A Local Presentation of Synchronizing Systems

R. Ramanujam
The Institute of Mathematical Sciences
Madras - 600 113, India.
e-mail: jam@imsc.ernet.in

Abstract

We study systems of sequential agents which communicate by synchronization, whose behaviours are given by a subclass of event structures. Transition system models for such structures typically require global state information which cannot be obtained by taking products of local transition systems. We offer a presentation whereby the notion of local state is modified, and an appropriate product operation precisely captures this class of behaviours. This is shown using a back-and-forth construction.

1 Introduction

Consider a system of several *agents* which act autonomously and periodically exchange information in rendezvous. This is a typical model studied in concurrency theory. One way of specifying "handshake" synchronization is by having letters of a common alphabet of actions between concurrent programs. For example, in a TCSP-style syntax, we can find programs like $(a\ +\ aa)\ c\quad||\quad(b\ +\ bb)\ c$ where we see two agents which synchronize after doing either one action or two actions locally. Note that this allows behaviours where synchronization occurs when one agent has done one local action whereas the other has done two local actions.

The distributed system is specified here as a parallel composition of some sequential processes. We can ask the following question : given a description of the behaviours of a distributed system in some fashion, when (and how) can we decompose the given system into a parallel composition of sequential processes, preserving behaviour ? This becomes a nontrivial question, when the given specification of the distributed system admits complicated types of synchronization. For example, in Mazurkiewicz trace theory [Maz], consider a concurrency alphabet $\{a, b, c\}, \{(a, b), (b, a)\}$, and the trace language $\{[abc], [aabbc]\}$ over it. According to it, when the synchronization occurs, both the agents have done one local action, or both have done two local actions, disallowing the possibility of one local action by one agent and two by the other.

In this paper, we address the question above, and present a subclass of event structures which can be obtained as products of transition systems. The attempt here is *not* to propose a new model of concurrency, but rather to highlight possible ways of distributing global information.

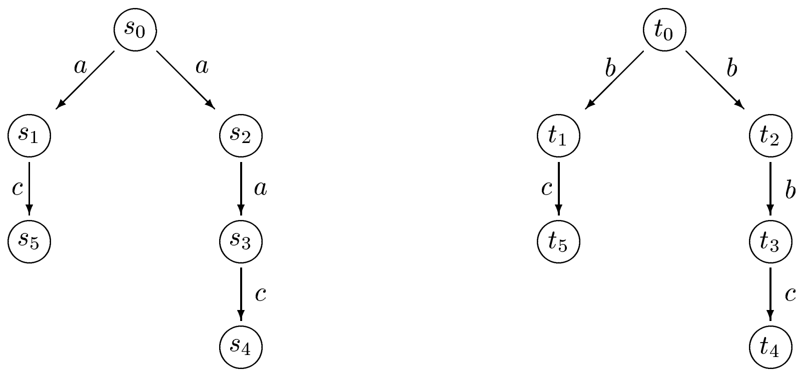

Figure 1: Two transition systems

Transition system presentation of non-sequential behaviour has been studied extensively; we refer the reader to [BC, Dro, Muk, NRT, WN] for a variety of interesting results. Typically, these presentations are at a global level. That is, the transition system consists of the set of all possible global states of the system and the allowed transitions between these states. There are models where the global states are obtained by taking products of local states, but even these require a global transition relation.

In a sense, this need for global information is inescapable. Suppose we consider a pair of transition systems as in Figure 1 to represent the situation above. When we consider the product transition system, a part of which is depicted in Figure 2, we find that c is enabled at the global state (s_1, t_3), when agent 1 has performed an a and agent 2 has done two b's. Indeed, the language $[abc] \cup [aabbc]$ cannot be obtained as a product of two transition systems, one over the alphabet $\{a, b\}$ and the other over $\{b, c\}$.

In this paper, we propose systems where local states can be *common* between agents. The idea is that a local state includes not only local information, but also information about the rest of the system available "as of date". Thus, an agent in a synchronizing system, has access not only to current values of local variables, but also values of other agents' variables as obtained by communication. This gives us a **local** presentation of synchronizing systems, where the understanding of the product operation is changed so that whenever a synchronization occurs, all agents participating in it reach an identical local state.

Formally, we consider the model of synchronizing systems presented as a subclass of event structures and show that these can be obtained as unfolded behaviours of transition systems with common local states. In a technical sense, this representation is faithful, as evidenced by a back-and-forth construction.

The paper is organized as follows: in the next section, we define the model of *Synchronization Structures* with finitely many agents and transition systems associated with such systems. In Section 3, we formally introduce the class of

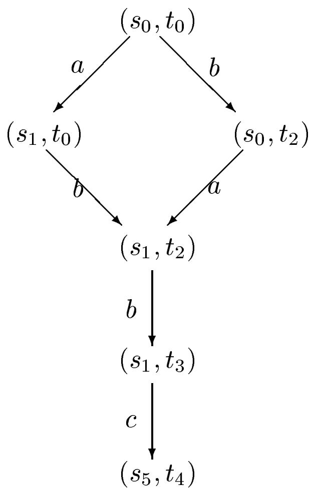

Figure 2: A part of the product transition system

Synchronizing Transition Systems, and associate a synchronization structure with each such system. After this, we show that these associations preserve behaviour both ways. We end the paper with a discussion.

2 Synchronization Structures

In [LRT], a model of distributed systems called Communicating Sequential Agents (CSAs) has been defined in the following manner: assume a collection of n agents, each of which is sequential, interacting by message passing. Each agent is 'tree-like', in the sense that its behaviour is given as a 'backwards-linear' poset of event occurrences. We present a subclass of CSAs below where interaction between agents is only by means of synchronization, modelled by shared event occurrences. Unlike [LRT], we study labelled systems here, for relating with transition systems in the next section.

Fix a finite set of *locations* $Loc = \{1, \ldots, n\}, n > 0$. The first notion we need is that of a **distributed alphabet**, a tuple $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, where each Σ_i is a finite nonempty set of *actions of agent* i. When an action a is in $\Sigma_i \cap \Sigma_j, i \neq j$, we think of it as a synchronization action between i and j. Given a distributed alphabet $\widetilde{\Sigma}$, we often speak of the set $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cup \ldots \cup \Sigma_n$ as the alphabet of the system. We also make implicit use of the associated function $loc : \Sigma \to 2^{\{1,\ldots,n\}}$ defined by $loc(a) \stackrel{\text{def}}{=} \{i \mid a \in \Sigma_i\}$.

