Bisimulations in the Join-Calculus

M. Boreale
Dipartimento di Scienze dell'Informazione, Università "La Sapienza," Via Salaria 113, 00198 Roma, Italy. `michele@dsi.uniroma1.it`

C. Fournet
INRIA Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France. `Cedric.Fournet@inria.fr`

C. Laneve
Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy. `laneve@cs.unibo.it`

This work is partly supported by the ESPRIT CONFER-2 WG-21836

Abstract
We develop a theory of bisimulation in the join-calculus. We introduce a refined operational model that makes interactions with the environment explicit, and we discuss the impact of the lexical discipline of the join-calculus on its extensional semantics. We propose several formulations of bisimulation and we establish that all formulations yield the same equivalence. We prove that this equivalence is finer than barbed congruence, but that both relations coincide in the presence of name-testing.

Keywords
Asynchronous processes, barbed congruence, bisimulation, chemical abstract machine, join-calculus, pi-calculus

1 INTRODUCTION

The join-calculus is a recent formalism for modeling mobile systems [7, 8]. Its main motivation is to relate two crucial issues in concurrency: distributed implementation and formal semantics. To this end, the join-calculus enforces a strict lexical discipline over the channel names that appear in processes: names can be sent and received, but their communication capabilities cannot be affected by the receivers. This is the *locality property.*

Locality yields a simpler distributed model, because the communication primitives of the calculus can be directly implemented via standard primitives of asynchronous systems [8, 10]. It also plays a prominent role in the design of implicit type systems for the join-calculus, because all contravariant occurrences of names are static [9]. In this paper, we show that locality strongly affects the treatment of bisimulation, and provides a semantics simpler than other proposals in the literature.

In order to reason about distributed processes, we must equip the join-calculus with semantics that have both a sensible discriminating power and some convenient proof techniques. The usual approach in concurrency is to focus on elementary forms of interaction between the process and its environment. Formally, this is achieved through the definition of two notions that are common to most process calculi: a *reduction relation* $\rightarrow$ that represents internal evolution and an *observation predicate* $\Downarrow_a$ that detects the ability of interacting at a given channel. Based on these two notions, numerous observational semantics can be defined. As regards discriminating power, it would be adequate to test the observation predicate under all possible contexts; indeed, the resulting *trace-based semantics* are relevant to many practical issues in distributed programming, especially when one is not interested in the branching structure of processes. In order to establish trace-based equivalences, however, one must cope with quantification over both contexts and traces, which makes the proofs particularly difficult [12].

To tackle these problems, it is convenient to consider behavioral equivalences that are finer and easier to check, possibly using them as indirect proof techniques for coarser semantics. Among these, bisimulation-based semantics are especially convenient, because co-inductive proofs need only to consider single reduction steps instead of traces [14]. *Barbed bisimilarity* has been proposed in [17] as a uniform basis to define sensible behavioral equivalences on different process calculi. This is the approach chosen for the join calculus in [7, 8], where *barbed congruence* is defined as the coarsest congruence that is a barbed bisimulation. Yet, checking that two processes are barbed congruent still requires explicit quantification over all contexts. Indeed, most proofs of [7] have to analyze in details how contexts and processes get intertwined, and to partition their reductions accordingly. This makes routine or automated checking of these properties problematic.

In this paper, we introduce a labeled operational model for the join calculus, and we equip it with the standard *weak bisimulation* [14], which enjoys the usual, purely co-inductive proof technique on labeled transitions. Then, we use weak bisimulation to establish several sensible equalities between join-calculus processes. We also consider alternate definitions of this equivalence and relate weak bisimulation to barbed congruence, showing that the difference between the two semantics is precisely due to the absence of a name-matching operator. A more precise account of our work follows.

The original semantics of the join-calculus is based on the *reflexive chemical abstract machine* (RCHAM) [7], which only accounts for the internal evolution of processes. In order to shape labeled bisimulation upon join-calculus, we first propose the *open* RCHAM, a refined operational semantics that explicitly models interaction with the environment. Via these interactions, the environment can get acquainted with local names of the process that are emitted at free names—the *extrusions*—and can later emit on these names—the *intrusions* ("intrusion" has another technical meaning in [16]). Due to the principle of locality, intrusions and extrusions occur on disjoint sets of names. To keep track of the environment's acquaintance with local names, definitions—the technical device for defining the computational steps of RCHAM's—are marked with extruded names as soon as these extrusions happen. In turn, intrusions are allowed on these extruded names. The resulting open syntax for the join-calculus has processes of the form $\mathsf{def}_S\ D\ \mathsf{in}\ P$, where S is a set of names defined by D and extruded to the environment. This also induces a refined chemical machine, called the open RCHAM.

Let us illustrate what locality means in our setting. The process $\mathsf{def}_{\{x\}}\ x\langle u\rangle \triangleright P\ \mathsf{in}\ x\langle v\rangle$ defines a name x such that, whenever a message $x\langle u\rangle$ is received, a fresh copy of process P is started. The name x is marked as extruded; the environment can therefore send messages on x to trigger copies of the process P. However, the environment cannot interfere with the definition of x; in particular, the message $x\langle v\rangle$ cannot be consumed by the environment, and as a consequence any sensible equivalence $\approx$ would identify the two processes below:

$$\mathsf{def}_{\{x\}}\ x\langle u\rangle \triangleright P\ \mathsf{in}\ x\langle v\rangle \quad \approx \quad \mathsf{def}_{\{x\}}\ x\langle u\rangle \triangleright P\ \mathsf{in}\ P\{^v/_u\}$$

Weak bisimulation is obtained by applying the standard definition to the open RCHAM. Weak bisimilarity is stable under renaming, is a congruence over contexts of the open calculus, and it suitably abstracts from the actual structure of the join-patterns. For instance, it is insensitive to buffering, as it is expected for an asynchronous semantics.

The open RCHAM allows intrusion of messages on every name that has been extruded; this clause embeds asynchrony in the operational semantics, but it also generates extraneous transitions in case these messages are not used by the process. That is, our model hides the actual receiving capabilities of the process, although this information is available from the syntax of the process. Unfortunately, weak bisimulation cannot benefit from this information, and needs to consider numerous useless intrusions.

To reduce the size of our model, we modify the open RCHAM and equip it with an alternative equivalence called *asynchronous bisimulation*. The new chemical machine restricts intrusions to sets of messages that immediately trigger a reaction rule; this technical device significantly augments the blocking capability of processes. Asynchronous bisimulation allows the intrusion of several messages at the same time, instead of single ones. We prove that weak bisimulation and asynchronous bisimulation coincide, which validates the use of asynchronous bisimulation in proofs.

The last characterization of bisimulation is given in terms of barbed congruence. The barbed congruence of [7] is strictly coarser than weak bisimulation, because contexts of barbed congruence have no internal name-testing capability, while labels of weak bisimulation separates distinct names that exhibit the same behavior. This is actually the only difference between the two equivalences. To establish this, we augment the join-calculus with a name-matching operator and we show that (a variant of) barbed congruence coincides with weak bisimulation.

Beside its use as a proof technique, our semantics yields a better understanding of the join-calculus, and provides a basis for comparing it with other calculi, which are usually equipped with weak bisimulations, and especially with the asynchronous π-calculus [5]. In these calculi, asynchrony means that message outputs have no continuation, and thus that there is no direct way to detect that a message has been received. Noticeably, the usual weak bisimulation of the π-calculus has too much discriminating power, and separates processes with the same behavior (e.g., $0 \not\approx x(u).\overline{x}\langle u\rangle$); several remedies are considered in [11, 1]. Our bisimulations for the join-calculus make use of similar accommodations, but they yield simpler semantics, mostly because locality constrains interaction with the environment. This prunes the number of transitions to consider, and rules out processes such as $z(x).x(u).\overline{x}\langle u\rangle$ where x is used by the environment both for input and output.

