Theoretical Computer Science 114 (1993) 31–61
Elsevier

Observing localities*

G. Boudol and I. Castellani

INRIA, Sophia-Antipolis, 2004 Route des Lucioles, 06561 Valbonne Cedex, France

M. Hennessy

School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton BN1 9QH, UK

A. Kiehn

Institut für Informatik, Technische Universität München, Arcisstrasse 21, W-8000 München 2, Germany

Abstract

Boudol, G., I. Castellani, M. Hennessy and A. Kiehn, Observing localities, Theoretical Computer Science 114 (1993) 31–61.

We introduce a refined version of observation for CCS which allows the observer to see the distributed nature of processes. Using several examples, we argue that a semantic theory based on such observations is not only intuitive but may also be of use when formalising the relationship between implementations and specifications. Technically, we show that the resulting theory of *location equivalence* is very similar to that of bisimulation equivalence, e.g. it can be characterised by a simple modal logic. A comparison with distributed bisimulations is also given.

1. Introduction

There are by now a number of well-established semantic theories of processes in the research literature which are based on principles of observation. The main idea is that processes are deemed to be equivalent if there is no possible observation which can distinguish them. Different formalisations of this idea, which give rise to a number of semantic equivalences, may be found in [13, 9, 10]. All these formalisations are based on the same simple notion of observation, namely communication: one may observe a process by communicating with it via a communication channel. The resulting semantic theories are often called *interleaving theories*; they do not distinguish between concurrency and nondeterminism or, more precisely, they equate a parallel

Correspondence to: G. Boudol, INRIA, Sophia-Antipolis, 2004 Route des Lucioles, 06561 Valbonne Cedex, France.

This work has been supported by the ESPRIT/BRA CEDISYS project.

process with the purely nondeterministic one obtained by interleaving its primitive computation steps or actions.

Some attempts have been made to generalise this observation-based approach in order to develop a semantic theory which does distinguish between these two phenomena [5, 2, 8, 3, 7]. Here we reexamine the approach taken in [5, 11], where the processes under observation are considered to be *distributed in nature*. So, the observer cannot only test the process by communicating with it but can also observe or distinguish that part of the distributed process which reacted to the test. A purely nondeterministic process is based at one site, whereas, in general, a concurrent one may be distributed among a number of different locations. It follows that an observer will be able to distinguish them.

In this extended introduction we will try to explain in detail our approach and to motivate it by indicating its usefulness.

We use as a starting point the process algebra CCS, a process description language which describes processes in terms of the *actions* they can perform. For example, a "cell", or a one-place bag B_1, which repeatedly performs the actions *in*, *out*, may be defined by

$$B_1 \Leftarrow in.out.B_1.$$

If we run two copies of this in parallel, we obtain a process which acts like a two-place bag:

$$B_2 \Leftarrow B_1 | B_1.$$

Here $|$ is the parallel operator of CCS which, in this context, defines a process which consists of two independent processes, two copies of B_1, running in parallel.

Processes running in parallel may also communicate or synchronise with each other. This is formalised by dividing the set of actions into two complementary subsets, the *input* actions and the *output* actions. Communication is then considered to be the simultaneous occurrence of complementary actions. Output actions are indicated by an overbar, such as $\bar{\alpha}$, $\overline{in}$, etc., input actions by the absence of an overbar and there is a distinguished action τ to indicate a communication or, more generally, internal and unobservable computation. So, if we define two processes *In*, *Out* by

$$In \Leftarrow in.\alpha.In,$$

$$Out \Leftarrow \bar{\alpha}.out.Out,$$

then the process $In|Out$ acts somewhat like B_2. However, *In*, *Out* are not obliged to synchronise via the action α. The actions α and $\bar{\alpha}$ may be performed independently, which corresponds to separate synchronisations with processes in their operating environments. To eliminate these possible communications with the environment and thereby force the synchronisation between the two processes, we limit the scope of

these actions using another operator of CCS, *restriction*, which is written as $\backslash\alpha$ or, more loosely, as $\backslash A$, where A is a set of actions. So, let NB_2 be defined by

$$NB_2 \Leftarrow (In \mid Out)\backslash\alpha.$$

These two processes, B_2 and NB_2, offer very similar behaviour to a user, particularly as the synchronisation between *In* and *Out* is not supposed to be visible externally. According to the theory developed in [13], they are *weak-bisimulation-equivalent*, denoted by $B_2 \approx NB_2$; in terms of the visible actions *in* and *out*, they offer the same possible behaviour to any user of the systems.

However, this reasoning is based on the assumption that the only property which can be observed of a process is its ability to perform particular actions. Now let us re-interpret the language by saying that $P|Q$ is a *distributed process*, where the subprocess P is at one site and Q is at another site; moreover, let us suppose that an observer can distinguish between sites in the sense that, when a distributed process performs an action, the observer knows the site responsible for it. Thus, one observer's view of the distributed process B_2 is as shown in Fig. 1. Here the observer has decided, out of personal choice, to call l_1 the site or location of the first subprocess, and l_2 the location of the second subprocess. Now it is not possible to construct a similar view of NB_2. For example, the distribution represented in Fig. 2 can easily be distinguished from B_2 as here all *in* actions are seen to occur at location l_1 and all *out* actions at location l_2. In contrast, they are distributed between l_1 and l_2 in the distributed process B_2.

The basic difference between these two processes is that in NB_2 one site is responsible for the *in* actions and one for the *out* actions, whereas B_2 has two equivalent sites, each acting like a one-place buffer. Viewing these as specifications, this is a useful and meaningful distinction. To implement B_2, it is necessary to have independent locations, each acting like buffers, whereas an implementation of NB_2

Fig. 1.

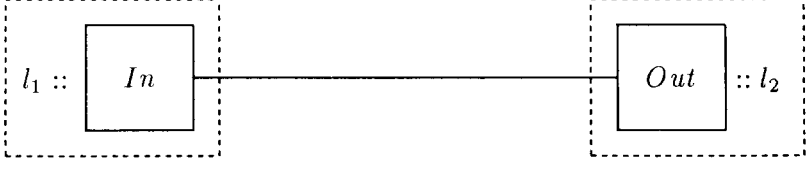

Fig. 2.

would always have to localise the responsibility for the *in* actions and the *out* actions in independent locations. In fact, NB_2 is a reasonable specification for a variety of communication protocols which have to transfer successfully messages across (possibly, faulty) mediums. For example, consider Pr_1, defined by

$$Sender \Leftarrow in.\overline{send}.Sender,$$

$$Medium \Leftarrow send.\overline{deliver}.Medium,$$

$$Receiver \Leftarrow deliver.out.Receiver,$$

$$Pr_1 \Leftarrow (Sender \mid Medium \mid Receiver)\setminus\{send, deliver\}.$$

Here the causal links between the *in* and *out* actions are more complicated but the responsibility for them is distributed in a manner similar to that in NB_2. So, as a distributed system, Pr_1 can be viewed by an observer in the same way as NB_2 above by naming the locations in the manner depicted in Fig. 3. Note that here the medium has not been assigned any location. This is reasonable because an observer will never see any actions which it performs and, therefore, he will never even know that it is there. Similar reasoning may be applied to more complicated protocols, where the transfers from the *Sender* to the *Receiver* follow a more tortuous route. As an example, consider a protocol which uses both a secure and a faulty wire. It tries first the faulty wire and, if it does not receive an acknowledgement, it retransmits on the secure one. The protocol Pr_2 can be defined by

$$Sender \Leftarrow in.\overline{send}.Sender,$$

$$Medium \Leftarrow send.\overline{try\,f}.(ack.Medium + \tau.\overline{try\,s}.Medium),$$

$$FWire \Leftarrow try\,f.(\overline{ack}.\overline{deliver}.FWire + FWire),$$

$$\mathrm{SWire} \Leftarrow try\,s.\overline{deliver}.SWire,$$

$$Receiver \Leftarrow deliver.out.Receiver,$$

$$Pr_2 \Leftarrow (Sender \mid Medium \mid FWire \mid SWire \mid Receiver)\setminus I,$$

$$\text{where } I = \{send, deliver, try\,f, try\,s, ack\}.$$

Here we use the choice operator $+$ to indicate that at certain times a process may act in either of two ways. Again this can be observed in a distributed fashion in

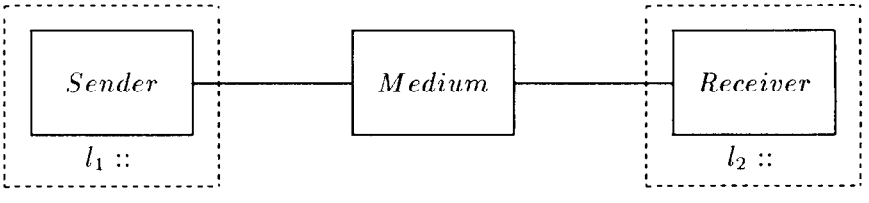

Fig. 3.

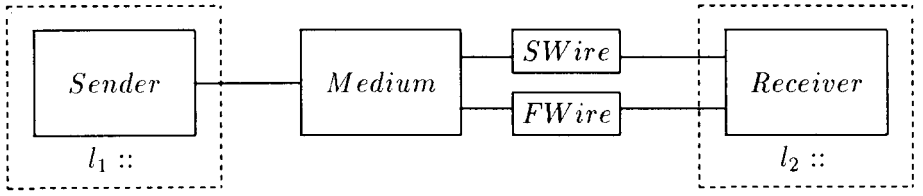

Fig. 4.

a manner similar to NB_2 above (see Fig. 4). So, both these implementations Pr_1 and Pr_2 match the *distributed specification* NB_2, while they are not equivalent to B_2.

As another example, consider the solution to a simple mutual-exclusion problem, where the access of two readers to a device is controlled by a semaphore. The system may be defined in CCS by

$$Reader \Leftarrow \bar{p}.enter.exit.\bar{v}.Reader,$$

$$Sem \Leftarrow p.v.Sem,$$

$$Sys \Leftarrow (Reader \mid Sem \mid Reader)\backslash\{p, v\}.$$

This system satisfies the specification defined by

$$Spec \Leftarrow enter.exit.Spec$$

in that $Spec \approx Sys$. However, it is also possible to have a faulty system implement this specification. This involves a faulty reader which may deadlock after exiting from the critical region:

$$Reader \Leftarrow \bar{p}.enter.exit.\bar{v}.Reader,$$

$$FReader \Leftarrow \bar{p}.enter.(exit.\bar{v}.FReader + exit.\bar{v}.nil),$$

$$Sem \Leftarrow p.v.Sem,$$

$$FSys \Leftarrow (Reader \mid Sem \mid FReader)\backslash\{p, v\}.$$

In our description of the faulty reader, *FReader*, we use *nil* to indicate a process which is deadlocked, but, in practice, the deadlock could arise because of some more complicated behaviour of a particular reader. One can check using the definition of *weak bisimulation equivalence* in [13] that $Spec \approx FSys$ although the system obviously has a deadlocked subsystem which has no counterpart in the specification.

However, if we view these systems as *distributed systems* then a difference can be perceived. An observer may view *Sys* as shown in Fig. 5, where there are two locations, in each of which there is a process repeatedly executing the actions *enter*, *exit*. There is no comparable view of the faulty system *FSys*. For example, the view shown in Fig. 6 leads to a different observable behaviour. Here it is possible to reach a state in which no more actions will ever be observed at location l_2, while this is not possible for the corresponding view of *Sys*.

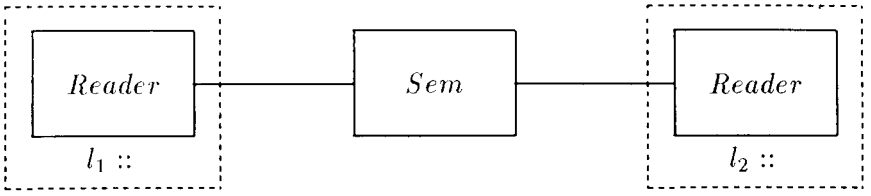

Fig. 5.

