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

April 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 into 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 translations into the distributed join calculus. In combination, these results imply that our translation preserves and reflects 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 are recalled, with an overview of both calculi.

*An extended abstract appears in IFIP TCS'2000.

Contents

1 Introduction **1**

2 From ambients to the join calculus **2**
2.1 An asynchronous algorithm 2
2.2 A simple translation 3
2.3 Dealing with other ambient constructs 6

3 Correctness of the translation **7**

4 A calculus of Ambients extended with transient states **8**

5 Coupled simulations and operational correspondence **11**
5.1 Correctness of the asynchronous algorithm. 12
5.2 Operational correspondence. 13

6 Distributed implementation **13**
6.1 The implementation in Jocaml 14
6.2 Controlling distribution at run-time 14

7 Conclusions **16**

A Two notions of mobile computations **18**
A.1 Operational semantics for ambients 18
A.2 Operational semantics for the join calculus 20
A.3 Overview for Ambients 20
A.4 Overview of the join calculus 22

B Proofs **24**
B.1 Correctness of the synchronization algorithm (theorem 2) 25
B.2 Correctness of the extended translation (theorem 3) 31

List of Figures

1 Translation from In/Out ambient processes to the join calculus . . . 4
2 Additional clauses for the full translation 5
3 Syntax for an ambient calculus extended with transient states 9
4 Semantics for an ambient calculus extended with transient states . . . 9
5 Syntax for the ambient calculus 19
6 Operational semantics for the ambient calculus 19
7 Syntax for the distributed join calculus 21
8 Operational semantics for the distributed join calculus 21

1 Introduction

We present a highly concurrent distributed implementation of the Cardelli-Gordons's calculus of Mobile Ambients [5] in Jocaml [14, 7]. 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 unrealistic 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 can be temporarily locked. 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 can control 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. 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 that is a model of Jocaml [11], 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 simulations for the correctness of the algorithm, then we use an hybrid barbed bisimulation for the actual translation into the join calculus. Technically, the first stage is a first application of coupled-simulations [19] 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 [18]. In combination, these results imply that the translation preserves and reflects 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 for these calculi can be found in [11] and in [5], respectively. In appendix B, we provide the proofs for section 5.

2 From ambients to the join calculus

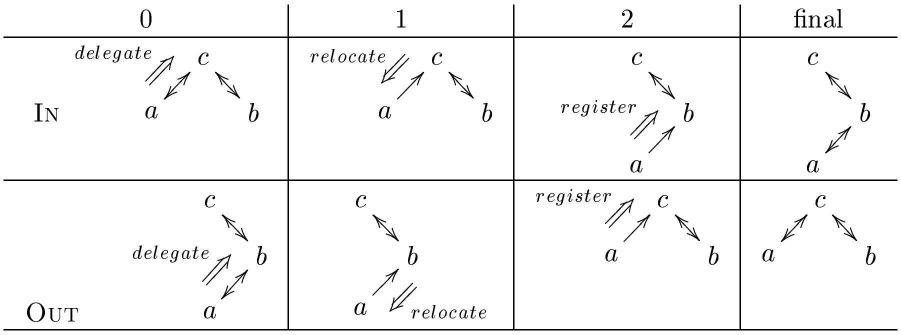
We describe the asynchronous algorithm, then we specify it as a translation from ambients to the join calculus. We begin with a fragment of the ambient calculus given by the grammar $P ::= \; a[\,P\,] \;\big|\; \mathtt{in}\; a.P \;\big|\; \mathtt{out}\; a.P \;\big|\; P \mid P' \;\big|\; \mathbf{0}$. 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 bears an ambient name; 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.

Moves of ambient a in and out of ambient 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., $c[\ a[\,\mathtt{in}\ b.Q\ ]\ \mid\ b[\,\mathbf{0}\,]\,] \to c[\ b[\ a[\,Q\,]\ ]\ ]$.

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 Q.

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, establishing the new downlink.

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 actions 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 the current 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 in the relocation message sent back to a.

2.2 A simple translation

The compositional translation $[\![\,\cdot\,]\!]_e$ appears in Figure 1. Overall, the tree of nested ambients is mapped to an isomorphic tree of nested locations. Each ambient is

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

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

$$
\begin{aligned}
D_0 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid in(b,\kappa)\ \ \triangleright s(a,i,e,l \cup \{\text{In } b\ \kappa\}) \mid e.sub_{in}(i,b,\kappa)\\
&\ \ \wedge\ \ s(a,i,e,l) \mid out(b,\kappa) \triangleright s(a,i,e,l \cup \{\text{Out } b\ \kappa\}) \mid e.sub_{out}(i,b,\kappa)\\
D_1 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid amb(k,c,e_c) \mid sub_{in}(k,b,\kappa) \triangleright\\
&\qquad s(a,i,e,l) \mid amb(j,b,e_b) \mid e_c.reloc(e_b,\kappa)\\
&\ \ \wedge\ \ s(a,i,e,l) \mid amb(j,b,e_b) \mid sub_{out}(j,a,\kappa) \triangleright s(a,i,e,l) \mid e_b.reloc(e,\kappa)\\
D_2 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid reloc(e',\kappa) \triangleright \texttt{go}(e'.here); (I_{a,e_h,e'} \mid \kappa() \mid Flush(l,in,out,\kappa))\\
D &\stackrel{\text{def}}{=} D_0 \wedge D_1 \wedge D_2
\end{aligned}
$$

$$
I_{a,e_h,e} \stackrel{\text{def}}{=} \texttt{def uid}\ i\ \texttt{in}\ s(a,i,e,\emptyset) \mid e.amb(i,a,e_h)
$$

$$
AM_{a,e}(P) \stackrel{\text{def}}{=} here\,\big[D\ :\ I_{a,e_h,e} \mid [\![P]\!]_{e_h}\big]
$$

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

mapped to a join calculus location containing the definition D 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 distributed 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 $c[\ a[\,\texttt{in}\ b.Q\ ]\ \mid\ b[\,\mathbf{0}\,]]$. 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 0-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 recorded in a's log as an entry In b κ, and the request is forwarded to the enclosing

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

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

$$
\begin{aligned}
D'_1 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid open(b,\kappa) \triangleright s(a,i,e,l) \mid e_b.opening(\kappa)\\
D'_2 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid opening(\kappa) \triangleright f(e) \mid \kappa() \mid Flush(l, e.in, e.out, \kappa)\\
D_C &\stackrel{\text{def}}{=} s(a,i,e,l) \mid recv(\kappa) \mid send(n) \triangleright s(a,i,e,l) \mid \kappa(n)\\
D_F &\stackrel{\text{def}}{=} f(e) \mid in(b,\kappa) \triangleright f(e) \mid e.in(b,\kappa)\\
&\wedge\ f(e) \mid out(b,\kappa) \triangleright f(e) \mid e.out(b,\kappa)\\
&\wedge\ f(e) \mid open(b,\kappa) \triangleright f(e) \mid e.open(b,\kappa)\\
&\wedge\ f(e) \mid amb(j,b,e_b) \triangleright f(e) \mid e.amb(j,b,e_b)\\
&\wedge\ f(e) \mid sub_{in}(k,b,\kappa) \triangleright f(e) \mid e.sub_{in}(k,b,\kappa)\\
&\wedge\ f(e) \mid sub_{out}(k,b,\kappa) \triangleright f(e) \mid e.sub_{out}(k,b,\kappa)\\
&\wedge\ f(e) \mid recv(\kappa) \triangleright f(e) \mid e.recv(\kappa)\\
&\wedge\ f(e) \mid send(b) \triangleright f(e) \mid e.send(b)
\end{aligned}
$$

$$
D \stackrel{\text{def}}{=} D_0 \wedge D_1 \wedge D'_1 \wedge D_2 \wedge D'_2 \wedge D_C \wedge D_F
$$

with the extended record notation

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

Figure 2: Additional clauses for the full translation

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 1-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 2-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 the activities of a with parent 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 tagged with i will never match a rule of D_1. In our implementation, these stale messages are actually discarded.) In addition, the message $\kappa()$ triggers 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 $out(d, \kappa')$ for every entry In $d\ \kappa'$ or Out $d\ \kappa'$ appearing in the log l and such that $\kappa' \neq \kappa$. These entries correspond to actions preempted by the migration; they will be delegated to a new parent through other iterations of 0-steps.

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 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*, *recv*, and *send*.

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 binders `fresh` a with the same scope in the join calculus.

Communication. Ambient communication is implemented by supplementing every ambient manager with a rule D_C that binds two channels *send* and *recv* and synchronizes message outputs and message requests. This encoding is much like the encoding of pi-calculus channels into the join calculus (see [9]).

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` capabilities. In contrast, join calculus names are statically attached 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 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 enclosing ambient. The indirection is achieved by using a persistent message sent on f defined in D_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.

	initial	1	2 & final
OPEN	$a \nearrow\!\!\!\swarrow b$	$a \nearrow b$, $b \xRightarrow{opening} a$	$(\cdot) \xRightarrow{forward} b$

3 Correctness of the translation

The distributed synchronization algorithm seems to depart from the operational semantics of ambients, and the translation of nested ambients yields arbitrarily large terms with numerous instances of the algorithm running in parallel. This makes the correctness of the distributed implementation problematic. Technically, both calculi have a reduction-based semantics, which can be equipped with standard notions of observation. 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 P indexed by names b, 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 $P \rightarrow^* P'\downarrow_b$.
- 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 $P \rightarrow^* P'$, we have $P'\Downarrow_b$.

In combination, these predicates give a precise content 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 new deadlocks, or even an empty translation. The preservation of convergence is of pragmatic importance. In addition, correctness for fair-must tests relates infinite computations [8], 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 translation at top-level with a definition D_t that reveals the presence of a particular barb $\downarrow_b$ in the source process. Using D and e_h as defined in figure 2, and for a given interface e, we define the top-level translation

$$\begin{aligned} [\![P]\!]^b &\stackrel{\text{def}}{=} here[D \wedge D_t \wedge \texttt{uid}\; i : s(a,i,e,\emptyset) \mid t(b) \mid [\![P]\!]_{e_h}] \\ D_t &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid t(b) \triangleright s(a,i,e,l) \mid amb(j,b,e_b) \mid yes() \end{aligned}$$

Without loss of generality, we always assume that the names in a, i, t, e, yes do not clash with any name free in P, and that the location and channel names introduced by 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 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 observations 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 1 appears at the end of appendix B.2.

4 A calculus of Ambients extended with transient states

In order to prove theorem 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 use *coupled simulations* [19] to relate the two semantics for ambients; then, we 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 that 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. 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

$$
\begin{array}{lll}
P & ::= & & \text{extended ambient process} \\
& & \dots & \quad \textit{all the constructors of Figure 5} \\
& | & Xn[P] & \quad \text{stub} \\
& | & i & \quad \text{scion} \\
& | & \nu i.P & \quad \text{marker restriction} \\
\\
X & ::= & & \text{state extension} \\
& & \bar{\imath}\{P\}\!- & \quad \text{the stub is committed to move to } i \\
& | & \mathtt{o}\{P\}\!- & \quad \text{the stub is being opened}
\end{array}
$$

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). In the following, we write $X^{=}n[P]$ for either $Xn[P]$ or $n[P]$.

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

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

$$
E(\cdot) ::= \;\cdot\; \;\big|\; P \mid E(\cdot) \;\big|\; E(\cdot) \mid P \;\big|\; X^{=}n[\,E(\cdot)\,] \;\big|\; \nu n.E(\cdot) \;\big|\; \nu i.E(\cdot)
$$

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:

$$
\text{R2X}\; \frac{m \neq n \qquad m \text{ is not free in } X}{Xn[\nu m.P] \equiv \nu m.Xn[P]}
$$

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

$$
\textsc{Stub}\quad \bar{\imath}\{Q\}\!-\!n[R] \xrightarrow{\bar{\imath}.n[Q|R]} \mathbf{0} \qquad\qquad \textsc{Scion}\quad i \xrightarrow{i.P} P
$$

Original 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:

$$
\textsc{In 1}\quad \begin{array}{ll} & m[P] \mid n[\mathtt{in}\, m.Q \mid R] \\ \rightarrow_1 & \nu i.\; m[i \mid P] \mid \bar{\imath}\{Q\}\!-\!n[R] \end{array} \qquad \textsc{Out 1}\quad \begin{array}{ll} & X^{=}m\big[P \mid n[\mathtt{out}\, m.Q \mid R]\big] \\ \rightarrow_1 & \nu i.\; i \mid X^{=}m\big[P \mid \bar{\imath}\{Q\}\!-\!n[R]\big] \end{array}
$$

$$
\textsc{Move 2}\; \frac{P \xrightarrow{\bar{\imath}.S} P' \qquad Q \xrightarrow{i.S} Q'}{\nu i.(P \mid Q) \rightarrow_2 P' \mid Q'}
$$

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

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

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

scions; completion steps $\to_2$ consume them.

$$\begin{tikzcd}
m[P] \mid n[\,\mathtt{in}\,m.Q \mid R\,] \arrow[rr, "\textsc{In}"] \arrow[dr, "\textsc{In 1}"'] & & m[\,n[\,Q \mid R\,] \mid P\,] \\
& \nu i.m[\,i \mid P\,] \mid \bar{\imath}\{Q\}\text{–}n[\,R\,] \arrow[ur, "\textsc{Move 2}"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
m[\,P \mid n[\,\mathtt{out}\,m.Q \mid R\,]\,] \arrow[rr, "\textsc{Out}"] \arrow[dr, "\textsc{Out 1}"'] & & n[\,Q \mid R\,] \mid m[\,P\,] \\
& \nu i.\ i \mid m[\,P \mid \bar{\imath}\{Q\}\text{–}n[\,R\,]\,] \arrow[ur, "\textsc{Move 2}"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
\mathtt{open}\,n.Q \mid n[\,R\,] \arrow[rr, "\textsc{Open}"] \arrow[dr, "\textsc{Open 1}"'] & & Q \mid R \\
& \mathtt{o}\{Q\}\text{–}n[\,R\,] \arrow[ur, "\textsc{Open 2}"'] &
\end{tikzcd}$$

The reduction rules for RECV and REPL are those of the original semantics, except that we write $\to_C$ instead of $\to$. Overall, we obtain a reduction system for extended ambient processes with steps $\to_{12C} \stackrel{\text{def}}{=} \to_1 \cup \to_2 \cup \to_C$. 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 predicate $\downarrow_b$ is insensitive to extended ambients $X b[\,Q\,]$.

