proved in the appendices. This latter theorem is stated as a *strong bisimulation up to bookkeeping* for the translations

of all reachable extended ambient processes. Since every significant transient state induced by the translation has been lifted to the extended ambient calculus, its proof essentially amounts to an operational correspondence between the two calculi. We partition reductions in the join calculus according to the static rule of the translation being used. For instance, we let $\rightarrow_1$ steps in the join calculus be the steps using a rule of a definition $D_1 \wedge D_1'$ of figure 2; these steps create a *reloc* or an *opening* message, and are in direct operational correspondence with source $\rightarrow_1$ steps. We obtain two main classes of join calculus steps: steps $\rightarrow_{12C}$ that can be traced back to extended ambient steps, and "bookkeeping" steps $\rightarrow_B$, which are auxiliary steps used to trigger continuations, manage the logs, or unfold new ambient managers.

The main lemmas describe dynamic simplifications of derivatives of the translation, which are required to obtain translations of derivatives in the extended source calculus. These lemmas are expressed as elementary commutation diagrams between simplification relations and some families of reduction steps. For instance, one lemma states that "stale messages" can be discarded; another, more complex lemma states that locations and ambient managers representing opened ambients can be eliminated, effectively merging the contents of opened ambients with the contents of their previously-enclosing ambient. To conclude, we exhibit a bisimulation relation between extended ambients equipped with steps $\rightarrow_{12C}$ and global translations of these extended ambients equipped with steps $\rightarrow_B^* \rightarrow_{12C} \rightarrow_B^*$. The whole proof is structured using the decreasing diagram technique of [17], whose conditions guarantee that every weak simulation diagram in the final proof can be obtained by gluing previously-established diagrams.

6 Distributed Implementation

In this section, we briefly describe the actual implementation, and we discuss the distribution of ambient programs at run-time. We refer to [11] for the source code, setup instructions, and examples.

6.1 The implementation in Jocaml

Our implementation closely follows the translation given in figures 1 and 2. Since Jocaml already provides support for mobility, local synchronization, and run-time distribution, our code is very compact—less than 400 lines for the interpreter, less than 40k in bytecode for the object files. The main differences between the formal translation and the code are given below:

- Messages in the implementation may pass names, but also arbitrary chains of capabilities, as in $\langle \texttt{in}\ a.\texttt{out}\ b \rangle \mid !(x).x.\langle x \rangle$ (cf. [5]).
- The implementation is an interpreter, rather than a global translation. Accordingly, it maintains an environment for local variables, and it performs dynamic type checking when a value is used either as a name or as a capability.
- The translation relies on non-linear join-patterns, which are not available in Jocaml. More explicitly, the implementation relies on hash tables to cache some messages: when a new message arrives on sub_{in}, sub_{out}, or *open*, the interpreter attempts to perform a step at once, in combination with previously-cached *amb* messages. Otherwise, the message is queued. Similarly, when a new message

arrives on *amb*, it is added to the cache, and the queue of deferred messages is scanned for enabled steps.

- The formal translation of replication always yields a diverging computation. More reasonably, the interpreter unfolds replication on demand: since every ambient reduction involves at most two copies of a replicated process, it suffices to initially unfold two copies, then to unfold an additional copy whenever a fresh copy is used or modified. Hence, $!\,a[\,]$ does not diverge, while $!\,a[\,\texttt{in}\ a\,]$ still does.

6.2 Controlling distribution at run-time

While ambients and locations have the same hierarchical structure, their interpretation in terms of physical distribution are significantly different.

By design, the join calculus induces a mostly "horizontal" model of distribution at run-time. Every site executing a Jocaml runtime implements a site-as-location, with its own tree of sublocations. Due to transparency, every pair of locations can interact as if they were simply running in parallel, independently of their host site. Practically, every migration also guarantees that the moving location and the new parent location are running on the same site, until the moving location migrates again.

The ambient calculus does not explicitly account for distribution, but intuitively ambients representing firewalls, administrative domains, or routers would be running at different sites, and their contents would also be mapped to different sites. Besides, the rich synchronization model within each ambient leads to a centralized implementation for every ambient manager, hence the use of remote subambients to decompose a computation into more asynchronous fragments. Overall, the ambient calculus induces a mostly "vertical" model of physical distribution.

In order to describe the distribution of an ambient computation, it is natural to annotate every process with a "site" label, but some delicate design decisions arise for the dynamics of distribution. Indeed, for most reduction steps there are several ways to relabel the processes affected by the step. For instance, consider the process $a[\,P\,] \mid b[\,\texttt{in}\ a.Q \mid R\,]$, and assume that a and b run at different sites named s_a and s_b. After the IN-step, b can run either at site s_a or s_b; in the case it runs at s_a, the same question arises for every subambient of b. Consider now the process $c[\,a[\,P \mid b[\,\texttt{out}\ a.Q \mid R\,]\,]\,]$. After the OUT-step, b can run at either s_b, or s_a, or s_c. Similarly, $b[\,\texttt{open}\ a.P \mid a[\,Q\,]\,]$ may lead to a process Q running at either s_a or s_b. Such choices define the dynamics of distribution. Inasmuch as every rearrangement of the ambient-as-location tree is enabled, our implementation can in principle accommodate any such choice by interleaving *go* statements and message passing. This does not alleviate the need for constructs in the ambient calculus to control these choices.

We arrive at the following model of distribution for our implementation. A global tree of running ambients is partitioned into several trees of locations running at different sites. Except for the location representing the top-level ambient, each top-level location in Jocaml is logically attached to a parent ambient running at another site. Within a given configuration, the execution of ambient reductions may involve messages coming from several sites, and may cause locations representing the affected ambients to migrate from one site to another, in accordance to the partition. Independently, the distributed configuration can be expanded by starting new ambient

interpreters at other sites. To this end, each interpreter is given an ambient process P to execute, and a pointer to an existing ambient at which P is logically started.

For instance, writing `x://jam code &` for the command that starts an interpreter at site `x` running the ambient process `code`, and using integers as site identifiers, consider the computation initiated by the commands

```
2://jam "public a[0] " &
1://jam "b[0]" -amb a &
3://jam "c[ m[out c.in b] ]" -amb a &
```

In the first command, the keyword `public` registers ambient a as an entry point for other interpreters; in other commands, `-amb a` refers to this entry point. The distributed computation proceeds as follows:

$$
\begin{array}{rlcccc}
 & a\Big[& \text{site 1} & | & \text{site 2} & | & \text{site 3} & \Big] \\
 & a\Big[& b[^{Db}\ \mathbf{0}] & | & ^{Da}\ \mathbf{0} & | & c[^{Dc}\ m[^{Dm}\ \texttt{out}\ c.\texttt{in}\ b]\ ] & \Big] \\
\to & a\Big[& b[^{Db}\ \mathbf{0}] & | & ^{Da}\ \mathbf{0} & | & m[^{Dm}\ \texttt{in}\ b]\ |\ c[^{Dc}\ \mathbf{0}] & \Big] \\
\to & a\Big[& b[^{Db}\ m[^{Dm}\ \texttt{in}\ b]\ ] & | & ^{Da}\ \mathbf{0} & | & c[^{Dc}\ \mathbf{0}] & \Big]
\end{array}
$$

where, for each translated ambient a, b, c, and m we represent its Jocaml ambient manager as D_a, D_b, D_c, and D_m. (We omit the Jocaml name-server used to bootstrap remote communications.)

During an initial registration stage, sites 1 and 3 receive the parent interface of a and, since a is distributed, patch this interface so that its apparent location is the local site. In particular, the ambient managers for b and c each send a message to D_a that manifest their presence as a subambient of a, while the code within b and c is run locally. For instance, the m ambient within c can perform its out action without synchronizing with a. As a result of the migration, two new messages are sent to D_a, a *amb* message and a sub_{in} request. At D_a, the synchronization occurs, and D_m is sent back the interface of b. Later, a location of site 3 will migrate toward a location of site 1, without involving site 2 anymore.

7 Conclusions

We translated Mobile Ambients into the join calculus, and gave a first, asynchronous, distributed implementation of the ambient calculus in Jocaml, with a high level of concurrency. The synchronization mechanisms of Ambients turned out to be challenging first to implement, then to prove correct. This provides an insight into the ambient calculus as a model of concurrency. At the same time, this shows how Jocaml and its formal model can be used to tackle distributed and mobile implementations. For instance, the translation takes full advantage of join patterns to describe complex local synchronization steps, while a more traditional language would decompose these steps into explicit series of reads and updates protected by locks.

In order to get a safer and more efficient implementation, one should care about typing information for passed values and mobility capabilities [6, 4]. Our implementation insures dynamic type-checking on values, whereas it would be preferable to use the static type-checking discipline of Jocaml. Similarly, static knowledge of the

actions that can appear in a given ambient can lead to more efficient, specialized ambient managers.

Finally, little is known about actual programming in ambient, or the relevant abstractions to build a high-level language on top of the ambient calculus. While we did not consider changing the source language, we believe that our implementation provides an adequate platform for experimenting with ambient-based language design. For instance, our translation would easily accomodate the co-capabilities proposed in [14].

Acknowledgments. This work benefited from discussions with Luca Cardelli, Fabrice Le Fessant, and Luc Maranget.

References

[1] G. Boudol. Asynchrony and the π-calculus (note). Technical Report 1702, INRIA Sophia-Antipolis, May 1992.

[2] L. Cardelli. Ambit, 1997. Available from `http://www.luca.demon.co.uk/Ambit/Ambit.html`.

[3] L. Cardelli. Mobile ambient synchronization. Technical note 1997-013, Digital Systems Research Center, July 1997.

[4] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *ICALP'99*, LNCS, 1999.

[5] L. Cardelli and A. Gordon. Mobile ambients. In *FoSSaCS '98*, volume 1378 of *LNCS*, pages 140–155, 1998.

[6] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *POPL'99*, pages 79–92. ACM, Jan. 1999.

[7] C. Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, Ecole Polytechnique, Palaiseau, Nov. 1998. INRIA, TU-0556.

[8] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *POPL '96*, pages 372–385. ACM, Jan. 1996.

[9] C. Fournet and G. Gonthier. Weak bisimulations by decreasing diagrams. Draft, March 1999.

[10] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *CONCUR '96*, volume 1119 of *LNCS*, pages 406–421, Aug. 1996.

[11] C. Fournet and A. Schmitt. An implementation of Ambients in JoCAML. Software available from `http://join.inria.fr/ambients.html`, 1999.

[12] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *FoSSaCS'99*, volume 1578 of *LNCS*, pages 212–226, 1999.

[13] F. Le Fessant. The JoCAML system prototype. Software and documentation available from `http://join.inria.fr/jocaml`, 1998.

[14] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL'00*. ACM, Jan. 2000.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–40 and 41–77, Sept. 1992.

[16] U. Nestmann and B. C. Pierce. Decoding choice encodings. In *CONCUR '96*, volume 1119 of *LNCS*, pages 179–194, Aug. 1996. Revised full version as report ERCIM-10/97-R051, 1997.

[17] V. Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126:259–280, 1994.

[18] J. Parrow and P. Sjödin. Multiway synchronization verified with coupled simulation. In *CONCUR '92*, volume 630 of *LNCS*, pages 518–533, 1992.

[19] J. Parrow and P. Sjödin. The complete axiomatization of cs-congruence. In *STACS '94*, volume 775 of *LNCS*, pages 557–568, 1994.

A Two notions of mobile computations

We relate two models of distributed mobile programming, the distributed join calculus [8, 10] and the ambient calculus [5]. These two models consist of name-passing process calculi in the spirit of the π-calculus. They make explicit the spatial structure of the computation by distributing processes over a tree of nested locations (a.k.a. ambients) that stand for both sites and agents. They provide mechanisms to rearrange the location tree as part of the computation, thereby describing agent migration. But these two models address different aspects of wide-area distributed computations, and yield different interpretations of locality.

In the join calculus, *locality is transparent*; it may affect performances, and even behavior in case of partial failure, but it does not affect communication and migration capabilities. Much as in a distributed object system, locations can inter-communicate as soon as they have access to names of their interfaces, independently of their respective positions in the location tree. To retain a realistic programming model, interaction between locations is restricted to the asynchronous sending of messages or sub-locations toward a given location. Practically, location transparency is implemented by a silent routing mechanism toward the site that hosts the receiving location. Interaction is local, since synchronization always occurs within a single location, independently of its sub-locations. Overall, locations in the join calculus are adequate for high-level programming with asynchronous messages and agents. Several distributed implementations are available, e.g., Jocaml [13].

In the ambient calculus, *locality and control are tightly connected*; each ambient is an opaque box, and interactions occur only between processes in adjacent ambients. The routing from one ambient to another is explicit, and moving ambients must be aware of paths in the ambient tree; if an intermediate ambient decides to block the migration, or if the path evolves during the migration, ambients may get stuck or lost. Interaction is thus local, but the synchronization between these processes may be complex. Overall, ambients are good at expressing administrative domains, highly dynamic environments, and controlled migration in wide-area networks [5]. Cardelli implemented an earlier variant of the ambient calculus in Java [2]. In this centralized implementation, each ambient process is mapped to a Java thread that repeatedly attempts to acquire a lock for every ambient potentially involved in a reduction step (usually three locks) [3]. However this global serialization is inadequate in a distributed setting.

A.1 Operational semantics for ambients

Our syntax and semantics of the calculus of ambients are given in figures 5 and 6.

This presentation slightly differs from Cardelli and Gordon's [5] on several counts. The structural equivalence is more restrictive here; it does not introduce or remove νx binders; it operates only in evaluation contexts. The operational semantics of ambients represents the unfolding of replication as a silent reduction step rather than a structural law. Also, communicated values are just ambient names, rather than both names and chains of capabilities. (Chains of capabilities are fully supported in the Jocaml implementation, but their encoding is heavy.) In spirit, our semantics is actually closer to the harness semantics of [12].

P	::=		ambient process
		$a[\,P\,]$	ambient
	$\mid$	$P \mid P'$	parallel composition
	$\mid$	$C.P$	guarded process
	$\mid$	$\nu a.P$	name restriction
	$\mid$	$\langle a \rangle$	asynchronous message
	$\mid$	$(x).P$	message reception
	$\mid$	$!P$	replication
	$\mid$	$\mathbf{0}$	inert process
C	::=		capability
		$\texttt{in}\ a$	ingoing migration
	$\mid$	$\texttt{out}\ a$	outgoing migration
	$\mid$	$\texttt{open}\ a$	ambient dissolution

Figure 5: Syntax for the ambient calculus

Evaluation contexts $E(\cdot)$ are defined by the grammar

$$E(\cdot) ::= \ \cdot \ \mid \ P \mid E(\cdot) \ \mid \ E(\cdot) \mid P \ \mid \ a[\,E(\cdot)\,] \ \mid \ \nu a.E(\cdot)$$

Structural equivalence $\equiv$ is the smallest equivalence relation closed by application of evaluation contexts, by α-conversion, and such that

$$\begin{array}{ll} \text{P0} & P \mid \mathbf{0} \equiv P \\ \text{P1} & P \mid P' \equiv P' \mid P \\ \text{P2} & (P \mid P') \mid P'' \equiv P \mid (P' \mid P'') \end{array} \qquad \begin{array}{ll} \text{R1} & \dfrac{a \text{ not free in } P}{P \mid \nu a.Q \equiv \nu a.(P \mid Q)} \\[2ex] \text{R2} & \dfrac{a \neq n}{a[\nu n.P] \equiv \nu n.a[P]} \end{array}$$

Ambient reduction $\rightarrow$ is the smallest relation closed by structural equivalence, by application of evaluation contexts, and such that

$$\textsc{In} \quad \begin{array}{ll} & m[\,P\,] \mid n[\,\texttt{in}\ m.Q \mid R\,] \\ \rightarrow & m[\,P \mid n[\,Q \mid R\,]\,] \end{array} \qquad \textsc{Out} \quad \begin{array}{ll} & m[\,n[\,\texttt{out}\ m.R \mid R'\,] \mid Q\,] \\ \rightarrow & m[\,Q\,] \mid n[\,R \mid R'\,] \end{array}$$

$$\textsc{Open} \quad \texttt{open}\ n.Q \mid n[\,R\,] \rightarrow Q \mid R$$

$$\textsc{Recv} \quad \langle a \rangle \mid (x).P \rightarrow P\{^{a}/_{x}\} \qquad\qquad \textsc{Repl} \quad !P \rightarrow P \mid !P$$

Figure 6: Operational semantics for the ambient calculus

$$
\begin{array}{lcll}
P & ::= & & \text{join calculus process} \\
 & & \mathbf{0} & \quad \text{inert process} \\
 & | & P \mid P' & \quad \text{parallel composition} \\
 & | & x(\widetilde{y}) & \quad \text{asynchronous message} \\
 & | & \mathtt{go}(a); P & \quad \text{migration request} \\
 & | & \mathtt{def}\ D\ \mathtt{in}\ P & \quad \text{local definition} \\
 & & & \\
D & ::= & & \text{join calculus definition} \\
 & & \top & \quad \text{void definition} \\
 & | & D \wedge D' & \quad \text{composition} \\
 & | & J \triangleright P & \quad \text{reaction rule} \\
 & | & a[D : P] & \quad \text{sub-location (named } a\text{, running } D \text{ and } P) \\
 & | & \alpha[D : P] & \quad \text{top-level location (with path } \alpha\text{, running } D \text{ and } P) \\
 & & & \\
J & ::= & & \text{join pattern} \\
 & & x(\widetilde{y}) & \quad \text{message pattern} \\
 & | & J \mid J' & \quad \text{synchronization}
\end{array}
$$

Figure 7: Syntax for the distributed join calculus

Structural equivalence $\equiv$ (on both processes and definitions) is the smallest equivalence relation closed by application of contexts $\cdot \wedge \cdot$, $\cdot \mid \cdot$ and $\alpha[\,\cdot : \cdot\,]$, by α-conversion on bound names, and such that:

$$
\begin{array}{llll}
\text{P0} & P \mid \mathbf{0} \equiv P & \text{D0} & D \wedge \top \equiv D \\
\text{P1} & P \mid P' \equiv P' \mid P & \text{D1} & D \wedge D' \equiv D' \wedge D \\
\text{P2} & (P \mid P') \mid P'' \equiv P \mid (P' \mid P'') & \text{D2} & (D \wedge D') \wedge D'' \equiv D \wedge (D' \wedge D'')
\end{array}
$$

$$
\textsc{Tree}\quad \begin{array}{rl} & \alpha\big[a[D' : P'] \wedge D : P\big] \\ \equiv & \alpha.a[D' : P'] \wedge \alpha[D : P] \end{array}
\qquad
\textsc{Scope}\ \frac{\text{names defined in } D' \text{ are fresh}}{\begin{array}{rl} & \alpha[D : P \mid \mathtt{def}\ D'\ \mathtt{in}\ P'] \\ \equiv & \alpha[D \wedge D' : P \mid P'] \end{array}}
$$

Join calculus reduction $\rightarrow$ is the smallest relation on configurations that is closed by structural equivalence and such that:

$$
\textsc{Comm}\ \frac{x \text{ is defined in } D'}{\begin{array}{rl} & E \wedge \alpha[D : P \mid x(\widetilde{v})] \wedge \beta[D' : P'] \\ \rightarrow & E \wedge \alpha[D : P] \wedge \beta[D' : P' \mid x(\widetilde{v})] \end{array}}
\qquad
\textsc{Join}\ \frac{\sigma \text{ operates on message contents of } J}{\begin{array}{rl} & E \wedge \alpha[D \wedge J \triangleright Q : P \mid J\sigma] \\ \rightarrow & E \wedge \alpha[D \wedge J \triangleright Q : P \mid Q\sigma] \end{array}}
$$

$$
\textsc{Go}\ \frac{a \text{ frozen}}{\begin{array}{rl} & E \wedge \beta.b[D' : P'] \wedge \alpha.a[D : P \mid \mathtt{go}(b); Q] \\ \rightarrow & E \wedge \beta.b[D' : P'] \wedge \beta.b.a[D : P \mid Q] \end{array}}
$$

Figure 8: Operational semantics for the distributed join calculus

A.2 Operational semantics for the join calculus

Our syntax and semantics for the join calculus are given in figures 7 and 8.