The formal definition of a synchronization structure is as follows:

Definition 2.1 *A Synchronization Structure with n agents (an n-SS) over a distributed alphabet $\widetilde{\Sigma}$ is a triple $\mathcal{E} = (E, \leq, \lambda)$ where $n \in \mathcal{N}, n > 0$, and*

(i) *E is a set of event occurrences,*

(ii) $\leq \ \subseteq E \times E$ *is a partial order called the causality relation,*

(iii) $\lambda : E \to \Sigma$ *is a labelling function such that* $\forall e \forall i \in \{1, ..., n\},\ \{e' | e' \leq e\} \cap E_i$ *is totally ordered by* $\leq$, *where* $E_i \stackrel{\text{def}}{=} \{e' \in E \mid \lambda(e') \in \Sigma_i\}$, *and*

(iv) $\leq \ = (\bigcup_i \leq_i)^*$, *where*

$\leq_i \stackrel{\text{def}}{=} \ \leq \bigcap (E_i \times E_i)$.

We will use the notation $\downarrow x$ for the initial segment of the partially ordered set $(X, \leq)$ up to the element x, i.e. for $x \in X$, $\downarrow x = \{y | y \leq x\}$; similarly for $X' \subseteq X$, $\downarrow X' = \{y | \exists x \in X',\ y \leq x\}$. We will also speak of SSs, leaving n implicit.

An agent is simply the set of event occurrences having the same name. We will often speak of the set E_i as agent i. We will also find it useful to define a *naming function* $\eta : E \to 2^{Loc}$ give by $\eta(e) \stackrel{\text{def}}{=} \{i \mid e \in E_i\}$. (Often, what we have called event occurrences are referred to as action occurrences; we do not need the distinction here.)

Note that condition (iii) imposes backward-linearity, making each agent tree-like. It is in this sense that the agents are sequential. When an event occurrence belongs to more than one agent, we interpret that as the occurrence of a synchronization.

$\leq$ is a causality relation in the sense that when $e_1 \leq e_2$, every observation of event occurrence e_2 necessarily implies an earlier observation of e_1. Note that due to condition (iv), when $\eta(e_1) \cap \eta(e_2) = \emptyset$, and $e_1 \ \leq \ e_2$, there must be a sequence of synchronization occurrences between e_1 and e_2 such that an agent in $\eta(e_1)$ participates in the first and an agent in $\eta(e_2)$ participates in the last.

λ and $\leq$ lead naturally to notions of conflict and concurrency in SSs. When we consider event occurrences which are incomparable under the causality ordering, conflicting ones are those which cannot both occur in the same computation, and concurrent ones are those which can. These are formally given below.

We say that event occurrences e_1 and e_2 are in *local conflict* when neither $e_1 \leq e_2$, nor $e_2 \leq e_1$, and $\eta(e_1) \cap \eta(e_2) \neq \emptyset$. Since agents are sequential, we do not interpret causal independence within agents as potential concurrency but as denoting choice made in computation. e_1 and e_2 are in *conflict*, if and only if there exist $e_1' \leq e_1, e_2' \leq e_2$ such that e_1' and e_2' are in local conflict.

When are two event occurrences concurrent ? We can say that e_1 and e_2 are *concurrent* if $\eta(e_1) \cap \eta(e_2) = \emptyset$, e_1 and e_2 are incomparable under the causal ordering, and $\downarrow e_1 \cup \downarrow e_2$ is conflict-free. Note that for any $e_1, e_2 \in E$, we have $e_1 \leq e_2$ or $e_2 \leq e_1$ or e_1 is in conflict with e_2 or e_1 and e_2 are concurrent.

We can now define a notion of a *global state* in an SS: $c \subseteq E$ is a global state iff $\downarrow c \subseteq c$ and c is conflict-free. Thus a state being downward-closed and

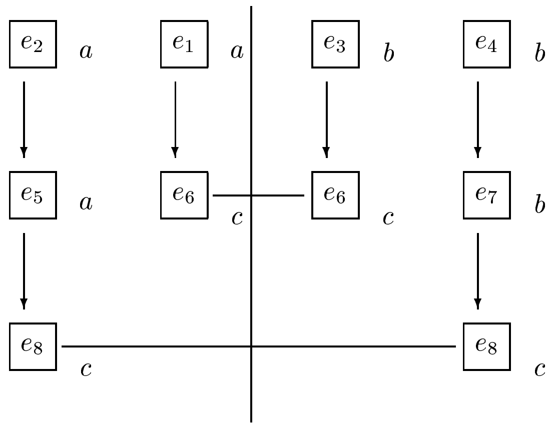

Figure 3: A synchronization structure over $\{a, c\}, \{b, c\}$

conflict-free, can be thought of as a partial run. We will often refer to global states as *configurations*. Note that the empty set is always a configuration. More importantly, for any $e \in E$, the set $\downarrow e$ is a configuration, a fact which we will use crucially later on.

Consider the 2-SS in Figure 3. Events e_2 and e_6 are in local conflict, whereas e_2 and e_3 are concurrent. For this example, $\{e_2, e_3, e_5\}$ is a configuration, whereas $\{e_3, e_5\}$ and $\{e_2, e_3, e_5, e_6\}$ are not.

We say that an SS is *finitary* if and only if $\downarrow e$ is finite, for every $e \in E$. In the context of computation, it is natural to restrict attention to finitary SSs, and we will do precisely that. In fact, we will also be interested only in the *finite* configurations of finitary SSs. Note that finitariness implies discreteness of the partial order and hence we will freely make use of $\lessdot$, the covering relation of $\leq$.

A configuration in an SS is a state in the sense that event occurrences cause state transitions. This is seen precisely in the transition system $T\mathcal{E}$ associated with the finitary SS $\mathcal{E}$. Let C denote the set of finite configurations of $\mathcal{E}$. (Here, and in what follows, we use the notation $c \oplus \{e\}$ to mean the set $c \cup \{e\}$ alongwith the assertion that $e \notin c$).