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 name i. This may seem difficult to implement, but actually scions 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 ($P \to P_1$), or b exits a first, and b is left empty ($P \to\to P_3$). 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.

$$\begin{tikzcd}
& P \stackrel{\text{def}}{=} \mathtt{open}\ a \mid a[\ b[\,\mathtt{out}\ a\,]\,] \arrow[dl, "\textsc{Open}"'] \arrow[dr, "\textsc{Out}"] & \\
P_1 \stackrel{\text{def}}{=} b[\,\mathtt{out}\ a\,] & & P_2 \stackrel{\text{def}}{=} \mathtt{open}\ a \mid a[\,] \mid b[\,] \arrow[dl, "\textsc{Open}"] \\
& P_3 \stackrel{\text{def}}{=} b[\,] &
\end{tikzcd}$$

$$
\begin{tikzcd}[column sep=small]
 & & P \stackrel{\text{def}}{=} \mathtt{open}\ a \mid a[\ b[\ \mathtt{out}\ a\ ]\ ] \arrow[dll, "\textsc{Open 1}"'] \arrow[drr, "\textsc{Out 1}"] & & \\
Q_1 \stackrel{\text{def}}{=} \mathtt{o}\text{–}a[\ b[\ \mathtt{out}\ a\ ]\ ] \arrow[d, "\textsc{Open 2}"'] \arrow[dr, "\textsc{Out 1}"] & & & & Q_2 \stackrel{\text{def}}{=} \mathtt{open}\ a \mid \nu i.(\ a[\ \bar{\imath}\text{–}b[\ ]\ ] \mid i) \arrow[dll, "\textsc{Open 1}"'] \arrow[d, "\textsc{Move 2}"] \\
 & & Q \stackrel{\text{def}}{=} \nu i.(\ \mathtt{o}\text{–}a[\ \bar{\imath}\text{–}b[\ ]\ ] \mid i) \arrow[dd, ->>, "\substack{\textsc{Open 2}\\ \textsc{Move 2}}"'] & & \\
P_1 \stackrel{\text{def}}{=} b[\ \mathtt{out}\ a\ ] & & & & P_2 \stackrel{\text{def}}{=} \mathtt{open}\ a \mid a[\ ] \mid b[\ ] \arrow[dll, ->>, "\substack{\textsc{Open 1}\\ \textsc{Open 2}}"] \\
 & & P_3 \stackrel{\text{def}}{=} b[\ ] & &
\end{tikzcd}
$$

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 [8] 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 $\rightarrow_a$, $\rightarrow_b$ and families of observations $\Downarrow_{a,x}$ and $\Downarrow_{b,x}$.

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

1. *if $P \rightarrow_a^* P'$, then there exists Q' such that $Q \rightarrow_b^* Q'$ and $P' \mathrel{\mathcal{R}} Q'$;*
2. *for all x, if $P \Downarrow_{a,x}$, then $Q \Downarrow_{b,x}$.*

$\mathcal{R}$ 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.)

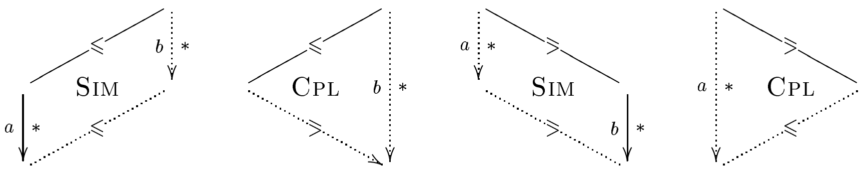
To address this issue of gradual commitment, Parrow and Sjödin proposed coarser relations called coupled simulations [19, 17]. We liberally adapt their definition to ambients:

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

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

In the initial definition of coupled simulations of [19], coupling conditions are required only for pairs of stable processes. As in [8], our definition is actually the barbed counterpart of the *weakly-coupled simulations* considered in [20].

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 coupled simulations 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 2 *Let $\lesseqgtr$ 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 $P \lesseqgtr P$.

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 ambient 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 any preceding step 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 2-step, and we conclude by showing that this 2-step 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 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 $[D \wedge D_t \wedge \mathtt{uid}\ i : s(a, i, e, \emptyset) \mid p(t) \mid [\![P]\!]_{e_0}]$. As before, we assume that names in a, i, 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 3 (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 3 is a corollary of theorem 4 in appendix B.2. 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 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, migrate, 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 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 proof is structured using the decreasing diagram technique of [18], 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 [12] for the source code, setup instructions, and programming 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 [5], as for instance in the ambient process $\langle \mathtt{in}\ a.\mathtt{out}\ b\rangle \mid !(x).x.\langle x\rangle$.
- Instead of a global translation, the implementation is an interpreter. Hence, the implementation translates guarded processes on the fly and maintains an environment for local variables. The interpreter also performs dynamic type checking whenever 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 caches some messages before they are processed: 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 cached. Similarly, when a new message arrives on amb, it is added to the cache, and the cache of deferred messages is scanned for enabled steps.
- The formal translation of replication always yields a diverging computation (as in the source ambient process). 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, the process $!\,a[\ ]$ does not diverge, while $!\,a[\,\mathtt{in}\ a\,]$ 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. We first describe those two interpretation, then we explain our choices in the implementation.

The join calculus has an “horizontal” model of distribution at run-time. Due to transparency, every pair of locations can interact as if they were simply running in parallel, regardless of their host sites. Every site executing a Jocaml runtime can be seen as a particular location and hosts a tree of sublocations. Dynamically, every migration 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 synchronization model of ambients involves nested ambients and vertical interactions. This suggests the use of ambients containing subambients running on remote machines. 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 condition 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 with 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 `$jam code` for the command that starts an interpreter running the ambient process `code`, and using integers as site identifiers, consider the computation initiated by the commands

`$jam "public a[0]"`	on site 2
`$jam "b[0]" -amb a`	on site 1
`$jam "c[ m[out c.in b] ]" -amb a`	on site 3

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:

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

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 manifests 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, an *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 never appear in a given ambient can lead to more efficient, specialized ambient managers.

Finally, little is known about actual programming with Ambients, 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 accommodate the co-capabilities proposed in [15].

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*, volume 1644 of *LNCS*, pages 230–239, 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] S. Conchon and F. Le Fessant. Jocaml: Mobile agents for objective-caml. In *ASA/MA'99*, pages 22–29. IEEE Computer Society, Oct. 1999.

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

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

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

[11] 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.

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

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

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

[15] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL'00*, pages 352–364. ACM, Jan. 2000.

[16] 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.

[17] 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.

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

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

[20] 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 [9, 11] and the ambient calculus [5]. These two models consist of name-passing process calculi in the spirit of the pi calculus [16]. 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 [14, 7].

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.

We define our syntax and semantics for the calculus of Mobile Ambients and for the distributed join calculus, then we give overviews of the two calculi.

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. In spirit, it is actually closer to the harness semantics of [13]. Our structural equivalence is more restrictive; it does not introduce or remove νx binders; it operates only in evaluation contexts. Our operational semantics 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.)

$$
\begin{array}{lll}
P & ::= & & \text{ambient process} \\
& & n[\,P\,] & \quad\text{ambient} \\
& | & P \mid P' & \quad\text{parallel composition} \\
& | & C.P & \quad\text{guarded process} \\
& | & \nu n.P & \quad\text{name restriction} \\
& | & \langle n \rangle & \quad\text{asynchronous message} \\
& | & (x).P & \quad\text{message reception} \\
& | & !P & \quad\text{replication} \\
& | & \mathbf{0} & \quad\text{inert process} \\
\\
C & ::= & & \text{capability} \\
& & \mathtt{in}\, n & \quad\text{ingoing migration} \\
& | & \mathtt{out}\, n & \quad\text{outgoing migration} \\
& | & \mathtt{open}\, n & \quad\text{ambient dissolution}
\end{array}
$$

Figure 5: Syntax for the ambient calculus

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

$$E(\cdot) ::= \cdot \quad\Big|\quad P \mid E(\cdot) \quad\Big|\quad E(\cdot) \mid P \quad\Big|\quad n[\,E(\cdot)\,] \quad\Big|\quad \nu n.E(\cdot)$$

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

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

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

$$
\begin{array}{lll}
\textsc{In} & & m[\,P\,] \mid n[\,\mathtt{in}\, m.Q \mid R\,] \\
& \rightarrow & m\big[\,P \mid n[\,Q \mid R\,]\,\big]
\end{array}
\qquad
\begin{array}{lll}
\textsc{Out} & & m\big[\,P \mid n[\,\mathtt{out}\, m.Q \mid R\,]\,\big] \\
& \rightarrow & m[\,P\,] \mid n[\,Q \mid R\,]
\end{array}
$$

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

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

Figure 6: Operational semantics for the ambient 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 except for the empty prefix (i.e. active locations form a tree whose nodes are indexed by location names), and such that every channel is defined in at most one location. In a configuration, a location with path $\alpha.a$ is *folded* 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 use a record notations $e = \{l_1 = x_1; \dots l_n = x_n\}$ as a shortcut for a tuple of names $x_1, \dots, x_n$ passed in a consistent order, and write $e.l_i$ for x_i. We use 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:

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

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]:

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

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:

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

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

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

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

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

Figure 8: Operational semantics for the distributed join calculus

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

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

$$\begin{array}{lll}
 & \nu w.\, w[\ k[\,\mathtt{out}\ w.\mathtt{in}\ k'.\mathtt{in}\ w\ ]\ \mid \mathtt{open}\ k'.P]\ \mid\ k'[\,\mathtt{open}\ k.Q\,] & (1)\\
\to & \nu w.\, w[\,\mathtt{open}\ k'.P\,]\ \mid\ k[\,\mathtt{in}\ k'.\mathtt{in}\ w\,]\ \mid\ k'[\,\mathtt{open}\ k.Q\,] & (2)\\
\to & \nu w.\, w[\,\mathtt{open}\ k'.P\,]\ \mid\ k'[\ k[\,\mathtt{in}\ w\ ]\ \mid \mathtt{open}\ k.Q] & (3)\\
\to & \nu w.\, w[\,\mathtt{open}\ k'.P\,]\ \mid\ k'[\,\mathtt{in}\ w \mid Q\,] & (4)\\
\to & \nu w.\, w[\,\mathtt{open}\ k'.P \mid k'[\,Q\,]\,] & \\
\to & \nu w.\, w[\,P \mid Q\,] & \\
\to & \nu w.\, w[\,P \mid (x).\mathit{Print}_x\ \mid\ \langle 4\rangle\,] & (5)\\
\to & \nu w.\, w[\,P \mid \mathit{Print}_4\,] & (6)
\end{array}$$

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 [9] is a variant of the asynchronous pi calculus [16, 1] with a distributed implementation and a uniform treatment of local and remote communication channels [14]. In this paper, we consider only the distributed join calculus without failures (named join calculus thereafter) [11, 8].

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 J express reception and synchronization of messages: the message 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 location 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.

$[\top$: def D_c in def u$[\top$: $P]$ in $\mathbf{0}]$

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

$[D_c$: def u$[\top$: $P]$ in $\mathbf{0}]$

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

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

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

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

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

$[D_c$ `:` $\mathbf{0}]$
$\wedge$ `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

$[D_c$ `:` $\mathbf{0}]$
$\wedge$ `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

$[D_c$ `:` $\mathbf{0}]$
$\wedge$ `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 [14].

B Proofs

We use the following set notations for processes: $\mathcal{J}$ is the set of join calculus processes; $\mathcal{A}$ and $\mathcal{E}$ are the sets of plain and extended ambient processes, respectively ($\mathcal{A} \subset \mathcal{E}$). We say that $P \in \mathcal{E}$ is *unrestricted* when it does not contain any restriction prefix νx in evaluation context.

B.1 Correctness of the synchronization algorithm (theorem 2)

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 2; in a second stage, we establish each of its requirements in a series of lemmas. Our statement of lemma B.1 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 2: $(P, \longrightarrow) \lesseqgtr (P, \rightarrow_{12C})$ for all $P \in \mathcal{A}$, it suffices to check that:*

1. *For all $P \in \mathcal{A}$ and $Q \in \mathcal{E}$, if $P \rightarrow_{12C}^{*} Q$, there is $P' \in \mathcal{A}$ such that $Q \rightarrow_{2}^{*} P'$.*
2. *For all $P, P' \in \mathcal{A}$, we have $P \longrightarrow^{*} P'$ iff $P \rightarrow_{12C}^{*} P'$.*
3. *For all $Q, Q' \in \mathcal{E}$, 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 $\geqslant = \{(P, Q) \mid P \rightarrow_{12C}^{*} Q\}$ and $\leqslant = \{(P, Q) \mid Q \rightarrow_{2}^{*} P\}$. We prove that $(\leqslant, \leqslant)$ are coupled simulations that preserve weak barbs and that are closed by application of evaluation contexts, and thus obtain $\geqslant \cap \leqslant \subseteq \lesseqgtr$. 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 below 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 substitutes 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 $\bar{\imath}\{Q\}$–$n[i]$ (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 $(P, Q) \in \mathcal{A} \times \mathcal{E}$ be processes such that $P \rightarrow_{12C}^{*} Q$. For some names $\widetilde{x}$ and process $R \in \mathcal{E}$, we have:*

1. *$Q \equiv \nu\widetilde{x}.R$ and R is unrestricted.*

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 $E(\circ\{P_o\}\text{-}n[\,R_o\,])$ 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 names $\widetilde{y}$ and extended evaluation context $E'(\cdot)(\cdot)$ whose holes are unrestricted 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 names $\widetilde{y}$ and process $R'_i \in \mathcal{E}$ 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 $R = E(\circ\{P_o\}\text{-}n[\,R_o\,])$:

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

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

   (c) *$Q' \equiv \nu\widetilde{y}.E(\circ\{P_o\}\text{-}n[\,R'_o\,])$ for some names $\widetilde{y}$ and process $R'_o \in \mathcal{E}$, with the same subcases as above (Internal step).*

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

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

   (f) *$Q' \equiv \nu\widetilde{x} \setminus j.F(\circ\{P_o\}\text{-}n[\,G(\,m[\,P_j \mid R_j\,])\,])(\mathbf{0})$ and beforehand we had $R = F(\circ\{P_o\}\text{-}n[\,G(j)\,])(\overline{\jmath}\{P_j\}\text{-}m[R_j])$ (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 of $\mathcal{A}$, there is no stub in R.