A *path* α is a string of location names a, b, ... *Active locations* are locations not under a `def`. They can be nested. The path of an active sublocation $a[D : P]$ is $\alpha.a$, where α is the path of its enclosing location. A *configuration* is a conjunction of top-level locations such that every location has a unique name, such that the set of paths for all active locations is prefix-closed (i.e. active locations form a tree whose nodes are indexed by active location names), and such that every channel is defined in at most one location. In a configuration, a location with path $\alpha.a$ is *frozen* when it is the only top-level location whose path contains a.

Names can be bound either as parameters $\widetilde{y}$ in a message pattern $x(\widetilde{y})$ or as names defined in D by `def` D `in` P. The definition $a[D' : P]$ defines a and names defined in D'. A definition containing a rule with message pattern $x(\widetilde{y})$ also defines x.

To simplify the translation of section 2, we supplement the join calculus with some convenient extensions, which are easily encoded in the plain join calculus. We supplement definitions with new constructs `uid` i and `fresh` a that bind names i and a, which we use to generate unique identifiers—we could use instead dummy rules such as $i() \triangleright 0$. We also use a record notation as a shortcut for tuples of names passed in a consistent order, and an algebraic notation In b κ, Out b κ for log entries with tags In, Out and names b, κ. We use finite sets of log entries, interpreted in the standard mathematical sense.

Rather than making explicit a standard encoding of set iterators for implementing the process $Flush(l, in, out, \kappa)$, we supplement the operational semantics of the join calculus with a rule for flushing logs of messages:

$$\textsc{Flush}\quad Flush(l, in, out, \kappa) \rightarrow \prod_{\textsc{In}\ d\ \kappa' \in l \mid \kappa \neq \kappa'} in(d, \kappa') \quad \mid \prod_{\textsc{Out}\ d\ \kappa' \in l \mid \kappa \neq \kappa'} out(d, \kappa')$$

A.3 Overview for Ambients

The calculus of ambients [5] describes movements of processes through administrative domains. An ambient process P may be either the inert process $\mathbf{0}$, or a composed process $P \mid P'$, or a replicated process $!P$, or an ambient $n[\,P\,]$ with name n, or a process $\nu a.P$ with local name a, or a process $C.P$ guarded by C, or a message reception $(x).P$, or an asynchronous emission $\langle a \rangle$ of name a. Capabilities C represent potential In/Out/Open actions.

A message $\langle a \rangle$ is communicated locally inside the current ambient, where any receptor $(x).P$ can catch it. In order to send a message to a receptor in a remote ambient, the message needs to be wrapped in an ambient which moves to and is opened by the receiving ambient. As in the join calculus, ambients have a tree structure. The (invisible) root represents the network. Deeper subambients are physical or logical mobile entities. Movements are local; they depend on the local tree structure. An ambient may move to the position of its parent sibling, or of a child of its sibling ambients. It can also open a sibling ambient. An ambient migrating to a remote ambient needs to be aware of the current tree structure to perform elementary moves toward the target ambient.

As an example, consider process $Q = \langle 4 \rangle$ which sends 4 to a secure printer $P = !(x).Print_x$ (assume $Print_x$ is a built-in process printing any value assigned to x). Take the following protocol derived from the firewall example in [5]

$$(\nu w.\, w[\ k[\mathtt{out}\ w.\mathtt{in}\ k'.\mathtt{in}\ w\ ]\ \mid \mathtt{open}\ k'.P])\ \mid\ k'[\,\mathtt{open}\ k.Q\,]$$

The firewall sends the subambient of name k to Q to check its key. Then the subambient comes back into the firewall with Q and allows the interaction between P and Q.

As in the π-calculus, the structural equivalence allows scope extrusion for bound ambient names. The equivalence has also commutative and monoidal rules for $|$.

$$\begin{aligned}
& \nu w.\, w[\ k[\mathtt{out}\ w.\mathtt{in}\ k'.\mathtt{in}\ w\ ]\ \mid \mathtt{open}\ k'.P]\ \mid\ k'[\,\mathtt{open}\ k.Q\,] && (1)\\
\rightarrow\ & \nu w.\, w[\,\mathtt{open}\ k'.P\,]\ \mid\ k[\,\mathtt{in}\ k'.\mathtt{in}\ w\,]\ \mid\ k'[\,\mathtt{open}\ k.Q\,] && (2)\\
\rightarrow\ & \nu w.\, w[\,\mathtt{open}\ k'.P\,]\ \mid\ k'[\ k[\,\mathtt{in}\ w\ ]\ \mid \mathtt{open}\ k.Q] && (3)\\
\rightarrow\ & \nu w.\, w[\,\mathtt{open}\ k'.P\,]\ \mid\ k'[\,\mathtt{in}\ w \mid Q\,] && (4)\\
\rightarrow\ & \nu w.\, w[\,\mathtt{open}\ k'.P \mid\ k'[\,Q\ ]\,] &&\\
\rightarrow\ & \nu w.\, w[\,P \mid Q\,] &&\\
\rightarrow\ & \nu w.\, w[\,P \mid (x).Print_x\ \mid\ \langle 4\rangle\,] && (5)\\
\rightarrow\ & \nu w.\, w[\,P \mid Print_4\,] && (6)
\end{aligned}$$

The previous process is equivalent to (1). The k-subambient moves out of the firewall, and the process OUT-reduces to (2). Now the k-ambient can move into the k'-ambient by an IN-reduction (3). Finally the k-ambient is opened (4). The calculation continues with IN/OPEN/REPL-reductions (5). Now P and Q are inside the same w-ambient and can exchange values (6).

Hence, ambient movements are sequences of IN/OUT/OPEN reductions, which modify the tree structure of ambients. As in the join calculus, an ambient moves with all its subambients. Each IN/OUT reduction moves a subtree of the tree of ambients, but an open reduction removes a single node, and attach sons of the canceled node to their grandparent. Communication with the RECV rule is a communication on an anonymous channel inside the current ambient.

A.4 Overview of the join calculus

The join calculus [8] is a variant of the asynchronous π-calculus [15, 1] with a distributed implementation and a uniform treatment of local and remote communication channels [13]. In this paper, we consider only the distributed join calculus without failures (named join calculus thereafter) [10, 7].

Join-terms are either processes or definitions. A process P can be either the inert process $\mathbf{0}$, or a composed process $P_1 \mid P_2$, or the asynchronous sending $x(\widetilde{y})$ of message $\widetilde{y}$ onto channel x, or a migration request $\mathtt{go}(a); Q$ to move the current location to location a and then do Q, or the process Q with local definition D, written $\mathtt{def}\ D\ \mathtt{in}\ Q$.

Definitions of channels give the receptive behavior of new channels. A definition D may be either the void definition $\top$, or a composed definition $D_1 \wedge D_2$, or a reaction $J \triangleright P$ consuming messages that match the pattern J and producing P instead, or a location $\alpha[D : P]$ with path α, active definition D, and running process P. Join-patterns express reception and synchronization of messages. Namely, the join-pattern $x(\widetilde{y})$ waits for the reception of any message $\widetilde{y}$ on channel x. Similarly $J \mid J'$ waits for passing J and J'.

For instance, a counter allocator is expressed as the following join calculus process:

```
def counter (x, κ) ▷
    def count(n) | inc(κ′) ▷ count(n+1) | κ′()
     ∧  count(n) | get(κ′) ▷ count(n) | κ′(n)
    in count(x) | κ(get, inc)
in P
```

The definition of channels `get`, `inc`, and `count` is local to the definition of `counter`. In P, access to `counter` is permitted, and one may pass to it an initial value x, and a continuation κ. A possible value for P is

```
def κ1 (g, i) ▷
    def κ2 () ▷ 0 in i(κ2) | i(κ2) | g(print)
in counter(3, κ1) | counter(10, κ1)
```

where `print` is a built-in channel printing its parameter on a terminal. Two counters are created with initial values 3 and 10, and two values $n \in \{3, 4, 5\}$ and $n' \in \{10, 11, 12\}$ are printed in any order. (Abbreviations for avoiding intensive use of continuations are possible, but will not be considered in this paper.)

Our counter may be localized and mobile. In

```
def counter (user, x, κ) ▷
    def a [    count(n) | inc(κ′) ▷ count(n+1) | κ′()
            ∧  count(n) | get(κ′) ▷ count(n) | κ′(n)
             : go(user); count(x) | κ(get, inc)] in 0
in Q
```

local channels `get`, `inc`, and `count` are now bundled on a new location a and will move with it. Here the target destination `user` is a new parameter of `counter`.

Locations are organized as a tree. Intuitively, the root location is the Internet; first level sub-locations are IP addresses; deeper sub-locations are logical entities ready to move. Here, location a is a sub-location of the current location of `counter`, but will become a sub-location of `user` after moving with `go(user)` and before executing `count(x)` | κ(`get`, `inc`).

To explore the dynamics of our example, rewrite the localized-mobile counter above as `def` D_c `in` Q, where D_c is of the form

```
counter (user, x, κ) ▷
def a [D_{i,g} : go(user); count(x) | κ(get, inc)] in 0
```

and where $D_{i,g}$ consists of the two rules defining `count`, `inc`, and `get`. Suppose our example is located at the anonymous root location and that Q contains a first-level location u running the process P.

```
[⊤ : def D_c in def u[⊤ : P] in 0]
```

By erasing the `def` binder (rule Scope), this definition is equivalent to

```
[D_c : def u[⊤ : P] in 0]
```

which again by erasing the `def` binders of u, then κ_1, is equivalent to

```
[D_c ∧ u[D_{κ1,κ2} : counter(u,3,κ1) | counter(u,10,κ1) ] : 0]
```

which by considering the tree-structure (rule Tree) is also equivalent to

$[D_c$: **0**$]$ $\wedge$ u$[D_{\kappa_1,\kappa_2}$: counter(u,3,κ_1) | counter(u,10,κ_1)]

These steps are part of the structural equivalence defined on join-terms. It requires some subtleties in the treatment of bound variables introduced by the `def` construct, which binds all definitions and locations appearing in it. The structural equivalence takes also into account the monoidal laws of | in processes and $\wedge$ in definitions, and the tree structure of locations.

There are also three reduction rules. The first one (COMM) moves a message on some channel to the (unique) location of its receptor. In our example, the previous definition reduces to

$[D_c$: counter(u,3,κ_1)$]$ $\wedge$ u$[D_{\kappa_1,\kappa_2}$: counter(u,10,κ_1)$]$

The second rule (JOIN) replaces a join-pattern by its definition body. The example definition now reduces to

$[D_c$: def a$[D_{i,g}$: go(u); count(3) | κ_1(get, inc)$]$ in **0**$]$
$\wedge$ u$[D_{\kappa_1,\kappa_2}$: counter(u,10,κ_1)$]$

which is structurally equivalent to (by rules SCOPE and TREE)

$[D_c$: **0**$]$
$\wedge$ a$[D_{i,g}$: go(u); count(3) | κ_1(get, inc)$]$
$\wedge$ u$[D_{\kappa_1,\kappa_2}$: counter(u,10,κ_1)$]$

The third rule (GO) is a subjective move of the location containing the go operator to a sub-location (with same name) of the target location. The example definition reduces to

$[D_c$: **0**$]$
$\wedge$ u.a$[D_{i,g}$: count(3) | κ_1(get, inc)$]$
$\wedge$ u$[D_{\kappa_1,\kappa_2}$: counter(u,10,κ_1)$]$

which is structurally equivalent to

$[D_c$: **0**$]$
$\wedge$ u$[D_{\kappa_1,\kappa_2}$ $\wedge$ a$[D_{i,g}$: count(3) | κ_1(get, inc)$]$: counter(u,10, κ_1)$]$

At this point, we see that our counter definition D_c is still at the Internet level, but an instance of the counter is at sub-location a of location u. The calculation proceeds in parallel with the second counter, or by executing continuation κ_1.

The previous three reduction rules express the full dynamics of the join calculus. It is important to notice that mobility affects the locality of terms, but not their functionality, since bindings are preserved through the reduction rules. Another important remark is about the linearity condition of receptors and locations. In any localized definition $a[D : P]$, channel names are defined in a single location and there should be a single definition of each sub-location. This is essential in a distributed implementation, since one always knows where to send any message. The calculus has been implemented in a language named Jocaml [13].

B Proofs

B.1 Correctness of the synchronization algorithm (theorem 5.3)

In this section, we let P, P' range over ambient processes and Q, Q', R, R' range over extended ambient processes. Assuming $i \in \widetilde{x}$, we write $\widetilde{x} \setminus i$ for the tuple obtained by deletion of i in $\widetilde{x}$.

We first give the proof technique that we use to establish theorem 5.3; in a second stage, we will establish each of its requirements in a series of lemmas. (The lemma is stated is specialized for ambients, but the proof technique applies to any coupled-simulation proof for identical processes equipped with different reduction relations.)

Lemma B.1 *To establish theorem 5.3: for all ambient process P, we have $(P, \longrightarrow) \lesssim (P, \rightarrow_{12C})$, it suffices to check that*

1. *For all processes P and extended processes Q, if $P \rightarrow_{12C}^* Q$, then there is a process P' such that $Q \rightarrow_2^* P'$.*
2. *For all processes P and P', we have $P \longrightarrow^* P'$ iff $P \rightarrow_{12C}^* P'$.*
3. *For all extended processes Q and Q', if $Q \downarrow_a$ and $Q \rightarrow_2 Q'$, then $Q' \downarrow_a$.*

Proof: We define relations between processes and extended processes as follows: we let $\geqslant = \{(P,Q) | P \rightarrow_{12C}^* Q\}$ and $\leqslant = \{(P,Q) | P \leftarrow_2^* Q\}$. We prove that $(\geqslant, \leqslant)$ are coupled simulations that preserve weak barbs and that are closed by application of evaluation contexts, and thus obtain $\geqslant \cap \leqslant \subseteq \lesssim$. We conclude by composing this inclusion with $\{(P,P)\} \subseteq \geqslant \cap \leqslant$.

Since the reduction relations $\rightarrow_1$, $\rightarrow_2$, and $\rightarrow_C$ are closed by application of evaluation contexts, this is also the case for $\geqslant$ and $\leqslant$.

We show that $\geqslant$ and $\leqslant$ satisfy the four diagrams of Definition 5.2. For the relation $\geqslant$, the first simulation diagram SIM holds with no step on the left, and with the composition of the two universal edges $\rightarrow_{12C}^*$ at the bottom. The first coupling diagram CPL is a combination of hypotheses 1 and 2. For the relation $\leqslant$, the second coupling diagram holds with identity instead of $\leqslant$ at the bottom, and with the $\rightarrow_2$ steps appearing in the top $\leqslant$ relation as reduction steps on the right. To establish the second SIM diagram, we close the diagram with identical processes at the bottom by assembling a series of reduction steps on the right that contains the $\rightarrow_2$ appearing in the top $\leqslant$ relation, followed by one or two steps $\rightarrow_{12C}$ for each step $\rightarrow$ on the left.

We finally show that $\leqslant$ and $\geqslant$ preserve all barbs. If $P \geqslant Q$ and $Q \Downarrow_a$, then also $Q \rightarrow_{12C}^* Q'$ with $P \geqslant Q' \downarrow_a$, by coupling $P \rightarrow^* P' \leftarrow_2^* Q'$, by hypothesis 3 we have $P' \downarrow_a$, and thus $P \Downarrow_a$. As regards $\leqslant$, if $P \Downarrow_a$ and $P \leftarrow_2^* Q$, then $P \rightarrow^* P' \downarrow_a$, by weak simulation $Q \rightarrow_{12C}^* Q'$ with $P' \leftarrow_2^* Q'$, hence $Q \Downarrow_a$. □

Intuitively, extended ambients evolve as follows. Every $\rightarrow_1$ step transforms an ambient in evaluation context into a stub, and also introduces a scion when it is an IN 1 or an OUT 1 step. Every OPEN 2 step deletes a stub. Every MOVE 2 step deletes a stub and substitute an ambient for the corresponding scion. Other steps do not affect stubs and scions. By definition, there cannot be any stub under a guard. Moreover, some extended processes such as $\overline{i}\{Q\}$–$n[i]$ (to which MOVE 2 does not apply) are not reachable from regular ambient processes.

The next lemma makes these claims formal, in the shape of an invariant, and details the residuals of 2-step redexes along reduction steps.