- $T\mathcal{E} \stackrel{\text{def}}{=} (C, \rightarrow, \emptyset)$, where
- $c \xrightarrow{a} c'$ iff $\exists e \in E, c' = c \oplus \{e\}$, and $\lambda(e) = a$.

The transition system describes a connected acyclic graph, with an initial state, namely the empty configuration (it has no in-coming edges, and every other

$$
\begin{tikzcd}
& \emptyset \arrow[dl, "a"'] \arrow[dr, "b"] & \\
\{e_1\} \arrow[dr, "b"'] & & \{e_4\} \arrow[dl, "a"] \\
& \{e_1, e_4\} \arrow[d, "b"'] & \\
& \{e_1, e_4, e_7\} &
\end{tikzcd}
$$

Figure 4: A part of transition system associated with the SS of Fig 3

node is reachable from it). Figure 4 shows a part of the transition system associated with the SS of Figure 3.

On the other hand, we can also define a notion of *local state* in an SS. Clearly, if $e \in E_i$, the set $\downarrow e$ can be thought of as a "local history" of agent i and this becomes a good candidate for a local state of that agent. Let $\mathcal{E}$ be a finitary SS. Define the transition system $T_i\mathcal{E}$ associated with $\mathcal{E}$ to be the tuple $(LC_i, \rightarrow_i)$, where the set of *local configurations* of agent i is given by :

- $LC_i \stackrel{\text{def}}{=} \{\emptyset\} \cup \{\downarrow e \mid e \in E_i\}$, and
- $x \stackrel{a}{\rightarrow}_i x'$ iff $\exists e \in E_i, x' \cap E_i = (x \cap E_i) \oplus \{e\}$, and $\lambda(e) = a, a \in \Sigma_i$.

On the other hand, we cannot simply take products of $T_i\mathcal{E}$ for all $i \in \{1, ..., n\}$ to obtain $T\mathcal{E}$. In the next section, we address these issues in the general context of transition systems.

In Figure 5, we see the local transition systems of the two agents in the SS of Figure 3. It looks almost the same as the pair of transition systems in Figure 1, but note that both s_5 and t_5 have been mapped to $\downarrow e_6$, and similarly for s_4 and t_4. At the global level, we find that Figure 4 is in correspondence with Figure 2, except for the missing transition. Thus, the synchronization structure in Figure 3 specifes the same behaviour as the trace language we mentioned in Section 1.

3 Synchronizing Transition Systems

A *labelled transition system* over a nonempty finite alphabet Σ is a tuple $TS = (Q, \rightarrow, q_0)$, where Q is an at most countable set of states, $q_0 \in Q$, and $\rightarrow \subseteq Q \times \Sigma \times Q$. We usually write $q \stackrel{a}{\rightarrow} q'$ to mean $(q, a, q') \in \rightarrow$, and interpret it to mean that the system can perform an action a at a state q and the resulting

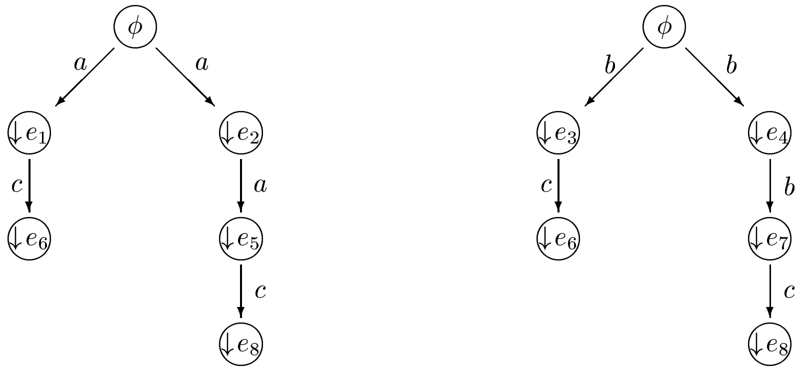

Figure 5: Local transition systems associated with structure of Fig 3

state is q'. A *run* of TS is a sequence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots$ — a possible "execution" of the system. Labelled transition systems provide a natural model for the study of system behaviour.

We are interested in interactions between systems. A natural way to represent interactions among n agents is by having n transition systems, each working on its own alphabet of actions, except the system undergoes "joint" transitions when common actions are encountered. The obvious idea is to take products of transition systems. However, this will in general include more transitions than those present in transition systems associated with SSs. (For instance, compare Figure 2 with Figure 4.) We somehow need the ability to synchronize on transitions rather than on action labels. The model defined below introduces that ability — whenever two agents perform a synchronized transition, we insist that the resulting local states be identical. For this, it is of course essential to consider agents which can "share" local states.

Definition 3.1 *Let* $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$ *be a distributed alphabet. A* **Synchronizing System** *of* n *agents over* $\widetilde{\Sigma}$ *is a tuple* $\mathcal{T} = (TS_1, \ldots, TS_n)$, *where* $TS_i = (Q_i, \rightarrow_i, q_i^0)$ *is a labelled transition system over* Σ_i, *for* $i \in Loc$, *and* $q_1^0 = q_2^0 \ldots = q_n^0$.

Given a system as above, we will often refer to the set $Q = \bigcup_i Q_i$ as the set of states of the system. Let L_i denote the runs of TS_i, for $i \in Loc$. (We will avoid the awkward usage 'synchronizing system of transition systems' and abbreviate synchronizing systems as STSs.)

We have specified here that all systems begin at the identical initial state. We can read this as 'initial common knowledge' in the system. That is, the initial values of local variables of every agent is available to every other agent, and the situation is one of perfect information among agents. As the agents start making asynchronous moves, their information about others goes "out of sync", and synchronization is an occasion to update ("resync") information about the system.

The remarks above are formalized in the definition of global runs of synchronizing systems. Before we proceed to the definition, we need some notation. Let $Q = \bigcup_i Q_i$ and let $\sigma \in Q^*$ such that $\sigma = q_0\sigma'$. We say that q in σ is i-maximal for $i \in Loc$, when $\sigma = q_0 q_1 \ldots q_m$ and $q = q_k$, where k is the largest index such that $q_k \in Q_i$. Note that for all $i \in Loc$, such an i-maximal element exists, since $q_0 \in \bigcap_i Q_i$. We can refer to this state as $max_i(\sigma)$, and it can be thought of as the 'current' local state of agent i.

