Distributed Processes and Location Failures

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 using existing symbolic techniques.

Keywords: distributed systems, process semantics, site failure, barbed congruence.

1 Introduction

Many semantic theories have been proposed for concurrent processes [25, 21, 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.

In [8, 27, 11] semantic theories of systems have been proposed which take this kind of property into account. 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*

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

stop model in which failures are independent of each other and the number of failures that can occur is unbounded. The assumption that sites may fail is clearly reasonable; indeed, much of the difficulty in designing distributed systems stems from requirements for fault-tolerance. Moreover this assumption changes considerably the standard semantic theories of processes, [25, 21]. Assuming that sites can fail, it is easy to see that Systems A and B, outlined above, are indeed different, in the sense that they can exhibit different behavior to an observer: 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 [3, 16, 4]. In these papers, distributed languages with location failures are defined and shown to be very expressive; 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.

The aim of this paper is to investigate the extent to which site failure can be incorporated into a reasonable semantic theory of systems. To focus as much as possible of the impact of site failure we consider a simple idealized language, based on CCS, in which some of the more interesting features of distributed systems, such as those expressible in FACILE [3], are absent. As we shall see site failure introduces serious complications into the semantic descriptions of even the simplest of processes. Therefore it makes much scientific sense to first try to understand its effect for a simplified language and then to investigate the extent to which this understanding can be applied and extended to more realistic languages. This paper is concerned only with the first topic, although the second is discussed briefly in the Conclusions.

In Section 2, we describe a simple language for *located processes* based on pure CCS [25], with which we assume familiarity. For example $\ell[\![a.p_1 + b.p_2]\!] \mid k[\![\overline{a}.q_1 + c.q_2]\!]$ is a system consisting of two processes, one located at ℓ and the other at k. As in CCS, communication is binary. In the example, the first process can perform the action a and the second can perform the complementary action $\overline{a}$; therefore, these processes can synchronize via the *silent*, or *internal* action τ (which allows no further synchronization) and evolve to $\ell[\![p_1]\!] \mid k[\![q_1]\!]$. In addition to the usual operators of CCS we have the following new operators:

- $\mathsf{move}(\ell, p)$, which spawns process p 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. If ℓ is already dead, this

construct may silently evolve to p (that is, it behaves like $\tau.p$).

- $\mathsf{if}\ \ell\ \mathsf{then}\ p\ \mathsf{else}\ q$, which silently evolves to either p or q, depending on whether ℓ is alive or dead when the test is performed.

We first 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 $L \triangleright P \overset{\alpha}{\longmapsto} L' \triangleright P'$, where P and P' are located processes and α is either a visible action, which permits synchronization, or the internal action τ. Each of the new operators in our language — *spawning* a process at some site, *killing* a site, or *querying* the status of a site — 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 indefinite amount of time. Note that the operational semantics does not record the location at which actions are performed.

Based on this labelled transition system, we wish to define an equivalence between process terms which is appropriate for the language. It is immediately apparent that standard equivalences, such as CCS bisimulation [25] are no longer appropriate. For example the terms $\ell[\![a]\!] \mid k[\![b]\!]$ and $k[\![a]\!] \mid \ell[\![b]\!]$ would not be differentiated by these equivalences although they can easily be distinguished by an observer that has the capability of killing ℓ or k. In short, the standard semantic equivalences are no longer preserved by all contexts in our language.

To decide on an appropriate equivalence we follow the basic approach advocated in [31] and also used for example in [1, 18, 5] to determine equivalences for a range of different languages. We define both strong and weak barbed equivalence between processes, $\dot{\sim}$ and $\dot{\approx}$. These definitions are defined in terms of the *reduction relation* $\overset{\tau}{\longmapsto}$ and a basic observation predicate. We then dictate that the required equivalence, which we refer to as *barbed bisimulation equivalence*, is defined (for example in the weak case) as:

$$P \approx Q \text{ if and only if for every suitable context } \mathbb{P}[\cdot],\ \mathbb{P}[P] \mathrel{\dot{\approx}} \mathbb{P}[Q]$$

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.

$$\begin{aligned} P_1 &= \big(\ell[\![\alpha]\!] \mid k[\![\overline{\alpha} + \tau.a]\!]\big) \backslash \alpha \\ Q_1 &= \big(\ell[\![\alpha + \tau]\!] \mid k[\![\overline{\alpha}.a]\!]\big) \backslash \alpha \end{aligned}$$

$$\begin{aligned} P_2 &= (\ell[\![\alpha]\!] \mid k[\![\overline{\alpha}.a]\!]) \backslash \alpha \\ Q_2 &= \ell[\![\mathsf{move}(k, a)]\!] \end{aligned}$$

The first main result of the paper is a characterization of these congruences using a variation of *bisimulation*. In Section 3 we define two bisimulation-based relations, strong and weak *Located-Failure equivalence* (*LF-equivalence*), $\simeq$ and $\approxeq$, and show that these coincide with the indirectly defined barbed congruences.

Since both strong and weak LF-equivalence are 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 [25, 10]. However, as we shall see, 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 [19] 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 $L \triangleright P \stackrel{\alpha}{\longmapsto} L' \triangleright P'$ 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* [19] requires only minor modification to capture $\simeq$ and $\approxeq$; hence the symbolic proof techniques and tools of [19] may be used to check the new semantic equivalences proposed in this paper.

Up to now the paper has concentrated on a semantic theory for *located processes*. In Section 5 we briefly show how the same framework can also be applied to *basic processes*; using a slight variation on LF-bisimulations we give a characterization of barbed congruence for *basic processes*. The Conclusion, in Section 6, summaries our results, discusses the extent to which they might be generalized to other more expressive languages and indicates some related work.

Notations used

A	§2.1	Process constants in *PConst*
k, ℓ, m	§2.1	Locations in *Loc*
K, L, M	§2.2	Livesets in $\{\star\} \subseteq L \subseteq Loc$
a, b, c	§2.1	Actions in *Act*
α	§2.1	Labels in $Act_\tau = Act \cup \{\tau\}$
μ	§3.1	Labels in $KAct_\tau = Act \cup \{\tau\} \cup \{kill\,\ell \mid \ell \in Loc\}$
δ	§3.1	Labels in $FAct_\tau = Act \cup \{\tau\} \cup \{kill\,\ell, fail\,\ell \mid \ell \in Loc\}$
p, q	§2.1	Basic processes in *BProc*
P, Q	§2.1	Located processes in *LProc*
C, D	§2.2	Configurations $L \triangleright P$ in *Config*
$\mathbb{P}[\cdot]$	§2.3	Located-process contexts
$\mathbb{P}[\cdot]$	§2.3	Basic-process contexts
$\mathbb{C}[\cdot]$	§3.3	Configuration contexts

$\overset{\alpha}{\longmapsto}$	§2.2	Transition relation (*Config* $\times$ *Config*)
$\xrightarrow{\delta}$	§3.1	Transition relation with explicit failures (*Config* $\times$ *Config*)
$\xrightarrow[L]{\delta}$	§3.1	Derived relation (*LProc* $\times$ *LProc*)
π, ρ	§4.1	Boolean formulae
$\xrightarrow[\pi]{\mu}$	§4.1	Symbolic transition relation (*LProc* $\times$ *LProc*)
ϑ	§4.2	Negative Boolean formulae
φ, ψ	§4.3	Temporal formulae
$\overset{\mu}{\underset{\varphi}{\Longrightarrow}}$	§4.3	Weak symbolic transition relation (*LProc* $\times$ *LProc*)
$\mathcal{K}, \mathcal{L}$	§4.3	Live sequence
$\lvert\mathcal{L}\rvert$	§4.3	Length of a live sequence
$\mathcal{L}_{(i)}$	§4.3	i^{th} element of a live sequence
$\mathcal{L}_{(\bullet)}$	§4.3	Last element of a live sequence
$\mathcal{L}_{\langle i,j\rangle}$	§4.3	Subsequence of a live sequence
$\overset{\mu}{\underset{\mathcal{L}}{\Longmapsto}}$	§4.3	Compound transition relation (*LProc* $\times$ *LProc*)
$\dot{\sim}$	§2.3	Barbed bisimilarity
$\sim$	§2.3	Barbed equivalence (congruence for *LProc*)
$\overset{c}{\sim}$	§5	Barbed congruence (congruence for *BProc*)
$\simeq$	§3	Located-Failures equivalence
$\simeq^{\mathsf{s}}$	§4	Symbolic equivalence

2 The Language

2.1 Syntax

The language we adopt is based on CCS, extended with constructs to *locate* and *spawn* processes, to *kill* locations, and to *query* the state of a location, that is, to test whether a location is dead or alive.

The syntax of processes is parameterized with respect to several syntactic sets. We assume a set *Loc* of *locations* k, ℓ, m and a set *PConst* of *process constants* A, used to define recursive processes. As in [3], we presume that the set of locations includes a distinguished element $\star \in Loc$, which represents an *unfailing* or immortal location; this location behaves differently from all others in that it cannot be killed. Of the results in the paper, only Theorem 5.1 depends on the use of $\star$; it also simplifies some examples. For simplicity we will usually consider *Loc* to be finite although none of the results of the paper relies on this assumption.

As usual for CCS, we also assume 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 of (strong) *actions* $Act_\tau = Act \cup \{\tau\}$ includes also the distinguished *silent action* τ. We use α to range over Act_τ. (In examples, we often use α and β for restricted communication actions, as in $P\backslash\alpha$.) The formal syntax is given in Table 1.

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. A basic process p is *located* at ℓ using the

Table 1 Syntax of basic and located processes

$$
\begin{array}{llllll}
p,q\ (\in BProc) & ::= & a.p & \mid\ \tau.p & \mid\ A & \mid\ \sum_{i\in I} p_i \\
 & & \mid\ \mathsf{move}(\ell,\, p) & \mid\ \mathsf{kill}\,\ell.p & \mid\ \mathsf{if}\ \ell\ \mathsf{then}\ p\ \mathsf{else}\ q & \\
 & & \mid\ p \mid q & \mid\ p\backslash a & \mid\ p\langle f\rangle & \\[1ex]
P,Q\ (\in LProc) & ::= & P \mid Q & \mid\ P\backslash a & \mid\ P\langle f\rangle & \mid\ \ell[\![p]\!]
\end{array}
$$

construct $\ell[\![p]\!]$. Located processes, then, may be combined using any of the *static operators* of CCS: parallel composition ($p \mid q$), action restriction ($p\backslash a$) and action renaming ($p\langle f\rangle$). 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. Note also that restriction and renaming operate only on actions, not locations. We make the usual assumptions about the renaming function f: $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$.

Basic processes may be combined using static or *dynamic* operators. The latter include all of the new constructs described in the introduction (spawn, kill and query) and the dynamic operators of CCS: action prefixing ($\alpha.p$), recursion via process constants (A) and CCS choice ($p+q$).

As usual, we write the inactive process ($\sum\varnothing$) as nil. In located process, we sometimes write $\ell[\![\mathsf{nil}]\!]$ as nil, dropping the location subscript; in basic processes, we almost always drop final nil term, writing "$a.\mathsf{nil}$" as "a". We also use the following abbreviations for query expressions:

$$
\begin{aligned}
\mathsf{if}\ \ell\ \mathsf{then}\ p &\stackrel{def}{=} \mathsf{if}\ \ell\ \mathsf{then}\ p\ \mathsf{else}\ \mathsf{nil} \\
\mathsf{if}\ \overline{\ell}\ \mathsf{then}\ p &\stackrel{def}{=} \mathsf{if}\ \ell\ \mathsf{then}\ \mathsf{nil}\ \mathsf{else}\ p
\end{aligned}
$$

The *location sort* of a process term reports the set of location names that occur in the term, regardless of behavior of the term when considered as a process. We define the function "locs" to map terms to their location sorts. For example $\mathrm{locs}(\mathsf{if}\ \overline{\ell}\ \mathsf{then}\ \mathsf{nil}) = \{\ell\}$ and $\mathrm{locs}(\ell[\![\mathsf{move}(k,\, \mathsf{nil})]\!]) = \{\ell, k\}$. If $\mathrm{locs}(P)$ is finite, we say that P is *location-finite*.

Table 2 (Part A) Transition system with configurations

$$(\mathsf{Act_c})\ \frac{\ell \in L}{L \triangleright \ell[\![a.p]\!] \stackrel{a}{\longmapsto} L \triangleright \ell[\![p]\!]} \qquad (\mathsf{Tau_c})\ \frac{\ell \in L}{L \triangleright \ell[\![\tau.p]\!] \stackrel{\tau}{\longmapsto} L \triangleright \ell[\![p]\!]}$$

$$(\mathsf{Kill_c})\ \frac{\ell \in L}{L \triangleright \ell[\![\mathsf{kill}\, m.p]\!] \stackrel{\tau}{\longmapsto} L' \triangleright \ell[\![p]\!]}\ L' = \begin{cases} L \backslash \{m\}, & \text{if } m \neq \star \\ L, & \text{otherwise} \end{cases}$$

$$(\mathsf{Live_c})\ \frac{\ell \in L \qquad m \in L}{L \triangleright \ell[\![\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \stackrel{\tau}{\longmapsto} L \triangleright \ell[\![p]\!]}$$

$$(\mathsf{Dead_c})\ \frac{\ell \in L \qquad m \notin L}{L \triangleright \ell[\![\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \stackrel{\tau}{\longmapsto} L \triangleright \ell[\![q]\!]}$$

$$(\mathsf{Spawn_c})\ \frac{\ell \in L}{L \triangleright \ell[\![\mathsf{move}(k,\, p)]\!] \stackrel{\tau}{\longmapsto} L \triangleright k[\![p]\!]}$$

$$(\mathsf{Sum_c})\ \frac{L \triangleright \ell[\![p_j]\!] \stackrel{\alpha}{\longmapsto} L' \triangleright k[\![p'_j]\!]}{L \triangleright \ell[\![\sum_{i \in I} p_i]\!] \stackrel{\alpha}{\longmapsto} L' \triangleright k[\![p'_j]\!]}\ j \in I$$

$$(\mathsf{Def_c})\ \frac{L \triangleright \ell[\![p]\!] \stackrel{\alpha}{\longmapsto} L' \triangleright k[\![p']\!]}{L \triangleright \ell[\![A]\!] \stackrel{\alpha}{\longmapsto} L' \triangleright k[\![p']\!]}\ A \stackrel{def}{=} p$$

2.2 Operational semantics

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 set of locations that includes $\star$. Intuitively, a liveset keeps track of the set of live locations. 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. When writing livesets we almost always omit explicit references to $\star$. Thus "$L = \{\ell\}$" should be read "$L = \{\ell, \star\}$" and "$L \subseteq Loc$" should be read "$\{\star\} \subseteq L \subseteq Loc$".