Assume that these properties hold for a derivation $P \rightarrow^n_{12C} Q$, 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

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

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

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

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

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

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). $\Box$

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 those 2-steps, we have the corollary:

Lemma B.3 (Completion) *For all $(P, Q) \in \mathcal{A} \times \mathcal{E}$ such that $P \rightarrow^*_{12C} Q$, there exists $P' \in \mathcal{A}$ such that $Q \rightarrow^*_2 P'$.*

We now compare successive $\rightarrow_1$ and $\rightarrow_2$ steps that operate on the same extension to the original $\longrightarrow$ step of the ambient calculus:

Lemma B.4 (Refinement) *For all $P, P' \in \mathcal{A}$, we have $P \longrightarrow P'$ if and only if either $P \rightarrow_1\rightarrow_2 P'$ or $P \rightarrow_C P'$.*

Proof: This is a direct consequence of lemma B.2 for series of two steps $P \rightarrow_1\rightarrow_2 P'$. 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 presented in section 4. $\Box$

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. (Formally, we may refine the extended semantics to produce and consume a $\nu i.$ binder and use extensions $\mathtt{o}(i)\{P\}$– instead of $\mathtt{o}\{P\}$–.)

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

Lemma B.5 *For all $(P, Q) \in \mathcal{A} \times \mathcal{E}$ such that $P \rightarrow^*_{12C} Q$, we have commutation properties:*

- *$Q \rightarrow^i_1 \rightarrow^j_1 Q'$ implies $Q \rightarrow^j_1 \rightarrow^i_1 Q'$ when the $\rightarrow^i_1$-step is not an* IN *1-step towards an ambient consumed by the following $\rightarrow^j_1$ step on the left-hand-side;*
- *$Q \rightarrow^i_1 \rightarrow^j_2 Q'$ implies $Q \rightarrow^j_2 \rightarrow^i_1 Q'$ when $i \neq j$ and the $\rightarrow^i_1$-step is not an* OUT *1-step from a stub consumed by the following $\rightarrow^j_2$-step on the left-hand-side;*
- *$Q \rightarrow^i_2 \rightarrow^j_1 Q'$ implies $Q \rightarrow^j_1 \rightarrow^i_2 Q'$ when the $\rightarrow^i_2$ step does not create a subterm used by the the following $\rightarrow^j_1$ step on the left-hand-side;*
- *$Q \rightarrow_2 \rightarrow_C Q'$ implies $Q \rightarrow_C \rightarrow_2 Q'$ when the $\rightarrow_C$-step does not use a subterm created by the $\rightarrow_2$-step on the left-hand-side;*
- *$Q \rightarrow_C \rightarrow_2 Q'$ implies $Q \rightarrow_2 \rightarrow_C Q'$;*
$Q \rightarrow_1 \rightarrow_C Q'$ implies $Q \rightarrow_C \rightarrow_1 Q'$;
$Q \rightarrow^i_2 \rightarrow^j_2 Q'$ implies $Q \rightarrow^j_2 \rightarrow^i_2 Q'$.

In particular, we always have $Q \rightarrow^i_1 \rightarrow^j_1 Q'$ implies $Q \rightarrow^j_1 \rightarrow^i_1 Q'$ when the second steps is internal with regards to i, and $Q \rightarrow^i_1 \rightarrow^j_2 Q'$ implies $Q \rightarrow^j_2 \rightarrow^i_1 Q'$ 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 $P \rightarrow^*_{12C} Q$, unfolding twice clause *(4.)* for the two steps being considered, and we check that these two steps also apply in the converse order.

For $\rightarrow^i_1 \rightarrow^j_1$, in the case $\rightarrow^i_1$ uses rule IN 1, this first step can we written

$$\nu\widetilde{x}.F(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) \quad \rightarrow^i_1 \quad \nu\widetilde{x}, i.F(m[Q_i \mid i])(\overline{\imath}\{P_i\}\text{–}n[\,R_i\,])$$

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 $E(\,\cdot\,)(\,\cdot\,) \rightarrow^j_1 \nu j.E'(\,\cdot\,)(\,\cdot\,)$. (Other cases are much simpler.) We further distinguish three subcases according to the position of the ambient being extended by $\rightarrow^j_1$: 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 $E(\,\cdot\,)(\,\cdot\,) \rightarrow_1 \nu j.E'(\,\cdot\,)(\,\cdot\,)$ can also be written more explicitly as $F(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,) \rightarrow_1 \nu j.F'(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,)$, and we also have

$$\begin{array}{rll} \nu\widetilde{x}.F(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) & \rightarrow^j_1 & \nu\widetilde{x}.\nu j F'(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) \\ & \rightarrow^i_1 & \nu\widetilde{x}.\nu j, i.F'(m[Q_i \mid i])(\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

In the second subcase, the step $E(\,\cdot\,)(\,\cdot\,) \rightarrow_1 \nu j.E'(\,\cdot\,)(\,\cdot\,)$ can also be written $F(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,) \rightarrow_1 \nu j.F(Q'_i \mid m[\,Q''_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,)$ in which Q'_i is either $\mathbf{0}$ for an internal step or j for an outgoing step, and we also have

$$\begin{array}{rll}\nu\widetilde{x}.F(m[Q_i])(n[\mathtt{in}\, m.P_i \mid R_i]) & \to_1^j & \nu\widetilde{x}.\nu j.F'(Q_i' \mid m[Q_i''])(n[\mathtt{in}\, m.P_i \mid R_i]) \\ & \to_1^i & \nu\widetilde{x}.\nu j, i.F'(Q_i' \mid m[Q_i'' \mid i])(\bar{\imath}\{P_i\}\text{–}n[R_i])\end{array}$$

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 any series of extended reductions between plain ambient processes correspond to a series of standard reductions.

Lemma B.6 (Correctness) *For all $P, P' \in \mathcal{A}$, if $P \to_{12C}^* P'$, then $P \to^* P'$.*

Proof: We establish the property above by induction on the length n of the series $P \to_{12C}^n P'$. 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 step is either $\to_C$ or $\to_1$ (not $\to_2$, because by hypothesis there is no stub in P).

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

If $P \to_1 Q \to_{12C}^* P'$, then the series of reductions contains a 2-step $\to_2$ that consumes this stub. Writing $\to_1^i$ and $\to_2^i$ to emphasize these matching steps, we can decompose the series into

$$P \to_1^i Q \to_{12C}^* Q' \to_2^i Q'' \to_{12C}^* P'$$

The stub created by $\to_1^i$ 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 $\to_2^i$, and we have $P \to_1^i Q \to_2^i R \to_{12C}^* P'$ for some extended process R. By Lemma B.4, we obtain that R is a regular process such that $P \to R$, and we conclude by induction hypothesis on $R \to_{12C}^* P'$.

2. There is at least one outgoing step. In this case, some of the steps between $\to_1^i$ and $\to_2^i$ may not commute with $\to_2^i$. We select another initial reduction as follows: assume $P \to_1^i Q\, (\to_{12C}^{ext})^* \to^j R \to_{12C}^* P'$ in which $(\to_{12C}^{ext})^*$ is a series of external steps and $\to^j$ is the first internal or outgoing step. This step $\to^j$ is either a $\to_C$ step or a $\to_1$ step. By Lemma B.5, this step commutes with every preceding external step, and also with the initial $\to_1^i$ step. We obtain $P \to^j \to_1^i (\to_{12C}^{ext})^* R \to_{12C}^* P'$, or simply $P \to^j \to_{12C}^* P'$.

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

   (b) Otherwise, we iterate the case analysis with $\to_1^j$ as initial reduction instead of $\to_1^i$. Since the nesting of ambients is preserved by all $\to_1$ steps, since every ambient process has a finite depth, and since case *2.* above always selects a new $\to_1$ 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 2: 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: 2-steps delete only stubs that do not contribute to the strong barbs of any process. $\square$

B.2 Correctness of the extended translation (theorem 3)

In a join calculus configuration obtained by translating an ambient process, there is a variety of reduction steps to consider; besides, the image of the translation is not closed by reduction. Thanks to our factorization of the proof, we circumscribe to this appendix the handling of these complications.

To found our case analyses on reductions in the translation, we first make explicit the structure of an approximation $\mathcal{T} \subset \mathcal{J}$ of the image of the translation that has simpler closure properties with regards to reduction (lemma B.14) and that essentially contains the image of the translation (lemma B.13). To achieve some operational correspondence, we then identify a few simplification relations on $\mathcal{T}$ that lead back to the image of the translation after reduction, and we study their commutation properties. To aggregate these properties in a systematic manner, we rely on the decreasing diagram technique of Van Oostrom [18, 10], which yields hybrid bisimulation diagrams between ambients and their translations. From these results, we finally derive our main theorems.

In the whole section, we abbreviate *reloc*, *opening*, *here* into r, o, h, respectively.

We begin with the definition of an auxiliary translation $[\![\,\cdot\,]\!]_e^\sharp$ that extends the translation of figure 2 to extended ambient constructs, and that implements global naming conventions. In order to avoid name clashes during structural rearrangements in join calculus configurations, we assign a distinct index to every ambient and extended ambient in evaluation context. We use n, m, p as indices and write D^n for the definition obtained from D in figure 2 by substituting h^n for *here*, r^n for *reloc*, o^n for *opening*, and x^n for any other name x defined in D. Using the same renaming, we write D_t^n for D_t and write R^n for R.

Definition B.7 (Auxiliary translation) *Let $[\![\,\cdot\,]\!]_e^\sharp$ be the function from indexed extended ambient processes $\mathcal{E}$ to join configurations $\mathcal{J}$ defined as:*

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

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

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

As in section 5, we define a top-level variant of the auxiliary translation. We use 0 as the top-level index. With the same notations (after renaming D and D_t to D^0 and D^0_t) we write $[\![\, \cdot \,]\!]^{t\sharp}$ for the translation that maps every extended ambient process P to the join configuration $h^0[D^0 \wedge D^0_t \wedge \mathtt{uid}\, i^0 : s^0(a, i^0, e, \emptyset) \mid p(t) \mid [\![P]\!]^{\sharp}_{e^0}]$, for names h^0, a, e, p, t, i^0, and yes that do not occur in P.

The definition above assumes that all scions are restricted; the translation of scions i and prefixes νi makes it clear that scions are used solely as passive targets for migration, represented as indices in the translation.

The next definition makes explicit a generalized shape for translations. Informally, the family $(\alpha_n)_{n \in N^+}$ accounts for the nested structure of ambients; the sets F and L gather free and local ambient names; the set I is a superset of the uids representing ambients; for every ambient with index n, the process S^n encodes the state of n, the messages M^n represent subprocesses running in n, the definition D^n is the ambient manager for n, and E^n defines all continuations allocated by n.

Definition B.8 (Translation state) *A join configuration P is a translation state, written $P \in \mathcal{T}$, when there is a set of indices N with a distinguished index $0 \in N$ such that*

1. *The structure of P is:*

$$\begin{array}{rcl} P & = & h^0[D^0 \wedge E^0 \wedge D^0_t \wedge \bigwedge_{a \in L} \mathtt{fresh}\, a \wedge \bigwedge_{i \in I} \mathtt{uid}\, i : p(t) \mid s^0(a, i^0, e, \emptyset) \mid M^0] \\ & \wedge & \displaystyle\bigwedge_{n \in N+} \alpha^n h^n[D^n \wedge E^n : S^n \mid M^n] \\ \\ E^n & = & \bigwedge_{\kappa \in K^n_a} \kappa() \triangleright [\![Q^\kappa]\!]_{e^n} \\ & \wedge & \bigwedge_{\kappa \in K^n_b} \kappa(a) \triangleright [\![Q^\kappa]\!]_{e^n} \\ & \wedge & \bigwedge_{\kappa \in K^n_c} \kappa() \triangleright [\![Q^\kappa]\!]_{e^n} \mid \kappa() \end{array}$$

   for some disjoint sets of names h^0, $\{a\}$, $fv(e)$, F, L, I, and $dv(D^n)$, K^n_a, K^n_b, K^n_c for each $n \in N$, for some pairwise-distinct names $i^n \in I$ indexed by $n \in N$, and some strings of names $\alpha^n \in \{h^n \mid n \in N^+\}^*$ indexed by $n \in N \setminus \{0\}$.

   We use notations $N^+ \stackrel{def}{=} N \setminus \{0\}$, $K^n_{ab} \stackrel{def}{=} K^n_a \cup K^n_b$, $K_{ab} \stackrel{def}{=} \bigcup_{n \in N} K^n_{ab}$, $K_c \stackrel{def}{=} \bigcup_{n \in N} K^n_c$, and $K \stackrel{def}{=} K_{ab} \cup K_c$.

2. *For every $n \in N$, M^n is a parallel composition of messages sent on names in $\{amb^n, in^n, out^n, open^n, recv^n, send^n, sub^n_{in}, sub^n_{out}, o^n, r^n\} \cup K^n$. M^0 does not contain messages on $\{in^0, out^0, o^0, n^0\}$. Every message $send^n(a)$ in M^n is such that $a \in F \cup L$.*

3. *For every $n \in N^+$, l^n is a set of entries of the form* $\mathrm{In}\ b\ \kappa$ *and* $\mathrm{Out}\ b\ \kappa$ *with $b \in F \cup L$ and $\kappa \in K_a$.*

4. *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 E^n, in stale messages, and at most once as described in 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* $\mathrm{In}\ b\ \kappa$ *within* l^m *(or* $sub^p_{out}(i^m, b, \kappa)$ *in* M^p *and* $\mathrm{Out}\ b\ \kappa$ *within* l^m*) with* $b \in F \cup L$.