Definition 3.2 *Let $\mathcal{T}$ be a synchronizing system of n agents over $\widetilde{\Sigma}$. A* **global run** *of $\mathcal{T}$ is a sequence $q_0, a_0, q_1, a_1, q_2 \ldots$, where*

- $\{q_0, q_1, \ldots\} \subseteq Q = \bigcup_i Q_i$,
- $\{a_0, a_1, \ldots\} \subseteq \Sigma = \bigcup_i \Sigma_i$,
- *q_0 is the common initial state and*
- *for all $k \geq 0$, for $j \in loc(a_k)$, $max_j(q_0 q_1 \ldots q_k) \xrightarrow{a_k}_j q_{k+1}$.*

This definition is the main definition of the paper, in some sense. Global runs represent arbitrary interleavings of actions of different agents. The run is extended by an action a if and only if for every agent i participating in that action, a is enabled at the current local state of i and the resulting local state is identical for all such agents. Common local states have been crucially used here. Note that given a system $\mathcal{T}$, the singleton sequence q_0 may very well be the only global run of the system.

In Figure 6, we present an STS with 2 agents which represents the behaviour specified by the trace language of Section 1. Note that this is isomorphic to the pair of local transition systems of Figure 5.

Another way of presenting global runs is by defining products of synchronizing systems as given below. This product construction is different from the standard notion. However, the set of runs of the product system is in bijection with the set of global runs of the system. In that sense, the definition above carries the same content as the one below.

Definition 3.3 *Let $\mathcal{T}$ be a synchronizing system over $\widetilde{\Sigma}$. The* **product system** *induced by $\mathcal{T}$ is the Σ-labelled transition system $TS = (Q, \Rightarrow, q_0)$, where*

- $\widetilde{Q} \stackrel{\text{def}}{=} Q_1 \times \ldots \times Q_n$,
- $q_0 \stackrel{\text{def}}{=} q_1^0 = \ldots = q_n^0$, *and*

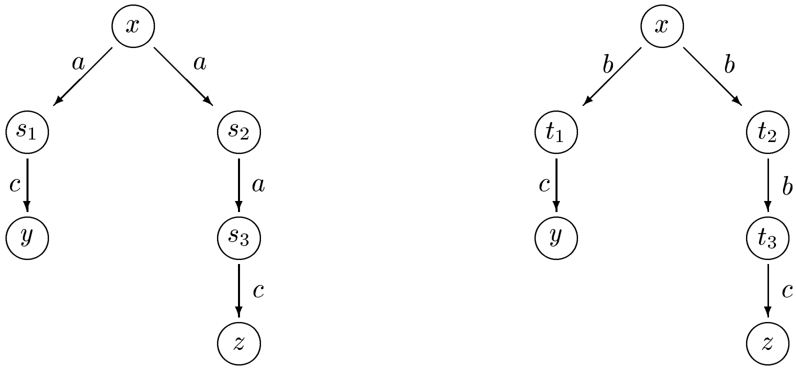

Figure 6: A 2-STS for Example 1

- $\Rightarrow \subseteq \widetilde{Q} \times \Sigma \times \widetilde{Q}$ *is the* **global** *transition function defined as follows:* $(q_1, \ldots, q_n) \overset{a}{\Rightarrow} (q'_1, \ldots, q'_n)$ *iff*
  - $\forall i \in loc(a), q_i \overset{a}{\rightarrow}_i q'_i$,
  - $\forall j \notin loc(a), q_j = q'_j$, *and*
  - $\forall i, j \in loc(a), q'_i = q'_j$.

When we form the induced product system for the STS of Figure 6, we find (as one may expect) that it is isomorphic to the global configuration system of Figure 4. The following proposition, not surprisingly, asserts that this is no accident.

Proposition 3.4 *Let $\mathcal{E}$ be an SS over $\widetilde{\Sigma}$. Then $\mathcal{T}_{\mathcal{E}} \overset{\text{def}}{=} (T_1\mathcal{E}, \ldots, T_n\mathcal{E})$ is a synchronizing system over $\widetilde{\Sigma}$, whose induced product is isomorphic to $T\mathcal{E}$.*

Having associated a synchronizing transition system with a given synchronization structure, we now proceed to construct the event structure given the transition system. This forms the main technical content of the paper, and the construction highlights the information transfer between agents discussed above. We will be working throughout with the global runs of Definition 3.2, rather than the product system of Definition 3.3. We will write global runs of the form $q_0, a_0, q_1, a_1, \ldots$ as $q_0 \overset{a_0}{\Longrightarrow} q_1 \overset{a_1}{\Longrightarrow} \ldots$.

For the rest of the section, fix $\mathcal{T}$, a synchronizing system of n agents over a distributed alphabet $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. Let R be the set of *finite non-null* global runs of this system. We use δ, δ' etc. to denote elements of R. Note that non-null runs have at least one transition. Define $\tau : R \to \Sigma$ by: $\tau(\delta) \overset{\text{def}}{=} a_k$ where $\delta = \$q_0 \overset{a_0}{\Longrightarrow} q_1 \overset{a_1}{\Longrightarrow} \ldots q_k \overset{a_k}{\Longrightarrow} q_{k+1}$. We will also use $\eta : R \to 2^{Loc}$ defined by $\eta(\delta) \overset{\text{def}}{=} loc(\tau(\delta))$. We use the notation $\preceq$ to denote the prefix ordering on sequences.

Consider $\delta \in R$ such that $\delta = q_0 \overset{a_1}{\Longrightarrow} q_1 \overset{a_2}{\Longrightarrow} \ldots \overset{a_k}{\Longrightarrow} q_k$. For every $i \in Loc$, let m_i be the largest index such that $q_{m_i} \in Q_i \cap Q_j$ for some $j \in loc(a_k)$. Clearly for $i \in loc(a_k), m_i = k$. In general, call the prefix $q_0 \overset{a_1}{\Longrightarrow} q_1 \overset{a_2}{\Longrightarrow} \ldots q_{m_i}$, denoted $\delta(i)$, the *i-snapshot at δ*. We can think of this as all the information available to agents in $loc(a_k)$ about agent i pooled together. Even though different agents in $loc(a_k)$ may see earlier or later states of i before the synchronization a_k, at the state q_k, they all see the latest.

