Internal Report TCS-92-2
November, 1992

CCS, Locations and Asynchronous Transition Systems

Madhavan Mukund [1] and Mogens Nielsen[2]

Abstract

Our aim is to provide a simple non-interleaved operational semantics for CCS in terms of a model that is easy to understand—asynchronous transition systems. Our approach is guided by the requirement that the semantics should identify the concurrency present in the system in a natural way, in terms of events occurring at independent locations in the system.

We extend the standard interleaving transition system for CCS by introducing labels on the transitions with information about the locations of events. We then show that the resulting transition system is an asynchronous transition system which has the additional property of being *elementary*, which means that it can also be represented by a 1-safe net.

We establish a close correspondence between our semantics and other approaches in terms of *foldings*.

We also introduce a notion of bisimulation on asynchronous transition systems which preserves independence. We conjecture that the induced equivalence on CCS processes coincides with the notion of *location equivalence* proposed by Boudol et al.

[1]School of Mathematics, SPIC Science Foundation, 92, G.N. Chetty Road, T. Nagar, Madras 600 017, India. E-mail: `madhavan@ssf.ernet.in`.

The work reported here was done while the author was visiting the Computer Science Department, Aarhus University, Denmark. The author's stay in Denmark was supported by a grant from the Aarhus University Science Foundation.

[2]Computer Science Department, Aarhus University, Ny Munkegade, DK-8000 Aarhus C, Denmark. E-mail: `mn@daimi.aau.dk`

This report is also published as DAIMI-PB-395, Computer Science Department, Aarhus University, Aarhus, Denmark.

1 Introduction

Process algebras like CCS [9] are a well established formalism for specifying concurrent systems. However, the traditional semantics for these languages is given in terms of labelled transition systems in which parallel composition is interpreted as non-deterministic interleaving.

Several attempts have been made to provide a non-interleaved semantics for CCS-like languages, in which the concurrency implicit in a process expression P is explicitly represented in the semantics of P.

One approach is to incorporate information about concurrency into the conventional sequential transition system describing the behaviour of P by introducting an algebra of transitions. This can be done implicitly, by decorating each transition with a "proof of its derivation" [3], or explicitly, as in [8].

Another approach is to interpret process expressions in terms of a richer model—typically Petri nets [3, 6, 11]. This involves decomposing a process into local "components" and then constructing a net from these components.

Both these approaches have the drawback that they are complicated. When one introduces an algebra over the transitions, one has to go through a second level of reasoning about the transition system in order to identify the underlying *events* in the system (where we use the term event in the sense of Petri nets).

On the other hand, translating a term directly into a Petri net suffers from the usual problems associated with explicitly manufacturing one particular net which gives rise to the required behaviour—a lot of effort has to be put into to creating the places and hooking them up correctly to the transitions and then proving that the net one has constructed does in fact exhibit the required behaviour.

Our aim is to provide a simple non-interleaved operational semantics for CCS in terms of a model that is easy to understand. The criteria we have in mind are the following.

- The semantics should be as close to the original (interleaving) transition system as possible.
- The semantics should identify the concurrency present in the system in a natural way, in terms of events occurring at independent locations in the system.
- The structure representing the behaviour should be finite, whenever possible—this means that we should treat recursion carefully and unfold the system only if necessary.
- There should be a way to formally relate our semantics to other existing approaches.
- The semantics should be simple!

With this in mind, we choose to interpret CCS over the class of asynchronous transition systems [2, 12, 13]. An asynchronous transition system is a normal transition system where the labels are viewed as events. The transition system comes equipped with a binary relation on the events which specifies when two events in the system are independent of each other.

Asynchronous transition systems and Petri nets are closely related to each other. In [13], it is shown that one can define a subclass of "elementary" asynchronous transition systems which are, in a precise sense, equivalent to 1-safe Petri nets.

To interpret CCS in terms of asynchronous transition systems, we restrict the language slightly. Instead of the normal operator + expressing non-deterministic choice, we use guarded choice. In other words, we restrict terms with + to be of the form $aP + bQ$, where a and b could be the invisible action τ—we do not permit general expressions of the form $P + Q$. The rest of the language is the same as in standard CCS. We claim that this language is still a very powerful and useful language for specifying concurrent systems. For instance, all the examples in [9] conform to our syntactic restriction.

To obtain an asynchronous transition system from a CCS expression, we decorate transitions with labels which indicate the *location* where the action occurs. While this is in the same spirit as decorating transitions with their proofs, it turns out that our labelling directly gives us the underlying events of the system. In other words, we do not need to impose an algebra on our labels to identify events—two transitions correspond to the occurrence of the same event iff they carry the same label. The independence relation we define on events reflects a natural notion of independence on locations.

We then prove that the asynchronous transition system $LTS(P)$ we associate with a process expression P is in fact elementary. This means that we obtain "for free" a Petri net semantics for our language, by appealing to the results of [13].

We express the connection between our semantics and some other approaches in terms of *foldings*. These are special types of bisimulations in which the target of the folding is, in general, a smaller, more compact representation of the behaviour described by the first system. We show that the normal interleaved transition system for our language, as defined in [9], can be folded onto the asynchronous transition system we define. On the other hand, in [13], a denotational semantics for CCS is presented in terms of asynchronous transition systems. We show that the denotational transition system associated with a term P can also be folded onto the transition system we associate operationally with P.

Finally, we define a notion of bisimulation on asynchronous transition systems which preserves independence. When applied to the asynchronous transition systems we use to describe the behaviour of CCS processes, this gives rise to an equivalence on process terms which we conjecture is equivalent to the notion of location equivalence defined in [5].

The paper is organized as follows. We begin with a brief introduction to asynchronous transition systems. The next section introduces the process language and its operational semantics. Section 4 establishes that the asynchronous transition system we define for a process P is elementary. In Section 5, we show how to relate our semantics with other standard approaches. The next section describes bisimulations on asynchronous transition systems. We conclude in Section 7 with a discussion of how our work relates to other approaches and suggestions for further study.

2 Asynchronous transition systems

We begin by recalling the standard notion of a (labelled sequential) transition system.

Definition 2.1 (Transition system)
A transition system *is a quadruple* $TS = (S, i, E, Tran)$ *where*

- *S is a set of* states, *with* initial state *i.*
- *E is a set of* events.
- *$Tran \subseteq S \times E \times S$ is the* transition relation.

Asynchronous transition systems were introduced independently by Bednarczyk [2] and Shields [12]. These transition systems incorporate information about concurrency explicitly, in terms of a binary relation which specifies which pairs of events are independent of each other. The particular definition we adopt here is from [13].

Definition 2.2 (Asynchronous transition systems)
An asynchronous transition system *is a structure $ATS = (S, i, E, I, Tran)$ such that*

- *$(S, i, E, Tran)$ is a transition system.*
- *$I \subseteq E \times E$ is an irreflexive, symmetric,* independence relation *satisfying the following four conditions:*

  (i) $e \in E \Rightarrow \exists s, s' \in S.\ (s, e, s') \in Tran.$

  (ii) $(s, e, s') \in Tran$ *and* $(s, e, s'') \in Tran \Rightarrow s' = s''.$

  (iii) $e_1 I e_2$ *and* $(s, e_1, s_1) \in Tran$ *and* $(s, e_2, s_2) \in Tran$
  $\Rightarrow \exists u.\ (s_1, e_2, u) \in Tran$ *and* $(s_2, e_1, u) \in Tran.$

  (iv) $e_1 I e_2$ *and* $(s, e_1, s_1) \in Tran$ *and* $(s_1, e_2, u) \in Tran$
  $\Rightarrow \exists s_2.\ (s, e_2, s_2) \in Tran$ *and* $(s_2, e_1, u) \in Tran.$

Condition (i) stipulates that every event in E must appear as the label of some transition in the system. The second condition guarantees that the system is deterministic. The third and fourth conditions express properties of independence: condition (iii) says that if two independent events are enabled at a state, then they should be able to occur "together" and reach a common state; condition (iv) says that if two independent events occur immediately after one another in the system, it should also be possible for them to occur with their order interchanged.

Asynchronous transition systems can be equipped with a natural notion of morphism to form a category [13]. In [13], Winskel and Nielsen establish a coreflection between a subcategory of asynchronous transition systems, which we shall call *elementary* asynchronous transition systems, and a category of 1-safe Petri nets.

Here, we shall restrict our attention to the correspondence between these two categories at the level of objects. At this level, what the coreflection establishes is that given an elementary asynchronous transition system, we can construct a 1-safe Petri net whose case graph is isomorphic to the transition system we started with.

The extra axioms characterizing elementary asynchronous transition systems are phrased in terms of generalized *regions*.

Definition 2.3 (Regions)
Let $ATS = (S, i, E, I, Tran)$ be an asynchronous transition system. A region *of ATS is a pair of functions $r = (r_S, r_E)$ where*
$r_S : S \to \{0, 1\}$ *and*
$r_E : E \to (\{0, 1\} \times \{0, 1\})$ *such that*

(i) $\forall (s, e, s') \in Tran.\ (r_E(e) = (1,0)\ or\ r_E(e) = (1,1)) \Rightarrow r_S(s) = 1.$

(ii) $\forall (s, e, s') \in Tran.\ r_S(s') = r_S(s) + x_2 - x_1,\ where\ r_E(e) = (x_1, x_2).$

(iii) *Let* $e_1, e_1' \in E$ *and* $r_E(e_1) = (x_1, x_2)$ *and* $r_E(e_1') = (x_1', x_2').$
If $e_1 I e_1'$ *then* $((x_1 = 1)\ or\ (x_2 = 1)) \Rightarrow x_1' = x_2' = 0.$

For convenience, we shall refer to both components, r_S and r_E, of a region r simply as r.

Regions correspond to the places of the 1-safe net that one would like to associate with an elementary asynchronous transition system ATS. Intuitively, we want to associate with ATS a 1-safe Petri net N such that ATS represents the case graph of N. Thus, the states of ATS should correspond to the reachable markings of N and the events of ATS should correspond to the transitions of N. Let r be a region in ATS. We specify whether or not the "place" r is marked at the "marking" s by $r(s)$. For each "transition" e, $r(e)$ says how r is "connected" to e in the associated net. Conditions (i) and (ii) in the definition of a region then correspond to the firing rule for Petri nets. Condition (iii) reflects the intuition that two transitions in a net are independent provided their neighbourhoods are disjoint.