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

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

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

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

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

5. *Each name* $\kappa \in K_b$ *occurs in its definition* E^n *and either in one message* $\kappa(a)$ *with* $a \in F \cup L$ *or in one message* $recv^p(\kappa)$ *with* $p \in N$

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

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

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

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

      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 no message* $r^n(\_, \_)$, $o^n(\_)$, *or* $amb^p(i^n, \_, \_)$ *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* $S^n = \mathsf{go}(h^p); I_{b,e^n,e^p} \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa)$. *We say that* n *is in a transient state.*

In the following, we use constrained notions of structural equivalence and reduction steps. For $P, Q \in \mathcal{J}$, we write $P \Rrightarrow Q$ when P and Q are structurally equivalent using rules P0–P2, D0–D2, using rule TREE only for removing syntactically nested locations in P, using α-conversion *only on names bound by a* `def` in P, and using rule SCOPE only for removing `def` binders in P. We write $P \ggg Q$ when $P \Rrightarrow Q$ and when Q has neither `def` binder nor syntactically-nested locations in evaluation contexts.

For a given family of reductions $\rightarrow_z$, we write $P \mapsto_z Q$ when $P \rightarrow_z Q' \Rrightarrow Q$ and when the step $P \rightarrow_z Q'$ uses structural equivalence only before the reduction rule, and only with the structural rules P0–P2, D0–D2 (excluding α-conversion). Accordingly, we sometimes identify translation states that are structurally equivalent using only rules P0–P2 and D0–D2. Assume $P \in \mathcal{T}$ and $P \rightarrow Q$. The structure of P as described by conditions 1, 2, and 7 of definition B.8 implies that for some R we have $P \mapsto R \equiv Q$.

We write $P \rightarrow_d P'$ when $P \rightarrow P'$ and this reduction uses either rule COMM or rule FLUSH.

For all $P, Q \in \mathcal{J}$, we write $P =_d Q$ when Q is obtained from P by moving definitions `fresh` a and `uid` i from one top-level location to another. We also let $\Rrightarrow_d \stackrel{\text{def}}{=} =_d \cap\, \mathcal{J} \times \mathcal{T}$ and $\twoheadrightarrow_d \stackrel{\text{def}}{=} \ggg \mapsto^*_d \Rrightarrow_d$.

The next lemmas gather simple properties on those relations. We omit their proofs.

Lemma B.9 (Deterministic steps) *We have*

1. *Deterministic steps commute with every join step, and strongly normalize.*
2. *If $P \in \mathcal{T}$, then $P \not\rightarrow_d$.*
3. $=_d\mapsto_d \subseteq \mapsto_d=_d$ *and* $=_d\mapsto \subseteq \mapsto=_d$.
4. $=_d\twoheadrightarrow_d \subseteq \twoheadrightarrow_d$, $\Rrightarrow\twoheadrightarrow_d \subseteq \twoheadrightarrow_d$, *and* $\mapsto_d\twoheadrightarrow_d \subseteq \twoheadrightarrow_d$.
5. $\Lleftarrow\rightarrow \subseteq \mapsto\equiv$.

Lemma B.10 *For all $P \in \mathcal{A}$, $[\![P]\!]_e \Rrightarrow [\![P]\!]_e^\sharp$. For $Q, Q' \in \mathcal{E}$, $P \equiv Q$ implies $[\![P]\!]_e^\sharp \equiv [\![Q]\!]_e^\sharp$ and $[\![P\{^a/_b\}]\!]_e^\sharp = [\![P]\!]_e^\sharp\{^a/_b\}$.*

Let us partition the steps $\mapsto$ that are enabled in a translation state. The analysis is by case on the reduction rule being used, with subcases for JOIN-steps. For each step, we suggest a correspondence with an extended ambient step, if any.

COMM, FLUSH By lemma B.9(2) such steps are disabled.

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

JOIN According to definition B.8, the step uses a reaction rule of D^n, E^n, or D_t^0, as defined in figure 2:

- D_0^n We call these steps *0-steps*, written $\mapsto_0$. Such steps correspond to preliminary steps of the algorithm of section 2.1, which are not detailed in the extended ambient calculus.
- D_1^n We call these steps *1-steps* and partition them further into $\mapsto_{\text{IN 1}}$, $\mapsto_{\text{OUT 1}}$, and $\mapsto_{\text{OPEN 1}}$ 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 *2-steps* and partition them further into $\mapsto_{\text{MOVE 2}}$ and $\mapsto_{\text{OPEN 2}}$ 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 $\mapsto_{\text{RECV}}$. 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. If $\kappa \in K_{ab}^n$, the step is a *continuation step*, written $\mapsto_\kappa$. It does not correspond to an ambient step. If $\kappa \in K_c^n$, the step is a *replication step*, written $\mapsto_{\text{REPL}}$. It corresponds to a REPL ambient step.
- D_t^0 By definition B.8, there is no message on t, hence this rule cannot be used.

For each family of steps $\mapsto_z$ above, we let $\twoheadrightarrow_z \stackrel{\text{def}}{=} \mapsto_z\twoheadrightarrow_d$ and let $\twoheadrightarrow_z^= \stackrel{\text{def}}{=} \twoheadrightarrow_z \cup \twoheadrightarrow_d$. Since deterministic normalization $\twoheadrightarrow_d$ is confluent, we partition reductions $\twoheadrightarrow$ using two families of relations: X gathers all steps associated with ambient steps; B gathers all other steps, which we name bookkeeping steps.

$$
\begin{array}{rcl}
X & \stackrel{\text{def}}{=} & \{\twoheadrightarrow_{\text{In } 1}, \twoheadrightarrow_{\text{Out } 1}, \twoheadrightarrow_{\text{Open } 1}, \twoheadrightarrow_{\text{Move } 2}, \twoheadrightarrow_{\text{Open } 2}, \twoheadrightarrow_{\text{Recv}}, \twoheadrightarrow_{\text{Repl}}\} \\
B & \stackrel{\text{def}}{=} & \{\twoheadrightarrow_0, \twoheadrightarrow_{Fwd}, \twoheadrightarrow_\kappa, \twoheadrightarrow_{Go}\}
\end{array}
$$

Next, we show that auxiliary translations $[\![Q]\!]^{t\sharp}$ actually reduce to translation states in a compositional manner, and that translations states are closed by reductions $\twoheadrightarrow$. We need a notion of *extension* between similar translation states.

Definition B.11 (State extension) *Let $P, Q \in \mathcal{T}$ with index sets $N_P \subseteq N_Q$. Let $m \in N_P$. We say that Q* extends *P in m when:*

1. *$F_P \subseteq F_Q$; $L_P \subseteq L_Q$; 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 $n \in N_P \setminus \{m\}$, $M_P^n = M_Q^n$ and $E_P^n = E_Q^n$;*
4. *$M_P^m \mid M'^m_P \equiv M_Q^m$; $K_P^m \subseteq K_Q^m$ with identical processes Q^κ for all $\kappa \in K_P^m$;*
5. *for all $q \in N_Q \setminus N_P$, $\alpha^m h^m$ is a prefix of $\alpha^q h^q$.*

Lemma B.12 (Ambient translation) *Let $Q \in \mathcal{E}$ and $P \in \mathcal{T}$ such that the names bound in $[\![Q]\!]^\sharp_{e^m}$ do not appear in P. Let $m \in N_P$ and $C(\cdot)$ be the context obtained from P by substituting $(\cdot \mid M^m)$ for M^m.*

There is an extension P' of P in m such that $C([\![Q]\!]^\sharp_{e^m}) \twoheadrightarrow_d P'$.

Proof: We establish the lemma by structural induction on Q for all P. By hypothesis, names bound in $[\![Q]\!]^\sharp_{e_m}$ and its subterms don't clash with any name of P; we also assume that they are pairwise disjoint.

$Q = \mathbf{0}$. Since $C(\mathbf{0}) \Rrightarrow P$ and state extension is reflexive, the statement holds for $P' = P$.

$Q = \nu i.Q'$. $[\![\nu i.Q']\!]^\sharp_e \stackrel{\text{def}}{=} [\![Q']\!]^\sharp_e$ and i does not appear in P, the statement holds by induction hypothesis.

$Q = \nu a.Q'$. Since a does not occur in P,

$$C([\![\nu a.Q']\!]^\sharp_{e^m}) \stackrel{\text{def}}{=} C(\texttt{def fresh}\ a\ \texttt{in}\ [\![Q']\!]^\sharp_{e^m}) \Rrightarrow C'_m([\![Q']\!]^\sharp_{e^m}) =_d C'_0([\![Q']\!]^\sharp_{e_m})$$

where $C'_m(\cdot)$ and $C'_0(\cdot)$ are obtained from $C(\cdot)$ by adding the binder $\texttt{fresh}\ a$ to the definitions of h^m and h^0, respectively. The process $P'_0 = C'_0(\mathbf{0})$ is a translation state that extends P with $L \uplus \{a\}$ instead of L. By induction hypothesis on P'_0 and Q', we obtain $C'_0([\![Q']\!]^\sharp_{e^m}) \twoheadrightarrow_d P'$ for some P' that extends P'_0, and also extends P. By lemma B.9(2), $C([\![Q]\!]^\sharp_{e^m}) =_d \twoheadrightarrow_d P'$ implies $C([\![Q]\!]^\sharp_{e^m}) \twoheadrightarrow_d P'$.

$Q = Q' \mid Q''$**.** By induction hypothesis on P and Q', we have $C(\llbracket Q' \rrbracket^{\sharp}_{e^m}) \twoheadrightarrow_d P'_1$ for some $P'_1 \in \mathcal{T}$ extending P in m. Let $C'(\cdot)$ be the extension context obtained from P'_1 for the same index m. We apply the same series of steps and equivalences to the process $C(\llbracket Q \rrbracket^{\sharp}_{e^m})$:

$$C(\llbracket Q \rrbracket^{\sharp}_{e^m}) \stackrel{\text{def}}{=} C(\llbracket Q' \rrbracket^{\sharp}_{e^m} \mid \llbracket Q'' \rrbracket^{\sharp}_{e^m}) \Rrightarrow \mapsto^*_d =_d C'(\llbracket Q'' \rrbracket^{\sharp}_{e^m})$$

By induction hypothesis on P'_1 and Q'', we have $P'_1 = C'(\llbracket Q'' \rrbracket^{\sharp}_{e^m}) \twoheadrightarrow_d P'$ for $P' \in \mathcal{T}$ extending P'_1 in m. Since successive extensions for the same index are transitive, P' also extends P in m.

$Q = !Q'$**.** By definition of the translation, and for some fresh name κ, we have:

$$C(\llbracket !Q' \rrbracket^{\sharp}_{e^m}) \stackrel{\text{def}}{=} C(\texttt{def}\ \kappa \triangleright \llbracket Q' \rrbracket_{e^m} \mid \kappa()\ \texttt{in}\ \kappa()) \Rrightarrow P'$$

where P' is an extension of P obtained by taking $Q^{\kappa} = Q'$, substituting $K^m_c \uplus \{\kappa\}$ for K^m_c, and substituting $\kappa() \mid M^m$ for M^m. We easily check that conditions 6 for κ and condition 2 for m hold in P', and that P' extends P in m.

$Q = (x).Q'$**,** $\texttt{open}\ a.Q'$**,** $\texttt{out}\ a.Q'$**, or** $\texttt{in}\ a.Q'$**.** We apply the same argument as above, checking condition 2 for the new message on $recv^m$, $open^m$, out^m, or in^m in M^m, and checking condition 5 or 4 for the new continuation in K^m_b or K^m_a.

$Q = \langle a \rangle$**.** By definition $C(\llbracket \langle a \rangle \rrbracket^{\sharp}_{e^m}) \stackrel{\text{def}}{=} C(send^m(a))$, so we extend P with the message $send^m(a)$ in M^m and the name a in F if $a \notin L \cup F$.

$Q = a[\,Q'\,]$**.** For some $n \notin N_P$, let $\alpha^n = \alpha^m h^m$ and $S^n = s^n(a, i^n, e^m, \emptyset)$. We have

$$\begin{array}{rll}
& C(\llbracket\, a[\,Q'\,]\,\rrbracket^{\sharp}_{e^m}) & \\
\stackrel{\text{def}}{=} & C(\texttt{def}\ h^n[D^n \wedge \texttt{uid}\ i^n : S^n \mid \llbracket Q' \rrbracket^{\sharp}_{e^n} \mid amb^m(i^n, a, e^n)]\ \texttt{in}\ \mathbf{0}) & \\
\Rrightarrow & C(\mathbf{0}) \wedge \alpha^n h^n[D^n \wedge \texttt{uid}\ i^n : S^n \mid \llbracket Q' \rrbracket^{\sharp}_{e^n} \mid amb^m(i^n, a, e^n)] & (1) \\
=_d & C'(\mathbf{0}) \wedge \alpha^n h^n[D^n : S^n \mid \llbracket Q' \rrbracket^{\sharp}_{e^n} \mid amb^m(i^n, a, e^n)] & (2) \\
\mapsto_d & C'(amb^m(i^n, a, e^n)) \wedge \alpha^n h^n[D^n : S^n \mid \llbracket Q' \rrbracket^{\sharp}_{e^n}] & (3) \\
\stackrel{\text{def}}{=} & C''(\llbracket Q' \rrbracket^{\sharp}_{e^n}) & (4)
\end{array}$$

where $C'(\mathbf{0})$ is $C(\mathbf{0})$ with a fresh name i^n added to I. (1) lifts the $\texttt{def}$ binder; no α-conversion is required. (2) moves $\texttt{uid}\ i^n$ from h^n to h^0; (3) is a COMM step transporting message $amb^m(i^n, a, e^n)$ to M^m; (4) defines the context $C''(\,\cdot\,)$.

The process $R = C''(\mathbf{0})$ extends P in m as follows (omitting identical bindings):

- $N_R = N \uplus \{n\}$;
- $I_R = I \uplus \{i^n\}$;
- if $a \in L$ then $F_R = F$ else $F_R = F \cup \{a\}$;
- $M^m_R = M^m \mid amb^m(i^n, a, e^n)$
- $\alpha^n_R = \alpha^n$ and $S^n_R = S^n$ as defined above; $M^n_R = \mathbf{0}$; $l^n_R = \emptyset$; $K^n = \emptyset$.

In particular, R is a translation state as defined in B.8. For instance, condition 7(a)i for n holds with $p = m$ because $a \in F_R \cup L$, $\alpha^n = \alpha^m h^m$, there is a message $amb^m(i^n, a, e^n)$, and there is no other message on r^n, o^n, or any message with identifier i^n (since these three names are fresh.)