For example, consider a situation like this : agents 1 and 2 synchronize independently of a synchronization between agents 3 and 4. At this juncture, 2 has a recent view of 1's local state, but in 3's view, 1 is still in the initial state. On the other hand, 3 has the latest information about 4, but 2 does not. Now, agents 2 and 3 synchronize, and in the joint view of agents 2 and 3, the current states of 1 and 4 become available to both. In this sense, global runs of STSs depict the "gossip" scenarios of [MS].

Define the *projection maps* $\phi_i : R \to L_i$, for $i \in Loc$ as follows: $\phi_i(\delta)$ is the sequence obtained by erasing all states not in Q_i and actions not in Σ_i. It can be easily seen that ϕ_i is well-defined, though it is clearly not onto.

Let the map $\epsilon : R \to L_1 \times ... \times L_n$ be defined as follows:
$\epsilon(\delta) = (\phi_1(\delta_1), \ldots, \phi_n(\delta_n))$,
where δ_i is the i-snapshot at δ. Clearly the map ϵ is well-defined. Define $\approx \subseteq R \times R$ by: $\delta_1 \approx \delta_2$ iff $\epsilon(\delta_1) = \epsilon(\delta_2)$. Let $[\delta]$ denote the equivalence class of δ under $\approx$. We use $\approx$ to associate an SS with the synchronizing transition system.

Below, assume that $\preceq$ is extended pointwise: that is, for tuples,
$(\rho_1, \ldots, \rho_n) \preceq (\rho'_1, \ldots, \rho'_n)$ iff $\forall i \in Loc, \rho_i \preceq \rho'_i$.

The following propositions are reasonably easy to show:

Proposition 3.5 *Suppose $\delta_1 \preceq \delta_2$ and $\eta(\delta_1) \cap \eta(\delta_2) \neq \emptyset$. Then $\epsilon(\delta_1) \preceq \epsilon(\delta_2)$.*

Proposition 3.6 *Suppose $\delta_1 \approx \delta_2$. Then $\tau(\delta_1) = \tau(\delta_2)$.*

Proposition 3.7 *Suppose $\delta_1 \preceq \delta_2$ and $\delta_2 \approx \delta'_2$. Then there exists $\delta'_1 \preceq \delta'_2$ such that $\delta_1 \approx \delta'_1$.*

Proposition 3.8 *Let $\mathcal{T} = (TS_1, \ldots, TS_n)$ be a synchronizing system over the distributed alphabet $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. Define the structure $\mathcal{E} = (E, \leq, \lambda)$ by:*

$$E \overset{\text{def}}{=} \{[\delta] | \delta \in R_{\mathcal{T}}\},$$

$$\leq \overset{\text{def}}{=} \{([\delta_1], [\delta_2]) | \text{ there exists } \delta \preceq \delta_2, [\delta] = [\delta_1]\} \text{ and}$$

$$\lambda([\delta]) \overset{\text{def}}{=} \tau(\delta).$$

Then $\mathcal{E}$ is a finitary synchronization structure over $\widetilde{\Sigma}$.

Proof: Reflexivity of $\leq$ is trivial. Antisymmetry follows from Proposition 3.5 above. Transitivity and backward linearity (within agents) can be proved using Proposition 3.7. Proposition 3.6 ensures that λ is well-defined. For finitariness, we only need to observe that every element of R is a finite sequence, and hence has only finitely many prefixes.

We need to show that $\leq$ is generated from $\leq_i$. It suffices to prove that whenever $[\delta_1] \lessdot [\delta_2]$, we have that $\eta(\delta_1) \cap \eta(\delta_2) \neq \emptyset$. Now assume that $[\delta_1] \lessdot [\delta_2]$, but that $\eta(\delta_1) \cap \eta(\delta_2) = \emptyset$. Let $\delta \preceq \delta_2$ such that $[\delta] = [\delta_1]$ and $\tau(\delta) = \tau(\delta_1) = a$, say. Let $\delta = \delta' \stackrel{a}{\Longrightarrow} q_k$ and $\delta_2 = \delta \stackrel{a_1}{\Longrightarrow} q_{k+1} \ldots \stackrel{a_m}{\Longrightarrow} q_{k+m}$. Now suppose there exists j between 1 and m such that $loc(a) \cap loc(a_j) \neq \emptyset$ and $loc(a_j) \cap loc(a_m) \neq \emptyset$. Then, by proposition above, $\epsilon(\delta) \leq \epsilon(\delta_j) \leq \epsilon(\delta_2)$ contradicting the assumption that $[\delta] \lessdot [\delta_2]$. Thus, we have that $loc(a) \cap loc(a_j) = \emptyset$, for every j such that $loc(a_j) \cap loc(a_m) \neq \emptyset$. But then, for $i \in loc(a)$, the i-snapshot at δ_2 is a strictly smaller prefix of the i-snapshot at δ, which contradicts the assumption that $\delta \leq \delta_2$. Thus, we must have that $\eta(\delta_1)(= \eta(\delta)) \cap \eta(\delta_2) \neq \emptyset$, as required. $\square$

We refer to the structure $\mathcal{E}$ as the SS associated with $\mathcal{T}$ and denote it $\mathcal{E}_{\mathcal{T}}$.

4 Back and Forth

Given an SS $\mathcal{E}$, we can associate an STS $\mathcal{T}_{\mathcal{E}}$ with it, and with this STS we can associate an SS $\mathcal{E}_{\mathcal{T}_{\mathcal{E}}}$. What is the relationship between $\mathcal{E}$ and $\mathcal{E}_{\mathcal{T}_{\mathcal{E}}}$?

Theorem 4.1 *Let $\mathcal{E} = (E, \leq, \lambda)$ be a finitary synchronization structure over a distributed alphabet $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, and let $\mathcal{E}_{\mathcal{T}_{\mathcal{E}}} = (E', \leq', \lambda')$. Then there is a bijection $f : E \to E'$ such that :*

$\forall e, e' \in E, e \leq e'$ iff $f(e) \leq' f(e')$ and

$\forall e \in E, \lambda(e) = \lambda'(f(e))$.

In other words, $\mathcal{E}_{\mathcal{T}_{\mathcal{E}}}$ is isomorphic to $\mathcal{E}$.