In Table 2 we define the transition relation $(\stackrel{\alpha}{\longmapsto}) \subseteq Config \times Config$ (the symmetric rule for parallel composition has been omitted). The definition uses the following simple structural equivalence on processes:

$$\ell[\![p \,|\, q]\!] \equiv \ell[\![p]\!] \,|\, \ell[\![q]\!] \qquad \ell[\![p \backslash a]\!] \equiv \ell[\![p]\!] \backslash a \qquad \ell[\![p\langle f \rangle]\!] \equiv \ell[\![p]\!]\langle f \rangle$$

Most of the rules in Table 2 are straightforward, being inherited directly from CCS, modulo the constraint that the process $\ell[\![p]\!]$ can only move if ℓ is alive. Note

Table 2 (Part B) Transition system with configurations (continued)

$$(\mathsf{Str_c})\ \frac{P \equiv P' \qquad L \triangleright P' \xmapsto{\alpha} L' \triangleright Q' \qquad Q' \equiv Q}{L \triangleright P \xmapsto{\alpha} L' \triangleright Q}$$

$$(\mathsf{Comm_c})\ \frac{L \triangleright P \xmapsto{a} L \triangleright P' \qquad L \triangleright Q \xmapsto{\overline{a}} L \triangleright Q'}{L \triangleright P \mid Q \xmapsto{\tau} L \triangleright P' \mid Q'}$$

$$(\mathsf{Par_c})\ \frac{L \triangleright P \xmapsto{\alpha} L' \triangleright P'}{L \triangleright P \mid Q \xmapsto{\alpha} L' \triangleright P' \mid Q}$$

$$(\mathsf{Restr_c})\ \frac{L \triangleright P \xmapsto{\alpha} L' \triangleright P'}{L \triangleright P\backslash a \xmapsto{\alpha} L' \triangleright P'\backslash a}\ \alpha \notin \{a, \overline{a}\}$$

$$(\mathsf{Ren_c})\ \frac{L \triangleright P \xmapsto{\alpha} L' \triangleright P'}{L \triangleright P\langle f \rangle \xmapsto{f(\alpha)} L' \triangleright P'\langle f \rangle}$$

that the three new operators — kill, spawn and query — are all deemed to take some computational effort and thus are modeled using $\xmapsto{\tau}$. For example, let $L = \{\ell, k\}$ and $P = \ell[\![\big(a.(\beta \mid \mathsf{move}(k, \overline{\beta}.b))\big)\backslash\beta]\!]$. Then P can engage in the following transitions:

$$\begin{aligned} L \triangleright P &\xmapsto{a} L \triangleright \ell[\![\big(\beta \mid \mathsf{move}(k, \overline{\beta}.b)\big)\backslash\beta]\!] \\ &\equiv\ L \triangleright \big(\ell[\![\beta]\!] \mid \ell[\![\mathsf{move}(k, \overline{\beta}.b)]\!]\big)\backslash\beta \\ &\xmapsto{\tau} L \triangleright \big(\ell[\![\beta]\!] \mid k[\![\overline{\beta}.b]\!]\big)\backslash\beta \end{aligned}$$

Note also that a process may move to a location which is dead, i.e.

$$L \triangleright \ell[\![\mathsf{move}(k, P)]\!] \xmapsto{\tau} L \triangleright k[\![P]\!]$$

even if k is not in L; intuitively a process can only be aware of the status of a location k by doing a specific query, using the construct $\mathsf{if}\ k\ \mathsf{then}\ \ldots\ \mathsf{else}\ \ldots$

Weak transitions are defined as usual:

$$\begin{aligned} &\stackrel{\varepsilon}{\Longmapsto} \stackrel{def}{=} (\xmapsto{\tau})^* \\ &\stackrel{\alpha}{\Longmapsto} \stackrel{def}{=} (\stackrel{\varepsilon}{\Longmapsto} \cdot \xmapsto{\alpha} \cdot \stackrel{\varepsilon}{\Longmapsto}) \end{aligned} \qquad\qquad \hat{\alpha} \stackrel{def}{=} \begin{cases} \varepsilon, & \text{if } \alpha = \tau \\ \alpha, & \text{otherwise} \end{cases}$$

The function $\hat{\cdot}$ relates the labels of strong transitions to those of the weak transitions. We also use standard abbreviations throughout the paper. For example, we write $C \xmapsto{\alpha}$ to indicate that for some C', $C \xmapsto{\alpha} C'$.

2.3 Barbed equivalence

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 [25]. (Strong) CCS *bisimulation* is the largest symmetric relation $\dot{\sim}^{\text{ccs}}$ on configurations such that whenever $C \dot{\sim}^{\text{ccs}} D$ and $C \overset{\alpha}{\longmapsto} C'$ there exists a D' such that $D \overset{\alpha}{\longmapsto} D'$ and $C' \dot{\sim}^{\text{ccs}} D'$. A weak version of this relation, $\dot{\approx}^{\text{ccs}}$, can be obtained by adapting this definition, in the usual way, to the weak transition relation.

To see that CCS bisimulation is not suitable for our language, for example is not a congruence, consider the "suicide process" $\ell[\![\mathsf{kill}\,\ell]\!]$; this is strong CCS bisimilar to $\ell[\![\tau]\!]$ in isolation, but not in a context that can perform an action at ℓ:

$$\{\ell\} \triangleright \ell[\![\mathsf{kill}\,\ell]\!] \;\dot{\sim}^{\text{ccs}}\; \{\ell\} \triangleright \ell[\![\tau]\!] \qquad \{\ell\} \triangleright \ell[\![a]\!] \,|\, \ell[\![\mathsf{kill}\,\ell]\!] \;\dot{\not\sim}^{\text{ccs}}\; \{\ell\} \triangleright \ell[\![a]\!] \,|\, \ell[\![\tau]\!]$$

A more interesting example is the following:

$$P_3 = \big(\ell[\![\alpha.a]\!] \,|\, k[\![\overline{\alpha}]\!]\big)\backslash\alpha \qquad\qquad Q_3 = \big(\ell[\![\alpha]\!] \,|\, k[\![\overline{\alpha}.a]\!]\big)\backslash\alpha$$

$\{\ell,k\} \triangleright P_3 \dot{\sim}^{\text{ccs}} \{\ell,k\} \triangleright Q_3$, but these processes can be distinguished by a context that kills location ℓ — so long as the kill action is performed after the initial communication on α.

The use of $\dot{\approx}^{\text{ccs}}$ for CCS has been justified in [31] 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, 18, 7], and the asynchronous pi-calculus [5], giving further evidence for the reasonableness of this approach. Roughly, two processes are barbed 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. The observable actions are the "barbs", for which we adopt the following standard notation:

$$C\!\downarrow_a \;\overset{def}{\Leftrightarrow}\; C \overset{a}{\longmapsto} \qquad\qquad C\!\Downarrow_a \;\overset{def}{\Leftrightarrow}\; C \overset{a}{\Longmapsto}$$

DEFINITION 2.1 (BARBED BISIMILARITY). Weak *barbed bisimilarity* ($\dot{\approx}$) is the largest symmetric relation over configurations such that whenever $C \dot{\approx} D$:

(a) $C \overset{\tau}{\longmapsto} C'$ implies $\exists D'\colon\; D \overset{\varepsilon}{\Longmapsto} D'$ and $C' \dot{\approx} D'$

(b) $\forall a\colon\; C\!\downarrow_a$ implies $D\!\Downarrow_a$

Strong barbed bisimilarity ($\dot{\sim}$) is obtained by replacing $\Longmapsto$ by $\longmapsto$ and $\Downarrow$ by $\downarrow$ everywhere in the definition. □

Barbed bisimilarity is a very weak relation; for example, it is not preserved by parallel composition. However, by closing over all contexts we can arrive at a reasonable semantic equivalence that by definition enjoys an important property, namely that it is a congruence. In our language we have two syntactic categories — basic and located processes — which induce different relations.

DEFINITION 2.2 (CONTEXTS, BARBED EQUIVALENCE AND CONGRUENCE). We say that $\mathbb{P}[\cdot]$ is a *located-process context* if for any located process P, $\mathbb{P}[P]$ is a located process. Similarly, $\mathbb{P}[\cdot]$ is a *basic-process context* if for any *basic* process p, $\mathbb{P}[p]$ is a located process.

Barbed equivalence ($\approx$) relates located processes, *barbed congruence* ($\overset{c}{\approx}$), instead, relates basic processes. They are defined as follows:

$$
\begin{aligned}
P \approx_L Q &\overset{def}{\Leftrightarrow} \text{ for every located-process context } \mathbb{P}[\cdot],\ L \triangleright \mathbb{P}[P] \mathrel{\dot{\approx}} L \triangleright \mathbb{P}[Q] \\
P \approx Q &\overset{def}{\Leftrightarrow} \text{ for every liveset } L,\ P \approx_L Q \\
p \overset{c}{\approx}_L q &\overset{def}{\Leftrightarrow} \text{ for every basic-process context } \mathbb{P}[\cdot],\ L \triangleright \mathbb{P}[p] \mathrel{\dot{\approx}} L \triangleright \mathbb{P}[q] \\
p \overset{c}{\approx} q &\overset{def}{\Leftrightarrow} \text{ for every liveset } L,\ p \overset{c}{\approx}_L q
\end{aligned}
$$

If $P \approx_L Q$ we say that P and Q are *barbed equivalent at* L, and similarly for the congruence.[1] Strong barbed equivalence ($\sim$) and congruence ($\overset{c}{\sim}$) are obtained in the same manner from $\dot{\sim}$. □

Remark 2.3. Our terminology is inspired by that of [31, 5], in which barbed *equivalence* is defined by closing over *static* contexts (that is, those contexts built up using only parallel composition, restriction and renaming) and barbed *congruence* is defined by closing over *all* contexts, including *dynamic* contexts such as $[\cdot]+a$.

As usual for bisimulation-based semantic theories, $\overset{c}{\approx}$ is strictly finer than $\approx$ which is strictly finer than $\dot{\approx}$. For most of the paper we concentrate on barbed *equivalence*, turning to the full congruence in Section 5. □

Remark 2.4. Note that to check $P \approx Q$, it is sufficient to check that $P \approx_L Q$ for every $L \subseteq \mathrm{locs}(P,Q)$, rather than for every $L \subseteq Loc$. Because our language has no facility for the creation of new locations, the liveset cannot increase as a configuration evolves; that is, if $L \triangleright P \overset{\alpha}{\longmapsto} L' \triangleright P'$, then $\mathrm{locs}(P') \subseteq \mathrm{locs}(P)$. In addition, extraneous locations do not affect behavior; that is, if $\ell \notin \mathrm{locs}(P)$ then:

$$L \triangleright P \overset{\alpha}{\longmapsto} L' \triangleright P' \ \text{ if and only if } \ L \backslash \{\ell\} \triangleright P \overset{\alpha}{\longmapsto} L' \backslash \{\ell\} \triangleright P'$$

Both of these properties are easily established by rule induction. Given these, it then follows immediately that:

$$
\begin{aligned}
Loc \triangleright P \mathrel{\dot{\approx}} Loc \triangleright Q \quad &\text{if and only if} \ \ \mathrm{locs}(P,Q) \triangleright P \mathrel{\dot{\approx}} \mathrm{locs}(P,Q) \triangleright Q \\
&\text{if and only if} \ \ \mathrm{locs}(P) \triangleright P \mathrel{\dot{\approx}} \mathrm{locs}(Q) \triangleright Q
\end{aligned}
$$

In particular for location-finite processes, barbed equivalence and congruence can be checked by considering only finite livesets. □

[1] We could also have defined $\approx$ and $\overset{c}{\approx}$ directly. For example $P \approx Q$ if for every context $\mathbb{C}[\cdot]$ such that $\mathbb{C}[P]$ and $\mathbb{C}[Q]$ are configurations, $\mathbb{C}[P] \mathrel{\dot{\approx}} \mathbb{C}[Q]$.

Although some results concerning translations between languages have been obtained using the definition of barbed equivalence directly [3, 16], the relation is obscure and difficult to use in practice because it requires quantification over all contexts. To show processes are distinguished it is necessary to find a live set and a context for which the resulting configurations are not barbed bisimilar. These can be found for the processes P_1 and Q_1, given in the introduction, and therefore they are distinguished by $\approx$. However P_2 and Q_2 are identified though it is far from obvious why. Even worse, processes P_6 and Q_6 (given on page 15) 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 examples. The processes $\ell[\![a]\!]$ and $k[\![a]\!]$ can be distinguished by a context that kills one of the two locations. The same context can be used to distinguish the basic processes $\mathsf{move}(\ell, a)$ and $\mathsf{move}(k, a)$, regardless of where they are located. These examples indicate that although the locations at which actions are performed are not directly observable, 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 $\ell[\![a]\!] \mid k[\![b]\!]$. On the other hand, only the first kill of a site is observable; thus, $\mathsf{kill}\,k.(\mathsf{kill}\,k + p)$ is indistinguishable from $\mathsf{kill}\,k.(\tau + p)$. The conditional exhibits a related property: $\ell[\![\mathsf{if}\ \ell\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \approx \ell[\![\tau.p]\!]$.

The spawn operator serves as a syntactic bridge between basic and located processes; thus, it is not surprising that top-level spawns can be eliminated:

$$\ell[\![\mathsf{move}(\ell, p)]\!] \approx \ell[\![\tau.p]\!]$$
$$\ell[\![\mathsf{move}(k, p)]\!] \approx \big(\ell[\![\overline{\alpha}]\!] \mid k[\![\alpha.p]\!]\big)\backslash\alpha, \text{ if } \alpha \text{ does not occur in } p$$

However, the interaction between spawn and parallel composition is quite subtle. Consider the basic processes:

$$p_4 = \mathsf{move}(k, r) \mid \mathsf{move}(k, s) \qquad\qquad q_4 = \mathsf{move}(k, r \mid s)$$

If $k[\![r]\!] \not\approx k[\![s]\!]$ and $\ell \neq \star$ then $\ell[\![p_4]\!] \not\approx \ell[\![q_4]\!]$; these processes can be distinguished by killing ℓ after p_4 has spawned one subprocess but not the other. Immortal locations are peculiar in this respect: $\star[\![p_4]\!] \approx \star[\![q_4]\!]$.

Within a site, parallelism can be reduced to nondeterminism; for example, $\ell[\![a]\!] \mid \ell[\![b]\!] \approx \ell[\![a.b + b.a]\!]$. However, this is not true across sites. Given the examples thus far, we would expect to have $\ell[\![a]\!] \mid k[\![b]\!] \approx a_\ell.b_k + b_k.a_\ell$, but the latter is not a term in our language; indeed a language with such terms would be difficult to understand computationally.

Table 3 Transition system with explicit kills

All rules but $\mathsf{Kill_c}$ from Table 2, with α replaced by μ and $\longmapsto$ replaced by $\longrightarrow$.

$$(\mathsf{Kill1_k})\ \frac{\ell \in L \qquad \neg\text{fallible}_L(m)}{L \triangleright \ell[\![\mathsf{kill}\, m.p]\!] \xrightarrow{\tau} L \triangleright \ell[\![p]\!]}$$

$$(\mathsf{Kill2_k})\ \frac{\ell \in L \qquad \text{fallible}_L(m)}{L \triangleright \ell[\![\mathsf{kill}\, m.p]\!] \xrightarrow{\mathit{kill}\, m} L \setminus \{m\} \triangleright \ell[\![p]\!]}$$

$$(\mathsf{Fail_k})\ \frac{\text{fallible}_L(m)}{L \triangleright P \xrightarrow{\mathit{fail}\, m} L \setminus \{m\} \triangleright P}$$

3 Located-Failures equivalence

In this section and the next we provide alternate characterizations of barbed equivalence for located processes. We start by giving an enriched configuration semantics which, while not strictly necessary, greatly simplifies the notation and sharpens many of the definitions. We then define both strong and weak LF-equivalence. The main technical result of this section is that LF-equivalence and barbed equivalence coincide. Through examples, we show that weak LF-equivalence is somewhat weaker than one might expect — with some surprising results.

3.1 Enriched configuration semantics

The examples at the end of Section 2.3 show that actions performed by the kill operator are sometimes observable, albeit indirectly. For example, $\mathsf{kill}\,\ell$ is different from τ if ℓ is alive, but it is the same otherwise. In Table 3 we introduce a transition relation ($\longrightarrow$) in which this distinction is manifest. The definition uses the predicate "fallible$_L$", defined as follows:

$$\text{fallible}_L(m) \overset{\mathit{def}}{\Leftrightarrow} m \neq \star \text{ and } m \in L$$

This enriched relation uses explicit *kill actions* ($\mathit{kill}\,\ell$) and *fail actions* ($\mathit{fail}\,\ell$).[2] Unless otherwise noted, we observe the following discipline when referring to labels in transition graphs:

$$\begin{aligned} \alpha &::= \tau \mid a \\ \mu &::= \alpha \mid \mathit{kill}\,\ell \\ \delta &::= \mu \mid \mathit{fail}\,\ell \end{aligned}$$

[2]Whereas kill actions are essential in the definition of symbolic LF-equivalence that we present in Section 4, fail actions are introduced purely for notational convenience.

Also let *KAct* be the set $Act \cup \{kill\,\ell \mid \ell \in Loc\}$; thus μ normally ranges over $KAct_\tau$. For a summary of the notation used in the paper, see Section 1.

Intuitively, a kill action marks an *effective* execution of the kill operator; *ineffective* executions of the kill operator are modeled by τ-transitions. A fail action marks the execution of an effective kill by the surrounding context. The rule Fail says, in effect, that any location may fail at any moment. Note that the rule does not depend on the process term, but only on the liveset; thus it is sufficient to use μ, rather than δ, in the inductive rules such as Par.

The relationship between the two transition systems is given by the following Lemma.