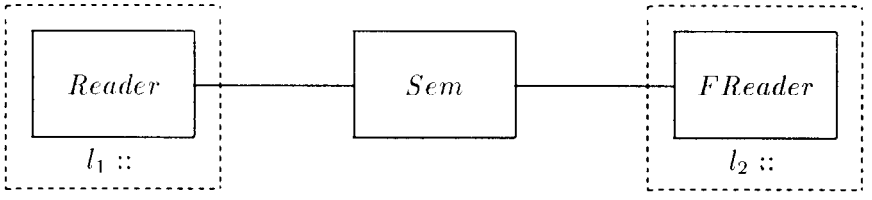

Fig. 6.

We hope that these examples show that, by allowing an observer to see the distributed nature of a process, a useful and intuitively reasonable semantic equivalence is obtained.

Let us now address the question of how exactly an observer should be allowed to perceive the distributed nature of a process. In this respect we are guided by principles of extensionality; we would like the resulting equivalence to be as extensional as possible, in the sense that the semantics of a process should be determined only by its external manifestations rather than by its internal structure or behaviour. It is reasonable to argue that at least some aspect of the distribution of subprocesses in a distributed system is a part of its extensional behaviour; therefore, if we are to view CCS as a language for describing distributed systems, an observer should be able to view $P|Q$ as a distributed system which potentially has two distinct sites; any externally visible action performed by P should be recognizable as emanating from one location and any performed by Q should be recognizable as coming from a different location.

But how is an observer to decide when more than one location is involved? The first point to note is that the notion of location has to be conceived of hierarchically because the distributed structure of a system may evolve dynamically. For example, any system of the form $a.P$ will initially be viewed as containing one location. This is true even when P has the form $Q|R$. The distributed nature of P will become apparent only when the action a has been performed. Thus, if an observer decides to call l_1 the unique location in $a.P$ when a is performed, he will then perceive that P is distributed among two locations and may allocate them the names l_2 and l_3. At this stage l_2 and l_3 are actually *sublocations* of the original location l_1 and in our formalisation they will correspond to the two locations $l_1.l_2$ and $l_1.l_3$. This notion of sublocation will serve to

distinguish the different kinds of distribution which appear in systems such as $a.(P|Q)$ and $a.R|S$.

One might hope that this allocation of location names by the observer could, at least to some extent, be static; at each stage during the observation period, the observer could allocate location names within the system under observation on the basis of the operator | and then proceed by examining the ability of the resulting system to perform specific actions at specific locations. However, this approach leads to some difficulties. For example, it would distinguish the two processes $a.nil$ and $((a+\bar{\alpha})|\alpha.a)\backslash\alpha$, which is difficult to justify intuitively. One would also have difficulty in equating terms such as P and $P|nil$ or even ensuring that | is associative.

In this paper we abandon this static approach and instead develop the idea that location names are assigned dynamically as part and parcel of the act of observation; when an observer sees an action being performed, this is seen as emanating from a particular location and the observer may then choose to allocate a name to that location. All subsequent actions performed at that location will then be recognised as emanating from this same location. Technically, this involves developing an operational semantics for the language by replacing the usual statements of the form $P \xrightarrow{a} Q$ with new ones of the form

$$P \xrightarrow[u]{a} Q,$$

which carry information about the location names which the observer has assigned to particular locations in the system. This is very similar to the approach taken in [5, 4, 11], where *distributed bisimulations* are defined, and later in the paper we will offer a detailed comparison.

We now briefly outline the remainder of the paper. In the next section we give a formal definition of the new equivalence, which we call (*weak*) *location equivalence* and develop some of its properties. This section also includes some examples. This is followed by a section containing technical results on the new equivalence. We then define a *modal logic* for location equivalence which characterises it in the same way as the modal logic HML characterises bisimulation equivalence. A detailed comparison with *distributed bisimulation* is then given, and we end with a list of further research problems which is suggested by the introduction of the idea of locations.

2. Location equivalence

In this section we introduce location equivalence and show some of its properties. As discussed in the introduction, we take a dynamic approach, that is, we are interested only in the observable distribution of a process. The site of a process is observable if and only if a visible action is performed at it. Seeing such an action emanating from a site, the observer allocates a location name to it. This name is then perceived with any further observation of an action at that site.

The language we use to formalize this approach is a slight extension of Milner's pure CCS. The extension is an additional operator, called *location prefixing*, representing the allocation of locations to processes. A process p prefixed by a location u will be denoted by $u::p$. Intuitively, process p is at a location called u. However, in general, we will assume these locations to be introduced via the observation of visible actions. That is, initially, before any experiment has been performed, the process under investigation does not contain any location. The observers we assume here are more powerful than those usually considered for CCS or other process algebras. With the observation of an action, the location of the action is also perceived and assigned a name. So, we will have a transition rule

$$a.p \xrightarrow[u]{a} u::p \quad \text{for any location name } u,$$

which means that a has been performed at a location to which the observer has permanently assigned the name u. Intuitively, a process of the form $u::p$ arises from the execution of some action at a location u, and p is the subprocess following this action. However, as usual in CCS, we will assume τ transitions as invisible and, therefore, no location will be observed when they are performed. Hence, $a.\tau.p$ also would evolve to $u::p$, i.e.

$$a.\tau.p \overset{a}{\underset{u}{\Rightarrow}} u::p.$$

If further experiments are performed on $u::p$ then the location u will always be observed. Moreover, the location called u may contain sublocations, which in turn may also be observed. For example,

$$u::(a.nil \mid b.nil) \xrightarrow[uv]{a} u::(v::nil \mid b.nil).$$

Here the location which has been called u by a previous observation contains two sublocations and at one of them a is performed. The name v is allocated to this subsite via the observation of a. The complete observation of a records both the general location u and the sublocation v. In general, we will have transitions of the form $\xrightarrow[u_1 \ldots u_n]{a}$, where each u_i, $i \geqslant 1$, is the name of a primitive site. The sequence $u_1 \ldots u_n$ identifies the location where the action a is actually executed, i.e. we identify a general location with its access path. This allows us to formalise the notion of a sublocation: the location $u_1 \ldots u_n$ may be considered to be a sublocation of $u_1 \ldots u_j$ for any j, $1 \leqslant j \leqslant n$. Before we go into a more detailed discussion of such transitions and of location equivalence, we will introduce formally the language we consider and also establish some straightforward properties.

We assume the reader to be familiar with Milner's pure CCS (see [13]). We have a set of actions Λ, ranged over by $\alpha, \beta, \ldots$ and a set of co-actions $\{\bar{\alpha} \mid \alpha \in \Lambda\}$, a disjoint copy of Λ, where the overbar represents a bijection such that $\bar{\bar{\alpha}} = \alpha$ for all $\alpha \in \Lambda$. The

invisible action is denoted by τ. We have $Act = A \cup \bar{A}$ ranged over by $a, b, c, \ldots$ and $Act_\tau = Act \cup \{\tau\}$ ranged over by $\mu, \nu, \ldots$ *Var* denotes a set of process variables ranged over by $x, y, z, \ldots$ The operators we consider are the standard CCS ones, *nil*, action prefixing $\mu.$, nondeterministic choice $+$, parallel composition $|$, relabelling $[f]$, restriction $\backslash\alpha$ and recursion *rec x*. Moreover, to express algebraic properties we will also use the auxiliary operators leftmerge $\lfloor$ and communication merge $|_c$ (cf. [1, 9]). Additionally, we introduce the new operator of location prefixing discussed above. To this end, we assume an infinite set of basic location names or site names *Loc* disjoint from Act_τ. These will be ranged over by $k, l, m, \ldots$, whereas general locations, sequences from *Loc**, will be ranged over by $u, v, w, \ldots$ As usual, we use ε to denote the empty word in *Loc**. Syntactically, we write $u :: p$ with the intuitive meaning that process p is at a general location called u. So, we work with the following abstract syntax

$$
\begin{aligned}
t ::= \; & nil \mid \mu.t \mid t+t \mid (t|t) \mid t[f] \mid t\backslash\alpha \\
& \mid x \mid rec\, x.t \\
& \mid t \lfloor t \mid t|_c t \\
& \mid u :: t,
\end{aligned}
$$

where f is a relabelling function $f: Act_\tau \to Act_\tau$ such that $f(\bar{a}) = \overline{f(a)}$ for all $a \in Act$ and $f(\tau) = \tau$. We assume the usual precedence rules for operators, where $\lfloor$ and $|_c$ have the same precedence as $|$ and $u :: p$ the same as $\mu.p$. As usual, we will often omit occurrences of *nil*, for example, rendering $a.nil$ as a. Let $\mathscr{E}$ be the set of all terms which can be generated by this syntax. For $t \in \mathscr{E}$, the set of free variables $\mathrm{fv}(t)$ is defined in the standard way. Closed terms are called processes and we use $\mathbb{P}$ to denote the set of such terms. We, furthermore, distinguish the set CCS containing all location-free processes and the sets CCS_f and CCS_{rf}, the former consisting of the recursion-free CCS processes and the latter those which are additionally restriction- and relabelling-free. Typical examples for processes in $\mathbb{P}$ are $l :: a.nil \mid b.nil$, $l :: (k :: a.nil)$ and $(l :: \alpha.nil \mid k :: \bar{\alpha}.nil)\backslash\alpha$. In examples of processes, we will often write $x \Leftarrow t$ instead of *rec x. t*. For example, the process NB_2 of the introduction is equivalent to $(rec\, x.in.\alpha.x \mid rec\, x.\bar{\alpha}.out.x)\backslash\alpha$. Another notational convention we use is $p\backslash A$ for $p\backslash\alpha_1\backslash\alpha_2\backslash\cdots\backslash\alpha_n$, where $A = \{\alpha_1, \ldots, \alpha_n\}$. Also, we work modulo the identifications $u_1 :: u_2 :: \cdots :: u_n :: p = u :: p$, where $u = u_1 u_2 \ldots u_n$, and $\varepsilon :: p = p$. We shall see that these identifications are valid in our semantics. Finally, we use $loc(p) \subseteq Loc$ to denote the set of basic location names occurring in p.

We will give two operational semantics to $\mathscr{E}$. The first one generalizes bisimulation equivalence in a straightforward way. The standard transition system is extended by a rule for location prefixing. Loosely speaking, the new transition rule preserves locations but also ignores them. The extended transition system is given in Fig. 7. Note that the rule for the communication merge $|_c$ cannot be defined without using

For each $\mu \in Act_\tau$, let $\xrightarrow{\mu} \subseteq (\mathbb{P} \times \mathbb{P})$ and $\stackrel{\varepsilon}{\Rightarrow} \subseteq (\mathbb{P} \times \mathbb{P})$ be the least binary relations satisfying the following axiom and rules:

(S1)	$\mu.p \xrightarrow{\mu} p$,		
(S2)	$p \xrightarrow{\mu} p'$	implies	$u :: p \xrightarrow{\mu} u :: p'$
(S3)	$p \xrightarrow{\mu} p'$	implies	$p+q \xrightarrow{\mu} p'$ $q+p \xrightarrow{\mu} p'$
(S4)	$p \xrightarrow{\mu} p'$	implies	$p \mid q \xrightarrow{\mu} p' \mid q$ $q \mid p \xrightarrow{\mu} q \mid p'$
(S5)	$p \xrightarrow{\mu} p'$	implies	$p \lceil q \xrightarrow{\mu} p' \mid q$
(S6)	$p \xrightarrow{\mu} p'$	implies	$p[f] \xrightarrow{f(\mu)} p'[f]$
(S7)	$p \xrightarrow{\mu} p'$ & $\mu \notin \{\alpha, \bar{\alpha}\}$	implies	$p \setminus \alpha \xrightarrow{\mu} p' \setminus \alpha$
(S8)	$t[rec\, x.t/x] \xrightarrow{\mu} p'$	implies	$rec\, x.t \xrightarrow{\mu} p'$
(S9)	$p \xrightarrow{a} p'$, $q \xrightarrow{\bar{a}} q'$	implies	$p \mid q \xrightarrow{\tau} p' \mid q'$
(S10)	$p \stackrel{\varepsilon}{\Rightarrow} p'$, $p' \xrightarrow{a} p''$ $q \stackrel{\varepsilon}{\Rightarrow} q'$, $q' \xrightarrow{\bar{a}} q''$	implies	$p \mid_c q \xrightarrow{\tau} p'' \mid q''$
(W1)	$p \stackrel{\varepsilon}{\Rightarrow} p$,		
(W2)	$p \xrightarrow{\tau} p'$, $p' \stackrel{\varepsilon}{\Rightarrow} p''$	implies	$p \stackrel{\varepsilon}{\Rightarrow} p''$

Fig. 7. Standard transitions.