Proof: Let $\mathcal{E} = (E, \leq, \lambda)$ be the given SS, and fix $\mathcal{T}_{\mathcal{E}}$ to be the STS $\mathcal{T} = (TS_1, \ldots, TS_n)$. Note that TS_i is simply the transition system $T_i\mathcal{E}$ defined in Section 2. It is given by the tuple $(LC_i, \rightarrow_i)$, where the set of local configurations of agent i is given by:

- $LC_i \stackrel{\text{def}}{=} \{\emptyset\} \cup \{\downarrow e \mid e \in E_i\}$, and
- $x \stackrel{a}{\rightarrow} x'$ iff $\exists e \in E_i, x' \cap E_i = (x \cap E_i) \oplus \{e\}$, and $\lambda(e) = a, a \in \Sigma_i$.

Let C denote the set of finite configurations of $\mathcal{E}$. A *schedule* of a configuration $c \in C$ is a sequence $e_1 ... e_k$ such that $l \neq m$ implies $e_l \neq e_m, c = \{e_1, ..., e_k\}$, and if $e_l \leq e_m$, then $l \leq m$. Note that every schedule (of any configuration) corresponds to a global run in T. Conversely for every global run from $\emptyset$ to a local state $\downarrow e$ in T of the form $\emptyset \stackrel{\lambda(e_1)}{\Longrightarrow} \downarrow e_1 \ldots \stackrel{\lambda(e_k)}{\Longrightarrow} \downarrow e_k$, the set $\{e_1, \ldots, e_k\}$ is indeed a configuration in C, and $e_1 \ldots e_k$ is a schedule of that configuration.

Given $e \in E, i \in \eta(e)$, consider the set $\{\downarrow e' | e' \leq e, i \in \eta(e')\}$; since $\mathcal{E}$ is finitary and backwards linear within agents, this set can be written as a finite sequence of elements $\downarrow e_1, ..., \downarrow e_k$, where for $j \in \{1, ..., k-1\}, e_j < e_{j+1}$ and $e_k = e$; call this sequence, prefixed by $\{\emptyset\}$, the local history of i at e, denoted $lh_i(e)$.

Consider a schedule of a configuration c ending at event occurrence e, and let $\lambda(e) = a$. Now consider the tuple $< \rho_1, \ldots, \rho_n >$, where for $j \in \{1, ..., n\}, \rho_j$ is the null sequence if $\downarrow e \cap E_j = \emptyset$, and otherwise it is $lh_j(e')$, where e' is the maximal j-event in $\downarrow e$. Call this tuple the $\eta(e)$-view at e. (Note that when $\{i, j\} \subseteq \eta(e)$, the i-view is the same as the j-view.)

We claim that the $\eta(e)$-view at e is indeed an event occurrence in $\mathcal{T}$ and hence a member of E'. To see this, observe firstly that for any $e' \in E, j \in \eta(e')$, $lh_j(e')$ is a local j-run in TS_j when $|lh(e')| > 1$. We can easily check that every such $\eta(e)$-view is generated as $\epsilon(\delta)$ for some run δ in $\mathcal{T}$.

We can thus meaningfully define the map $F : E \rightarrow E'$: given by $F(e) =$ the $\eta(e)$-view at e.

To prove that F is injective, suppose $e_1 \neq e_2$. If $\eta(e_1) \cap \eta(e_2) \neq \emptyset$. Then for every $i \in \eta(e_1) \cap \eta(e_2)$, clearly $lh_i(e_1) \neq lh_i(e_2)$ and hence $\eta(e_1)$-view at e_1 is distinct from the $\eta(e_2)$-view at e_2. Otherwise, suppose that $\eta(e_1) \cap \eta(e_2) = \emptyset, i \in \eta(e_1), j \in \eta(e_2)$, but that the $\eta(e_1)$-view at e_1 is identical to the $\eta(e_2)$-view at e_2. This means that e_1 is the maximal i-event in $\downarrow e_2$, and that e_2 is the maximal j-event in $\downarrow e_1$. Then in particular, we get $e_1 \leq e_2, e_2 \leq e_1$ and $e_1 \neq e_2$, contradicting the antisymmetry of $\leq$. Thus F is injective.

Now consider $e' \in E', i \in \eta'(e')$. Then e' is an i-event occurrence in $\mathcal{T}_{\mathcal{E}}$ say, $< \sigma_1, ..., \sigma_n >$. Since $|\sigma_i| > 1$, it is of the form $\sigma x x'$, where for some $e \in E_i, (x' \cap E_i) = (x \cap E_i) \oplus \{e\}$. It can then be easily checked that e' is indeed the $\eta(e)$-view at e, that is $F(e) = e'$, proving surjectivity of F.

The fact that $\lambda(e) = \lambda'(F(e))$ is trivial. Now consider $e_1, e_2 \in E$ such that $e_1 \leq e_2$. Then every schedule δ_2 for $\downarrow e_2$ includes as a prefix a schedule δ_1 for $\downarrow e_1$. Since $\delta_1 \preceq \delta_2$, $\epsilon(\delta_1) \leq' \epsilon(\delta_2)$, that is, $F(e) \leq' F(e')$, as required.

On the other hand, suppose $e_1, e_2 \in E$ such that $F(e_1) \leq' F(e_2)$. Again let $F(e_1)$ be an i-event occurrence in $\mathcal{T}_{\mathcal{E}}$ and $F(e_2)$ a j-event occurrence. Then if $F(e_2) =< \rho_1, ..., \rho_n >$, then ρ_j is of the form $\rho x x'$ where $x' \cap E_j = (x \cap E_j) \oplus \{e_2\}$, and ρ_i is of the form $\rho' x_1 x_2 \rho'' x_3 x_4$, where $x_2 \cap E_i = (x_1 \cap E_i) \oplus \{e_1\}, x_2 \subseteq x_3, x_4 \cap E_i = (x_3 \cap E_i) \oplus \{e_3\}$, and e_3 is the i-maximal event in $\downarrow e_2$. Thus we get $e_1 \leq e_3 \leq e_2$, and thus $e_1 \leq e_2$. Thus F is order-preserving too, and we have the result. $\square$

We now consider the tour in the other direction : given an STS $\mathcal{T}$, we can associate an SS $\mathcal{E}_{\mathcal{T}}$ with it, and with this SS we can associate a STS $\mathcal{T}_{\mathcal{E}_{\mathcal{T}}}$. What is the relationship between $\mathcal{T}$, and $\mathcal{T}_{\mathcal{E}_{\mathcal{T}}}$? Obviously, we cannot again hope for an isomorphism, but we can expect an unfolding, and we define this first.

