Foundations of Software Technology and Theoretical Computer Science,
12th Conference, Proceedings, R. Shyamasundar (Ed.),
Lecture Notes in Computer Science **652**, Springer-Verlag (1992) 328–341.

CCS, Locations and Asynchronous Transition Systems

Madhavan Mukund* and Mogens Nielsen**

Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark

Abstract. We provide a simple non-interleaved operational semantics for CCS in terms of asynchronous transition systems. We 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 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 Introduction

Process algebras like CCS [9] are a well established formalism for specifying concurrent systems. 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 introducing 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, 13].

Both these approaches have some drawbacks. 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

* Permanent address: School of Mathematics, SPIC Science Foundation, 92, G.N. Chetty Road, T. Nagar, Madras 600 017, India. Email: `madhavan@ssf.ernet.in`. The author's stay in Denmark was supported by a grant from the Aarhus University Research Foundation.

** Email: `mn@daimi.aau.dk`

gives rise to the required behaviour—a lot of effort has to be put into 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.

All existing attempts at defining a non-interleaved semantics for CCS result in transition systems whose state spaces are considerably larger than those of the transition systems generated by the traditional interleaving semantics. In fact, Olderog conjectures in [13] that *"It is impossible to find an operational Petri net semantics which represents all the intended concurrency of process terms and for which the reachable markings are in 1-1 correspondence with the global states of the standard operational interleaving semantics."*

In this paper, we provide a simple non-interleaved operational semantics for a slightly restricted version of CCS satisfying the properties of Olderog's conjecture.

The restriction we make on CCS syntax is as follows. 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 major examples in [9] conform to our syntactic restriction.

We interpret CCS over the class of asynchronous transition systems [2, 14, 15]. An asynchronous transition system is a labelled 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 [15], 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 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. 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 [15].

We go on to define a notion of bisimulation on asynchronous transition systems which preserves independence. When applied to the transition systems we use to describe 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 and discusses the connection between our semantics and the traditional interleaved semantics for CCS. The next section describes bisimulations on asynchronous transition systems. We conclude in Section 6 with a discussion of how our work relates to other approaches and suggestions for further study.

2 Asynchronous transition systems

Asynchronous transition systems were introduced independently by Bednarczyk [2] and Shields [14]. 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 [15].

Definition 2.1. *An* asynchronous transition system *is a structure* $ATS = (S, i, E, I, Tran)$ *such that*

- $(S, i, E, Tran)$ *is a transition system, where S is a set of* states *(with* initial state *i), E is a set of* events *and $Tran \subseteq S \times E \times S$ is the* transition relation*.*
- $I \subseteq E \times E$ *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 [15]. In [15], 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. At the level of objects, 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* [7, 10, 12].

Definition 2.2. *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. 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.

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 2.3. *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.

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 2.4. *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* $l : E \rightarrow \Sigma$ *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.

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 $^-$ 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. 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 all free occurrences of 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 $\{0, 1\} \cap \Lambda = \emptyset$):

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

$P_0 \xrightarrow[u]{\mu} P_0'$ *implies* $P_0||P_1 \xrightarrow[0u]{\mu} P_0'||P_1$ **(Par)**

$P_1||P_0 \xrightarrow[1u]{\mu} P_1||P_0'$

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

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

$P \xrightarrow[u]{\mu} P',\ P \equiv P_1$ *and* $P' \equiv P_1'$ *implies* $P_1 \xrightarrow[u]{\mu} P_1'$ **(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 is 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 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 5.

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.

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, labelled a and b. In contrast, the process $ab + ba$ has four distinct events.

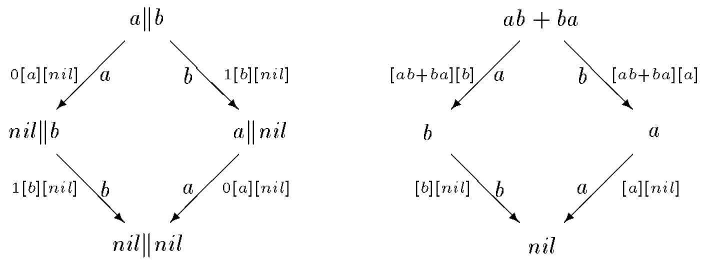

Fig. 1. Concurrency versus non-deterministic interleaving

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 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

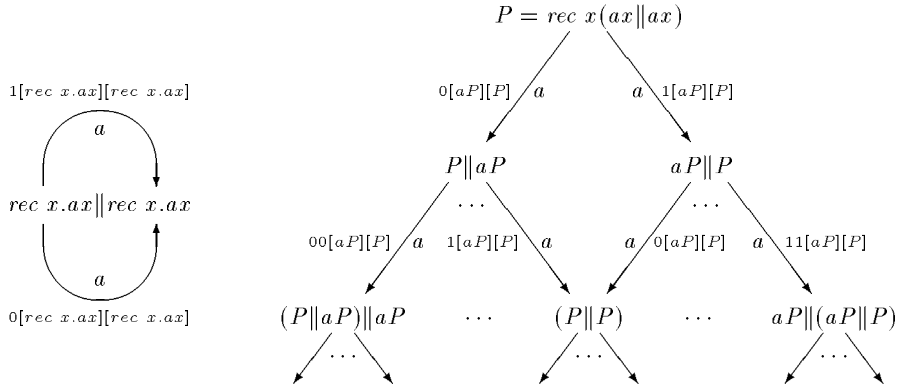

Fig. 2. Recursion

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.

