Distributed Processes and Location Failures

(Extended Abstract)

James Riely and Matthew Hennessy*

Abstract

Site failure is an essential aspect of distributed systems; nonetheless its effect on programming language semantics remains poorly understood. To model such systems, we define a process calculus in which processes are run at distributed *locations*. The language provides operators to kill locations, to test the status (dead or alive) of locations, and to spawn processes at remote locations. Using a variation of bisimulation, we provide alternative characterizations of strong and weak barbed congruence for this language, based on an operational semantics that uses *configurations* to record the status of locations. We then derive a second, symbolic characterization in which configurations are replaced by logical formulae. In the strong case the formulae come from a standard propositional logic, while in the weak case a temporal logic with past time modalities is required. The symbolic characterization establishes that, in principle, barbed congruence for such languages can be checked efficiently using existing techniques.

1 Introduction

Many semantic theories have been proposed for concurrent processes [18, 16, 6]. Although these theories have been fruitfully applied to the analysis of some distributed systems, for the most part they ignore an essential feature of such systems, namely their *distribution*.

As a simple example consider two implementations of a client-server application in which the client can demand an interactive service provided by the server, such as previewing or updating a document. In one implementation (System A) the server spawns a process to handle the document at its own site, the remote location, and the client previews the document remotely. In the other (System B) the server sends a process, including the document, to the client site, and the client previews the document locally. Using the semantic theories mentioned above it would be difficult to distinguish between these implementations, as the only difference between them is the location at which activity occurs. We aim to develop a useful *extensional* theory of systems which would take this type of property into account.

*Research funded by EPSRC project GR/K60701. Authors' address: School of Cognitive and Computing Sciences, Univ. of Sussex, Falmer, Brighton, BN1 9QH, UK, {jamesri,matthewh}@cogs.susx.ac.uk
Acknowledgement: We thank Flavio Corradini and Alan Jeffrey; both made important comments and suggestions in the early stages of this work.

In [8, 20, 10] such theories have been proposed. All of these theories, however, are based on a very strong assumption: that an observer, or user, can determine the location at which every action is performed. Here we start from a weaker premise: that in distributed systems sites are liable to *failure*. The model of failure we have adopted is a *fail stop* model in which failures are independent of each other and the number of failures that can occur is unbounded. Assuming that sites can fail, it is easy to see that Systems A and B, outlined above, are indeed different: if, after the client has begun interaction with the document, a failure occurs at the remote site, then in System A the client deadlocks, while in System B it can continue operation unaffected.

Our work is motivated by the papers [2, 12]. In these papers, distributed languages with location failures are defined and shown to be very expressive. In both of these papers, the semantics is based on *barbed equivalence*, which requires quantification over all program contexts and thus is difficult to use directly. In each of the cited works, the authors provide a translation from their language into a simpler (non-distributed) language and prove that the translations are adequate or fully abstract in some sense. While these translations provide theoretical results about the relative expressiveness of distributed and interleaving calculi, they are sufficiently complicated to make reasoning about examples, even simple ones, very difficult.

By restricting attention to an asynchronous language, Amadio [4] has recently improved on the results of [2], providing simpler translations. Although our work developed independently of [4], the language we study has much in common with the language developed there. The main difference is that our language has no value-passing, allowing us to concentrate on the effects of location failure and simplifying the statement of many of our results. Since the issues raised by failures and value passing are largely independent, this paper may be seen as providing two extensional views of a language similar to Amadio's; the first of these is concrete, as is his translation, the second is more abstract.

In Section 2, we consider a simple language for *located processes* based on pure CCS [18], with which we assume familiarity. For example $(a.p)_\ell$ is a process located at ℓ which, if ℓ is alive, may perform the action a and then behave as $(p)_\ell$. In addition to the usual operators of CCS we have the following new operators: $\mathsf{spawn}(\ell, p)$ which starts process p running at location ℓ; $\mathsf{kill}\,\ell.p$ which, if location ℓ is alive, kills ℓ (with the result that any process located at ℓ is deactivated) and then behaves as p; and $\mathsf{if}\ \ell\ \mathsf{then}\ p\ \mathsf{else}\ q$ which silently evolves to either p or q, depending on whether ℓ is alive or dead when the test is performed.

We give an operational semantics for this language in terms of a labelled transition system. The judgments depend on a set L, of *live* locations, and are of the form $L \triangleright P \xmapsto{\alpha} L' \triangleright P'$, where P and P' are located processes and α is either a visible action, which permits synchronization, or the internal action τ. To decide on an appropriate equivalence between process terms we follow the approach advocated in [22]. We define both strong and weak barbed equivalence between processes, $\dot{\sim}$ and $\dot{\approx}$. We then dictate that the required equivalence, which we refer to as *barbed bisimulation equivalence*, is defined (for example in the weak case) as: $P \approx Q$ if and only if for every suitable context $\mathbb{C}[\,]$, $\mathbb{C}[P] \dot{\approx} \mathbb{C}[Q]$. Although this may be reasonable, it is not a very useful definition; the reader is invited to determine whether the following pairs of processes should be equivalent or distinguished.

$$\begin{aligned} P_1 &= \left[(\alpha)_\ell \,|\, (\overline{\alpha}+\tau.a)_k\right]\backslash\alpha & \qquad P_2 &= [(\mathsf{if}\ k\ \mathsf{then}\ \alpha\ \mathsf{else\ nil})_\ell \,|\, (\overline{\alpha}.a)_k]\backslash\alpha \\ Q_1 &= \left[(\alpha+\tau)_\ell \,|\, (\overline{\alpha}.a)_k\right]\backslash\alpha & \qquad Q_2 &= (\mathsf{spawn}(k,a))_\ell \end{aligned}$$

In Section 3 we define two bisimulation-based relations, strong and weak *Located-Failure equivalence* (*LF-equivalence*) and show that these coincide with the indirectly defined barbed congruences. Since LF-equivalence is defined using bisimulations, the problem of deciding that two systems are semantically congruent can, in principle, be solved using standard proof techniques associated with bisimulation [18]. However, constructing an LF-bisimulation requires that one consider the behavior of the systems under all possible sequences of kills, by both the systems themselves and the environment. The number of states that must be explored may be exponentially larger than the number needed to construct a CCS bisimulation.