Definition 4.2 *Let $TS = (S, \rightarrow, s_0)$ and $TS' = (S', \rightarrow', s'_0)$ be labelled transition systems over the same alphabet Σ. We say that TS' is a* simulation *of TS iff there exists a map $f : S' \rightarrow S$: such that*

(i) $f(s'_0) = s_0$,

(ii) *for $s'_1, s'_2 \in S'$, if $s'_1 \xrightarrow{a}{}' s'_2$ then $f(s'_1) \xrightarrow{a} f(s'_2)$,*

(iii) *for $s_1, s_2 \in S$ such that $s_1 \xrightarrow{a} s_2$, if there exists s'_1 such that $f(s'_1) = s_1$, then there exists s'_2 such that $f(s'_2) = s_2$ and $s'_1 \xrightarrow{a} s'_2$.*

A simulation is said to be strong *if it is also surjective. A map which only satisfies conditions (i) and (ii) above is called a* weak *simulation.*

Note that a weak simulation only says that the behaviour in TS' is generated by the structure of TS. The latter might in addition have many states which have no representatives at all in S', and even when f happens to be onto, we might well have $s_1 \xrightarrow{a} s_2$ in TS, and $f(s'_1) = s_1$, but no s'_2 such that $f(s'_2) = s_2$.

Definition 4.3 *Let $\mathcal{T} = (TS_1, \ldots, TS_n)$ and $\mathcal{T}' = (TS'_1, \ldots, TS'_n)$ be synchronizing transition systems over the same distributed alphabet $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$. We say that $\mathcal{T}'$ is a* local (weak / strong) simulation *of $\mathcal{T}$ iff for every $i \in Loc$, TS' is a (weak / strong) simulation of TS.*

For the rest of this section, fix an STS (over $\widetilde{\Sigma}$), $\mathcal{T} = (TS_1, \ldots, TS_n)$, where each TS_i is a tuple $TS_i = (Q_i, \rightarrow_i, q_i^0)$. Let $\mathcal{E}_{\mathcal{T}} = (E, \leq, \lambda)$, and $\mathcal{T}_{\mathcal{E}_{\mathcal{T}}} = (TS'_1, \ldots, TS'_n)$, denoted $\mathcal{T}'$.

We will show that the latter STS weakly simulates the former. The reason why we cannot expect any stronger simulation is this: in general, in a given STS, there can be many transitions which are "useless" in the sense that they are never part of any global run. For instance, consider a system with 2 agents. One has a single transition labelled a from the initial state, the other has no transitions, and the alphabet is the same for both, namely the single letter a. Clearly, the local transition can never be part of any global run. In fact, for this system, the unfolding consists of the trivial run with a single element – the initial state.

For each $i \in Loc$, we need to define the weak simulation f_i from TS'_i to TS_i. Now, TS'_i is nothing but $T_i\mathcal{E}$. Thus, we want a map from LC_i to Q_i. Obviously, we can set $f_i(\emptyset) = q_i^0$, where the latter is the initial state of TS_i. What about $f_i(\downarrow e), e \in E_i$?

Clearly, $e = [\delta]$, for some δ such that $\tau(\delta) = a \in \Sigma_i$. Notice that δ is of the form $\delta' q' \xRightarrow{a} q$ where $q \in Q_i$. Furthermore, for every other δ_1 such that $[\delta] = [\delta_1]$, δ_1 is of the form $\delta'_1 q'' \xRightarrow{a} q$. Thus the function f_i defined by: $f_i(\downarrow e) = q$, where $e = [\delta]$ and δ ends with q, is well-defined.

Now suppose that $\downarrow e_1 \xrightarrow{a} \downarrow e_2$ in $T_i\mathcal{E}$. Then, there exist δ_1, δ_2 such that $e_1 = [\delta_1], e_2 = [\delta_2]$ and $\delta'_2 \preceq \delta_2$ such that $[\delta_1] = [\delta'_2]$. Now suppose that $f_i(\downarrow e_1) = q, f_i(\downarrow e_2) = q'$. By the remarks above, we find that δ'_2 ends with q and δ_2 ends with q'. It is easy to see that δ'_2 is the i-maximal prefix of δ_2 (if not, there would be a prefix δ_3 of δ_2 suffix of δ'_2, and hence we would get $[\delta_3]$ between e_1 and e_2). Now, by definition of global run, for δ_2 to end with $q'' \xrightarrow{a} q'$, we must have $q \xrightarrow{a}_i q'$, as required.

These remarks prove the following lemma, and the theorem below follows.

Lemma 4.4 *The global runs of T are in bijection with those of $\mathcal{T}_{\mathcal{E}_\mathcal{T}}$,*

Theorem 4.5 *Let $\mathcal{T}$ be an STS. Then $\mathcal{T}_{\mathcal{E}_\mathcal{T}}$ is a local weak simulation of $\mathcal{T}$.*

In a sense, the image of f_i above consists of the actual local states reached by computations of the system, and every i-transition which figures in some global run is actually mimicked in the unfolding. Call a transition $q \xrightarrow{a}_i q'$ in TS_i in an STS $\mathcal{T}$ **useful** when there exists a global run $\delta \stackrel{a}{\Longrightarrow} q'$ in $\mathcal{T}$, where q is i-maximal in δ. Call TS_i **complete** when every transition in TS_i is useful, and every state in Q_i reachable from the initial state.

Theorem 4.6 *If $\mathcal{T}$ is an STS made up of complete subsystems, then $\mathcal{T}_{\mathcal{E}_\mathcal{T}}$ is a local strong simulation of $\mathcal{T}$.*

Unfortunately, this fact about useful transitions is not very useful. Deciding whether a transition in a finite STS is useful seems quite hard.

5 Discussion

The preceding section shows that SSs can be captured as products of STSs, where the product construction includes the condition that after any synchronization, the resulting local state for each of the participants in the synchronization must be identical. We can read this result as a way of decomposing synchronization structures.