LEMMA 3.1. $C \overset{a}{\longmapsto} C'$ *if and only if* $C \xrightarrow{a} C'$
$C \overset{\tau}{\longmapsto} C'$ *if and only if* $C \xrightarrow{\tau} C'$ *or* $\exists k\colon\ C \xrightarrow{kill\,k} C'$

Proof. Straightforward rule induction. □

Note that if $L \triangleright P \xrightarrow{\delta} L' \triangleright P'$, then L' is determined by L and δ. To emphasize this, we adopt the following notation. For each action δ, the function $\mathrm{iafter}_\delta(L)$ (immediately after δ) reflects the effect of action δ on L; for example, $\mathrm{iafter}_a(L) = L$ and $\mathrm{iafter}_{kill\,\ell}(\{\ell,k\}) = \{k\}$. The relations $\xrightarrow[L]{\delta}$ and $\overset{\delta}{\underset{L}{\Longrightarrow}}$ describe the δ-transitions of a process under liveset L. Thus if $P = \ell[\![\alpha.a]\!] \mid k[\![\overline{\alpha}]\!]$, then $P \overset{a}{\underset{\{\ell,k\}}{\Longrightarrow}} \mathsf{nil}$, but P has no a-transition under the liveset $\{k\}$. The formal definitions are as follows:

$$\mathrm{iafter}_\delta(L) \overset{def}{=} \begin{cases} L\backslash\{k\}, & \text{if } \delta = kill\,k \text{ or } \delta = fail\,k \\ L, & \text{if } \delta \in Act \cup \{\tau,\varepsilon\} \end{cases}$$

$$P \xrightarrow[L]{\delta} P' \overset{def}{\Leftrightarrow} L \triangleright P \xrightarrow{\delta} \mathrm{iafter}_\delta(L) \triangleright P'$$
$$P \overset{\delta}{\underset{L}{\Longrightarrow}} P' \overset{def}{\Leftrightarrow} L \triangleright P \overset{\delta}{\Longrightarrow} \mathrm{iafter}_\delta(L) \triangleright P'$$

3.2 Strong LF-equivalence

We would like to define LF-equivalence directly on process terms, rather than configurations. It is not difficult to see that to do so, we will first have to define an equivalence that is parameterized by the set locations that are dead (or conversely *alive*) at the time that the processes are reached. For example consider the following processes:

$$P_5 = k[\![\mathsf{if}\ \overline{\ell}\ \mathsf{then}\ \mathsf{if}\ \ell\ \mathsf{then}\ a]\!] \qquad Q_5 = k[\![\mathsf{if}\ \overline{\ell}\ \mathsf{then}\ \mathsf{if}\ \ell\ \mathsf{then}\ b]\!]$$

These processes are barbed equivalent, but establishing this fact relies on comparing the processes "if ℓ then a" and "if ℓ then b" under the assumption that ℓ is already dead. We are thus lead to a definition in two steps. First we define a parameterized equivalence ($\simeq_L$) which compares located processes under the assumption that the set of locations L are alive. In order to take into account all possible initial contexts, we then quantify over all such livesets to define the equivalence $\simeq$.

DEFINITION 3.2 (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 $P\ \mathcal{S}_L\ Q$:

$$P \xrightarrow[L]{\delta} P' \text{ implies } \exists Q' :\ Q \xrightarrow[L]{\delta} Q' \text{ and } P'\ \mathcal{S}_{\mathrm{iafter}_\delta(L)}\ Q'$$

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 $L \subseteq Loc$. □

The definition of LF-bisimulation is similar to the definition of CCS bisimulation. Here, however, a kill action by P must be matched by exactly the same kill action by Q; in CCS bisimulation kill actions could be matched by any silent action. However, it is the use of fail actions that is more important; because of fail actions, P and Q have the same behavior in the face of any kill actions that the surrounding context might perform. The following Lemma shows how LF-bisimularity may be defined without the explicit use of fail actions.

LEMMA 3.3. *$\mathcal{S}$ is a strong LF-bisimulation if and only if for every L, $\mathcal{S}_L$ is symmetric and whenever $P\ \mathcal{S}_L\ Q$:*

(a) $P \xrightarrow[L]{\mu} P'$ *implies* $\exists Q' :\ Q \xrightarrow[L]{\mu} Q'$ *and* $P'\ \mathcal{S}_{\mathrm{iafter}_\mu(L)}\ Q'$

(b) *for every* $k \in L \quad P\ \mathcal{S}_{L\setminus\{k\}}\ Q$

Proof. Immediate from the definitions. □

THEOREM 3.4. *For all located processes, $P \simeq_L Q$ if and only if $P \sim_L Q$.*

Proof. Similar to that of Theorem 3.11, which is more difficult. □

The following Lemma demonstrates that the strong behavior of located processes depends only on the set of locations that are known to be *dead*, and therefore for location-finite processes $\simeq$ (which quantifies over all initial livesets) coincides with $\simeq_{Loc}$ (see Remark 2.4). Surprisingly, this property does not extend to the weak case.

LEMMA 3.5. (a) *Let P and Q be location-finite. Then $P \simeq_L Q$ and $M \subseteq L$ imply $P \simeq_M Q$.* (b) *If Loc is finite, then $\mathcal{S}$ is a strong LF-bisimulation if and only if for every L, $\mathcal{S}_L$ is symmetric and whenever $P\ \mathcal{S}_L\ Q$:*

$$M \subseteq L \text{ and } P \xrightarrow[M]{\mu} P' \text{ implies } \exists Q' :\ Q \xrightarrow[M]{\mu} Q' \text{ and } P'\ \mathcal{S}_{\mathrm{iafter}_\mu(M)}\ Q'$$

Proof. (a) follows from Lemma 3.3. (b) is immediate from the definition of strong LF-bisimulation. □

3.3 Weak LF-equivalence

We start with an example. Consider the following processes under weak barbed equivalence:

$$\begin{array}{rl} P_6 = & \Big(\ell[\![b.\beta.\alpha + b.(\alpha+\tau)]\!] \mid k[\![\overline{\beta}.(\overline{\alpha}+\tau.a) + \overline{\alpha}.a]\!]\Big)\backslash\alpha\backslash\beta \\ Q_6 = & \Big(\ell[\![b.(\alpha+\tau)]\!] \mid k[\![\overline{\alpha}.a]\!]\Big)\backslash\alpha \end{array}$$

If ℓ is initially dead, P_6 and Q_6 are clearly equivalent: both are strong equivalent to nil; if only k is initially dead, they are weak equivalent to b.nil. If ℓ and k are both initially alive, however, the situation is not so clear. The questionable move is P_6's b-transition to:

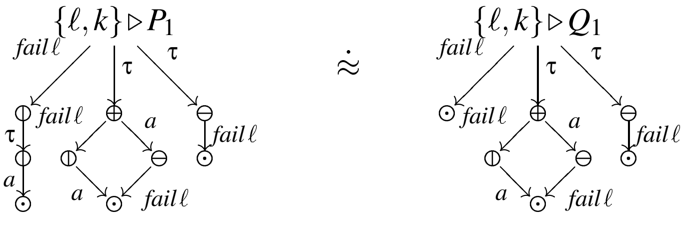
$$P_1 \simeq \big(\ell[\![\alpha]\!] \mid k[\![\overline{\alpha}+\tau.a]\!]\big)\backslash\alpha$$

To match this move Q_6 must perform a weak b-transition to:

$$Q_1 \simeq \big(\ell[\![\alpha+\tau]\!] \mid k[\![\overline{\alpha}.a]\!]\big)\backslash\alpha$$

But P_1 and Q_1 are not barbed equivalent: if ℓ is dead, then P_1 is capable of an a-transition that Q_1 cannot match. This would lead one to believe that P_6 and Q_6 are *not* barbed equivalent; however, they are.

Intuitively this is true because when P_6 reaches P_1, ℓ must be alive — if ℓ had been dead, the b-transition to P_1 would have been impossible. 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 P_1 and Q_1 only by killing ℓ, but it cannot control internal activity on the part of Q_1 before ℓ is dead. Killing k doesn't help to distinguish the two processes. The relevant sections of the transition systems are shown below. To improve readability, we have not shown the transitions labelled *fail* k; in addition, we have marked the states with different symbols, each symbol indicating a set of bisimilar states.



DEFINITION 3.6 (WEAK LF-EQUIVALENCE). $\mathcal{S}$ is a *weak LF-bisimulation* if for every L, $\mathcal{S}_L$ is symmetric and whenever $P\ \mathcal{S}_L\ Q$:

$$P \xrightarrow[L]{\delta} P' \text{ implies } \exists Q':\ Q \overset{\hat{\delta}}{\underset{L}{\Longrightarrow}} Q' \text{ and } P'\ \mathcal{S}_{\mathrm{iafter}_\delta(L)}\ Q'$$

We write $\approx$ for *weak LF-equivalence*, and $\approx_L$ for *weak LF-equivalence at L.* □

We can also give the definition without fail actions, making the cases explicit:

LEMMA 3.7. *$\mathcal{S}$ is a weak LF-bisimulation if and only if for every L, $\mathcal{S}_L$ is symmetric and whenever $P\ \mathcal{S}_L\ Q$:*

$$\text{(a)}\quad P \xrightarrow[L]{\mu} P' \text{ implies } \exists Q' :\ Q \overset{\hat{\mu}}{\underset{L}{\Longrightarrow}} Q' \text{ and } P'\ \mathcal{S}_{\mathrm{iafter}_\mu(L)}\ Q'$$

$$\text{(b)}\quad \text{for every } k \in L \quad \exists Q' :\ Q \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{k\}}{\Longrightarrow}} Q' \text{ and } P\ \mathcal{S}_{L\setminus\{k\}}\ Q'$$

Proof. Immediate from the definitions. □

The last clause of Lemma 3.7 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). Unrolling the recursive definition, if $k,m \in L$ then there must exist Q_1' and Q_2' equivalent to P under $L\setminus\{k,m\}$ such that

$$Q \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{k\}}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{k,m\}}{\Longrightarrow}} Q_1' \quad \text{and} \quad Q \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{m\}}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{k,m\}}{\Longrightarrow}} Q_2'$$

and likewise for any subset of L.

It may also be surprising that in Lemma 3.7 we do not need to allow for the possibility that a τ-transition is matched by a kill-transition. This fact is explained by the following Lemma.

LEMMA 3.8. $P \overset{kill\,k}{\underset{L}{\Longrightarrow}} P'$ *implies* $P \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{k\}}{\Longrightarrow}} P'$

Proof. Immediate from the operational semantics. □

Remark 3.9. Let $\approxeq_L'$ be the relation obtained by substituting $P \overset{\hat{\delta}}{\underset{L}{\Longrightarrow}} P'$ for $P \xrightarrow[L]{\delta} P'$ in Definition 3.6. As usual with weak bisimulation relations, $\approxeq_L' = \approxeq_L$. □

We now show that $\approxeq$ is a congruence on located processes (in other words, that $\approxeq$ is substitutive in all static contexts) — LF-equivalence is also a congruence for most operators on basic processes, as we will discuss in Section 5. We then present the main result of this section: that barbed equivalence and LF-equivalence coincide.

THEOREM 3.10. *Each relation $\approxeq_L$, and therefore $\approxeq$, is a congruence for located processes. That is, if $P \approxeq_L Q$ and $\mathbb{P}[\cdot]$ is a context such that $\mathbb{P}[P]$ and $\mathbb{P}[Q]$ are located processes then $\mathbb{P}[P] \approxeq_L \mathbb{P}[Q]$.*

Proof. By induction on the structure of contexts. Note that we must only consider the operators for parallel composition, renaming and restriction. In all three cases the argument is standard; for example, in the case of parallel composition we define a relation $\mathcal{S}_L = \{\langle P \,|\, R,\ Q \,|\, R\rangle \mid P \approxeq_L Q\}$ and show that $\mathcal{S}$ is a LF-bisimulation. □

THEOREM 3.11. *For each L, $P \approxeq_L Q$ if and only if $P \approx_L Q$.*

The remainder of this section is devoted to the proof of this theorem, an obvious corollary of which is that $\approxeq = \approx$.

One direction of Theorem 3.11 ($\approxeq_L \subseteq \approx_L$) follows immediately from the fact that each $\approxeq_L$ is a congruence. In the other direction, we must show that $P \approx_L Q$ implies $P \approxeq_L Q$ for each L. This involves constructing a collection of contexts $\mathbb{C}_L^{i,j}$ — mapping located processes to configurations — such that the relation

$$\mathcal{S}_L \stackrel{def}{=} \{ \langle P, Q \rangle \mid \exists i, j:\ \mathbb{C}_L^{i,j}[P] \dot{\approx} \mathbb{C}_L^{i,j}[Q] \}$$

is a LF-bisimulation.

ASSUMPTION 3.12. To simplify the exposition, we will assume that *Loc* is finite. The Theorem also holds if *Loc* is infinite, as we explain in Remark 3.14. □

The contexts are based on those of Sangiorgi [31]. We assume that the set of action names is partitioned into sets Act_1 and Act_2 with all actions that appear in process terms coming from Act_1. For each $a \in Act_1$ we assume that there is a corresponding action in Act_2 that is different from all other actions in Act; let a' denote this action. We also assume that Act_2 contains some other actions — c, c', $\{d_i, d_i' \mid 0 \leq i\}$ and $\{live_\ell \mid \ell \in Loc\}$ — and that all these actions are unique. Given these assumptions, the required contexts are as follows. (We drop parameters from $\mathbb{C}_L^{i,j}$ whenever they are unimportant or clear from context.)

$$\begin{aligned}
\mathsf{LSensor} &\stackrel{def}{=} \textstyle\prod_{\ell \in Loc} \ell[\![live_\ell]\!] \\
\mathsf{LKiller} &\stackrel{def}{=} (\textstyle\sum_{\ell \in Loc} \mathsf{kill}\, \ell.\overline{c}'.\mathsf{LKiller}) + \tau.d_0' \\
\mathsf{LCount}_i &\stackrel{def}{=} d_i' + c'.\mathsf{LCount}_{i+1}, \quad i \geq 2
\end{aligned}$$

$$\begin{aligned}
\mathsf{ASensor} &\stackrel{def}{=} \big(\textstyle\sum_{a \in Act_1} \overline{a}.\overline{c}.\overline{c}.\,(\tau.a' + \tau.\mathsf{ASensor})\big) + \tau.d_0 + \tau.d_1 \\
\mathsf{ACount}_j &\stackrel{def}{=} d_j + c.\mathsf{ACount}_{j+1}, \quad j \geq 2
\end{aligned}$$

$$\mathbb{C}_L^{i,j} \stackrel{def}{=} L \triangleright [\cdot] \mid \mathsf{LSensor} \mid \star[\![\mathsf{LKiller} \mid \mathsf{LCount}_i \mid \mathsf{ASensor} \mid \mathsf{ACount}_j]\!]$$

The contexts are designed so that if $\mathbb{C}[P]$ is barbed bisimilar to $\mathbb{C}[Q]$ and $\mathbb{C}[P] \stackrel{\varepsilon}{\Longmapsto} \mathbb{C}'[P']$ then there must be a Q' such that $\mathbb{C}[Q] \stackrel{\varepsilon}{\Longmapsto} \mathbb{C}'[Q']$ and $\mathbb{C}'[P'] \dot{\approx} \mathbb{C}'[Q']$. Significantly, the context $\mathbb{C}'[\cdot]$ must be the same for both processes; that is, the structure of the context must be preserved through matching moves. This is achieved by making "observable" — via barbs — any change in the state of the context processes.

The processes ACount and $\mathsf{ASensor}$ are taken directly from [31, Theorem 3.3.2]. In $\mathbb{C}[P]$, $\mathsf{ASensor}$ identifies the communication actions performed by P, whereas ACount controls the *number* of these actions that P can perform. We refer the reader to [31] for more details on the use of these contexts. To these we add three new processes, LKiller, LSensor and LCount. LSensor senses kill actions performed by P (in addition to those performed by the context), in much the same way as ASensor senses actions of P. LKiller mimics the manner in which the context kills locations and LCount disciplines its use of LKiller, preventing the context from engaging in infinite internal activity.[3]

The resulting contexts are strong enough to recover LF-equivalence from barbed bisimilarity — specifically, they are strong enough to show that $\mathcal{S}$ is a weak LF-bisimulation up to $\equiv$. To prove this we first need the following Lemma:

LEMMA 3.13. *If $\mathbb{C}[P] \stackrel{\varepsilon}{\Longmapsto} C$, $\mathbb{C}[Q] \stackrel{\varepsilon}{\Longmapsto} D$ and $C \dot{\approx} D$, then C and D must have exactly the same livesets.*

Proof. Follows from the fact that LSensor offers a distinct communication for each live location and is incapable of communication or silent transitions, within the context $\mathbb{C}[\cdot]$. $\square$

We now complete the proof of Theorem 3.11.

Proof of Theorem 3.11. We show that $\mathcal{S}$ is an LF-bisimulation up to $\equiv$. We use the characterization of LF-bisimulation given in Lemma 3.7. Suppose that $P\ \mathcal{S}_L\ Q$ and $P \xrightarrow[L]{\alpha} P'$. We examine the three clauses of this Lemma in turn.

To satisfy the first clause, we must show that Q can perform the same action, evolving to a matching state Q'. The proof can be copied directly from Sangiorgi, using Lemma 3.13 to establish the only additional requirement: that the livesets are unchanged during transitions of $\mathbb{C}[P]$ and $\mathbb{C}[Q]$.