The rest of the paper is organized as follows. In Section 2 we introduce our language and its model. In Section 3 we define weak bisimulation in this setting, we give examples, and we study its basic properties. In Section 4 we introduce asynchronous bisimulation, and we prove that it coincides with weak bisimulation. In Section 5 we relate our bisimulations to barbed congruence, and study the impact of name-testing. In Section 6 we discuss related work for the asynchronous π-calculus. We conclude in Section 7. Due to space limitations, the proofs are omitted; they can be found in the full paper [3].

2 THE OPEN JOIN-CALCULUS

In this section, we define the open join-calculus and its operational semantics as extensions of the join-calculus and the RCHAM of [7].

Syntax and Scopes. Let $\mathcal{N}$ be a countable set of *names* ranged over by $x, y, u, v, \ldots$; tuples of names are noted $u_i{}^{i\in 1..p}$ or simply $\widetilde{u}$. Names can carry fixed-length messages, and we assume a recursive sorting discipline on names that avoids arity mismatch, in the style of [15]. We consider only well-sorted terms, and we assume an infinite number of names of each sort. We omit the sorting discipline, as it is not important to understand the contents of the paper.

The open join-calculus processes A and the join-calculus processes P, the definitions D, and the *join-patterns* J are the terms defined as follows:

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
A & \stackrel{\text{def}}{=} & \mathsf{0} & P & \stackrel{\text{def}}{=} & \mathsf{0} \\
 & & x\langle u_i{}^{i\in 1..p}\rangle & & & x\langle u_i{}^{i\in 1..p}\rangle \\
 & & A \mid A & & & P \mid P \\
 & & \mathsf{def}_S\ D \text{ in } A & & & \mathsf{def}\ D \text{ in } P \\
D & \stackrel{\text{def}}{=} & J \triangleright P & J & \stackrel{\text{def}}{=} & x\langle u_i{}^{i\in 1..p}\rangle \\
 & & D \wedge D & & & J \mid J
\end{array}
$$

A process P can be the inert process 0, a message sent on a name x that carries a tuple of names $\widetilde{u}$, a parallel composition of processes, or the definition of local names and the process within their scope; a definition D is a conjunction of reaction rules $J \triangleright P$ that associate join-patterns J to guarded processes P; the intended meaning is that, whenever messages match the pattern J, the guarded process P can be triggered. An open process A is like a process, except that it has open definitions at top-level instead of local ones. The open definition $\mathsf{def}_S\ D \ \mathsf{in}\ \ldots$ exhibits a subset S of names defined by D that are visible from the environment: the *extruded names*. When omitted, the index set S is empty (no extruded names so far). We identify such open definitions $\mathsf{def}_\emptyset\ D\ \mathsf{in}\ P$ with local definitions $\mathsf{def}\ D\ \mathsf{in}\ P$.

Definition 1 *We partition names in terms as follows:* visible names *are either* free names $\mathsf{fv}[D]$, $\mathsf{fv}[A]$ *or* extruded names $\mathsf{xv}[A]$*;* local names *are either* received names $\mathsf{rv}[J]$ *or* defined names $\mathsf{dv}[J], \mathsf{dv}[D]$.

$$
\begin{array}{rcl@{\qquad}rcl}
J:\ \mathsf{rv}[x\langle\widetilde{v}\rangle] & \stackrel{def}{=} & \{u \in \widetilde{v}\} & \mathsf{rv}[J \mid J'] & \stackrel{def}{=} & \mathsf{rv}[J] \uplus \mathsf{rv}[J'] \\
\mathsf{dv}[x\langle\widetilde{v}\rangle] & \stackrel{def}{=} & \{x\} & \mathsf{dv}[J \mid J'] & \stackrel{def}{=} & \mathsf{dv}[J] \uplus \mathsf{dv}[J'] \\
D:\mathsf{dv}[J \triangleright P] & \stackrel{def}{=} & \mathsf{dv}[J] & \mathsf{dv}[D \wedge D'] & \stackrel{def}{=} & \mathsf{dv}[D] \cup \mathsf{dv}[D'] \\
\mathsf{fv}[J \triangleright P] & \stackrel{def}{=} & \mathsf{dv}[J] \cup (\mathsf{fv}[P] \backslash \mathsf{rv}[J]) & \mathsf{fv}[D \wedge D'] & \stackrel{def}{=} & \mathsf{fv}[D] \cup \mathsf{fv}[D'] \\
A:\ \mathsf{fv}[A \mid A'] & \stackrel{def}{=} & (\mathsf{fv}[A] \backslash \mathsf{xv}[A']) \cup (\mathsf{fv}[A'] \backslash \mathsf{xv}[A]) & \mathsf{fv}[0] & \stackrel{def}{=} & \emptyset \\
\mathsf{xv}[A \mid A'] & \stackrel{def}{=} & \mathsf{xv}[A] \uplus \mathsf{xv}[A'] & \mathsf{xv}[0] & \stackrel{def}{=} & \emptyset \\
\mathsf{fv}[\mathsf{def}_S\ D\ \mathsf{in}\ A] & \stackrel{def}{=} & (\mathsf{fv}[A] \cup (\mathsf{fv}[D] \backslash \mathsf{xv}[A])) \backslash \mathsf{dv}[D] & \mathsf{fv}[x\langle\widetilde{v}\rangle] & \stackrel{def}{=} & \{x, \widetilde{v}\} \\
\mathsf{xv}[\mathsf{def}_S\ D\ \mathsf{in}\ A] & \stackrel{def}{=} & S \uplus \mathsf{xv}[A] & \mathsf{xv}[x\langle\widetilde{v}\rangle] & \stackrel{def}{=} & \emptyset
\end{array}
$$

The above definition induces well-formed conditions on terms, which are made more explicit below. As usual, the operator $\uplus$ means disjoint union. In the whole paper, we consider processes modulo α-conversion on *bound names*, namely received names and defined non-extruded names, and we always assume that:

1. join-patterns are linear: as in the join-calculus of [7], both defined variables and received variables appear at most once in every join pattern. The restriction on received variables rules out name-matching. The restriction on defined variables does not affect expressiveness (cf. [7, 6]) but significantly simplifies our definitions;
2. sets of names extruded by different open sub-processes are disjoint—these names are defined in different definition;
3. open definitions $\mathsf{def}_S\ D\ \mathsf{in}\ P$ define all their extruded names ($S \subseteq \mathsf{dv}[D]$).

The *interface* of an open process A consists of two disjoint sets of names: free names $\mathsf{fv}[A]$ used in A to send messages out, and extruded names $\mathsf{xv}[A]$ used by the environment to send message in.

A *global renaming* σ is a substitution on the interface of open processes that is injective on the image of extruded names (for all names $x \in \mathsf{xv}[A]$, $y \in \mathsf{xv}[A] \cup \mathsf{fv}[A]$, if $x\sigma = y\sigma$ then $x = y$).