ε-transitions.[1] Based on this transition system, weak bisimulation is defined as usual. We use $\stackrel{\mu}{\Rightarrow}$ to denote $\stackrel{\varepsilon}{\Rightarrow} \xrightarrow{\mu} \stackrel{\varepsilon}{\Rightarrow}$ and $\hat{\mu}$ to denote a if $\mu = a$, $a \in Act$, or $\hat{\mu} = \varepsilon$ if $\mu = \tau$.

Definition 2.1 (*Bisimulation equivalence*). A symmetric relation $R \subseteq \mathbb{P} \times \mathbb{P}$ is called a *bisimulation* iff $R \subseteq B(R)$, where

$(p, q) \in B(R)$ iff for all $\mu \in Act_\tau$,

$p \stackrel{\mu}{\Rightarrow} p'$ implies $q \stackrel{\hat{\mu}}{\Rightarrow} q'$ for some $q' \in \mathbb{P}$ such that $(p', q') \in R$.

p and q are *bisimulation-equivalent* (or *bisimilar*), $p \approx q$, if and only if there is a bisimulation R such that $(p, q) \in R$.

Two expressions $t, t' \in \mathscr{E}$ are *bisimulation-equivalent* (or *bisimilar*), $t \approx t'$, iff $t\rho \approx t'\rho$ for all substitutions $\rho : \mathrm{fv}(t) \cup \mathrm{fv}(t') \to \mathbb{P}$.

It is well known that $\approx$ is an equivalence relation and, restricted to $\mathbb{P}$, the largest symmetric fixed point of the equation $R = B(R)$.

[1] As pointed out by L. Aceto, an operational rule for $|_c$ based entirely on strong transitions would yield an operator which does not preserve $\approx^+$, the coarsest relation contained in $\approx$ preserved by $+$.

Bisimulation equivalence considers the ability of performing visible actions and only in this respect bisimilar processes exhibit the same behaviour. The new semantics we give to $\mathscr{E}$ additionally takes the distribution in space into account. As already discussed above, there will be two points in which it differs from the standard one. The first point is that locations may be introduced via the observation of actions. Secondly, processes may contain locations, and actions in the scope of locations will be observed at those locations. Formally, these two points are reflected by the following two rules in the location transition system.

(L1) $a.p \xrightarrow[u]{a} u::p, \quad u \in Loc^*,$

(L2) $p \xrightarrow[u]{a} p'$ implies $v::p \xrightarrow[vu]{a} v::p'.$

Here u is an access path representing a location where the action a is performed and in the second rule this is extended by v to give the new location vu. With these two rules we are able to derive, up to the identification $u::v::p = uv::p$,

$$\begin{aligned} a.b.c.nil &\xrightarrow[u]{a} u::b.c.nil \\ &\xrightarrow[uv]{a} uv::c.nil \\ &\xrightarrow[uvw]{a} uvw::nil. \end{aligned}$$

This example demonstrates the incremental allocation of locations in the course of observations. But one might wonder whether this is necessary when considering sequential processes only. However, with the rule (L1) it is not obligatory to assign a new location with each action performed because u may be instantiated to ε. So, we could also have the derivation

$$\begin{aligned} u::b.c.nil &\xrightarrow[u]{b} u::c.nil \\ &\xrightarrow[u]{c} u::nil. \end{aligned}$$

The rule for parallel composition is the usual one:

(L4) $p \xrightarrow[u]{a} p'$ implies $p|q \xrightarrow[u]{a} p'|q.$

Using this rule together with (L1), we can derive

$$\begin{aligned} a.(b.nil\,|\,c.nil) &\xrightarrow[u]{a} u::(b.nil\,|\,c.nil) \\ &\xrightarrow[uv]{b} u::(v::nil\,|\,c.nil) \\ &\xrightarrow[uw]{c} u::(v::nil\,|\,w::nil). \end{aligned}$$

We can now see how parallelism is differentiated from nondeterminism. For the process $a.nil \mid b.nil$, we can derive $a.nil \mid b.nil \xrightarrow[u]{a} u::nil \mid b.nil$, while its nondeterministic counterpart would perform the transition $a.b.nil + b.a.nil \xrightarrow[u]{a} u::b.nil$. Now with the observation of the action b different locations would be perceived. In $u::nil \mid b.nil \xrightarrow[v]{b} u::nil \mid v::nil$ the b is performed at the location v which is independent of u, whereas in $u::b.nil \xrightarrow[uv]{b} uv::nil$ it is performed at a sublocation of u, namely uv.

The τ-transitions are considered, as usual, to be invisible; so, no location is observed when they are performed. They are of the form $p \xrightarrow{\tau} p'$ and are defined through the standard transition system for CCS given in Fig. 7. Visible transitions are defined by the location transition system given in Fig. 8. They always have the form $p \xrightarrow[u]{a} p'$, where we call u the location where the action a is performed. Weak transitions are defined in the same way as for the previous transition system:

$$p \overset{\varepsilon}{\Rightarrow} p_1,\ p_1 \xrightarrow[u]{a} p_2,\ p_2 \overset{\varepsilon}{\Rightarrow} p' \text{ implies } p \overset{a}{\underset{u}{\Rightarrow}} p'.$$

Note that $p \overset{a}{\underset{\varepsilon}{\Rightarrow}} p'$ implies $p \overset{a}{\Rightarrow} p'$, up to the identification $\varepsilon::q = q$, and that, in general, the reverse is not true, due to rule (S2).

If we apply the transition rules to our examples from the introduction, we can derive

$$In \xrightarrow[u]{in} u::\alpha.In.$$

Formally, In is represented by the term $rec\,x.in.\alpha.x$ and

$$rec\,x.in.\alpha.x \xrightarrow[u]{in} u::\alpha.rec\,x.in.\alpha.x$$

For each $a \in Act$, let $\xrightarrow[u]{a} \subseteq (\mathbb{P} \times Loc^* \times \mathbb{P})$ be the least binary relation satisfying the following axioms and rules:

(L1)	$a.p \xrightarrow[u]{a} u::p$	$u \in Loc^*$	
(L2)	$p \xrightarrow[u]{a} p'$	implies	$v::p \xrightarrow[vu]{a} v::p'$
(L3)	$p \xrightarrow[u]{a} p'$	implies	$p+q \xrightarrow[u]{a} p'$, $q+p \xrightarrow[u]{a} p'$
(L4)	$p \xrightarrow[u]{a} p'$	implies	$p \mid q \xrightarrow[u]{a} p' \mid q$, $q \mid p \xrightarrow[u]{a} q \mid p'$
(L5)	$p \xrightarrow[u]{a} p'$	implies	$p[f] \xrightarrow[u]{a} p'[f]$
(L6)	$p \xrightarrow[u]{a} p'$	implies	$p[f] \xrightarrow[u]{f(a)} p'[f]$
(L7)	$p \xrightarrow[u]{a} p'$ & $a \notin \{\alpha, \bar{\alpha}\}$	implies	$p \setminus \alpha \xrightarrow[u]{a} p' \setminus \alpha$
(L8)	$t[rec\,x.t/x] \xrightarrow[u]{a} p'$	implies	$rec\,x.t \xrightarrow[u]{a} p'$

Fig. 8. Location transitions.

because

$$in.\alpha.x[rec\,x.in.\alpha.x/x] \xrightarrow[u]{in} u::\alpha.rec\,x.in.\alpha.x$$

by (L1) and (L8). Similarly, recalling that $In \Leftarrow in.\alpha.In$ and $Out \Leftarrow \bar{\alpha}.out.Out$ and using in addition (L4) and (L7) and our notational conventions

$$\begin{aligned}(In \mid Out)\backslash\alpha &\xrightarrow[u]{in} (u::\alpha.In \mid Out)\backslash\alpha \\ &\xrightarrow{\tau} (u::In \mid out.Out)\backslash\alpha \\ &\xrightarrow[v]{out} (u::In \mid v::Out)\backslash\alpha \\ &\overset{in}{\Longrightarrow}_{u} (u::In \mid v::out.Out)\backslash\alpha.\end{aligned}$$

This means that the observer can discern two different locations in the system, one where *in* is performed and the other where *out* is performed.

Based on the transition system given in Figs. 7 and 8, we now define location equivalence. Two processes p and q are location-equivalent if every move of one of them is matched by a similar move of the other and, in particular, if, for every visible transition $p \overset{a}{\underset{u}{\Rightarrow}} p'$, the matching transition $q \overset{a}{\underset{u}{\Rightarrow}} q'$ has the same location.

Definition 2.2 (*Location equivalence*). A symmetric relation $R \subseteq \mathbb{P} \times \mathbb{P}$ is called a *location bisimulation* iff $R \subseteq C(R)$, where $(p,q) \in C(R)$ iff

(i) $p \overset{\varepsilon}{\Rightarrow} p'$ implies $q \overset{\varepsilon}{\Rightarrow} q'$ for some $q' \in \mathbb{P}$ such that $(p',q') \in R$,

(ii) $p \overset{a}{\underset{u}{\Rightarrow}} p'$, $a \in Act$, $u \in Loc^*$ implies $q \overset{a}{\underset{u}{\Rightarrow}} q'$ for some $q' \in \mathbb{P}$ such that $(p',q') \in R$.

Two processes p and q are said to be *location-equivalent*, $p \approx_\ell q$, iff there is a location bisimulation R such that $(p,q) \in R$.

Two expressions $t, t' \in \mathscr{E}$ are *location-equivalent*, $t \approx_\ell t'$, iff, for all substitutions $\rho : fv(t) \cup fv(t') \to \mathbb{P}$, $t\rho \approx_\ell t'\rho$.

We reconsider the examples of the introduction. As argued there, the processes

$$B_2 \Leftarrow B_1 \mid B_1$$

and

$$NB_2 \Leftarrow (In \mid Out)\backslash\alpha$$

should be distinguished if their distributed nature is taken into account. Indeed, B_2 and NB_2 are not location-equivalent. Consider the move

$$B_2 \xrightarrow[u]{in} u::out.B_1 \mid B_1 \;=\; B_2'.$$

Then NB_2 would have to match it with

$$NB_2 \xrightarrow[u]{in} (u::\alpha.In \mid Out)\backslash\alpha = NB_2^1$$

or

$$NB_2 \overset{in}{\underset{u}{\Rightarrow}} (u::In \mid out.Out)\backslash\alpha = NB_2^2.$$

Both of these, NB_2^1 and NB_2^2, may perform an *out* action at any location v,

$$NB_2^i \overset{out}{\underset{v}{\Rightarrow}} NB_2^{i'},$$

which cannot be matched by a similar move by B_2' at the location v if v is chosen to be different from u. On the other hand, $NB_2 \approx_l Pr_1$ because a location bisimulation containing this pair of processes may be defined, using NB and Pr as generic names for states of the systems NB_2 and Pr_1, as follows, using $I = \{send, deliver\}$:

$(NB, Pr) \in R$ iff

$$NB = (u::In \mid v::Out)\backslash\alpha \text{ and } Pr = (u::Sender \mid Medium \mid v::Receiver)\backslash I \quad \text{or}$$

$$NB = (u::\alpha.In \mid v::Out)\backslash\alpha \quad \text{or}$$

$$NB = (u::In \mid v::out.Out)\backslash\alpha$$

and

$$Pr = (u::\overline{send}.Sender \mid Medium \mid v::Receiver)\backslash I \quad \text{or}$$

$$Pr = (u::Sender \mid \overline{deliver}.Medium \mid v::Receiver)\backslash I \quad \text{or}$$

$$Pr = u::Sender \mid Medium \mid v::out.Receiver)\backslash I.$$

Finally, it can be checked that $Pr_1 \approx_l Pr_2 \approx_l NB_2$ considering the location bisimulation in Fig. 9, where $I = \{send, deliver, try f, try s\}$.