On the other hand, suppose that $P \xrightarrow[L]{kill\,k} P'$ and therefore $\mathbb{C}^{i,j}_{L}[P] \stackrel{\tau}{\longmapsto} \mathbb{C}^{i,j}_{L\backslash\{k\}}[P'] \stackrel{def}{=} C'$. This must be matched by a move $\mathbb{C}^{i,j}_{L}[Q] \stackrel{\varepsilon}{\Longmapsto} D'$ such that $C' \dot{\approx} D'$. We show that D' must be of the form $\mathbb{C}^{i,j}_{L\backslash\{k\}}[Q']$, up to $\equiv$. From Lemma 3.13 it is easy to see that the liveset in D' must be $L\backslash\{k\}$. To see that the rest of the context must be unchanged, note that C' can silently move to a state in which the only d-actions possible are d'_0, d'_i, d_0, and d_j. D' can only match this state if $D' = \mathbb{C}^{i,j}_{L\backslash\{k\}}[Q']$ for some Q'. Therefore, although k has died, the context could not have killed it (since the LKiller has not moved). Thus Q must have killed k, and we have that $Q \xrightarrow[L]{kill\,k} \cdot \xRightarrow[L\backslash\{k\}]{\varepsilon} Q'$ and $P'\ \mathcal{S}_{L\backslash\{k\}}\ Q'$, as required.

We now turn to the final requirement of Lemma 3.7. We must show that $Q \xRightarrow[L]{\varepsilon} \cdot \xRightarrow[L\backslash\{k\}]{\varepsilon} Q'$ for some Q' such that $P'\ \mathcal{S}_{L\backslash\{k\}}\ Q'$.

We know that $\mathbb{C}^{i,j}_{L}[P] \stackrel{\tau}{\longmapsto} \cdot \stackrel{\tau}{\longmapsto} \mathbb{C}^{i+1,j}_{L\backslash\{k\}}[P] = C'$, by the context killing the location k. Therefore it must be that $\mathbb{C}^{i,j}_{L}[Q]$ is able to silently reach a configuration

[3]There is a risk of such activity because once a site is dead any further attempts to kill it are treated as τ-actions. LCount makes every move of LKiller visible, thus preventing it from internal moves that would otherwise go unnoticed. We use separate counters for LKiller and ASensor so that communication-transitions of P cannot be matched by kill-transitions.

D' that is barbed bisimilar to C'. Reasoning as before, we can show that D' must structurally equivalent to $\mathbb{C}^{i+1,j}_{L\backslash\{k\}}[Q']$ for some Q'. The only question is: who killed k? Q or the context? Consulting Lemma 3.8, however, the question proves to be irrelevant. If $Q \xRightarrow[L]{\mathsf{kill}\,k} Q'$, then there exists a Q'' such that:

$$\mathbb{C}^{i,j}_{L}[Q] \overset{\varepsilon}{\Longmapsto} \mathbb{C}^{i,j}_{L}[Q''] \overset{\varepsilon}{\Longmapsto} \mathbb{C}^{i+1,j}_{L\backslash\{k\}}[Q''] \overset{\varepsilon}{\Longmapsto} \mathbb{C}^{i+1,j}_{L\backslash\{k\}}[Q']$$

Thus we have as required that $Q \xRightarrow[L]{\varepsilon} \cdot \xRightarrow[L\backslash\{k\}]{\varepsilon} Q'$ and $P' \; \mathcal{S}_{L\backslash\{k\}} \; Q'$. □

Remark 3.14. To simplify the exposition we have assumed that *Loc* is finite; however, the proof is only slightly more complicated if *Loc* is infinite. In this case, we must change the contexts so that they do not include an infinite number of processes. The culprit is LSensor which can be changed as follows:

$$\mathsf{LSensor}' \stackrel{def}{=} (\textstyle\sum_{\ell\in Loc} \mathsf{if}\ \ell\ \mathsf{then}\ live_{\ell}) + d_1'$$

The sole purpose of LSensor is to guarantee Lemma 3.13. Using LSensor$'$, the proof of Lemma 3.13 is only slightly more complicated. The summand d_1' is necessary to keep LSensor from moving in the case that all locations are dead. Note that we could achieve the same result by sensing *dead* rather than live locations. □

Remark 3.15. If we restrict the language, disallowing terms of the form $\mathsf{kill}\,k.p$ in which $p \neq \mathsf{nil}$, the results do not change. To accommodate this language with “asynchronous kills,” LKiller must be changed as follows:

$$\mathsf{LKiller}' \stackrel{def}{=} \textstyle\sum_{\ell\in Loc} \overline{c}'.(\mathsf{kill}\,\ell \mid \mathsf{LKiller})$$

Using this definition, the result follows by extending the structural equivalence with the following absorption law: $\mathsf{nil} \mid P \equiv P$. (This is necessary so that the residual of the term $\mathsf{kill}\,\ell.\mathsf{nil}$ in the context can be ignored.) The proof also makes use of Lemma 3.8. □

4 Symbolic characterizations

While LF-equivalence provides a great deal of insight into the meaning of barbed congruence in distributed process description languages such as ours, it may be unwieldy to use in practice. For a start it is based on an operational semantics which uses configurations rather than processes. Moreover this operational semantics needs to take into consideration not only all the kill actions which the processes can perform but also the possible kills which can be carried out by the environment. As a result the labelled transition systems associated with even the simplest processes are very complex.

In this section, we define a *symbolic* transition system directly on located process terms, then give 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. By adapting the algorithms in [19], one could derive an alternative method for automatically checking LF-equivalence on finite state processes. But the symbolic characterizations are not only useful for automated proof; they also greatly simplify reasoning by hand; symbolic graphs are typically an order of magnitude smaller than their concrete counterparts.

We begin by giving the symbolic operational semantics.

ASSUMPTION 4.1. Throughout this section we will assume that *Loc* is finite. The results can be generalized in to location-finite processes, but the notation required is tedious. □

4.1 Symbolic semantics

The symbolic transition relation makes use of Boolean formulae π, ρ, in which location names serve as the literals.

$$\pi, \rho \ ::= \ \mathsf{tt} \mid \ell \mid \overline{\ell} \mid \textstyle\bigvee_{i \in I} \pi_i \mid \pi \wedge \rho$$

Whereas *literals* are drawn from *Loc*, *atoms* are drawn from $Loc \cup \{\overline{\ell} \mid \ell \in Loc\}$. *Positive atoms* occur in *Loc* whereas *negative atoms* occur in $\{\overline{\ell} \mid \ell \in Loc\}$. We say that π is a *positive formula* if it contains no instance of a negative atom; *negative formulae* are defined similarly. If ℓ appears as an atom in π, we say that ℓ *appears positively* in ρ. If $\overline{\ell}$ appears in π the ℓ *appears positively* in ρ. Thus tt is both a positive and a negative formula.

Remark 4.2. We do not require full negation, although including it would not pose any difficulties; we write $\overline{\ell}$ for the negation of ℓ. On the other hand, we *do* allow infinitary disjunction; were we to restrict our attention to image-finite processes, finitary disjunction would be sufficient. □

Intuitively, a formula indicates a set of constraints on the status of locations (dead or alive) at the time that the transition is enabled. For example, if $P \xrightarrow[\ell \wedge \overline{m}]{\mu} P'$ then P is capable of making an μ-transition to P' if location ℓ is alive and m is dead; that is, $P \xrightarrow[L]{\mu} P'$ if $\ell \in L$ and $m \notin L$. The semantics of the logic is given with respect to live sets:

$L \vDash \mathsf{tt}$	always	$L \vDash \pi \wedge \rho$	if $L \vDash \pi$ and $L \vDash \rho$
$L \vDash \ell$	if $\ell \in L$	$L \vDash \bigvee_i \pi_i$	if $\exists j\colon\ L \vDash \pi_j$
$L \vDash \overline{\ell}$	if $\ell \notin L$		

In Table 4 we define the transition relation $(\xrightarrow[\pi]{\mu}) \subseteq LProc \times LProc$ (the symmetric rules for parallel composition have been omitted). The relationship between the two transition systems is summarized in the following Lemma. We defer examples to Section 4.3.

Table 4 Symbolic transition system

$$(\mathsf{Act_s})\ \frac{}{\ell[\![a.p]\!] \xrightarrow[\ell]{a} \ell[\![p]\!]} \qquad (\mathsf{Tau_s})\ \frac{}{\ell[\![\tau.p]\!] \xrightarrow[\ell]{\tau} \ell[\![p]\!]}$$

$$(\mathsf{Kill1_s})\ \frac{}{\ell[\![\mathsf{kill}\, m.p]\!] \xrightarrow[\ell\wedge\overline{m}]{\tau} \ell[\![p]\!]} \qquad (\mathsf{Kill2_s})\ \frac{}{\ell[\![\mathsf{kill}\, m.p]\!] \xrightarrow[\ell\wedge m]{kill\, m} \ell[\![p]\!]}$$

$$(\mathsf{Live_s})\ \frac{}{\ell[\![\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \xrightarrow[\ell\wedge m]{\tau} \ell[\![p]\!]} \quad (\mathsf{Dead_s})\ \frac{}{\ell[\![\mathsf{if}\ m\ \mathsf{then}\ p\ \mathsf{else}\ q]\!] \xrightarrow[\ell\wedge\overline{m}]{\tau} \ell[\![q]\!]}$$

$$(\mathsf{Spawn_s})\ \frac{}{\ell[\![\mathsf{move}(k,\, p)]\!] \xrightarrow[\ell]{\tau} k[\![p]\!]} \qquad (\mathsf{Sum_s})\ \frac{\ell[\![p_j]\!] \xrightarrow[\pi]{\mu} \ell[\![p'_j]\!]}{\ell[\![\sum_{i\in I} p_i]\!] \xrightarrow[\pi]{\mu} \ell[\![p'_j]\!]}\ j\in I$$

$$(\mathsf{Str_s})\ \frac{P\equiv P' \quad P' \xrightarrow[\pi]{\mu} Q' \quad Q'\equiv Q}{P \xrightarrow[\pi]{\mu} Q} \qquad (\mathsf{Def_s})\ \frac{\ell[\![p]\!] \xrightarrow[\pi]{\mu} \ell[\![p']\!]}{\ell[\![A]\!] \xrightarrow[\pi]{\mu} \ell[\![p']\!]}\ A \overset{def}{=} p$$

$$(\mathsf{Comm_s})\ \frac{P \xrightarrow[\pi]{a} P' \quad Q \xrightarrow[\rho]{\overline{a}} Q'}{P\,|\,Q \xrightarrow[\pi\wedge\rho]{\tau} P'\,|\,Q'} \qquad (\mathsf{Par_s})\ \frac{P \xrightarrow[\pi]{\mu} P'}{P\,|\,Q \xrightarrow[\pi]{\mu} P'\,|\,Q}$$

$$(\mathsf{Restr_s})\ \frac{P \xrightarrow[\pi]{\mu} P'}{P\backslash a \xrightarrow[\pi]{\mu} P'\backslash a}\ \mu\notin\{a,\overline{a}\} \qquad (\mathsf{Ren_s})\ \frac{P \xrightarrow[\pi]{\mu} P'}{P\langle f\rangle \xrightarrow[\pi]{f(\mu)} P'\langle f\rangle}$$

LEMMA 4.3 (STRONG TRANSITION LEMMA).

$$P \xrightarrow[L]{\mu} P' \ \textit{if and only if}\ \exists\pi\colon\ P \xrightarrow[\pi]{\mu} P' \ \textit{and}\ L\vDash\pi$$

Proof. By rule induction in both directions. □

NOTATION. We use sets of locations L in logical formulae to denote the conjunction of the literals in L. Similarly, $\overline{L}$ represents the formula $\bigwedge\left\{\overline{\ell}\mid\ell\in L\right\}$. As usual ff is shorthand for $\bigvee\varnothing$. □

4.2 Strong symbolic bisimulation

The standard definition of symbolic bisimulation [19] requires that we define entailment between formulae, which we do in the standard way:

$$\pi \Vdash \rho \ \text{ if and only if }\ \forall L\colon\ L\vDash\pi \text{ implies } L\vDash\rho$$

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

Remark 4.4. We write $\Vdash$ rather than $\vdash$ to emphasize that the relation is *semantic* entailment (and because we have already used the symbol $\vDash$). A proof system for $\Vdash$ can be found in any introductory book on logic. We use semantic entailment throughout the paper because it is sufficient for our purposes, and we are not here concerned with implementation issues. $\square$

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 for location-finite processes $P \simeq_L Q$ and $M \subseteq L$ imply $P \simeq_M Q$, this is achieved by restricting attention to *negative formulae* in the recursive definition of symbolic equivalence.

We write $\mathrm{neg}(\pi)$ for the projection of π onto the set of negative formulae, that is, the formula obtained by substituting $\mathtt{tt}$ for every occurrence of a positive atom in π. For example, $\mathrm{neg}(\ell \wedge \overline{k}) = \mathtt{tt} \wedge \overline{k}$.

Suppose that P can take a μ-transition to P' under the condition ρ and we are attempting to show that P is symbolically equivalent to Q. The definition will require a Q' that is μ-reachable under the same condition. The definition also determines the conditions under which we must subsequently compare P' and a potential Q'. These are determined by the transformations "after_μ" defined as follows:

DEFINITION 4.5.

$$\begin{aligned} M \vDash \mathrm{after}_\alpha(\rho) &\ \text{ if and only if } \ \exists L\colon\ L \vDash \rho \text{ and } M \subseteq L \\ M \vDash \mathrm{after}_{kill\,k}(\rho) &\ \text{ if and only if } \ \exists L\colon\ L \vDash \rho \text{ and } M \subseteq L \backslash \{k\} \end{aligned}$$

$\square$

Note that for any formula, $\mathrm{after}_\mu(\rho)$ is unique up to $\dashv\Vdash$, the symmetric closure of $\vDash$. Since our logic is very simple, it is straightforward to calculate $\mathrm{after}_\mu(\rho)$; a step in this direction is the following:

$$\mathrm{after}_\mu(\rho) = \begin{cases} \mathsf{ff} & \text{if } \rho \Vdash \mathsf{ff} \\ \mathrm{neg}(\rho) \wedge \overline{k} & \text{if } \mu = kill\,k \\ \mathrm{neg}(\rho) & \text{otherwise} \end{cases}$$

If ρ is unsatisfiable then $\mathrm{after}_\mu(\rho)$ is simply ff. Otherwise it corresponds to the *negative information* in ρ; if the action performed is a kill action $kill\,k$, then we must also include the requirement that k be dead, that is, $\overline{k}$.

We now have all the ingredients necessary to give our definition of strong bisimulation equivalence.

DEFINITION 4.6 (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 there exist π_i, ρ_i, and Q_i such that for all i,

(a) $\vartheta \wedge \pi \Vdash \bigvee_i \rho_i$, (c) $Q \xrightarrow[\pi_i]{\mu} Q_i$, and
(b) $\rho_i \Vdash \pi_i$, (d) $P'\ \mathcal{S}_{\mathrm{after}_\mu(\rho_i)}\ Q_i$

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

One can read the definition as follows: If $P \simeq^{\mathsf{s}}_\vartheta Q$ and $P \xrightarrow[\pi]{\mu} P'$, then there must be a way to partition the set of possible worlds which satisfy $\vartheta \wedge \pi$ (the ρ_i provide the partitions) such that in each partition, Q can make a matching move. Clauses (b) and (c) ensure that Q can move to the state Q_i, and clause (d) ensures that this state matches P' under all possible worlds allowed by ρ_i.

Note that the definition implicitly quantifies over the index set I from which the i are drawn. In particular, if $\vartheta = \mathsf{ff}$, it is sufficient to let $I = \varnothing$; therefore for any P, Q we have that $P \simeq^{\mathsf{s}}_{\mathsf{ff}} Q$. It is also true that strengthening formulae preserves bisimilarity: $P \simeq^{\mathsf{s}}_\vartheta Q$ and $\vartheta' \Vdash \vartheta$ imply $P \simeq^{\mathsf{s}}_{\vartheta'} Q$.

We aim to show that $\simeq^{\mathsf{s}}_\vartheta$ characterizes LF-equivalence in the sense that $P \simeq_L Q$ if and only if there is some negative formula ϑ such that $P \simeq^{\mathsf{s}}_\vartheta Q$ and $L \vDash \vartheta$. We examine the two implications separately.

PROPOSITION 4.7. *$P \simeq^{\mathsf{s}}_\vartheta Q$ and $L \vDash \vartheta$ imply $P \simeq_L Q$.*

Proof. Let $\mathcal{S}_K$ be defined as follows:

$$\mathcal{S}_K \stackrel{def}{=} \{\langle P, Q\rangle \mid \exists\vartheta\colon\ K \vDash \vartheta \text{ and } P \simeq^{\mathsf{s}}_\vartheta Q\}$$

If $P \simeq^{\mathsf{s}}_\vartheta Q$ and $L \vDash \vartheta$ then clearly $P\ \mathcal{S}_L\ Q$.