Open chemistry. Let us introduce the operational semantics of the open join-calculus through an example. It will also explain the dynamic role played by the index S in a definition. Consider the process $\mathsf{def}_\emptyset\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ z\langle x\rangle$. The interface contains no extruded name and two free names y, z. The message $z\langle x\rangle$ can be consumed by the environment, thus exporting x.

$$\mathsf{def}_\emptyset\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ z\langle x\rangle \quad \xrightarrow{\{x\}\overline{z}\langle x\rangle} \quad \mathsf{def}_{\{x\}}\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ 0\ .$$

Once x is known from the environment, it cannot be considered local anymore—the environment can emit on x—, but is not free either—the environment cannot modify or extend its definition. A new transition is enabled:

$$\mathsf{def}_{\{x\}}\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ 0 \quad \xrightarrow{x\langle\rangle} \quad \mathsf{def}_{\{x\}}\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ x\langle\rangle$$

Now the process can input some more messages on x, and also perform the two transitions below to emit on y:

$$\begin{array}{lll} \mathsf{def}_{\{x\}}\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ x\langle\rangle & \longrightarrow & \mathsf{def}_{\{x\}}\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ y\langle\rangle \\ & \xrightarrow{\{\}\overline{y}\langle\rangle} & \mathsf{def}_{\{x\}}\ x\langle\rangle \triangleright y\langle\rangle\ \mathsf{in}\ 0 \end{array}$$

In order to model the interactions with the environment, we extend the RCHAM model defined in [7]. This is accomplished by means of two dual operations: *extrusion* of defined names on free names, and *intrusion* of messages sent on previously extruded names. The explicit bookkeeping of extruded names is performed by suitably augmenting the chemical solutions.

Formally, let the functions $\mathsf{dv}[\cdot], \mathsf{fv}[\cdot], \mathsf{xv}[\cdot]$ be extended to multisets of terms in the expected way. *Open chemical solutions*, ranged over by $\mathcal{S}, \mathcal{T}, \ldots$, are triples $(\mathcal{R}, S, \mathcal{A})$, written $\mathcal{R} \vdash_S \mathcal{A}$, where

- $\mathcal{A}$ is a multiset of open processes with disjoint sets of extruded names;
- $\mathcal{R}$ is a multiset of definitions such that $\mathsf{dv}[\mathcal{R}] \cap \mathsf{xv}[\mathcal{A}] = \emptyset$;
- $S \subseteq \mathsf{dv}[\mathcal{R}]$ is a set of extruded names.

The definitions of free and extruded names are lifted to solutions in this way: $\mathsf{fv}[\mathcal{R} \vdash_S \mathcal{A}] \stackrel{\text{def}}{=} (\mathsf{fv}[\mathcal{R}] \cup \mathsf{fv}[\mathcal{A}]) \setminus \mathsf{dv}[\mathcal{R}]$ and $\mathsf{xv}[\mathcal{R} \vdash_S \mathcal{A}] \stackrel{\text{def}}{=} S \uplus \mathsf{xv}[\mathcal{A}]$.

The chemical rules for the open RCHAM are given in Table 1; they define families of transitions $\rightharpoonup$, $\leftharpoondown$, $\longrightarrow$, $\xrightarrow{\alpha}$, $\alpha \in \{S\overline{x}\langle\widetilde{v}\rangle, x\langle\widetilde{v}\rangle\}$ between open solutions. As usual, each chemical rule mentions only the processes and the definitions it involves, and the transition applies to every chemical solution $\mathcal{S}$ whose multisets contain these processes and definitions.

Let us comment on the rules. The first five rules are those of the RCHAM. The structural rules that define the inverse transitions $\rightharpoonup$ and $\leftharpoondown$ are unchanged, except for Rule S-DEF that performs some extra bookkeeping on extruded

S-NIL	$\Vdash 0$	$\rightleftharpoons$	$\Vdash$
S-PAR	$\Vdash A \mid A'$	$\rightleftharpoons$	$\Vdash A,\, A'$
S-AND	$D \wedge D \Vdash$	$\rightleftharpoons$	$D,\, D' \Vdash$
S-DEF	$\Vdash_S \mathsf{def}_{S'}\, D \;\mathsf{in}\; A$	$\rightleftharpoons$	$D\sigma \Vdash_{S \uplus S'} A\sigma$
RED	$J \triangleright P \Vdash J\rho$	$\longrightarrow$	$J \triangleright P \Vdash P\rho$
EXT	$\Vdash_S x\langle v_i{}^{i\in 1..p}\rangle$	$\xrightarrow{S'\overline{x}\langle v_i{}^{i\in 1..p}\rangle}$	$\Vdash_{S\cup S'}$
INT	$\Vdash_{\{x\}\cup S}$	$\xrightarrow{x\langle v_i{}^{i\in 1..p}\rangle}$	$\Vdash_{\{x\}\cup S} x\langle v_i{}^{i\in 1..p}\rangle$

Side conditions on the reacting solution $\mathcal{S} = (\mathcal{R} \Vdash_S \mathcal{A})$:

S-DEF	σ replaces $\mathsf{dv}[D] \setminus S'$ with distinct fresh names.
RED	ρ substitutes names for $\mathsf{rv}(J)$;
EXT	x is free, and $S' = \{v_i \mid i \in 1..p\} \cap (\mathsf{dv}[\mathcal{R}] \setminus S)$;
INT	$v_i^{i\in 1..p}$ are either free, or fresh, or extruded.

Table 1 The open RCHAM of the open-join-calculus

names. The reduction rule RED says when a message can trigger a process in a definition. In the sequel, $\rightleftharpoons$ is the symmetric relation $\rightharpoonup \cup \leftharpoondown$ that contains all structural moves, its transitive closure $\rightleftharpoons^*$ is called structural equivalence; $\rightleftharpoons\!\!\!\!\rightarrow$ is the relation $\rightleftharpoons \cup \rightarrow$ that gathers all silent moves.

The two last rules model interaction with the context. According to rule EXT, messages can be emitted at free names; messages can export defined names that were not previously known to the environment, thus causing the scope of their definitions to be opened. This is made explicit through the label S' that appears on the transition. Our rule resembles the Open rule for restriction in the π-calculus [16], with an important restriction due to locality: messages are either emitted on free names, to be consumed by EXT, or on names defined in the open solution, to be consumed by RED.

The intrusion rule INT can be viewed as a disciplined version of one of the Input rule for the asynchronous π-calculus proposed by Honda and Tokoro [11]: the latter allows intrusion of arbitrary messages at any stage, whereas our rule allows intrusion of messages only on defined-extruded names.

Notation Let $\mathcal{S}$ be the open solution $\mathcal{R} \vdash_S \mathcal{A}$ and $M = \prod x\langle\widetilde{v}\rangle$ be a parallel composition of messages that can be intruded using Rule INT. Then $\mathcal{S} \mid M$ is the open solution $\mathcal{R} \vdash_S \mathcal{A},\, M$. The *restriction* on extruded names is partially defined as

$$(\mathcal{R} \vdash_{S \uplus S'} \mathcal{A}) \setminus S' \quad \stackrel{\text{def}}{=} \quad \mathcal{R} \vdash_S \mathcal{A}\ .$$

Formally, the original RCHAM of [7] is the restriction of our open RCHAM that operates on chemical solutions with no extruded variables, and that uses every chemical rule except Ext and Int.