3. Properties of location equivalence

In this section we prove some properties of $\approx_l$. For example, a property one would expect is that $p \approx_l q$ implies $p \approx q$, that is, location equivalence is included in bisimulation equivalence. In order to show a more general result, we introduce some notation concerning the renaming and erasing of locations.

Let π be a mapping $\pi: Loc \to Loc^*$. We call such a mapping a *location renaming*. Now π may be extended to words in the obvious way: $\pi(\varepsilon) = \varepsilon$ and $\pi(lu) = \pi(l)\pi(u)$. Further, π may be transferred homomorphically to a mapping between processes $\pi: \mathscr{E} \to \mathscr{E}$: for example, we will have $\pi(u::p) = \pi(u)::\pi(p)$. For a renaming affecting only

R is defined by $(NB, Pr) \in R$ iff

$$NB = (u :: In \mid v :: Out) \setminus \alpha$$

and

$$Pr = (u :: Sender \mid Medium \mid FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Medium \mid \overline{ack}.\overline{deliver}.FWire + FWire \mid SWire \mid v :: Receiver) \setminus I$$

or

$$NB = (u :: \alpha.In \mid v :: Out) \setminus \alpha \quad \text{or}$$

$$NB = (u :: In \mid v :: out.Out) \setminus \alpha$$

and

$$Pr = (u :: \overline{send}.Sender \mid Medium \mid FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Wirehandler \mid FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Wirehandler \mid FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid ack.Medium + \tau.try\,s.Medium$$
$$\mid \overline{ack}.\overline{deliver}.FWire + FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Medium \mid \overline{deliver}.FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Medium \mid FWire \mid SWire \mid v :: out.Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid try\,s.Medium$$
$$\mid \overline{ack}.\overline{deliver}.FWire + FWire \mid SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Medium$$
$$\mid \overline{ack}.\overline{deliver}.FWire + FWire \mid \overline{deliver}.SWire \mid v :: Receiver) \setminus I \quad \text{or}$$

$$Pr = (u :: Sender \mid Medium$$
$$\mid \overline{ack}.\overline{deliver}.FWire + FWire \mid SWire \mid v :: out.Receiver) \setminus I$$

Fig. 9. A location bisimulation for Pr_1 and Pr_2.

one location, we will frequently use the notation $t[l \to u]$, meaning the result of applying to t the renaming π defined by

$$\pi(k) = \begin{cases} u & \text{if } k = l, \\ k & \text{otherwise.} \end{cases}$$

Let *pure* be the location renaming which erases all locations, i.e. $\forall l \in Loc$, $pure(l) = \varepsilon$; then $pure(p)$ yields a CCS-process for any $p \in \mathbb{P}$ (up to the identification $\varepsilon :: q = q$). Now, it is easy to check that, for any $\mu \in Act_\tau$, we have $p \xrightarrow{\mu} p'$ if and only if $pure(p) \xrightarrow{\mu} pure(p')$. Then the following property is straightforward.

Proposition 3.1. $pure(p) \approx p$.

In fact, we have a stronger result, namely that $pure(p)$ and p are *strong bisimulation-equivalent* [13] but we shall only use Proposition 3.1 in the following.

We now show that location equivalence implies ordinary bisimulation equivalence. This result relies on the next lemma, which shows the basic interrelation between the transition systems $\xrightarrow{a}$ and $\xrightarrow[u]{a}$.

Lemma 3.2. (1) *If* $p \xrightarrow[u]{a} p'$ *then* $pure(p) \xrightarrow{a} pure(p')$.

(2) *If* $pure(p) \xrightarrow{a} r$ *then* $\exists u \in Loc^*$, $p' \in \mathbb{P}$ *s.t.* $p \xrightarrow[u]{a} p'$ *and* $r = pure(p')$.

(3) *Properties* (1) *and* (2) *also hold for weak transitions.*

Proof. By induction on the proof of transitions. □

Proposition 3.3. $p \approx_l q$ *implies* $pure(p) \approx pure(q)$.

Proof. We show that $R = \{(pure(p), pure(q)) \mid p \approx_l q\}$ is a bisimulation. Suppose $pure(p) \overset{a}{\Rightarrow} r$. By Lemma 3.2, we have $p \overset{a}{\underset{u}{\Rightarrow}} p'$ with $r = pure(p')$. As $p \approx_l q$, we know that $q \overset{a}{\underset{u}{\Rightarrow}} q'$ for some q' such that $p' \approx_l q'$. Applying Lemma 3.2, we obtain $pure(q) \overset{a}{\Rightarrow} pure(q')$, which is a suitable matching move since $(pure(p'), pure(q')) \in R$.

The case $pure(p) \overset{\varepsilon}{\Rightarrow} r$ is even easier, since we have $pure(p) \overset{\varepsilon}{\Rightarrow} r$ iff $\exists p'$ s.t. $p \overset{\varepsilon}{\Rightarrow} p'$ and $r = pure(p')$. □

The inclusion of location equivalence into bisimulation equivalence follows directly from Propositions 3.1 and 3.3.

Corollary 3.4. $p \approx_l q$ *implies* $p \approx q$.

We now study how location renamings affect the behaviour of processes. The next lemma establishes the relation between the transitions of processes p and $\pi(p)$.

Lemma 3.5. *Let* $p \in \mathbb{P}$, *and* π *be an arbitrary location renaming.*

(1) *If* $p \xrightarrow{\tau} p'$ *then* $\pi(p) \xrightarrow{\tau} \pi(p')$.

(2) *If* $\pi(p) \xrightarrow{\tau} r$ *then* $\exists p'$ *s.t.* $p \xrightarrow{\tau} p'$ *and* $\pi(p') = r$.

(3) *If* $p \xrightarrow[u]{a} p'$ *then* $\pi(p) \xrightarrow[\pi(u)]{a} \pi(p')$.

(4) *For any* L *s.t.* $loc(p) \subseteq L \subset Loc$: $\pi(p) \xrightarrow[u]{a} r$ *implies* $\exists \pi'$, $\exists v \in Loc^*$, $\exists s \in \mathbb{P}$ *such that* $\pi' \lceil L = \pi \lceil L$ *and* $p \xrightarrow[v]{a} s$, *with* $\pi'(v) = u$, $\pi'(s) = r$.

(5) *Properties* (1)–(4) *also hold for weak transitions.*

Proof. We prove only point (4), as the others are easy. The proof is by induction on the proof of the transition $\pi(p) \xrightarrow[u]{a} r$. We examine two cases, the other ones being immediate.

(i) $a.q \xrightarrow[u]{a} u::q$, with $\pi(p)=a.q$, $r=u::q$. Then $\exists p'$ s.t. $p=a.p'$ and $\pi(p')=q$. Let now $l \in Loc-L$. Define the renaming π' by

$$\pi'(k)=\begin{cases} u & \text{if } k=l, \\ \pi(k) & \text{otherwise.} \end{cases}$$

Then $\pi' \lceil L=\pi \lceil L$, and the required move of p is $a.p' \xrightarrow[l]{a} l::p'$, since $\pi'(l)=u$ and $\pi'(p')=\pi(p')=q$ because $loc(p')=loc(p) \subseteq L$.

(ii) $w::q \xrightarrow[wu']{a} w::q'$, $\pi(p)=w::q$, $u=wu'$ and $r=w::q'$, and the transition of $w::q$ is inferred from $q \xrightarrow[u']{a} q'$. Since $w::q=\pi(p)$, there exist w', p' s.t. $p=w'::p'$, with $\pi(w')=w$, $\pi(p')=q$. Then, by induction, $\exists \pi'$, $\exists v'$, $\exists s'$ s.t. $\pi' \lceil L=\pi \lceil L$ and $p' \xrightarrow[v']{a} s'$, with $\pi'(v')=u'$, $\pi'(s')=q'$. From this we deduce $w'::p' \xrightarrow[w'v']{a} w'::s'$, which is the required move for p since $\pi'(w')=\pi(w')=w$ because $loc(p) \subseteq L$ and, thus, $\pi'(w'v')=wu'=u$ and $\pi'(w'::s')=w::q'=r$. □

Proposition 3.6. *Let $p, q \in \mathbb{P}$, and π be an arbitrary location renaming. Then $p \approx_\ell q$ implies $\pi(p) \approx_\ell \pi(q)$.*

Proof. Let $R=\{(\pi(p), \pi(q)) \mid p \approx_\ell q\}$. We show that R is a location bisimulation. Assume $\pi(p) \overset{a}{\underset{u}{\Rightarrow}} r$. Take now L such that $loc(p) \cup loc(q) \subseteq L \subset Loc$. Applying Lemma 3.5(4), we deduce that there exists π' s.t. $\pi' \lceil L=\pi \lceil L$ and a transition $p \overset{a}{\underset{v}{\Rightarrow}} p'$ such that $\pi'(v)=u$, $\pi'(p')=r$. As $p \approx_\ell q$, there is a transition $q \overset{a}{\underset{v}{\Rightarrow}} q'$ such that $p' \approx_\ell q'$. Now since $\pi(q)=\pi'(q)$, applying Lemma 3.5(3), we obtain $\pi'(q) \overset{a}{\underset{\pi'(v)}{\Longrightarrow}} \pi'(q')$, where, by definition of R, $(\pi'(p'), \pi'(q')) \in R$.

The case $\pi(p) \overset{\varepsilon}{\Rightarrow} r$ is treated similarly, using clauses (1) and (2) of Lemma 3.5. □

Like (weak) bisimulation equivalence, location equivalence is a congruence with respect to the operators of prefixing, parallel composition, renaming, restriction and recursion. It is also preserved by the new operator of location prefixing. We do not give the proof here, the proof technique being essentially the same as for bisimulation equivalence [13].

Proposition 3.7. *Let $p, q, r \in \mathbb{P}$, $t, t' \in \mathscr{E}$, and suppose $p \approx_\ell q$, $t \approx_\ell t'$. Then*

(1) $\mu.p \approx_\ell \mu.q$,

(2) $u::p \approx_\ell u::q$,

(3) $p|r \approx_\ell q|r$,

(4) $p|_c r \approx_\ell q|_c r$,

(5) $p[f] \approx_\ell q[f]$,

(6) $p \setminus \alpha \approx_\ell q \setminus \alpha$,

(7) $rec\, x.t \approx_\ell rec\, x.t'$. □

It is well known that the (weak) bisimulation equivalence $\approx$ is not preserved by nondeterministic choice nor by the leftmerge operator (for the latter consider the following example: $a \approx \tau.a$ but $a \lfloor b \not\approx \tau.a \lfloor b$), and for similar reasons neither is location equivalence. We proceed here as for bisimulation equivalence, and work with $\approx_\ell^c$ – the coarsest equivalence contained in $\approx_\ell$ which is preserved by all operators. The equivalence $\approx_\ell^c$ can be characterized in two ways. Again, these are standard and we omit the proofs.

Proposition 3.8. *$p \approx_\ell^c q$ iff*

(i) *$p \stackrel{\tau}{\Rightarrow} p'$ implies $q \stackrel{\tau}{\Rightarrow} q'$ for some $q' \in \mathbb{P}$ such that $p' \approx_\ell q'$,*
(ii) *$p \xrightarrow[u]{a} p'$ implies $q \stackrel{a}{\underset{u}{\Rightarrow}} q'$ for some $q' \in \mathbb{P}$ such that $p' \approx_\ell q'$,*
(iii) *$q \stackrel{\tau}{\Rightarrow} q'$ implies $p \stackrel{\tau}{\Rightarrow} p'$ for some $p' \in \mathbb{P}$ such that $p' \approx_\ell q'$,*
(iv) *$q \xrightarrow[u]{a} q'$ implies $p \stackrel{a}{\underset{u}{\Rightarrow}} p'$ for some $p' \in \mathbb{P}$ such that $p' \approx_\ell q'$,*

iff

$$p + a \approx_\ell q + a \quad \text{for some } a \notin sort(p) \cup sort(q),$$

where, as usual, $sort(p)$ yields the set of visible actions of p.