Lemma B.2 *Let P be a process and Q be an extended process such that $P \rightarrow_{12C}^{*} Q$. There is a tuple of names $\widetilde{x}$ and an extended process R such that:*

1. *$Q \equiv \nu\widetilde{x}.R$ and R does not contain restrictions $\nu x.T$ in evaluation context.*
2. *For every stub in R, either R is of the form $E(i)(\overline{\imath}\{P_i\}\text{–}n[\,R_i\,])$ for some i in $\widetilde{x}$ and two-hole extended evaluation context $E(\cdot)(\cdot)$, or R is of the form $E(\,\mathrm{o}\{P_o\}\text{–}n[\,R_o\,])$ for some extended evaluation context $E(\cdot)$.*
3. *Let $\prec$ be the relation on extended ambients defined as $j \prec i$ when there are evaluation contexts $E(\cdot)$ and $F(\cdot)$ such that $R = E(\overline{\imath}\{P_i\}\text{–}n[\,F(j)\,])$—informally, $j \prec i$ when the scion j is within the stub i.*

   The relation $\prec$ is cycle-free, i.e. we never have $i \prec^{*} i$.
4. *Every step $Q \rightarrow_{12C} Q'$ has one of the following effects on each stub of R. For every stub of the first form, with $R = E(i)(\overline{\imath}\{P_i\}\text{–}n[\,R_i\,])$:*

   (a) *$Q' \equiv \nu\widetilde{x} \setminus i.E(\,n[\,P_i \mid R_i\,])(\mathbf{0})$ (Completion step).*

   (b) *$Q' \equiv \nu\widetilde{y}.E'(i)(\overline{\imath}\{P_i\}\text{–}n[\,R_i\,])$ for some tuple $\widetilde{y}$ and extended evaluation context $E'(\cdot)(\cdot)$ whose holes are not restricted and such that (External step):*

      i. *$\widetilde{y} = \widetilde{x}$ and $E(\cdot)(\cdot) \rightarrow_{12C} E'(\cdot)(\cdot)$, or*

      ii. *$\widetilde{y} = \widetilde{x}, j$ and $E(\cdot)(\cdot) \rightarrow_{1} \nu j.E'(\cdot)(\cdot)$ for some fresh name j, or*

      iii. *$\widetilde{y} = \widetilde{x} \setminus j$ and $\nu j.E(\cdot)(\cdot) \rightarrow_{2} E'(\cdot)(\cdot)$ for some $j \in \widetilde{x} \setminus i$.*

   (c) *$Q' \equiv \nu\widetilde{y}.E(i)(\overline{\imath}\{P_i\}\text{–}n[\,R_i'\,])$ for some tuple $\widetilde{y}$ and extended process R_i' such that (Internal step):*

      i. *$\widetilde{y} = \widetilde{x}$ and $R_i \rightarrow_{12C} R_i'$, or*

      ii. *$\widetilde{y} = \widetilde{x} \setminus j$ and $\nu j.R_i \rightarrow_{2} R_i'$ for some $j \in \widetilde{x} \setminus i$.*

   (d) *$Q' \equiv \nu\widetilde{x}, j.E(i)(j \mid \overline{\imath}\{P_i\}\text{–}n[\,R_i'\,])$ for some extended process R_i' and fresh name j such that $\overline{\imath}\{P_i\}\text{–}n[\,R_i\,] \rightarrow_{1} \nu j.j \mid \overline{\imath}\{P_i\}\text{–}n[\,R_i'\,]$ (Outgoing* Out 1 *step).*

   (e) *$Q' \equiv \nu\widetilde{x} \setminus j.F(i)(\overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,])(\,m[\,P_j \mid R_j\,])$ and beforehand we had $R = F(i)(\overline{\imath}\{P_i\}\text{–}n[\,G(\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,])\,])(j)$ (Outgoing* Move 2 *step).*

   (f) *$Q' \equiv \nu\widetilde{x} \setminus j.F(i)(\overline{\imath}\{P_i\}\text{–}n[\,G(\,m[\,P_j \mid R_j\,])\,])(\mathbf{0})$ and beforehand we had $R = F(i)(\overline{\imath}\{P_i\}\text{–}n[\,G(j)\,])(\overline{\jmath}\{P_j\}\text{–}m[R_j])$ (Entering* Move 2 *step).*

   For every stub of the second form, with $R = E(\,\mathrm{o}\{P_o\}\text{–}n[\,R_o\,])$:

   (a) *$Q' \equiv \nu\widetilde{x}.E(P_o \mid R_o)$ (Completion step).*

   (b) *$Q' \equiv \nu\widetilde{y}.E'(\,\mathrm{o}\{P_o\}\text{–}n[\,R_o\,])$ for some tuple $\widetilde{y}$ and evaluation context $E'(\cdot)(\cdot)$, with the same subcases as above (External step).*

   (c) *$Q' \equiv \nu\widetilde{y}.E(\,\mathrm{o}\{P_o\}\text{–}n[\,R_o'\,])$ for some tuple $\widetilde{y}$ and extended process R_o', with the same subcases as above (Internal step).*

   (d) *$Q' \equiv \nu\widetilde{x}, j.E(j \mid \mathrm{o}\{P_o\}\text{–}n[\,R_o'\,])$ for some extended process R_o' and fresh name j such that $\mathrm{o}\{P_o\}\text{–}n[\,R_o\,] \rightarrow_{1} \nu j.j \mid \mathrm{o}\{P_o\}\text{–}n[\,R_o'\,]$ (Outgoing* Out 1 *step).*

   (e) *$Q' \equiv \nu\widetilde{x} \setminus j.F(\,\mathrm{o}\{P_o\}\text{–}n[\,G(\mathbf{0})\,])(\,m[\,P_j \mid R_j\,])$ and beforehand we had $R = F(\mathrm{o}\{P_o\}\text{–}n[G(\overline{\jmath}\{P_j\}\text{–}m[R_j])])(j)$ (Outgoing* Move 2 *step).*

(f) $Q' \equiv \nu\widetilde{x} \setminus j.F(\,\mathtt{o}\{P_o\}\text{–}n[\,G(\,m[\,P_j \mid R_j\,]\,)\,]\,)(\mathbf{0})$ *and beforehand we had* $R = F(\,\mathtt{o}\{P_o\}\text{–}n[\,G(j)\,]\,)(\overline{\jmath}\{P_j\}\text{–}m[R_j])$ *(Entering* MOVE 2 *step).*

Proof: We establish the invariant by a global induction on the length of the derivation $P \rightarrow_{12C}^{*} Q$ for the first three properties of the lemma, and by checking the fourth property as part of the induction step.

Properties *1–3* clearly hold for $P = Q$: the form $\nu\widetilde{x}.R$ is obtained by lifting all restrictions at top-level after α-conversion; by definition, there is no stub in R.

Assume that these properties hold for a derivation $P \rightarrow_{12C}^{n} Q$, and let $Q \rightarrow_{12C} Q'$. We establish properties *1–3* at rank $n+1$ and property *4* at rank n by a case analysis on the rule being used for the additional $\rightarrow_{12C}$ step. In each case, we describe an extended process R' obtained from R such that $Q' \equiv \nu\widetilde{y}.R'$, we classify this reduction with regards to every stub of R, and we study its impact on the relation $\prec$.

IN 1**:** A stub is created, of the first form. Let i be a fresh name. For some evaluation context $E(\,\cdot\,)$ whose two holes are in the same parallel composition, we have

$$\begin{aligned} R &= E(\,m[\,Q_i\,]\,)(\,n[\,\mathtt{in}\; m.P_i \mid R_i\,]\,) \\ Q' &\equiv \nu\widetilde{x}.\nu i.E(\,m[\,i \mid Q_i\,]\,)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) \end{aligned}$$

For every stub that encloses the holes of $E(\,\cdot\,)(\,\cdot\,)$, this is an internal step *(4c,i)* followed by a scope extrusion for i. For every other stub of R, this is an external step *(4b,ii)*.

As regards $\prec$, let J be the set of names attached to enclosing stubs of the first form, and let K be the set of scions appearing in R_i. The relation $\prec$ is left unchanged on previously existing scions. The relation $\prec$ is extended with $k \prec i$ for $k \in K$ and $i \prec j$ for $j \in J$. This extension cannot create any new cycle: if a cycle appears in R', it contains $k \prec i \prec j$ for some $k, j \in K \times J$ and, since $k \prec j$ in R, we had a shorter cycle in R, in contradiction with the induction hypothesis *3*.

OUT 1**:** Similarly, a stub of the first form is created. Let i be a fresh name. Up to the rearrangement of the parallel composition within the parent ambient, for some evaluation context $E(\,\cdot\,)$ and extended processes R_i and R_i', we have

$$\begin{aligned} R &= E(\,X^{=}m[\,n[\,\mathtt{out}\; m.P_i \mid R_i\,] \mid R_i'\,]\,) \\ Q' &\equiv \nu\widetilde{x}.\nu i.E(i \mid X^{=}m[\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,] \mid R_i'\,]\,) \end{aligned}$$

Depending on $X^{=}$, the parent ambient may be either a regular ambient, or a stub of the first form with marker l, or a stub of the second form.

If the parent ambient is a stub, this is an outgoing step *(4d)*. For every other enclosing stub, this is an internal step *(4c,i)* followed by scope extrusion. For every other stub, this is an external step *(4b,ii)*.

As regards $\prec$, the situation is similar to the previous case. With the same definitions for J and K, new relations $k \prec i$ and $i \prec j$ appear for $k \in K$ and $j \in J$, but they do not introduce cycles. Besides, if the parent is a stub of the first form with marker l, all relations $k \prec l$ for $k \in K$ disappear.

MOVE 2**:** The stub being consumed is a subprocess of R of the first form. Let i be the scion also consumed in this step; by hypothesis *3.*, this scion is outside of the consumed stub. Explicitly, for some extended evaluation contexts

$A_0(\cdot)$, $A_1(\cdot)$, and $A_2(\cdot)$ whose hole is not in parallel composition with another process, and some extended processes R_0, R_1, R_2, and R_i, we have

$$\begin{aligned} R &= A_0\big(R_0 \mid A_1(R_1 \mid i) \mid A_2(R_2 \mid \overline{\imath}\{P_i\}\text{–}n[\,R_i\,])\big) \\ Q' &\equiv \nu\widetilde{x} \setminus i.A_0\big(R_0 \mid A_1(R_1 \mid n[P_i \mid R_i]) \mid A_2(R_2)\big) \end{aligned}$$

For the consumed stub, this is a completion step of the first form *(4a)*. For every enclosing stub of $A_0(\cdot)$, this is an internal step *(4c,ii)*. For every enclosing stub of $A_1(\cdot)$, this is an ingoing MOVE 2 step *(4f)*. For every enclosing stub of $A_2(\cdot)$, this is an outgoing MOVE 2 step *(4e)*. For all other stub, this is an external step *(4b,ii)*.

Let K be the set of scions appearing in R_i, J_1 and J_2 be the set of names of enclosing ambients of the first form in $A_1(\cdot)$ and $A_2(\cdot)$, respectively. The relation $\prec$ evolves as follows: all pairs containing the reacting scion i are deleted. Pairs $k \prec j$ are also deleted for all $k, j \in K \times J_2$. Conversely, pairs $k \prec j$ are created for all $k, j \in K \times J_1$; since we already had $k \prec i \prec j$, this cannot create cycles.

OPEN 1**:** A stub of the second form is created. For every other stub, this is either an external or an internal step. The relation $\prec$ is not affected.

OPEN 2**:** A stub of the second form is consumed. For this stub, this is a completion step. For all others, this is either an external or an internal step. The relation $\prec$ is not affected.

REPL, RECV**:** These steps may produce new processes, but they do not affect the extensions. For every existing stub, it is either an internal step or an external step.

In any case, we obtain a new process R that meets all the conditions of the lemma after lifting all restrictions that appeared in evaluation context to the top level (up to α-conversion). □

For every stub described in Lemma B.2, a completion step *(4a.)* is enabled, hence this stub can always be discarded in a single 2-step. By iterating these steps, we prove the corollary:

Lemma B.3 (Completion) *For all processes P and extended processes Q such that $P \to_{12C}^* Q$, there is a process P' such that $Q \to_2^* P'$.*

We can now compare successive $\to_1$ and $\to_2$ steps that operate on the same extension to the original $\longrightarrow$ step of the ambient calculus:

Lemma B.4 (Refinement) *For all regular processes P and P', we have $P \to P'$ if and only if either $P \to_1 \to_2 P'$ or $P \to_C P'$.*

Proof: This is a direct consequence of lemma B.2 for series of two reductions $P \to_1 \to_2 P'$. The first step is enabled if and only if the corresponding initial step of the ambient calculus is enabled, and since P' contains no extension, the second step must be the matching completion step. We obtain the commutative reduction diagrams of section 4. □

Relying on Lemma B.2, we use the markers i, j, k introduced by 1-steps to keep track of stubs and scions, and to match these 1-steps with the 2-steps eliminating these stubs and scions. By extension, we also use fresh markers for matching OPEN 1 and OPEN 2 steps. (More formally, we may refine the extended semantics to produce and consume a $\nu i.$ binder and use extensions $o(i).P$ instead of $o.P$.)

We will use the following commutation properties of the extended reductions.

Lemma B.5 *Let P be a process and Q be an extended process such that $P \to_{12C}^* Q$. We have the following commutation properties:*

- *$Q \to_1^i \to_1^j Q'$ implies $Q \to_1^j \to_1^i Q'$ when the $\to_1^i$-step is not a* IN 1*-step towards an ambient consumed by the following $\to_1^j$ step on the left-hand-side;*
- *$Q \to_1^i \to_2^j Q'$ implies $Q \to_2^j \to_1^i Q'$ when $i \neq j$ and the $\to_1^i$-step is not a* OUT 1*-step from a stub consumed by the following $\to_2^j$-step on the left-hand-side;*
- *$Q \to_2^i \to_1^j Q'$ implies $Q \to_1^j \to_2^i Q'$ when the $\to_2^i$ step does not create a subterm used by the the following $\to_1^j$ step on the left-hand-side;*
- *$Q \to_2 \to_C Q'$ implies $Q \to_C \to_2 Q'$ when the $\to_C$-step does not use a subterm created by the $\to_2$-step on the left-hand-side;*
- *$Q \to_C \to_2 Q'$ implies $Q \to_2 \to_C Q'$;*
  $Q \to_1 \to_C Q'$ implies $Q \to_C \to_1 Q'$;
  $Q \to_2^i \to_2^j Q'$ implies $Q \to_2^j \to_2^i Q'$.

In particular, we always have $Q \to_1^i \to_1^j Q'$ implies $Q \to_1^j \to_1^i Q'$ when the second steps is internal with regards to i, and $Q \to_1^i \to_2^j Q'$ implies $Q \to_2^j \to_1^i Q'$ when $i \neq j$ and the first step is not an outgoing step with regards to j.

Proof: For each commutation property, we apply lemma B.2 to $P \to_{12C}^* Q$, unfolding twice clause *(4.)* for the two reduction steps being considered, and we check that these two steps also apply in the converse order.

For $\to_1^i \to_1^j$, in the case $\to_1^i$ uses rule IN 1, this first step can we written

$$\nu\tilde{x}.F(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) \quad \to_1^i \quad \nu\tilde{x}, i.F(m[Q_i \mid i])(\overline{\imath}\{P_i\}{-}n[\,R_i\,])$$

where $F(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,)$ is the evaluation context $E(\,\cdot\,)(\,\cdot\,)$ of lemma B.2*(2)*. The second step is either an external step *(4b,i)* or *(4b,iii)*, or an internal step *(4c,i)*, or an outgoing step *(4d)*. We detail only the case *(4b,iii)* when it creates a stub of the first form, of name j, in a step $E(\,\cdot\,)(\,\cdot\,) \to_1^j \nu j.E'(\,\cdot\,)(\,\cdot\,)$. (Other cases are much simpler.) We further distinguish three subcases according to the position of the ambient being extended by $\to_1^j$: an ambient in $F(\,\cdot\,)(\,\cdot\,)$, or a sub-ambient in Q_i, or the target ambient $m[\,Q_i\,]$.

In the first subcase, the step $E(\,\cdot\,)(\,\cdot\,) \to_1 \nu j.E'(\,\cdot\,)(\,\cdot\,)$ can also be written more explicitly as $F(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,) \to_1 \nu j.F'(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,)$, and we also have

$$\begin{array}{lll} \nu\tilde{x}.F(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) & \to_1^j & \nu\tilde{x}.\nu j F'(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) \\ & \to_1^i & \nu\tilde{x}.\nu j, i.F'(m[Q_i \mid i])(\overline{\imath}\{P_i\}{-}n[\,R_i\,]) \end{array}$$

In the second subcase, the step $E(\,\cdot\,)(\,\cdot\,) \to_1 \nu j.E'(\,\cdot\,)(\,\cdot\,)$ can also be written $F(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,) \to_1 \nu j.F(Q_i' \mid m[\,Q_i'' \mid (\,\cdot\,)\,]\,)(\,\cdot\,)$ in which Q_i' is either $\mathbf{0}$ for an

internal step or j for an outgoing step, and we also have

$$\begin{array}{lll} \nu\tilde{x}.F(m[Q_i])(n[\texttt{in}\ m.P_i \mid R_i]) & \rightarrow_1^j & \nu\tilde{x}.\nu j.F'(Q_i' \mid m[Q_i''])(n[\texttt{in}\ m.P_i \mid R_i]) \\ & \rightarrow_1^i & \nu\tilde{x}.\nu j, i.F'(Q_i' \mid m[Q_i'' \mid i])(\overline{\imath}\{P_i\}\text{–}n[R_i]) \end{array}$$

In the third subcase, however, the target ambient $m[Q_i]$ can be used in rule In 1 only before it gets extended, so that two steps do not commute. □

Next, we establish a key lemma stating that the extended semantics does not introduce more reductions between regular ambient processes than those of the standard semantics.

Lemma B.6 (Correctness) *For all processes P and P', if $P \rightarrow_{12C}^* P'$, then $P \rightarrow^* P'$.*

Proof: We establish the property above by induction on the length n of the series $P \rightarrow_{12C}^n P'$. For a given pair (P, P'), every rewriting step of the proof will preserve the length of the series from P to P'; accordingly, we leave the lengths of intermediate series implicit.

The property trivially holds for an empty series. Otherwise, the first reduction step is either $\rightarrow_C$ or $\rightarrow_1$ (not $\rightarrow_2$, because by hypothesis there is no stub in P).

If $P \rightarrow_C Q \rightarrow_{12C}^* P'$, then Q is a regular process and we apply the induction hypothesis on $Q \rightarrow_{12C}^* P'$.

If $P \rightarrow_1 Q \rightarrow_{12C}^* P'$, then the series of reductions contains a 2-step $\rightarrow_2$ that consumes this stub. Writing $\rightarrow_1^i$ and $\rightarrow_2^i$ to emphasize these matching steps, we can decompose the series into

$$P \rightarrow_1^i Q \rightarrow_{12C}^* Q' \rightarrow_2^i Q'' \rightarrow_{12C}^* P'$$

The stub created by $\rightarrow_1^i$ is preserved by every step between Q and Q'. We rely on lemma B.2 to partition reduction steps between Q and Q' into external steps, internal steps, outgoing steps, and entering steps according to their relation with this first stub.