In Section 4 we use the ideas of [15] to give alternative *symbolic* characterizations of LF-equivalence that can be decided using a much smaller state space. The idea is to replace the operational judgments $L \triangleright P \overset{\alpha}{\longmapsto} L' \triangleright P'$ with judgments of the form $P \xrightarrow[\varphi]{\alpha} P'$, where φ is a logical formula that describes the circumstances under which the action α can be performed. In the strong case the required logic is straightforward: a propositional logic that describes the state (dead or alive) of the sites in the system. In the weak case, however, we require a more complicated logic that can express statements of the form *site ℓ was alive at some point in the past*. Using these symbolic transitions, the standard definition of *symbolic bisimulation* [15] requires only minor modification to capture $\simeq$ and $\approxeq$; hence the symbolic proof techniques and tools of [15] may be used to check the new semantic equivalences proposed in this paper.

In this extended abstract we have omitted several formal definitions and all proofs. The full version [21] includes additional results and examples, including a discussion of basic processes and comparisons with other equivalences.

2 The Language

The syntax of processes is parameterized with respect to several syntactic sets. We assume a set *Loc* of *locations* k, ℓ, m, a set *PConst* of *process constants* A used to define recursive processes, and a set *Act* of *communication actions* a, b, c, such that every action $a \in Act$ has a complement $\overline{a} \in Act$ ($\overline{\cdot}$ is a bijection on *Act*). The set $Act_\tau = Act \cup \{\tau\}$ of *actions* α includes also the distinguished *silent action* τ. The formal syntax is as follows. Most of the operators should be familiar from CCS; all of the new constructs have been described in the introduction.

$$\begin{aligned} p,q\ (\in BProc) &::= \alpha.p \;\Big|\; \mathsf{spawn}(\ell,p) \;\Big|\; \mathsf{kill}\,\ell.p \;\Big|\; \mathsf{if}\ \ell\ \mathsf{then}\ p\ \mathsf{else}\ q \;\Big|\; A \;\Big|\; \textstyle\sum_{i\in I} p_i \\ &\quad\;\Big|\; p \,|\, q \;\Big|\; p\backslash a \;\Big|\; p[f] \\ P,Q\ (\in LProc) &::= P \,|\, Q \;\Big|\; P\backslash a \;\Big|\; P[f] \;\Big|\; (p)_\ell \end{aligned}$$

We have adopted a two-level syntax which distinguishes between *basic* processes p and *located* processes P. Intuitively, a basic process corresponds to what one normally thinks of as a *process*: a collection of threads of computation that must be run at a single site. A located process, instead, corresponds to a *distribution* of basic processes over several sites.

Note that many basic processes may be located at a single site, and a basic process may share a private channel (unknown to other basic processes running at the same site) with a remote process.

The ability of a process to perform an action is dependent on the set of live locations, and consequently the transition relation determining the operational semantics is defined between *configurations*. A *liveset* L is any subset of Loc. A *configuration* $(L \triangleright P)$ is a pair comprising a liveset L and a located process term P. The set of all configurations is $Config$, ranged over by C and D.

In giving the intensional semantics of processes, it will be convenient for later development if we distinguish executions of the operator $\mathsf{kill}\,\ell.p$ depending upon whether ℓ is alive or dead at the time of execution. To capture this distinction, we extend the set of actions to the set $KAct = Act \cup \{kill\,\ell \mid \ell \in Loc\}$, which includes the *kill actions* $kill\,\ell$. Unless otherwise specified, μ ranges over $KAct_\tau = KAct \cup \{\tau\}$. In Table 1 we define the transition relation $(\xrightarrow{\mu}) \subseteq Config \times Config$. The definition uses the following simple structural equivalence on processes:

$$(p \,|\, q)_\ell \equiv (p)_\ell \,|\, (q)_\ell \qquad (p\backslash a)_\ell \equiv (p)_\ell\backslash a \qquad (p[f])_\ell \equiv (p)_\ell[f]$$

While the transition relation $\longrightarrow$ distinguishes effective kill actions from those that have no effect, a basic tenet of our study is that the precise moment of location failure should be unobservable. Thus we extract from $\longrightarrow$ a transition relation $\longmapsto$ in which all kill actions have been replaced with silent actions. It is this derived relation $\longmapsto$ that we take to be fundamental.

Definition 1 ($\longmapsto$). $C \overset{a}{\longmapsto} C'$ iff $C \xrightarrow{a} C'$ □
$C \overset{\tau}{\longmapsto} C'$ iff $C \xrightarrow{\tau} C'$ or $\exists k\colon\ C \xrightarrow{kill\,k} C'$

Most of the rules in Table 1 are straightforward, being inherited directly from CCS, modulo the constraint that the process $(p)_\ell$ can only move if ℓ is alive. Note that the three new operators — kill, spawn and the conditional — are modeled as τ-transitions; this reflects the fact that in a distributed system the implementation of these operators would involve some computation and thus the passage of some time.