Consider now the equations in Figs. 10–12. The first set of equations D contains the so-called static laws of CCS, that is, the basic laws governing the static operators, parallel composition, relabelling and restriction. The equations E and E' give a set of very simple laws about the distribution of location names through the other operators and will be used later in the paper. Finally, the equation set G is a complete set of equations for distributed bisimulation equivalence on CCS_{rf}, finite CCS without restriction and relabelling. This particular set of equations is taken from [11] and will also be referred to later in the paper. The last equation in Fig. 12 expresses an

(P1)	$x \mid nil = x$
(P2)	$x \mid y = y \mid x$
(P3)	$x \mid (y \mid z) = (x \mid y) \mid z$
(R1)	$x \backslash \alpha = x$ if $\alpha, \bar{\alpha} \notin sort(x)$
(R2)	$x \backslash \alpha \backslash \beta = x \backslash \beta \backslash \alpha$
(R3)	$(x \mid y) \backslash \alpha = (x \backslash \alpha \mid y \backslash \alpha)$ if $\alpha, \bar{\alpha} \notin sort(x) \cap sort(y)$
(R4)	$a.x \backslash \beta = \begin{cases} a.(x \backslash \beta) & \text{if } a \neq \beta, \bar{\beta} \\ nil & \text{otherwise} \end{cases}$
(RU1)	$(x \backslash \alpha)[f] = x[f'] \backslash \beta$ if $\beta \notin f(sort(x))$ and $f'(c) = \begin{cases} f(c) & \text{if } c \neq \alpha \\ \beta & \text{otherwise} \end{cases}$
(U1)	$x[id] = x$
(U2)	$x[f][g] = x[g \circ f]$
(U3)	$(x \mid y)[f] = x[f] \mid y[f]$

Fig. 10. Equations D, the static laws.

(E1)	$u::(x\|y)=u::x\,\|\,u::y$
(E2)	$u::(x[f])=(u::x)[f]$
(E3)	$u::(x\setminus\alpha)=(u::x)\setminus\alpha$
(E4)	$\varepsilon::x=x$
(E5)	$u::nil=nil$
(E6)	$u::(v::y)=uv::y$
(E'1)	$u::(x\|_c y)=u::x\,\|_c\,u::y$
(E'2)	$u::(x\lceil y)=u::x\lceil u::y$
(E'3)	$u::(x+y)=u::x+u::y$

Fig. 11. Equations $E\cup E'$, equations for location names.

(A1)	$x+(y+z)=(x+y)+z$
(A2)	$x+y=y+x$
(A3)	$x+nil=x$
(A4)	$x+x=x$
(LP1)	$(x+y)\lceil z=x\lceil z+y\lceil z$
(LP2)	$(x\lceil y)\lceil z=x\lceil(y\lceil z)$
(LP3)	$x\lceil nil=x$
(LP4)	$nil\lceil x=nil$
(I1)	$x+\tau x=\tau.x$
(I2)	$\mu.\tau x=\mu.x$
(I3)	$\mu.(x+\tau.y)+\mu.y=\mu.(x+\tau.y)$
(NI1)	$\tau.x\lceil y=\tau.(x\lceil y)$
(NI2)	$x\lceil y=x\lceil\tau.y$
(NI3)	$x\lceil(y+\tau.z)+x\lceil z=x\lceil(y+\tau.z)$
(CPE)	$x\|y=x\lceil y+y\lceil x+x\|_c y$
(CP0)	$\tau.x\|_c y=x\|_c y$
(CP1)	$(x+y)\|_c z=(x\|_c z)+(y\|_c z)$
(CP2)	$x\|_c y=y\|_c x$
(CP3)	$x\|_c nil=nil$
(CP4)	$(a.x\lceil x')\|_c(b.y\lceil y')=\begin{cases}\tau.(x\|y)\lceil(x'\|y') & \text{if } a=\bar{b}\\ nil & \text{otherwise}\end{cases}$
(CP5)	$a.(x\|x')\lceil(y\|y')\sqsubset a.(c.x\lceil x'+v)\lceil(\bar{c}y\lceil y'+w)$

Fig. 12. Equations G.

absorption of terms: the notation $y\sqsubset x$ means that y is absorbed by x, that is, $x+y=x$.

Now it may be checked that all these equations are satisfied by $\approx_\ell^c$:

Proposition 3.9. *The sets of equations D (in Fig. 10), E, E' (in Fig. 11) and G (Fig. 12) are sound for $\approx_\ell^c$.*

We show now that for *sequential* CCS processes the equivalence $\approx_\ell$ reduces to the bisimulation equivalence $\approx$. This will ensure us that introducing locations adds discriminations between processes only as far as their distributed aspect is concerned.

Let CCS_{seq} be the set of sequential processes of CCS, that is, processes built without the parallel operator. The following is easy to prove, using the laws (E2), (E3) and (E6).

Lemma 3.10. *Let* $p \in CCS_{seq}$ *and* $u, v \in Loc^*$. *Then*

(1) *if* $v :: p \stackrel{a}{\Rightarrow}_u r$ *then* $\exists p' \in CCS_{seq}$ *s.t.* $p \stackrel{a}{\Rightarrow} p'$, *with* $r \approx_l u :: p'$,

(2) *if* $p \stackrel{a}{\Rightarrow} p'$ *then* $\forall w \in Loc^*$. $\exists r \in CCS_{seq}$ *s.t.* $v :: p \stackrel{a}{\Rightarrow}_{vw} r$, *with* $r \approx_l vw :: p'$.

Proposition 3.11. *If* $p, q \in CCS_{seq}$ *then* $p \approx q$ *implies* $p \approx_l q$.

Proof. Let $R = \{(v :: p, v :: q) \mid v \in Loc^*, p, q \in CCS_{seq} \text{ and } p \approx q\}$. We want to show that $R \subseteq \approx_l$. To this end, it is sufficient to show that R is a location bisimulation up to location equivalence. More precisely, we will show that if $v :: p \stackrel{a}{\Rightarrow}_u r$ then $v :: q \stackrel{a}{\Rightarrow}_u s$, with $r \approx_l \cdot R \cdot \approx_l s$ (and, similarly, for ε-moves).

Suppose $v :: p \stackrel{a}{\Rightarrow}_u r$. Then by Lemma 3.10(1), $\exists p' \in CCS_{seq}$ s.t. $p \stackrel{a}{\Rightarrow} p'$, with $r \approx_l u :: p'$. Note that u is necessarily of the form $u = vw$. Now since $p \approx q$, corresponding to $p \stackrel{a}{\Rightarrow} p'$ there is a move $q \stackrel{a}{\Rightarrow} q'$ s.t. $p' \approx q'$. But now, by Lemma 3.10(2), $\exists s \in CCS_{seq}$ s.t. $v :: q \stackrel{a}{\Rightarrow}_{vw} s$, with $s \approx_l vw :: q' = u :: q'$. Then $(u :: p', u :: q') \in R$ and, thus, $r \approx_l u :: p' \, R \, u :: q' \approx_l s$.

Take now $v :: p \stackrel{\varepsilon}{\Rightarrow} v :: p'$. This is because $p \stackrel{\varepsilon}{\Rightarrow} p'$. Then, since $p \approx q$, we have $q \stackrel{\varepsilon}{\Rightarrow} q'$ with $p' \approx q'$ and, thus, $(v :: p', v :: q') \in R$.

We have, thus, proved that $p \approx q$ implies $v :: p \approx_l v :: q$ and, hence, in particular, $\varepsilon :: p \approx_l \varepsilon :: q$. Now, by the soundness of law (E4), we have $\varepsilon :: p \approx_l p$; whence, we conclude that $p \approx_l \varepsilon :: p \approx_l \varepsilon :: q \approx_l q$. □

We next show a decomposition result for location equivalence. This will be used in Section 5 to show that location equivalence and distributed bisimulation equivalence coincide on a subset of CCS. This decomposition result concerns only $\mathbb{P}_f$, the set of finite processes of $\mathbb{P}$. We introduce first some notation and preliminary results. Let $p \in \mathbb{P}_f$. Define the *observable length* of p, denoted as $|p|$, to be the maximal length of a chain of observable actions of p, that is,

$$\forall p \in \mathbb{P}_f: \quad |p| = max\{n \mid p \stackrel{a_1}{\Rightarrow} \cdots \stackrel{a_n}{\Rightarrow} p'\}.$$

The following properties are easy to show.

Lemma 3.12. *Let* $p, q \in \mathbb{P}_f$. *Then*

(1) $p \approx q$ *implies* $|p| = |q|$,

(2) $p \stackrel{\varepsilon}{\Rightarrow} p'$ *implies* $|p| \geq |p'|$,

(3) $p \stackrel{a}{\Rightarrow} p'$ *implies* $|p| > |p'|$,

(4) $p \stackrel{a}{\Rightarrow}_u p'$ *implies* $|p| > |p'|$.

Note that from clause (1) we may deduce also $|u :: p| = |p|$. Also, it may be noted that, as a consequence of Proposition 3.1, we have $\pi(p) \approx p$ for any location renaming π, since $p \approx pure(p) = pure(\pi(p)) \approx \pi(p)$. Then, from clause (1), it follows also that $|p| = |\pi(p)|$.

The next lemma generalizes Lemma 3.2. It also expresses the fact that in the location transition system the location allocated at each step may be chosen arbitrarily.

Lemma 3.13. *Let* $p, q \in \mathbb{P}$. *Then*

(1) *if* $p \xrightarrow{a} p'$ *then* $\exists u \in loc(p)^*$ *s.t.* $\forall l \notin loc(p)$: $p \xrightarrow[ul]{a} p''$, *with* $p' = p''[l \to \varepsilon]$,

(2) *if* $p \xrightarrow{a} p'$ *then* $\exists v \in loc(p)^*$, $w \in Loc^*$ *s.t.* $u = vw$ *and* $\forall l \notin loc(p)$: $p \xrightarrow[vl]{a} p''$, *with* $p' = p''[l \to w]$.

We need another preliminary result.

Lemma 3.14. *Let* $p, q, r, s \in \mathbb{P}_f$ *and* $u \in Loc^*$, $l \in Loc$, *with* $l \notin loc(p) \cup loc(q) \cup loc(r) \cup loc(s)$. *Then*

$$ul :: p|q \approx_\ell ul :: r|s \text{ implies } |p| = |r| \text{ and } |q| = |s|.$$

Proof. Note first that $|(p|q)| = |p| + |q|$. Let now $q \overset{a_1}{\underset{v_1}{\Rightarrow}} \cdots \overset{a_n}{\underset{v_n}{\Rightarrow}} q'$, where $n = |q|$. By Lemma 3.13(2), we may assume that l does not occur in any v_i. Now $ul :: p|q \overset{a_1}{\underset{v_1}{\Rightarrow}} \cdots \overset{a_n}{\underset{v_n}{\Rightarrow}} ul :: p|q'$. Therefore, $\exists r', s'$ s.t. $ul :: r|s \overset{a_1}{\underset{v_1}{\Rightarrow}} \cdots \overset{a_n}{\underset{v_n}{\Rightarrow}} ul :: r'|s'$. Now r cannot be responsible for any a_i-move, since l does not occur in any v_i and, thus, $|s| \geqslant |q|$. A symmetric argument shows $|q| \geqslant |s|$; hence, $|q| = |s|$ and, by the remark above, also $|p| = |r|$. □

We may now prove the decomposition result.

Proposition 3.15 (Decomposition). *Let* $p, q, r, s \in \mathbb{P}_f$ *and* $u \in Loc^*$, $l \in Loc$, *with* $l \notin loc(p) \cup loc(q) \cup loc(r) \cup loc(s)$. *Then*

$$ul :: p|q \approx_\ell ul :: r|s \text{ implies } p \approx_\ell r \text{ and } q \approx_\ell s.$$

Proof. Let $R = \{(p, r), (q, s) \mid ul :: p|q \approx_\ell ul :: r|s\}$. We want to show that R is a location bisimulation. We consider only the pair (p, r).