By induction hypothesis on R and Q', using index n, we have $C''([\![Q']\!]^{\sharp}_{e^n}) \twoheadrightarrow_d P'$ for some $P' \in \mathcal{T}$ that extends R in n. Actually, $N_P \subseteq N_{R'}$ and P' also extends P in m:

1. $F_P \subseteq F_R \subseteq F_{P'}$; $L_P = L_{R'} \subseteq L_{P'}$, 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 $p \in N_P \setminus \{m\}$, $M_P^p = M_{R'}^P = M_{P'}^p$ and $E_P^p = E_{R'}^p = E_{P'}^p$, since $n \notin N_P$;
4. we have $M_P^m \mid M_P'^m = M_{R'}^m = M_{P'}^m$ and $K_{P'}^m = K_{P'}^m$;
5. 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 $\alpha'^q h^q$.

$Q = \bar{\imath}\{Q'\}$–$a[Q'']$, (or $Q = \circ\{Q'\}$–$a[Q'']$). These cases are handled much as $Q = a[Q']$ above, with the following differences: there is no message on amb^m, hence no need for a COMM step; instead, we have $K_a^n = \{\kappa\}$ for some fresh name κ, and $M^n = r^n(e^p, \kappa)$ with $p \in N$ (or $M^n = o^n(\kappa)$). We end up with a process P' that meets conditions 4d and 7(a)ii for κ and n (or conditions 4f and 7(a)iii for κ and n). ◻

Corollary B.13 *For all $Q \in \mathcal{E}$, there exists $P \in \mathcal{T}$ such that $[\![Q]\!]^{t\sharp} \twoheadrightarrow_d P$.*

Proof: We check $[\![\mathbf{0}]\!]^{t\sharp} \in \mathcal{T}$, then apply lemma B.12 to extend $[\![\mathbf{0}]\!]^{t\sharp}$ with the translation of Q at index 0. ◻

Lemma B.14 (Translation invariant) *For all $(P, P') \in \mathcal{T} \times \mathcal{J}$ such that $P \mapsto P'$, there is $Q \in \mathcal{T}$ such that $P' \twoheadrightarrow_d Q$.*

Proof: Let $P \in \mathcal{T}$ and $P \mapsto P'$. Relying on the partition of steps in translation states, we proceed by case on the steps $P \mapsto P'$: 1-steps, 2-steps, communication steps, replication steps, Go-steps, continuation steps, 0-steps and forwarding steps.

Go-step: by definition B.8, $P \in \mathcal{T}$ meets condition 7(b)ii for some n such that S^n is the form $\mathsf{go}(h^p); (I^n_{b,e^n,e^p} \mid \kappa() \mid F^n)$ with $F^n \stackrel{\text{def}}{=} Flush(l^n, in^n, out^n, \kappa)$ and such that

$$P \;=\; C_{\alpha^n}(S^n) \;\mapsto_{Go}\; C_{\alpha^p h^p}(I^n_{b,e^n,e^p} \mid \kappa() \mid F^n) \;\Rrightarrow\!\!\!\!\!/\;\; P'$$

for some parameterized context $C_\alpha(\,\cdot\,) \stackrel{\text{def}}{=} T \wedge \alpha h^n[D^n \wedge E^n : \;\cdot\; \mid M^n]$. By hypothesis, i is fresh in $C_\alpha(\,\cdot\,)$. Let G^n such that $F^n \to G^n$ using rule FLUSH, and let $C'_\alpha(\,\cdot\,)$ be $C_\alpha(\,\cdot\,)$ with $I \uplus \{i\}$ instead of I. Resuming from the equation above, we have

$$\begin{aligned}
P' \quad &=_d \quad C'_{\alpha^p h^p}(s^n(b, i, e^p, \emptyset) \mid amb^p(i, b, e^n) \mid \kappa() \mid F^n) && (1)\\
&\mapsto_d \quad C'_{\alpha^p h^p}(s^n(b, i, e^p, \emptyset) \mid amb^p(i, b, e^n) \mid \kappa() \mid G^n) && (2)\\
&\mapsto_d \mapsto_d^{=} \quad Q && (3)
\end{aligned}$$

where (1) lifts the binder `uid` i from I^n_{b,e^n,e^p} to the root location, where (2) is a FLUSH step reducing F^n, and where (3) is a COMM step moving message $s^n(b,i,e^p,\emptyset)$ to M^p followed by a COMM step moving $\kappa()$ to M^q in the case $\kappa \in K^q_a$ with $q \neq n$.

We check that $Q \in \mathcal{T}$. Most conditions of definition B.8 for Q hold as in P. The differences between P and Q are:

1. $\alpha^n_Q = \alpha^p h^p$; $\alpha^m_Q = \alpha^m$ if h^n does not occur in α^m; $\alpha^m_Q = \alpha^p h^p h^n \beta$ when α^m is of the form $\alpha^n h^n \beta$ (otherwise h^n does not occur in α^m and $\alpha^m_Q = \alpha^m$).
2. $S^n_Q = s^n(b,i,e^p,\emptyset)$; $I_Q = I \uplus \{i\}$;
3. $M^p_Q = M^p \mid amb^p(i,b,e^n)$;
   $M^n_Q = M^n \mid G^n$;
   $M^q_Q = M_Q \mid \kappa()$ (assuming $\kappa \in K^q_a$ and $q \neq n$).

We detail how the conditions of definition B.8 are affected by these changes.

1. holds for the bindings above, with $i^n_Q = i$.

2. holds for all $m \in N$, with simple checks for M^n_Q, M^p_Q, and M^q_Q where $\kappa \in K^q_a$.

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

4. The new continuation message κ is unique by condition 7(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. By condition 4 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 $\kappa^s \in l^n$. By condition 7 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. By condition 7(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 4. 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, $open^r$, $recv^r$ or o^r message in P (because of condition 4). 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 4 is true for P'.

5. and ***6.*** hold unchanged.

7. holds because $S^n = s^n(b,i',e^p,\emptyset)$ and h^p is the enclosing location of h^n. By condition 7(b)ii for P, there were neither $amb^r(i^n,\_,e^n)$ message nor any r^n nor o^n message in P. However, a message $amb^p(i,b,e^n)$ appears in Q, so condition 7(a)i holds with $\beta = \epsilon$ being the empty word.

Continuation step, Replication step By definition, the message being consumed appears in some M^n with $n \in N$, on a name $\kappa \in K^n_{abc}$. We detail the case $\kappa \in K^n_a$, then discuss cases $\kappa \in K^n_b$ and $\kappa \in K^n_c$. Let $C(\cdot)$ be the context obtained from P by substituting $(\cdot)$ for $\kappa()$ in M^n, and assume that the rule defining $\kappa()$ in E^n is $\kappa() \triangleright [\![Q^\kappa]\!]_{e^n}$. We choose fresh indices and names for an auxiliary translation of Q^κ. Since $C([\![Q^\kappa]\!]_{e^n}) \Rrightarrow C([\![Q^\kappa]\!]^\sharp_{e^n})$ by lemma B.10, we have

$$P \stackrel{\text{def}}{=} C(\kappa()) \mapsto C([\![Q^\kappa]\!]_{e^n}) \Rrightarrow C([\![Q^\kappa]\!]^\sharp_{e^n})$$

We have $C(\mathbf{0}) \in \mathcal{T}$ because $P \in \mathcal{T}$ (only condition 4 involves the message $\kappa()$, and it is preserved by deletion of such messages). We apply lemma B.12 to extend $C(\mathbf{0})$ with the translation of Q with index n: $C([\![Q^\kappa]\!]^\sharp_{e^n}) \twoheadrightarrow_d P''$ for some $P'' \in \mathcal{T}$.

A similar argument applies for a message $\kappa(a)$ with $\kappa \in K^n_b$ and $a \in F \cup L$, since $[\![Q^\kappa]\!]_{e^n}\{^a/_x\} = [\![Q^\kappa\{^a/_x\}]\!]_{e^n}$ (lemma B.10) if the continuation definition is $\kappa(x) \triangleright [\![Q^\kappa]\!]_{e^n}$ after α-converting bound names so that a does not occur bounded in Q^κ.

This argument also applies for a message $\kappa()$ with $\kappa \in K^n_c$ (replication step). The message is unique by condition 6 in P, and the same message is reemitted in the step, thus validating condition 6 in Q.

Communication step of the form: $C(\mathit{recv}^n(\kappa) \mid \mathit{send}^n(a)) \mapsto C(\kappa(a))$. We have $a \in F \cup L$ by condition 5 on P. Possibly after a COMM step that brings the message on κ to its defining location, all conditions hold in P'. As regards condition 5, the message $\mathit{recv}^n(\kappa)$ appears in M^n, so there is no other occurrence of κ in P.

0-step of the form $C(s(a, i^n, e^m, l^n) \mid in^n(b, \kappa)) \mapsto C(s(a, i^n, e^m, l^n \cup \{\text{In } b\ \kappa) \mid \mathit{sub}^m_{in}(i^n, b, \kappa))$ (or of the same form with a message on out^n instead of in^n)

Condition 2 holds after a COMM step that moves the resulting message $\mathit{sub}^m_{in}(b, \kappa)$ to M^m, Conditions 3 and 4b hold for n and κ in Q because we had condition 4a for n and κ in P, hence that $b \in F \cup L$, $\kappa \in K^n_a$ and that κ does not occur in l^n. Other conditions are unchanged.

1-steps using a rule in D^n_1 and consuming a message on amb^n. 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, 5, and 6 hold immediately.

Condition 4 holds because in the $\mathit{sub}^n_{in}(i^p, \_, \_)$ and $\mathit{sub}^n_{out}(i^p, \_, \_)$ 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 $\mathit{amb}^n(i^p, b, e^p)$ be present, which implies (condition 7(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 7a immediately holds for h^n. For h^p, the presence of an $\mathit{amb}^n(i^p, a, e^p)$ message implies that $S^p = s^p(a, i^p, \_, \_)$ (by condition 7(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 $\mathit{amb}^n(\_, \_, e^p)$, which is unique because of condition 7(a)i in P. The step creating a $r^p(e^r, \kappa^s)$ message fulfill condition 7(a)ii and this condition only, because there is no

$amb^t(\_,\_,e^p)$ nor o^p message in P'. The step creating a $o^p(\kappa^s)$ satisfies condition 7(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'.

2-step with the rule being used being in D_2^n. In both reductions, conditions 6, 5, and 3 clearly hold.

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 2.

By condition 4 on P, we know that any κ in a message $sub^p_{in}(i^n,b,\kappa)$ or $sub^p_{out}(i^n,b,\kappa)$ has an entry in l^n. Since the $s(a,i^n,e^m,l^n)$ message is being consumed and not re-emitted, these messages are stale. By condition 7a, 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 4), thus the $in^m(\_,\_)$ and $out^m(\_,\_)$ messages generated by the flush satisfy condition 4.

Condition 7(b)i is partially satisfied because S^n is now $f^n(e^m)$, $\alpha^n = \alpha^m h^m$ because of condition 7(a)iii for P. However, since the $s^n(a,i^n,e^m,l^n)$ message disappears, this breaks condition 7 if there is a message $amb^q(i^n,a,e^n,r^n,o^n)$. This is not possible by condition 7(a)iii for h^n in P. Thus condition 7 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 2 and 5. Condition 4 also holds because the continuation in the r^n message was present in the l^n (condition 4 in P) and is now only present in l^n.

Condition 7 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 7(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 study steps using any rule of the form $f^n(e^m) \mid C^n \triangleright f^n(e^m) \mid e^m.C$ where C ranges over the forwarded messages. P' is the process after the reduction and a Comm step that transports the forwarded message. In all cases, conditions 1, 2, 3, 5, and 6 immediately hold for P', and condition 7 holds for h^n in P'.

If $C = send(b)$ then all other conditions hold.

If $C = in(b,\kappa)$, $C = out(b,\kappa)$, $C = open(b,\kappa)$, or $C = recv(\kappa)$, then a message carrying a continuation name is replaced by another message carrying the same continuation, so condition 4 still holds.

If $C = sub_{in}(i^p,b,\kappa)$ or $C = sub_{out}(i^p,b,\kappa)$ then, by condition 4, κ is also in l^p and nowhere else. This still holds after the step, so condition 4 holds.

If $C = amb(i^p,b,e^p)$, then condition 7(a)i holds for some h^s in P, and still holds after the reduction, with a new $amb^m(i^p,b,e^p)$ message, so condition 7 holds with $\alpha^p = \alpha^m h^m h^n \beta$: by condition 7(a)i on P, for all h^t in β, $S^t = f^t(\_)$; we also have $S^n = f^n(\_)$. Thus, condition 7(a)i holds. $\square$

Corollary B.15 *Let $P \in \mathcal{T}$ such that $P \twoheadrightarrow P'$. We have $P' \in \mathcal{T}$.*

Proof: This is a direct consequence of lemma B.14 and the confluence of the $\twoheadrightarrow_d$ relation. □

In order to relate the dynamic trees of ambients and locations, we introduce a notion finer than the one of translation state:

Definition B.16 *$P \in \mathcal{J}$ is* well-formed *when there is $(Q, Q') \in \mathcal{A} \times \mathcal{E}$ such that $Q \rightarrow^*_{12C} Q'$ and $[\![Q']\!]^{t\sharp} \twoheadrightarrow_d \twoheadrightarrow^* P$.*

By lemma B.13 and corollary B.15, every well-formed process P is also a translation state. In addition, the following lemma states that migrations are always enabled in a well-formed process. It is similar to condition 3 of lemma B.2. (In the join calculus, migration is blocked when the target location is a sublocation of the moving location.)

Lemma B.17 *Let $P \in \mathcal{T}$ be a well-formed process. For any $n \in N$ in a transient state, the $\mathsf{go}(e^p.here)$ statement in S^n is enabled (i.e., h^p is not a sublocation of h^n.)*

Proof: For a given $P \in \mathcal{T}$, we order the index set N by location inclusion: $n \leq p$ when $h^p \in \alpha^n h^n$. In addition, we write $n \lessdot p$ when P meets condition 7 for n either in subcase 7(a)ii with $M^n \equiv M'_n \mid r^n(e^p, \kappa)$ or in subcase 7(b)ii with S^n of the form $\mathsf{go}(e^p.here); \_$. Intuitively, $n \lessdot p$ when n is going within p. We say that $P \in \mathcal{T}$ is *cycle-free* when the relation $\lessdot\leq$ for P has no cycle.