We now discuss the problem of defining an appropriate semantic equivalence for located processes, based on the transition relation $\longmapsto$. An obvious possibility is to adapt the bisimulation equivalences of CCS [18]. (Strong) CCS-*bisimulation* is the largest symmetric relation $\stackrel{.}{\sim}^{\text{ccs}}$ on configurations such that whenever $C \stackrel{.}{\sim}^{\text{ccs}} D$ and $C \overset{\alpha}{\longmapsto} C'$ there exists a D' such that $D \overset{\alpha}{\longmapsto} D'$ and $C' \stackrel{.}{\sim}^{\text{ccs}} D'$. A weak version of this relation, $\stackrel{.}{\approx}^{\text{ccs}}$, can be obtained by adapting this definition to the *weak* transition relation $\Longmapsto$, defined as usual. To see that CCS-bisimulation is not suitable for our language, for example is not a congruence, consider the processes $P_3 = \big[(\alpha.a)_\ell \,|\, (\overline{\alpha})_k\big]\backslash\alpha$ and $Q_3 = \big[(\alpha)_\ell \,|\, (\overline{\alpha}.a)_k\big]\backslash\alpha$. $P_3 \stackrel{.}{\sim}^{\text{ccs}} Q_3$, but P_3 and Q_3 can be distinguished by a context that kills location ℓ, if this kill action is performed after the initial communication on α.

The use of $\stackrel{.}{\approx}^{\text{ccs}}$ for CCS has been justified in [22] by the fact that it coincides with the congruence obtained from a simple notion of observation called *barbed bisimulation*. Similar results have been obtained for lazy and eager functional languages [1, 14, 7], giving further evidence for the reasonableness of this approach. Roughly, two processes are barbed

Table 1 Transition system with configurations (symmetric rules for $|$ omitted)

Act_c)	$L \triangleright (\alpha.p)_\ell \xrightarrow{\alpha} L \triangleright (p)_\ell$	if	$\ell \in L$
Spawn_c)	$L \triangleright (\mathsf{spawn}(k,p))_\ell \xrightarrow{\tau} L \triangleright (p)_k$	if	$\ell \in L$
$\mathsf{Kill1}_c$)	$L \triangleright (\mathsf{kill}\, m.p)_\ell \xrightarrow{kill\, m} L \setminus \{m\} \triangleright (p)_\ell$	if	$\ell \in L, m \in L$
$\mathsf{Kill2}_c$)	$L \triangleright (\mathsf{kill}\, m.p)_\ell \xrightarrow{\tau} L \triangleright (p)_\ell$	if	$\ell \in L, m \notin L$
$\mathsf{Cond1}_c$)	$L \triangleright (\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q)_\ell \xrightarrow{\tau} L \triangleright (p)_\ell$	if	$\ell \in L, m \in L$
$\mathsf{Cond2}_c$)	$L \triangleright (\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q)_\ell \xrightarrow{\tau} L \triangleright (q)_\ell$	if	$\ell \in L, m \notin L$
Sum_c)	$L \triangleright (\sum_{i \in I} p_i)_\ell \xrightarrow{\mu} L' \triangleright (p'_j)_k$	if	$L \triangleright (p_j)_\ell \xrightarrow{\mu} L' \triangleright (p'_j)_k,\ j \in I$
Def_c)	$L \triangleright (A)_\ell \xrightarrow{\mu} L' \triangleright (p')_k$	if	$L \triangleright (p)_\ell \xrightarrow{\mu} L' \triangleright (p')_k,\ A \overset{def}{=} p$
Str_c)	$L \triangleright P \xrightarrow{\mu} L' \triangleright Q$	if	$P \equiv P', L \triangleright P' \xrightarrow{\mu} L' \triangleright Q', Q' \equiv Q$
Par_c)	$L \triangleright P \mid Q \xrightarrow{\mu} L' \triangleright P' \mid Q$	if	$L \triangleright P \xrightarrow{\mu} L' \triangleright P'$
Comm_c)	$L \triangleright P \mid Q \xrightarrow{\tau} L' \triangleright P' \mid Q'$	if	$L \triangleright P \xrightarrow{a} L' \triangleright P', L \triangleright Q \xrightarrow{\bar{a}} L' \triangleright Q'$
Restr_c)	$L \triangleright P \backslash a \xrightarrow{\mu} L' \triangleright P' \backslash a$	if	$L \triangleright P \xrightarrow{\mu} L' \triangleright P', \mu \notin \{a, \bar{a}\}$
Ren_c)	$L \triangleright P[f] \xrightarrow{f(\mu)} L' \triangleright P'[f]$	if	$L \triangleright P \xrightarrow{\mu} L' \triangleright P'$

bisimilar if every silent transition of one can be matched by a silent transition of the other in such a way that the derived states are capable of exactly the same observable actions; in addition, the derived states must also be barbed bisimilar. For our language, the formal definition is as follows.