We are interested in the steps that precede the first outgoing step, if any. (These steps do not contain any entering step, which are enabled only after at least one outgoing step carrying a stub.) We distinguish the following cases:

1. There is no outgoing step. By Lemma B.5, every step between Q and Q' thus commutes with $\rightarrow_2^i$, and we have $P \rightarrow_1^i Q \rightarrow_2^i R\mathit{rext}^* P'$ for some extended process R. By Lemma B.4, we obtain that R is a regular process such that $P \rightarrow R$, and we conclude by induction hypothesis on $R\mathit{rext}^* P'$.

2. There is at least one outgoing step. In this case, some of the steps between $\rightarrow_1^i$ and $\rightarrow_2^i$ may not commute with $\rightarrow_2^i$. We select another initial reduction as follows: assume $P \rightarrow_1^i Q\ (\mathit{rext}^{ext})^* \rightarrow^j R\mathit{rext}^* P'$ in which $(\mathit{rext}^{ext})^*$ is a series of external steps and $\rightarrow^j$ is the first internal or outgoing step. This step $\rightarrow^j$ is either a $\rightarrow_C$ step or a $\rightarrow_1$ step. By Lemma B.5, this step commutes with every preceding external step, and also with the initial $\rightarrow_1^i$ step. We obtain $P \rightarrow^j \rightarrow_1^i (\mathit{rext}^{ext})^* R\mathit{rext}^* P'$, or simply $P \rightarrow^j \mathit{rext}^* P'$.

   (a) In the case $\rightarrow^j$ is a $\rightarrow_C$ step, we conclude as above.

(b) Otherwise, we iterate the case analysis with $\rightarrow_1^j$ as initial reduction instead of $\rightarrow_1^i$. Since the nesting of ambients is preserved by all $\rightarrow_1$ steps, since every ambient process has a finite depth, and since case *2.* above always selects a new $\rightarrow_1$ step strictly deeper than the previous one, we eventually end up with an initial reduction that can be eliminated by case *1.* □

The proof of the main correctness theorem easily follows:

Proof of Theorem 5.3: We check all the conditions of Lemma B.1.

(1) is Lemma B.3

(2 $\Rightarrow$) is Lemma B.4 for 1- and 2-steps, and is obvious for RECV and REPL steps.

(2 $\Leftarrow$) is Lemma B.6

(3) follows from our definition of barbs: $\rightarrow_2$ steps delete only stubs that do not contribute to the strong barbs of any process. □

B.2 Correctness of the extended translation (theorem 5.4)

Despite our factorization of the proof, the correctness proof for the translation remains complicated. In every stage of the translation, there is a large variety of reduction steps to consider. Also, the translation does not commute with reduction and some additional relations are required. To reduce the number of cases to deal with, we restrict the shape of the terms we manipulate, and prove that they are invariant by reductions and include the translation of any extended ambient process. To simplify the proof, we use a decreasing diagram proof technique [17, 9], and prove simpler diagrams which are then used to tile the bisimulation diagram we want to establish.

After some definitions, we start by defining *translation states* which give the general shape of the join terms we manipulate, and we partition the reductions enabled on these translation states between steps in direct correspondence with source ambient steps and other auxiliary steps, which we name *bookkeeping steps*. Then, we establish simple commutation diagrams, or tiles, that deal with each simplification relation, and finally assemble these diagrams to obtain a weak bisimulation result for the translation.

Next, we define an auxiliary translation $[\![\cdot]\!]_e^\sharp$ from extended ambient processes to join calculus configurations. This translation extends the translation of figure 2 for extended ambient processes.

In the following, we use the auxiliary translation on configurations which are α-renamed such that bound names be pairwise distinct.

In order to avoid name clashes during structural rearrangements in the join calculusconfigurations, we perform preventive α-conversions on the names defined in each ambient manager. We write D^n, D_t^n, and R^n for the definitions $D \wedge D'$, D_t and processes R obtained by substituting x^n for every name x defined in $D \wedge D'$ and h^n for *here* . Finally, we abbreviate *reloc* and *opening* with r and o, respectively.

Definition B.7 (Auxiliary translation) *We translate extended ambients into join calculus configurations as follow:*

$$\begin{aligned}
[\![P \mid Q]\!]_e^\sharp &\stackrel{def}{=} [\![P]\!]_e^\sharp \mid [\![Q]\!]_e^\sharp \\
[\![\nu a.P]\!]_e^\sharp &\stackrel{def}{=} \texttt{def fresh } a \texttt{ in } [\![P]\!]_e^\sharp \\
[\![\nu i.P]\!]_e^\sharp &\stackrel{def}{=} [\![P]\!]_e^\sharp \\
[\![i]\!]_e^\sharp &\stackrel{def}{=} \mathbf{0} \\
[\![a[\,P\,]]\!]_e^\sharp &\stackrel{def}{=} H^n(s^n(a, i^n, e, \emptyset) \mid amb(i^n, a, e^n)) \\
[\![\overline{\imath}\{Q\}\text{–}a[\,P\,]]\!]_e^\sharp &\stackrel{def}{=} H^n(\texttt{def } \kappa() \triangleright [\![Q]\!]_{e^n} \texttt{ in } s^n(a, i^n, e, \emptyset) \mid r^n(e^p, \kappa)) \\
[\![\mathtt{o}\{Q\}\text{–}a[\,P\,]]\!]_e^\sharp &\stackrel{def}{=} H^n(s^n(a, i^n, e, \emptyset) \mid \texttt{def } \kappa() \triangleright [\![Q]\!]_{e^n} \texttt{ in } o^n(\kappa)) \\
[\![P]\!]_e^\sharp &\stackrel{def}{=} [\![P]\!]_e \textit{ for all other ambient constructs}
\end{aligned}$$

where n is the index of the translated ambient, where p is the index of the ambient in which the scion i appears, and where we use the context

$$H^n(\,\cdot\,) \stackrel{def}{=} \texttt{def } h^n[D^n \wedge \texttt{uid } i^n : [\![P]\!]_{e^n}^\sharp \mid \cdot] \texttt{ in } \mathbf{0}$$

The auxiliary translation makes it clear that i is used solely as a passive target for migration, as its translation is the empty process.

As in section 4, we also define a top-level variant for the auxiliary translation. With the same notations, after renaming D, D_t, and Q to D^0, D_t^0, and Q^0 respectively, we write $[\![\,\cdot\,]\!]^{t\sharp}$ for the translation that maps every extended ambient process P to the join configuration $[D^0 \wedge D_t^0 : Q^0 \mid p(t) \mid [\![P]\!]_{e^0}^\sharp]$, for some names a, e, p, t, and *yes* that do not occur in P.

We now restrict the class of join-terms that we consider. The next definition makes explicit the general shape of translated processes.

Definition B.8 (Translation state) *A join configuration P is a translation state, written $P \in \mathcal{T}$, when:*

1. *The structure of P can be decomposed as follows:*

$$\begin{aligned}
P &= [D^0 \wedge E^0 \wedge R \wedge D_t^0 : p(t) \mid s^0(a, i^0, e, \emptyset) \mid M^0] \wedge \bigwedge_{n \in N^+} \alpha^n h^n [D^n \wedge E^n : S^n \mid M^n] \\
E^n &= \textstyle\bigwedge_{\kappa \in K_a^n} \kappa() \triangleright [\![Q^\kappa]\!]_{e^n} \\
&\wedge \textstyle\bigwedge_{\kappa \in K_b^n} \kappa(a) \triangleright [\![Q^\kappa]\!]_{e^n} \\
&\wedge \textstyle\bigwedge_{\kappa \in K_c^n} \kappa() \triangleright [\![Q^\kappa]\!]_{e^n} \mid \kappa() \\
R &= \textstyle\bigwedge_{a \in L} \texttt{fresh } a \\
&\wedge \textstyle\bigwedge_{i \in I} \texttt{uid } i
\end{aligned}$$

   for some index set N with a distinguished label $0 \in N$, some strings of names $\alpha^n \in \{h^n \mid n \in N^+\}^*$ for every $n \in N^+$ ($N^+ \stackrel{def}{=} N \setminus \{0\}$), and some disjoint sets of names $\{a, i^0\}$, $fv(e)$, F, L, I, and $dv(D^n)$, K_a^n, K_b^n, K_c^n for each $n \in N$.

   In the following, we use the additional notations $K_{ab}^n = K_a^n \cup K_b^n$, $K_{ab} = \bigcup_{n \in N} K_{ab}^n$, $K_c = \bigcup_{n \in N} K_c^n$, and $K = K_{ab} \cup K_c$. α_0 and h^0 are undefined, but for uniformity we extend our notations with $\alpha^0 h^0 \stackrel{def}{=} \epsilon$.

2. *The process* M^0 *is a parallel composition of messages sent on names in* $\{amb^0, open^0, rcv^0, snd^0, sub^0_{in}, sub^0_{out}\} \cup K^0$. *For every message* $snd^0(a)$, *we have* $a \in F \cup L$.

3. *For every* $n \in N^+$, *the process* M^n *is a parallel composition of messages sent on names in* $\{amb^n, in^n, out^n, open^n, rcv^n, snd^n, sub^n_{in}, sub^n_{out}, o^n, r^n\} \cup K^n$. *For every message* $snd^n(a)$, *we have* $a \in F \cup L$.

4. *For every* $n \in N^+$, *the set* l^n *contains entries of the form* In b κ *and* Out b κ *with* $b \in F \cup L$ *and* $\kappa \in K_a$.

5. *We say that* $sub^n_{in}(i, c, \kappa)$ *or* $sub^n_{out}(i, c, \kappa)$ *is a* stale message *when there is no message* $s^m(\_, i, \_, \_)$ *in any* S^m *with* $m \in N$.

   Each name $\kappa \in K_a$ *occurs in its definition in* E^n, *in stale messages, and as described in at most one of the following cases, for some* $m \in N^+$ *and* $p \in N$:

   (a) $in^m(b, \kappa)$ *in* M^m *(or* $out^m(b, \kappa)$ *in* M^m*), with* $b \in F \cup L$.

   (b) $sub^p_{in}(i^m, b, \kappa)$ *in* M^p *and* In b κ *within* l^m *(or* $sub^p_{out}(i^m, b, \kappa)$ *in* M^p *and* Out b κ *within* l^m*) with* $b \in F \cup L$.

   (c) *within* $S^m = \mathsf{go}(e^p.here); I_{b,e^m,e^p} \mid \kappa() \mid Flush(l^m, in^m, out^m, \kappa)$, *with at most one* In b κ *within* l^m *(or one* Out b κ *within* l^m*).*

   (d) $r^n(e^p, \kappa)$ *in* M^n *and at most one* In b κ *within* l^n *(or one* Out b κ *within* l^n*).*

   (e) $open^p(b, \kappa)$ *in* M^p.

   (f) $o^m(\kappa)$ *in* M^m.

   (g) $\kappa()$ *in* M^p.

6. *Each name* $\kappa \in K_b$ *occurs in its definition in* E^n *and as a single message* $\kappa(a)$ *or* $rcv^p(\kappa)$ *for some* $p \in N$ *and* $a \in F \cup L$.

7. *Each name* $\kappa \in K_c$ *occurs in its definition in* E^n, *and as a single message* $\kappa()$ *in* M^n.

8. *For every* $n \in N^+$, *one of the following holds:*

   (a) *either* $S^n = s^n(b, i^n, e^m, l^n)$ *for* $b \in F \cup L$; *we say that* n *is alive. Then,* $\alpha^n = \alpha^m h^m$ *and* P *contains a single message of the following forms:*

      i. $amb^p(i^n, \_, e^n)$. *Then,* $\alpha^p h^p \beta = \alpha^m h^m$ *for some string* β *of location names* h^s *with* s *opened.*

      ii. $r^n(e^p, \kappa)$ *with* $\kappa \in K^n_a$

      iii. $o^n(\kappa)$ *with* $\kappa \in K^n_a$.

   (b) *or there is none of those messages in* P, *and*

      i. *either* $S^n = f^n(e^m)$; *we say that* n *is opened. Then,* $\alpha^n = \alpha^m h^m$.

      ii. *or* $S^n = \mathsf{go}(e^p.here); I_{b,e^n,e^p} \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa)$. *We say that* n *is in a transient state.*

In the following, we identify translation states that only differ by associativity and commutativity of $\mid$ and $\wedge$, and by the presence of $\mathbf{0}$ processes.

Definition B.9 (Strong Barbed Bisimulations) *A relation $\mathcal{R}$ between two process calculi is a strong simulation when, for all $P\,\mathcal{R}\,Q$, if $P \to P'$, then there exists Q' such that $Q \to Q'$ and $P'\,\mathcal{R}\,Q'$.*

For a given process calculus equipped with barbs, barbed bisimilarity $\sim$ *is the largest symmetric strong simulation that respects strong barbs ($P \sim Q$ and $P \downarrow_b$ implies $Q \downarrow_b$).*

Definition B.10 (Deterministic steps) *In the join calculus, we say that a reduction step $P \to P'$ is deterministic when it uses one of the rules* Comm *and* Flush.

For all $P, Q \in \mathcal{J}$, we write $P \equiv_d Q$ when Q is obtained from P by structural equivalence and by changing the location of top-level definitions `fresh` *a and* `uid` *i. We write $\Rrightarrow_d$ for $\equiv_d \cap \mathcal{J} \times \mathcal{T}$.*

We write $\to_d$ for a deterministic step, and $\twoheadrightarrow_d$ for the relation:

$$\twoheadrightarrow_d \stackrel{def}{=} \{(P,Q) \mid \exists Q'.P \to_d^* Q' \wedge Q' \not\to_d \wedge\, Q' \Rrightarrow_d Q\}$$

In the following, $\twoheadrightarrow_d$ is called deterministic normalization.

Deterministic steps strongly normalize, by first reducing all Flush constructs, then by doing all Comm steps. Deterministic steps also commute with every join step.

Lemma B.11 *$\equiv_d$ is a strong barbed bisimulation for $\to_d$ and $\to$.*

Lemma B.12 *If $P \in \mathcal{T}$, then $P \not\to_d$. P.*

Proof: We proceed by case on the rule being used (definition B.10).

Comm By conditions 1, 2, 3, and 8 all messages reside in their defining location, thus no Comm step is enabled.

Flush By the same conditions, no Flush process is running. □

In the following, we write $\mapsto$ for join calculus steps which do not use the structural rules Tree and Scope.

We now partition the steps that are enabled in a translation state and give an intuitive idea of the corresponding extended ambient semantics. These steps are enabled without using the structural rules Tree nor Scope. The analysis is by case on the reduction rule being used, with subcases for Join-steps.

Comm, Flush By lemma B.12, no Comm nor Flush step is enabled.

Go We call these steps *migration steps*, written $\mapsto_{go}$.

Join We consider the reduction rule being used, following the definitions of D^n and E^n in definition B.8 and figure 2:

- D_0^n We call these steps *delegation steps*, written $\mapsto_0$. Such steps correspond to initial steps of the algorithm of section 2.1, which are not detailed in the extended ambient calculus.
- D_1^n We call these steps *initial steps* and partition them further in $\mapsto_{1,In}$, $\mapsto_{1,Out}$, and $\mapsto_{1,Open}$ steps depending on the rule being used, in that order. These steps correspond to ambient steps In 1, Out 1, and Open 1, respectively.

D_2^n We call these steps *completion steps*and partition them further in $\mapsto_{2,Open}$ and $\mapsto_{2,Move}$ steps depending on the rule being used, in that order. These steps correspond to ambient steps OPEN 2 and MOVE 2 respectively.

D_C^n We call these steps *communication steps*, written $\mapsto_C$. They correspond to ambient steps RECV.

D_F^n We call these steps *forwarding steps*, written $\mapsto_{Fwd}$. Such steps deal with the forwarding of messages in opened locations. They do not correspond to ambient steps.

E^n Let κ be the name defined in the rule being used.

- $\kappa \in K_{ab}^n$**:** these steps are called *continuation steps*, written $\mapsto_\kappa$. They do not correspond to ambient steps.
- $\kappa \in K_c^n$**:** these steps are called *replication steps*, written $\mapsto_{Repl}$. They correspond to REPL ambient steps.

D_t^0 By definition B.8, there is no message on t, hence this rule cannot be used.

We obtain two sets of steps, the first intuitively corresponding to steps in relation with the ambient steps, the second corresponding to other, bookkeeping steps.

$$X \stackrel{\text{def}}{=} \{\mapsto_{1,In}, \mapsto_{1,Out}, \mapsto_{1,Open}, \mapsto_{2,Move}, \mapsto_{2,Open}, \mapsto_{Rcv}, \mapsto_{Repl}\}$$
$$B \stackrel{\text{def}}{=} \{\mapsto_0, \mapsto_{Fwd}, \mapsto_\kappa, \mapsto_{go}\}$$

Next, we define a notion of *extension* between translation states. This definition uses set inclusion on processes and definitions as a shorthand for set inclusion on the sets of parallel processes and composed reaction rules respectively.

Definition B.13 *Let $P, Q \in \mathcal{T}$. We say that Q* extends *P when there is some $m \in N_P$ such that:*

1. *$N_P \subseteq N_Q$, $L_P \subseteq L_Q$, and $I_P \subseteq I_Q$;*
2. *for all $n \in N_P^+$, $S_P^n = S_Q^n$ and $\alpha_P^n = \alpha_Q^n$;*
3. *for all $n \in N_P$ such that $n \neq m$, $M_P^n = M_Q^n$, $E_P^n = E_Q^n$;*
4. *$M_P^m \mid M_P'^m \equiv M_Q^m$;*
5. *$K_P^m \subseteq K_Q^m$ with identical rules on K_P^m;*
6. *for all $q \in N_Q \setminus N_P$, $\alpha^m h^m$ is a prefix of $\alpha^q h^q$.*

We now prove that extended translations $[\![Q]\!]^{t\sharp}$ reduce to translation states, and that translations states are closed by $\mapsto \twoheadrightarrow_d$.

Lemma B.14 (Ambient translation) *Let $P \in \mathcal{T}$, let $m \in N$ where N is the index set of P, and let $C(\cdot)$ be the context obtained from P by substituting $\cdot \mid M^m$ for M^m.*

1. *For every $Q \in \mathcal{E}$, we have $C([\![Q]\!]^\sharp_{e^m}) \twoheadrightarrow_d P'$ for some $P' \in \mathcal{T}$ that extends P.*
2. *For every $Q \in \mathcal{A}$, we have $C([\![Q]\!]_{e^m}) \twoheadrightarrow_d P'$ for some $P' \in \mathcal{T}$ that extends P.*

Proof: We first show that statement 2 can be deduced from statement 1.

As $Q \in \mathcal{A}$, we have by definition B.7 $[\![Q]\!]^\sharp_e \equiv [\![Q]\!]_e$.

We now prove statement 1 by structural induction on Q, for any $C(\,\cdot\,)$ meeting the hypotheses of the lemma.