When applied to open solutions, the structural rules capture the intended meaning of extruded names: messages sent on extruded names can be moved inside or outside of their defining process. Assuming x has been extruded ($x \in S \cup S' \cup \mathsf{xv}[A]$), and modulo α-conversion to avoid clashes with the arguments ($\{\widetilde{v}\} \cap \mathsf{dv}[D] \subseteq S'$), we have

$$\vdash_S \; x\langle\widetilde{v}\rangle \,|\mathsf{def}_{S'}\, D \;\mathsf{in}\; A \quad \rightleftharpoons^* \quad \vdash_S \; \mathsf{def}_{S'}\, D \;\mathsf{in}\; (x\langle\widetilde{v}\rangle \,|\, A)$$

Remark 2 *In its original presentation [7], the join-calculus is equipped with an auxiliary labeled transition system. Its intent, however, is quite different from ours. It is used to give an alternative, syntactic description of chemical reduction steps; transitions are labeled by whole reaction rules; these large labels are discarded when the transition applies within a definition that contains the reaction rule. In contrast, our open* RCHAM *provides an extensional semantics of the calculus; labels are much simpler than definitions, and we do not define silent steps as matching labeled transitions plus hiding.*

3 WEAK BISIMULATION

The join-calculus has been "opened" to support the standard notion of bisimulation for process calculi [14]. In this section, we define this equivalence for open solutions and we investigate its properties.

Definition 3 *A symmetric relation $\mathcal{R}$ over open solutions is a* weak bisimulation *if, whenever $\mathcal{S} \;\mathcal{R}\; \mathcal{T}$, then*

- *if $\mathcal{S} \rightleftharpoons^* \rightarrow \rightleftharpoons^* \mathcal{S}'$ then $\mathcal{T} \Longrightarrow^* \mathcal{T}'$ and $\mathcal{S}' \;\mathcal{R}\; \mathcal{T}'$;*
- *if $\mathcal{S} \rightleftharpoons^* \xrightarrow{\alpha} \rightleftharpoons^* \mathcal{S}'$ then $\mathcal{T} \Longrightarrow^* \xrightarrow{\alpha} \Longrightarrow^* \mathcal{T}'$ and $\mathcal{S}' \;\mathcal{R}\; \mathcal{T}'$, for all labels α of shape $x\langle\widetilde{v}\rangle$ or $S\overline{x}\langle\widetilde{v}\rangle$ where names in S are fresh in $\mathcal{T}$.*

Let $\approx$ be the largest weak bisimulation. We shall write $A \approx B$ for $(\emptyset \Vdash A) \approx (\emptyset \Vdash B)$.

As an immediate consequence of its definition, weak bisimulation is an equivalence relation, and we can tell whether two processes are bisimilar by reasoning on their synchronization tree, without the need to exhibit discriminating contexts. For example, $x\langle u\rangle \not\approx x\langle v\rangle$ because they perform emissions with different labels. Likewise, $x\langle y\rangle \not\approx \mathsf{def}\; z\langle u\rangle \triangleright y\langle u\rangle \;\mathsf{in}\; x\langle z\rangle$ because the first process emits a free name (label $\overline{x}\langle y\rangle$) while the latter emits a local name that gets extruded (label $\{z\}\overline{x}\langle z\rangle$). The following properties should convince the reader that $\approx$ is a reasonable semantics for the join-calculus.

Bisimilarity and chemical interface. The set of extruded names $\mathsf{xv}[\mathcal{S}]$ is the input interface of the solution $\mathcal{S}$, and rule INT makes weak bisimulation sensitive to it: $\mathcal{S} \approx \mathcal{T}$ implies $\mathsf{xv}[\mathcal{S}] = \mathsf{xv}[\mathcal{T}]$.

To check this property, let us consider the partition of open RCHAMs induced by $\mathsf{xv}[\cdot]$. The resulting classes are stable for all chemical rules except EXT, and in particular, any solution is structurally equivalent to a "fully-heated" (containing only simple reaction rules and messages) solution with the same extruded variables; by definition of INT, a fully-heated solution can perform a transition with label $x\langle\widetilde{v}\rangle$ if and only if $x \in \mathsf{xv}[\mathcal{S}]$.

Conversely, rule EXT defines the only transitions that can separate solutions with the same extruded names. Obviously, open solutions whose derivatives never perform EXT are bisimilar if and only if they have the same extruded names to begin with, and this provides some "garbage-collection" properties such as $\mathsf{fv}[P] = \emptyset$ implies $P \approx \mathsf{0}$.

Equivalent definitions. The following examples show that weak bisimulation is largely insensitive to the shape of join-patterns.

Weak bisimulation is insensitive to buffering, as one expects from an asynchronous semantics. For instance we have:

$$\mathsf{def}_S\ D \ \mathsf{in}\ A \quad \approx \quad \mathsf{def}_S\ x\langle\widetilde{u}\rangle \triangleright x'\langle\widetilde{u}\rangle \wedge D' \ \mathsf{in}\ A$$

where $x \in \mathsf{dv}[D]$, x' is a fresh name, and D' is obtained from D by substituting $x'\langle\widetilde{v}\rangle$ for $x\langle\widetilde{v}\rangle$ in every join-pattern. We also have:

$$\mathsf{def}_S\ J \triangleright P \wedge D \ \mathsf{in}\ A \quad \approx \quad \mathsf{def}_S\ J \triangleright x\langle\widetilde{\mathsf{rv}[J]}\rangle \wedge x\langle\widetilde{\mathsf{rv}[J]}\rangle \triangleright P \wedge D \ \mathsf{in}\ A$$

where x is a fresh name and $\widetilde{\mathsf{rv}[J]}$ is a tuple that conveys the names in $\mathsf{rv}[J]$.

These properties of asynchrony are easy to prove (it suffices to consider the relation that contains processes that differ only because of a buffering reduction, which is included in a bisimulation, or equivalently, to use the bisimulation up-to expansion proof technique of [21]). Notice, however, that buffering of partial join-patterns would not usually preserve bisimulation, because it affects the branching structure of processes.

It is well-known that weak bisimulation is insensitive to divergence; for instance $\mathsf{0} \approx \mathsf{def}\ x\langle\rangle \triangleright x\langle\rangle \ \mathsf{in}\ x\langle\rangle$. Likewise, any reversible synchronization is invisible; for instance, we can temporarily grab arbitrary messages J within a definition D

$$\mathsf{def}_S\ D \ \mathsf{in}\ A \quad \approx \quad \mathsf{def}_S\ J \triangleright J' \wedge J' \triangleright J \wedge D \ \mathsf{in}\ A$$

where $\mathsf{dv}[J] \subseteq \mathsf{dv}[D]$, $\mathsf{dv}[J']$ fresh, and $\mathsf{rv}[J'] = \mathsf{rv}[J]$.

Another standard property of bisimulation is that it is not possible to observe the internal state of processes. As mentioned in [7] for the weaker barbed congruence, we have

$$\mathsf{def}_S\ J \triangleright P \ \mathsf{in}\ A \quad \approx \quad \mathsf{def}_S\ J|s\langle\widetilde{v}\rangle \triangleright P|s\langle\widetilde{v}\rangle \ \mathsf{in}\ A|s\langle\widetilde{v}\rangle$$

where s is a fresh name and $\{\widetilde{v}\} \cap \mathsf{rv}[J] = \emptyset$. Indeed, there is always one available message $s\langle\widetilde{v}\rangle$ that conveys the same names, that were initially bound in a process *within the same scope*. This equality suffices to prove interesting properties with regards to our scoping rules: if we take $\{\widetilde{v}\} = \mathsf{dv}[J] \cup \{s\}$, then all occurrences of names of $\mathsf{dv}[J]$ that appeared in P—we call them recursive occurrences—are now bound as received variables. Up-to bisimulation, we can therefore eliminate recursion from every definition. If we take $\{\widetilde{v}\} = (\mathsf{fv}[P] \cup \{s\}) \setminus \mathsf{rv}[J]$, then all names in P are now bound as received variables, which is reminiscent of lambda-lifting in the λ-calculus. In combination with the forthcoming congruence properties, this validates a compilation scheme for the join-calculus that would replace every process by an equivalent process with simpler binders (either receptions or immediate definitions).