Definition 2 (Barbed bisimulation). Weak *barbed bisimilarity* ($\dot{\approx}$) is the largest symmetric relation over configurations such that whenever $C \mathrel{\dot{\approx}} D$: (a) $C \stackrel{\tau}{\longmapsto} C'$ implies that for some D', $D \stackrel{\varepsilon}{\Longmapsto} D'$ and $C' \mathrel{\dot{\approx}} D'$; and (b) for every a, $C \stackrel{a}{\longmapsto}$ implies $D \stackrel{a}{\Longmapsto}$. Strong barbed bisimilarity ($\dot{\sim}$) is obtained by replacing $\Longmapsto$ by $\longmapsto$ everywhere in the definition. $\square$

Barbed bisimulation is a very weak relation; for example, it is not preserved by parallel composition. However, by closing over all contexts we arrive at a reasonable semantic equivalence that by definition enjoys an important property, namely that it is a congruence.

Definition 3 (Barbed equivalence). Located processes P and Q are (weak) *barbed equivalent* ($P \approx Q$) if for every context $\mathbb{C}[\ ]$ such that $\mathbb{C}[P]$ and $\mathbb{C}[Q]$ are configurations, $\mathbb{C}[P] \mathrel{\dot{\approx}} \mathbb{C}[Q]$. Strong barbed equivalence ($\sim$) is obtained in the same manner from $\dot{\sim}$. $\square$

Because it requires quantification over all contexts, barbed equivalence is difficult to use directly. For example the processes P_1 and Q_1, given in the introduction, are distinguished by $\approx$ whereas P_2 and Q_2 are identified; it is far from obvious why. Even worse, processes P_5 and Q_5 (given in Section 3) are related, although establishing this fact requires that one prove that P_1 and Q_1 are *related* under the assumption that ℓ is alive at the time P_1 and Q_1 are compared, that is, ℓ is *initially alive*.

We end this section with some additional, simpler examples. The processes $(a)_\ell \mid (b)_k$ and $(b)_\ell \mid (a)_k$ can be distinguished by a context that kills ℓ. The same context can be used to distinguish the basic processes $\mathsf{spawn}(\ell, a)$ and $\mathsf{spawn}(k, a)$, regardless of where they are located. These examples indicate that although the location of an action is not reflected directly in the operational semantics they do impinge on the behavior of processes. The order in which kill actions are executed is also significant. For example $\mathsf{kill}\,\ell.\mathsf{kill}\,k$ can be distinguished from $\mathsf{kill}\,k.\mathsf{kill}\,\ell$ using the process $(a)_\ell \mid (b)_k$.

3 Located-Failures Equivalence

In this section and the next, we provide alternative characterizations of barbed equivalence for our language. Note that if $L \triangleright P \xrightarrow{\mu} L' \triangleright P'$, then L' is determined by L and μ. To emphasize this, we adopt the following notation. For each action μ, we define a function "iafter$_\mu$" which reflects the immediate effect of action μ on a liveset. We also define the relations $\xrightarrow[L]{\mu}$ and $\xRightarrow[L]{\mu}$ on process terms, which capture the capability of action μ under liveset L.

$$\mathrm{iafter}_\mu(L) \overset{def}{=} \begin{cases} L \setminus \{k\}, & \text{if } \mu = kill\,k \\ L, & \text{if } \mu \in Act \cup \{\tau, \varepsilon\} \end{cases} \quad \left| \quad \begin{array}{l} P \xrightarrow[L]{\mu} P' \overset{def}{\Leftrightarrow} L \triangleright P \xrightarrow{\mu} \mathrm{iafter}_\mu(L) \triangleright P' \\ P \xRightarrow[L]{\mu} P' \overset{def}{\Leftrightarrow} L \triangleright P \xRightarrow{\mu} \mathrm{iafter}_\mu(L) \triangleright P' \end{array} \right.$$

For example, $\mathrm{iafter}_\alpha(L) = L$ for any α, and $\mathrm{iafter}_{kill\,\ell}(\{\ell, k\}) = \{k\}$. If $P = (\alpha.a)_\ell \mid (\overline{\alpha})_k$, then $P \xRightarrow[\{\ell,k\}]{a} \mathsf{nil}$, but P has no a-transition under the liveset $\{k\}$.

We first present the strong case.

Definition 4 (Strong LF-equivalence). Let $\mathcal{S} = \{\mathcal{S}_L\}_{L \subseteq Loc}$ be an indexed family of relations on $LProc$. $\mathcal{S}$ is a *strong* LF-*bisimulation* if for every L, $\mathcal{S}_L$ is symmetric and whenever $P \;\mathcal{S}_L\; Q$:

$$\begin{array}{ll} \text{(a)} & P \xrightarrow[L]{\mu} P' \text{ implies } \exists Q' :\; Q \xrightarrow[L]{\mu} Q' \text{ and } P' \;\mathcal{S}_{\mathrm{iafter}_\mu(L)}\; Q' \\ \text{(b)} & \text{for every } k \in L \quad P \;\mathcal{S}_{L \setminus \{k\}}\; Q \end{array}$$

P and Q are strong LF-*equivalent under* L ($P \simeq_L Q$) if there exists a strong LF-bisimulation $\mathcal{S}$ with $P \;\mathcal{S}_L\; Q$.