We now show that $\mathcal{S}_K$ is a LF-bisimulation using the characterization in Lemma 3.5. Suppose that $P\ \mathcal{S}_L\ Q$ and therefore:

$$L \vDash \vartheta \ \text{ and } \ P \simeq^{\mathsf{s}}_\vartheta Q \tag{1}$$

Further suppose that $P \xrightarrow[M]{\mu} P'$ for some $M \subseteq L$, and therefore by the Strong Transition Lemma, there must exist a π such that:

$$M \vDash \pi \ \text{ and } \ P \xrightarrow[\pi]{\mu} P' \tag{2}$$

Since $P \simeq^{\mathsf{s}}_\vartheta Q$, we know that there must be π_i, ρ_i and Q_i that satisfy the conditions of Definition 4.6. Using (1), (2) and the fact that ϑ is negative, $M \vDash \vartheta \wedge \pi$. Thus by the definition of $\simeq^{\mathsf{s}}$, there must be some j such that:

$$M \vDash \rho_j \text{ and } \rho_j \Vdash \pi_j \tag{3a}$$

$$Q \xrightarrow[\pi_j]{\mu} Q_j \tag{3b}$$

$$P' \simeq^{\mathsf{s}}_{\mathrm{after}_\mu(\rho_j)} Q_j \tag{3c}$$

We show that $Q \xrightarrow[M]{\mu} Q_j$ and $P' \;\mathcal{S}_{\mathrm{after}_\mu(M)}\; Q_j$.

From (3a), we know that $M \vDash \pi_i$. Then using the Strong Transition Lemma and (3b) we arrive at $Q \xrightarrow[M]{\mu} Q_j$. To show $P' \;\mathcal{S}_{\mathrm{after}_\mu(M)}\; Q_j$, it suffices — because of (3c) — to show that $\mathrm{iafter}_\mu(M) \vDash \mathrm{after}_\mu(\rho_j)$; but this is immediate from (3a) and the definitions. □

PROPOSITION 4.8. *$P \simeq_L Q$ implies $P \simeq^{\mathsf{s}}_{\overline{Loc \setminus L}} Q$.*

Proof. Let $\mathcal{S}_\vartheta$ be defined as follows:

$$\mathcal{S}_\vartheta \stackrel{def}{=} \{\langle P, Q\rangle \mid \forall K\colon\; K \vDash \vartheta \text{ implies } P \simeq_K Q\}$$

If $P \simeq_L Q$ then, by Lemma 3.5, $M \subseteq L$ implies $P \simeq_M Q$. Further $M \vDash \overline{Loc \setminus L}$ implies $M \subseteq L$, and therefore $P \;\mathcal{S}_{\overline{Loc \setminus L}}\; Q$.

We now show that $\mathcal{S}_\vartheta$ is a symbolic bisimulation. Suppose that $P \;\mathcal{S}_\vartheta\; Q$ and therefore:

$$\forall K\colon\; K \vDash \vartheta \text{ implies } P \simeq_K Q \tag{4}$$

Further suppose that $P \xrightarrow[\pi]{\mu} P'$. Enumerate Q's symbolic μ-transitions as $Q \xrightarrow[\pi_i]{\mu} Q_i$, and for each i let ρ_i be the boolean formula determined by

$$M \vDash \rho_i \;\text{ if and only if }\; M \vDash \pi_i \text{ and } P' \simeq_{\mathrm{iafter}_\mu(M)} Q_i \tag{5}$$

We now show that these ρ_i satisfy the conditions for a symbolic bisimulation. The requirements that $Q \xrightarrow[\pi_i]{\mu} Q_i$ and that $\rho_i \Vdash \pi_i$ are met by definition. We must only show that:

$$\vartheta \wedge \pi \Vdash \bigvee_i \rho_i \tag{6a}$$

$$\forall K\colon\; K \vDash \mathrm{after}_\mu(\rho_i) \text{ implies } P' \simeq_K Q_i \tag{6b}$$

For (6a), suppose that $M \vDash \vartheta \wedge \pi$ and therefore (using Equation 4) $P \simeq_M Q$. We show that for some j, $M \vDash \rho_j$. Using the suppositions that $P \xrightarrow[\pi]{\mu} P'$ and $M \vDash \vartheta \wedge \pi$, we can apply the Strong Transition Lemma to conclude that $P \xrightarrow[M]{\mu} P'$ and therefore that there must exist some Q' such that:

$$Q \xrightarrow[M]{\mu} Q' \text{ and } P' \simeq_{\mathrm{iafter}_\mu(M)} Q'$$

By the Strong Transition Lemma there must be some j such that $Q' = Q_j$ and $M \vDash \pi_j$. Because $M \vDash \pi_j$ and $P' \simeq_{\mathrm{iafter}_\mu(M)} Q_j$, we can use (5) to conclude that $M \vDash \rho_j$.

Finally we prove (6b). If $K \vDash \mathrm{after}_\mu(\rho_i)$ then by the definition of "after", there must be some $L \supseteq K$ such that $L \vDash \rho_i$. By (5), $P' \simeq_{\mathrm{iafter}_\mu(L)} Q_i$. Again using the definition of "after", $K \subseteq L$; therefore we may use Lemma 3.5 to conclude, as required, that $P' \simeq_K Q_i$. □

Combining these two Lemmas we obtain the following.

THEOREM 4.9. *$P \simeq_L Q$ if and only if there exists a negative formulae ϑ such that $P \simeq^{\mathsf{s}}_{\vartheta} Q$ and $L \vDash \vartheta$. In addition $(\simeq) = (\simeq^{\mathsf{s}}_{\mathtt{tt}})$.*

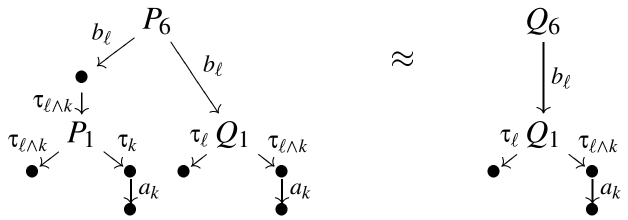
Proof. The first result follows immediately from the two previous Propositions. The result for $\simeq$ follows from that for $\simeq_L$. One direction is immediate while to prove $\simeq \subseteq \simeq^{\mathsf{s}}_{\mathtt{tt}}$ it is sufficient to take the disjunction of the ϑ_L for each L. □

4.3 Weak symbolic bisimulation

As a first attempt to define weak symbolic bisimulation, let us try simply replacing the strong transitions in Definition 4.6 with weak edges defined by conjoining formulae. For example, we would have $P \overset{\varepsilon}{\underset{\mathtt{tt}}{\Longrightarrow}} P$; also $P \overset{a}{\underset{\pi\wedge\rho}{\Longrightarrow}} P'$ if $P \overset{\varepsilon}{\underset{\pi}{\Longrightarrow}} \cdot \overset{a}{\underset{\rho}{\Longrightarrow}} P'$.

Unfortunately the equivalence resulting from this definition does not suffice. Consider the processes P_6 and Q_6 defined in Section 3.3; their symbolic transition graphs are given below (where we have written $\xrightarrow[\pi]{\mu}$ as $\xrightarrow{\mu_\pi}$ to improve readability).

$$
\begin{aligned}
P_6 &= \Big(\ell[\![b.\beta.\alpha + b.(\alpha+\tau)]\!] \mid k[\![\overline{\beta}.(\overline{\alpha}+\tau.a)+\overline{\alpha}.a]\!]\Big)\backslash\alpha\backslash\beta \\
Q_6 &= \Big(\ell[\![b.(\alpha+\tau)]\!] \mid k[\![\overline{\alpha}.a]\!]\Big)\backslash\alpha
\end{aligned}
$$



We know from Section 3.3 that $P_6 \approx Q_6$. Thus we expect P_6 and Q_6 to be related symbolically under the formula $\mathtt{tt}$; however, using the first attempted definition the relation does not hold.

$$
\boxed{\begin{aligned} P_1 &= \big(\ell[\![\alpha]\!] \mid k[\![\overline{\alpha}+\tau.a]\!]\big)\backslash\alpha \\ Q_1 &= \big(\ell[\![\alpha+\tau]\!] \mid k[\![\overline{\alpha}.a]\!]\big)\backslash\alpha \end{aligned}}
$$

The problem occurs when we try to match P_6's b-transition to P_1 with Q_6's transition to Q_1. In this case we end up comparing P_1 and Q_1 under the assumption $\mathtt{tt}$, which is equivalent to $\text{after}_b(\ell \wedge k)$, yet we have already established that P_1 and Q_1 are not LF-equivalent with respect to all live sets. As noted in Section 3.3, P_1 and Q_1 are only related under the *positive* assumption that ℓ is (initially) alive; yet "after_μ" removes all positive information from a formula.

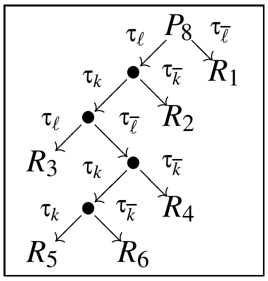
As a second attempt, we might simply change the recursive requirement of the definition (in the case that the action μ being matched is not a kill) to read $P' \mathcal{S}_{\rho_i} Q_i$, allowing positive as well as negative information to carry over into the next phase of the bisimulation. Whereas our first attempt produced an equivalence that was too strong, the revised definition is too weak. For example, the following processes would be identified though they are not barbed congruent.

$$
\begin{array}{ccc}
P_7 = \big(\ell[\![\alpha.a]\!] \mid k[\![\overline{\alpha}]\!]\big)\backslash\alpha & & Q_7 = \big(\ell[\![\alpha]\!] \mid k[\![\overline{\alpha}.a]\!]\big)\backslash\alpha \\
\big\downarrow{\scriptstyle \tau_{\ell\wedge k}} & \not\approx & \big\downarrow{\scriptstyle \tau_{\ell\wedge k}} \\
P_7' & & Q_7' \\
\big\downarrow{\scriptstyle a_\ell} & & \big\downarrow{\scriptstyle a_k} \\
\bullet & & \bullet
\end{array}
$$

Using the second definition, P_7' and Q_7' 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_7' and Q_7' finish executing their first weak action. Yet it is possible, for example, that the environment kills ℓ before P_7' performs its a-transition; Q_7' is incapable of matching this sequence of events.

From these two examples, we can see that positive information must be carried over into the recursive requirement of the symbolic version of weak LF-bisimulation, but that the use of this information is more subtle than can be expressed in our propositional logic for locations. We require a logic that is capable of expressing the *changes* in the liveset as weak actions are performed.

The next example, P_8, shows that this logic must be able to express arbitrary properties of the form "ℓ and k must have been alive, then ℓ must have died, and after that k must have died." Notice that this sequence of requirements corresponds to the state marked R_6 in the graph to the right. The conditional construct (along with $\star$) allows us to express such a process graph in the syntax of the language.



Pick an arbitrary process, say $\star[\![a]\!]$. Under what conditions is P_8 equivalent to $\star[\![a]\!]$? The equivalence holds if and only if each of the states R_i is equivalent to $\star[\![a]\!]$. To capture this requirement in the definition of symbolic bisimulation, we must find appropriate logical formulae φ_i such that $\bigvee_{i\in I}\varphi_i$ is a tautology for $I = \{1,\ldots,6\}$, but not for any smaller set I. For example, if $R_i = \star[\![a]\!]$ for $i \leq 5$, but $R_6 = \star[\![b]\!]$, then clearly P_8 is not equivalent to $\star[\![a]\!]$; this is due to the behavior of P_8 in the world in which ℓ and k are both initially alive and then both die, ℓ first. To capture such possible worlds, our logic must capture properties of *sequences* of livesets.

Our solution is to define weak symbolic edges using a *past-time temporal logic* [24, 32]. Our notion of "time" is quite restrictive: time passes only when a site fails; in addition, any two site failures must be temporally ordered — that is, failures occur one at a time. This intuition is formalized in the notion of a *live sequence*. For example, $\langle\{\ell\}, \varnothing\rangle$ is a live sequence, but $\langle\{\ell\}, \{\ell\}\rangle$ and $\langle\{\ell,k\}, \varnothing\rangle$ are not.

DEFINITION 4.10 (LIVE SEQUENCE). A *live sequence* $\mathcal{L}$ is a finite nonempty sequence of location sets $\langle L_1, \ldots, L_n\rangle$, such that for every i between 1 and $n-1$ there exists a location k such that $L_{i+1} = L_i \backslash \{k\}$. $\square$

NOTATION. We write $|\mathcal{L}|$ for the length of $\mathcal{L}$, $\mathcal{L}_{(i)}$ for the i^{th} element of $\mathcal{L}$, and $\mathcal{L}_{\langle i,j\rangle}$ for the subsequence extending from the i^{th} to the j^{th} element inclusive. If $i \leq 1$ or $i \geq |\mathcal{L}|$ then $\mathcal{L}_{(i)}$ is undefined, and similarly for $\mathcal{L}_{\langle i,j\rangle}$. $L \cdot \mathcal{L}$ denotes the live sequence obtained by prepending L to $\mathcal{L}$. Finally, we write $\mathcal{L}_{(\bullet)}$ for $\mathcal{L}_{(|\mathcal{L}|)}$, that is, the last element of $\mathcal{L}$. □

These sequences are used to give the semantics of temporal formulae φ, ψ, which may include the past-time modalities $\ominus$, $\diamondminus$ and $\boxminus$.

$\mathcal{L} \models \mathtt{tt}$	always	$\mathcal{L} \models \bigvee_{i\in I} \varphi_i$	if $\exists j \in I\colon\ \mathcal{L} \models \varphi_j$
$\mathcal{L} \models \ell$	if $\ell \in \mathcal{L}_{(\bullet)}$	$\mathcal{L} \models \ominus\varphi$	if $\mathcal{L}_{\langle 1,\|\mathcal{L}\|-1\rangle} \models \varphi$
$\mathcal{L} \models \overline{\ell}$	if $\ell \notin \mathcal{L}_{(\bullet)}$	$\mathcal{L} \models \diamondminus\varphi$	if $\exists j \leq \|\mathcal{L}\|:\ \mathcal{L}_{\langle 1,j\rangle} \models \varphi$
$\mathcal{L} \models \varphi \wedge \psi$	if $\mathcal{L} \models \varphi$ and $\mathcal{L} \models \psi$	$\mathcal{L} \models \boxminus\varphi$	if $\forall j \leq \|\mathcal{L}\|:\ \mathcal{L}_{\langle 1,j\rangle} \models \varphi$

We also adopt two abbreviations:

$$\begin{array}{rcll} \varphi \fatsemi \pi & \stackrel{def}{=} & \pi \wedge \diamondminus\varphi & \text{“}\varphi \text{ then } \pi\text{”} \\ \varphi \fatsemi^{1} \pi & \stackrel{def}{=} & \pi \wedge \ominus\varphi & \text{“}\varphi \text{ then immediately } \pi\text{”} \end{array}$$

Note that we allow only Boolean formulae π on the right-hand side of these operators. Thus they “associate to the right”; for example, $\varphi \fatsemi \pi \fatsemi \rho = (\varphi \fatsemi \pi) \fatsemi \rho$.[4]

The atomic proposition ℓ is interpreted to mean that ℓ is alive *now*, in state $\mathcal{L}_{(\bullet)}$, which is the final state of $\mathcal{L}$; it therefore follows that ℓ must have been alive at all points in the past. The proposition $\overline{\ell}$ specifies that ℓ is now dead, although it may have been alive in the past. $\boxminus\varphi$ specifies that at all points up to now, φ has been true. $\diamondminus\varphi$ specifies that at some point — now or in the past — φ was true. The formula $\varphi \fatsemi \pi$ specifies that φ was true in the past and π is true now. Note that because live sequences must be strictly decreasing, $\overline{\ell} \fatsemi \ell$ is unsatisfiable; however $\langle\{\ell\}, \varnothing\rangle \models \ell \fatsemi \overline{\ell}$.

NOTATION. For the rest of the paper we will use the symbol $\models$ only for temporal formula, whose models are live sequences. If we wish to refer to the satisfaction relation for Boolean formulae, we will add a subscript: $\models_b$. □

The definition of weak symbolic transitions, which uses formulae from our extended logic, is given in Table 5. Intuitively $P \xRightarrow[\varphi]{\mu} P'$ 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 $\varphi_1 = (\ell \wedge k) \fatsemi \ell$ and $\varphi_2 = (\ell \wedge k) \fatsemi k$ then P_7 has the

[4] The general form, $\varphi \fatsemi \psi \stackrel{def}{=} \psi \wedge \diamondminus\varphi$, is not associative, since:

$$(\varphi_1 \fatsemi \varphi_2) \fatsemi \psi_1 = \psi \wedge \diamondminus(\varphi_2 \wedge \diamondminus\varphi_1) \not\models\!\!\dashv (\psi \wedge \diamondminus\varphi_2) \wedge \diamondminus\varphi_1 = \varphi_1 \fatsemi (\varphi_2 \fatsemi \psi_1)$$

The inequality can be seen as deriving from the fact that $\diamondminus$ does not distribute through $\wedge$. An alternative is to use the “chop” operator of [28]. We have decided to use the standard operators precisely because they are standard; they may also allow for more efficient decision procedures [30].