We first show that, for all $(R, Q) \in \mathcal{A} \times \mathcal{E}$, if $R \rightarrow^* Q$ and $[\![Q]\!]^{t\sharp} \twoheadrightarrow_d P$, then P is cycle-free. By hypothesis $R \rightarrow^* Q$, the extended ambient Q is as described in lemma B.2, and in particular the nesting of stubs and scions in Q is as described in condition 3. We index source ambients in Q with the same index set N as in P. Since the auxiliary translation is compositional and maps ambient boundaries to location boundaries, the ordering $\leq$ on N is also the subambient ordering on Q. By definition B.7, for each pair $n \lessdot p$ we have $Q = C(\bar{\imath}_n\{S_n\}\text{–}a_n[\,T_n\,])$ and $Q = C'_p(i_n)$ for some evaluation contexts $C(\cdot)$ and $C'_p(\cdot)$ with the hole of $C'_p(\cdot)$ in ambient a_p. For each pair $n \lessdot \leq q$ we thus have $Q = E(\,X_q a_q[\,F(i_n)\,]\,)$ for evaluation contexts $E(\cdot)$ and $F(\cdot)$. We show that $n \lessdot \leq m \lessdot \leq p$ implies $i_n \prec i_m$ in Q. Using the identities above twice for n, m and m, p, we have

$$Q = E(\,X_m a_m[\,F(i_n)\,]\,) = C(\,\bar{\imath}_m\{S_m\}\text{–}a_m[\,T_m\,]\,)$$

and, by matching the occurrences of ambient a_m, we obtain $T_m = F_n(i_n)$; this is the definition of $i_n \prec i_m$ in Q. To conclude, we remark that cycles in $\lessdot\leq$ yield cycles in $\prec$, which is ruled out by condition 3 on Q.

We now show that, for all well-formed cycle-free $P \in \mathcal{T}$, if $P \twoheadrightarrow P'$, then P' is cycle-free. (P' is well formed by definition). The proof is similar to the proof of condition 3 of lemma B.2. We write $\lessdot'$, $\leq'$ for the relations $\lessdot$,$\leq$ in P', and we track their changes. We rely on the same case analysis as in the proof of lemma B.14, considering reductions followed by normalization.

Go step. By definition B.8, there is $n \lessdot p$ such that the step migrates location h^n to location h^p. We have $\lessdot' = \lessdot \setminus \{(n,p)\}$ and $\leq' \subseteq (\leq \cup \{(n,p)\})^*$, so $\lessdot'\leq' \subseteq \lessdot (\leq \cup \lessdot)^*$ and P' is cycle-free if P is.

Continuation step, replication step. Such steps extend P as described in lemma B.12 using some $N' \supseteq N$. However, continuations trigger only translations of plain ambient processes, so $\lessdot' = \lessdot$, $\leq' \cap (N \times N) = \leq$, and thus $\lessdot'\leq' = \lessdot \leq$.

0-step, Communication step, 2-Open step. Such steps do not affect $\lessdot$ and $\leq$.

Forwarding step. Messages $r^n(e^p, \kappa)$ are not forwarded, so $\lessdot \leq$ is unchanged.

In 1 **step.** Let D_1^m contain the reaction rule being used, and let $sub_{in}(i^n, a, \kappa)$, $amb^m(i^n, \_, e^n)$, and $amb^m(i^p, a, e^p)$ be the messages of M^m consumed by the step. We obtain $\lessdot' = \lessdot \cup \{(n, p)\}$ and $\leq' = \leq$.

P meets condition 7(a)i for both n and p, so $n \leq m$ and $p \leq m$. In particular, P cannot meet condition 7(a)ii or 7(b)ii for p, hence $p \not\lessdot$. Condition 7(a)i for p also provides us with a string β of intermediate location names h^s with s opened that appear between p and m. Since P meets condition 7(b)i for all s, we also have $s \not\lessdot$.

Assume $n \lessdot' p$ appears in a cycle. Then, there is a cycle using $n \lessdot' p$ just once, of the form $n \lessdot' p \ (\lessdot \cup \leq)^* \ n$. The case $p \leq n$ is excluded because $p \leq n \leq m$ implies $n = p$, $n = m$, or $n = s$ with $h^s \in \beta$, which leads to contradictory conditions 7 for n and for m, p, or s. The cycle is thus of the form $n \lessdot' p \leq p' \lessdot \leq (\lessdot \leq)^* \ n$ and, by the arguments above, p' cannot be p or s with $h^s \in \beta$. We obtain $m \leq p'$, $m \leq (\lessdot \leq)^* n$ and finally $n \leq m \leq (\lessdot \leq)^* n$, which contradicts the hypothesis P cycle-free.

Out 1 **step.** Let D_1^m contain the reaction rule being used. This rule consumes themessages $amb^m(i^n,, e^n)$ and $s^m(\_, \_, e^p, \_)$; by condition 7a for m we have $m \leq p$; and by condition 7(a)i for n we have $n \leq m$. We obtain $\lessdot' = \lessdot \cup \{(n, p)\}$ and $\leq' = \leq$. However $n \leq p$, hence $\lessdot' \subseteq \lessdot \cup \leq$ does not create cycles.

Move 2 **step.** For some $n \lessdot p$, a process $\mathsf{go}(e^p.here); \_$ is substituted for the message $r^n(e^p, \kappa)$ and the state message. This does not affect $\lessdot$ and $\leq$. □

Next, we define simplification relations on translation states, and express their properties as commutation diagrams. A first simplification relation discards stale messages in translation states:

Lemma B.18 *For all $P, Q \in \mathcal{T}$, we write $P \succeq_{stale} Q$ when P is Q with a single additional stale message (as defined by condition 4). We have the diagrams:*

$$\begin{tikzcd}
P \arrow[r, "\succeq_{stale}"] \arrow[d, two heads, "Fwd"'] \arrow[dr, phantom, "(1)"] & Q \arrow[d, dotted, two heads, "{Fwd,=}"] \\
P' \arrow[r, dotted, "\succeq_{stale}"'] & Q'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, "\succeq_{stale}"] \arrow[d, two heads, "z"] \arrow[dr, phantom, "(2)"] & Q \arrow[d, dotted, two heads, "z"] \\
P' \arrow[r, dotted, "\succeq_{stale}"'] & Q'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, "\succeq_{stale}"] \arrow[d, dotted, two heads, "z"] \arrow[dr, phantom, "(3)"] & Q \arrow[d, two heads, "z"] \\
P' \arrow[r, dotted, "\succeq_{stale}"'] & Q'
\end{tikzcd}$$

where $\twoheadrightarrow_z$ ranges over $X \cup B \setminus \{\twoheadrightarrow_{Fwd}\}$ in (2) and $\twoheadrightarrow_z$ ranges over $X \cup B$ in (3).

Proof: Assume M^n contains a stale message $sub_{in}^n(i, \_, \_)$ or $sub_{out}^n(i, \_, \_)$. This message may be consumed using either a rule of D_1^n or a rule of D_F^n. However, by definition B.8 for P, we have $i \neq i^m$ for all $m \in N$ that meets condition 7a; hence, by sub-condition 7(a)i, there is no message $amb^n(i, \_, \_)$ in M^n, so the rule in D_1^n is not enabled. In the case D_F^n consumes the stale message, the message is forwarded on a different name sub_{in}^m or sub_{out}^m with the same contents, so the new message is also stale. Other steps apply independently of the stale message. Moreover, go steps produce messages on s^n with fresh identifiers, and all other step producing messages on s^n preserve their identifiers i^n, so a stale message in P is also stale in P'. □

Another simplification relation $\succeq_\kappa$ removes definitions that are not reachable.

Lemma B.19 (Useless definition) *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 $\twoheadrightarrow_z \in X \cup B$, we have diagrams

$$\begin{tikzcd} P \arrow[r, "\succeq_\kappa"] \arrow[d, "z"] & Q \arrow[d, dotted, "z"] \\ {} \arrow[r, dotted, "\succeq_\kappa"] & {} \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} P \arrow[r, "\succeq_\kappa"] \arrow[d, dotted, "z"] & Q \arrow[d, "z"] \\ {} \arrow[r, dotted, "\succeq_\kappa"] & {} \end{tikzcd} .$$

Proof: By definition, κ (or i) never appears in a redex on either side of $\succeq_\kappa$. When these names are introduced on the right-hand-side using a continuation step, preliminary α-conversion on the left-hand-side prevent clashes. $\square$

Location boundaries in the join calculus cannot be deleted 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.20 (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. *$S_P^m = f^m(e^n)$ (hence $\alpha_P^m = \alpha_P^n h^n$ by condition 7(b)i in definition B.8);*
2. *M_P^m is a parallel composition of messages on names in K_P^m;*

and that Q is P with the following changes:

3. *$N_Q = N_P \setminus \{m\}$;*
4. *$K_Q^n = K_P^n \cup K_P^m$ (with guarded translations of identical processes Q^κ);*
5. *$M_Q^n = M_P^n \mid M_P^m$;*
6. *for every $p \in N_Q$, α_Q^p is obtained from α_P^p by deletion of h^m (when present);*
7. *the names in e^n are substituted for the corresponding names in e^m.*

For every $\twoheadrightarrow_z \in X \cup B$, we have diagrams

$$\begin{tikzcd} P \arrow[rr, "\succeq_{Fwd}"] \arrow[d, "z"] & & Q \arrow[d, dotted, "z"] \\ P' \arrow[r, dotted, two heads, "{Fwd,*}"] & {} \arrow[r, dotted, "\succeq_{Fwd}"] & {} \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} P \arrow[rr, "\succeq_{Fwd}"] \arrow[d, dotted, "z"] & & Q \arrow[d, "z"] \\ P' \arrow[r, dotted, two heads, "{Fwd,*}"] & {} \arrow[r, dotted, "\succeq_{Fwd}"] & {} \end{tikzcd} .$$

Let $P, Q \in \mathcal{T}$ that meet all the statements above except statement 2. We have $P \twoheadrightarrow_{Fwd}^* \succeq_{Fwd} Q$.

Proof: To prevent confusion, we call invariants the statements of the lemma above, and use conditions only for the statements of definition B.8.

We begin with the second part of the lemma. Assume that P and Q meet all the invariants of the lemma except invariant 2 on M_P^m. Since $P \in \mathcal{T}$, by condition 2 on m, $M_P^m \equiv M_{P,\kappa}^m \mid M_{P,f}^m$ where $M_{P,\kappa}^m$ is a parallel composition of messages sent on names in K_P^m and where $M_{P,f}^m$ is a parallel composition of messages of the form $x^m(\widetilde{v})$ for some $x \in \{amb^n, in^n, out^n, open^n, recv^n, send^n, sub_{in}^n, sub_{out}^n, o^n, r^n\}$. By condition 7 on m, x^m cannot be r^n or o^n. By definition, D^m_F thus contains a rule of the form

$$f^m(e) \mid x^m(\widetilde{v}) \triangleright f^m(e) \mid e.x(\widetilde{v})$$

Moreover, we have $S^m_P = f^m(e^n)$ by invariant 1, so the rule above is enabled, and there is a forwarding step that substitutes the message $e^n.x(\widetilde{v}) = x^n(\widetilde{v})$ for $x^m(\widetilde{v})$. Using a COMM step, the message can be moved to S^n_P. By performing these two steps for every message in $M^m_{P,f}$, we obtain $P \twoheadrightarrow^*_{Fwd} P'$ and, since x^n is already substituted for x^m in Q by invariant 5, $P' \succeq_{Fwd} Q$.

To establish the diagrams of the lemma, we establish a direct correspondence between steps $P \twoheadrightarrow_z P'$ and steps $Q \twoheadrightarrow_z Q'$, obtained by substituting n for m in every redex. For every pair of matching steps, we often rely on the second part of the lemma to obtain $P' \twoheadrightarrow^*_{Fwd} P'' \succeq_{Fwd} Q'$. We partition steps in P and Q according to $X \cup B$:

Go-step reducing $M^p = \mathsf{go}(h^q); I^p$. By invariant 1, we have $p \neq m$, so the migration is enabled in P if and only if it is enabled in Q. In the case $q = m$, the Go-step in Q moves the location into h^n, while the corresponding Go-step in P moves the location into h^m; the additional message on amb^m in P (obtained after normalization) appears in M^n in P', and appears in M^m in Q'. Still, from P' the message can be forwarded to M^m, hence $P' \twoheadrightarrow_{Fwd} P'' \succeq_{Fwd} Q'$. In the case the migration is towards h^m or one of its sub-location—or conversely the migration is from a sublocation of h^m towards a location that isn't—we check that invariant 6 is preserved for all paths α of the moving locations.

Continuation step, Replication step Steps consuming a message on $\kappa \in K^n_P$ in M^m_P and M^n_Q, respectively, trigger processes in h^m and h^n. After normalization, we obtain processes related by the second part of the lemma, possibly with a larger set of continuations defined in h^m. Steps consuming a message on $\kappa \in K_P \setminus K^n_P$ are in direct correspondence, with $P' \succeq_{Fwd} Q'$.

0-step. If the step occurs with index $p \in N_P$ and P meets condition 7(a)i with S^p of the form $s^p(\_,\_,e^n,\_)$, then a COMM step moves the delegation message to M^n in P and to M^m in Q. However, in P' the message in M^n can be forwarded to M^m. All other steps are in direct correspondence.

Communication step, 1-step, 2-step By invariant 1 and condition 7 for n, these steps cannot occur with index n. In the case these steps release a continuation on a name in $K^n_{P,ab}$, we obtain (after a COMM step) the same message in M^n_P and M^m_Q, respectively, as described by invariant 5. Otherwise, these steps are in direct correspondence.

Forwarding step. By invariant 2, this step cannot forward a message of M^n_P. However, a message may be forwarded to M^n_P from an opened sublocation of h^n. In that case, we apply the second part of the lemma to forward the message one step higher. Otherwise, these steps are in direct correspondence. $\square$

We are now ready to study the commutation properties of bookkeeping steps with the steps enabled in well-formed translation states.