Let $p \overset{\varepsilon}{\Rightarrow} p'$. We want to show that $r \overset{\varepsilon}{\Rightarrow} r'$ with $(p', r') \in R$. From $p \overset{\varepsilon}{\Rightarrow} p'$, we deduce $ul :: p|q \overset{\varepsilon}{\Rightarrow} ul :: p'|q$. Then $ul :: r|s$ has a corresponding move $ul :: r|s \overset{\varepsilon}{\Rightarrow} ul :: r'|s'$ such that $ul :: p'|q \approx_\ell ul :: r'|s'$. Then we have $(p', r') \in R$ and $(q, s') \in R$.

We want to show now that $r \overset{\varepsilon}{\Rightarrow} r'$ and $s \overset{\varepsilon}{\Rightarrow} s'$. We prove then, by contradiction, that in $ul :: r|s \overset{\varepsilon}{\Rightarrow} ul :: r'|s'$ there was no communication between r and s. For, assume that there was such a communication, that is, $ul :: r|s \overset{\varepsilon}{\Rightarrow} ul :: r''|s'' \overset{\varepsilon}{\Rightarrow} ul :: r'|s'$ because $ul :: r \overset{\bar{a}}{\Rightarrow} ul :: r''$ and $s \overset{a}{\Rightarrow} s''$ for some $a \in Act$. Due to this communication, the observable length of s has decreased, i.e. $|s| > |s'|$. However, Lemma 3.14 applied to the precondition and to $ul :: p'|q \approx_\ell ul :: r'|s'$ yields $|q| = |s|$ and $|q| = |s'|$, respectively; thus, $|s| = |s'|$, contradicting $|s| > |s'|$.

Therefore, $ul :: r|s \overset{\varepsilon}{\Rightarrow} ul :: r'|s'$ because $r \overset{\varepsilon}{\Rightarrow} r'$ and $s \overset{\varepsilon}{\Rightarrow} s'$ and, thus, $r \overset{\varepsilon}{\Rightarrow} r'$ is the required move of r.

The case $p \stackrel{a}{\Rightarrow}_t p'$ is similar. It relies on the case $p \stackrel{\varepsilon}{\Rightarrow} p'$ above, and uses Lemmas 3.12, 3.14 and 3.13(2). □

4. Logical characterization

It is well known that bisimulation equivalence may be characterized using a simple modal language called *HML*, in the sense that two processes are bisimulation-equivalent if and only if they satisfy exactly the same set of formulae [13]. This characterization has generated considerable further research into the relationship between processes and behavioural properties, which, in turn, has given rise to significant practical applications [6, 14, 12]. Here we show that a similar logical characterization of location equivalence may be given and in future research we hope to use this characterization to extend the work on model checking, proof systems for modal properties, etc., in these papers to this new setting.

HML is a simple modal logic based on two modalities $\langle a \rangle$ and $[a]$, where a is an arbitrary action. So, an obvious extension to cope with location equivalence is to parameterise these modalities by locations. However, we will introduce a slightly more general modal language, which we feel is somewhat more natural. If a process contains no locations then it should be unnecessary for the formulae which characterise it to contain locations. In such processes, elements of CCS, locations are potential rather than actual and it should also be the case in its characterising formulae. For this reason we introduce location variables and quantification over these variables. One can imagine an expressive term language for locations but here we consider only a very simple language given by

$$t ::= l, l \in Loc \mid x, x \in LVar \mid \varepsilon \mid t.t.$$

Here *LVar* is an infinite set of location variables, disjoint from *Loc*, ε represents the degenerate location and . is sequence concatenation. As usual, we omit this symbol, writing $t.t'$ as tt'. The language for property formulae is then defined by the following abstract syntax:

$$\begin{aligned} \Phi ::= & \bigwedge \{\Phi \mid \Phi \in I\} \mid \neg \Phi \\ & \langle a \rangle_t \Phi \mid \langle \varepsilon \rangle \Phi \\ & t = t' \mid \exists x. \Phi. \end{aligned}$$

In the formula $\langle a \rangle_t \Phi$ the term t represents a location and intuitively the formula is satisfied by a process which can perform an action a at a location specified by t and in doing so reaches a state which satisfies Φ.

Other logical operators may be defined in the standard way in this language. For example, $\forall x. \Phi$ stands for $\neg \exists x. \neg \Phi$, $[a]_t \Phi$ stands for $\neg \langle a \rangle_t \neg \Phi$ and tt stands for $\bigwedge \{\Phi \mid \Phi \in \emptyset\}$ which is vacuously true for all processes. As with processes, we use $loc(\Phi)$ to denote the set of all locations occurring in Φ. Both $\forall$ and $\exists$ bind location variables

and this leads to the usual definition of free and bound variables. We are interested only in formulae which are closed, i.e. which contain no free occurrences of location variables, which we denote by $\mathscr{L}$. More generally for $L \subseteq Loc$, we let $\mathscr{L}_L$ denote the set of closed formulae which use locations only from L. So, in particular, formulae in $\mathscr{L}_\emptyset$ use no locations and all location variables are bound.

The satisfaction relation between processes and formulae, $\models \subseteq \mathbb{P} \times \mathscr{L}$, is a straightforward extension of the standard one [13], and is defined by structural induction on formulae. We extend the notation for renaming locations already defined on processes to formulae. We use $\Phi[x \to t]$ to denote the formula which results from substituting t for all free occurrences of x in Φ. If l denotes a simple location, i.e. $l \in Loc$, then $\Phi[l \to u]$ is the formula which results from substituting u for all occurrences of l in Φ and $\Phi[l \to x]$ is defined similarly:

$p \models \bigwedge \Phi_i$ if, for each $i \in I$, $p \models \Phi_i$,

$p \models \neg \Phi$ if not $p \models \Phi$,

$p \models \langle a \rangle_u \Phi$ if, for some p', $p \overset{a}{\underset{u}{\Rightarrow}} p'$ and $p' \models \Phi$,

$p \models \langle \varepsilon \rangle \Phi$ if, for some p', $p \overset{\varepsilon}{\Rightarrow} p'$ and $p' \models \Phi$,

$p \models u = w$ if u is w,

$p \models \exists x.\Phi$ if, for some $u \in Loc^*$, $p \models \Phi[x \to u]$.

Note that the variables range over general locations, i.e. sequences from Loc^* rather than simply elements of Loc. We consider some examples:

$$a|b \models \forall x.\forall y.\langle a \rangle_x \langle b \rangle_y tt.$$

This expresses the fact that the process $a|b$ can perform the actions a and b at arbitrary and, therefore, independent locations. This is not true for the process $a.b + b.a$, which, consequently, does not satisfy this formula. Instead, it satisfies a formula which expresses the idea that, whenever an action a is followed by an action b, the latter is always executed at a sublocation of the former:

$$a.b + b.a \models \forall x.\forall y.[a]_x[b]_y tt \to \exists z.(y = x.z).$$

Let us introduce the notation $t \; subloc \; t'$ to abbreviate the formula $\exists z.(t = t'.z)$, where z is some variable not occurring in t or t'. It expresses the fact that t is a sublocation of t'. Then the above formula may be rewritten as

$$\forall x \forall y [a]_x [b]_y tt \to y \; subloc \; x.$$

As another example, let p, q denote the processes $(a.\alpha.c|b.\bar{\alpha}.d)\backslash\alpha$, $(a.\alpha.d|b.\bar{\alpha}.c)\backslash\alpha$, respectively. Then

$$p \models \forall x.\forall y.\forall z.[a]_x[b]_y[c]_z tt \to z \; subloc \; x,$$

while q satisfies the dual property

$$q \models \exists x.\exists y.\exists z.\langle a\rangle_x\langle b\rangle_y\langle c\rangle_z tt \wedge \neg(z \; subloc \; x).$$

We can also use these properties to distinguish between the two different kinds of buffers, NB_2 and B_2, discussed in the introduction:

$$B_2 \models \forall x.\forall y.[in]_x[out]_y tt \rightarrow y \; subloc \; x.$$

This expresses the fact that whenever an *out* action follows an *in* action it must take place at the same location. This property is not true of NB_2. In fact, we have

$$NB_2 \models \forall x.\forall y.\langle in\rangle_x\langle out\rangle_y tt,$$

which emphasises the fact that the locations where the *in* and *out* actions are performed are independent.

Our aim is to show that location equivalence is characterised by the formulae in $\mathscr{L}$ which processes satisfy. We first show a lemma which says that certain locations may be renamed without affecting the satisfaction relation.

Lemma 4.1. *For $l, k \in Loc$, $p \models \Phi$ implies $p[k\rightarrow l] \models \Phi[k\rightarrow l]$ provided l does not occur in p or Φ.*

Proof (*by induction on the definition of $p \models \Phi$*). We give three examples.

(i) *Φ is $\langle a\rangle_u \Psi$.* Then $p \stackrel{a}{\Rightarrow}_u p'$ such that $p' \models \Psi$. By induction, $p'[k\rightarrow l] \models \Psi[k\rightarrow l]$. Also, from Lemma 3.5, it is easy to show that $p \stackrel{a}{\Rightarrow}_u p'$ implies $p[k\rightarrow l] \stackrel{a}{\Rightarrow}_{u'} p'[k\rightarrow l]$, where u' is $u[k\rightarrow l]$. So, $p[k\rightarrow l] \models \langle a\rangle_{u'} \Psi[k\rightarrow l]$, i.e. $p[k\rightarrow l] \models \Phi[k\rightarrow l]$.

(ii) *Φ is $\exists x.\Psi$.* We must show $p[k\rightarrow l] \models \exists x.\Psi[k\rightarrow l]$, i.e. for some u, $p[k\rightarrow l] \models (\Psi[k\rightarrow l])[x\rightarrow u]$. We know that $p \models \Psi[x\rightarrow w]$ for some w. If l does not occur in $\Psi[x\rightarrow w]$ then, by induction, $p[k\rightarrow l] \models (\Psi[x\rightarrow w])[k\rightarrow l]$. The required u is, therefore, $w[k\rightarrow l]$ since

$$(\Psi[x\rightarrow w])[k\rightarrow l] = (\Psi[k\rightarrow l])[x\rightarrow w[k\rightarrow l]].$$

Otherwise, i.e. if l occurs in w, let l' be a location name which does not occur in p nor $\Psi[x\rightarrow w]$. Then, by induction, $p[l\rightarrow l'] \models (\Psi[x\rightarrow w])[l\rightarrow l']$, that is, $p \models \Psi[x\rightarrow w']$, where $w'\rightarrow w[l\rightarrow l']$, and we are back to the previous case.

(iii) *Φ is $\neg\Psi$.* We must show that $p[k\rightarrow l] \models \neg\Psi[k\rightarrow l]$, i.e. not $p[k\rightarrow l] \models \Psi[k\rightarrow l]$. Suppose to the contrary that $p[k\rightarrow l] \models \Psi[k\rightarrow l]$. Then, by induction, since k does not appear in $p[k\rightarrow l]$ or $\Psi[k\rightarrow l]$, $p[k\rightarrow l][l\rightarrow k] \models \Psi[k\rightarrow l][l\rightarrow k]$. Since l does not appear in p or Ψ, this reduces to $p \models \Psi$, which contradicts $p \models \neg\Psi$. □

As a corollary, we have that if $p \models \Phi$ and Φ contains a location not in $loc(p)$ then the role of that location is essentially arbitrary.

Corollary 4.2. *If $p \models \Phi[x\rightarrow l]$, where $l \notin loc(p) \cup loc(\Phi)$ then, for every $k \notin loc(p) \cup loc(\Phi)$, $p \models \Phi[x\rightarrow k]$.*

Proof. By the previous lemma, $p[l \to k] \models \Phi[x \to l][l \to k]$. But $p[l \to k]$ is p and $(\Phi[x \to l])[l \to k]$ is $\Phi[x \to k]$ since neither k nor l occur in p or Φ. □

We are now ready to state the main result of this section. For $L \subseteq Loc$, let $\mathscr{L}_L(p)$ denote $\{\Phi \in \mathscr{L}_L \mid p \models \Phi\}$.

Theorem 4.3. $p \approx_l q$ iff $\mathscr{L}_L(p) = \mathscr{L}_L(q)$, *where* $L = loc(p) \cup loc(q)$.