$Q = \mathbf{0}$**:** Since $C(\mathbf{0}) \equiv P$ and since extension is reflexive, the statement holds by structural equivalence.

$\nu i.Q$**:** Since $[\![\nu i.Q]\!]^\sharp_e \stackrel{\text{def}}{=} [\![Q]\!]^\sharp_e$, the statement holds by induction hypothesis.

$\nu a.Q$**:** By definition and structural equivalence, we assume that a does not occur in $C(\,\cdot\,)$, and we have

$$C([\![\nu a.Q]\!]^\sharp_{e^m}) \stackrel{\text{def}}{=} C(\texttt{def fresh}\ a\ \texttt{in}\ [\![Q]\!]^\sharp_{e^m}) \equiv C'([\![Q]\!]^\sharp_{e^m})$$

where $C'(\,\cdot\,)$ is $C(\,\cdot\,)$ with an additional `fresh` a definition in h^m. Unless $m = 0$, $C'([\![Q]\!]^\sharp_{e^m})$ is not a translation state and we apply lemma B.11 to move `fresh` a to R in h^0:

$$C'([\![Q]\!]^\sharp_{e^m}) \equiv_d C''([\![Q]\!]^\sharp_{e_m})$$

where $C''(\mathbf{0})$ is a translation state that extends P with `fresh` a in R. By induction hypothesis, we have $C''([\![Q]\!]^\sharp_{e^m}) \twoheadrightarrow_d P'$ for some $P' \in \mathcal{T}$ that extends $C''(\mathbf{0})$, and also extends P. By lemma B.11 ($\equiv_d$ is a strong bisimulation for $\rightarrow_d$), we have the following diagram:

$$\begin{array}{ccccc} C([\![\nu a.Q]\!]^\sharp_{e^m}) & \xrightarrow{\equiv_d} & C''([\![Q]\!]^\sharp_{e_m}) & & \\ \Big\downarrow{\scriptstyle n,d} & & \Big\downarrow{\scriptstyle n,d} & & \\ R & \xrightarrow{\equiv_d} & P'' & \xrightarrow{\equiv_d} & P' \end{array}$$

where n represents the number of deterministic steps, which is the same on each side by bisimimulation. Since we have $P'' \not\rightarrow_d$, we necessarily have $R \not\rightarrow_d$. Thus $C([\![\nu a.Q]\!]^\sharp_{e^m}) \twoheadrightarrow_d P'$, $P' \in \mathcal{T}$ and P' extends P.

$Q = Q' \mid Q''$**:** by induction hypothesis, $C([\![Q']\!]^\sharp_{e^m}) \twoheadrightarrow_d P'_1$ for some $P'_1 \in \mathcal{T}$. Let $C'(\cdot)$ be the context obtained from P'_1 for the same $m \in N$. Such a context exist because P'_1 extends P. Then, by applying the same series of reductions, we have:

$$\begin{array}{rcl} C([\![Q' \mid Q'']\!]^\sharp_{e^m}) & \stackrel{\text{def}}{=} & C([\![Q']\!]^\sharp_{e^m} \mid [\![Q'']\!]^\sharp_{e^m}) \\ & \twoheadrightarrow_d & C'([\![Q'']\!]^\sharp_{e^m}) \end{array}$$

and we conclude by induction using context $C'(\,\cdot\,)$ and the same index m. The resulting configuration is an extension of P since it was extended twice according to the same index m.

$Q = !Q'$**:** By definition of the translation, we have:

$$C([\![!Q']\!]^\sharp_{e^m}) \stackrel{\text{def}}{=} C(\texttt{def}\ \kappa \triangleright [\![Q']\!]_{e^m} \mid \kappa()\ \texttt{in}\ \kappa()) \equiv C'(\kappa)$$

where $C'(\,\cdot\,)$ is $C(\,\cdot\,)$ with the additional definition for the name κ. Since conditions 7 and 3 are satisfied, $C'(\kappa)$ is a translation state which extends P.

$Q = (x).Q'$, $\texttt{open}\ a.Q'$, $\texttt{out}\ a.Q'$, $\texttt{in}\ a.Q'$: The same argument as above holds. The only conditions to check are conditions 3, because a message is added, and 5, which hold trivially. In all cases, the translated configuration is an extension of P.

$Q = \langle a \rangle$: We have:

$$C([\![\langle a \rangle]\!]^\sharp_{e^m}) \stackrel{\text{def}}{=} C(snd^m(a))$$

In the case where $a \notin L$, F is extended to contain a. The translated configuration is a translation state.

$Q = a[\,Q'\,]$: we have

$$\begin{array}{rcll} C([\![\, a[\,Q'\,]\,]\!]^\sharp_{e^m}) & \equiv_d & C'(0) \wedge \alpha^m h^m h^n[D^n : s^n(a, i^n, e^m, \emptyset) \mid [\![Q']\!]^\sharp_{e^n} \mid amb^m(i^n, a, e^n)] & (7) \\ & \rightarrow_d & C'(amb^m(i^n, a, e^n)) \wedge \alpha^m h^m h^n[D^n : s^n(a, i^n, e^m, \emptyset) \mid [\![Q']\!]^\sharp_{e^n}] & (8) \\ & = & R'' & (9) \end{array}$$

where n is a new index which is added to the set of indexes N, $C'(\mathbf{0})$ is $C(\mathbf{0})$ with an additional definition $\texttt{uid}\ i^n$.

The strong bisimulation (7) uses lemma B.11 to move the $\texttt{uid}\ i^n$ definition from h^n to h^0. The $\rightarrow_d$ reduction (8) is the COMM reduction transporting the amb^m message to $\alpha^m h^m$.

We write R' for R'' where $[\![Q']\!]^\sharp_{e^n}$ is replaced by $\mathbf{0}$. The configuration R' satisfies conditions 1, 2 and 3 of definition B.8, as it is P with the following differences:

- $N' = N \uplus \{n\}$;
- if $a \notin L$ then $F' = F \uplus \{a\}$;
- $\alpha'^n = \alpha^m h^m$;
- $R' = R \wedge \texttt{uid}\ i^n$;
- $I' = I \uplus \{i^n\}$;
- $S'^n = s^n(a, i^n, e^m, \emptyset)$;
- $M'^n = \mathbf{0}$;
- $M'^m = M^m \mid amb^m(i^n, a, e^n)$
- $l'^n = \emptyset$.

All conditions are immediately satisfied. For instance, condition 8(a)i is met as $a \in F' \cup L'$, $\alpha'^n = \alpha'^m h^m$, there is a message $amb^m(i^n, \_, e^n)$ and no other such message since e^n is composed of names just introduced, there is no message on r^n nor o^n.

By induction, using ambient Q' and the context obtained from R' with index n, we have $R'' \twoheadrightarrow_d P'$ with $P' \in \mathcal{T}$ and P' extends R'. P' also extends P because:

1. $N_P \subseteq N_{R'} \subseteq N_{P'}$, $L_P = L_{R'} \subseteq L_{P'}$, and $I_P \subseteq I_{R'} \subseteq I_{P'}$;
2. for all $p \in N_P^+$, $S_P^p = S_{R'}^p = S_{P'}^p$ and $\alpha_P^n = \alpha_{R'}^n = \alpha_{P'}^n$;
3. for all $p \in N_P$ such that $p \neq m$, $M_P^p = M_{R'}^p = M_{P'}^p$ and $E_P^p = E_{R'}^p = E_{P'}^p$, since $n \notin N_P$;

4. we have $M_P^m \mid M_P^{'m} = M_{R'}^m = M_{P'}^m$;
5. we have $E_P^m = E_{R'}^m = E_{P'}^m$;
6. for all $q \in N_{P'} \setminus N_{R'}$, $\alpha^{'n} h^n$ is a prefix of $\alpha^{'q} h^q$, since P' extends R'. As $N_{R'} = N_P \uplus \{n\}$, we have $N_{P'} \setminus N_P = \{n\} \cup N_{P'} \setminus N_{R'}$. Since $\alpha^{'n} = \alpha^m h^m$, for all $q \in N_{P'} \setminus N_P$, $\alpha^m h^m$ is a prefix of $\alpha^{'q} h^q$.

We conclude as we did in case $\nu a.Q$ by deriving a reduction $C(\,a[\,Q'\,]\,) \twoheadrightarrow_d P'$.

$Q = \overline{\imath}\{Q'\}\text{–}a[Q'']$:

$$
\begin{aligned}
& C([\![\overline{\imath}\{Q'\}\text{–}a[Q'']]\!]^\sharp_{e^m} && (10)\\
\equiv_d\ & C'(0) \wedge \alpha^m h^m h^n[D^n \wedge \kappa \triangleright [\![Q']\!]_{e^n} : s^n(a, i^n, e^m, \emptyset) \mid [\![Q'']\!]^\sharp_{e^n} \mid r^n(e^p, \kappa)] && (11)\\
=\ & R'' && (12)
\end{aligned}
$$

where n is a new index which is added to the set of indexes N, p is the index of the translation of the ambient which contains the scion process i, and $C'(\mathbf{0})$ is $C(\mathbf{0})$ with an additional definition $\texttt{uid}\ i^n$.

The $\equiv_d$ step (11) uses lemma B.11 to move the $\texttt{uid}\ i^n$ definition from h^n to h^0.

We write R' for R'' with $\mathbf{0}$ substituted for $[\![Q'']\!]^\sharp_{e^n}$. We have $R' \in \mathcal{T}$ for the same reasons as above, with the only differences being the presence of a continuation definition ($K_a^{'n} = \{\kappa\}$) and a relocation message $r^n(e^p, \kappa)$ in M^n. Condition 5 is satisfied since the name κ is not present in P, and condition 8(a)ii is satisfied since there is no message of the form $amb^r(i^n, \_, e^n)$ nor any message on o^n.

By induction on Q'', with the context obtained from R' with index n, we obtain a configuration $P' \in \mathcal{T}$ which extends R', as well as P. We conclude as before.

$Q = \mathrm{o}\{Q'\}\text{–}a[Q'']$:

$$
\begin{aligned}
& C([\![\mathrm{o}\{Q'\}\text{–}a[Q'']]\!]^\sharp_{e^m}) && (13)\\
\equiv_d\ & C'(0) \wedge \alpha^m h^m h^n[D^n \wedge \kappa \triangleright [\![Q']\!]_{e^n} : s^n(a, i^n, e^m, \emptyset) \mid [\![Q'']\!]^\sharp_{e^n} \mid o^n(\kappa)] && (14)\\
=\ & R'' && (15)
\end{aligned}
$$

where n is a new index which is added to the set of indexes N, and $C'(\mathbf{0})$ is $C(\mathbf{0})$ with an additional definition $\texttt{uid}\ i^n$

The $\equiv_d$ step (14) uses lemma B.11 to move the $\texttt{uid}\ i^n$ definition from h^n to h^0.

We write R' for R'' with $\mathbf{0}$ substituted for $[\![Q'']\!]^\sharp_{e^n}$. As before, we have $R' \in \mathcal{T}$, with condition 8(a)iii in this case.

By induction on Q'' with the context obtained from R' with index n, we obtain a configuration $P' \in \mathcal{T}$ which is an extension of R' and of P. We conclude as before. □

Corollary B.15 *If Q is an extended ambient, there is a $S \in \mathcal{T}$ such that $[\![Q]\!]^{t\sharp} \twoheadrightarrow_d S$.*

Proof: Since $[\![\mathbf{0}]\!]^{t\sharp}$ is a translation state, we apply lemma B.14 with $[\![\mathbf{0}]\!]^{t\sharp}$ to generate the context using index 0, and with extended ambient Q. □

Lemma B.16 (Translation Invariant) *Let $P \in \mathcal{T}$ and $P' \in \mathcal{J}$ such that $P \mapsto P'$, then there is a $Q \in \mathcal{T}$ such that $P' \twoheadrightarrow_d Q$.*

Proof: Let $P \in \mathcal{T}$ and $P \mapsto P'$. The set of strings $\{\alpha^n h^n \mid n \in N\}$ is prefix-closed and h^n never appears in α^n.

Relying on the partition of reduction steps in translation states, we proceed by case on the steps $P \mapsto P'$: initial steps, completion steps, communication steps, replication steps, migration steps, continuation steps, delegation steps and forwarding steps. (No deterministic step initially applies in a translation state.)

Migration step Let $C_\alpha(\cdot) \stackrel{\text{def}}{=} T \wedge \alpha h^n[D^n \wedge E^n : \cdot \mid M^n]$. We have:

$$
\begin{array}{rcll}
P & \equiv & C_{\alpha^n}(\mathtt{go}(e^p.here); (I^n_{b,e^n,e^p} \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa))) & (16)\\
& \mapsto_{Go} & C_{\alpha^p h^p}(I^n_{b,e^n,e^p} \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa)) & (17)\\
& \equiv_d & C'_{\alpha^p h^p}(s^n(b,i,e^p,\emptyset) \mid e^p.amb(i,b,e^n) \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa)) & (18)\\
& \to_d & C'_{\alpha^p h^p}(s^n(b,i,e^p,\emptyset) \mid e^p.amb(i,b,e^n) \mid \kappa() \mid F^n) & (19)\\
& \to^*_d & P' & (20)
\end{array}
$$

where we assume that `uid` i in I^n_{b,e^n,e^p} binds a name i that does not appear elsewhere in P, where $C'_\cdot(\cdot)$ is $C_\cdot(\cdot)$ with the additional definition `uid` i in R, and where F^n is the parallel composition of flushed messages $in^n(d, \kappa^s)$ and $out^n(e, \kappa^s)$ for all entries in the log l^n.

The structural equivalence (16) is a structural rearrangement of top-level locations, and α-conversion to a globally-fresh name i. Step (17) is a migration step. The equivalence (18) is Lemma B.11 applied to the definition `uid` i. Step (19) is the deterministic reduction of the $Flush$ construct. Steps in (20) are COMM steps moving the $e^p.amb$ message to h^p ($e^p.amb = amb^p$), and possibly moving the κ message to its defining location.

By hypothesis, P meets all the conditions of definition B.8; and P' is P with the following differences: for all $m \in N$ such that $\alpha^m = \alpha^n h^n \beta$, $\alpha'^m = \alpha^p h^p \beta$, $I' = I \wedge$ `uid` i, $M'^p = M^p \mid amb^p(i,b,e^n)$, $S'^n = s^n(b,i,e^p,\emptyset)$, $M'^n = M^n \mid F^n$. There is also a message $\kappa()$ in its defining location.

1. holds with $\alpha'^n = \alpha^p h^p$.

2. holds since n being a transient state for P implies $n \neq 0$.

3. holds because it held for P, thus $M'^p = M^p \mid amb^p(i', b, e^n)$ is well-formed because M^p was. M'^n is well-formed because it is a parallel composition of M^n and in^n and out^n messages.

4. holds because the log l^n is empty.

5. The new continuation message κ is unique because of the condition 8(b)ii for P: the only possible occurrence of κ was removed from the log if it was present, and all the messages generated by $Flush$ contain a different continuation message name. Let us consider all the other occurrences of any κ^s that were in l^n. Because of condition 5 for P, any occurrence in a sub^r_{in} message is in a message of the form $sub^r_{in}(i^n, c, \kappa^s)$, because $\kappa^s \in l^n$. Because of condition 8 on P, we know there is no $s^n(b, i^n, e^m, l^n)$ message in P for any m. Thus all messages on sub^r_{in} with a continuation κ^s as argument were stale. Because of condition 8(b)ii, there cannot be any r^n

message in P, so there is none in P'. Thus the log being empty in P' respects condition 5. All the κ^s in the in^n and out^n messages created by $Flush$ are unique because they were in the log, thus not already present in a in^r, out^r, $open^r$, rcv^r or o^r message in P (because of condition 5). There is no $sub^r_{in}(i', c, \_)$ nor any $sub^r_{out}(i', c, \_)$ because none was present (the uid is new) and none was created. There is no r^n message either, l^n is empty, and all the κ^s in newly created in^n and out^n are unique. Thus condition 5 is true for P'.

6. holds because no modification occured for these messages.

7. holds because no $\kappa \in K_c$ was consumed or created.

8. holds because S^n is the $s^n(b, i', e^p, \emptyset)$ message and h^p is the enclosing location of h^n. By condition 8(b)ii, there were no $amb^r(i^n, \_, e^n)$ message nor any r^n nor o^n message in P. However, in P', a message $amb^p(i, b, e^n)$ exists, so condition 8(a)i is verified for β being the empty word, and both other conditions are not met so condition 8 is true.

Continuation step If the definition used is $\kappa() \triangleright [\![Q^\kappa]\!]_{e^n}$, let us consider the context $C(\cdot)$ such that $C(\kappa()) = P$. We have $C(\mathbf{0}) \in \mathcal{T}$. (The only condition to check is condition 5, but $C(\mathbf{0})$ is P without the message $\kappa()$, thus the condition holds.) By lemma B.14, we have: $P \mapsto C([\![Q^\kappa]\!]_{e^n}) \twoheadrightarrow_d P'$ where $P' \in \mathcal{T}$.

The same argument holds if the continuation holds a name a as an argument, with $a \in F \cup L$, since $[\![Q^\kappa]\!]_{e^n}\{^a/_x\} = [\![Q^\kappa\{^a/_x\}]\!]_{e^n}$ if the continuation definition is $\kappa(x) \triangleright [\![Q^\kappa]\!]_{e^n}$ after α-renaming bound names so that a does not occur bounded in Q^κ.

Replication step The same argument holds if the continuation calls another copy of κ because the only copy (condition 7) was consumed, thus there is one and only one copy present after emitting it.

Delegation step Condition 3 is verified after an additional COMM step to bring the sub^m_{in} or sub^m_{out} message to h^m.

Condition 4 holds because the entry added is of the correct form and $\kappa \in K_a$ because of condition 5 with $b \in F \cup L$.

Condition 5 is verified because the only reference to κ^s in the in^n or out^n message is removed, and a new reference appears in the $sub^m_{in}(i^n, \_, \_)$ or $sub^m_{out}(i^n, \_, \_)$ message, as well as once in l^n.

All other conditions are preserved.

Communication step of the form:

$$C(rcv^n(\kappa) \mid snd^n(a)) \mapsto C(\kappa(a))$$

with $a \in F \cup L$ by condition 6 on P. After a possible COMM step to bring the message on κ to its defining location, all conditions except condition 6 hold immediately.

$rcv^n(\kappa)$ being present in M^n, there is no other occurrence of κ in P, thus condition 6 holds in P'.

Initial steps with the rule being used being in D_1^n consuming a message on amb^p. In all cases, after a COMM step to bring the r^p or the o^p message to its defining location, conditions 1, 2, 3, 4, 6, and 7 hold immediately.