Table 5 Weak symbolic transitions

$$
\frac{}{P \xRightarrow[\mathrm{tt}]{\varepsilon} P}
\qquad
\frac{P \xRightarrow[\varphi]{\varepsilon} \cdot \xrightarrow[\pi]{\tau} P'}{P \xRightarrow[(\varphi\wedge\pi)\fatsemi\mathrm{tt}]{\varepsilon} P'}
\qquad
\frac{P \xRightarrow[\varphi]{\mu} \cdot \xrightarrow[\pi]{\tau} P'}{P \xRightarrow[\varphi\wedge\pi]{\mu} P'}
$$

$$
\frac{P \xRightarrow[\varphi]{\varepsilon} \cdot \xrightarrow[\pi]{kill\,k} P'}{P \xRightarrow[(\varphi\wedge\pi)\fatsemi^{1}\overline{k}]{kill\,k} P'}
\qquad
\frac{P \xRightarrow[\varphi]{\varepsilon} \cdot \xrightarrow[\pi]{\alpha} P'}{P \xRightarrow[\varphi\wedge\pi]{\alpha} P'}
$$

symbolic transition $\xRightarrow[\varphi_1]{a}$ but not $\xRightarrow[\varphi_2]{a}$, whereas for Q_7 it is the opposite. Recall that the definition of P_7 and Q_7 are as follows:

$$P_7 = \big(\ell[\![\alpha.a]\!] \mid k[\![\overline{\alpha}]\!]\big)\backslash\alpha \qquad\qquad Q_7 = \big(\ell[\![\alpha]\!] \mid k[\![\overline{\alpha}.a]\!]\big)\backslash\alpha$$

The definition of weak symbolic bisimulation is similar to that for the strong case. Thus, as for the strong case, we must specify a collection of formulae with which to parameterize the recursive definition as well as an operator on formulae — corresponding to "after$_\mu$" — for generating them. Note that, unlike in the strong case, the transformation function need not be parameterized by the action μ since the relevant information is already encoded in the temporal formulae.

The formulae we choose as parameters to the relation are simply 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 instead define the function "initially" which converts Boolean formulae into temporal formulae with this interpretation in mind. The transformation function for generating formulae, which we call "finally", must then take a temporal formula and transform it into a propositional one. The definitions (unique up to $\dashv\Vdash$) are as follows:

$$
\begin{array}{lll}
\mathcal{L} \vDash \text{initially}(\pi) & \text{if and only if} & \mathcal{L}_{(1)} \vDash_b \pi \\
M \vDash_b \text{finally}(\varphi) & \text{if and only if} & \exists\mathcal{L}:\ \mathcal{L} \vDash \varphi \text{ and } M = \mathcal{L}_{(\bullet)}
\end{array}
$$

For example:

$$
\begin{aligned}
\text{initially}((\ell\wedge\overline{k})\vee m) &= ((\Diamond\ell)\wedge(\boxminus\overline{k}))\vee\Diamond m \\
\text{finally}((\ell\wedge\overline{k})\fatsemi(m\wedge n)\fatsemi^{1}(\overline{m}\wedge j)) &= \overline{k}\wedge n\wedge\overline{m}\wedge j
\end{aligned}
$$

The function "initially" is easy to calculate: on unsatisfiable formulae it is ff; on satisfiable formulae it is a homomorphism everywhere but for atoms; on atoms, initially$(\ell) = \Diamond\ell$ and initially$(\overline{\ell}) = \boxminus\overline{\ell}$. The calculation of "finally" is difficult in general; however, the results of this paper require only a well-behaved subset of formulae. We discuss the calculation of "finally" further in Appendix A.

DEFINITION 4.11 (WEAK SYMBOLIC BISIMULATION). Let $\mathcal{S}$ be a family of relations on *LProc* indexed by Boolean formulae π. $\mathcal{S}$ is a *weak symbolic bisimulation* if for every π, $\mathcal{S}_\pi$ is symmetric and whenever $P\ \mathcal{S}_\pi\ Q$ and $P \overset{\hat{\mu}}{\underset{\varphi}{\Longrightarrow}} P'$ then there exist φ_i, ψ_i, and Q_i such that for all i:

(a) $\mathsf{initially}(\pi) \wedge \varphi \Vdash \bigvee_i \psi_i$,
(b) $\psi_i \Vdash \varphi_i$,
(c) $Q \overset{\hat{\mu}}{\underset{\varphi_i}{\Longrightarrow}} Q_i$, and
(d) $P'\ \mathcal{S}_{\mathsf{finally}(\psi_i)}\ Q_i$

We write $P \approxeq^{\mathsf{s}}_\pi Q$ to indicate that there exists a weak symbolic bisimulation $\mathcal{S}$ with $P\ \mathcal{S}_\pi\ Q$.[5] □

Before presenting the alternative characterization theorem, we first discuss some of the examples. Consider P_1 and Q_1. To show these two processes symbolically bisimilar, there are two interesting transitions that must be matched. First, consider the transition $Q_1 \overset{\varepsilon}{\underset{\varphi}{\Longrightarrow}} \mathsf{nil}$, where $\varphi = \ell \,⨟\, \mathsf{tt}$. To match this, we must use two ε-transitions of P_1: $P_1 \overset{\varepsilon}{\underset{\varphi_1}{\Longrightarrow}} P_1$, where $\varphi_1 = \mathsf{tt}$ and $\psi_1 = \ell \wedge \overline{k} \,⨟\, \mathsf{tt}$, and $P_1 \overset{\varepsilon}{\underset{\varphi_2}{\Longrightarrow}} \mathsf{nil}$, where $\varphi_2 = \ell \wedge k \,⨟\, \mathsf{tt}$ and $\psi_2 = \ell \wedge k \,⨟\, \mathsf{tt}$. These choices for the ψ_i, meet all of the requirements for the definition, even if we take $\pi = \mathsf{tt}$. In particular, $\varphi \Vdash \psi_1 \vee \psi_2$, $P_1 \approxeq^{\mathsf{s}}_{\mathsf{finally}(\varphi_1)} \mathsf{nil}$ where $\mathsf{finally}(\varphi_1) = \overline{k}$, and $\mathsf{nil} \approxeq^{\mathsf{s}}_{\mathsf{finally}(\varphi_2)} \mathsf{nil}$ where $\mathsf{finally}(\varphi_2) = \mathsf{tt}$.

Second, consider the transition $P_1 \overset{a}{\underset{\varphi}{\Longrightarrow}} \mathsf{nil}$, for $\varphi = k \,⨟\, k$. Q_1 cannot match this transition at tt because Q_1's only a-transition is parameterized by $\ell \wedge k \,⨟\, k$ and k does not entail $\ell \wedge k \,⨟\, k$; however, the transitions can be matched under the assumption $\pi = \ell$ since $(\diamondsuit\ell) \wedge k$ *does* entail $\ell \wedge k \,⨟\, k$.

P_6 and Q_6 then, are related at tt, since the definition ensures that P_1 and Q_1 are compared under the assumption that ℓ is alive. By the same token, P_7 and Q_7 can only be related at formulae that entail $\overline{\ell} \vee \overline{k}$; P_7' and Q_7' are also related under the assumption $\ell \wedge k$, but neither P_7 nor Q_7 can generate such a formula, due to the weakening that happens in ε-transitions. Note that the construction of weak symbolic edges, which differs for visible and non-visible actions, is crucial in achieving the correct results for these examples.

THEOREM 4.12. *$P \approxeq_L Q$ if and only if there exists a Boolean formula π such that $P \approxeq^{\mathsf{s}}_\pi Q$ and $L \vDash_b \pi$. Hence $(\approxeq) = (\approxeq^{\mathsf{s}}_{\mathsf{tt}})$.*

We devote the rest of this section to the proof of this theorem. As a first step we must relate the symbolic moves to (sequences of) concrete actions. This is achieved by defining concrete moves which are parameterized by live sequences:

[5] If one uses $P \overset{\mu}{\underset{\rho}{\rightarrow}} P'$ rather than $P \overset{\hat{\mu}}{\underset{\varphi}{\Longrightarrow}} P'$ in the definition, then clause (a) becomes:

$$\begin{array}{lll} \mathsf{initially}(\pi) \wedge (\rho \,⨟\, \mathsf{tt}) & \Vdash \bigvee_i \psi_i, & \text{if } \mu = \tau \\ \mathsf{initially}(\pi) \wedge (\rho \,⨟^{a}\, \overline{k}) & \Vdash \bigvee_i \psi_i, & \text{if } \mu = kill\,k \\ \mathsf{initially}(\pi) \wedge \ \rho & \Vdash \bigvee_i \psi_i, & \text{otherwise} \end{array}$$

DEFINITION 4.13. For each live sequence $\mathcal{L}$ and $\hat{\mu} \in KAct_{\varepsilon}$, $\overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}}$ is the least relation satisfying the following:

$$
\begin{aligned}
P \overset{\hat{\alpha}}{\underset{\langle L\rangle}{\Longmapsto}} P' &\text{ if } P \overset{\hat{\alpha}}{\underset{L}{\Longrightarrow}} P' \\
P \overset{kill\,k}{\underset{\langle L, L\setminus\{k\}\rangle}{\Longmapsto}} P' &\text{ if } P \overset{kill\,k}{\underset{L}{\Longrightarrow}} P' \\
P \overset{\hat{\mu}}{\underset{L\cdot\mathcal{L}}{\Longmapsto}} P'' &\text{ if } P \overset{fail\,k}{\underset{L}{\Longrightarrow}} \cdot \overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}} P' \text{ and } \mathcal{L}_{(1)} = L\setminus\{k\}
\end{aligned}
\qquad \square
$$

Note that this use of the symbol $\Longmapsto$ is entirely different from that of Section 2.2. Also recall that $P \overset{fail\,k}{\underset{L}{\Longrightarrow}} P''$ if and only if $k \in L$ and $P \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus\{k\}}{\Longrightarrow}} P''$. In order to relate these transitions to weak symbolic transitions, we first provide an alternate view of the symbolic transitions.

LEMMA 4.14. (a) $P \overset{\varepsilon}{\underset{\varphi}{\Longrightarrow}} P'$ *if and only if there exist* P_i, π_i *and* h *such that* $1 \le h$, $P_1 = P$, $P_h = P'$, *and the following hold:*

$$
\begin{gathered}
\textit{for every } 1 \le i < h,\ P_i \overset{\tau}{\underset{\pi_i}{\longrightarrow}} P_{i+1},\ \textit{and} \\
\varphi \dashv\Vdash \pi_1 \mathbin{⨟} \ldots \mathbin{⨟} \pi_{h-1} \mathbin{⨟} \mathtt{tt}
\end{gathered}
$$

(b) $P \overset{\mu}{\underset{\varphi}{\Longrightarrow}} P'$ *if and only if there exist* P_i, π_i, h *and* n *such that* $1 \le h \le n$, $P_1 = P$, $P_{n+1} = P'$, *and the following hold:*

$$
\begin{gathered}
\textit{for every } 1 \le i \le n,\ i \ne h,\ P_i \overset{\tau}{\underset{\pi_i}{\longrightarrow}} P_{i+1},\ \textit{and} \\
\textit{if } \mu = \alpha \quad \textit{then } P_h \overset{\alpha}{\underset{\pi_h}{\longrightarrow}} P_{h+1} \textit{ and } \varphi \dashv\Vdash \pi_1 \mathbin{⨟} \ldots \mathbin{⨟} \pi_{h-1} \mathbin{⨟} \pi_h \wedge \pi_{h+1} \wedge \ldots \wedge \pi_n \\
\textit{if } \mu = kill\,k \textit{ then } P_h \overset{kill\,k}{\underset{\pi_h}{\longrightarrow}} P_{h+1} \textit{ and } \varphi \dashv\Vdash \pi_1 \mathbin{⨟} \ldots \mathbin{⨟} \pi_{h-1} \mathbin{⨟} \pi_h \mathbin{⨟}^{1} \pi_{h+1} \wedge \ldots \wedge \pi_n
\end{gathered}
$$

Proof. The forward direction (only if) follows by rule induction. The reverse direction follows by induction on n. $\square$

LEMMA 4.15 (WEAK TRANSITION LEMMA).

$$
P \overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}} P' \ \textit{ if and only if } \exists\psi:\ P \overset{\hat{\mu}}{\underset{\psi}{\Longrightarrow}} P' \textit{ and } \mathcal{L} \vDash \psi
$$

Proof. In both directions by induction on the definition of weak transitions, using the Strong Transition Lemma and Lemma 4.14. $\square$

The proof of the theorem depends on the following characterization of LF-bisimulation equivalence (compare Lemma 3.3).

LEMMA 4.16. $\mathcal{S}$ *is a weak LF-bisimulation if and only if for every* L, $\mathcal{S}_L$ *is symmetric and whenever* $P\ \mathcal{S}_L\ Q$*:*

$$
\mathcal{L}_{(1)} = L \textit{ and } P \overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}} P' \ \textit{ imply } \exists Q':\ Q \overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}} Q' \textit{ and } P'\ \mathcal{S}_{\mathcal{L}_{(\bullet)}}\ Q'
$$

Proof. Straightforward. $\square$

We now prove the main theorem, treating each direction separately.

PROPOSITION 4.17. *For any Boolean formula π, if $P \approxeq^{s}_{\pi} Q$ and $L \vDash_b \pi$ then $P \approxeq_L Q$.*

Proof. Let $\mathcal{S}_K$ be defined as follows: $\mathcal{S}_K \stackrel{def}{=} \{\langle P, Q\rangle \mid \exists\pi\colon\ K \vDash_b \pi \text{ and } P \approxeq^{s}_{\pi} Q\}$. If $P \approxeq^{s}_{\pi} Q$ and $L \vDash_b \pi$ then $P\ \mathcal{S}_L\ Q$.

Using the characterization given above we now show that $\mathcal{S}_K$ is an LF-bisimulation. Suppose that $P\ \mathcal{S}_{\mathcal{L}_{(1)}}\ Q$ and therefore we fix a Boolean formula π such that $P \approxeq^{s}_{\pi} Q$ and $\mathcal{L}_{(1)} \vDash_b \pi$, that is, $\mathcal{L} \vDash \text{initially}(\pi)$. Further suppose that $P \overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}} P'$. Using the Weak Transition Lemma, there must exist a φ such that:

$$\mathcal{L} \vDash \varphi \ \text{ and } \ P \overset{\hat{\mu}}{\underset{\varphi}{\Longrightarrow}} P' \tag{7}$$

Since $P \approxeq^{s}_{\pi} Q$, we know that there must be some j such that:

$$\mathcal{L} \vDash \psi_j \text{ and } \psi_j \Vdash \varphi_j \tag{8a}$$
$$Q \overset{\hat{\mu}}{\underset{\varphi_j}{\Longrightarrow}} Q_j \tag{8b}$$
$$P' \approxeq^{s}_{\text{finally}(\psi_j)} Q_j \tag{8c}$$

From (8a), we know that $\mathcal{L} \vDash \varphi_i$. Using the Weak Transition Lemma and (8b) we may conclude that $Q \overset{\hat{\mu}}{\underset{\mathcal{L}}{\Longmapsto}} Q_j$. It remains only to show that $P'\ \mathcal{S}_{\mathcal{L}_{(\bullet)}}\ Q_j$, but this follows using (8c) and the definition of "finally" $\square$

PROPOSITION 4.18. *If $P \approxeq_L Q$ then $P \approxeq^{s}_{L \wedge \overline{Loc \setminus L}} Q$.*

Proof. Let $\mathcal{S}_\pi$ be defined as follows:

$$\mathcal{S}_\pi \stackrel{def}{=} \left\{ \langle P, Q\rangle \mid \forall K\colon\ K \vDash_b \pi \text{ implies } P \approxeq^{s}_{K} Q \right\}$$

If $\pi = L \wedge \overline{Loc \setminus L}$ then clearly $L \vDash_b \pi$; thus if $P \approxeq_L Q$ then $P\ \mathcal{S}_\pi\ Q$.