In [13], regions (or *conditions*, as they are called there) are described slightly differently, in terms of subsets of states and transitions. Our formulation is equivalent to the one in [13]. Our regions are generalizations of those originally used to describe the connection between elementary net systems and elementary transition systems [7, 10].

We introduce some notational conventions for regions, borrowed from net theory. Given a region r and an event e, we say that $r \in {}^\bullet e$ if $r(e) = (1,0)$ or $r(e) = (1,1)$. Similarly, we say that $r \in e^\bullet$ if $r(e) = (0,1)$ or $r(e) = (1,1)$. We say that $e \in {}^\bullet r$ if $r \in e^\bullet$ and $e \in r^\bullet$ if $r \in {}^\bullet e$. Finally, for $s \in S$, we sometimes say that $s \in r$, or that r *holds at* s, to indicate that $r(s) = 1$.

We can now characterize elementary asynchronous transition systems.

Definition 2.4 (Elementary asynchronous transition systems)
Let $ATS = (S, i, E, I, Tran)$ *be an asynchronous transition system.* ATS *is said to be* elementary *if it satisfies the following three conditions.*

(i) *Every state in* S *is reachable by a finite sequence of transitions from the initial state* i. *(Reachability)*

(ii) $\forall s, s' \in S.\ s \neq s' \Rightarrow \exists$ *a region* $r.\ r(s) \neq r(s').$ *(Separation)*

(iii) $\forall s \in S.\ \forall e \in E.$ *If there does not exist* s' *such that* $(s, e, s') \in Tran$, *then there exists an* $r \in {}^\bullet e$ *such that* $r(s) = 0.$ *(Enabling)*

Call a region r *non-trivial* iff there exists some $e \in E$ such that $r \in {}^\bullet e$ or $r \in e^\bullet$. Clearly, for a trivial region r, $r(s) = r(s')$ for all $s, s' \in S$. So, it follows that conditions (ii) and (iii) in the definition of elementary asynchronous transition systems actually require the existence of a non-trivial region satisfying the required properties.

As we have mentioned before, given an elementary asynchronous transition system ATS, we can construct a 1-safe net N_{ATS} whose case graph is isomorphic to ATS. The transitions of N_{ATS} are given by the events of ATS and the places of N_{ATS} are given by

the regions of ATS. We shall not go into the details of the construction of N_{ATS}. The interested reader is referred to [13]. (A similar construction is described for going from elementary transition systems to elementary net systems in [10]).

A final point that should be emphasized is that so far we have been working with *unlabelled* asynchronous transition systems. Thus, the "labels" on the transitions correspond to the events of the system and are analogous to the "names" of the transitions of a Petri net. To relate asynchronous transition systems to, say, process algebras like CCS, we have to add an extra layer of labelling by means of a labelling function as follows.

Definition 2.5 (Labelled asynchronous transition systems) *Let Σ be an alphabet. A Σ-labelled asynchronous transition system is a pair (ATS, l) where $ATS = (S, i, E, I, Tran)$ is an asynchronous transition system, and $l : E \to \Sigma$ is a labelling function.*

Thus, for the CCS term $a\ nil \| a\ nil$, we would associate an asynchronous transition system with two events e_1 and e_2 which are independent of each other, both labelled a.

Given a labelled elementary asynchronous transition system, we can easily transport the labelling to the corresponding net that we construct since we keep track of the underlying events. In fact, in [13] it is shown that the coreflection between unlabelled elementary asynchronous transition systems and 1-safe Petri nets can be lifted in a natural way to a coreflection between the corresponding categories of labelled systems. We shall not go into the details here.

3 The language and its operational semantics

The process language we consider is a subset of CCS where choice is always guarded.

We fix a set of actions $Act = \Lambda \cup \overline{\Lambda}$, where Λ is a set of names ranged over by $\alpha, \beta, \ldots$ and $\overline{\Lambda}$ is the corresponding set of co-names $\{\overline{\alpha} \mid \alpha \in \Lambda\}$. As usual, we assume that $\bar{}$ is a bijection such that $\overline{\overline{\alpha}} = \alpha$ for all $\alpha \in \Lambda$. The symbol $\tau \notin Act$ denotes the invisible action. We use $a, b, c \ldots$ to range over Act and $\mu, \nu \ldots$ to range over $Act_\tau = Act \cup \{\tau\}$. We also assume a set V of process variables and let $x, y, \ldots$ range over V.

The set of process expressions $Proc$ is given by the following grammar.

$$P ::= \sum_{i \in I} \mu_i P_i \mid P \| P \mid P \backslash \alpha \mid x \mid rec\ x.P$$

where $\mu_i \in Act_\tau$, $\alpha \in \Lambda$ and $x \in V$.

Thus, the guarded sum $\sum_{i \in I} \mu_i P_i$ represents the process which can execute any one of the actions μ_i (which could be τ) and evolve into the corresponding process E_i. The indexing set I could, in general, be infinite. We abbreviate by *nil* the process consisting of a guarded sum indexed by the empty set. The normal CCS prefixing operator aP is represented by a guarded sum over a singleton index set.

The other constructs are standard. $P_1 \| P_2$ denotes the parallel composition of P_1 and P_2—i.e. the process consisting of P_1 and P_2 executing independently, with the possibility of synchronization. $P \backslash \alpha$ denotes the restriction of P with respect to α—$P \backslash \alpha$ is the process that behaves like P but is not permitted to perform visible actions α or $\overline{\alpha}$. $rec\ x.P$ is the process satisfying the recursive definition $x = P$. We assume that each occurrence of x in P is guarded by an action $\mu \in Act_\tau$—in other words, each x in P appears within

a subterm of the form $\sum_{i\in I}\mu_i P_i$. We do not consider the relabelling operator, though it could be incorporated without too much difficulty into our setup.

We enrich the standard operational semantics of CCS by adding some information on the labels of the transitions which will permit us to directly extract the underlying events of the transition system representing the behaviour of a CCS term.

We have to depart slightly from the traditional transition system for CCS, where states are given directly by process expressions. We will need to identify recursive processes with their one-step unfoldings. So, define $\equiv$ to be the least congruence with respect to the operators $\|$ and $\backslash\alpha$ such that

$$rec\ x.P \equiv P[rec\ x.P/x]$$

where, as usual, $P[rec\ x.P/x]$ denotes the term obtained by substituting $rec\ x.P$ for x in P. For $P \in Proc$, let $[P]$ denote the set of process expressions equivalent to P. The states of our transition system will effectively be equivalence classes of process expressions.

Our operational semantics is defined as follows (henceforth we assume that $\{0,1\}\cap\Lambda = \emptyset$):

$$P = \sum_{i\in I}\mu_i P_i \xrightarrow[[P][P_i]]{\mu_i} P_i \qquad \textbf{(Sum)}$$

$$P_0 \xrightarrow[u]{\mu} P_0' \quad implies \quad P_0\|P_1 \xrightarrow[0u]{\mu} P_0'\|P_1,\ P_1\|P_0 \xrightarrow[1u]{\mu} P_1\|P_0' \qquad \textbf{(Par)}$$

$$P_0 \xrightarrow[u]{a} P_0',\ P_1 \xrightarrow[v]{\overline{a}} P_1' \quad implies \quad P_0\|P_1 \xrightarrow[\langle 0u,1v\rangle]{\tau} P_0'\|P_1' \qquad \textbf{(Com)}$$

$$P \xrightarrow[u]{\mu} P' \quad implies \quad P\backslash\alpha \xrightarrow[\alpha u]{\mu} P'\backslash\alpha,\ \mu \notin \{\alpha,\overline{\alpha}\} \qquad \textbf{(Res)}$$

$$P \xrightarrow[u]{\mu} P',\ P \equiv P_1\ and\ P' \equiv P_1' \quad implies \quad P_1 \xrightarrow[u]{\mu} P_1' \qquad \textbf{(Struct)}$$

So, for a basic action performed by a process of the form $\sum_{i\in I}\mu_i P_i$, we tag the transition with the source and target process expressions. We extend the tag with 0's and 1's on the left as we lift the transition through the left and right branches of a parallel composition. With each communication, we keep track of the tags corresponding to the two components participating in the communication. By extending the tag to the left with α for each restriction $\backslash\alpha$, we keep track of the nesting of restrictions with respect to the overall structure of the process. This will be crucial in order to be able to determine whether or not a communication is possible even though the visible actions which make up the communication are restricted away. Finally, (Struct) ensures that processes from the same equivalence class are capable of making exactly the same moves.

The string of 0's and 1's which we use to tag a transition when we move down the left and right branches of a parallel composition essentially pins down the *location* where the transition occurs. Locations will provide us with a natural way of identifying independence between transitions. Our notion of location is closely related to the static approach advocated by Aceto [1] for dealing with the idea of locations introduced by Boudol et al [4, 5]. We have more to say on the connection between our approach and the approach of [1, 4, 5] in Section 6.

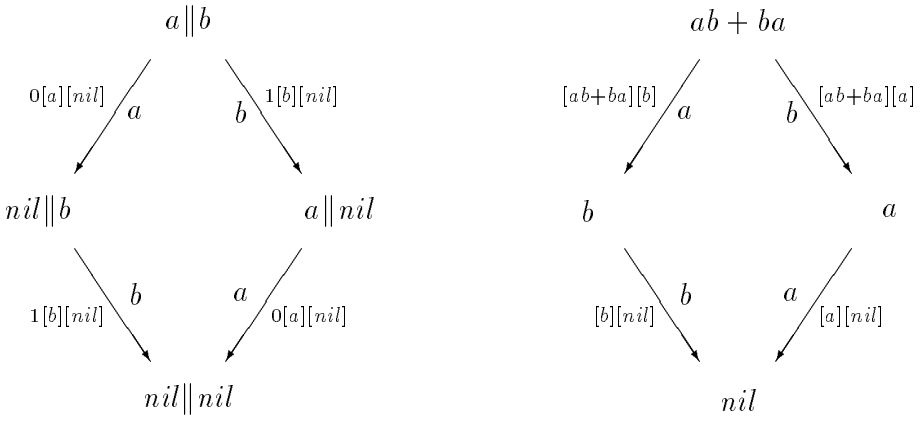

Figure 1: Concurrency versus non-deterministic interleaving

It is not difficult to establish that the states of the transition system defined by our operational semantics are in fact equivalence classes of process expressions. Formally, we have the following proposition.