Condition 5 holds because in the $sub_{in}^n(i^p, \_, \_)$ and $sub_{out}^n(i^p, \_, \_)$ case, the presence of this message implies (by the same condition for P) that the only other occurrence of the continuation is in l^p. These messages cannot be stale because both reductions demand that a message $amb^n(i^p, b, e^p)$ be present, which implies (condition 8(a)i) the presence of a message $s^p(b, i^p, e^p, l^p)$ in P. In both case, a r^p message is created, carrying the same continuation. The l^p log is still the only other place where the continuation can be found. The introduction of a message on $open^n$ is simpler because it cannot be stale and no log holds the continuation, thus it is unique in P and also in P' in the o^p message.

Condition 8a immediately holds for h^n. For h^p, the presence of a $amb^n(i^p, a, e^p)$ message implies that $S^p = s^p(a, i^p, \_, \_)$ (by condition 8(a)i for P), and this message is not consumed. There is also no r^p nor o^p message in P. All three possible reductions of D_1 remove the unique message of the form $amb^n(\_, \_, e^p)$, which is unique because of condition 8(a)i in P. The reductions creating a $r^p(e^r, \kappa^s)$ message fulfill condition 8(a)ii and this condition only, because there is no $amb^t(\_, \_, e^p)$ nor o^p message in P'. The reduction creating a $o^p(\kappa^s)$ satisfies condition 8(a)iii, This message is also unique because only one is created and there were none before, and there is also no $amb^t(\_, \_, e^p)$ nor r^p message in P'.

Completion step with the rule being used being in D_2^n. In both reductions, conditions 2, 7, and 6 are immediately verified, as well as condition 4, because the log is not modified.

Let us first study the opening case. Let P' be the process obtained after the reduction $s^n(a, i^n, e^m, l^n) \mid o^n(\kappa) \triangleright f^n(e^m) \mid \kappa() \mid Flush(l^n, e^m.in, e^m.out)$, the deterministic flushing of l^n and COMM transport of in^m, out^m and κ if it is not in K^n. P' satisfies conditions 1 and 3.

Because of condition 5 on P, we know that any κ in a message $sub_{in}^p(i^n, b, \kappa)$ or $sub_{out}^p(i^n, b, \kappa)$ has an entry in l^n. Since the $s(a, i^n, e^m, l^n)$ message is being consumed and not reemitted, these messages are stale. Because of condition 8a, we know there is no r^n message, so no associated continuation in the log. Thus all the non-stale continuations in l^n only occur in the log (because of condition 5), thus the $in^m(\_, \_)$ and $out^m(\_, \_)$ messages generated by the flush satisfy condition 5.

Condition 8(b)i is partially satisfied because S^n is now $f^n(e^m)$, $\alpha^n = \alpha^m h^m$ because of condition 8(a)iii for P. However, since the $s^n(a, i^n, e^m, l^n)$ message disappears, this breaks condition 8 if there is a message $amb^q(i^n, a, e^n, r^n, o^n)$. This is not possible by condition 8(a)iii for h^n in P. Thus condition 8 holds.

Let us study the relocation case. Let P' is the process after the reduction $s^n(a, i^n, e^t, l^n) \mid r^n(e^m, \kappa) \triangleright \mathsf{go}(e^m.here); R'$. P' immediately satisfies conditions 3 and 6. Condition 5 also holds because the continuation in the r^n message was present in the l^n (condition 5 in P) and is now only present in l^n.

Condition 8 is satisfied because $S^n = \mathsf{go}(e^m.here).R'$ and, as was just shown, κ only occurs once in l^n and nowhere else in P', except maybe in stale messages.

We also know because of condition 8(a)ii and the presence of r^n that there is no message of the form $amb^p(\_,\_,e^n)$, nor any o^n, which is still true in P'. The unique message on r^n has disappeared.

Forwarding step We will study reductions of the form $f^n(e^m) \mid C^n \triangleright f^n(e^m) \mid e^m.C$ where C is one of the messages forwarded. P' is the process after the reduction and a COMM reduction to transport the forwarded message. In all cases, conditions 1, 2, 3, 4, 6, and 7 immediately hold for P', and condition 8 holds for h^n in P'.

If C is a message $snd(b)$ then all other conditions hold.

If C is a message $in(b,\kappa)$, $out(b,\kappa)$, $open(b,\kappa)$ or $rcv(\kappa)$, then condition 5 hold because the message carrying the continuation is replaced by another message carrying the same continuation.

If C is a message $sub_{in}(i^p,b,\kappa)$ or $sub_{out}(i^p,b,\kappa)$, then because of condition 5, κ is also in l^p and nowhere else. This is still true after the reduction, so condition 5 holds.

If C is a message $amb(i^p,b,e^p)$, then condition 8(a)i is true for some h^s in P. It is still true after the reduction with the new $amb^m(i^p,b,e^p)$ message, so condition 8 holds with $\alpha^p = \alpha^m h^m h^n \beta$: by condition 8(a)i on P, for all h^t in β, $S^t = f^t(\_)$; we also have $S^n = f^n(\_)$. Thus, condition 8(a)i holds. □

In the following, we write $\twoheadrightarrow_x$ for $\twoheadrightarrow_d \mapsto_x$ with $x \in B \cup X$. This reduction is defined for $P \in \mathcal{J}$ only if $\twoheadrightarrow_d$ is defined, hence only if there is a $P' \in \mathcal{T}$ such that $P \twoheadrightarrow_d P'$.

Lemma B.17 *There is no infinite series of normalized bookkeeping reductions $\twoheadrightarrow_B$.*

Proof: Let us consider a series of normalized bookkeping reductions:

$$P_1 \twoheadrightarrow_d Q_1 \mapsto_B P_2 \twoheadrightarrow_d Q_2 \ldots$$

From this series, we extract the series $Q_1, Q_2, \ldots$, and prove that this series is necessarily finite. By definition of $\twoheadrightarrow_d$, we know that $Q_i \in \mathcal{T}$ for all i. We define the depth of a message defined in D^n as the word size of β if $\alpha^n h^n = \alpha^p \beta$ where α^p is the first enclosing location which is alive. Using the partition we just defined, we write g, κ, o, for the number of go steps, continuation steps, delegation steps respectively enabled. We write f_0 for the sum of the depth of messages on in, out, $open$, amb, rcv, and snd in Q, and f_1 for the sum of the depth of messages on sub_{in} and sub_{out} in Q.

We consider the well-founded lexical order on the tuple (g, κ, f_0, o, f_1), and associate such a tuple to each Q_i. We check that the tuple strictly decreases as i increases, which implies that the series is necessarily finite.

Forwarding steps for messages on sub_{in} and sub_{out} decreases their depth, and eventually only enable initial steps, which are not bookkeeping steps. Delegation steps consume messages on in, out or $open$ which were at depth 0, but decreases the number of delegation steps enabled. They create sub_{in} or sub_{out} messages which might contribute to f_1, but the resulting tuple is still strictly smaller. Forwarding of other messages decreases their depth and do not enable any continuation nor migration steps. Continuation steps do not create any migration steps nor any other continuation step. Migration steps do not enable new migration steps. □

We introduce a notion finer that the one of translation state, as some properties needed hereafter are true only for these processes.

Definition B.18 *P is a well-formed process if there is an ambient process Q and an extended ambient process Q' such that $Q \rightarrow^*_{12C} Q'$ and $[\![Q']\!]^{t\sharp} \rightarrow^* \twoheadrightarrow_d P$.*

The following lemma insures that migrations are always enabled in a well-formed process. It is similar to condition 3 of lemma B.2. By definition of $\twoheadrightarrow_d$, we have $P \in \mathcal{T}$.

Lemma B.19 *Let P be a well-formed process. For any $n \in N$ such that h^n is in a transient state, the $\mathsf{go}(e^p.here)$ is not dead-locked (h^p is not a sub location of h^n).*

Proof: We first define a partial order on locations h^n, $n \in N$. We say that $h^n < h^p$ iff $h^p \in \alpha^n$ (h^n is a sublocation of h^p). We also define a relation $\mathcal{R}$ on locations. We say that $h^n \mathcal{R} h^p$ iff $r^n(e^p, \kappa)$ or $\mathsf{go}(e^p.here); R'^n$ is in M^n. We define $<_f$ as $\mathcal{R} \leq$.

We prove that the following hypothesis H is preserved through reduction: *there is no cycle in $<_f$*; and is valid for the translation of a correct extended ambient process. If H is true for P, then any migration is enabled, because the destination location is not a sublocation of the migrating location.

We first prove that the translation of a correct extended ambient Q satisfies H. A correct extended ambient is an extended ambient that satisfies lemma B.2. Following the definition of the translation, if $h_a <_f h_b$ and $h_b <_f h_c$, then three subprocesses are present in the extended ambient process:

$$\begin{aligned} Q1 &= X_c, c[E_c(i)] \\ Q2 &= \overline{i}.P_2, b[E_b(j)] \\ Q3 &= \overline{j}.P_3, a[R_a] \end{aligned}$$

where h^a, h^b, h^c are the locations corresponding to the ambients a, b and c, and where E_b and E_c are evaluation contexts. As can be seen in the subprocesses, necessarily $j \prec i$ as was defined in B.2. Thus if there is a cycle of length greater than one in the translated process for $<_f$, then there is also one for the extended ambient process, which violates condition 3 of lemma B.2.

If the cycle is of length one, then the extended ambient process has a subprocess of the form: $\overline{i}.P_i, a[E_a(i)]$, which also implies a cycle for $\prec$.

Let us now prove that H is preserved through reduction. The proof is very similar to the proof of condition 3 of lemma B.2. For the sake of clarity, we write $<'$, $\mathcal{R}'$ and $<'_f$ for the relations after the reduction. There are only three cases which modify the reductions: a continuation step which creates new locations, an initial step which creates a message on some r^n and a migration step. Completion step do not modify the location tree and replace the $r^a(e^b, \kappa)$ message with a $\mathsf{go}(e^b.here); R'^a$ primitive, which do not modify the relations.

Continuation step A continuation step does not modify the $\mathcal{R}$ relation —the translation involves non-extended ambients— but extends the $<$ relations. Let h^p be the location name of a newly created location. Then it is not possible that $h^a \mathcal{R} h^b$ and $h^b \leq' h^p$ because no newly created location can be a parent of an already present location. Thus $<'_f = <_f$ and H is preserved.

Initial step

In 1 Let us suppose the reduction occurs in h^d. Then M^d contains the messages $amb^d(j,b,e^b)$, $amb^d(k,a,e^a)$ and $sub^d_{in}(k,b,\kappa)$. This reduction creates a message $r^a(e^b,\kappa)$, thus $\mathcal{R}' = \mathcal{R} \cup \{(h^a,h^b)\}$. The location tree is not modified, thus $\leq$ does not change. We then have $<'_f = <_f \cup \{(h^a,h^c) | h^b \leq h^c\}$.

Let us consider a cycle in $<'_f$. This cycle did not exist before thus it necessarily includes a couple $h^a <'_f h^c$ with $h^b \leq h^c$. If there are several such couples, a shorter cycle can be found. Indeed, if $h^1 <'_f h^a <'_f h^2 <'_f \ldots <'_f h^3 <'_f h^a <'_f h^4$ are part of the cycle, a shorter cycle would contain $h^1 <'_f h^a <'_f h^4$ instead. Thus it is possible to only study the case where there is only one couple of the cycle which was not in $<_f$.

If h^c is h^a then $h^b \leq h^a$. Thus $h^a \in \alpha^b$. By condition 8(a)i of lemma B.16 and the message $amb^d(j,b,e^b)$, we have $\alpha^b = \alpha^d h^d \beta$. h^a is not opened (by condition 8 and the presence of the message $amb^d(k,a,e^a)$) and h^a is different from h^d, so $h^d < h^a$, which is impossible by condition 8(a)i for h^a.

h^c being different from h^a, the cycle has a length greater than one, thus there is some $h^e <'_f h^a$. This cannot be a new couple of the relation $<_f$, therefore $h^e <_f h^a$. By condition 8(a)i, $h^a \leq h^d$. Thus we have $h^e <_f h^d$. We know that $h^b \leq h^c$. Either $S^c = \mathsf{go}(e^z.here)$ or $r^c(e^z,\kappa)$ is a message of M^c, thus by condition 8 of translation states, h^c is not opened and there is no message $amb^p(\_,\_,e^c)$. Thus $h^b < h^c$ (they cannot be equal because of the message $amb^d(j,b,e^b)$). By condition 8(a)i and the presence of message $amb^d(j,b,e^b)$, we have $\alpha^b = \alpha^d h^d \beta$ and every $h^s \in \beta$ is opened. Thus we have $h^d \leq h^c$. This implies $h^e <_f h^c$, which in turn implies that the cycle exist in $<_f$, which is not possible.

Out 1 Let us suppose the reduction occurs in h^d. Then M^d contains the messages $s(d,i^d,e^b,l^d)$, $amb^d(k,a,e^a)$ and $sub^d_{out}(k,d,\kappa)$. This reduction creates a message $r^a(e^b,\kappa)$, thus $\mathcal{R}' = \mathcal{R} \cup \{(a,b)\}$. The location tree is not modified, thus $\leq$ does not change, and $<'_f = <_f \cup \{(h^a,h^c) | h^b \leq h^c\}$.

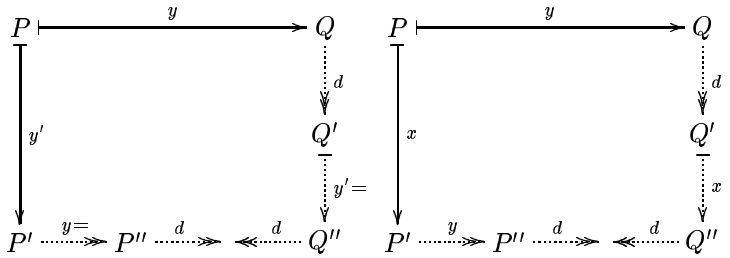
By the same argument as before, we reduce a cycle of $<'_f$ to a cycle with all couple in $<_f$ except one: $h^a <'_f h^c$. We have $h^b \leq h^c$.

By conditions 8 and 8(a)i of translation states, we necessarily have $h^a < h^d < h_b$. If the cycle is of length one, then $h^b \leq h^a$ which is impossible. Otherwise, the sequence $h^e <_f h^a <'_f h^c$ can be replaced by $h^e <_f h^c$ and the cycle was present before the reduction, which is not possible.

Migration step Let us suppose that $h^a <_f h^b$. Let us define $Z = \{h^z | h^z \leq h^a\}$ and $C = \{h^c | h^b \leq h^c\}$. We then have $\mathcal{R}' = \mathcal{R} \backslash \{(a,b)\}$, and if $h^p <' h^q$, then either $h^p < h^q$ or $h^p \in Z$ and $h^q \in C$ are true. We thus have $<'_f \subset <_f \cup \{(h^c,h^d) | h^c <_f h^a \wedge h^a <_f h^d\}$. Let us suppose there is a cycle. Any couple in this cycle that is not a couple of $<_f$ can be replaced by two consecutive couples of $<_f$, thus there was a cycle in $<_f$, which is not possible. □

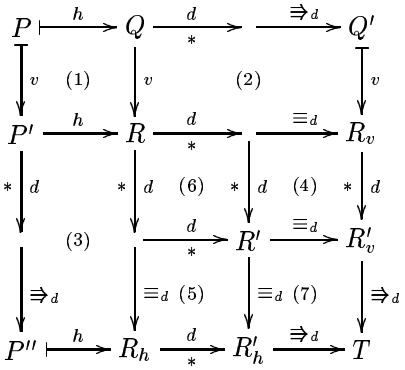
Lemma B.20 (Commutation lemma) *Let P be a well formed process. The fol-*

lowing diagrams hold for $\mapsto_y, \mapsto'_y \in B$, $\mapsto_x \in X$.



Proof: We first show that if the two redexes involved in the two steps do not interfere, then it is possible to close the diagram. We write $\rightarrow_h$ and $\rightarrow_v$ for the join steps corresponding to the redexes being reduced horizontally and vertically, respectively.

The following diagram holds, using the same configuration names as in the lemma:



The first tile **1** holds by the non-interference hypothesis of the two reduction steps considered. Tile **2** holds by lemma B.16 to yield translation state Q', by definition B.10 of deterministic steps which commute with all other steps, and by lemma B.11 which states that $\equiv_d$ is a strong bisimulation. Tile **3** holds for the same reasons, with translation state P''. Tiles **4** and **5** are built by applying lemma B.16, and extracting the series of deterministic steps. Tile **6** holds by commutation of deterministic steps, with R' having no deterministic step enabled: by definition of deterministic steps, the configuration reached after all deterministic steps is deterministic. Tile **7** holds by chosing the adequate vertical and horizontal step: translation state T exists by lemma B.16 from either Q' or P'', and both path $\equiv_d \Rrightarrow_d$ can be chosen to converge, by definition of $\equiv_d$ and $\Rrightarrow_d$.

We now proceed by case on the vertical reduction, proving that the reductions do not interfere.

Migration step Migration steps do not prevent any other step from occuring, nor are impeded by any other step, except other migrations. Since the process is a well formed process, a property preserved by reduction after deterministic normalization, by lemma B.19, the migration is enabled and can occur.

Continuation step Continuation steps commute with all considered steps, as they only depend on the location in which they are defined, which is not modified by the other steps, and as they do not remove any message except the continuation call itself.

Replication step As for continuation steps, replication steps commute with all considered steps.

Delegation step A possible problem concerning the commutation of a delegation step with the considered step is a Fwd reduction which would involve the message on in^n or out^n. However, if there is such a reduction, it implies that there is a message on f^n, which in turn, by condition 8(b)i of lemma B.16, implies that there is no message on s^n, thus that delegation is not possible.

Another problem is the modification of the log. However, as the log used in the deterministic $Flush$ step is held as an argument of the $Flush$, this does not prevent commutation.

Communication step As for delegation steps, the only horizontal step which could prevent the communication to occur would be a Fwd reduction, which is not possible by a similar reasoning. The communication step itself cannot prevent any horizontal reduction from occurring, with the exact same results. Thus all commutations hold.

Initial step Once again, this step could be impeded by the forwarding of one of the necessary messages (on amb^n and sub^n_{in}, sub^n_{out} or $open^n$). This is not compatible with the presence of the message on s^n. Nothing else prevents any commutation with horizontal steps.

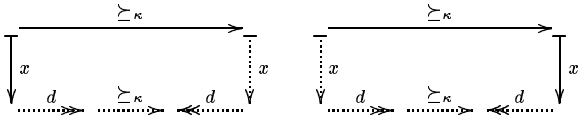
Completion step Every completion step immediately commute with every horizontal step.