3.1 Renaming and Congruence properties

The rest of the section is devoted to proving that weak bisimulation is indeed a congruence. This is not very surprising since our calculus has no operator for external choice. Nevertheless the proof is not completely standard and some interesting properties are required. In order to establish that weak bisimulation is persistent under global renaming, we first study the correspondence between transitions in a chemical solution $\mathcal{S}$ and transitions in $\mathcal{S}\sigma$, where σ is a global renaming.

Lemma 1 *Let $\mathcal{S}$ be an open solution and σ be a global renaming for $\mathcal{S}$.*

1. *$\mathcal{S} \rightleftharpoons^* \rightarrow \rightleftharpoons^* \mathcal{T}$ implies $\mathcal{S}\sigma \rightleftharpoons^* \rightarrow \rightleftharpoons^* \mathcal{T}\sigma$; conversely, $\mathcal{S}\sigma \rightleftharpoons^* \rightarrow \rightleftharpoons^* \mathcal{T}$ implies $\mathcal{S} \rightleftharpoons^* \xrightarrow{S_1\overline{x_1}\langle\widetilde{v}_1\rangle} \xrightarrow{y_1\langle\widetilde{v}_1\rangle} \cdots \xrightarrow{S_n\overline{x_n}\langle\widetilde{v}_n\rangle} \xrightarrow{y_n\langle\widetilde{v}_n\rangle} \rightleftharpoons^* \rightarrow \rightleftharpoons^* \mathcal{S}'$, where $n \geq 0$ and, for all $i \leq n$, $\sigma(x_i) = \sigma(y_i)$ and $\mathcal{T} = (\mathcal{S}' \setminus (\bigcup_{i=1\ldots n} S_i))\sigma$;*
2. *$\mathcal{S} \rightleftharpoons^* \xrightarrow{x\langle\widetilde{v}\rangle} \rightleftharpoons^* \mathcal{T}$ if and only if $\mathcal{S}\sigma \rightleftharpoons^* \xrightarrow{(x\langle\widetilde{v}\rangle)\sigma} \rightleftharpoons^* \mathcal{T}\sigma$;*
3. *if $\sigma(x)$ is free for $\mathcal{S}\sigma$, $\mathcal{S} \rightleftharpoons^* \xrightarrow{S\overline{x}\langle\widetilde{v}\rangle} \rightleftharpoons^* \mathcal{T}$ implies $\mathcal{S}\sigma \rightleftharpoons^* \xrightarrow{(S\overline{x}\langle\widetilde{v}\rangle)\sigma} \rightleftharpoons^* \mathcal{T}\sigma$; conversely, $\mathcal{S}\sigma \rightleftharpoons^* \xrightarrow{S\overline{x}\langle\widetilde{v}\rangle} \rightleftharpoons^* \mathcal{T}$ implies $\mathcal{S} \rightleftharpoons^* \xrightarrow{S'\overline{x'}\langle\widetilde{v}'\rangle} \rightleftharpoons^* \mathcal{S}'$, where $\mathcal{S}'\sigma \rightleftharpoons^* \mathcal{T}$ and $(S'\overline{x'}\langle\widetilde{v}'\rangle)\sigma = S\overline{x}\langle\widetilde{v}\rangle$.*

The statements in the above lemma are natural, except perhaps for the last part of Case 1. This case occurs when an internal step consumes messages on extruded names $\sigma(x_i)$ where x_i are free names in the initial solution $\mathcal{S}$. For every such message that is consumed in the internal step, this is mimicked as an extrusion on x_i followed by an intrusion on y_i.

Corollary 2 *$\mathcal{S} \approx \mathcal{T}$ implies $\mathcal{S}\sigma \approx \mathcal{T}\sigma$ for every global renaming σ.*

The congruence property essentially relies on this proposition; its proof is standard.

Rules S-JOIN, S-AND, S-DEF, EXT are as in Table 1;		
INT-J	$J \triangleright P \Vdash_S M \xrightarrow{M'} J \triangleright P \Vdash_S P\rho$	
Side condition:	$J\rho \rightleftharpoons^* M \mid M'$, $Dom(\rho) = \mathsf{rv}[J]$, $\mathsf{dv}[M'] \subseteq S$, $\mathsf{rv}[M']$ free, fresh or extruded.	

Table 2 The J-open RCHAM of the open join calculus

Theorem 1 *Weak bisimulation is a congruence.*

4 ASYNCHRONOUS BISIMULATION

In order to prove that two chemical solutions are bisimilar, the large number of multiset configurations to consider is a nuisance; for example, a process with an extruded name already has an infinite set of configurations, even if no "real" computation is ever performed. Indeed, for checking the obvious equivalence

$$\mathsf{def}\ x\langle u\rangle|y\langle v\rangle \triangleright P \ \mathsf{in}\ z\langle x\rangle \quad \approx \quad \mathsf{def}\ x\langle u\rangle \mid y\langle v\rangle \triangleright Q \ \mathsf{in}\ z\langle x\rangle$$

one is confronted to infinite models on both sides, with a distinct chemical solution for every multiset of messages that have been intruded on x so far. This flaw in our definition of bisimulation motivates the following alternative formulation.

We refine the open RCHAM by allowing inputs only when they trigger some guarded process. Thus, we augment the blocking ability of inputs. For example, the two processes above become inert, and trivially bisimilar.

If we apply this refinement keeping the previous labels for input, however, we obtain a dubious result. The solution $x\langle\rangle|y\langle\rangle|z\langle\rangle \triangleright P \Vdash_{\{x,y\}} z\langle\rangle$ can transit inasmuch as it first inputs the two messages $x\langle\rangle$ and $y\langle\rangle$, then performs a silent step that consumes them together with the local message $z\langle\rangle$ already in the solution. Yet, neither $x\langle\rangle$ nor $y\langle\rangle$ alone can trigger the process P, and therefore this solution would becomes inert too. This suggests the use of *join*-inputs on x and y in new reduction steps such that

$$x\langle\rangle \mid y\langle\rangle \mid z\langle\rangle \triangleright P \Vdash_{\{x,y\}} z\langle\rangle \xrightarrow{x\langle\rangle \mid y\langle\rangle} x\langle\rangle \mid y\langle\rangle \mid z\langle\rangle \triangleright P \Vdash_{\{x,y\}} P \ .$$

On the other hand, the solution $x\langle\rangle|y\langle\rangle|z\langle\rangle \triangleright P \Vdash_{\{x\}} z\langle\rangle$ is truly inert, since P can never be triggered. Here join-inputs have a greater blocking ability than atomic inputs, and our refinement suppresses all input transitions.

4.1 The J-open RCHAM

The J-open RCHAM is defined in Table 2, by replacing the intrusion rule. Unlike the previous rule INT, the new rule INT-J permits intrusion of messages only if they are immediately used to trigger a process in a definition. This is formalized by allowing labels M' that are multisets of messages, and by handling them as partial join-patterns: if the solution can supply a complementary join-pattern M such that the combination $M \mid M'$ matches the join-pattern in a reaction rule, then the transition occurs and triggers this reaction rule. As for INT, we restrict intrusions in M' to messages on extruded names. Note that rule RED is now subsumed by rule INT-J where $M' = 0$, although we keep the distinction between internal moves and proper input moves in the following discussion.

The same chemical solution now has two different models: for instance, $x\langle\rangle|y\langle\rangle \triangleright P \vdash_{\{x\}}$ has no transition in the J-open RCHAM, while it has infinitely many $\xrightarrow{x\langle\rangle}$ transitions in the open RCHAM In the sequel, we shall keep the symbol $\xrightarrow{\alpha}$ for the open RCHAM and use $\xrightarrow{\alpha}_J$ for the J-open RCHAM; the subscript J is dropped whenever no ambiguity can arise.