Proposition 3.1
$\forall P, P'.\ \forall \mu.\ \forall u.\ (\ P \xrightarrow[u]{\mu} P'\ \textit{iff}\ \forall P_1 \equiv P.\ \forall P_1' \equiv P'.\ P_1 \xrightarrow[u]{\mu} P_1'\ ).$

We conclude this section with a couple of examples. The first example illustrates how our semantics distinguishes concurrency from non-deterministic interleaving. Figure 1 shows the behaviour of the processes $a\|b$ and $ab + ba$. Notice that the transition system for $a\|b$ has four transitions, but only two distinct labels on the transitions. This captures the fact that there are only two underlying (independent) events, one labelled a and the other labelled b. In contrast, the process $ab + ba$ has four distinct events.

The next example illustrates how our semantics deals with recursion. Consider the two processes $rec\ x.ax\|rec\ x.ax$ and $rec\ x.(ax\|ax)$. The transition systems corresponding to these processes are shown in Figure 2.

The standard interleaved transition system for the first process would consist of a single state with a single a-labelled transition looping back to that state. On the other

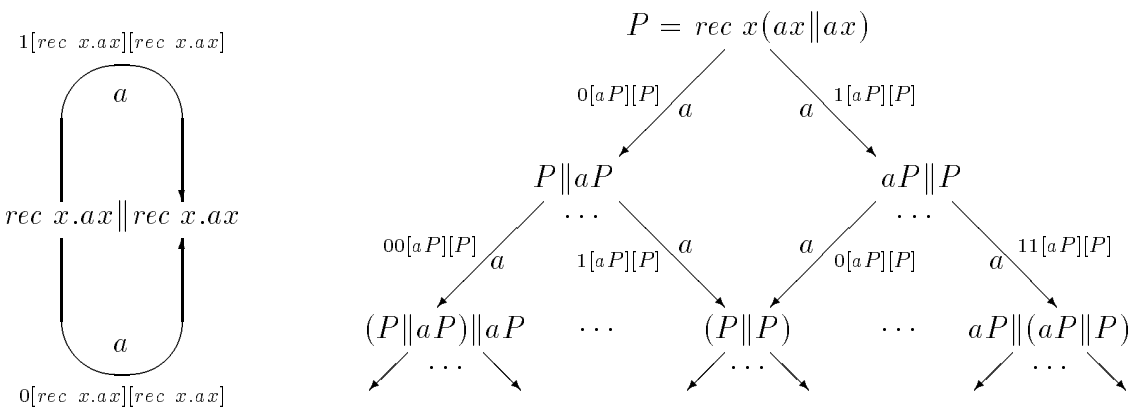

Figure 2: Recursion

hand, our semantics generates *two* a-labelled transitions, because, intuitively, the a could occur at two different locations in the process. However, notice that our semantics still yields a finite transition system for this term.

The standard interleaved transition system for $rec\ x.(ax \| ax)$ is infinite, because unfolding the term creates more and more components in parallel. Our semantics also assigns an infinite transition system in such a case. Notice though, that it still keeps track of the underlying events in a consistent manner. Thus, the two top level a-transtions (labelled $\xrightarrow[0[aP][P]]{a}$ and $\xrightarrow[1[aP][P]]{a}$) correspond to events which are distinct from the a-transitions arising in the new components generated by the unfolding (for example, the transition labelled $\xrightarrow[00[aP][P]]{a}$).

The two examples in Figure 2 illustrate a general point about our semantics—our semantics will assign a finite transition system to a process P whenever the standard interleaving semantics would do so. This connection is made more precise in Section 5.

4 From CCS to asynchronous transition systems

We would like to establish that the transition system describing the behaviour of a process $P \in Proc$ is in fact a (labelled) elementary asynchronous transition system.

To do this, we first show that the extra information that we have introduced into the labels of the transitions when defining the operational semantics of our process language is sufficient to distinguish the events of the underlying transition system and define a natural notion of independence between events.

We then prove that we have "enough" regions around to ensure that the transition system we are considering satisfies the two regional separation properties which are required for it to be elementary.

We begin by establishing a simple fact about the nature of the labels in the transition system defined by our operational semantics.

Proposition 4.1 (Syntax of labels)

For any transition $\hat{P} \xrightarrow[u]{\mu} \hat{P}'$, *$u$ is of the form*

(i) $s[P][P']$, *where* $s \in (\{0,1\} \cup \Lambda)^*$, *and* $\mu \in Act$.

(ii) $s[P][P']$, *where* $s \in (\{0,1\} \cup \Lambda)^*$, *and* $\mu = \tau$.

(iii) $s\langle s_0[P_0][P_0'], s_1[P_1][P_1']\rangle$, *where* $s, s_0, s_1 \in (\{0,1\} \cup \Lambda)^*$, *and* $\mu = \tau$.

Proof By induction on the length of the derivation of the transition $\hat{P} \xrightarrow[u]{\mu} \hat{P}'$. □

Henceforth, for convenience, we shall omit the brackets around the process expressions in the event labels and simply write sPP' instead of $s[P][P']$ and $s\langle s_0P_0P_0', s_1P_1P_1'\rangle$ instead of $s\langle s_0[P_0][P_0'], s_1[P_1][P_1']\rangle$.

Each distinct label in our transition system will correspond to an event, as follows.

Definition 4.2 (Events) *Define the set of events Ev as follows:*

$$Ev = \{(\mu, u) \mid \exists P, P' \in Proc.\ P \xrightarrow[u]{\mu} P'\}$$

For $e \in Ev$, we can identify $Loc(e) \subseteq \{0,1\}^*$, the location(s) *where e occurs, as follows:*

$$\forall e = (\mu, u).\ Loc(e) = \begin{cases} \{s \downarrow_{\{0,1\}}\} & \text{if } u = sPP' \\ \{ss_0 \downarrow_{\{0,1\}}, ss_1 \downarrow_{\{0,1\}}\} & \text{if } u = s\langle s_0 P_0 P_0', s_1 P_1 P_1'\rangle \end{cases}$$

where, for $s \in (\{0,1\} \cup \Lambda)^*$, $s \downarrow_{\{0,1\}}$ denotes the projection of s onto $\{0,1\}$. In other words, $s \downarrow_{\{0,1\}}$ is the subsequence of s obtained by erasing all elements not in $\{0,1\}$.

From the way we introduce information about locations into our event labels, it is clear that the location $Loc(e)$ of an event e is a string which identifies the nested component where e occurs. We can identify a natural independence relation on locations and lift it to events as follows.

Definition 4.3 (Independence on events)

Define an independence relation on locations $I_l \subseteq \{0,1\}^* \times \{0,1\}^*$ *as follows:*

$$\forall s, s' \in \{0,1\}^*.\ (s, s') \in I_l \text{ iff } s \not\preceq s' \text{ and } s' \not\preceq s$$

where $\preceq$ is the prefix relation on strings.

We can extend this to a relation $\hat{I}_l \subseteq (\{0,1\} \cup \Lambda)^* \times (\{0,1\} \cup \Lambda)^*$ in the obvious way.

$$\forall \hat{s}, \hat{s}' \in (\{0,1\} \cup \Lambda)^*.\ (\hat{s}, \hat{s}') \in \hat{I}_l \text{ iff } (\hat{s} \downarrow_{\{0,1\}}, \hat{s}' \downarrow_{\{0,1\}}) \in I_l$$

For convenience, we shall write both $\hat{I}_l$ and I_l as I_l.

Using I_l we can define an independence relation on events $I \subseteq Ev \times Ev$ *as follows:*

$$\forall e, e' \in Ev.\ (e, e') \in I \text{ iff } \forall s \in Loc(e).\ \forall s' \in Loc(e').\ (s, s') \in I_l$$

Having defined the set of events Ev and the independence relation I on events, we have essentially all the data we need to define an asynchronous transition system corresponding to the operational behaviour of a process P. However, we shall hold off defining this transition system for a while and first analyze the behaviour of P as given by the operational semantics in greater detail.

To analyze the behaviour of a process term P, we will need to decompose it into its sequential components.

Definition 4.4 (Component of P at location s)

$$Comp : (\{0,1\} \cup \Lambda)^* \times Proc \rightharpoonup Proc$$

is a (partial) function defined inductively as follows.

$$
\begin{array}{lcl}
Comp(\varepsilon, P) & = & P, \textit{ provided } P \neq \textit{rec } x.P_x \\
 & & \textit{(where } \varepsilon \textit{ is the empty string)} \\
Comp(0s, P_0 \| P_1) & = & Comp(s, P_0) \\
Comp(1s, P_0 \| P_1) & = & Comp(s, P_1) \\
Comp(\alpha s, P \backslash \alpha) & = & Comp(s, P) \\
Comp(s, \textit{rec } x.P) & = & Comp(s, P[\textit{rec } x.P/x])
\end{array}
$$

The following observation will prove useful later on.

Proposition 4.5

(i) $\forall P \in Proc.\ Comp(\varepsilon, P) \equiv P.$

(ii) If $P \equiv P'$ *and* $Comp(s, P)$ *is defined, then* $Comp(s, P) \equiv Comp(s, P')$.

(iii) $\forall P \in Proc.\ Comp(s_1 s_2, P) \equiv Comp(s_2, Comp(s_1, P))$.

Proof These follow in a straightforward way from the definition of *Comp*. □

Having identified the sequential components of a process term P, the next lemma shows that we can use the extra labelling information on each transition to "project" each move of P down to the actual sequential component where it occurs.

Lemma 4.6 (Decomposing transitions)

(i) $\forall s.\ \forall P.$ *If* $P \xrightarrow{\mu}_{s P_1 P_1'} P'$

then $P_1 = \sum_{i \in I} \mu_i P_i \xrightarrow{\mu_j}_{P_1 P_1'} P_j = P_1'$, *where* $j \in I$, $\mu_j = \mu$
and $Comp(s, P) \equiv P_1$, $Comp(s, P') \equiv P_1'$
and neither μ *nor* $\overline{\mu}$ *appears in* s.

(ii) $\forall s, s_0, s_1.\ \forall P.$ *If* $P \xrightarrow{\tau}_{s\langle s_0 P_0 P_0', s_1 P_1 P_1' \rangle} P'$

then $P_0 \xrightarrow{a}_{P_0 P_0'} P_0'$, $P_1 \xrightarrow{\overline{a}}_{P_1 P_1'} P_1'$, *for some* $a, \overline{a} \in Act$,
$Comp(ss_0, P) \equiv P_0$, $Comp(ss_0, P') \equiv P_0'$,
$Comp(ss_1, P) \equiv P_1$, $Comp(ss_1, P') \equiv P_1'$
and $s_0 = 0s_0'$ *and* $s_1 = 1s_1'$
and neither a *nor* $\overline{a}$ *appears in* s_0 *or* s_1.