Lemma B.21 (Bookkeeping) *For every well-formed $P \in \mathcal{T}$, let $\twoheadrightarrow_b \in B$ and $\twoheadrightarrow_z \in B \cup X$, and consider the diagrams:*

$$
\begin{tikzcd}
P \arrow[rr, two heads, "0"] \arrow[dd, "\textsc{Move 2}"'] \arrow[ddrr, phantom, "(1)"] & & Q \arrow[d, dotted, two heads, "\textsc{Move 2}"] \\
& & {} \arrow[d, dotted, two heads, "Go"] \\
R \arrow[r, dotted, two heads, "Go"] & {} & P' \arrow[l, dotted, "\preceq_{stale}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[rr, two heads, "0"] \arrow[d, "\textsc{Open 2}"'] \arrow[drr, phantom, "(2)"] & & Q \arrow[d, dotted, two heads, "\textsc{Open 2}"] \\
R \arrow[r, dotted, two heads, "Fwd"] & {} & P' \arrow[l, dotted, "\preceq_{stale}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, two heads, "b"] \arrow[d, "b"'] \arrow[dr, phantom, "(3)"] & Q \arrow[d, dotted, two heads, "b^{=}"] \\
R \arrow[r, dotted, two heads, "b^{=}"] & P'
\end{tikzcd}
\qquad
\begin{tikzcd}
P \arrow[r, two heads, "b"] \arrow[d, "z"'] \arrow[dr, phantom, "(4)"] & Q \arrow[d, dotted, two heads, "z"] \\
R \arrow[r, dotted, two heads, "b"] & P'
\end{tikzcd}
$$

For $b = 0$ and $z =$ Move 2 *we have either (1) or (4); For $b = 0$ and $z =$* Open 2 *we have either (2) or (4); For $b = z$ we have (3); For all other b and z we have (4).*

Proof: For each $\mapsto_b \twoheadrightarrow_d \in B$, we use definition B.8 to identify conflicting steps, if any:

Go-step. Such steps consume go statements of S^n; they don't interfere with steps using other rules. Two go steps may be mutually-exclusive in the join calculus, but this is ruled out by lemma B.17 on P.

Continuation step, Replication step. Such steps consume single messages on $\kappa \in K$. By condition 1 on P, there is a single definition in P for every κ, so these steps commute with one another and with other steps.

Forwarding step. Such a step uses a rule in some D^n_F; it requires that P meets condition 7(b)i for n, which excludes any step using another rule of D^n. Besides, every rule of D^n_F leaves the message $S^n = f^n(e^m)$ unchanged, so forwarding steps commute with one another.

0-step. For some $n \in N$, the 0-step uses a rule in D^n_0 that consumes a message $in^n(a, \kappa)$ (or a message $out^n(a, \kappa)$) and a message $S^n = s^n(b, i^n, e^m, l^n)$, and that produces a message $amb^m_{in}(i^n, a, \kappa)$ (or $amb^m_{out}(i^n, a, \kappa)$) and the message $s^n(b, i^n, e^m, l')$ where the log l' is l^n with an additional entry. Thus, the step requires that P meet condition 7a for n, and it preserves that condition. As mentioned above, this rules out a step using D^n_F to consume messages on in^n and out^n. Other steps consuming the message S^n and not producing the same message use a rule of D^n_0 or D^n_2. Steps using rule D^n_0 commute with one another (extending logs with the same entries in a different order). The two other cases are handled in diagrams (1) and (2).

In diagram (1), the Move 2 step applies before and after the 0-step. In both cases, a Go-step is enabled (applying lemma B.17) and its normalization flushes the log of messages and generates a message on s^n with a fresh uid i. If the 0-step occurs before Move 2, the log contains an additional entry that yields back the message $in^n(a, \kappa)$ (or $out^n(a, \kappa)$) consumed by the 0-step. Besides, the message produced by the 0-step contains the original identifier i^n and, since all names i^m are distinct, this message now is a stale message.

In diagram (2), the Open 2 step applies before and after the 0-step. As above, a preliminary 0-step may consume a message on in^n or out^n, extend the log, and produce a message on sub^m_{in} or sub^m_{out} with identifier i^n. The Open 2 flushes the log (using names in^p and out^p instead of in^n and out^n), thus producing an additional message $in^p(a, \kappa$ (or $out^p(a, \kappa)$) if the 0-step occurred beforehand. Besides, the message produced by the 0-step contains the original identifier i^n, so this message now is a stale message. Instead, if the 0-step did not occur, a rule in D_F^n consumes the message $in^n(a, \kappa$ (or $out^n(a, \kappa)$) in M^n and produces the message $in^p(a, \kappa$ (or $out^p(a, \kappa)$). $\square$

Definition B.22 *Let $(P, Q) \in \mathcal{A} \times \mathcal{E}$. We write $Q \in \mathcal{E}^{\#}$ when $P \rightarrow_{12C}^{*} Q$, and when $Q = \nu\widetilde{a}.Q'$ in which Q' is unrestricted. We say that Q is a normalized extended ambient.*

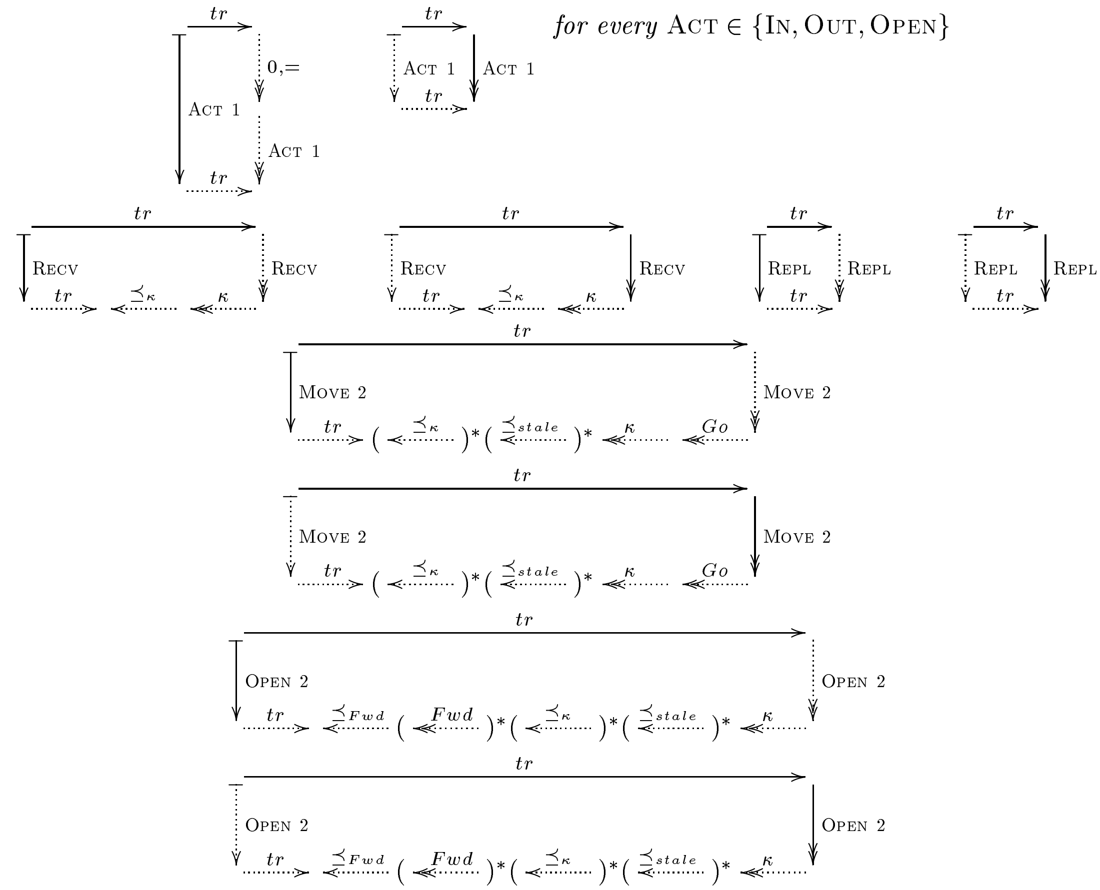
In the extended ambient calculus, We write $P \mapsto_x Q$ when $Q \in \mathcal{E}^{\#}$ and when $P \rightarrow_x Q$ using neither α-conversion nor scope extrusion before the step, and using α-convertion and scope extrusion only for newly-introduced restrictions.

We let $\rightarrow_{tr} \stackrel{\text{def}}{=} [\![\ \cdot\ ]\!]^{t\sharp} \twoheadrightarrow_d \twoheadrightarrow_0^*$ 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 0-steps.

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

Lemma B.23 (Operational correspondence) *We have the following diagrams:*

for every $\textsc{Act} \in \{\textsc{In}, \textsc{Out}, \textsc{Open}\}$



Proof: We detail the proofs for MOVE 2.

Let $P, P' \in \mathcal{E}^{\#}$ and $Q, Q_1 \in \mathcal{T}$ such that $P \mapsto_{\text{MOVE 2}} P'$ and $[\![P]\!]^{t\sharp} \twoheadrightarrow_d Q_1 \twoheadrightarrow_0^* Q$. Since $P \in \mathcal{E}^{\#}$, invariant 2 of lemma B.2 implies that $P = E(i)(\bar{\imath}\{P_i\}\!-\!n[\,R_i\,])$. Let p be the index of the ambient where the scion i appears, and n the index of the corresponding stub. We write E^n for $D^n \wedge E'^n \wedge \kappa() \triangleright [\![Q^\kappa]\!]_e$. By definition of the translation, we have:

$$Q_1 = C_i(I)(M_i^p)(M_i^m)(\alpha^m h^m h^n[E^n : s^n(n, i^n, e^m, \emptyset) \mid r^n(e^p, \kappa) \mid M_i^n])$$

for some context C_i such that the first hole contains the set of identifiers, the second hole the M process for p, the third hole the M process for m, and the fourth hole is the location for ambient with index n.

We now consider the series of steps $Q_1 \twoheadrightarrow_0^* Q$. We have:

$$Q = C(I)(M^p)(M^m)(\alpha^m h^m h^n[D^n \wedge E^n : s^n(n, i^n, e^m, l^n) \mid r^n(e^p, \kappa) \mid M^n])$$

where $M^m = M'^m \mid M_s^m$ with M_s^m being the messages delegated from n to m, where $M^n \uplus G^n = M_i^n \uplus M'^n$ with G^n being the in^n and out^n messages delegated to m and composing the log l^n, and M'^n the messages delegated to n.

In the following we write I_i for $I \uplus \{i\}$, I_Q for the set of identifier introduced in the translation of Q^κ, I' for $I_i \uplus I_Q$, M_n^p for $M^p \mid amb^p(i, n, e^n)$, $E_Q^n = E^n \wedge E_Q$ where E_Q is the definition introduced by the translation of Q^κ, and M_Q^n for $M^n \mid M_Q$ where M_Q is the process introduced by the translation of Q^κ.

We have:

$$\begin{array}{rll}
Q & \twoheadrightarrow_2 & C(I)(M^p)(M^m)(\alpha^m h^m h^n[E^n : \mathsf{go}(h^p); (I^n_{b,e^n,e^p} \mid \kappa() \mid F^n) \mid M^n]) \\
& \twoheadrightarrow_{Go} & C(I_i)(M_n^p)(M^m)(\alpha^p h^p h^n[E^n : s^n(n, i, e^p, \emptyset) \mid \kappa() \mid G^n \mid M^n]) \\
& \twoheadrightarrow_\kappa & C_Q(I')(M_n^p)(M^m)(\alpha^p h^p h^n[E_Q^n : s^n(n, i, e^p, \emptyset) \mid G^n \mid M_Q^n]) \\
& \succeq^*_{stale} & C_Q(I')(M_n^p)(M'^m)(\alpha^p h^p h^n[E_Q^n : s^n(n, i, e^p, \emptyset) \mid G^n \mid M_Q^n]) \\
& \succeq^*_\kappa & C_Q(I_f)(M_n^p)(M'^m)(\alpha^p h^p h^n[E_f^n : s^n(n, i, e^p, \emptyset) \mid G^n \mid M_Q^n])
\end{array} \qquad (1)$$

where I_f is $I' \setminus \{i^n\}$ and E_f^n is $D^n \wedge E'^n \wedge E_Q$, that is E_Q^n without the definition for κ, and where C_Q is C extended with the locations occuring in the translation of Q^κ.

Let $C_{i,Q}$ be the context C_i extended with the locations occuring in the translation of Q^κ. By compositionality of the translation, we have:

$$[\![P']\!]^{t\sharp} \twoheadrightarrow_d C_{i,Q}(I_f)(M_i^p)(M_i^m)(\alpha^p h^p h^n[E_f^n : s^n(n, i, e^p, \emptyset) \mid M_i^n \mid M_Q])$$

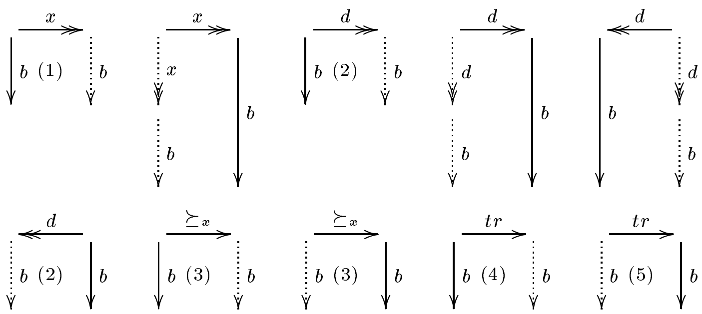
Every 0-step from Q_1 to Q is still enabled, and performing all of them except those involving G^n, yield configuration (1). □

We now establish diagrams concerning the barbs. First we define a notion of barbs for the translation states using the context $T_b(\,\cdot\,)$ defined in section 3

Definition B.24 (Translated barbs) *For all $P \in \mathcal{J}$, we let $P \downarrow_b^4$ iff $T_b(P) \rightarrow^4 \downarrow_{yes}$.*

These four steps are as follows: a COMM step brings message $p(t)$ to its defining location h_T; the message is consumed, triggering a message $t(b)$; a COMM step brings message $t(b)$ to the top-level location defining t in D_t^0; if a message of the form $amb^0(\_, b, \_)$ is present in M^0, the rule D_t^0 yields a message $yes()$. For any $P \in \mathcal{T}$, this is the only reduction path to observe the barb $\downarrow_{yes}$, so $P \downarrow_b^4$ if and only if there is a message $amb^0(i^m, b, e^m)$ in M^0 with $b \in F$. In order to deal with barbs using diagrams, to each observation predicate $\downarrow$ we associate a partial relation between processes and a singleton $\mathbf{0}$ defined as $\{(P, \mathbf{0}) \mid P \downarrow\}$. We consider diagrams crossing barbs with horizontal relations.