P and Q are strong LF-*equivalent* ($P \simeq Q$), if $P \simeq_L Q$ for every subset L of Loc. □

In the full paper, we prove that $\simeq$ and $\sim$ coincide. The alternative characterization of weak barbed equivalence is more complicated: it is not sufficient to change the strong arrows in Definition 4 to weak arrows. To see this, consider the following processes:

$$\begin{array}{ll} P_5 = \big[\big(b.\beta.\alpha + b.(\alpha+\tau)\big)_\ell \big| \big(\overline{\beta}.(\overline{\alpha}+\tau.a) + \overline{\alpha}.a\big)_k\big] \setminus\alpha\setminus\beta \\ Q_5 = \qquad\quad \big[\big(b.(\alpha+\tau)\big)_\ell \big| \qquad\qquad\qquad\quad (\overline{\alpha}.a)_k\big] \setminus\alpha \end{array}$$

If ℓ is initially dead, P_5 and Q_5 are clearly equivalent: both are strong equivalent to nil. If ℓ is initially alive, however, the situation is not so clear. The questionable move is P_5's b-transition to $P_1 \simeq \big[(\alpha)_\ell \mid (\overline{\alpha}+\tau.a)_k\big]\setminus\alpha$. To match this move Q_5 must perform a weak b-transition to $Q_1 \simeq \big[(\alpha+\tau)_\ell \mid (\overline{\alpha}.a)_k\big]\setminus\alpha$. But P_1 and Q_1 are not barbed equivalent: if ℓ

is dead, then Q_1 is capable of a a transition that P_1 cannot match. This would lead one to believe that P and Q are *not* barbed equivalent; however, they are.

Intuitively this is true because when P_5 reaches P_1, ℓ must be alive; thus P_1 and Q_1 need only be compared under the constraint that ℓ is initially alive. Once this comparison has begun, the environment can distinguish Q_1 from P_1 only by killing ℓ, but it cannot control internal activity on the part of P_1 before ℓ is dead.

Definition 5 (Weak LF-equivalence). For $\mu \in Act_\tau$, define $\widehat{\mu}$ such that $\widehat{a} = a$ and $\widehat{\tau} = \varepsilon$. The definition of $\approxeq$ is similar to that for $\simeq$, except that when $P \; \mathcal{S}_L \; Q$, we require:

$$\begin{aligned}
&\text{(a)}\;\; P \xrightarrow[L]{\mu} P' \text{ implies } \exists Q'\colon\; Q \overset{\widehat{\mu}}{\underset{L}{\Longrightarrow}} Q' \text{ and } P' \; \mathcal{S}_{\mathrm{iafter}_\mu(L)} \; Q' \\
&\text{(b)}\;\; \text{for every } k \in L \quad \exists Q'\colon\; Q \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\backslash\{k\}}{\Longrightarrow}} Q' \text{ and } P \; \mathcal{S}_{L\backslash\{k\}} \; Q'
\end{aligned} \qquad \square$$

Whereas the first clause in the definition of weak LF-bisimulation is as one would expect, the second clause is somewhat surprising. It says, in effect, that if the environment kills a location k, then Q must be able to (silently) evolve to a process Q' that matches P; but in reaching Q', Q may exploit the intermediate states of the system (that is, k alive, then k dead).

Theorem 6. *For all located processes $P \approxeq Q$ if and only if $P \approx Q$.* $\square$

4 Symbolic characterizations

While the LF-equivalences provide a great deal of insight into the meaning of barbed equivalence in distributed process description languages such as ours, they are unwieldy to use in practice. For the most part, this is due to the use of configurations in the operational semantics. In this section, we improve this situation by defining a *symbolic* transition system directly on located process terms, then giving characterizations of strong and weak LF-equivalence using these symbolic transitions. As one should expect, the weak case is quite a bit more subtle than the strong.

We begin by giving the symbolic operational semantics. The symbolic transition relation makes use of propositional formulae π, ρ, which are given a semantics in terms of livesets. Intuitively, a formula indicates a set of constraints on the status of locations (dead or alive) at the time that the transition is enabled. If $P \xrightarrow[\bar{0}\wedge 1]{\mu} P'$ then if location 0 is dead and 1 is alive, P is capable of making an μ-transition to P'; that is, if $0 \notin L$ and $1 \in L$ then $P \xrightarrow[L]{\mu} P'$. In Table 2 we define the transition relation $\xrightarrow[\pi]{\mu} \subseteq LProc \times LProc$. The two transition systems are related by the fact that $P \xrightarrow[L]{\mu} P'$ if and only if there exists a π such that $P \xrightarrow[\pi]{\mu} P'$ and $L \vDash \pi$.

The standard definition of symbolic bisimulation [15] requires that we define entailment between formulae, which we do in the standard way:

$$\pi \Vdash \rho \;\text{ iff }\; \forall L\colon\; L \vDash \pi \text{ implies } L \vDash \rho$$

Note that entailment is a preorder on formulae. If $\pi \Vdash \rho$ we say that π is *stronger* than ρ. $\mathtt{ff}$ is the strongest formula under $\Vdash$, $\mathtt{tt}$ the weakest.

Table 2 Symbolic transition system (symmetric rules for $|$ omitted)