Proof

(i) We proceed by induction on n, the length of the derivation of the transition $P \xrightarrow{\mu}_{s P_1 P_1'} P'$.

$n = 1$: Then P must be of the form $\sum_{i \in I} \mu_i P_i$ with $\mu = \mu_j$ for some $j \in I$ and the required result follows trivially.

$n > 1$: We have to consider the rule applied at the final step of the derivation of this transition.

In all cases the proof follows in a straightforward manner from the induction hypothesis and we omit the details.

(ii) We proceed by induction on $n = |s|$.

$n = 0$: We then know that the last rule applied to derive the transition $P \xrightarrow{\tau}_{\langle s_0 P_0 P_0', s_1 P_1 P_1' \rangle} P'$ which modified its label was (Com).

By the definition of (Com), it follows that $P = P_l \| P_r$ and $P' = P_l' \| P_r'$, with $P_l \xrightarrow{a}_{s_0' P_0 P_0'} P_l'$ and $P_r \xrightarrow{\overline{a}}_{s_1' P_1 P_1'} P_r'$, where $a \in Act$ and $s_0 = 0s_0'$ and $s_1 = 1s_1'$. The rest of the result then follows by appealing to (i).

After the last application of (Com), (Struct) may have been applied one or more times to obtain the actual transition. However, in applying (Struct), the label on the transition is left unchanged. Further, the new process expressions which are the source and target of the transition are structurally equivalent to the original ones, so, by Proposition 4.5 the required property continues to hold.

$n > 0$: We have to consider the cases where $s = 0s'$, $s = 1s'$ and $s = \alpha s'$ for some $\alpha \in \Lambda$. All three cases follow in a straightforward manner from the induction hypothesis. □

Just as we can project moves down from a process to its sequential components, we can identify when the moves of the sequential components can be lifted to the whole process. (This is not completely straightforward. The move of the sequential component may be forbidden in the overall process because the component lies within the scope of a restriction. This is where we crucially use the information in the labels about the nesting of restriction symbols with respect to the locations).

Lemma 4.7 (Composing transitions)

(i) $\forall s.\ \forall P.$ *If* $Comp(s, P) \equiv P_1$ *and* $P_1 = \sum_{i \in I} \mu_i P_i \xrightarrow{\mu_j}_{P_1 P_1'} P_j = P_1'$, *where* $j \in I$ *and neither* μ_j *nor* $\overline{\mu_j}$ *appears in* s
then $P \xrightarrow{\mu_j}_{s P_1 P_1'} P'$, *where* $Comp(s, P') \equiv P_1'$

(ii) $\forall s, s_0, s_1.\ \forall P.$ *If* $Comp(ss_0, P) \equiv P_0$, $Comp(ss_1, P) \equiv P_1$, *with* $P_0 \xrightarrow{a}_{P_0 P_0'} P_0'$, $P_1 \xrightarrow{\overline{a}}_{P_1 P_1'} P_1'$, *for some* $a, \overline{a} \in Act$, *where* $s_0 = 0s_0'$, $s_1 = 1s_1'$ *and neither* a *nor* $\overline{a}$ *appears in* s_0 *or* s_1.
then $P \xrightarrow{\tau}_{s\langle s_0 P_0 P_0', s_1 P_1 P_1' \rangle} P'$ *where* $Comp(ss_0, P') \equiv P_0'$ *and* $Comp(ss_1, P') \equiv P_1'$.

Proof

(i) We proceed by induction on $n = |s|$, the length of s.

$n = 0$: Then $Comp(\varepsilon, P) \equiv P_1 = \sum_{i \in I} \mu_i P_i$. Clearly $P \xrightarrow{\mu_j}_{P_1 P_1'} P'$ as required, where $Comp(\varepsilon, P') \equiv P' \equiv P_1'$.

$n > 0$: We have three cases to consider—$s = 0s'$, $s = 1s'$ and $s = \alpha s'$, for some $\alpha \in \Lambda$.

Let $s = 0s'$. Then, since $Comp(0s', P)$ is defined, P must be equivalent to an expression of the form $P_l \| P_r$. Since $Comp(0s', P) \equiv Comp(s', P_l) \equiv P_1$, by the induction hypothesis, we know that $P_l \xrightarrow{\mu_j}_{s' P_1 P_1'} P_l'$ for some P_l' such that $Comp(s', P_l') \equiv P_1'$. Applying the rule (Par) (and possibly (Struct)), it then follows that $P \equiv P_l \| P_r \xrightarrow{\mu_j}_{s P_1 P_1'} P_l' \| P_r \equiv P'$ with $Comp(s, P') \equiv P_1'$ by the definition of $Comp$.

The case $s = 1s'$ is symmetric to the case $s = 0s'$.

The last case is when $s = \alpha s'$. Then it is clear that P is equivalent to an expression of the form $P_\alpha \backslash \alpha$. Since $Comp(\alpha s', P) \equiv Comp(s', P_\alpha) \equiv P_1$, by the induction hypothesis $P_\alpha \xrightarrow{\mu_j}_{s' P_1 P_1'} P_\alpha'$ with $Comp(s', P_\alpha) \equiv P_1$ and $Comp(s', P_\alpha') \equiv P_1'$. Since we know that neither μ_j nor $\overline{\mu_j}$ appear in $\alpha s'$, we can apply (Res) (and possibly (Struct)) to obtain $P \equiv P_\alpha \backslash \alpha \xrightarrow{\mu_j}_{s P_1 P_1'} P_\alpha' \backslash \alpha \equiv P'$ with $Comp(s, P') \equiv P_1'$ by the definition of $Comp$.

(ii) By induction on $n = |s|$.

$n = 0$:

Since $s_0 = 0s_0'$ and $s_1 = 1s_1'$ and both $Comp(s_0, P)$ and $Comp(s_1, P)$ exist, it must be the case that P is equivalent to an expression of the form $P_l \| P_r$.

Then, by appealing to (i) we know that $P_l \xrightarrow{a}_{s_0' P_0 P_0'} P_l'$ and $P_r \xrightarrow{\overline{a}}_{s_1' P_1 P_1'} P_r'$, where $Comp(s_0', P_l') \equiv P_0'$ and $Comp(s_1', P_r') \equiv P_1'$. Applying (Com) (and possibly (Struct)), it then follows that $P \equiv P_l \| P_r \xrightarrow{\tau}_{\langle s_0 P_0 P_0', s_1 P_1 P_1' \rangle} P_l' \| P_r' \equiv P'$. From the definition of $Comp$ it is straightforward to check that $Comp(s_0, P') \equiv P_0'$ and $Comp(s_1, P') \equiv P_1'$ as required.

$n > 0$:

The result follows in a straightforward manner from the induction hypothesis. The proof is similar to the induction step in part (i) of this lemma and we omit the details.

□

From the way we have described the "local" behaviour of processes, it is natural to expect that the occurrence of an event e should leave sequential components lying at locations independent of $Loc(e)$ untouched. This is expressed by the following lemma.

Lemma 4.8

(i) $\forall s, s'.\ \forall P.\ P \xrightarrow{\mu}_{s P_1 P_1'} P'$ *and* $(s, s') \in I_l$ *implies that* $Comp(s', P) = Comp(s', P')$.

(ii) $\forall s, s_0, s_1, s'.\ \forall P.\ P \xrightarrow{\tau}_{s \langle s_0 P_0 P_0', s_1 P_1 P_1' \rangle} P'$ *and* $(ss_0, s') \in I_l$ *and* $(ss_1, s') \in I_l$ *implies that* $Comp(s', P) = Comp(s', P')$.

Proof

(i) By induction on $n = |s|$.

$n = 0$: Then $s = \varepsilon$ and so for any s', $(s, s') \notin I_l$ and there is nothing to prove.

$n > 0$: First, consider the case where $s = 0s_1$. Then P must be equivalent to a process of the form $P_l \| P_r$.

Then $Comp(s,P) = Comp(s_1, P_l)$. Since the transition must have come from the left side of a parallel compostion, we have $P_l \xrightarrow[s_1 P_1 P_1']{\mu} P_l'$. It is also straightforward to argue that P' must be equivalent to $P_l' \| P_r$.

For $Comp(s', P)$ to be defined, s' must either be of the form $0s_1'$ or $1s_1'$.

Suppose that $s' = 0s_1'$. Then, we must have $(s_1, s_1') \in I_l$ as well and, by the induction hypothesis, $Comp(s_1', P_l) = Comp(s_1', P_l')$. Since $Comp(s', P) = Comp(s_1', P_l)$ and $Comp(s', P') = Comp(s_1', P_l')$, the result follows.

On the other hand, if $s' = 1s_1'$, then we know that $Comp(s', P) = Comp(s_1', P_r) = Comp(s', P')$ and we are done.

The case $s = 1s_1$ is symmetric to the case $s = 0s_1$.

The final case is where $s = \alpha s_1$. Then P must be equivalent to a process of the form $P_\alpha \backslash \alpha$. It then follows that s' must also be of the form $\alpha s_1'$ in order for $Comp(s', P)$ to be defined. But then $(s_1, s_1') \in I_l$ and the result follows easily from the induction hypothesis.

(ii) Again by induction on $n = |s|$.

This time we have to analyze ss_0 and ss_1 with respect to s'. The result follows by a straightforward, though tedious argument, similar to the one used to prove part (i) of this lemma.

□

We now have enough results about the operational behaviour of a process to be able to prove the result we are after. Define the obvious transition system $TS_{CCS} = (S, i, E, I, Tran)$ where

- $S = \{[P] \mid P \in Proc\}$.
- $E = Ev$.
- $I \subseteq Ev \times Ev$ is given by Definition 4.3.
- $Tran \subseteq S \times Ev \times S = \{([P], (\mu, u), [P']) \mid P \xrightarrow[u]{\mu} P'\}$.

The only problem is that this transition system does not have an initial state i. It can be checked, however, that the notion of a region does not depend on the existence of an initial state. So, for convenience, we shall define regions over this (large) transition system without an initial state, and then prove that the axioms for elementariness hold for the reachable part of the transition system for an arbitrary choice of initial state.

In what follows, we shall, for simplicity, denote an element $[P]$ of S simply as P.