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.

4 From CCS to asynchronous transition systems

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

Due to space constraints, we restrict ourselves to an informal account of the proof. Full details can be found in [11].

We first look at the labels associated with transitions by our operational semantics. We can show that for any transition $\hat{P} \xrightarrow[u]{\mu} \hat{P}'$, u is either of the form $s[P][P']$, where $s \in (\{0,1\} \cup \Lambda)^*$ and $\mu \in Act_\tau$, or of the form $s\langle s_0[P_0][P_0'], s_1[P_1][P_1']\rangle$, where $s, s_0, s_1 \in (\{0,1\} \cup \Lambda)^*$ and $\mu = \tau$. The first kind of label is associated with an "independent" action performed by an individual component of a process, while the second kind of label arises out of a synchronization. Henceforth, we shall often write P instead of $[P]$, for convenience.

Each distinct label in our transition system corresponds to an event. We can define the set of events Ev as follows:

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

Using the information present in the labels, for each event $e \in Ev$, 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. First, let $I_l \subseteq \{0,1\}^* \times \{0,1\}^*$ be given by:

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

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

Now, define an *independence relation on events* $I \subseteq Ev \times Ev$ as follows:

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

To analyze the behaviour of a process term P, we need to decompose it into its basic components. Components are identified by a (partial) function $Comp : (\{0,1\} \cup \Lambda)^* \times Proc \rightharpoonup Proc$ which, given a process term P and a location s, returns the subprocess of P located at s. $Comp$ is defined inductively as follows.

$$\begin{aligned}
&Comp(\varepsilon, P) &&= P, \text{ provided } P \not\equiv rec\ x.P_x \text{ (where } \varepsilon \text{ is the empty string)}\\
&Comp(0s, P_0 \| P_1) &&= Comp(s, P_0)\\
&Comp(1s, P_0 \| P_1) &&= Comp(s, P_1)\\
&Comp(\alpha s, P \backslash \alpha) &&= Comp(s, P)\\
&Comp(s, rec\ x.P) &&= Comp(s, P[rec\ x.P/x])
\end{aligned}$$

Notice that a process expression given by a guarded sum cannot be broken down further into components. Such expressions can be regarded as *sequential components*.

Using the labels on the transitions, we can "project" each move of P down to the sequential component where it occurs. For instance, given a transition of the form $P \xrightarrow[sP_1P_1']{\mu} P'$, we can assert that this event occurs in the sequential component $P_1 \equiv Comp(s, P)$, resulting in the component at s being transformed to the expression $P_1' \equiv Comp(s, P')$.

Conversely, we can identify when the moves of the sequential components can be lifted to the whole process. This is not completely straightforward. The moves of a 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.

Finally, it turns out that the occurrence of an event e leaves sequential components lying at locations independent of $Loc(e)$ untouched. This confirms that our definition of the independence relation on events, based on the locations where they occur, is a natural one.

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

- $S_{\hat{P}} = \{[P'] \mid P' \in Proc$ *and* $\exists P_0, P_1, \ldots, P_n.\ P = P_0, P' = P_n$ *and for all* $i \in \{1, 2, \ldots, n\}$, *there exists a move* $P_{i-1} \xrightarrow[u_i]{\mu_i} P_i$*—i.e.* P' *is reachable from* $\hat{P}\}$.

  $[\hat{P}]$ *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 independence relation on events defined earlier.*
- $Tran_{\hat{P}} \subseteq S_{\hat{P}} \times E_{\hat{P}} \times S_{\hat{P}} = \{([P], (\mu, u), [P']) \mid P \xrightarrow[u]{\mu} P'\}$
- $l_{\hat{P}} : E_{\hat{P}} \to Act_\tau$ 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.

To show that it is elementary we have to identify enough regions to satisfy the properties of "separation" and "enabling" given in Definition 2.3.

Given $s \in (\{0,1\} \cup \Lambda)^*$ and P of the form $\sum_{i \in I} \mu_i P_i$, we can define a region $R(s, P)$, consisting of all process terms whose sequential component at s is given by P.

So, the region $R(s, P)$ holds at a state P' iff the component of P' at location s is a sequential component equivalent to P—in other words, $Comp(s, P') \equiv P$.

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 $R(s, P) \in e^\bullet$.

We can then show that regions of the form $R(s, P)$ are sufficient to ensure that for every process term $\hat{P}$, $LTS(\hat{P})$ satisfies the regional axioms required for it to be elementary. In this way, we achieve what we set out to prove.

Theorem 4.1. *$\forall P \in Proc$. $LTS(P)$ is an elementary asynchronous transition system.*

In [11], we compare our semantics with the traditional interleaved semantics for CCS in terms of *foldings*. Here we shall just briefly present the main results. Details can be found in [11].