$\mathsf{Act_s}$)	$(\alpha.p)_\ell \xrightarrow[\ell]{\alpha} (p)_\ell$		
$\mathsf{Spawn_s}$)	$(\mathsf{spawn}(k,p))_\ell \xrightarrow[\ell]{\tau} (p)_k$		
$\mathsf{Kill1_s}$)	$(\mathsf{kill}\, m.p)_\ell \xrightarrow[\ell\wedge m]{kill\, m} (p)_\ell$		
$\mathsf{Kill2_s}$)	$(\mathsf{kill}\, m.p)_\ell \xrightarrow[\ell\wedge \overline{m}]{\tau} (p)_\ell$		
$\mathsf{Cond1_s}$)	$(\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q)_\ell \xrightarrow[\ell\wedge m]{\tau} (p)_\ell$		
$\mathsf{Cond2_s}$)	$(\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q)_\ell \xrightarrow[\ell\wedge \overline{m}]{\tau} (q)_\ell$		
$\mathsf{Sum_s}$)	$(\sum_{i\in I} p_i)_\ell \xrightarrow[\pi]{\mu} (p'_j)_\ell$	if	$(p_j)_\ell \xrightarrow[\pi]{\mu} (p'_j)_\ell,\ j\in I$
$\mathsf{Def_s}$)	$(A)_\ell \xrightarrow[\pi]{\mu} (p')_\ell$	if	$(p)_\ell \xrightarrow[\pi]{\mu} (p')_\ell,\ A \overset{def}{=} p$
$\mathsf{Str_s}$)	$P \xrightarrow[\pi]{\mu} Q$	if	$P\equiv P',\ P' \xrightarrow[\pi]{\mu} Q',\ Q'\equiv Q$
$\mathsf{Par_s}$)	$P\,\vert\,Q \xrightarrow[\pi]{\mu} P'\,\vert\,Q$	if	$P \xrightarrow[\pi]{\mu} P'$
$\mathsf{Comm_s}$)	$P\,\vert\,Q \xrightarrow[\pi\wedge\rho]{\tau} P'\,\vert\,Q'$	if	$P \xrightarrow[\pi]{a} P',\ Q \xrightarrow[\rho]{\overline{a}} Q'$
$\mathsf{Restr_s}$)	$P\backslash a \xrightarrow[\pi]{\mu} P'\backslash a$	if	$P \xrightarrow[\pi]{\mu} P',\ \mu\notin\{a,\overline{a}\}$
$\mathsf{Ren_s}$)	$P[f] \xrightarrow[\pi]{f(\mu)} P'[f]$	if	$P \xrightarrow[\pi]{\mu} P'$

We must also identify a set of formulae suitable as parameters in the recursive definition of symbolic equivalence, that is, the analogs of the parameters L in the definition of LF-equivalence. Intuitively, when we say that P and Q are LF-equivalent under L, we are limiting attention to a single possible world, namely that in which exactly the sites in L are alive. The idea of symbolic equivalences, instead, is to treat many possible worlds simultaneously (via entailment). In the case of strong LF-bisimulation, where $P \simeq_L Q$ and $M \subseteq L$ imply $P \simeq_M Q$, this is achieved by restricting attention to *negative formulae* — formulae which contain no positive atoms — in the recursive definition of symbolic equivalence. Finally, we identify a transformation on formulae (indexed by actions) which specifies the conditions under which residual processes are to be compared:

$$M \vDash \mathsf{after}_\alpha(\rho)\ \text{ iff }\ \exists L:\ L \vDash \rho \text{ and } M \subseteq L$$
$$M \vDash \mathsf{after}_{kill\,k}(\rho)\ \text{ iff }\ \exists L:\ L \vDash \rho \text{ and } M \subseteq L\backslash\{k\}$$

Definition 7 (Strong symbolic bisimulation). Let $\mathcal{S}$ be a family of relations on *LProc* indexed by negative formulae ϑ. $\mathcal{S}$ is a *strong symbolic bisimulation* if for every ϑ, $\mathcal{S}_\vartheta$ is symmetric and whenever $P\ \mathcal{S}_\vartheta\ Q$ and $P \xrightarrow[\pi]{\mu} P'$ then for some π_i, ρ_i, and Q_i:

$$\begin{array}{llll} \text{(a)} & \vartheta\wedge\pi \Vdash \bigvee_i \rho_i, & \text{(c)} & Q \xrightarrow[\pi_i]{\mu} Q_i, \text{ and} \\ \text{(b)} & \rho_i \Vdash \pi_i, & \text{(d)} & P'\ \mathcal{S}_{\mathsf{after}_\mu(\rho_i)}\ Q_i \end{array}$$

We write $P \simeq^{\mathsf{s}}_\vartheta Q$ to indicate that there exists a symbolic bisimulation $\mathcal{S}$ with $P\ \mathcal{S}_\vartheta\ Q$. □

Theorem 8. *$P \simeq_L Q$ iff $\exists\vartheta$: $P \simeq^{\mathsf{s}}_\vartheta Q$ and $L \vDash \vartheta$. In addition, $(\simeq) = (\simeq^{\mathsf{s}}_{\mathsf{tt}})$.* □

As a first attempt to define weak symbolic bisimulation, let us try simply replacing the strong transitions in Definition 7 with weak edges defined by conjoining formulae. For example, we would have $P \xRightarrow[\mathtt{tt}]{\varepsilon} P$ and $P \xRightarrow[\pi\wedge\rho]{a} P'$ if $P \xrightarrow[\pi]{\tau} \cdot \xrightarrow[\rho]{a} P'$. Unfortunately, this definition does not suffice. Consider the processes P_5 and Q_5, previously defined; these have the following symbolic transition graphs (where we have write $\xrightarrow[\pi]{\mu}$ as $\xrightarrow{\mu_\pi}$):

$$\begin{tikzcd}
 & & P_5 \arrow[dl, "b_\ell"'] \arrow[ddrr, "b_\ell"] & & & & & Q_5 \arrow[dd, "b_\ell"] & \\
 & \bullet \arrow[d, "\tau_{\ell\wedge k}"'] & & & & \approx & & & \\
 & P_1 \arrow[dl, "\tau_{\ell\wedge k}"'] \arrow[dr, "\tau_k"] & & & Q_1 \arrow[dl, "\tau_\ell"'] \arrow[dr, "\tau_{\ell\wedge k}"] & & & Q_1 \arrow[dl, "\tau_\ell"'] \arrow[dr, "\tau_{\ell\wedge k}"] & \\
\bullet & & \bullet \arrow[d, "a_k"] & \bullet & & \bullet \arrow[d, "a_k"] & \bullet & & \bullet \arrow[d, "a_k"] \\
 & & \bullet & & & \bullet & & & \bullet
\end{tikzcd}$$