Considered as subclasses of prime event structures, the structures we have defined, namely SSs, have an important restriction: the conflict relation between event occurrences is generated by local conflict. This rules out specification of a priori global conflict iin the system. Consider the prime event structure with 2 events e_1 and e_2, labelled a and b respectively, over the distributed alphabet $(\{e_1\}, \{e_2\})$, where e_1 and e_2 are in conflict. This cannot be obtained as an SS.

As it turns out, the construction given in the paper can be carried out for such structures as well, provided we allow STSs to have **multiple initial states**. Formally, we can define a *trace event structure* over a distributed alphabet $\widetilde{\Sigma}$ to be a λ-labelled finitary prime event structure $(E, \leq, \#)$ such that whenever for concurrent events e_1 and e_2, $loc(\lambda(e_1)) \cap loc(\lambda(e_2)) = \emptyset$, and whenever $e_1 \lessdot e_2, loc(\lambda(e_1)) \cap loc(\lambda(e_2)) \neq \emptyset$. On the other hand, we can define a *trace STS* to be a tuple $(TS_1, \ldots, TS_n)$, where $TS_i = (Q_i, \rightarrow_i, I_i)$ is a Σ_i-labelled transition system with $I_i \subseteq Q_i$, and $I_1 = \ldots = I_n$. We can then associate a trace-STS with a trace event structure and vice versa such that results similar to the ones in the last section hold. The details of this construction will be given in the full paper.

The more interesting question is regarding **finite state** STSs. Obviously, we can consider tuples of transition systems above, where each is given as $(Q_i, \rightarrow_i, I_i, F_i)$, and $F_i \subseteq Q_i$ are *final states*. This leads to a notion of accepting languages $L \subseteq \Sigma^*$, where a string is accepted if there is a run in the induced product system from one of the common initial states using that string to a global state where each of the local states is final. It can be easily seen that languages accepted by finite-state STSs in this sense, are regular and I-consistent (where I is the obvious independence relation associated with the distributed alphabet: aIb iff $loc(a) \cap loc(b) = \emptyset$). We believe that the converse is also true, though we do not as yet have a proof. (That would yield a local presentation for Zielonka automata [Zie].)

On a very different track, global runs of STSs can be seen as very natural candidates for temporal logics like TrPTL [Thi], where atomic propositions are interpreted over local ones, and global formulas are built up from local ones. TrPTL crucially makes use of the fact that the local views of agents are identical after synchronization, and this property is reflected in global runs of STSs.

We can also see the states of STSs as states of knowledge of agents and synchronization as perfect knowledge transfer. Can this intuition be made precise in some sense ? In the theory of knowledge in distributed systems, it is customary to define an agent i's knowledge by an equivalence relation $\approx_i$ on the global states of the system. The idea is that at any state s, the agent knows exactly those properties which hold for all states $s' \in [s]_i$. The relation $\approx_i$ can be seen as an indistiguishability relation for agent i.

In the context of distributed systems, knowledge is usually defined by equivalence relations on histories; two histories look the same to an agent if the projections to events which the agent participates in are the same ([HM]). Given such a notion of knowledge, we can go on to study the notion of *implicit group knowledge* — intuitively, a property is implicitly known to a group of agents, if some agent in the group knows it. Communication is necessary for making this knowledge explicit. Synchronous communication not only makes implicit knowledge explicit, but also makes it common knowledge to the members of the group who synchronize. Conversely, achieving this kind of knowledge requires synchronization.

What has all this got to do with STSs ? When we define the notion of knowledge (as above) on runs of STSs, it turns out that $\epsilon(\delta)$ in fact represents the group common knowledge at δ for the group $loc(\tau(\delta))$. This means that the local states of STSs are, in effect, states of implicit group knowledge. We believe that in a precise sense, the *Knowledge Transition Systems* of [KR] (suitably modified to rule out asynchronous communication) correspond to product systems induced by STSs.

Finally, this work needs to be done in a categorical set-up, so that the product construction given here can be seen as a categorical product operation.

References

[BC] Boudol, G. and Castellani, I., "Flow models of distributed computations: event structures and nets", *Rapports de Recherche 1482*, INRIA, July 1991.

[Dro] Droste, M., "Event structures and domains", *Theoretical Computer Science*, vol. 68, 1989, 37-47.

[HM] Halpern, J., and Moses, Y., "Knowledge and Common Knowledge in a Distributed Environment", *JACM*, vol. 37, 1991, 549-578.

[KR] Krasucki, P., and Ramanujam, R., "Knowledge and the ordering of events in distributed systems", *TARK V*, Theoretical Aspects of Reasoning about Knowledge, 1994, 267-283.

[LPRT] Lodaya, K., Parikh, R., Ramanujam, R. and Thiagarajan, P.S., "A logical study of distributed transition systems", *Report* **IMSc/92/07** 1992, to appear in *Information and Computation*.

[LRT] Lodaya, K., and Ramanujam, R., and Thiagarajan, P.S., "Temporal Logics for Communicating Sequential Agents: I", *Int. J. Found. Comp. Sci.*, vol. 3, 2, 1992, 117-159.

[Maz] Mazurkiewicz, A., "Basic notions of trace theory", *LNCS 354*, 1989, 285-363.

[Muk] Mukund, M., "Petri nets and step transition systems", *Int. J. Found. Comp. Sci.*, vol. 3, 4, 1992, 443-478.

[MS] Mukund, M. and Sohoni, M., "Keeping track of the latest gossip: bounded time-stamps suffice", *LNCS 761*, 1993, 388-399.

[NRT] Nielsen, M., Rozenberg, G., and Thiagarajan, P.S., "Behavioural notions for elementary net systems", *Distr. Comput.*, vol. 4, 1990, 45-57.

[Thi] Thiagarajan, P.S., "A trace based extension of propositional linear time temporal logic", *Proc IEEE LICS*, 1994, 438-447.

[WN] Winskel, G., and Nielsen, M., "Models for concurrency", in *Handbook of Logic in Computer Science*, Eds: Abramsky, Gabbay and Maibaum, Oxford Science Publications.

[Zie] Zielonka, W., "Notes on finite asynchronous automata", RAIRO-Inf. Theor. et Appli., vol. 21, 1987, 99-135.