We now show that $\mathcal{S}_\pi$ is a symbolic bisimulation. Suppose that $P\ \mathcal{S}_\pi\ Q$; thus:

$$\forall K\colon\ K \vDash_b \pi \text{ implies } P \simeq_K Q \tag{9}$$

Further suppose that $P \overset{\hat{\mu}}{\underset{\varphi}{\Longrightarrow}} P'$. Enumerate Q's symbolic $\hat{\mu}$-transitions as $Q \overset{\hat{\mu}}{\underset{\varphi_i}{\Longrightarrow}} Q_i$, and let ψ_i be the temporal formula characterized by

$$\mathcal{M} \vDash \psi_i \ \text{ if and only if } \ \mathcal{M} \vDash \varphi_i \text{ and } P' \simeq_{\mathcal{M}_{(\bullet)}} Q_i \tag{10}$$

We now show that these ψ_i satisfy the conditions for a symbolic bisimulation. The requirements that $Q \overset{\hat{\mu}}{\underset{\varphi_i}{\Longrightarrow}} Q_i$ and that $\psi_i \Vdash \varphi_i$ are met by definition. We must only show that:

$$\text{initially}(\pi) \wedge \varphi \ \Vdash\ \bigvee_i \psi_i \tag{11a}$$
$$\forall K\colon\ K \vDash_b \text{finally}(\psi_i) \ \text{ implies } \ P' \simeq_K Q_i \tag{11b}$$

For (11a), suppose that $\mathcal{M} \vDash \mathrm{initially}(\pi) \wedge \varphi$ and therefore, using (9), $P \simeq_{\mathcal{M}_{(1)}} Q$. We show that for some j, $\mathcal{M} \vDash \psi_j$. Using the suppositions that $P \overset{\hat{\mu}}{\underset{\varphi}{\Longrightarrow}} P'$ and $\mathcal{M} \vDash \mathrm{initially}(\pi) \wedge \varphi$, we can apply the Weak Transition Lemma to conclude that $P \overset{\hat{\mu}}{\underset{\mathcal{M}}{\models\!\!\Longrightarrow}} P'$ and therefore, since $P \simeq_{\mathcal{M}_{(1)}} Q$, that there exists some Q' such that:

$$Q \overset{\hat{\mu}}{\underset{\mathcal{M}}{\models\!\!\Longrightarrow}} Q' \text{ and } P' \simeq_{\mathcal{M}_{(\bullet)}} Q'$$

By the Weak Transition Lemma there must be some j such that $Q' = Q_j$ and $\mathcal{M} \vDash \varphi_j$. Because $\mathcal{M} \vDash \varphi_j$ and $P' \simeq_{\mathcal{M}_{(\bullet)}} Q_j$, we can use (10) to conclude that $\mathcal{M} \vDash \psi_j$.

Finally we prove (11b). Suppose that $K \vDash_b \mathrm{finally}(\psi_i)$. Then there must exist some $\mathcal{L}$ such that $\mathcal{L}_{(\bullet)} = K$ and $\mathcal{L} \vDash \psi_i$. By (10), $P' \simeq_{\mathcal{L}_{(\bullet)}} Q_i$, and therefore we have, as required, that $P' \simeq_K Q_i$. □

5 Basic processes

In this section we turn our attention to the semantics of *basic processes*. In order examine the behavior of such processes using our operational semantics we need to locate them at a specific site. Moreover it is rather obvious that the choice of this site cannot be ignored. For example, if $p = \mathsf{kill}\,\ell \mid a$, then the meaning of $\ell[\![p]\!]$ is different from that of $k[\![p]\!]$:

$$\ell[\![\mathsf{kill}\,\ell \mid a]\!] \;\dot{\sim}\; \ell[\![\tau + a.\tau]\!] \;\dot{\not\approx}\; \ell[\![\tau \mid a]\!] \;\dot{\sim}\; k[\![\mathsf{kill}\,\ell \mid a]\!]$$

Another example of this is $p = \mathsf{move}(\ell, a) \mid b$.

An interesting feature of basic processes is that they determine the semantics of all located processes; any located process P can be translated into a primitive processes p such that $P \approxeq_L \star[\![p]\!]$. (For such a translation to hold generally, we believe that the use of the immortal location is essential.) The translation is defined as follows:

$$(\ell[\![p]\!])^\bullet = \mathsf{move}(\ell, p) \qquad (P\backslash a)^\bullet = P^\bullet\backslash a$$
$$(P \mid Q)^\bullet = P^\bullet \mid Q^\bullet \qquad (P\langle f\rangle)^\bullet = P^\bullet\langle f\rangle$$

THEOREM 5.1. *For any L, $P \approxeq_L \star[\![P^\bullet]\!]$*

Proof. By induction on the structure of P. The proof uses the fact that for any L, $\star[\![\mathsf{move}(\ell, p)]\!] \approxeq_L \ell[\![\tau.p]\!]$ and $\ell[\![\tau.p]\!] \approxeq_L \ell[\![p]\!]$. □

This theorem suggests that it might be appropriate to define a semantic equivalence between basic processes by comparing their behavior at the immortal site $\star$. However this would ignore important behavior of processes, namely what they can do when their principal site fails.

Instead we suggest that the semantics of basic processes should be defined by comparing their behavior at some arbitrary new location, different from $\star$. The following lemmas show that it the choice of new location does not matter. First a lemma about weak symbolic bisimulation equivalence.

LEMMA 5.2. *Suppose that $\ell \neq \star$. Then $P \approxeq^{s}_{\vartheta} Q$ implies $P\{^k/_\ell\} \approxeq^{s}_{\vartheta\{^k/_\ell\}} Q\{^k/_\ell\}$.*

Proof. The proof depends on the following properties of the symbolic operational semantics which are easily established by rule induction.

1. $P \xrightarrow[\varphi]{\mu} Q$ implies $P\{^k/_\ell\} \xrightarrow[\varphi\{^k/_\ell\}]{\mu\{^k/_\ell\}} Q\{^k/_\ell\}$
2. $P\{^k/_\ell\} \xrightarrow[\varphi']{\mu'} Q'$ implies $\exists \mu, \varphi, Q$: $P \xrightarrow[\varphi]{\mu} Q$ where $\mu' = \mu\{^k/_\ell\}$, $Q' = Q\{^k/_\ell\}$, and $\varphi' = \varphi\{^k/_\ell\}$.

It also uses the fact that $\varphi \Vdash \psi$ implies $\varphi\{^k/_\ell\} \Vdash \psi\{^k/_\ell\}$, the proof of which can be found in, for example, [24]. $\square$

As an immediate corollary we have the following:

COROLLARY 5.3. *If ℓ and k are different from $\star$ and neither appear in the basic processes p, q then $\ell[\![p]\!] \approxeq_L \ell[\![q]\!]$ implies $k[\![p]\!] \approxeq_L k[\![q]\!]$.*

Proof. Follows from the previous lemma and Theorem 4.12. $\square$

With this result we have a natural way in which to extend semantic equivalences from located processes to basic processes:

DEFINITION 5.4. For any relation $\mathcal{R}$ on located processes, we extend $\mathcal{R}$ to basic processes as follows:

$$p \;\mathcal{R}\; q \quad \stackrel{def}{\Leftrightarrow} \quad \ell[\![p]\!] \;\mathcal{R}\; \ell[\![q]\!], \text{ where } \ell \notin (\mathrm{locs}(p) \cup \mathrm{locs}(q)) \qquad \square$$

We examine two such equivalences, $\approxeq_L$, parameterized on the live set L, and $\approxeq$. It turns out that the former, which behave well on located processes, are unsuitable for basic processes; they are preserved only by a very restricted class of contexts:

LEMMA 5.5. *Suppose that $\ell[\![p]\!] \approxeq_L \ell[\![q]\!]$, then:*

$$\ell[\![\tau.p]\!] \approxeq_L \ell[\![\tau.q]\!] \qquad \ell[\![\mathsf{if}\ k\ \mathsf{then}\ p]\!] \approxeq_L \ell[\![\mathsf{if}\ k\ \mathsf{then}\ q]\!]$$
$$\ell[\![\mathsf{move}(k,\, p)]\!] \approxeq_L \ell[\![\mathsf{move}(k,\, q)]\!]$$

Proof. Immediate from the definitions. $\square$

These equivalences are not preserved by other dynamic contexts, as we show by examples. The examples are based on P_1 and Q_1 and their translations p_1 and q_1 (via $(\,)^\bullet$):

$$Q_1 = \big(\ell[\![\alpha+\tau]\!] \mid k[\![\overline{\alpha}.a]\!]\big)\backslash\alpha \qquad q_1 = \big(\mathsf{move}(\ell,\, \alpha+\tau) \mid \mathsf{move}(k,\, \overline{\alpha}.a)\big)\backslash\alpha$$
$$P_1 = \big(\ell[\![\alpha]\!] \mid k[\![\overline{\alpha}+\tau.a]\!]\big)\backslash\alpha \qquad p_1 = \big(\mathsf{move}(\ell,\, \alpha) \mid \mathsf{move}(k,\, \overline{\alpha}+\tau.a)\big)\backslash\alpha$$

For $L = \{\ell, k\}$, we have already established that $P_1 \approxeq_L Q_1$, and therefore the same holds for $\star[\![p_1]\!]$ and $\star[\![p_2]\!]$. It is also not difficult to show that $m[\![p_1]\!] \approxeq_L m[\![p_2]\!]$

for some new location m. But $m[\![a.p_1]\!] \not\approx_L m[\![a.q_2]\!]$. The reason for this is that the environment can kill ℓ before a is executed, forcing the matching process to do the same; thus p_1 and p_2 are compared under the liveset $\{k\}$, and they are clearly different under this liveset. Thus $\approx_L$ is not preserved by action prefixing.

Exactly the same reasoning can be used to show that $\approx_L$ is not preserved by the contexts $\mathsf{kill}\,\ell.[\cdot]$ or $\mathsf{if}\ \overline{\ell}\ \mathsf{then}\ [\cdot]$. Less obviously, we can adapt the example to also show that $\approx^{\mathsf{s}}_L$ is not preserved by contexts of the form $\mathsf{if}\ m\ \mathsf{then}\ [\cdot]\ \mathsf{else}\ q$. We consider the context $\mathbb{P}[\cdot] = \mathsf{if}\ m\ \mathsf{then}\ [\cdot]\ \mathsf{else}\ b$. The graphs for $\mathbb{P}[p_1]$ and $\mathbb{P}[q_1]$ are given below:

$$\begin{tikzcd} & \mathbb{P}[p_1] \arrow[d, "\tau_m"'] \arrow[dr, "b_{\overline{m}}"] & \\ & p_1 \arrow[dl, Rightarrow, "\varepsilon_{\ell\wedge k}"'] \arrow[ddr, Rightarrow, "\varepsilon_k"] & \bullet \\ \bullet & & \\ & & \bullet \arrow[d, "a_k"] \\ & & \bullet \end{tikzcd} \quad \not\approx_{\{\ell,k,m\}} \quad \begin{tikzcd} & \mathbb{P}[q_1] \arrow[d, "\tau_m"'] \arrow[dr, "b_{\overline{m}}"] & \\ & q_1 \arrow[dl, Rightarrow, "\varepsilon_{\ell}"'] \arrow[ddr, Rightarrow, "\varepsilon_{\ell\wedge k}"] & \bullet \\ \bullet & & \\ & & \bullet \arrow[d, "a_k"] \\ & & \bullet \end{tikzcd}$$

Suppose that when comparing $\mathbb{P}[p_1]$ and $\mathbb{P}[q_1]$ under the liveset $\{\ell, k, m\}$, location ℓ fails. After the failure of ℓ, $\mathbb{P}[p_1]$ still has available to it the action b, in case m fails, and the action a, in case m remains alive. $\mathbb{P}[q_1]$ cannot reach any matching state. If it remains at $\mathbb{P}[q_1]$, it loses the ability to perform the a action; if it moves to q_1, it loses the ability to perform the b.

The relation $\approx$ is more suitable for basic processes.

LEMMA 5.6. *Suppose that $\ell[\![p]\!] \approx \ell[\![q]\!]$, then:*

$$\begin{array}{rr} \ell[\![\alpha.p]\!] \approx \ell[\![\alpha.q]\!] & \ell[\![\mathsf{if}\ k\ \mathsf{then}\ p\ \mathsf{else}\ r]\!] \approx \ell[\![\mathsf{if}\ k\ \mathsf{then}\ q\ \mathsf{else}\ r]\!] \\ \ell[\![\mathsf{kill}\,k.p]\!] \approx \ell[\![\mathsf{kill}\,k.q]\!] & \ell[\![\mathsf{if}\ k\ \mathsf{then}\ r\ \mathsf{else}\ p]\!] \approx \ell[\![\mathsf{if}\ k\ \mathsf{then}\ r\ \mathsf{else}\ q]\!] \\ \ell[\![\mathsf{move}(k,\,p)]\!] \approx \ell[\![\mathsf{move}(k,\,q)]\!] & \end{array}$$

Proof. Straightforward calculations. □

Unfortunately $\approx$ suffers from one of the standard problems of CCS bisimulation: it is not preserved by choice, which is a context for basic processes. As usual, however, a minor adjustment is sufficient to turn it into a congruence:

DEFINITION 5.7 (LF-CONGRUENCE). We say that P and Q are *LF-congruent at* L ($P \overset{c}{\approx}_L Q$) if $P \approx_L Q$ and

$$P \xrightarrow[L]{\tau} P' \text{ implies } \exists Q':\ Q \overset{\tau}{\underset{L}{\Longrightarrow}} Q' \text{ and } P' \approx_L Q'$$
$$Q \xrightarrow[L]{\tau} Q' \text{ implies } \exists P':\ P \overset{\tau}{\underset{L}{\Longrightarrow}} P' \text{ and } P' \approx_L Q'$$

P and Q are *LF-congruent* ($P \overset{c}{\approx} Q$) if for every L: $P \overset{c}{\approx}_L Q$. □

The following theorem shows that $\overset{c}{\approx}$, when lifted to basic processes, is a congruence and moreover it coincides with barbed congruence:

THEOREM 5.8. *For location-finite processes, $p \overset{c}{\cong} q$ if and only if $p \overset{c}{\approx} q$.*

Proof. One can show that $\overset{c}{\cong}$ is a congruence by using standard techniques to adapt Lemma 5.6. The interesting problem is to show that $\overset{c}{\approx} \subseteq \overset{c}{\cong}$, that is, for every liveset L, $\overset{c}{\approx} \subseteq \overset{c}{\cong}_L$. The contexts are based on those of Section 3.3, which we assume the reader has fresh in mind.

For finite set of locations $K = \{k_1, \ldots, k_n\}$, write "$\mathsf{if}\ \overline{K}\ \mathsf{then}\ p$" for "$\mathsf{if}\ \overline{k}_1\ \mathsf{then} \ldots \mathsf{if}\ \overline{k}_n\ \mathsf{then}\ p$". Let p and q have location sort J (that is, $\mathrm{locs}(p,q) \subseteq J$). We suppose some additional actions in Act_2: $\{initial_M \mid M \subseteq J\}$ and b. Let $\mathcal{S}$ be as in Section 3.3, and $\mathbb{P}[\cdot]$ be defined as follows:

$$\mathbb{P}[\cdot] \overset{def}{=} \textstyle\sum_{M \subseteq J} \mathsf{if}\ \overline{J \backslash M}\ \mathsf{then}\ initial_M.\,([\cdot] + b)$$
$$\mathcal{R} \overset{def}{=} \{\langle p,q \rangle \mid \mathbb{P}[p]\ \mathcal{S}_L\ \mathbb{P}[q]\}$$

It is straightforward to show that $\mathcal{R}$ satisfies the conditions of Definition 5.7. □

6 Conclusions

In this paper we have investigated the effect of site failure on the semantics of simple distributed processes. By applying the theory of *barbed bisimulations* [31] to an operational semantics for *located processes* we have justified new forms of bisimulation equivalences appropriate to such processes in the presence of potential site failure. We also gave *symbolic* characterizations of the new equivalences which means that they can, in principle, be investigated using the symbolic methods of [19]. In the weak case the development of this symbolic equivalence, rather surprisingly, required the use of a temporal logic with a past time modality. The discussion in Section 4 should convince the reader that reasoning about the effect of site failures, even for our simplified language, is very complicated; we feel that our logic is a convenient tool for expressing the subtleties involved.

The equivalences we have defined are quite robust in the sense that for many variations of the operators in our language, barbed equivalence and LF-equivalence coincide. For example, barbed equivalence does not change if we remove the conditional from the language, nor if we strengthen the conditional so that it does not perform an initial τ-action. It is also unchanged if one removes the spawn operator but retains the conditional. Neither does it change if we disallow terms of the form $\mathsf{kill}\, k.p$ where $p \neq \mathsf{nil}$. Further it is unaffected if one allows distributed choices, using a syntax closer to that of [3]. We conjecture that our results can extended in a straightforward way to value-passing languages and to languages based on the pi-calculus which retain the assumption that all failures are independent, such as the languages in [3, 4]. More delicate is the extension to languages such as the distributed join-calculus [16] 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.

Even more problematic is the extension to languages such as [20] in which channel names have only local significance. In our language channel names have global scope, which introduces interdependencies between locations. For example in the system

$$\ell[\![\overline{a}.p + b.q]\!] \mid k[\![a.p' + \overline{b}.q']\!]$$

the resolution of the choice at location ℓ, say, requires the co-operation of the location k. Nevertheless we also conjecture that our main characterization theorem, Theorem 3.11, can also be extended to languages in which choice is purely local and channel names are localized, as for example in [20], and outputs are asynchronous, as for example in [5].