Definition 4.9 (Regions)

Let $s \in (\{0,1\} \cup \Lambda)^*$ *and* $P = \sum_{i \in I} \mu_i P_i$. *Then* $R(s,P)$ *is a pair of functions*

$R(s,P)_S : S \to \{0,1\}$ *and*

$R(s,P)_E : E \to (\{0,1\} \times \{0,1\})$ *defined as follows:*

$$\forall P' \in S.\ R(s,P)_S(P') = \begin{cases} 1 & \textit{if } Comp(s,P') \equiv P \\ 0 & \textit{otherwise} \end{cases}$$

$$\forall e = (\mu, s_1 P_1 P_1') \in E.\ R(s,P)_E(e) = (x,y),\ \textit{where :}$$

$$x = \begin{cases} 1 & \textit{if } s = s_1 \textit{ and } P \equiv P_1 \\ 0 & \textit{otherwise} \end{cases}$$

$$y = \begin{cases} 1 & \textit{if } \exists s_1'.\ s_1 s_1' = s \textit{ and } Comp(s_1', P_1') \equiv P \\ 0 & \textit{otherwise} \end{cases}$$

$$\forall e = (\tau, s'\langle s_0 P_0 P_0', s_1 P_1 P_1'\rangle) \in E.\ R(s,P)_E(e) = (x,y),\ \textit{where :}$$

$$x = \begin{cases} 1 & \textit{if } (s = s' s_0 \textit{ and } P \equiv P_0) \\ & \quad \textit{or } (s = s' s_1 \textit{ and } P \equiv P_1) \\ 0 & \textit{otherwise} \end{cases}$$

$$y = \begin{cases} 1 & \textit{if } \exists s_0'.\ s' s_0 s_0' = s \textit{ and } Comp(s_0', P_0') \equiv P \\ & \quad \textit{or } \exists s_1'.\ s' s_1 s_1' = s \textit{ and } Comp(s_1', P_1') \equiv P \\ 0 & \textit{otherwise} \end{cases}$$

So, a region $R(s,P)$ holds at a state P' iff the component of P' at location s is a sequential component equivalent to P.

For $e = (\mu, u)$, $R(s,P) \in {}^\bullet e$ iff e is located at s and e originates from a sequential component equivalent to P.

To decide when $R(s,P) \in e^\bullet$ is a little more complicated. If $e = (\mu, s_1 P_1 P_1')$, the naïve expectation is that $R(s,P) \in e^\bullet$ provided that $s = s_1$ and $P_1' \equiv P$. However P_1' need not be a sequential term, so we actually have to link e to the sequential components of P_1'. In other words, we have to check that $Comp(s_1', P_1') \equiv P$ for some s_1'. Since s_1' is the "sub-location" of the component with respect to s_1, the location where e occurs, it must in fact be the case that $s_1 s_1' = s$ in order that $R(s,P) \in e^\bullet$.

Lemma 4.10 $\forall s.\ \forall P = \sum_{i \in I} \mu_i P_i.\ R(s,P)$ *is a region of the (pseudo)-transition system* TS_{CCS}.

Proof We have to establish three facts:

(i) $\forall R(s,P).\ \forall e, e' \in E.$ If $(e,e') \in I$ then $R(s,P) \in ({}^\bullet e \cup e^\bullet) \Rightarrow R(s,P) \notin ({}^\bullet e' \cup e'^\bullet)$.

(ii) $\forall R(s,P).\ \forall (P_1, (\mu,u), P_1') \in Tran.\ R(s,P) \in {}^\bullet(\mu,u)$ implies $R(s,P)(P_1) = 1$.

(iii) $\forall R(s,P).\ \forall (P_1, (\mu,u), P_1') \in Tran.\ R(s,P)(P_1') = R(s,P)(P_1) + y - x$, where $R(s,P)(\mu,u) = (x,y)$.

(i) follows in a straightforward manner from the definition of the independence relation I and the definition of regions.

(ii) follows directly from Lemma 4.6 and the definition of regions.

To establish (iii) we have to consider the various possibilities. First, assume that the event e is of the form $(\mu, s_2 P_2 P_2')$. In other words, we are looking at a transition $(P_1, (\mu, s_2 P_2 P_2'), P_1') \in Tran$. Consider an arbitrary region $r = R(s,P)$.

(a) Suppose that $r \in {}^\bullet e$ and $r \notin e^\bullet$. Since $r \in {}^\bullet e$, we know that $s = s_2$ and $P \equiv P_2$. Since $r \notin e^\bullet$, we know that $Comp(\varepsilon, P_2') \not\equiv P$ (since $s_2\varepsilon = s$). By Lemma 4.6, we know that $Comp(s, P_1) = Comp(s_2, P_1) \equiv P_2 \equiv P$ and so $R(s,P)(P_1) = 1$. On the other hand, again by Lemma 4.6, $Comp(s_2, P_1') \equiv P_2'$, which implies that $Comp(s, P_1') = Comp(s_2, P_1') \not\equiv P$. So $R(s,P)(P_1') = 0$ and we are done.

(b) Next, suppose that $r \notin {}^\bullet e$ and $r \in e^\bullet$. Since $r \in e^\bullet$, we know that there exists s_2' such that $s_2 s_2' = s$ and $Comp(s_2', P_2') \equiv P$. To find out whether $Comp(s, P_1') \equiv P$, we can rewrite this as $Comp(s_2 s_2', P_1')$. This is equivalent to $Comp(s_2', Comp(s_2, P_1'))$. But, by Lemma 4.6, we know that $Comp(s_2, P_1') \equiv P_2'$ and so we have $Comp(s, P_1') \equiv Comp(s_2', P_2') \equiv P$. So $R(s,P)(P_1') = 1$.

On the other hand, since $r \notin {}^\bullet e$, we know that either $s \neq s_2$ or $s = s_2$ and $P_2 \not\equiv P$. Suppose $s = s_2$. Then $Comp(s, P_1) \equiv P_2$ (by Lemma 4.6) and, since $P_2 \not\equiv P$, we have $R(s,P)(P_1) = 0$. On the other hand, if $s \neq s_2$, we know from the fact that $r \in e^\bullet$ that $s_2 \preceq s$—i.e. there exists s_2' such that $s_2 s_2' = s$. But we know from Lemma 4.6 that $Comp(s_2, P_1) \equiv P_2$ and, furthermore, $P_2 = \sum_i \mu_i P_i$. Hence $Comp(s, P_1) \equiv Comp(s_2', Comp(s_2, P_1)) \equiv Comp(s_2', P_2)$ must be undefined, because $s_2' \neq \varepsilon$. Thus $Comp(s, P_1) \not\equiv P$ and so $R(s,P)(P_1) = 0$.

(c) Next, suppose that $r \in {}^\bullet e$ and $r \in e^\bullet$. Then, from arguments similar to the ones used for (a) and (b), it follows that $R(s,P)(P_1) = R(s,P)(P_1') = 1$.

(d) Finally, suppose that $r \notin {}^\bullet e$ and $r \notin e^\bullet$.

If $s = s_2$, by Lemma 4.6, it is easy to see that $Comp(s, P_1) = Comp(s_2, P_1) \equiv P_2$. So $R(s,P)(P_1) = 1$ iff $P \equiv P_2$ iff $R(s,P) \in {}^\bullet e$. So, we must have $R(s,P)(P_1) = 0$. Similarly, we can argue that $R(s,P)(P_1') = 0$.

Next consider the case when $s \neq s_2$.

Suppose that $(s, s_2) \in I_l$. Then, by Lemma 4.8, it follows that $R(s,P)(P_1) = R(s,P)(P_1')$ and we are done.

On the other hand, suppose that $s \preceq s_2$. Then there exists s' such that $ss' = s_2$. We know, by Lemma 4.6, that $Comp(s_2, P_1) \equiv P_2$. So, if $Comp(s, P_1) \equiv P$, where $P = \sum_i \mu_i P_i$, then we would have $Comp(s_2, P_1) \equiv Comp(s', Comp(s, P_1)) \equiv Comp(s', P)$ which is undefined, since $s' \neq \varepsilon$. This is a contradiction, so we cannot have $Comp(s, P_1) \equiv P$ and therefore $R(s,P)(P_1) = 0$. By a similar argument, it then follows that $R(s,P)(P_1') = 0$ as well.

The final situation is when $s_2 \preceq s$. Then there exists s_2' such that $s_2 s_2' = s$. We know that $Comp(s, P_1) \equiv Comp(s_2', Comp(s_2, P_1)) \equiv Comp(s_2', P_2)$ (by Lemma 4.6.) But, $P_2 = \sum_i \mu_i P_i$, so $Comp(s_2', P_2)$ must be undefined for $s_2' \neq \varepsilon$ and hence $R(s,P)(P_1) = 0$. To show that $R(s,P)(P_1') = 0$, notice that $Comp(s_2', P_2') \not\equiv P$ because $R(s,P) \notin e^\bullet$. But $Comp(s, P_1') = Comp(s_2 s_2', P_1') \equiv Comp(s_2', Comp(s_2, P_1')) \equiv Comp(s_2', P_2')$, by Lemma 4.6. So $Comp(s, P_1') \not\equiv P$ and $R(s,P)(P_1') = 0$.

We have to do a similar analysis in case e corresponds to a synchronization of the form $(\tau, s'\langle s_l P_l P_l', s_r P_r P_r'\rangle)$. The details are essentially the same and we omit them.

□

We now define precisely the asynchronous transition system which we wish to prove elementary.

Let $\hat{P}$ be any process expression. Then $LTS(\hat{P}) = ((S_{\hat{P}}, [\hat{P}], E_{\hat{P}}, I_{\hat{P}}, Tran_{\hat{P}}), l_{\hat{P}})$ where

- $S_{\hat{P}} = \{[P'] \mid P' \in Proc \text{ and } P' \text{ is reachable from } \hat{P} \text{ in } TS_{CCS}\}$. $[\hat{P}]$ *is the initial state.*
- $E_{\hat{P}} = \{(\mu, u) \in Ev \mid \exists [P'], [P''] \in S_{\hat{P}}.\ P' \xrightarrow[u]{\mu} P''\}$.
- $I_{\hat{P}} = I \cap (E_{\hat{P}} \times E_{\hat{P}})$, *where* I *is the relation defined in Definition 4.3.*
- $Tran_{\hat{P}} \subseteq S_{\hat{P}} \times E_{\hat{P}} \times S_{\hat{P}} = \{([P], (\mu, u), [P']) \mid P \xrightarrow[u]{\mu} P'\}$
- $l_{\hat{P}} : E_{\hat{P}} \to Act_\tau$ is given by $l_{\hat{P}}(\mu, u) = \mu$.