Foldings 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. Foldings have also been defined in [6], where they are called *transition-preserving (tp) homomorphisms.*

For a process term P in our language, let $TS(P)$ denote the standard transition system associated with P by the interleaved semantics given in [9]. Then, we can show the following.

Theorem 4.2. *For each $P \in Proc$, $TS(P)$ can be folded onto $LTS(P)$ (when we "forget" about the independence relation and regard $LTS(P)$ as just a labelled transition system).*

This means that our approach yields a tractable system whenever the conventional approach would. This has an important bearing on the possibilities of mechanically verifying properties of such systems.

We can extend the notion of a folding to asynchronous transition systems in such a way that the folding preserves the independence relation. We can then relate our operational semantics to Winskel and Nielsen's denotational semantics for CCS in terms of asynchronous transition systems [15]. For $P \in \mathit{Proc}$, let $\mathit{Den}(P)$ denote the asynchronous transition system associated with P by the denotational semantics of [15]. Then we have the following.

Theorem 4.3. *For each $P \in \mathit{Proc}$, there is an (independence-preserving) folding from $\mathit{Den}(P)$ onto $LTS(P)$.*

This provides a formal justification for our choice of the independence relation between events.

5 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 [15]. Though transition system morphisms preserve behaviour, they are unsatisfactory for defining a natural notion of bisimulation because they map states and events in such a way that independence is 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 $ATS_k = (S_k, i_k, E_k, I_k, \mathit{Tran}_k)$, $k = 1, 2$.

Definition 5.1. *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 $\varphi \subseteq E_1 \times E_2$.

To define a notion of equivalence based on these lI-consistent relations between events, 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 5.2. $\sim_\Phi = \{\sim_\varphi \mid \varphi \in \Phi\}$ *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 5.3. $ATS_1 \sim ATS_2$ *iff* $i_1 \sim_\emptyset i_2$.

It is easy to verify the following.

Proposition 5.4. $\sim$ *is an equivalence relation on labelled asynchronous transition systems.*

Example 5.1. 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 5.2. 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 $\sim$ 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 $ATS = (S, i, E, I, \mathit{Tran})$, define the weak transistion relation $\Rightarrow \subseteq S \times E \times S$ in the obvious way.

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

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

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

Now consider a pair of Act_τ-labelled asynchronous transition systems (ATS_1, l_1) and (ATS_2, l_2), where $ATS_k = (S_k, i_k, E_k, I_k, \mathit{Tran}_k), k = 1, 2$, 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 5.5. *$\simeq_\Phi = \{\simeq_\varphi \mid \varphi \in \Phi\}$ 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* $P \approx_l P'$.

6 Discussion

In this paper, we have described how to provide a semantics for CCS in terms of elementary asynchronous transition systems which is close to the standard operational semantics for CCS. By appealing to the coreflection between this class of transition systems and 1-safe Petri nets established in [15], 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. 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.

It may appear that our semantics is but a special case of, say, the proved transition approach of [3]. However, if we derive an asynchronous transition system describing a term P in our language using their semantics, with the independence relation given by their concurrency relation on proofs, the resulting transition system is *not* necessarily elementary.

In comparison with the approaches of [3, 8], our semantics is simple in the sense that labels on transitions directly yield the underlying events of the system, without resorting to a second level of reasoning about the transitions.

In the approaches which directly yield a Petri net semantics [3, 6, 13], 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.

The fact that the traditional interleaved transition system for a process P can be folded onto the asynchronous transition system $LTS(P)$ assigned by our semantics means that our approach yields a tractable system whenever the conventional approach would.

Another interesting feature of our semantics 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 the full language.

In this paper, we have also introduced a notion of bisimulation over labelled asynchronous transition systems which preserves independence along individual runs. 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]. The equivalence defined in [5] 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 we would have 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 5.1).

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, *Springer LNCS 469*, 96–141 (1990).
4. G. Boudol, I. Castellani, M. Hennessy, A. Kiehn: Observing localities, *Springer LNCS 520*, 93–102 (1991).
5. G. Boudol, I. Castellani, M. Hennessy, A. Kiehn: A theory of processes with localities, *Report 1632*, INRIA, Sophia-Antipolis (1992)
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, *Springer LNCS 431*, 162-176 (1990).
9. R. Milner: *Communication and Concurrency*, Prentice-Hall, London (1989).
10. M. Mukund: Transition system models for concurrency, *Report DAIMI-PB-399*, Computer Science Department, Aarhus University, Aarhus, Denmark (1992).
11. M. Mukund, M. Nielsen: CCS, locations and asynchronous transition systems, *Report DAIMI-PB-395*, Computer Science Department, Aarhus University, Aarhus, Denmark (1992).
12. M. Nielsen, G. Rozenberg, P.S. Thiagarajan: Elementary transition systems, *Theoretical Computer Science*, **96**, 1, 3–33 (1992).
13. E.-R. Olderog: *Nets, Terms and Formulas*, Cambridge University Press, Cambridge (1991).
14. M.W. Shields: Concurrent machines, *Computer Journal*, **28**, 449–465 (1985).
15. 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*).