As a direct consequence of the definitions of the J-open semantics, we have the following relation between our two models:

Proposition 3 *The intrusions of the J-open RCHAM and those of the open RCHAM are related as follows:*

1. $\mathcal{S} \xrightarrow{x_1\langle\widetilde{v}_1\rangle \mid \cdots \mid x_n\langle\widetilde{v}_n\rangle}_J \mathcal{S}'$ *implies* $\mathcal{S} \xrightarrow{x_i\langle\widetilde{v}_1\rangle} \cdots \xrightarrow{x_n\langle\widetilde{v}_n\rangle} \rightleftharpoons^* \rightarrow \mathcal{S}'$.
2. *For any A, if* $A \mid x\langle\widetilde{u}\rangle \rightleftharpoons^* \xrightarrow{M}_J \rightleftharpoons^* B$, *then either* $A \rightleftharpoons^* \xrightarrow{M}_J \rightleftharpoons^* A'$ *with* $A' \mid x\langle\widetilde{u}\rangle \rightleftharpoons^* B$, *or* $A \rightleftharpoons^* \xrightarrow{M'}_J \rightleftharpoons^* B$ *with* $M' \rightleftharpoons^* M \mid x\langle\widetilde{u}\rangle$.

All other transitions are common to both chemical machines.

Asynchronous bisimulation

We now try to shape the definition of weak bisimulation (Definition 3) to the new J-open RCHAM. Consider for instance the two processes:

$$P \;=\; \mathsf{def}\ x\langle\rangle \triangleright a\langle\rangle \wedge a\langle\rangle|y\langle\rangle \triangleright R\ \mathsf{in}\ z\langle x, y\rangle \qquad\qquad Q \;=\; \mathsf{def}\ x\langle\rangle|y\langle\rangle \triangleright R\ \mathsf{in}\ z\langle x, y\rangle$$

With the previous underlying model (open RCHAM), P and Q are weakly bisimilar, but with the new one (J-open RCHAM) this is not true anymore, because, after emitting on z, P can input on $x\langle\rangle$ while Q cannot. If we equip the J-open RCHAM with the weak bisimulation of Section 3, Q becomes inert after the extrusion on z because join inputs are not considered. But if we consider the weak bisimulation that uses join-input labels instead of single ones, Q can input on $x\langle\rangle|y\langle\rangle$ while P cannot, and P and Q are still separated. It turns out that weak bisimulation discriminates too much in the J-open RCHAM.

In order to avoid detection of the structure of join patterns, weak bisimulation must be weakened further on. At least, we must consider that a process simulate another one even if it does not immediately consume a join-input.

Definition 4 *A symmetric relation $\mathcal{R}$ over* J*-open* RCHAM*s is an* asynchronous bisimulation *if, whenever $\mathcal{S}\ \mathcal{R}\ \mathcal{T}$, then*

- $\mathsf{xv}[\mathcal{S}] = \mathsf{xv}[\mathcal{T}]$
- *if $\mathcal{S} \rightleftharpoons^* \xrightarrow{S\overline{x}\langle\widetilde{v}\rangle} \rightleftharpoons^* \mathcal{S}'$ and names in S are fresh in $\mathcal{T}$, then $\mathcal{T} \Longrightarrow^* \xrightarrow{S\overline{x}\langle\widetilde{v}\rangle} \Longrightarrow^* \mathcal{T}'$ and $\mathcal{S}'\ \mathcal{R}\ \mathcal{T}'$;*
- *if $\mathcal{S} \rightleftharpoons^* \xrightarrow{M} \rightleftharpoons^* \mathcal{S}'$, then $\mathcal{T} \mid M \Longrightarrow^* \mathcal{T}'$ and $\mathcal{S}'\ \mathcal{R}\ \mathcal{T}'$.*

Let $\approx_a$ be the largest asynchronous bisimulation. We write $A \approx_a B$ for $(\emptyset \Vdash \{A\}) \approx_a (\emptyset \Vdash \{B\})$.

Let us show the equality $P \approx_a Q$, where P and Q are the processes at the beginning of this section. Both P and Q perform the same emission, therefore it suffices to prove

$$A = \mathsf{def}_{\{x,y\}}\ x\langle\rangle \triangleright a\langle\rangle\ \wedge\ a\langle\rangle|y\langle\rangle \triangleright R \ \mathsf{in}\ 0 \quad \approx_a \quad B = \mathsf{def}_{\{x,y\}}\ x\langle\rangle|y\langle\rangle \triangleright R \ \mathsf{in}\ 0$$

Observe that $A \xrightarrow{x\langle\rangle} A'$, while $B \not\xrightarrow{x\langle\rangle}$. Nevertheless, it is possible to prove that $A' \approx_a \mathsf{def}_{\{x,y\}}\ x\langle\rangle \mid y\langle\rangle \triangleright R\ \mathsf{in}\ x\langle\rangle$. Hence, bisimilarity can be still achieved, by delaying the actual use of a message. Conversely, there is B' such that $B \xrightarrow{x\langle\rangle \mid y\langle\rangle} B'$, whilst this move cannot be mimicked by A. Again, one can prove that B' is asynchronous bisimilar to $\mathsf{def}_{\{x,y\}}\ x\langle\rangle \triangleright a\langle\rangle \wedge a\langle\rangle \mid y\langle\rangle \triangleright R\ \mathsf{in}\ R$, which is a derivative of $A \mid x\langle\rangle \mid y\langle\rangle$. All other transitions are the same, and we can conclude that $P \approx_a Q$.

Remark 5 *In the definition of asynchronous bisimulation, it is necessary to require that related solutions have the same extruded names. Otherwise, Definition 3 would not carry over asynchronous bisimulation, as for instance the open deadlocked solution $x\langle\rangle|y\langle\rangle \triangleright P \Vdash_{\{y\}}$ would be equivalent to the empty solution $\Vdash_\emptyset$.*

The relations $\approx$ (over open RCHAMs) and $\approx_a$ (over J-open RCHAMs) do match; this is a direct consequence of Proposition 3.

Theorem 2 $\approx_a = \approx$.

5 THE DISCRIMINATING POWER OF MATCHING

The semantics of the join-calculus in [7] is *barbed congruence*. We recall the definition, and refer to [7] for discussion.

Definition 6 *The* output barb $\Downarrow_x$ *is a predicate over open* RCHAMs *that tests for the potential emission on the free name x:*

$$\mathcal{S} \Downarrow_x \quad \stackrel{def}{=} \quad \exists S, \widetilde{v} \text{ such that } \mathcal{S} \rightleftharpoons^* \xrightarrow{S\overline{x}\langle\widetilde{v}\rangle}$$

A symmetric relation $\mathcal{R}$ *over open* RCHAMs *is a* barbed bisimulation *if whenever* $\mathcal{S}\ \mathcal{R}\ \mathcal{T}$ *the following holds:*

- *If* $\mathcal{S} \Downarrow_x$ *then* $\mathcal{T} \Downarrow_x$*;*
- *If* $\mathcal{S} \rightleftharpoons^* \rightarrow \rightleftharpoons^* \mathcal{S}'$ *then* $\mathcal{T} \rightleftharpoons^* \mathcal{T}'$ *and* $\mathcal{S}'\ \mathcal{R}\ \mathcal{T}'$.