Proof. $\Rightarrow$: It is sufficient to show that if $p \approx_l q$ and Φ in $\mathscr{L}_L(p)$ then $q \models \Phi$. The proof is straightforward by induction on the structure of Φ and is omitted.

$\Leftarrow$: Let $R = \{(p, q) \mid \mathscr{L}_L(p) = \mathscr{L}_L(q), L = loc(p) \cup loc(q)\}$. We show that R is a location bisimulation. The proof is by contradiction. If R is not a location bisimulation then there can only be two reasons (up to symmetry):

(i) $p \stackrel{\varepsilon}{\Rightarrow} p'$ and, for every q', $q \stackrel{\varepsilon}{\Rightarrow} q'$, $(p', q') \notin R$ or

(ii) for some $u \in Loc^*$ and $a \in Act$, $p \stackrel{a}{\underset{u}{\Rightarrow}} p'$ and, for every q' such that $q \stackrel{a}{\underset{u}{\Rightarrow}} q'$, $(p', q') \notin R$.

We examine only the second possibility as the first is similar but easier.

Let $\{q_i \mid i \in I\}$ denote $\{q' \mid q \stackrel{a}{\underset{u}{\Rightarrow}} q'\}$. Then, for each $i \in I$, there exists a closed formula Φ_i which uses at most the locations from q_i and p' such that $p' \models \Phi_i$ and $q_i \not\models \Phi_i$. So, $p \models \Phi$ and $q \not\models \Phi$, where Φ denotes $\langle a \rangle_u \bigwedge \{\Phi_i \mid i \in I\}$. Note, however, that in general $loc(\Phi) \not\subseteq loc(p) \cup loc(q)$. We need to find a formula which is satisfied by p and not by q but which contains locations only from $loc(p) \cup loc(q)$. We know that u must be of the form vw, where the basic locations in v belong to $loc(p)$ and, therefore, the only new basic locations occurring in Φ which are not in $loc(p) \cup loc(q)$ must appear in w. Let l be one such location. We show that there exists a formula Φ' such that $loc(\Phi') = loc(\Phi) - \{l\}$ and $p \models \Phi'$, $q \not\models \Phi'$. By iterating this elimination procedure, we will eventually obtain a formula which differentiates between p and q and only contains the basic locations only from $loc(p) \cup loc(q)$. Let $new(x)$ be a formula which expresses the fact that x is a basic location which does not occur in $loc(p) \cup loc(q)$. If $l_1, \ldots, l_n$ is an enumeration of this set then $new(x)$ may be defined by

$$\neg(x = l_1) \wedge \neg(x = l_2) \wedge \cdots \wedge \neg(x = l_n) \wedge \neg(x = \varepsilon)$$
$$\wedge ((x = y.z) \to y = \varepsilon \vee z = \varepsilon).$$

Provided we choose a fresh variable x, it then follows that $p \models \exists x.(\Phi[l \to x] \wedge new(x))$ while $q \not\models \exists x.(\Phi[l \to x] \wedge new(x))$. The latter follows from the previous corollary and these two statements contradict the fact that $(p, q) \in R$. To see in detail why q does not satisfy this formula, suppose to the contrary that it does, i.e. $q \models \exists x.(\Phi[l \to x] \wedge new(x))$. Then $q \models \Phi[l \to x][x \to u] \wedge new(u)$ for some location u. This means that u is, in fact, a basic location k from Loc which is not in $loc(q) \cup loc(\Phi[l \to x])$ and $q \models \Phi[l \to x][x \to k]$. So, we may apply the previous corollary to obtain $q \models \Phi[l \to x][x \to l]$, i.e. $q \models \Phi$. □

As an immediate corollary we have the following result.

Corollary 4.4. *For* $p, q \in$ CCS, $p \approx_l q$ *if and only if* $\mathscr{L}_\emptyset(p) = \mathscr{L}_\emptyset(q)$.

5. The relationship to distributed bisimulation

The first approach to a semantics for CCS which takes the distributed nature of processes into account – as opposed to their causal structure – can be found in [5, 4], with an extension in [11]. The basic idea is similar to that of location equivalence. The capabilities of observers are increased so that they can observe actions together with the location where they are performed. However, observers have a different strategy. In the present paper an observer sees an action together with its location, chooses a name and assigns it to the observed site. Within the world of distributed bisimulations, observers are mobile and may move from one location to another. So, when seeing an action together with its location, an observer moves to this location and appoints a new observer to observe the remainder of the process. Thus, in the course of experimenting on a process, the number of observers increases by one with each visible action. On the other hand, locations are observed without assigning names to them.

In this section we give a formal comparison of distributed bisimulations and location equivalence for the finite language without renaming and restriction. We will show that for these processes both the semantics coincide. This allows the transfer of results already available for distributed bisimulations to location equivalence. For example, we have a complete axiomatization.

In what follows, we give a brief discussion of distributed bisimulations for finite processes without restriction and renaming. The definition we choose here is based on *local and concurrent* observers as first introduced in [4] as opposed to the *local and global* approach in [5]. However, we consider an operational semantics slightly different from that in [4]; this can be found in [11].

We will distinguish the local subprocess by including it in brackets [ ], as, for example, $(a.nil \mid [b.nil]) \mid c.\bar{b}.nil$. Therefore, the syntax for "processes with a local component" is given by the following grammar, where p stands for any CCS_{rf}-term, that is, for any finite CCS term written without restriction and renaming:

$$P ::= [p] \mid (P|p) \mid (p|P)$$

Let $PROC_{[\ ]}$ denote this language. We use $\mathscr{C}[p], \mathscr{D}[q], \ldots$ to range over $PROC_{[\ ]}$. In $\mathscr{C}[p]$, p denotes the local subprocess included in [ ]-brackets, while $\mathscr{C}$ stands for its context; p will often be called the process at the current location. For example, for $(a.nil \mid [b.nil]) \mid c.\bar{b}.nil$, we have $p = b.nil$ and $\mathscr{C} = (a.nil|.) \mid c.\bar{b}.nil$. We will also use the notation $\mathscr{C}(p)$, where $\mathscr{C}$ and p are as in $\mathscr{C}[p]$ but p is embedded in $\mathscr{C}$ without the [ ]-brackets. Processes in $PROC_{[\ ]}$ will be used to exhibit the location which has been observed with the execution of a visible action. They determine the new position the observer will take and the remaining part of the process to be observed by a new observer. So, visible transitions have the form $p \xrightarrow{a}_{d} \mathscr{C}[p']$. The observer who observed the action a of p will move to the location of a, and from then on he can see only actions performed by the *local* component, that is, p'. A newly appointed observer will observe the *remaining* part $\mathscr{C}(nil)$.

For each $a \in Act$, let $\xrightarrow{a}_{\mathrm{d}} \subseteq (\mathrm{CCS}_{\mathrm{rf}} \times PROC_{[\ ]})$ be the least binary relation satisfying the following axiom and rules:

(D1) $a.p \xrightarrow{a}_{\mathrm{d}} [p]$,

(D2) $p \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p']$ implies $p+q \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p']$, $q+p \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p']$

(D3) $p \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p']$ implies $p|q \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p']|q$, $q|p \xrightarrow{a}_{\mathrm{d}} q|\mathscr{C}[p']$

(D4) $p \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p']$ implies $p \lceil q \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p'] \rceil q$.

Fig. 13. Distributed transitions.

In this semantics one observes locations of actions as in the location transition system, but no names are assigned to locations. The rules for (strong) distributed transitions are given in Fig. 13. These rules apply only to restriction- and renaming-free terms.

Weak transitions are derived by the rule

(WD) $p \overset{\varepsilon}{\Rightarrow} p_1 \xrightarrow{a}_{\mathrm{d}} \mathscr{C}[p'] \overset{\varepsilon}{\Rightarrow} \mathscr{D}[p'']$ implies $p \xrightarrow{a}_{\mathrm{d}} \mathscr{D}[p'']$,

where transitions $\mathscr{C}[p'] \overset{\varepsilon}{\Rightarrow} \mathscr{D}[p'']$ are defined, in conjunction with the single arrow relations $\xrightarrow{\mu}$, as the least binary relations satisfying (S1)–(W2) in Fig. 7, with, in place of (S2):

(S2′) $p \xrightarrow{\mu} p'$ implies $[p] \xrightarrow{\mu} [p']$.

As an example, we can derive

$$(a \mid \bar{a}.b) \mid c.\bar{b} \xrightarrow{\bar{a}}_{\mathrm{d}} (a \mid [b]) \mid c.\bar{b},$$

$$(a \mid \bar{a}.b) \mid c.\bar{b} \overset{c}{\Rightarrow}_{\mathrm{d}} (nil \mid nil) \mid [nil].$$

So, the main difference between the location semantics and that based on distributed bisimulations does not lie in their transition system but in the equivalences based on them; in the former observing a location results in assigning a name to it, while in the latter it results in splitting the process into a local subprocess and a remaining subprocess.

Definition 5.1 (*Distributed bisimulation equivalence*). A symmetric relation $R \subseteq \mathrm{CCS}_{\mathrm{rf}} \times \mathrm{CCS}_{\mathrm{rf}}$ is called a *distributed bisimulation* iff $R \subseteq D(R)$, where $(p,q) \in D(R)$ iff

(i) $p \overset{\varepsilon}{\Rightarrow} p'$ implies $q \overset{\varepsilon}{\Rightarrow} q'$ for some $q' \in \mathrm{CCS}_{\mathrm{rf}}$ such that $(p',q') \in R$,

(ii) $p \overset{a}{\Rightarrow}_{\mathrm{d}} \mathscr{C}[p']$ implies $q \overset{a}{\Rightarrow}_{\mathrm{d}} \mathscr{D}[q']$ for some $\mathscr{D}[q'] \in PROC_{[\ ]}$ such that $(p',q') \in R$ and $(\mathscr{C}(nil), \mathscr{D}(nil)) \in R$.

Two distributed processes p and q are said to be *distributed-bisimulation-equivalent*, $p \approx_{\mathrm{d}} q$, if and only if there is a distributed bisimulation R such that $(p,q) \in R$. □

As an example of nonequivalent processes, consider $a.b+b.a$ and $a|b$. If $a.b+b.a \stackrel{a}{\Rightarrow}_d [b]$ then $a|b$ can match this move only with $a|b \stackrel{a}{\Rightarrow}_d [nil]|b$. So, we have to compare the local subprocesses, b and nil, and the remaining ones, i.e. $\mathscr{C}(nil)=nil$ and $\mathscr{D}(nil)=nil|b$. Obviously, in both cases the equivalence does not hold.

On the other hand,

$$p_1=(a|\bar{a}.b)|c.\bar{b}+c.\bar{b}|b \approx_d (a|\bar{a}.b)|c.\bar{b}=q_1.$$

For example, $p_1 \stackrel{c}{\Rightarrow}_d [\bar{b}]|b$, which can be matched by q_1 with $q_1 \stackrel{c}{\Rightarrow}_d nil|b|[\bar{b}]$; it is obvious that the processes at the current locations are equivalent and so are the remaining processes $nil|b$ and $nil|b|nil$.

Distributed bisimulation equivalence is not preserved by $+$ and $|$, for the same reason as location and bisimulation equivalence are not preserved by them and, so, again we consider the largest relation $\approx_d^c$ contained in $\approx_d$ which is preserved by all operators. It can be characterised in the standard way.