Lemma B.25 *Let $\mapsto_x \in B$ and $\succeq_x \in \{\succeq_\kappa, \succeq_{stale}, \succeq_{Fwd}\}$. We have:*



Proof: Tile **(1)** holds because the only bookkeeping step that may delete a message on amb^0 is a forwarding step that consumes a message on f^0, and because such messages are excluded by condition 2 on translation states.

Tiles **(2)** hold because deterministic steps commute with other steps, thus the four steps involved in a barb, because COMM steps and FLUSH steps cannot remove barbs, and because barbs are preserved by $=_d$.

Tiles **(3)** hold because no simplification relation removes nor add a message on amb^0. In the case of $\preceq_{Fwd}$, this property holds because all forwarding steps have been performed beforehand.

Tile **(4)** holds because there is a top-level ambient with non-restricted name b, thus there is a message on amb^0 with this same name b by definition of the translation. The additional $\twoheadrightarrow_0$ steps do not remove this message. Since b is not restricted in the extended ambient, $b \in F$ in the translation.

Tile **(5)** holds for similar reasons. The presence of the message $amb^0(i^n, b, e^n)$ implies, by translation state condition 7(a)i, that there is no message on r^n and o^n. By definition of the translation, there is a plain ambient named b in the extended ambient process. This ambient is at top-level, because the translation yield no opened location and because of condition 7(a)i of the translation state for n. This ambient with name b is not restricted, because otherwise b would be in L, thus not in F.

Other tiles are immediate. $\square$

Next we define our candidate bisimulation, as a relation between well-formed processes and join configurations that contains the auxiliary translation:

Lemma B.26 *Let H' be the set of relations on join configurations*

$$H' \stackrel{def}{=} \{\twoheadleftarrow_\kappa, \twoheadleftarrow_{Fwd}, \twoheadleftarrow_0, \twoheadleftarrow_{Go}, \preceq_\kappa, \preceq_{Fwd}, \preceq_{stale}, \succeq_{stale}\}$$

and let $\mathcal{H} \stackrel{def}{=} \rightarrow_{tr} \left(\bigcup_{\phi \in H'} \phi\right)^*$. We have $[\![\cdot]\!]^{t\sharp} \subseteq \mathcal{H}$.

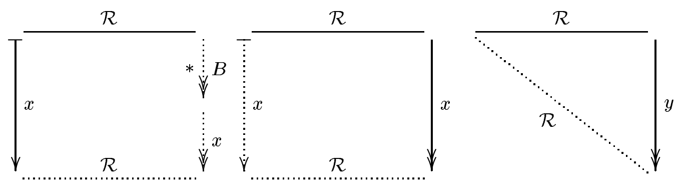
Proof: Let $Q \in \mathcal{E}$ be well-formed. By definition of $\xrightarrow{tr}$, we have the diagram

$$\begin{tikzcd} Q \arrow[r, "[\![\cdot]\!]^{t\sharp}"] \arrow[rr, bend right, "tr"'] & [\![Q]\!]^{t\sharp} \arrow[r, two heads, "d"] & (\xrightarrow{0})^* R \end{tikzcd}$$.

□

We are now ready to establish the correctness of the translation. The next theorem is more precise than Theorem 3. It uses a notion of hybrid barbed strong bisimulation up to bookkeeping.

Theorem 4 *Let $\mathcal{R}$ be a relation between extended ambient processes and join configurations such that $P \mathrel{\mathcal{R}} Q$ implies $P \Downarrow_b$ iff $T_b(Q) \Downarrow_{yes}$ and that satisfies the following diagrams for every $\mapsto_x \in X$ and $\mapsto_y \in B$:*



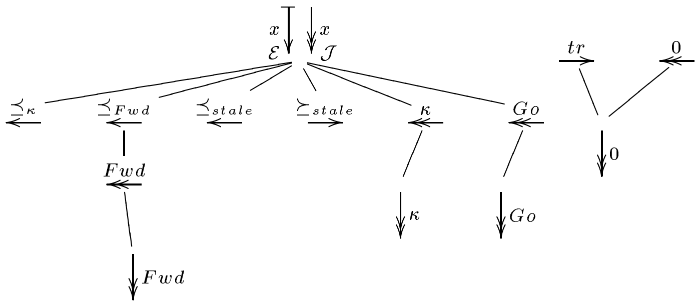
We call such relations hybrid barbed strong bisimulation*.*

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

Proof: In order to close each of the three diagrams for $\mathcal{H}$, and also to establish the correspondence of barbs, we glue the elementary diagrams (or tiles) of lemmas B.18, B.19, B.20, B.21, B.23, and B.25. Most diagrams introduce further horizontal and vertical steps, so the existence of a finite tiling is not obvious. To establish finite tiling, we apply a general confluence result relying on *decreasing diagrams*, proposed by Van Oostrom [18] and adapted to bisimulation proofs in [10].

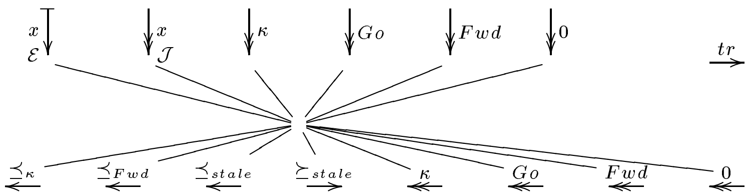
For each target diagram, we identify "horizontal" and "vertical" relations that may appear in the tiles, and we partially order these relations. Then, we check that for every product of an horizontal and a vertical relation, we have a tile that is decreasing for the strict partial order on relations. The decreasing diagram theorem then yields the commutation for all series of horizontal and vertical relations, obtained by gluing these tiles.

For establishing the finite tiling of the top-left diagram for each extended ambient step $\mapsto_x$, we use the following partial order:



This partial order is extended to prove the finite tiling for weak barbs, with $\Downarrow_b$ being uncomparable to any other relation.

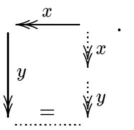
For establishing the finite tiling of the top-right diagram for each reduction $\twoheadrightarrow_x \in X$, we use the following partial order:



Extending this partial order to include the uncomparable relation $\Downarrow_b$ establishes the finiteness of the tiling for weak barbs. This partial order is also used to prove the third diagram of theorem 4.

The table below takes the cartesian product of horizontal and vertical relations; for each pair, the proof requirement consists of two decreasing tiles; we refer to the lemmas that provide the tiles, and ckeck that they are decreasing. (The partial orderings above are obtained by collecting the minimal assumptions that make all tiles decreasing.) We write $\emptyset$ when the tile holds vacuously. When each tile is obtained independently, we mention first the argument for the top-left tile, then the argument for the top-right tile.

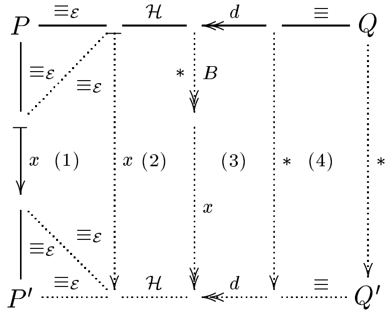
	$\xrightarrow{tr}$	$\xleftarrow{\preceq_\kappa}$	$\xleftarrow{\preceq_{Fwd}}$	$\xleftarrow{\preceq_{stale}}$, $\xrightarrow{\succeq_{stale}}$	$\xtwoheadleftarrow{B}$
$\downarrow^{B,X}_{\mathcal{J}}$	$\emptyset$ B.23	B.19	B.20	B.18	**(1)** B.21
$\downarrow^{X}_{\mathcal{E}}$	B.23 $\emptyset$	$\emptyset$			
$\downarrow_b$	B.25				

Each decreasing tile in **(1)** immediately follows from our choice of the ordering: since the relations in B appear both as horizontal and vertical relations, for any $\twoheadrightarrow_x \in B$ and $\twoheadrightarrow_y \in B \cup X$, we use  . $\square$

To complete our proof, we consider the candidate relation $\mathcal{R} \stackrel{\text{def}}{=} \equiv_\mathcal{E} \mathcal{H} \twoheadleftarrow_d \equiv$.

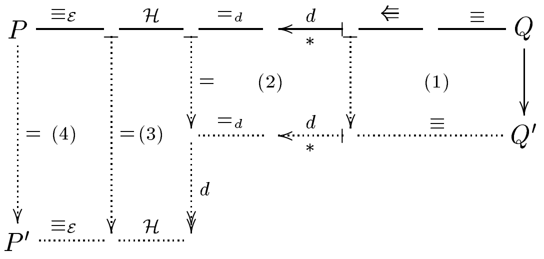
Corollary B.27 *$\mathcal{R}$ is a weak hybrid barbed bisimulation.*

Proof: To establish that $\mathcal{R}$ is a simulation, we use the diagram



(1) immediately holds by definition of $\equiv_{\mathcal{E}}$, since $\rightarrow\ \subseteq\ \equiv_{\mathcal{E}}\mapsto\equiv_{\mathcal{E}}$; **(2)** holds by theorem 4; **(3)** holds by lemma B.9 (3 and 4), percolating the $=_d$ relation to the end of the reduction series; and **(4)** holds by structural equivalence in the join calculus.

To establish the converse simulation property, let $P \in \mathcal{E}$ be a well formed process, and let $Q, Q' \in \mathcal{J}$ such that $P\ \mathcal{R}\ Q$ and $Q \rightarrow Q'$. We have:



(1) holds by lemma B.9(5). **(2)** holds by lemmas B.9(1) and B.9(3), commuting the vertical step with the deterministic steps and the $=_d$ relation. There is no corresponding vertical step if the step considered was deterministic. **(3)** uses lemma B.14 to yield the vertical deterministic normalization, and by theorem 4. **(4)** holds by definition of structural equivalence for ambients. We conclude by lemma B.9(4) which states that $\Rrightarrow\mapsto_d^*=_d\twoheadrightarrow_d\ \subseteq\twoheadrightarrow_d$, thus $P'\ \mathcal{R}\ Q'$.

The preservation of weak barbs follows from their preservation in the relations $\mathcal{H}$, $\equiv$, $\equiv_{\mathcal{E}}$, and $\twoheadleftarrow_d$, as stated in lemma B.25 and theorem 4. $\square$

Lemma B.28 *For every $P \in \mathcal{T}$, the length of normalized bookkeeping reductions $P \twoheadrightarrow_B^*$ is bounded.*

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

$$P = P_1 \twoheadrightarrow_B P_2 \twoheadrightarrow_B P_3 \ldots$$

We prove that this series is finite. By definition of $\twoheadrightarrow$, we know that $P_i \in \mathcal{T}$ for all i. We define the depth of a message on a name defined in D^n as the length of β if $\alpha^n h^n = \alpha^p \beta$ where α^p is the first enclosing location that is alive, as defined in condition 7a of a translation state (definition B.8). Using the partition B, we write g, κ, o, for the number of `go` steps, continuation steps, 0-steps respectively enabled. We write f_0 for the sum of the depth of messages on *in*, *out*, *open*, *amb*, *recv*, and *send* 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 $(g,\ \kappa,\ f_0,\ o,\ f_1)$, and associate such a tuple to each P_i. We check that the tuple strictly decreases with i, 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 1-steps, which are not bookkeeping steps. 0-steps consume messages on *in*, *out* or *open* which were at depth 0, but decreases the number of enabled 0-steps. 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 does not enable any continuation nor Go-steps. Continuation steps do not create any Go-step nor any other continuation step. Go-steps do not enable new Go-steps.

We conclude using the strong normalization of deterministic steps as stated in lemma B.9(1). $\square$

Lemma B.29 *The relation $\mathcal{R}$ preserves divergence.*

Proof: Let $P \in \mathcal{E}$ be a well-formed process and $Q \in \mathcal{J}$ such that $P \mathcal{R} Q$. We first prove that $P \Uparrow$ implies $Q \Uparrow$. Divergence is preserved by extended ambient structural equivalence, by the first diagram of theorem 4, by the relation $\twoheadleftarrow_d$ and by join calculus structural equivalence.

Conversely, assume that Q has an infinite series of steps, and let $Q' \in \mathcal{T}$ such that $P \equiv_{\mathcal{E}}\mathcal{H} \;\; Q' \twoheadleftarrow_d \equiv Q$. We build an alternate series $Q' \twoheadrightarrow \twoheadrightarrow \dots$ as in the proof of the second bisimulation diagram of lemma B.27. This series is infinite since it reflects every steps in the original series except deterministic steps, and deterministic steps strongly normalize. By lemma B.28, there is no infinite series of normalized bookkeeping reductions $\twoheadrightarrow_B$, thus there are also infinitely many $\twoheadrightarrow_X$ steps. Since the second diagram of theorem 4 preserves divergence, we obtain an infinite series of steps starting from P. $\square$

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

Proof of Theorem 1: Let P be an ambient process. As regards weak- and fair-must barbs, we compose the equations of theorem 2 ($P \lessgtr P$ with different operational semantics) and of theorem 3 ($P \approx^{aj} [\![P]\!]^t$).

We begin with a remark on fair-must barbs that applies to each of these equations. Let $\lessgtr$ 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, $\lessgtr$ also respects fair-must barbs: if $P \lessgtr Q$, $\Box P \Downarrow_b$, and $Q \rightarrow'^* Q'$, then $P \rightarrow^* P' \geqslant Q'$ (using the simulation property of $P \geqslant Q$) and $P' \rightarrow^* P'' \leqslant Q'$ (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 $P'' \leqslant Q$) and finally $\Box Q \Downarrow'_b$.

For the notion of translated barb $\downarrow^4_b$ used in the statement of theorem 3, 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 3 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 4, in combination with the finite-bookkeeping lemma B.28: 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. $\square$