Related work. Site failure has also played a role in languages studied in [3, 4, 16]. In these papers abstract languages based on Facile [17] or the pi-calculus [26, 5] are studied. There the main thrust has been to understand the implications of site failure by providing translations from a distributed source language into a simpler (non-distributed) target, proving that the translations are adequate or fully abstract for subsets of the original language. For example, the translation of [4] is shown to be fully-abstract for *location closed systems*, intuitively those in which the environment, or context, can neither kill locations nor test their status.

Here we have sought to understand the effect of site failure directly in terms of a bisimulation equivalence on the source language, rather than indirectly in terms of properties of a translation into a simpler language. Although we have addressed a simpler language, we believe that our results can be extended to languages of comparable complexity.

A number of location-based equivalences already exist in the literature [8, 9, 27, 29, 11]; 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. Indeed in all but the last two of these papers the identity of locations is unimportant. In Appendix B we give a series of counter-examples which show that $\approxeq$ is incomparable with all of the equivalences proposed in these papers.

Implementation issues. For finite-state processes, one can check LF-bisimulation automatically, either by using the concrete semantics and a tool such as the Concurrency Workbench [10], or by using the symbolic semantics and adapting the algorithm given by Hennessy and Lin [19]. In implementing the symbolic techniques, it would be convenient to have a decision procedure for entailment between formulae. In the strong case, where the formulae are Boolean, such decision procedures are well known. In the weak case, in which we use a linear-time temporal logic, more work is required.

Since we allow only a restricted class of models for our temporal formulae, the usual axiomatizations of linear-time temporal logic [24] do not directly apply.

However, we speculate that a proof system for our logic (or a conservative extension of it) can be derived from the standard axiomatization by adding three axiom schemas: 1. if ℓ is dead, then it must be dead at all points in the future; 2. if ℓ is alive, then it must be alive at all points in the past; 3. at each increment of time, exactly one site dies. One way to approach the implementation would be to marry a tool for temporal logic, such as the StEP prover [15], to the existing implementation of Hennessy and Lin's algorithm.

Other models of failure. We have assumed a simple model in which failures are permanent and independent and the number of failures that can occur is unbounded. Our approach can also be adapted to other models of failure.

For example, one may wish to consider a language in which multiple sites can be killed simultaneously (for example, using an operator $\mathsf{kill}\,L.p$ where $L \subseteq Loc$). Such a model may be of interest if communication links are subject to failure and we wish our processes to be equivalent regardless of the network topology; failure of a communication link may be modeled by the failure of all nodes connected to the link. In the weak case, the induced equivalence is defined, using the concrete semantics, by adapting Lemma 3.7, changing clause (b) to read:

$$\text{for every } K \subseteq L \quad \exists Q' : \; Q \overset{\varepsilon}{\underset{L}{\Longrightarrow}} \cdot \overset{\varepsilon}{\underset{L\setminus K}{\Longrightarrow}} Q' \text{ and } P \, \mathcal{S}_{L\setminus K} \, Q'$$

Using the symbolic semantics, we need simply enlarge the class of models for the logic, relaxing the restriction that between two states in a live sequence exactly one site must fail. Obviously this change in the definition of live sequences will also change the entailment relation between formulae.

Perhaps a more interesting change would be to limit the number of failures that can occur. Such models of failure are often used in the distributed-algorithms literature. This model could be accommodated in the concrete semantics simply by changing the definition of the predicate "fallible" given in Section 3. In the symbolic case, one could again accommodate the new model by changing the definition of live sequences (to contain a minimum number of locations), with a corresponding change in the axiomatization of entailment.

The symbolic approach is particularly attractive because of its modularity. In the concrete case, these models of failure require changes in the transition system or in the definition of bisimulation, whereas in the symbolic semantics only the proof system for entailment need be changed.

One might also wish to relax the assumption that failures are permanent, replacing the kill operator with the operators *pause* and *resume*. In this case, the induced equivalence is much finer than LF-equivalence. (For location-finite processes we believe that it will be at least as fine as LF/LA-equivalence, which we discuss in Appendix B.) We leave the precise characterization of barbed congruence for such a language to future work.

Acknowledgement

We thank Alan Jeffrey and Flavio Corradini, both of whom made important comments and suggestions in the early stages of this work. We also appreciate the useful comments of the anonymous referees.

A Computing "finally"

We remind the reader of the definition of "finally" and also define the auxiliary transformation "$\underline{\text{neg}}$", which we use in this appendix:

$$\begin{array}{lll} M \vDash_b \text{finally}(\varphi) & \text{if and only if } \exists \mathcal{L}\colon & \mathcal{L} \vDash \varphi \text{ and } M = \mathcal{L}_{(\bullet)} \\ M \vDash_b \underline{\text{neg}}(\varphi) & \text{if and only if } \exists \mathcal{L}\colon & \mathcal{L} \vDash \varphi \text{ and } M \subseteq \mathcal{L}_{(\bullet)} \end{array} \tag{12}$$

Ignoring $\ominus$ for the moment, we might hope to compute "finally" using a function f which is a homomorphism everywhere but for the temporal operators $\diamondminus$ and $\boxminus$; for these operators $f(\boxminus\varphi) = f(\varphi)$ and $f(\diamondminus\varphi) = \underline{\text{neg}}(\varphi)$. Such a definition, however, will fail on the full logic, even for satisfiable formulae. For example, if

$$\varphi_1 = (k \wedge \boxminus\overline{\ell}) \vee (\ell \wedge \overline{k} \wedge \diamondminus k) \qquad\qquad \varphi_2 = (k \wedge \boxminus\overline{\ell}) \vee (\ell \wedge \diamondminus\overline{k})$$

then finally$(\varphi_1 \wedge \varphi_2) = k \wedge \overline{\ell}$, whereas $f(\varphi_1 \wedge \varphi_2) = (k \wedge \overline{\ell}) \vee (\ell \wedge \overline{k})$.

The problem is the full generality of conjunction. Fortunately, none of our results require this generality; in particular, we can limit our attention (modulo $\dashv\Vdash$) to formulae generated by the following BNF:

$$\begin{array}{lll} \vartheta & ::= & \pi \mid \vartheta \,\text{;}\, \pi \\ & & \mid (\vartheta \wedge \ell) \,\text{;}^{1}\, (\overline{\ell} \wedge \pi), \text{ where } \ell \text{ does not appear in } \vartheta \text{ or } \pi \\ \varphi & ::= & \bigvee_i (\vartheta_i \wedge \text{initially}(\pi_i)) \end{array}$$

We call such formulae *admissible*.

To substantiate our claim that this set of formulae is sufficient, note that all the formulae decorating weak symbolic edges are admissible, as are all initial formulae. In addition, the formulae ψ_i used in the definition of weak symbolic bisimulation (Definition 4.11) can be assumed to be admissible, as we now demonstrate. These ψ_i are required to satisfy the property: $\mathcal{M} \vDash \psi_i$ if and only if $\mathcal{M} \vDash \varphi_i$ and $P' \simeq_{\mathcal{M}_{(\bullet)}} Q_i$, where φ_i decorates a transition and therefore is admissible. Admissible ψ_i that satisfy this requirement can be found as follows:

$$\psi_i = \bigvee \big\{ \vartheta \mid \exists M\colon\ P' \simeq_M Q_i \ \text{ and } \ \vartheta = \varphi_i \wedge \overline{Loc \backslash M} \wedge M \ \text{ and } \ \vartheta \text{ satisfiable} \big\}$$

For admissible formulae, we can calculate "finally" — using the auxiliary function "$\underline{\text{neg}}$" — as follows. If ϑ is unsatisfiable, then finally$(\vartheta) = \text{ff}$, likewise for formulae φ; otherwise:

$$\begin{aligned}
\text{finally}(\pi) &= \pi \\
\text{finally}(\vartheta \fatsemi \pi) &= \underline{\text{neg}}(\vartheta) \wedge \pi \\
\text{finally}((\vartheta \wedge \ell) \fatsemi^{1} (\overline{\ell} \wedge \pi)) &= \text{finally}(\vartheta) \wedge \overline{\ell} \wedge \pi \\
\text{finally}(\bigvee \vartheta_i \wedge \text{initially}(\pi_i)) &= \bigvee(\text{finally}(\vartheta_i) \wedge \text{neg}(\pi_i))
\end{aligned}$$

"$\underline{\text{neg}}$" is defined similarly, except that wherever π occurs on the right-hand side it should be replaced by $\text{neg}(\pi)$. (Recall from Section 4.2 that $\text{neg}(\pi)$ replaces all positive literals in π with $\mathtt{tt}$.)

The proof that this calculation captures (12) follows (in each direction) by induction on the structure of admissible formulae. For the final clause, one uses the fact that $\mathcal{L} \vDash \vartheta$ implies $L \cdot \mathcal{L} \vDash \vartheta$.

B Comparison with other equivalences

In this appendix we show that LF-equivalence differs from all of the location- and cause-based equivalences that we are aware of. The equivalences we discuss have been characterized in many ways. The location-based equivalences have been studied, for example, in [8, 2, 9, 27, 29, 11]; the cause-based equivalences have appeared, for example, in [12, 13]. Comparisons between these approaches appear in [22, 27, 14].

Most of these equivalences are defined for CCS, which does not have explicitly located processes. To apply these definitions to terms in our language, we first perform an implicit syntactic transformation that removes explicit location references from the terms.

Below, we list the equivalences we consider; the interested reader should refer to the original papers for further information.

- CCS interleaving equivalence [25] was defined in Section 2.3.
- *Causal* (C) equivalence [12] distinguishes processes based on the causality of actions.
- *Locations* (L) equivalence [8] distinguishes processes based on the local causality of actions.
- *Local/Global* (LG) equivalence [22] distinguishes processes based on a combination of their local and global causes; it is strictly finer than the intersection of the C- and L-equivalences.
- *Located Action* (LA) equivalence [29, 11] is a finer form of locations equivalence in which location names appear in the syntax of the language, as they do in our language.

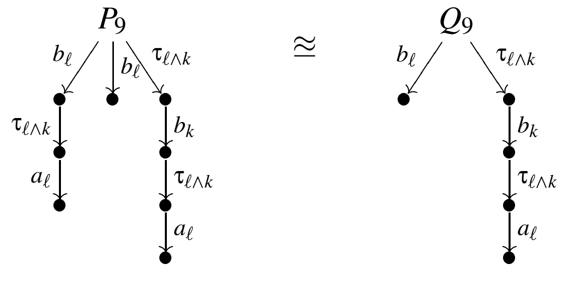
- *Located Failure* (LF) equivalence is the relation studied in this paper.
- *Located Failure/Located Action* (LF/LA) is defined to be the intersection of the LF- and LA-equivalences. A more explicit characterization of this equivalence is easy to derive.

The relationships between these equivalences are summarized in the following diagram, where an arrow $A \to B$ indicates that A is coarser than B. If there is no arrow between two nodes, this indicates that the equivalences are unrelated.

$$\begin{tikzcd}
& & & & LF/LA & \\
& GL & & LA \arrow[ur] & & LF \arrow[ul] \\
C \arrow[ur] & & L \arrow[ul] \arrow[ur] & & & \\
& CCS \arrow[ul] \arrow[ur] & & & &
\end{tikzcd}$$

The closest of these equivalences to ours is LA equivalence because it is the only one of these relations defined on a language with explicitly located processes. The following example, due to Flavio Corradini, shows that two LF-equivalent processes need not be LA-equivalent.

$$\begin{aligned}
P_9 &= \Big(\ell[\![b.\alpha.a + b + \beta.\alpha.a]\!] \mid k[\![\overline{\alpha} + \overline{\beta}.b.\overline{\alpha}]\!]\Big)\backslash\alpha\backslash\beta \\
Q_9 &= \Big(\ell[\![b + \beta.\alpha.a]\!] \mid k[\![\overline{\beta}.b.\overline{\alpha}]\!]\Big)\backslash\alpha\backslash\beta
\end{aligned}$$



To compare these using the LA-equivalence, it is sufficient to erase all of the subscripts from τ actions and then treat visible actions with different subscripts as distinct actions. From this view, the processes are not even trace equivalent because $P_9 \stackrel{b_\ell}{\Longrightarrow} \cdot \stackrel{a_\ell}{\longrightarrow}$ and Q_9 has no matching pair of transitions. These processes are also not L-equivalent.

The following processes P_{10} and Q_{10} are LF/LA-equivalent but not causally equivalent.

$$\begin{aligned}
P_{10} &= \big(\ell[\![a.b]\!] \mid k[\![c.d]\!]\big) \\
Q_{10} &= \big(\ell[\![a.(\alpha.b + b)]\!] \mid k[\![c.(\overline{\alpha}.d + d)]\!]\big)\backslash\alpha
\end{aligned}$$

The counterexample in the other direction is more obvious since LF-equivalence is sensitive to the location of unguarded τ-actions, but none of other equivalences are:

$$\begin{aligned} P_{11} &= \big(\ell[\![\overline{\alpha}.a]\!]\ |k[\![\alpha]\!]\big)\backslash\alpha \\ Q_{11} &= \ \ \ell[\![\tau.a]\!] \end{aligned}$$

These processes are equated by all of the location- and cause-based relations, but distinguished by LF-equivalence.

Finally we note that it is important that in the definition of LF equivalence the location which fails is observable. One could easily define an alternative equivalence in which it is observable that a site failed, but not which one. The resulting equivalence is strictly weaker than LF equivalence, as shown by the following processes. P_{12} and Q_{12} would be related by such an equivalence, whereas they are distinguished by LF equivalence.

$$\begin{aligned} P_{12} &= \big(m[\![\overline{\alpha}]\!]\ |\ \ell[\![\alpha.a]\!]\ \ |\ k[\![\alpha.a]\!]\big)\backslash\alpha \\ Q_{12} &= \big(m[\![\overline{\alpha}]\!]\ |\ \ell[\![\alpha.a]\!]\ \ |\ k[\![\mathsf{nil}]\!]\big)\backslash\alpha \end{aligned}$$

References

[1] Samson Abramsky. The lazy lambda calculus. In David Turner, editor, *Research Topics in Functional Programming*, UT Year of Programming Series, pages 65–117. Addison-Wesley, 1990.

[2] Luca Aceto. A static view of localities. *Formal Aspects of Computing*, pages 201–222, 1994.

[3] R. Amadio and S. Prasad. Localities and failures. In *Proc. 14th Foundations of Software Technology and Theoretical Computer Science*, volume 880 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[4] Roberto Amadio. An asynchronous model of locality, failure, and process mobility. In *COORDINATION '97*, volume 1282 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[5] Roberto Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, pages 291–324, 1998.

[6] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. 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] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of finite-state systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

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

[12] P. Darondeau and P. Degano. Causal trees. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi della Rocca, editors, *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 234–248, Stresa, Italy, July 1989. Springer-Verlag.

[13] P. Degano, R. De Nicola, and U. Montanari. Partial orderings descriptions and observations of nondeterministic concurrent processes. In J.W. de Bakker, W.-P de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 438–466. Springer-Verlag, June 1988. School/Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency (1988: Noordwijkerhout, Netherlands).

[14] P. Degano and C. Priami. Causal trees. In Kuich [23], pages 629–640.

[15] Zohar Manna *et al*. STeP: The stanford temporal prover. Technical report, Stanford University, Computer Science Laboratory, June 1994.

[16] C. Fournet and G. Gonthier. The refliexive CHAM and the join-calculus. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, Paris, January 1996. ACM Press.

[17] 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.

[18] Andrew D. Gordon. Bisimilarity as a theory of functional programming. In *Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Theoretical Computer Science* (`http://www.elsevier.nl/locate/entcs`). Elsevier, 1995.

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

[20] Matthew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents (Extended Abstract) In *3rd International Workshop on High-Level Concurrent Languages (HLCL'98)*, volume 16(3) of *Electronic Notes in Theoretical Computer Science* . Available from `http://www.elsevier.nl/locate/entcs`, 1998. Full version to appear in *Information and Computation*.

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

[22] A. Kiehn. Comparing locality and causality based equivalences. *Acta Informatica*, 31(8):697–718, 1994.

[23] W. Kuich, editor. *Proceedings of the International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, Wien, Austria, July 1992. Springer-Verlag.

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

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

[26] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.

[27] Ugo Montanari and Daniel Yankelovich. Partial order localities. In Kuich [23], pages 617–628.

[28] Ben Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986.

[29] David Murphy. Observing located concurrency. In *Proceedings of the Symposium on Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[30] Y. S. Ramakrishna, P. M. Melliar-Smith, L. E. Moser, L. K. Dillon, and G. Kutty. Interval logics and their decision procedures: Part i: An interval logic. *Theoretical Computer Science*, 166:1–47, 1996.

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

[32] Colin Stirling. Modal and temporal logics. In Samson Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of logic in computer science*, volume 2 Background: Computational Structures, pages 477–563. Oxford University Press, 1992.