Forwarding step These steps do not prevent any horizontal reduction from occurring with the same result, thus commute with all of them. □

Next, we define simplification relations on translation states, and express their properties as commutation diagrams. The first simplification $\succeq_\kappa$ consists of the removing of definitions which cannot be used anymore.

Lemma B.21 *For all $P, Q \in \mathcal{T}$, we write $P \succeq_\kappa Q$ when P is obtained from Q by adding κ to $K_{Q,ab}$ (or i to I_Q), for some name κ (or i) that does not occur in Q.*

For every $\mapsto_x \in X \cup B$, we have the diagrams:

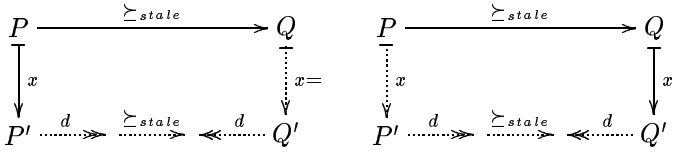


Proof: The proof is immediate as the definition of κ (or i) in P is never used, and as new local continuation names appearing in Q can be taken different from κ. A translation state is reached after each vertical step by applying lemma B.16. □

Another simplification relation $\succeq_{stale}$ removes stale messages from translation states.

Lemma B.22 *For all $P, Q \in \mathcal{T}$, we write $P \succeq_{stale} Q$ when P is Q with a single additional stale message.*

For every $\mapsto_x \in X \cup B$, we have the diagrams:



Proof: Let us first prove the second diagram. By lemma B.16, there is a $Q'' \in \mathcal{T}$ such that $Q' \twoheadrightarrow_d Q''$. Since P is Q with an additional message, the series of reduction $Q \mapsto_x \twoheadrightarrow_d Q''$ also applies in P, yielding P'' which is Q'', with the same additional message at the same location, hence $P'' \succeq_{stale} Q''$.

Let us now prove the first diagram. Assume $P \mapsto_x P'$; if this steps does not consume the additional message, we conclude as above. Otherwise, let $sub^n_{in}(i^m, \_, \_)$ (or $sub^n_{out}(i^m, \_, \_)$) be the stale message. According to our analysis of steps enabled in translation states, $\mapsto_x$ is either an initial step or a forwarding step using a rule of D^n.

By definition of stale messages, there is no message of the form $s^m(\_, i^m, \_, \_)$ in P. By condition 8b on translation states, there is no message of the form $amb^n(i^m, \_, \_)$ in M^n, hence no initial step may consume the stale message.

A forwarding step $P \mapsto_{Fwd} P'$ uses $S^n = f(e^p)$ to consume the stale message and produce a message $sub^p_{in}(i^m, \_, \_)$ (or $sub^p_{out}(i^m, \_, \_)$). Using a deterministic COMM-step $P' \to_d P''$, this message is moved to M^p, and becomes a stale message in M^p. We obtain $P'' \succeq_{stale} Q$ for the same state Q, with a different stale message. □

Location boundaries in the join calculus cannot be torn apart to match ambient OPEN 2 steps; instead, we rely on a simplification relation $\succeq_{Fwd}$ that merges an opened location and its enclosing location. In the lemma below, m is opened in n and all forwarding steps in m have been performed.

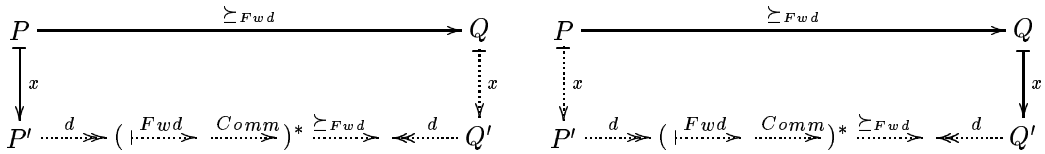
Lemma B.23 (Opened ambient) *Let $P, Q \in \mathcal{T}$. We write $P \succeq_{Fwd} Q$ when, for some $m \in N_P$, P meets the conditions:*

1. *$S^m_P = f^m(e^n)$ (hence $\alpha^m_P = \alpha^n_P h^n$ by condition 8(b)i);*
2. *M^m_P is a parallel composition of messages on names in K^m_P;*

and that Q is P with the following changes:

3. *$N_Q = N_P \setminus \{m\}$;*
4. *$K^n_Q = K^n_P \cup K^m_P$ (with identical guarded processes);*
5. *$M^n_Q = M^n_P \mid M^m_P$;*
6. *for every $p \in N_Q$, α^p_Q is obtained from α^p_P by deletion of h^m (when present);*
7. *e^n is substituted for every occurence of e^m.*

For every $\mapsto_x \in X \cup B$, we have the following diagrams:



Proof: The proof consists of showing that both configurations execute the same reduction, with some additional steps for the configuration on the left which might need to forward the results of the reductions to reach a state that is still in the relation.

In the following, we say that a step $P \to P'$ involves the opened ambient if it occurs in h^m, creates a message on a name defined in D^m, creates a `Flush` message with arguments defined in D^m, or involves an occurrence of e^m.

If the step does not involve the opened ambient, then by lemma B.16 we have $P' \twoheadrightarrow_d P''$ with $P'' \in \mathcal{T}$.

These deterministic steps necessarily do not involve the opened ambient: Flush steps create messages which, by definition, are not defined in D^m, do not occur in h^m and do not involve e^m. Comm steps cannot bring messages to h^m since the messages created are not on names defined in D^m. Since P is a translation state, there are no other deterministic steps than the one created by the reduction $P \mapsto_x P'$.

We derive a series of steps corresponding the the reduction $P \mapsto_x \twoheadrightarrow_d P''$ for Q. Since the first step does not involve the opened ambient, there is a corresponding redex which can do the same step. The resulting configuration Q' is very similar to P', with the same differences as between P and Q. Since both configuration are obtained from the reduction of the same redex, the series of deterministic steps for P', which do not involve the opened ambient, is also enabled in Q'. This yields a configuration Q'' which does not have any other deterministic step enabled: there were none other in Q' than those created by the step $Q \mapsto_x Q'$. By lemma B.16, we have $Q' \twoheadrightarrow_d Q'''$ with $Q''' \in \mathcal{T}$. Since deterministic steps commute, and by definition of $\Rightarrow_d$, we have $Q'' \Rightarrow_d Q'''$.

Hence, from Q, reductions $Q \mapsto_x \twoheadrightarrow_d Q'''$ matching those from P yield a translation state. We check that we have $P'' \succeq_{Fwd} Q'''$.

Conditions 1 and 2 are immediately met, because the message on f^m cannot disappear, and no message was create in h^m. Condition 3 is immediately verified, since each create index name is created in both configurations. Condition 4 is met because no definition could be added to K_P^m, and if one was added to K_P^n, it was also added to K_Q^n by the same series of steps. Condition 5 is met because no message was added to M_P^m, and any message added to M_P^n is also added to M_Q^n by the same steps. Condition 6 can be affected if the reduction is a $\mathtt{go}(h^p)$ for some h^r and $h^m \in \alpha^p$. However, the same reduction occurs in Q, $\alpha_P^r = \alpha_P^n h^n h^m \delta^p h^p$ and $\alpha_Q^r = \alpha_Q^n h^n \delta^p h^p$ thus the condition is met. Condition 7 is immediately met, since no occurence of e^m is involved in the steps.

Let us consider the cases where the opened ambient is involved, by case on the reduction.

Migration step Then it is necessarily a $\mathtt{go}(e^m.here)$ reduction for some h_P^r to P_1. Because of the relation between P and Q, there is a corresponding $\mathtt{go}(e^n.here)$ in h_Q^r. h^n is not a sublocation of h^r because otherwise h^m would be one too and the reduction could not occur. Thus Q can also execute the reduction to Q_1. The `Flush` step can be executed by both configurations, to yield P_2 and Q_2. These configurations are not yet translation states because of the message on $e^m.amb$ for P_2 and $e^n.amb$ for Q_2. The message in Q_2 is simply sent to h^n by a Comm step and the resulting configuration Q' is a translation state. The message in P_2 is forwarded to h^m by a Comm step. The resulting configuration P_3 is a translation state but does not meet condition 2 of the relation. Since we have $S^m = f^m(e^n)$, we use the definition $f^m(e^n) \mid amb^m(j, b, e_b) \triangleright f^m(e^n) \mid$

$e^n.amb(j, b, e_b)$ followed by a COMM step, to create the forwarded message on amb^n and to forward it to yield configuration P'. All conditions are now met for the relation between P' and Q'.

Continuation step If the reduction involves a continuation (ie a message whose name is defined in $K^m_{P,a}$ or $K^m_{P,b}$), then conditions 4 and 5 insure that a reduction involving the same definition can occur in h^n_Q. Every messages, except for continuations, which are created in M^m_P are forwarded using the appropriate forwarding step and sent to h^n_P using a COMM step. These messages are directly created in M^n_Q and do not need to be forwarded. This accounts for all the messages beside continuations because an ambient translation creates messages only on *in*, *out*, *open*, *rcv*, *snd* and *amb*. New definitions introduced are necessarily on names added to K^m_P, I_P and L_P, and they correspond to the same names added to K^n_Q, I_Q and L_Q. Thus, the configurations obtained are still in the relation.

Replication step The exact same argument applies to replication.

Delegation step A delegation step involving the opened ambient necessarily is a delegation step which creates a message on sub^m_{in} or sub^m_{out}, followed by a COMM step to yield P''. The corresponding step in Q will directly create a message on sub^n_{in} or sub^n_{out} with the same arguments, which is sent to h^n by a Comm step. An additional forwarding step followed by a COMM step for P'' will yield a configuration that is in the relation with Q''.

Communication step The only possibility for a communication step to involve the opened ambient is if the resulting continuation is defined in the opened ambient. This directly yields two processes which still are in the relation.

Initial step NO IN 1 or OPEN 1 reductions involve the opened ambient. An OUT 1 reduction might involve the opened ambient by creating a $r^p(e^m, \kappa)$ message. However, the two processes obtained after the reduction are still in the relation, using condition 7.

Completion step An OPEN 2 reduction might involve the opened ambient if the ambient is being opened in h^m, which implies that the Flush step contains arguments defined in D^m. After the Flush step, forwarding steps will be necessary to carry the messages to h^n. All conditions are easily verified. Another possibility for involving the opened ambient is if the continuation is defined in h^m. Then, the possibly affected conditions 4 and 5 are still verified after the reduction.

A MOVE 2 reduction simply yields a process which trivially satisfies condition 7, which is the only one affected.

Forwarding step A forwarding step involving the opened ambient when the destination of the forwarding is h^m. The relation $\succeq_{Fwd}$ insures that the same forwarding step in Q will carry directly the message in h^n. An additional forwarding step followed by a COMM step will achieve the same for P.

The proof of the second diagram is exactly the same, with the only exception of some occurrences of channels defined in n as defined in m. Both reductions proceed as described before, and after some forwarding steps and Comm steps, if the step

involves a name defined in h^m, the configurations obtained are still in the relation. □

Definition B.24 *Let $Q \in \mathcal{E}$. We write $Q \in \mathcal{E}^{\#}$ when $Q = \nu\tilde{a}.Q'$ where Q' has no restriction in evaluation context, and say that Q is a normalized extended ambient.*

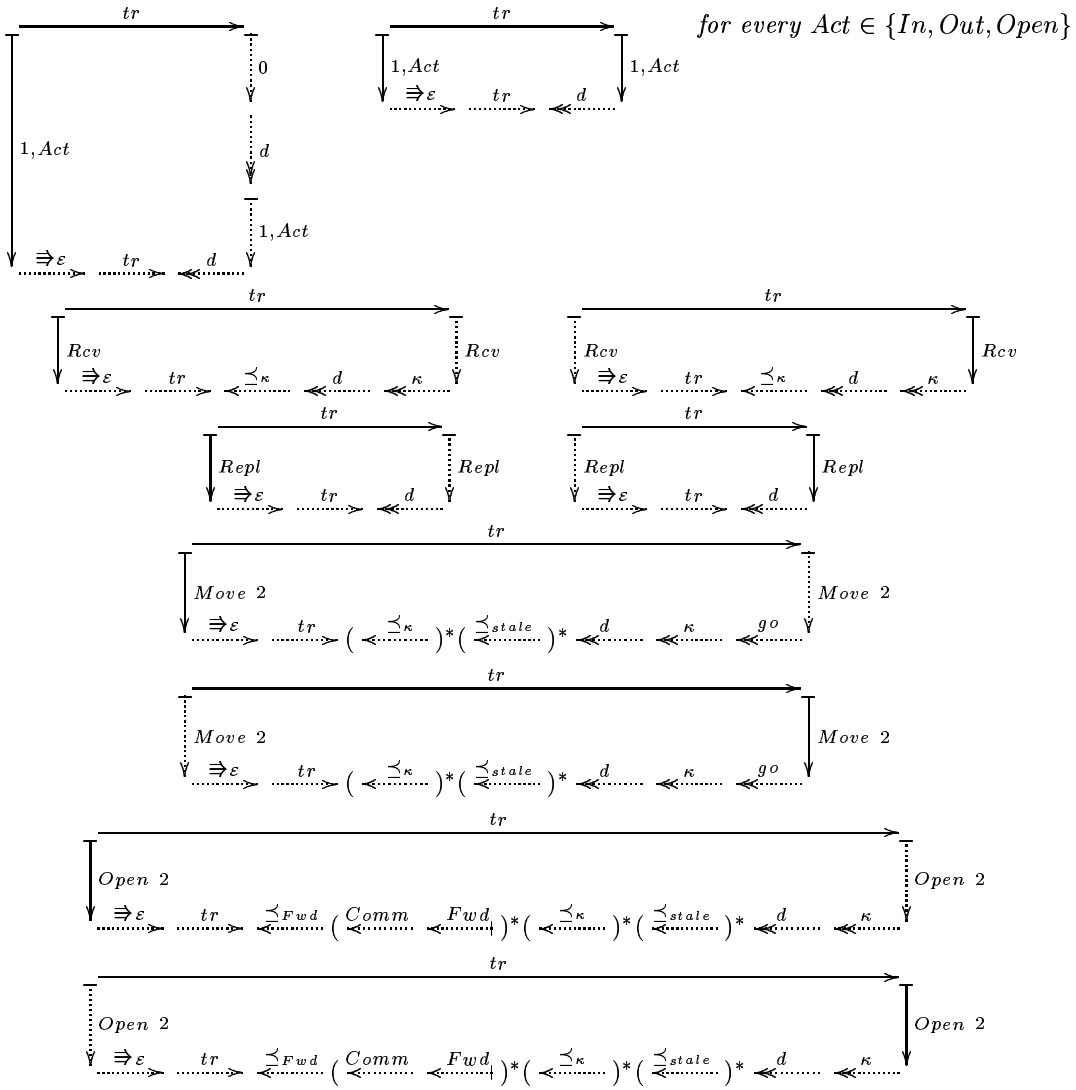
Definition B.25 *Let $P, Q \in \mathcal{E}$. We write $P \Rrightarrow_{\mathcal{E}} Q$ when $P \equiv Q$ by α-renaming and scope extrusion, and $Q \in \mathcal{E}^{\#}$.*

We restrict extended ambient steps to steps which are closed under structural equivalence which does not involve α-renaming nor scope extrusion. These extended ambient steps are written $\mapsto_X$.

We let $\xrightarrow{tr} \stackrel{\text{def}}{=} \xrightarrow{[\![]\!]^{t\sharp}} (\xrightarrow{0}\!\!\!\!\rightarrow)^* \xrightarrow{d}\!\!\!\!\rightarrow$ be a relation between normalized extended ambients $\mathcal{E}^{\#}$ and translation states $\mathcal{T}$. This new relation corresponds to translating the ambient, then doing an arbitrary number of delegation steps, before deterministic normalization.

We now state the main diagram, which establishes the relation between reduction steps in extended ambients and reduction steps in their translations.

Lemma B.26 (Operational Correspondance) *We have the following diagrams:*



Proof: The proof of each of the diagram is straightforward but fairly tedious. The main issue consists of proving that each relation can be used. Most of the work was accomplished by proving lemma B.16 which describe the shape of the join processes involved.

To apply an ACT 1 step, an additional 0 step might be necessary if it was not achieved in the 0 steps contained in the horizontal $\rightarrow_{tr}$ reduction series. Otherwise, the vertical part of the closure if straightforward, following the translation definition.

To apply the relation $\preceq_\kappa$, the continuation must not appear anywhere in the term. This condition is met as there is no stale message in the join term (stale messages are created by a MOVE 2 or a OPEN 2 step, and are removed with the $\preceq_{stale}$ relation) and because of condition 5 on translation states, which states that if there is a continuation message in the term, then there is no other occurrence of the name, except in stale messages. Removing the used continuation definition is necessary to yield a process that is a translation of the extended process.

The relation $\preceq_{stale}$ is applied as many times as there are stale messages in the term. These stale messages are created after a MOVE 2 or a OPEN 2 step, and are message on sub^n_{in} or sub^n_{out} if h^n was the location moving or being opened. The messages that are removed are those which were created in 0 steps of the horizontal $\rightarrow_{tr}$ series of reductions. As they carry a uid which is not correct anymore, they cannot appear in the translation of the extended ambient. They are actually replaced by the messages that are flushed from the log, with the correct uid. Thus, the 0 steps of the bottom $\rightarrow_{tr}$ series of reductions are the same as before. The strong equivalence that follows stale message simplification corresponds to the removing of the `uid` i binder for the this uid which does not occur in the term, as a new binder was introduced after the `go` step.

After an OPEN 2 step, an extended ambient has disappeared, while the corresponding location is still present with a message on f^n. To put these processes in correspondence, it is necessary to remove this location. First, all messages except those defined in K^n are forwarded to the location in which the ambient was opened through Fwd steps. This location is necessarily alive as there can not be any opened location generated by the translation. Then the process meets the conditions to apply $\preceq_{Fwd}$, which removes the opened location and puts its K^n definitions in the location in which it was opened. As a translation cannot generate continuation calls, as the continuation created by the OPEN 2 step was used and as the corresponding definition was removed with the $\preceq_\kappa$ relation, the only definitions and processes concerned are those for restriction and replication. □

We will now show some diagrams concerning barbs. First we define a notion of barbs for the translation states.