Let $\dot{\approx}_b$ *be the largest barbed bisimulation. As usual, this induces an equivalence on open terms, also noted* $\dot{\approx}_b$: $A \dot{\approx}_b A' \stackrel{def}{=} \emptyset \vdash_\emptyset \{A\} \dot{\approx}_b \emptyset \vdash_\emptyset \{A'\}$.

A barbed congruence *is a barbed bisimulation that is also a congruence for open join-calculus contexts. Let* $\approx_b$ *be the largest barbed congruence.*

Weak bisimulation is a congruence and a barbed bisimulation, and thus a barbed congruence ($\approx \subset \approx_b$). This containment is strict, as can be seen from the paradigmatic example of barbed congruence:

$$x\langle z\rangle \quad \approx_b \quad \mathsf{def}\ u\langle v\rangle \triangleright z\langle v\rangle\ \mathsf{in}\ x\langle u\rangle$$

That is, emitting a free name z is the same as emitting a bound name u that forwards all the messages it receives to z, and the two processes are indistinguishable, up-to an extra internal move for every use of z. These two processes are distinguished by weak bisimulation, however, because their respective extrusion labels reveal that z is free and u is extruded.

Nevertheless, barbed congruence can easily be equipped with such a discriminating power, provided it can use richer contexts. To this end, we extend the syntax of the join-calculus with the standard name-matching prefix of [16].

$$A \stackrel{\text{def}}{=} \cdots \mid [x = y]\, A \qquad\qquad P \stackrel{\text{def}}{=} \cdots \mid [x = y]\, P$$

Accordingly, we extend our chemical machines with a new reduction rule.

$$\textsc{Match} \qquad\qquad \vdash [x = x]\, A \ \rightarrow \vdash A$$

Observe that barbed congruence can now separate $x\langle z\rangle$ from $\mathsf{def}\ u\langle v\rangle \triangleright z\langle v\rangle\ \mathsf{in}\ x\langle u\rangle$ by using the context $C[\,] = \mathsf{def}\ x\langle y\rangle \triangleright [y = z]\, a\langle\rangle\ \mathsf{in}\ [\,]$.

The usual drawback of name matching is that global renamings do not preserve weak bisimulation anymore: Corollary 2 is false in the extended calculus, and for instance $0 \approx [x = y]\, x\langle\rangle$, while after applying the renaming $\{^x/_y\}$, $0 \not\approx [x = x]\, x\langle\rangle$. Accordingly, bisimulation is not a congruence anymore; it is easy to check that the context $C[\,] \stackrel{\text{def}}{=} \mathsf{def}\ z\langle x, y\rangle \triangleright [\,]\ \mathsf{in}\ z\langle u, u\rangle$ separates the two processes above.

In order to retain the full congruence property, we may consider the coarsest equivalence contained into $\approx$ and closed under renaming. It is possible to prove, in the same style of Theorem 1, that this equivalence is indeed a congruence; however this equivalence is not a bisimulation.

Alternatively, we consider equivalences that are not full congruences. To this aim, we define the set of *evaluation contexts* $C[\,] \stackrel{\text{def}}{=} [\,] \mid C[\,]|P \mid \mathsf{def}\ D\ \mathsf{in}\ C[\,]$ up to structural congruence.

Lemma 4 $\approx$ *is a congruence with respect to evaluation contexts.*

We define *barbed equivalence* $\dot{\approx}_{be}$ over open join-calculus processes with matching as the largest barbed bisimulation that is a congruence for all evaluation contexts. We have

Theorem 3 $\dot{\approx}_{be} = \approx$.

6 A COMPARISON WITH THE π-CALCULUS

The join-calculus can be considered as a disciplined version of the asynchronous π-calculus, where the syntax enforces locality in the usage of names. Still, both calculi have a lot in common, and our semantics largely draw upon the bisimulations developed for the π-calculus [16, 17, 21, 11, 1]. In this section, we relate our definitions to previous proposals in the literature, and we compare the equivalences obtained by applying similar definitions to both the join-calculus and the π-calculus.

In the sequel, we use the following grammar for asynchronous π-calculus processes.

$$P \ ::= \ \mathsf{0} \ \mid \ \overline{x}\langle v\rangle \ \mid \ x(y).P \ \mid \ \nu x.P \ \mid \ !P \ \mid \ P|P$$

We refer to [5] for the definition of its semantics. We recall the basic reduction step, which matches complementary pairs of emission and reception, and substitutes actual names for the variables in the receiving process:

$$\overline{x}\langle v\rangle \mid x(y).Q \longrightarrow Q\{^{y}/_{v}\}$$

Other transitions render intrusion or extrusion of messages, and their labels carry the same information than those we use in the open RCHAM. In particular output labels explicitly mention names that are being extruded.

The join-calculus and the asynchronous π-calculus have the same expressive power, at least up-to barbed congruence [7]. Their treatment of names, however, are quite different. In the open join-calculus, we partition the visible names of a process into names for extrusions (*free names*), which do not have a local definition, and names for intrusions and internal reduction (*extruded names*), which do have a local definition. As a result, interaction with the environment has more structure, and this significantly prunes the number of transitions that are enabled, even before introducing join-message labels.

In the π-calculus, on the contrary, a free name can always be used for intrusion, extrusion and internal reduction. Furthermore, a received name can be re-defined, as in $x(y).y(z).P$. In this respect, the barbed equivalence of Section 5 is illuminating. Consider the rough π-calculus encodings of the processes mentioned in its example, namely $\overline{x}\langle z\rangle$ and $\nu u.(!u(v).\overline{z}\langle v\rangle \mid \overline{x}\langle u\rangle)$. These two processes are not barbed congruent, and this can be established using a context that invalidates our locality property. For example, if we choose the context

$$C[\cdot] \quad \stackrel{\text{def}}{=} \quad \nu x, z.(x(a).a(u).\overline{u}\langle\rangle \mid \overline{z}\langle b\rangle \mid [\cdot])$$

we obtain that $C[\overline{x}\langle z\rangle] \Downarrow_b$ and $C[\nu u.(!u(v).\overline{z}\langle v\rangle \mid \overline{x}\langle u\rangle)] \not\Downarrow_b$. As could be expected, locality restricts the discriminating power of observers.

We now compare the observational semantics for the open join calculus with previous proposals for the asynchronous π-calculus, notably [11] and [1]. In these works, a major issue is to accommodate the semantics inherited from the standard π-calculus [16] to asynchrony. As message-output is not a prefix anymore, emitters in contexts cannot detect whether a message is actually read. Technically, this leads to a special treatment of input actions, either in the definition of transitions or in the definition of bisimulation.

In [11], Honda and Tokoro take as observational semantics the standard notion of weak bisimulation. As a consequence, they are forced to change the intensional semantics. Let us discuss this issue with a paradigmatic example. Take for instance the π-calculus processes 0 and $x(u).\,\overline{x}\langle u\rangle$. In order to progress, the process $x(u).\,\overline{x}\langle u\rangle$ has to consume an emission $\overline{x}\langle v\rangle$, thus exhibiting a new emission $\overline{x}\langle v\rangle$. The same happens for the process 0 that "takes" $\overline{x}\langle v\rangle$ and "gives" the same $\overline{x}\langle v\rangle$ (just nothing is consumed or produced). Therefore 0 and $x(u).\,\overline{x}\langle u\rangle$ should be reasonably equated in an asynchronous regime: indeed, they are barbed congruent. However, standard bisimulation obviously discriminates between 0 and $x(u).\,\overline{x}\langle u\rangle$. To cope with this problem, Honda and Tokoro adopt an operational model where asynchrony of communication with the environment is rendered as the total receptiveness of the process. In other words, any message can be intruded from the environment at any moment, using extended structural equivalence in combination with the rule:

$$\textsc{Input} \qquad\qquad 0 \xrightarrow{x\langle v\rangle} \overline{x}\langle v\rangle \ .$$

The intrusion rule of the open-RCHAM is reminiscent of this kind of operational semantics, with two important differences: (i) Our rule INT can be used only with names that have previously been extruded, and there are finitely many of them; conversely, their input rule immediately yields an infinite transition system; and (ii) extraneous inputs in the join calculus are not observable: if they are not used by the process, they can safely be garbage-collected, as we are not faced with the problem of contexts that attempt to read back their emissions.