As noted in Section 3, in order to prove these processes equivalent we must compare the processes P_1 and Q_1 under the assumption that ℓ is initially alive, but using our provisional definition we would end up comparing P_1 and Q_1 under the assumption $\mathtt{tt} = \mathrm{neg}(\ell \wedge k)$, which is not strong enough to prove that they are related.

As a second attempt, we might simply allow all positive information to carry over into the recursive formula ϑ_i, that is, change the last clause of Definition 7 to $\vartheta_i = \rho_i$. Whereas our first attempt produced an equivalence that was too strong, the revised definition is too weak. For example, the following processes would be identified even though they are not barbed equivalent.

$$\begin{tikzcd}
P_6 = \big[(\alpha.a)_\ell \,|\, (\overline{\alpha})_k\big]\backslash\alpha \arrow[d, "\tau_{\ell\wedge k}"] & \not\approx & Q_6 = \big[(\alpha)_\ell \,|\, (\overline{\alpha}.a)_k\big]\backslash\alpha \arrow[d, "\tau_{\ell\wedge k}"] \\
P_6' \arrow[d, "a_\ell"] & & Q_6' \arrow[d, "a_k"] \\
\bullet & & \bullet
\end{tikzcd}$$

Here P_6' and Q_6' would be compared under the formula $\ell \wedge k$. This formula, however, says something more than we would like, namely that ℓ and k remain alive until P_6' and Q_6' execute their first action. More complicated examples can be constructed to show that we must be able to express properties such as "ℓ and k must have been alive, then ℓ must have died, and after that k must have died."

Our solution is to define weak symbolic edges using a *past-time temporal logic* [17], interpreted over sequences of livesets. A *live sequence* $\mathcal{L}$ is a finite nonempty sequence of livesets $\langle L_1, \ldots, L_n\rangle$, such that for every i between 1 and $n-1$ there exists a location k such that $L_{i+1} = L_i \backslash \{k\}$. For example, $\langle \{\ell\}, \varnothing\rangle$ is a live sequence, but $\langle \{\ell\}, \{\ell\}\rangle$ and $\langle \{\ell, k\}, \varnothing\rangle$ are not. We write $\mathcal{L}_{(i)}$ for the i^{th} element of $\mathcal{L}$ and, where clear from context, use n to refer to the length of $\mathcal{L}$. Thus, for example, $\mathcal{L}$ models $\overline{\ell}$ if $\ell \notin \mathcal{L}_{(n)}$ and $\mathcal{L}$ models $\diamondminus\varphi$ if $\mathcal{L}$ or some prefix of $\mathcal{L}$ models φ. Because live sequences must be strictly decreasing, $\ell \wedge \diamondminus\overline{\ell}$ is unsatisfiable; however $\langle \{\ell\}, \varnothing\rangle \vDash \overline{\ell} \wedge \diamondminus\ell$. Weak symbolic transitions are defined as follows:

$$\begin{array}{ll}
P \xRightarrow[\mathtt{tt}]{\varepsilon} P & P \xRightarrow[\varphi\wedge\pi]{\alpha} P' \;\text{ if } P \xRightarrow[\varphi]{\varepsilon} \cdot \xrightarrow[\pi]{\alpha} P' \\
P \xRightarrow[\diamondminus(\varphi\wedge\pi)]{\varepsilon} P' \;\text{ if } P \xRightarrow[\varphi]{\varepsilon} \cdot \xrightarrow[\pi]{\tau} P' & P \xRightarrow[\overline{k}\wedge\ominus(\varphi\wedge\pi)]{\mathit{kill}\,k} P' \;\text{ if } P \xRightarrow[\varphi]{\varepsilon} \cdot \xrightarrow[\pi]{\mathit{kill}\,k} P' \\
P \xRightarrow[\varphi\wedge\pi]{\mu} P' \;\text{ if } P \xRightarrow[\varphi]{\mu} \cdot \xrightarrow[\pi]{\tau} P' &
\end{array}$$

Intuitively $P \overset{\mu}{\underset{\varphi}{\Longrightarrow}} P'$ means that P can perform the action μ to become P' in an environment where the change in live sets satisfies the formula φ. For example if $\varphi_1 = (\ell \wedge k) \mathbin{;} \ell$ and $\varphi_2 = (\ell \wedge k) \mathbin{;} k$ then P_6 has the symbolic transition $\overset{a}{\underset{\varphi_1}{\Longrightarrow}}$ but not $\overset{a}{\underset{\varphi_2}{\Longrightarrow}}$, whereas for Q_6 it is the opposite.