Definition B.27 (Translated barbs) *We let $\downarrow^4_b$ be the predicate on join processes such that $P \downarrow^4_b$ iff $T_b(P) \rightarrow^4 \downarrow_{yes}$.*

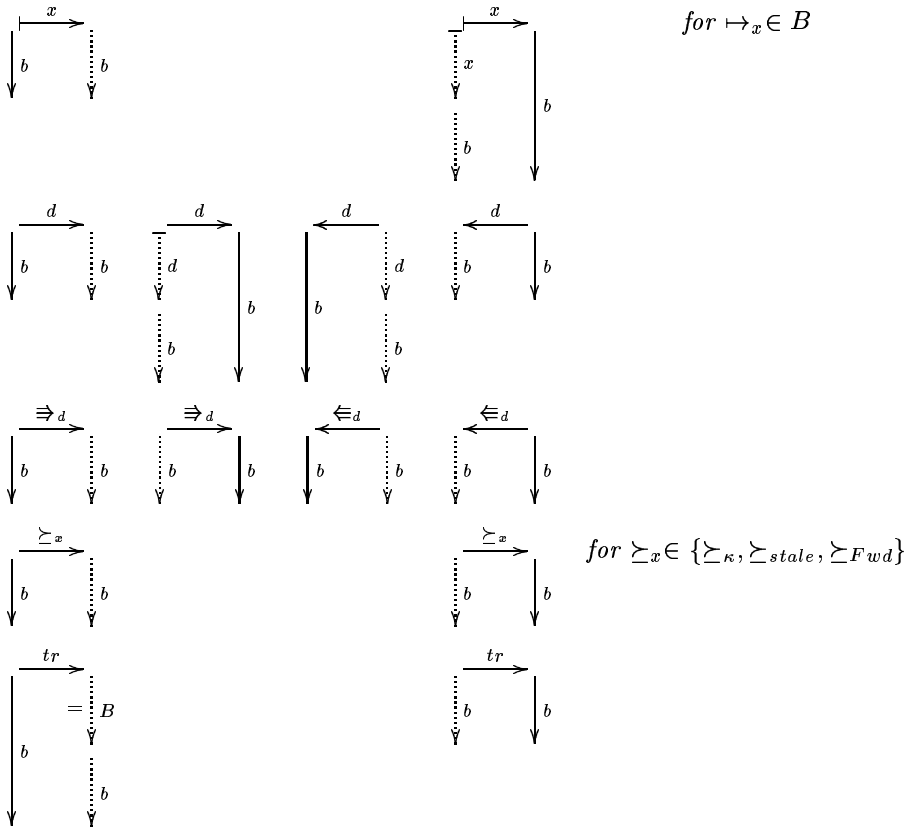
These four steps are as follows: a COMM step to bring the $p(t)$ message to its defining location, its consumption, a COMM step to bring the resulting $t(b)$ message to the location where D^0_t is, and its consumption if a message of the form $amb^0(i^m, b, e^m)$ is present in M^0. As this is the shortest reduction path to observe a barb on *yes* for some $P \in \mathcal{T}$, $P \downarrow^4_b$ if and only if $amb^0(i^m, b, e^m)$ is present in M^0.

Now we define a new reduction which represents the observation of a barb.

Definition B.28 *Let $P \in \mathcal{E}$, we let $\rightarrow_{\downarrow b}$ be the relation $P \rightarrow_{\downarrow b} \mathbf{0}$ if and only if $P \downarrow_b$. Similarly, let $R \in \mathcal{J}$, we let $R \rightarrow_{\downarrow b} \mathbf{0}$ if and only if $R \downarrow^4_b$. In the following diagrams, we simply write $\downarrow_b$ for these relations.*

We now close the diagrams where all the horizontal relations introduced are tried against the vertical barb relation.

Lemma B.29 *The following diagrams hold:*



Proof: To be written.

□

Lemma B.30 *Let $\mathcal{H}$ be the relation $\Rrightarrow_{\mathcal{E}} \xrightarrow{tr} \mathcal{H}'^*$ between well-formed processes and join configurations where*

$$\mathcal{H}' = \{\Rrightarrow_d, \Lleftarrow_d, \xleftarrow{d}, \xrightarrow{d}, \xleftarrow{\kappa}, \xleftarrow{Fwd}, \xleftarrow{0}, \xleftarrow{go}, \xleftarrow{\preceq_\kappa}, \xleftarrow{\preceq_{Fwd}}, \xleftarrow{\preceq_{stale}}, \}$$

We have $[\![\cdot]\!]^{t\sharp} \subseteq \mathcal{H}$.

Proof: We prove that $[\![\cdot]\!]^{t\sharp} = \Rrightarrow_{\mathcal{E}} \xrightarrow{tr} \xleftarrow{d} (\xleftarrow{0})^*$.

First we relate the translations of two structurally equivalent processes of the extended calculus. Let $Q, Q' \in \mathcal{E}$ such that $Q \equiv Q'$. We first prove by induction on the structural equivalence proof that for any e and any D, $[D : [\![Q]\!]^\sharp_e] \equiv_d [D : [\![Q']\!]^\sharp_e]$. The proof is straightforward since the structural equivalence is preserved through the traduction, except for scope extrusion across ambient boundaries (structural rules

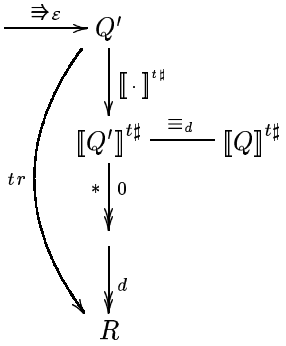
R2 and R2X). Let us consider the R2 case.

$$
\begin{aligned}
& [D : [\![\nu n.a[P]]\!]^\sharp_e] && (21)\\
\stackrel{\text{def}}{=}\ & [D : \texttt{def fresh}\ n\ \texttt{in}\ [\![a[P]]\!]^\sharp_e] && (22)\\
\equiv\ & [D \wedge \texttt{fresh}\ n : [\![a[P]]\!]^\sharp_e] && (23)\\
\stackrel{\text{def}}{=}\ & [D \wedge \texttt{fresh}\ n : \texttt{def}\ h^n[D^n \wedge \texttt{uid}\ i^n : [\![P]\!]^\sharp_{e^n} \mid s^n(a, i^n, e, \emptyset) \mid amb(i^n, a, e^n)]\ \texttt{in}\ (24)\\
\equiv\ & [D \wedge \texttt{fresh}\ n : \mathbf{0}] \wedge h^n[D^n \wedge \texttt{uid}\ i^n : [\![P]\!]^\sharp_{e^n} \mid s^n(a, i^n, e, \emptyset) \mid amb(i^n, a, e^n)] && (25)\\
\equiv_d\ & [D : \mathbf{0}] \wedge h^n[D^n \wedge \texttt{uid}\ i^n \wedge \texttt{fresh}\ n : [\![P]\!]^\sharp_{e^n} \mid s^n(a, i^n, e, \emptyset) \mid amb(i^n, a, e^n)] && (26)\\
\equiv\ & [D : \texttt{def}\ h^n[D^n \wedge \texttt{uid}\ i^n \wedge \texttt{fresh}\ n : [\![P]\!]^\sharp_{e^n} \mid s^n(a, i^n, e, \emptyset) \mid amb(i^n, a, e^n)]\ \texttt{in}\ (27)\\
\stackrel{\text{def}}{=}\ & [D : [\![a[\nu n.P]]\!]^\sharp_e] && (28)
\end{aligned}
$$

The structural equivalence of step 23 is allowed by choosing n fresh, the second structural equivalence step 25 is allowed since $a \neq n$ by definition of R2, and the $\equiv_d$ step 26 corresponds to definition B.10.

Hence, by definition of the translation B.7, we have $[\![Q]\!]^{t\sharp} \equiv_d [\![Q']\!]^{t\sharp}$.

We have the following diagram:



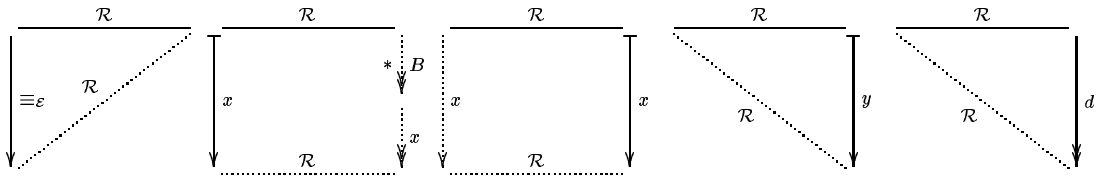
This diagram holds by the definition of the $\xrightarrow{tr}$ relation as well as the relation between the translations of two structurally equivalent ambient processes. We conclude using the definition of $\equiv_d$ and $\xrightarrow{0}$ to derive:

$$\equiv_d \xrightarrow{0} \;=\; \equiv_d \xrightarrow{d} \xrightarrow{0} \;=\; \xrightarrow{d} \xrightarrow{0} \;=\; \xrightarrow{0}$$

□

We are now ready to establish the correctness of the translation. The next theorem is more precise than Theorem 5.4. It uses a notion of hybrid barbed strong bisimulation up to bookkeeping.

Theorem B.31 *Let $\mathcal{R}$ be a relation between extended ambient processes and join configurations such that $P \mathcal{R} Q$ implies $P \Downarrow_b$ iff $T_b(Q) \Downarrow_{yes}$ and that satisfies the following diagrams for every $\mapsto_x \in X$ and $\mapsto_y \in B$:*



Such a relation is called a hybrid barbed strong bisimulation.

The relation $\mathcal{H}$ is a hybrid barbed strong bisimulation.

Proof: In order to close each of the four diagrams for $\mathcal{H}$, and also to establish the correspondence of barbs, we glue the elementary diagrams (or tiles) of lemmas B.20, B.21, B.22, B.23, B.26, and B.29. Since most of these diagrams introduce further horizontal and vertical steps, the existence of a finite tiling is unclear. Technically, we apply a general confluence technique relying on *decreasing diagrams*, proposed by Van Oostrom [17] and adapted to bisimulation proofs in [9]:

For each target diagram, we identify "horizontal" and "vertical" relations that may appear in the tiles, and we strictly order these relations. Then, we check that we have a tile for every product of an horizontal and a vertical relation that is decreasing for the strict ordering on relations. To conclude, we apply the decreasing diagram theorem.

To prove diagrams of theorem B.31 for all $\mapsto_x \in X$, and the preservation of weak barbs we use the sets of relations:

$$H = \{\Rrightarrow_{\mathcal{E}}, \xrightarrow{tr}, \Leftarrow_d, \Rrightarrow_d, \xleftarrow{d}, \xrightarrow{d}, \xleftarrow{\kappa}, \xleftarrow{Fwd}, \xleftarrow{0}, \xleftarrow{go}, \xleftarrow{\preceq_\kappa}, \xleftarrow{\preceq_{Fwd}}, \xleftarrow{\preceq_{stale}}\}$$

$$V = B \cup X \cup \{\downarrow_{\equiv_{\mathcal{E}}}, \downarrow_{\Rrightarrow_d}, \downarrow_b\}$$

For each diagram we strictly order these relations, and check that every divergence of an horizontal and a vertical relation can be closed as a decreasing tile for the chosen order.

Formally, we thus have $2 * 13 * 15$ tiles to check. All the relevant diagrams of the previous lemmas are decreasing with regards to this order.

(diagrams to be typeset)

□

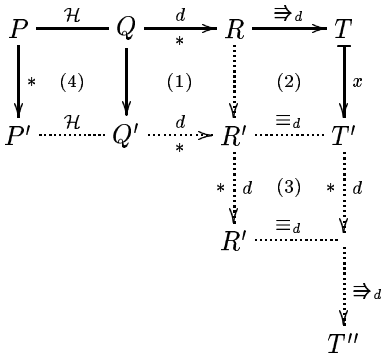
Lemma B.32 *$\mathcal{H}$ is a weak barbed bisimulation.*

Proof: We first prove that if $P \twoheadrightarrow_d \twoheadleftarrow_d Q$ and $P \to P'$ then $Q \to^* Q'$ and $P' \twoheadrightarrow_d \twoheadleftarrow_d Q'$. If the step $P \to P'$ is deterministic, then we have $P' \twoheadrightarrow_d \twoheadleftarrow_d Q$ by commutation of deterministic steps and by lemma B.11. Otherwise, by lemma B.16, there is a $P'' \in \mathcal{T}$ such that $P' \twoheadrightarrow_d P''$. Hence, by commutation of deterministic steps, and by lemma B.11, we have $Q \to_d^* \to Q'$ and $Q' \equiv_d P''$. Since we have $P'' \in \mathcal{T}$, we have $P'' \Leftarrow_d Q'$.

We now prove the first bisimulation diagram. Let $P, P', R, R' \in \mathcal{E}$ be well formed processes, and $Q \in \mathcal{J}$, such that $P \equiv_{\mathcal{E}} R \mapsto_x R' \equiv_{\mathcal{E}} P'$ and $P \mathcal{H} Q$. Then, by theorem B.31, we have $R \mathcal{H} Q$. By the same theorem, there is a series of reductions $Q(\twoheadrightarrow_d \mapsto_B)^* \twoheadrightarrow_d \mapsto_x Q'$ with $R' \mathcal{H} Q'$, hence $P' \mathcal{H} Q'$. Since we have $Q \twoheadrightarrow_d \twoheadleftarrow_d Q$, by the argument above, we have $Q \to^* S'$ with $Q' \twoheadrightarrow_d \twoheadleftarrow_d S'$, hence we have $P' \mathcal{H} S'$.

We now prove the second bisimulation diagram. Let $P \in \mathcal{E}$ be a well formed process, and $Q, Q' \in \mathcal{J}$ such that $P \mathcal{H} Q$ and $Q \to Q'$. We have the following

diagram:



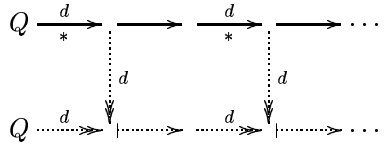
By lemmas B.14, B.16, and by definition of $\mathcal{H}$, there is a T such that $Q \twoheadrightarrow_d T$, and we have $P\mathcal{H}T$. Tile **1** holds by definition B.10 of deterministic steps, as they commute with other steps. If the considered step $Q \to R$ is a deterministic step, then by definition of $\mathcal{H}$, we have $P\mathcal{H}R$. Otherwise, there is a step $R \to R'$ to consider. In this case, this step corresponds to a step of $T \mapsto_x T'$ with $x \in B \cup X$. By lemma B.16, the existence of T'' is derived. Tile **2** holds by definition of $\Rrightarrow_d$, which is a relation included in the strong bisimulation $\equiv_d$. Tile **3** holds by bisimulation. Finally, tile **4** holds by applying theorem B.31, which results in $P'\mathcal{H}T'$. By definition of $\mathcal{H}$, we have $P'\mathcal{H}T''$, thus $P'\mathcal{H}R$.

The preservation of barbs is a direct consequence of their preservation in the relation $\mathcal{H}$. □

Lemma B.33 *The relation $\mathcal{H}$ preserves divergence.*

Proof: Let $P \in \mathcal{E}$ be a well-formed process and $Q \in \mathcal{J}$ such that $P\mathcal{H}Q$. If $P \Uparrow$, by theorem B.31, following the same argument than the proof of lemma B.32, we have $Q \Uparrow$.

Let us suppose there is an infinite series of steps starting with configuration Q. This series satisfies the following diagram:



After partitioning deterministic and non-deterministic steps in the original series, each tile is built using lemma B.16.

Since there cannot be an infinity of successive deterministic reductions, there is an infinity of tiles. Thus we have a divergence of the form: $Q \twoheadrightarrow_d \mapsto \twoheadrightarrow_d \mapsto \ldots$. By lemma B.17, there is no infinite series of normalized bookkeeping reductions $\twoheadrightarrow_d \mapsto_B$, thus there is an infinity of $\mapsto_X$ steps. By theorem B.31, we have a divergence for P. □

We finally combine our technical results to establish correctness, as stated in section 3.

Proof of Theorem 3.1: Let P be an ambient process. As regards weak- and fair-must barbs, we compose the equations of theorem 5.3 ($P \lesssim P$ with different operational semantics) and of theorem 5.4 ($P \approx^{aj} [\![P]\!]^t$).

We begin with a remark on fair-must barbs that applies to each of these equations. Let $\lesssim$ be at least a barbed coupled simulation between two families of processes equipped with reductions and barbs $\rightarrow, \downarrow_b$ and $\rightarrow', \downarrow'_b$. We write $\Downarrow'_b$ for $\rightarrow'^* \downarrow'_b$. Then, $\lesssim$ also respects fair-must barbs: if $P \lesssim Q$, $\Box P \Downarrow_b$, and $Q \rightarrow'^* Q'$, then $P \rightarrow^* P' \gtrsim Q'$ (using the simulation property of $P \gtrsim Q$) and $P' \rightarrow^* P'' \lesssim Q'$ (by coupling) with $P'' \Downarrow_b$ (by definition of $\Box P \rightarrow^* \downarrow b$), hence also $Q' \Downarrow'_b$ (using the weak barb property of $P'' \lesssim Q$) and finally $\Box Q \Downarrow'_b$.

For the notion of translated barb $\downarrow^4_b$ used in the statement of theorem 5.4, we thus obtain $P \Downarrow_b$ iff $[\![P]\!]^t \rightarrow^* \downarrow^4_b$, and $\Box P \Downarrow_b$ iff $\Box [\![P]\!]^t \rightarrow^* \downarrow^4_b$.

The statement of theorem 5.4 uses another variant of the top-level translation, so we still have to prove that $[\![P]\!]^t \rightarrow^* \downarrow^4_b$ iff $[\![P]\!]^b \Downarrow_{yes}$. The direct implication immediately follows from the definition of $\downarrow^4_b$. Conversely, $[\![P]\!]^b \Downarrow_{yes}$ is $C([\![P]\!]^t) \rightarrow^* T \downarrow_{yes}$. By induction, we prove that every step in the series of reductions leading to T is either one of the four steps expressing the barb $\downarrow_{yes}$, or a step that is independent for the context $C(\cdot)$ and commutes with each of these four steps.

The divergence property is established independently (since it is not generally preserved by the equations above). From $(P, \rightarrow)$ to $(P, \rightarrow_{12C})$, the extended semantics only refines some steps into pairs of steps, which does not affect divergence. From $(P, \rightarrow_{12C})$ to $[\![P]\!]^t$, we apply the strong bisimulation up-to bookkeeping of theorem B.31, in combination with the finite-bookkeeping lemma B.17: if $[\![P]\!]^t \rightarrow^n$ for all $n \geq 0$, then also $[\![P]\!]^t ((\rightarrow_B)^* \rightarrow_{1,2,c} (\rightarrow_B)^*)^m$ for all $m \geq 0$, hence by strong bisimulation $P \rightarrow^m_{12C}$. From $[\![P]\!]^b$ to $[\![P]\!]^t$, there is at most one extra step. $\Box$