In [1], Amadio *et al.* take the opposite approach. They keep the standard synchronous semantics, and they modify the notion of bisimulation. For two processes to be bisimilar, they do not require that every input on one side be necessarily simulated by an input of the other side. Rather, they introduce a *delay* clause in bisimulation, by which an input can be simulated by adding a new message in parallel on the other side. This is crucial, for example, for proving that 0 is bisimilar to $x(u).\,\overline{x}\langle u\rangle$. Their resulting asynchronous bisimulation offers three advantages: it eliminates the need for total receptiveness, it is consistent with external sum, and it relies on a widely-used semantics.

Our asynchronous bisimulation presented in Section 4 relies on similar motivations—handling asynchrony in the definition of bisimulation with a relaxed clause for intrusions—, but the clause we use is different from theirs: we deal with multiple intrusions, and we allow the simulating process to perform arbitrary internal moves after parallel composition with the intruded messages; in their terminology, this would place our equivalence between 1-bisimulation and asynchronous bisimulation. Notice, also, that the J-open RCHAM equipped with asynchronous bisimulation is not meant to be a standard semantics, but rather a technical device to reduce the branching of the underlying transition system. Arguably, the open RCHAM gives a more intuitive meaning to extruded names.

From a technical point of view, we often discuss the same, standard properties (stability through renaming, congruence, relation with barbed bisimulation); as we compare our work to the formal treatment of bisimulation in the π-calculus, we observe that locality makes most proofs simpler: it forbids the detection of messages carried on defined names, and reduces the number of cases to consider in the interaction between transitions and global renamings.

Weak bisimulation and asynchronous bisimulation coincide in the open join-calculus, for simple reasons. In contrast, the correspondence between the bisimulations of [11] and those of [1], discussed in details in the latter, is a delicate issue; it is unclear, for instance, whether both approaches yield the same relation in the weak case.

Independently, several notions of locality somehow related to the join-calculus have also been considered in a pure π-calculus setting. In [2], a fragment of the π-calculus is considered where received names cannot be used for input (i.e., cannot be redefined) and it is proven that this language can be compositionally compiled into a very simple, substitution-free calculus. The condition of transmitting only names with output capabilities is further strengthened into *uniform receptiveness* in [20] by also demanding that names are available in input-replicated form as soon as created. Uniform receptiveness is actually the locality property, which is formulated by syntactic means in join-calculus and by means of a typing system in [20] for the π-calculus.

7 CONCLUSIONS

In this paper we have developed the theory of bisimulation for the join-calculus. We started from the initial definitions of [7], which are purely based on internal reductions and observational equivalence; we provided extensions of the RCHAM that encompass intrusions and extrusions, in two different styles, and we studied for each style a relevant definition of bisimulation.

Asynchronous bisimulation, and more specifically its ground variant, is best suited for proofs, as it relies on smaller synchronization trees. In contrast with barbed congruence, bisimulation is therefore "easy" to check, and should be amenable to automated verification by means of the existing algorithms (see, for instance, [22] and [18]).

As regards the comparison between the π-calculus and the join-calculus, there are fully abstract encodings up to barbed congruence [7], but it would be interesting to know what is the status between the open join-calculus equipped with weak bisimulation and the asynchronous π-calculus.

The precise role of name-testing deserve further investigation. The coincidence of weak bisimilarity and barbed equivalence only holds in the presence of matching, which invalidates useful process equalities. Thus, numerous useful properties of the join-calculus as a programming language are not directly captured by weak bisimulation. Instead, they may be proved by using first an encoding of name-testing, then weak bisimulation techniques. Of course, it would be interesting to find direct, purely co-inductive characterization of barbed equivalence in the absence of matching. This issue is addressed in [4] in a π-calculus setting. Finally, one may wonder whether our approach carries over the full distributed join-calculus of [8].

Acknowledgments. We thank Jean-Jacques Lévy, Uwe Nestmann, and especially Georges Gonthier for their fruitful comments.

REFERENCES

[1] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. RR 2913, INRIA, June 1996. A shorter version appeared in [19].

[2] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. In Montanari and Sassone [19].

[3] M. Boreale, C. Fournet, and C. Laneve. Bisimulations in the join-calculus. Available electronically (`ftp://ftp.cs.unibo.it/pub/laneve/equiv.ps.gz`), Oct. 1997.

[4] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. Submitted for publication, Dec. 1997.

[5] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[6] C. Fournet. *A calculus for distributed programming.* PhD thesis, Ecole Polytechnique, Palaiseau, 1998. Forthcoming.

[7] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the

join-calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.
[8] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In Montanari and Sassone [19].
[9] C. Fournet, C. Laneve, L. Maranget, and D. Rémy. Implicit typing à la ML for the join-calculus. In Mazurkiewicz and Winkowski [13].
[10] C. Fournet and L. Maranget. The join-calculus language. Available electronically (`http://pauillac.inria.fr/join`), June 1997.
[11] K. Honda and M. Tokoro. On asynchronous communication semantics. In P. Wegner, M. Tokoro, and O. Nierstrasz, editors, *Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing*, volume 612 of *LNCS*, pages 21–51, 1992.
[12] C. Laneve. May and must testing in the join-calculus. Technical Report UBLCS 96-04, University of Bologna, March 1996. Revised: May 1996.
[13] A. Mazurkiewicz and J. Winkowski, editors. *Proceedings of the 8th International Conference on Concurrency Theory*, volume 1243 of *LNCS*, Warsaw, Poland, 1–4 July 1997.
[14] R. Milner. *Communication and Concurrency.* Prentice Hall, New York, 1989.
[15] R. Milner. The polyadic π-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification.* Springer-Verlag, 1993.
[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] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695, Vienna, 1992.
[18] U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In I. Lee and S. A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory*, volume 962 of *LNCS*, 1995.
[19] U. Montanari and V. Sassone, editors. *Proceedings of the 7th International Conference on Concurrency Theory*, volume 1119 of *LNCS*, Pisa, Italy, 26–29 Aug. 1996.
[20] D. Sangiorgi. The name discipline of uniform receptiveness. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *LNCS*, pages 303–313, 1997. Also Technical Report, INRIA Sophia-Antipolis, December 1996.
[21] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In W. R. Cleaveland, editor, *Proceedings of CONCUR'92*, volume 630 of *LNCS*, pages 32–46, 1992.
[22] B. Victor and F. Moller. The mobility workbench – a tool for the π-calculus. In *CAV'94*, volume 818 of *LNCS*, pages 428–440, 1994.

Michele Boreale is a researcher at Università "La Sapienza", Rome, where he got the PhD in 1995. His research interests concern formal methods for the specification and the verification of concurrent systems.
Cédric Fournet is a PhD student at INRIA-Rocquencourt. His main research interests are process calculi, concurrent and functional programming, types.
Cosimo Laneve is a researcher at University of Bologna. He got the PhD in 1993 at the University of Pisa. His research interests include semantics and implementation of functional and concurrent languages, types.