It is straightforward to verify that for each $\hat{P}$, $LTS(\hat{P})$ is in fact a labelled asynchronous transition system. What we need to show is that it is elementary.

In what follows, fix a process expression $\hat{P}$ and the corresponding transition system $LTS(\hat{P})$. For simplicity, we shall refer to the states $[P]$ of $LTS(\hat{P})$ simply as P.

We first have the following simple proposition, which we state without a proof.

Proposition 4.11 *Let $R(s, P)$ be a region of TS_{CCS} as defined above. Then $R(s, P)$ is a (possibly trivial) region of $LTS(\hat{P})$.*

To establish that the asynchronous transition system $LTS(\hat{P})$ that we have defined is elementary, we have to establish that the two regional axioms hold with respect to regions which are non-trivial when restricted to $LTS(\hat{P})$.

Lemma 4.12 (Separation by regions)
Let $LTS(\hat{P}) = ((S_{\hat{P}}, [\hat{P}], E_{\hat{P}}, I_{\hat{P}}, Tran_{\hat{P}}), l_{\hat{P}})$. $\forall P_1, P_2 \in S_{\hat{P}}$. $P_1 \not\equiv P_2$ implies that there exist s and P such that $R(s, P)$ is a non-trivial region *of $LTS(\hat{P})$ and either $P_1 \in R(s, P)$ and $P_2 \notin R(s, P)$ or $P_1 \notin R(s, P)$ and $P_2 \in R(s, P)$.*

Proof It is fairly obvious from our definition of *Comp* that if $P_1 \not\equiv P_2$ then there is some s for which $Comp(s, P_1) \not\equiv Comp(s, P_2)$. So there is no problem finding a region $R(s, P)$ which separates P_1 and P_2.

What we have to argue is that the region separating P_1 and P_2 is non-trivial. However, notice that the previous lemma about regions applies to *all* regions. Consider any trivial region $R(s', P')$. If $R(s', P')$ holds somewhere in $LTS(\hat{P})$, $R(s', P')$ can cease to hold only by the occurrence of some event e such that $R(s', P') \in {}^\bullet e$. Similarly, $R(s', P')$ could have begun to hold only after the occurrence of some event e such that $R(s', P') \in e^\bullet$. Since $R(s', P')$ was assumed to be trivial, it therefore follows that it either holds for all states in $S_{\hat{P}}$ or fails to hold for all states in $S_{\hat{P}}$.

In other words, if $Comp(s', P_1) \equiv P'$, but $R(s', P')$ is a trivial region, we know that $Comp(s', P_2) \equiv P'$ as well and so this is *not* the component on which P_1 and P_2 differ. Thus, any component on which P_1 and P_2 differ must correspond to a non-trivial region, and we are done. □

Lemma 4.13 (Enabling) *Let* $LTS(\hat{P}) = ((S_{\hat{P}}, [\hat{P}], E_{\hat{P}}, I_{\hat{P}}, Tran_{\hat{P}}), l_{\hat{P}})$.
$\forall P \in S_{\hat{P}}.\ \forall (\mu, u) \in E_{\hat{P}}$. *If there does not exist a* P' *such that* $(P, (\mu, u), P') \in Tran_{\hat{P}}$, *then there is some* non-trivial *region* $R(s', P') \in {}^\bullet(\mu, u)$ *such that* $P \notin R(s', P')$.

Proof First consider the case where $e = (\mu, u)$ is of the form $(\mu, s_1 P_1 P_1')$. Then, since e must occur somewhere in the transition system, we know that there exists P_2 and P_2' such that $P_2 \xrightarrow[s_1 P_1 P_1']{\mu} P_2'$. So, by the definition of a region, $R(s_1, P_1) \in {}^\bullet e$. Further, by Lemma 4.6, neither μ nor $\overline{\mu}$ appears in s_1.

Now, if $P \in R(s_1, P_1)$, then, by Lemma 4.7, there must exist a P' such that $P \xrightarrow[s_1 P_1 P_1']{\mu} P'$. Since we have assumed that there is no $\xrightarrow[s_1 P_1 P_1']{\mu}$ move leading out of P, we can conclude that $P \notin R(s_1, P_1)$ and we are done.

The situation when e is of the form $(\tau, s\langle s_0 P_0 P_0', s_1 P_1 P_1' \rangle)$ is handled similarly, appealing to part (ii) of Lemmas 4.6 and 4.7.

□

Theorem 4.14 $\forall P \in Proc.\ LTS(P)$ *is an elementary asynchronous transition system.*

5 Relationships to other approaches

We now show how the operational semantics we have provided for this language relates to two other approaches for providing a semantics for CCS.

The first connection we draw is with respect to the standard interleaving transition system for CCS, as defined by Milner in [9]. We "project" down from the asynchronous transition system $LTS(P)$ we assign to a term P to obtain a corresponding labelled sequential transition system $TS(P)$ by forgetting the extra information about the events. We then exhibit a strong relationship between $TS(P)$ and the transition system for P which would be generated by the semantics in [9].

The other comparison is with respect to the way Winskel and Nielsen define a denotational semantics for a process language in [13]. They provide a semantics in the category of asynchronous transition systems, where the operators of the process language are interpreted as appropriate categorical constructions. We will sketch their approach briefly, without going into too much technical detail, and then describe how the asynchronous transition systems they assign denotationally compare to the asynchronous transition systems we assign operationally.

We begin by relating our semantics to the standard interleaving semantics for CCS. Recall the definition of $LTS(P)$ for a term P. $LTS(P) = ((S_P, [P], E_P, I_P, Tran_P), l_P)$ where

- $S_P = \{[P'] \mid P' \in Proc$ *and* P' *is reachable from* P *in* $TS_{CCS}\}$. $[P]$ *is the initial state.*
- $E_P = \{(\mu, u) \in Ev \mid \exists [P'], [P''] \in S_P.\ P' \xrightarrow[u]{\mu} P''\}$.
- $I_P = I \cap (E_P \times E_P)$, *where* I *is the relation defined in Definition 4.3.*

- $Tran_P \subseteq S_P \times E_P \times S_P = \{([P], (\mu, u), [P']) \mid P \xrightarrow[u]{\mu} P'\}$.

- $l_P : E_P \to Act_\tau$ is given by $l_P(\mu, u) = \mu$.

We extract a labelled (sequential) transition system $TS(P)$ from $LTS(P)$ in the obvious way, by forgetting the extra information in the labels of the events.

Definition 5.1 *Let* $LTS(P) = ((S_P, [P], E_P, I_P, Tran_P), l_P)$. *Then,* $TS_P = (S'_P, [P], E'_P, Tran'_P)$ *where*

- $S'_P = S_P$, *with* $[P]$ *as the initial state.*

- $E'_P = \{\mu \mid (\mu, u) \in E_P\}$.

- $Tran'_P \subseteq S'_P \times E'_P \times S'_P = \{([P], \mu, [P']) \mid ([P], (\mu, u), [P']) \in Tran_P\}$.

Let $Seq(T)$ be the standard sequential transition system for a process term $P \in Proc$. Essentially, this is the transition system obtained using our rules (Sum), (Par), (Com) and (Res), ignoring the labels "below the arrow", and adding in place of (Struct) the following rule.

$$P[rec\ x.P/x] \xrightarrow{\mu} P' \quad implies \quad rec\ x.P \xrightarrow{\mu} P' \qquad \textbf{(Rec)}$$

To relate these two approaches formally, we define a folding map on transition systems as follows.

Definition 5.2 (Foldings) *Let* $TS_k = (S_k, i_k, E_k, Tran_k), k = 1, 2$, *be two labelled sequential transition systems. Then, a* folding *from* TS_1 *onto* TS_2 *a pair of functions* $f = (f_S, f_E)$ *where*
$f_S : S_1 \to S_2$ *and*
$f_E : E_1 \to E_2$ *such that:*

(i) f_S *is onto, with* $f_S(i_1) = i_2$.

(ii) $\forall (s_1, e_1, s'_1) \in Tran_1.\ (f_S(s_1), f_E(e_1), f_S(s'_1)) \in Tran_2$.

(iii) $\forall (f_S(s_1), e_2, s'_2) \in Tran_2.\ \exists (s_1, e_1, s'_1) \in Tran_1$ *such that* $f_E(e_1) = e_2$ *and* $f_S(s'_1) = s'_2$.

A similar notion has been defined in [6] where it is called a transition preserving homomorphism.

If $f_E = id$, the identity function, then a folding corresponds to a special type of a bisimulation, where the second system is, in general, a smaller, more compact representation of the behaviour described by the first system.

Theorem 5.3 *There is a folding from* $Seq(P)$ *onto* $TS(P)$ *whose map on the events is the identity and whose map on states takes a process term* P *to its equivalence class* $[P]$.

Proof The proof follows by induction on the structure of P and we omit the details. □

So, we have shown that the asynchronous transition system we generate is, in some sense, a "practical" transition system. It is finite whenever the normal interleaved transition system is finite and has, in fact, slightly fewer states in general.

However, as we have pointed out in Section 3, the asynchronous transition system we associate with a process $P \in Proc$ could have more transitions than the standard interleaved transtion system for P. For instance, we have shown that $P = rec\ x.ax \| rec\ x.ax$ would generate two a-labelled transitions in our set up whereas it would have only one a-labelled transition in the standard approach. However, when we project these two a transitions down to $TS(P)$, we only get a single a transition because sequential transition systems are extensional—there cannot be two different transitions with the same label connecting the same pair of states. It is not difficult to see, though, that the set of transitions that we add at any state in going from $Seq(P)$ to $LTS(P)$ is always finite.

This is also a good place to discuss why we identify states in our transition system as equivalence classes $[P]$ rather than just process expressions P. The natural way to work directly with process expressions as states in our framework would be to extend the rule (Rec) with extra labels on the transitions as follows.

$$P[rec\ x.P/x] \xrightarrow[u]{\mu} P' \quad implies \quad rec\ x.P \xrightarrow[u]{\mu} P' \qquad (\mathbf{Rec'})$$

However, since the moves of $rec\ x.P$ are then *exactly* the same as those of $P[rec\ x.P/x]$, the asynchronous transition we end up with is no longer elementary, in general. For instance, the transition system corresponding to the term $a\ rec\ x.ax$ would have two states, $a\ rec\ x.ax$ and $rec\ x.ax$. There would be a transition from $a\ rec\ x.ax$ to $rec\ x.ax$ via the event $(a, [a\ rec\ x.ax][rec\ x.ax])$ and a transition via the *same* event looping from the state $rec\ x.ax$ back to itself. It is not difficult to show that this does not correspond to an elementary asynchronous transition system.