Proposition 5.2. *$p \approx_d^c q$ iff*

(i) *$p \stackrel{\tau}{\Rightarrow} p'$ implies $q \stackrel{\tau}{\Rightarrow} q'$ such that $p' \approx_d q'$,*

(ii) *$p \stackrel{a}{\Rightarrow}_d \mathscr{C}[p']$ implies $q \stackrel{a}{\Rightarrow}_d \mathscr{D}[q']$ such that $p' \approx_d q'$ and $\mathscr{C}(nil) \approx_d \mathscr{D}(nil)$,*

(iii) *$q \stackrel{\tau}{\Rightarrow} q'$ implies $p \stackrel{\tau}{\Rightarrow} p'$ such that $p' \approx_d q'$,*

(iv) *$q \stackrel{a}{\Rightarrow}_d \mathscr{D}[q']$ implies $p \stackrel{a}{\Rightarrow}_d \mathscr{C}[p']$ such that $p' \approx_d q'$ and $\mathscr{C}(nil) \approx_d \mathscr{D}(nil)$*

iff $p+a \approx_d q+a$ for some $a \notin sort(p) \cup sort(q)$. □

We next show that $\approx_l$ and $\approx_d$ for processes in CCS_{rf} coincide. The major difference between these two equivalences lies in the fact that in the case of distributed-bisimulation equivalence, after a visible move, the process being observed is split into two, the two parts being subsequently observed independently. Another difference is the way locations are observed. We establish with the next lemma the relationship between the different kinds of observations. Based on that, we then obtain the required splitting for location terms by applying the decomposition proposition (Proposition 3.15).

Lemma 5.3. *Let $p \in CCS_{rf}$ and $\mathscr{D}'=\{(P2),(P3)\}$ (cf. Fig. 10). Then*

(1) *if $p \stackrel{a}{\underset{l}{\Rightarrow}} p'$ then there exist $\mathscr{C}$ and $p'' \in CCS_{rf}$ such that $p \stackrel{a}{\Rightarrow}_d \mathscr{C}[p'']$ and $p' \equiv_{D'} (l::p''|\mathscr{C}(nil))$.*

(2) *if $p \stackrel{a}{\Rightarrow}_d \mathscr{C}[p'']$ then there exists p' such that $p \stackrel{a}{\underset{l}{\Rightarrow}} p'$ and $p' \equiv_{D'} (l::p''|\mathscr{C}(nil))$.*

Proof. By induction on the length of the proof of transitions. □

Proposition 5.4. *Let $p, q \in CCS_{rf}$. Then $p \approx_l q$ iff $p \approx_d q$.*

Proof. By induction on the sum of sizes of p and q, where the size of a term is defined as the number of actions occurring in it. Without mentioning it, we will make use of the fact that equations in D' preserve the size of processes.

$\Rightarrow$: Assume that $p \stackrel{a}{\Rightarrow}_{\mathrm{d}} \mathscr{C}[p'']$.

By the previous lemma, $p \stackrel{a}{\underset{l}{\Rightarrow}} p'$, where $p' \equiv_{D'} (l :: p'' \mid \mathscr{C}(nil))$. Since $p \approx_\ell q$, there is $q \stackrel{a}{\underset{l}{\Rightarrow}} q'$ such that $p' \approx_\ell q'$. By Lemma 5.3 again, there exist $\mathscr{D}$ and q'' such that $q \stackrel{a}{\Rightarrow}_{\mathrm{d}} \mathscr{D}[q'']$ with $q' \equiv_{D'} (l :: q'' \mid \mathscr{D}(nil))$. As the equations in D' are sound for $\approx_\ell$, we derive $(l :: q'' \mid \mathscr{D}(nil)) \approx_\ell (l :: p'' \mid \mathscr{C}(nil))$. Applying the decomposition of Proposition 3.15, we obtain $q'' \approx_\ell p''$ and $\mathscr{D}(nil) \approx_\ell \mathscr{C}(nil)$. Now induction yields $q'' \approx_{\mathrm{d}} p''$ and $\mathscr{D}(nil) \approx_{\mathrm{d}} \mathscr{C}(nil)$, which is what we were required to show.

The case $p \stackrel{\varepsilon}{\Rightarrow} p'$ is immediate, since the ε-transitions are the same in both systems.

$\Leftarrow$: Assume $p \stackrel{a}{\underset{l}{\Rightarrow}} p'$.

By Lemma 5.3, there exist $\mathscr{C}$ and p'' such that $p \stackrel{a}{\Rightarrow}_{\mathrm{d}} \mathscr{C}[p'']$ with $p \equiv_{D'} (l :: p'' \mid \mathscr{C}(nil))$. Moreover, since $p \approx_{\mathrm{d}} q$, there is $q \stackrel{a}{\Rightarrow}_{\mathrm{d}} \mathscr{D}[q'']$ such that $p'' \approx_{\mathrm{d}} q''$ and $\mathscr{C}(nil) \approx_{\mathrm{d}} \mathscr{D}(nil)$. Induction yields $p'' \approx_\ell q''$ and $\mathscr{C}(nil) \approx_\ell \mathscr{D}(nil)$. By Lemma 5.3 again, there exists q' such that $q \stackrel{a}{\underset{l}{\Rightarrow}} q'$, with $q' \equiv_{D'} (l :: q'' \mid \mathscr{D}(nil))$. Now, as the equations in D' are sound for $\approx_\ell$, we can conclude $p' \approx_\ell q'$.

The case $p \stackrel{\varepsilon}{\Rightarrow} p'$ is obvious. □

Corollary 5.5. *Let* $p, q \in \mathrm{CCS}_{\mathrm{rf}}$. *Then* $p \approx_\ell^{\mathrm{c}} q$ *iff* $p \approx_{\mathrm{d}}^{\mathrm{c}} q$.

The equivalence of $\approx_\ell$ and $\approx_{\mathrm{d}}$ immediately yields the following result.

Proposition 5.6. *On* $\mathrm{CCS}_{\mathrm{rf}}$, $\approx_\ell^{\mathrm{c}}$ *is equal to the congruence generated by the equations G in Fig.* 12.

Proof. See Theorem 4.12 in [11].

In [11] the definition of distributed bisimulation was extended to all of finite CCS. We do not give the definition here but we can show that the resulting equivalence is different from $\approx_\ell$. For example, the two processes p and q, defined below are equivalent according to the generalised definition of distributed bisimulation but are differentiated by $\approx_\ell$.

$$p = (c.(\alpha.\textstyle\sum_a + \beta.\prod_a + \bar{\alpha}.\sum_b + \bar{\beta}.\prod_b) \mid (\bar{\beta}.\sum_b + \bar{\alpha}.\prod_b + \alpha.\sum_a + \beta.\prod_a)) \setminus \{\alpha, \beta\},$$

$$q = (c.(\alpha.\textstyle\sum_a + \beta.\prod_a + \bar{\alpha}.\sum_b + \bar{\beta}.\prod_b) \mid (\bar{\alpha}.\sum_b + \bar{\beta}.\prod_b + \beta.\sum_a + \alpha.\prod_a)) \setminus \{\alpha, \beta\},$$

where

$$\textstyle\sum_a = a_1.a_2.nil + a_2.a_1.nil$$

and

$$\textstyle\prod_a = a_1.nil \mid a_2.nil.$$

Intuitively, the difference between these two equivalences is due to the fact that, in the course of experimenting on processes by means of distributed bisimulation, locations which have been observed are forgotten in later states. Hence, a visible transition of

one process can be matched with a visible transition of the counterprocess which originally was at a different location. This shows that location equivalence is a better choice than generalized distributed bisimulation when looking for an extension of distributed bisimulation to CCS processes with restriction.

Finally, we should point out that location equivalence appears to be orthogonal to causality-based equivalences. For example, $\approx_l$ distinguishes the processes $(a.\alpha.c \mid b.\bar{\alpha}.d)\backslash\alpha$, $(a.\alpha.d \mid b.\bar{\alpha}.c)\backslash\alpha$ which would be identified by behavioural equivalences based on causality such as that in [7]. On the other hand, the processes $a.b \mid c.d$ and $(a.(\alpha.b+b) \mid c(\bar{\alpha}.d+d))\backslash\alpha$ are identified by $\approx_l$ although they have a different causal structure.

6. Conclusions

Most of the recent research into noninterleaving semantic theories for concurrent systems has centred on introducing some aspect of causality into their own observation. We believe that it is very difficult for an external observer to confirm causal dependencies between actions performed by independent systems and, if such dependencies can be determined, this may be done only in a very indirect manner. For this reason, we believe that it is inappropriate to base an extensional semantic theory of processes on notions such as causality although they may be appropriate for intensional purposes, such as defining the operational behaviour of processes. Instead, we have focused on a different aspect of systems, namely, their distribution in space. For distributed systems, it seems natural to assume that an external observer can discern which sites within the system react to particular experiments or requests for information. We hope that the present paper gives some arguments to convince the reader that a reasonable and useful semantic theory can be based on such ideas and that it can be applied to nontrivial languages.

Very similar ideas underlie the various definitions of *distributed-bisimulation equivalence* [4, 5, 11] and indeed the main contribution of the present paper is to reformulate this approach in order to make it more accessible. This new location-based equivalence is easy to understand, applies to all of CCS, and its formulation makes its relationship with the standard bisimulation equivalence transparent. Indeed, it should be possible to adapt much of the work done on bisimulation equivalence to location equivalence. This includes generalising the algorithms for checking and generating bisimulations and those for checking that processes satisfy recursively defined modal formulae [6]. However, this generalisation is not completely straightforward because, for example, the relations which establish location equivalence between even finite processes are infinite. This was certainly the case with the examples we examined in the earlier sections.

Another interesting area of research would be the development of a specification language, an extension of CCS, more suited to location equivalence or, more generally, to expressing properties of the distribution of processes and the application of such a language to nontrivial examples.

The introduction of locations into the language for describing processes also enables us to define what it means for one location to be a sublocation of another. This has been shown to be of use in the modal language defined in Section 4. It could also be used to define a "concurrency preorder" between processes. Briefly, $p \leqslant q$ would mean that p and q have more or less the same behaviour but q is possibly more concurrent than p. Such a preorder could be defined by relaxing the condition that matching actions be performed at exactly the same locations; instead, an action from q could be matched by one from p performed at a more specified location. This would imply that p is possibly more sequential than q or q is possibly more concurrent than p. However, there is considerable room for discussion as to what exactly the formal definition should be.

Finally, we should mention that location equivalence lacks a finite axiomatisation when restricted to finite terms in CCS. This is one of the most attractive features of bisimulation equivalence and it would be nice to have such an axiomatisation for location equivalence, especially if it did not use many auxiliary operators.

Acknowledgment

The authors thank Luca Aceto for many useful and illuminating discussions.

References

[1] J. Bergstra and J.W. Klop, Algebra of communicating processes with abstraction. *Theoret. Comput. Sci.* **37** (1985) 77–121.

[2] G. Boudol and I. Castellani, On the semantics of concurrency: partial orders and transition systems, in: *Proc. of TAPSOFT 87*, Lecture Notes in Computer Science, Vol. 249 (Springer, Berlin, 1987) 123–137.

[3] G. Boudol and I. Castellani, Permutation of transitions: an event structure semantics for CCS and SCCS, in: *Proc. of Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Lecture Notes in Computer Science, Vol. 354 (Springer, Berlin, 1989) 411–427.

[4] I. Castellani, Bisimulations for concurrency. Ph.D. Thesis, University of Edinburgh, 1988.

[5] I. Castellani and M. Hennessy, Distributed bisimulations. *J. ACM* **10** (1989) 887–911.

[6] R. Cleaveland, J. Parrow and B. Steffen, A semantics based verification tool for finite state systems, in: *Proc. 9th Internat. Symp. on Protocol Specification, Testing and Verification* (North-Holland, Amsterdam, 1989).

[7] P. Degano and P. Darondeau, Causal trees, in: *Proc. ICALP 88*, Lecture Notes in Computer Science, Vol. 372 (Springer, Berlin, 1989) 234–248.

[8] P. Degano, R. De Nicola and U. Montanari, A distributed operational semantics for CCS based on condition/event systems. *Acta Inform.* **26** (1988) 59–91.

[9] M. Hennessy, Axiomatising finite concurrent processes, *SIAM J. Comput.* **17** (1988) 997–1017.

[10] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[11] A. Kiehn, Distributed bisimulations for finite CCS, Report 7/89, University of Sussex, 1989.

[12] K. Larsen, Proof systems for satisfiability in Hennessy–Milner logic with recursion, *Theoret. Comput. Sci.* **72** (1990) 265–288.

[13] R. Milner, *Communication and Concurrency* (Prentice-Hall, Englewood Cliffs, NJ, 1989).

[14] C. Stirling, Modal logics for communicating systems, *Theoret. Comput. Sci.* **49** (1987) 311–347.