As parameters to the weak relation we simply take Boolean formulae, but now interpreted on the initial liveset of a live sequence. Rather than use two logics in the definition or introduce additional operators, we define the function "initially" which converts Boolean formulae into temporal formulae with this interpretation in mind. The transformation function for generating formulae after an action is performed, which we call "finally", must then transform temporal formula into Boolean ones. The definitions are as follows: (In the full paper, we show how to calculate these functions.)

$$\begin{array}{rll} \mathcal{L} \vDash \text{initially}(\pi) & \text{iff} & \mathcal{L}_{(1)} \vDash \pi \\ M \vDash \text{finally}(\varphi) & \text{iff} & \exists \mathcal{L}\colon\ \mathcal{L} \vDash \varphi \text{ and } M = \mathcal{L}_{(n)} \end{array}$$

Definition 9 (Weak symbolic bisimulation). Similar to Definition 7, except that when $P \ \mathcal{S}_\pi\ Q$ and $P \overset{\mu}{\underset{\varphi}{\Longrightarrow}} P'$ we require:

$$\begin{array}{llll} \text{(a)} & \text{initially}(\pi) \wedge \varphi \Vdash \bigvee_i \psi_i, & \text{(c)} & Q \overset{\widehat{\mu}}{\underset{\varphi_i}{\Longrightarrow}} Q_i, \text{ and} \\ \text{(b)} & \psi_i \Vdash \varphi_i, & \text{(d)} & P' \ \mathcal{S}_{\text{finally}(\psi_i)}\ Q_i \end{array} \qquad \square$$

Theorem 10. *$P \approxeq_L Q$ iff $\exists \pi\colon\ L \vDash \pi$ and $P \approxeq^{\mathsf{s}}_\pi Q$. In addition, $(\approxeq) = (\approxeq^{\mathsf{s}}_{\mathsf{tt}})$.* $\square$

5 Conclusions

In this paper we have proposed a new semantic theory for distributed systems which takes into account the possibility of failures at sites. This theory is an adaptation of standard bisimulation-based theories [18] using an operational semantics for *located processes*. The new semantic equivalences are justified in terms of *barbed bisimulations* [22]. We also give *symbolic* characterizations of the new equivalences, which means that they can be investigated using the symbolic methods of [15].

Site failure has also played a role in languages studied in [2, 4, 12]. In these papers abstract languages based on Facile [13] or the pi-calculus [19, 5] are studied. The original motivation for this paper was to provide an alternative characterization of barbed equivalence for languages such as these. Although we have not treated value passing or references, we postulate that our results can be extended in a straightforward way to value-passing languages which retain the assumption that all failures are independent, such as the languages in [2, 4]. More delicate is the extension to languages such as the distributed join-calculus [12] in which the independence assumption is dropped. In this case the logical language used for symbolic bisimulations must be extended to allow statements about the interdependence of locations; we leave this to future work.

A number of location-based equivalences already exist in the literature [8, 9, 20, 10]; however, none of these theories addresses the possible failure of sites. Their emphasis, rather, is to define a measure of the concurrency or distribution of a process: two processes are deemed equivalent only if, informally, they have the same degree of concurrency. In the full paper we give a series of counter-examples which show that $\approxeq$ is incomparable with all of the equivalences proposed in these papers; we also discuss variations on the language and model of failure.

References

[1] Samson Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–117. Addison-Wesley, 1990.

[2] Roberto Amadio and Sanjiva Prasad. Localities and failures. In *FST-TCS*, volume 880 of *LNCS*. Springer, 1994.

[3] Roberto Amadio. From a concurrent λ-calculus to the π-calculus. In *Foundations of Computation Theory*, volume 965 of *LNCS*. Springer, 1995.

[4] Roberto Amadio. An asynchronous model of locality, failure, and process mobility. Technical report, Laboratoire d'Informatique de Marseille, 1997.

[5] Roberto Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. In *CONCUR96*, volume 1119 of *LNCS*, pages 147–162. Springer, 1996.

[6] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[7] G. Boudol. A lambda calculus for (strict) parallel functions. *Information and Control*, 108:51–127, 1994.

[8] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. *Formal Aspects of Computing*, 6:165–200, 1994.

[9] I. Castellani. Observing distribution in processes: static and dynamic localities. *International Journal of Foundations of Computer Science*, 6(6):353–393, 1995.

[10] Flavio Corradini. *Space, Time and Nondeterminism in Process Algebras*. PhD thesis, Università Degli Studi di Roma "La Sapienza", 1996.

[11] C. Fournet and G. Gonthier. The refliexive CHAM and the join-calculus. In *POPL94*. ACM Press, 1994.

[12] C. Fournet, G. Gonthier, J.J. Levy, L. Marganget, and D. Remy. A calculus of mobile agents. In *CONCUR96*, volume 1119 of *LNCS*, pages 406–421. Springer, 1996.

[13] A. Giacalone, P. Mishra, and S. Prasad. A symmetric integration of concurrent and functional programming. *International Journal of Parallel Programming*, 18(2):121–160, 1989.

[14] Andrew D. Gordon. Bisimilarity as a theory of functional programming. In *MFPS*, volume 1 of *ENTCS*, `http://pigeon.elsevier.nl/mcs/tcs/pc/Menu.html`. Elsevier, 1995.

[15] M. C. B. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[16] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[17] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent System: Specification*. Springer, 1992.

[18] Robin Milner. *Communication and concurrency*. Prentice-Hall, 1989.

[19] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Information and Computation*, 100(1), September 1992.

[20] Ugo Montanari and Daniel Yankelovich. Partial order localities. In *ICALP92*, volume 623 of *LNCS*, pages 617–628. Springer, 1992.

[21] James Riely and Matthew Hennessy. Distributed processes and location failures. Technical Report 2/97, University of Sussex, Department of Computer Science, `http://www.cogs.susx.ac.uk`, 1997.

[22] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.