Next, we describe, somewhat informally, the approach taken by Winskel and Nielsen in [13] to provide a denotational semantics for CCS-like languages in terms of asynchronous transition systems.

Let the "basic" operators which are used to build up process terms be prefixing (aP), choice ($+$), parallel composition ($\|$), restriction ($\backslash\alpha$) and recursion ($rec\ x.P$).

Assuming inductively that we have built up an asynchronous transition system $Den(P)$ denoting a term P, the transition system corresponding to aP is obtained by adjoining a new initial state to $Den(P)$ and adding a new event labelled a which connects the new initial state to the initial state of $Den(P)$.

$P_1 \| P_2$ is modelled by a version the categorical product of $Den(P_1)$ and $Den(P_2)$. This produces a transition system which is essentially the same as the one our operational semantics produces for $P_1 \| P_2$.

Restriction is also handled directly in a categorical framework, using cartesian liftings and fibrations, which essentially achieve the same effect as one would expect intuitively.

The first major difference between the denotational approach and our operational appoach arises in the treatment of $+$. The denotational transition system corresponding to the term $P_1 + P_2$ is obtained by taking the categorical coproduct of $Den(P_1)$ and $Den(P_2)$. This operation essentially consists of taking disjoint copies of $Den(P_1)$ and $Den(P_2)$ and

fusing together their initial states. This means that the denotation of the term $a\ nil + a\ nil$ would be a transition system with two distinct a-labelled transitions leading from the initial state. On the other hand, our operational semantics would generate only a single event for this process. Thus, in general, the denotational treatment of $+$ would give rise to "wider" transition systems than our operational semantics.

The other major difference is in the treatment of recursion. In the denotational approach, a term of the form $rec\ x.P$ is *always* completely unfolded. Thus any process of this form gives rise to an infinite transition system. On the other hand, terms of the form $rec\ x.ax$ would produce finite cyclic transition systems according to our operational semantics.

Thus, the denotation of a term P would, in general, be an "unfolded" version of the transition system that our operational semantics would generate. The "unfolding" would be both "horizontal" (because of the treatment of $+$) and "vertical" (because of the treatment of recursion).

To relate these two approaches, we extend our definition of a folding to asynchronous transition systems as follows.

Definition 5.4 (Strong foldings) *Let (ATS_1, l_1) and (ATS_2, l_2) be two Σ-labelled transition systems, where $ATS_k = (S_k, i_k, E_k, I_k, Tran_k), k = 1, 2$. Then, a* strong folding *from (ATS_1, l_1) onto (ATS_2, l_2) is a pair of functions $f = (f_S, f_E)$ where*

$f_S : S_1 \to S_2$ *and*

$f_E : E_1 \to E_2$ *such that:*

(i) *f is a folding from $TS_1 = (S_1, i_1, E_1, Tran_1)$ onto $TS_2 = (S_2, i_2, E_2, Tran_2)$.*

(ii) *$\forall e_1 \in E_1.\ \forall e_2 \in E_2.\ e_2 = f_E(e_1)$ implies $l_2(e_2) = l_1(e_1)$.*

(iii) *$\forall e_1, e_1' \in E_1.\ (e_1, e_1') \in I_1$ implies $(f_E(e_1), f_E(e_1')) \in I_2$.*

(iv) *$\forall (f_S(s_1), e_2', s_2'), (f_S(s_1), e_2'', s_2'') \in Tran_2.\ (e_2', e_2'') \in I_2$ implies $\exists (s_1, e_1', s_1'), (s_1, e_1'', s_1'') \in Tran_1$ such that $f_E(e_1') = e_2'$, $f_S(s_1') = s_2'$, $f_E(e_1'') = e_2''$, $f_S(s_1'') = s_2''$ and $(e_1', e_1'') \in I_1$.*

The extra requirements on a strong folding are that the map on the underlying events preserve labels and the independence relation. The last clause adds the requirement that concurrent steps in the second system be pulled back to concurrent steps in the first system. This is a bit weaker than saying that every pair of independent events in the second system has a pair of independent events in the first system as its pre-image via f_E, because we could have "unused" independences in the second system which never actually give rise to a concurrent step. (For instance, in the term $a(b\|c) + d(e\|f)$, the event labelled b and the event labelled f would be independent by our operational semantics, though there is no state where they are simultaneously enabled and, in fact, no run where they both occur).

For $P \in Proc$, let $Den(P)$ be the denotational transition system corresponding to a term (as defined in [13]) and let $LTS(P)$ be the asynchronous transition system corresponding to P defined in the previous section. We then have the following result.

Theorem 5.5 *There exists a folding from $Den(P)$ onto $LTS(P)$.*

Proof We shall not go into the details, because it will also involve describing the denotational semantics more precisely. The proof is fairly straightforward, by induction on the structure of the term P. □

6 Bisimulations on asynchronous transition systems

We now examine the question of how to define a sensible notion of bisimulation on labelled asynchronous transition systems which respects the independence relation on the underlying events.

As we had mentioned earlier, asynchronous transition systems can be equipped with a natural notion of morphism [13]. These morphisms map states and events in such a way that independence is preserved globally.

Though transition system morphisms preserve behaviour, they appear to be unsatisfactory for defining a natural notion of bisimulation. The main problem is the stipulation that independence must be preserved globally.

Intuitively, a bisimulation describes how systems match each other's behaviour along individual runs. In the conventional framework, the existence of a bisimulation between two systems ensures that the branching behaviour of each system can be faithfully simulated by the other along each run. When extending this to asynchronous transition systems, it is natural to further require that the independence relation be preserved by the bisimulation along each run, but *not necessarily globally*.

To illustrate this point, consider the two CCS expressions $(b + \alpha \| \overline{\alpha} b) \backslash \alpha$ and $b + \tau b$. In the first process, the two b moves would be considered independent in our set up, because they appear on different sides of the $\|$ operator. However, it is easy to see that in any run of the first process, exactly one of the two b moves will occur. So, it seems reasonable to expect that these two processes should be bisimilar, though the second process has *no* independent events.

Thus one needs a way of relating two systems along each run. Unfortunately, this means that it no longer suffices to present the bisimulation in terms of a relation on states—we have also to "remember" how we reached the state. In particular, we need to remember the independences we have observed so far. This ensures that in extending the run from that state, we can remain consistent with the choices already made while relating events along this run.

One way to formalize this intuition is to follow the approach Aceto uses to characterize a static version of location equivalence [1].

For the next few definitions, fix a pair of labelled asynchronous transition systems (ATS_1, l_1) and (ATS_2, l_2), where $ATS_k = (S_k, i_k, E_k, I_k, Tran_k)$, $k = 1, 2$.

Definition 6.1 *A relation $\varphi \subseteq E_1 \times E_2$ is lI-*consistent *provided*

- $\forall (e_1, e_2) \in \varphi.\ l_1(e_1) = l_2(e_2)$.
- $\forall e_1, e_1' \in E_1.\ \forall e_2, e_2' \in E_2.\ (e_1, e_2) \in \varphi$ *and* $(e_1', e_2') \in \varphi \Rightarrow e_1 I_1 e_1'$ *iff* $e_2 I_2 e_2'$.

The first condition says that the labels on the related events must match, whereas the second condition says that the independence relation must be preserved by φ in a strong way. In other words, φ is consistent with respect to both l and I.

Let Φ denote the family of all lI-consistent relations $\varphi \subseteq E_1 \times E_2$.

We now wish to define a notion of equivalence based on these lI-consistent relations between events. The idea is straightforward—as we match events from the two systems along a given run, we have to ensure at each stage that the events we have related constitute an lI-consistent relation. To make this a bisimulation, we have to further ensure that at each stage the choices available to each process can be matched by the other process in such a way that the current lI-consistent relation extends to a larger one.

To achieve this, we associate with each $\varphi \in \Phi$, a relation $\sim_\varphi \subseteq S_1 \times S_2$. $s_1 \sim_\varphi s_2$ is to be read as follows; if we reach s_1 in TS_1 and s_2 in TS_2 during a simulation along which we have associated events by φ, then it is possible to extend the simulation along all possible choices of transitions at s_1 and s_2 in a manner consistent with φ.

Formally, we have the following definition:

Definition 6.2 $\sim_\Phi = \{\sim_\varphi | \ \varphi \in \Phi\}$ *is the largest* Φ*-indexed family of symmetric relations on* $S_1 \times S_2$ *satisfying:*

If $s_1 \sim_\varphi s_2$ *then* $\forall (s_1, e_1, s_1') \in Tran_1.\ \exists (s_2, e_2, s_2') \in Tran_2$ *such that* $\varphi \cup \{(e_1, e_2)\} \in \Phi$ *and* $s_1' \sim_{\varphi \cup \{(e_1,e_2)\}} s_2'$.

Two labelled asynchronous transition systems are bisimilar if their initial states are related by the empty relation between events.

Definition 6.3 $ATS_1 \sim ATS_2$ *iff* $i_1 \sim_\emptyset i_2$.

It is easy to verify the following.

Proposition 6.4 $\sim$ *is an equivalence relation on labelled asynchronous transition systems.*

Example 6.1 Let $P_1 = (b + \alpha \| \overline{\alpha} b) \backslash \alpha$ and $P_2 = b + \tau b$. Then $LTS(P_1) \sim LTS(P_2)$, as shown below.

$LTS(P_1)$ has three events, $e_1 = (b, \alpha 0[b + \alpha][nil])$, $e_2 = (\tau, \alpha\langle 0[b + \alpha][nil], 1[\overline{\alpha} b][b]\rangle)$, and $e_3 = (b, \alpha 1[b][nil])$.

$LTS(P_2)$ also has three events, $e_1' = (b, [b + \tau b][nil])$, $e_2' = (\tau, [b + \tau b][b])$, and $e_3' = (b, [b][nil])$.

We can define

- $\sim_\emptyset = \{(P_1, P_2), (P_2, P_1)\}$.
- $\sim_{\{(e_1,e_1')\}} = \{((nil \| \overline{\alpha} b) \backslash \alpha, nil), (nil, (nil \| \overline{\alpha} b) \backslash \alpha)\}$.
- $\sim_{\{(e_2,e_2')\}} = \{((nil \| b) \backslash \alpha, b), (b, (nil \| b) \backslash \alpha)\}$.
- $\sim_{\{(e_2,e_2'),(e_3,e_3')\}} = \{((nil \| nil) \backslash \alpha, nil), (nil, (nil \| nil) \backslash \alpha)\}$.

Example 6.2 Let $P_1 = (a\alpha c \| b\overline{\alpha} d) \backslash \alpha$ and $P_2 = (a\alpha d \| b\overline{\alpha} c) \backslash \alpha$. Then $LTS(P_1) \not\sim LTS(P_2)$.

Both processes are deterministic, so they both have only one possible run. Since each event has a unique label in each process, along this run we will have to relate, for instance,

the a and c labelled events in P_1 to the corresponding a and c events in P_2. However, these events are *not* independent in P_1 whereas they *are* independent in P_2. So, there exists no lI-consistent relation corresponding to the (unique) maximal run of P_1 and P_2.

The equivalence we have described applies in general to all labelled asynchronous transition systems. When restricted to the transition systems we construct for CCS terms, it amounts to defining a notion of strong bisimulation over terms in *Proc*. We can extend the definition smoothly to deal with weak bisimulation over CCS terms as follows.

First, given an Act_τ-labelled asynchronous transition system (ATS, l), where $ATS = (S, i, E, I, Tran)$, define the weak transistion relation $\Rightarrow \subseteq S \times E \times S$ in the obvious way.

$$\begin{aligned} \Rightarrow = \{(s, e, s') \mid\ & \exists s_0, s_1, \ldots, s_n.\ \exists e_0, e_1, \ldots, e_{n-1}.\ s = s_0, s' = s_n \\ & \textit{and}\ \forall 0 \leq i < n.\ (s_i, e_i, s_{i+1}) \\ & \textit{such that}\ e = e_j, 0 \leq j < n,\ \textit{where}\ l(e) \neq \tau \\ & \textit{and}\ \forall 0 \leq i < n.\ i \neq j \Rightarrow l(e_i) = \tau\} \end{aligned}$$

We need to extend $\Rightarrow$ to describe purely internal transitions between states. In this framework, it is convenient to construct a second relation $\rightsquigarrow \subseteq S \times S$ such that

$$\begin{aligned} \rightsquigarrow = \{(s, s') \mid s = s'\ \textit{or}\ & \exists s_0, s_1, \ldots, s_n.\ \exists e_0, e_1, \ldots, e_{n-1}.\ s = s_0, s' = s_n \\ & \textit{and}\ \forall 0 \leq i < n.\ (s_i, e_i, s_{i+1}),\ \textit{where}\ l(e_i) = \tau\} \end{aligned}$$

Now consider a pair of Act_τ-labelled asynchronous transition systems (ATS_1, l_1) and (ATS_2, l_2), where $ATS_k = (S_k, i_k, E_k, I_k, Tran_k), k = 1, 2$, with weak transition relations $\Rightarrow_k$ and $\rightsquigarrow_k$, $k = 1, 2$ respectively.

We retain the notion of an lI-consistent relation as before. We can now define a weak notion of equivalence $\simeq$ between transition systems. We first begin with the Φ-indexed versions of $\simeq$.

Definition 6.5 $\simeq_\Phi = \{\simeq_\varphi \mid \varphi \in \Phi\}$ *is the largest* Φ*-indexed family of symmetric relations on* $S_1 \times S_2$ *satisfying:*

If $s_1 \simeq_\varphi s_2$ *then*

- $\forall (s_1, e_1, s_1') \in \Rightarrow_1\ .\ \exists (s_2, e_2, s_2') \in \Rightarrow_2$ *such that* $\varphi \cup \{(e_1, e_2)\} \in \Phi$ *and* $s_1' \simeq_{\varphi \cup \{(e_1, e_2)\}} s_2'$.
- $\forall (s_1, s_1') \in \rightsquigarrow_1\ .\ \exists (s_2, s_2') \in \rightsquigarrow_2$ *such that* $s_1' \simeq_\varphi s_2'$.

Once again we say that $ATS_1 \simeq ATS_2$ iff $i_1 \simeq_\emptyset i_2$. It can be verified that $\simeq$ is an equivalence relation on Act_τ-labelled asynchronous transition systems.

The induced weak equivalence $\simeq$ on process terms is closely related to the notion of location equivalence defined by Boudol et al [5]. Aceto [1] has provided an alternative characterization of this equivalence for a sublanguage where processes are viewed as "networks" of sequential components—i.e. parallel composition is restricted to the top level.

It is not hard to see that by restricting our language to a language like the one Aceto considers, our definition of $\simeq$ coincides with his notion of location equivalence, and hence, with the notion of Boudol et al. In fact, we believe that this is true for the entire language we consider. Let $\approx_l$ denote the location equivalence of [5].

Conjecture $\forall P, P' \in Proc.\ LTS(P) \simeq LTS(P')$ *iff* $P \approx_l P'$.

7 Discussion

In this paper, we have described how to provide a semantics for CCS in terms of elementary asynchronous transition systems. By appealing to the coreflection between this class of transition systems and 1-safe Petri nets established in [13], we obtain as a corollary a Petri net semantics for the language we consider.

Admittedly, the language we consider is not full CCS. However, as we have already mentioned, we believe that the language studied here is a powerful and useful subset of CCS which is sufficient for specifying most concurrent systems of interest.

As we had pointed out in the Introduction, many other people have provided non-interleaved semantics for CCS [3, 6, 8, 11]. Our claim is that our semantics is simpler and more natural than those described elsewhere. To a large part, this is because all these approaches deal with full CCS, whereas we avoid having to deal with processes of the form $P + (Q \| R)$, which are the main source of complications for these approaches. However, we feel that the benefits that we obtain by restricting the syntax more than justify the choice we have made.

For one, we use a very straightforward extension of the standard operational semantics. To actually read off the events of the transition system and the independence relation on the events from our operational semantics is trivial. Having proved here once and for all that the resulting asynchronous transition system is elementary, we can be sure that we are working with a "nice" object. So, for instance, feeding our operational description of a term into a verification tool should be no more difficult than feeding the standard interleaved description of the term.

In contrast, using "proved transitions" [3] or working with an algebra of transitions [8] always requires a second level of reasoning about the transitions to recover the underlying events of the system.

In the approaches which directly yield a Petri net semantics [3, 6, 11], for every term P one has to first construct a "concrete" implementation of a net describing the behaviour of P and then work back to recover the global states, because one typically needs to reason about the global states of the system. Instead, we directly provide the global states and, through the independence relation, provide a means of recovering the local states if they are required.

Another interesting feature of the semantics we describe here is that the independence relation on events directly reflects the idea of events occurring at independent locations. This seems to be a very natural way to think about independence. We feel that it would be difficult to extend this idea in a straightforward way to deal with the full language.

We also believe that our result establishing that the normal interleaved transition system for a term can be folded onto our asynchronous transition system is quite valuable. This means that our approach yields a tractable system whenever the conventional approach would and, once again, has a bearing on the possibilities of mechanically verifying properties of such systems. (In [6], a similar result is proved, but in the opposite direction—i.e. they show that the non-interleaved transition system they define can be folded into the standard interleaved one).

In this paper, we have also introduced a notion of bisimulation over labelled asyn-

chronous transition systems which preserves independence in a fairly natural way. This permits us to equip our language with a simple notion of equivalence which respects the non-interleaved nature of our semantics and yet abstracts away from the concrete syntax. This is a feature which has been lacking in earlier approaches to providing a non-interleaved semantics for CCS.

It is clear that there is a close connection between the equivalence we define and the location equivalence of [5]. A problem with the equivalence defined in [5] is that it is based on a transition system which is infinitely branching, even for the simplest of process terms. If our conjecture that the two equivalences coincide is true, then our definition would provide an effective way of checking location equivalence for a very large class of CCS processes.

An issue which is yet to be resolved is how best to define bisimulations over asynchronous transition systems. We believe that our approach reflects the right intuition. However, we would be happier with a more "global" definition of how to relate two systems, rather than the incremental definition we have provided here, which makes is rather clumsy to actually present a bisimulation (as in Example 6.1).

Acknowledgment
We have benefited greatly from many discussions with P.S. Thiagarajan.

References

[1] L. Aceto: A static view of localities, Report 1483, INRIA, Sophia-Antipolis (1991).

[2] M.A. Bednarczyk: Categories of asynchronous systems, PhD Thesis, Report 1/88, Computer Science, University of Sussex (1988).

[3] G. Boudol, I. Castellani: Three equivalent semantics for CCS, *Semantics of Systems of Concurrent Processes*, *Springer LNCS 469*, 96–141 (1990).

[4] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn: Observing localities, *Proc. MFCS '91*, *Springer LNCS 520*, 93–102 (1991) (full version appears as Report 4/91, Computer Science, University of Sussex (1991)).

[5] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn: A theory of processes with localities, Report 1632, INRIA, Sophia-Antipolis (1992) (an earlier version appears as Report 13/91, Computer Science, University of Sussex (1991)).

[6] P. Degano, R. de Nicola, U Montanari: A distributed operational semantics for CCS based on Condition/Event systems, *Acta Informatica*, **26**, 59–91 (1988).

[7] A. Ehrenfeucht, G. Rozenberg: Partial 2-structures; Part II: State spaces of concurrent systems, *Acta Informatica*, **27**, 348–368 (1990).

[8] G.L. Ferrari, U. Montanari: Towards the unification of models for concurrency, *Proc. CAAP '90*, *Springer LNCS 431*, 162–176 (1990)

[9] R. Milner: *Communication and Concurrency*, Prentice-Hall, London (1989).

[10] M. Nielsen, G. Rozenberg, P.S. Thiagarajan: Elementary transition systems, *Theoretical Computer Science*, **96**, 1, 3–33 (1992).

[11] E.-R. Olderog: *Nets, Terms and Formulas*, Cambridge University Press, Cambridge (1991).

[12] M.W. Shields: Concurrent machines, *Computer Journal*, **28**, 449–465 (1985).

[13] G. Winskel, M. Nielsen: Models for concurrency, (to appear in S. Abramsky, D.M. Gabbay, T.S.E. Maibaum eds. *Handbook of Logic in Computer Science*).