Internal Report TCS-90-3
September, 1990

Models and Logics for True Concurrency

Kamal Lodaya[1], Madhavan Mukund[2],
R. Ramanujam[1], P.S. Thiagarajan[2]

Abstract

A distributed computer system consists of different processes or *agents* that function largely autonomously and coördinate their actions by communicating with each other. In such a situation, actions may be performed by different agents of the system locally, in a concurrent manner.

In this paper, we first discuss formal models of distributed systems in which concurrency is specified *explicitly*, in contrast to more traditional approaches where concurrency is represented *implicitly* as a non-deterministic choice between all possible sequentializations of concurrent actions. This naturally leads to models based on partially-ordered sets of actions rather than sequences of actions and is often called the *true concurrency* approach. The models we focus on are distributed transition systems, elementary net systems and event structures.

In the second half of the paper, we develop a family of logics to specify and reason about the behavioural properties of the models we have described. The logics we define are extensions of temporal logic with new modalities to directly describe concurrency.

This paper is essentially a survey of work done by the authors during the last few years on modelling distributed systems with true concurrency and using logic to reason about these models. The emphasis is on motivating definitions through examples and on presenting major results, without going into too many formal details. We provide pointers to the literature where these details can be found.

[1]The Institute of Mathematical Sciences, Madras 600 113, INDIA.

[2]School of Mathematics, SPIC Science Foundation, 92 G.N. Chetty Road, T. Nagar, Madras 600 017, INDIA.

This report is also published as IMSc/90/12, The Institute of Mathematical Sciences, Madras 600 113, INDIA.

Introduction

The study of distributed systems and computations is an important topic of research in computer science. A distributed system consists of a number of essentially autonomous components that work together to perform a complex task.

A computer network which brings together a heterogeneous collection of computing resources and users dispersed over a wide geographic area is a classic example of a distributed system. Distributed databases constitute yet another class of examples. At a lower level, computer protocols which facilitate efficient and reliable transmission of electronic data and operating systems which coördinate the activities of multiple processes (programs) in the presence of multiple processors can also be viewed as distributed systems. With the advent of VLSI systems, the notion of a distributed system is also becoming relevant at the circuit level.

The *theory* of distributed systems consists of formulating abstract mathematical models of distributed systems and studying the properties of these models. A basic motivation in the study of formal models is to develop tools and techniques using which one can specify, analyze and implement distributed systems. Another goal is to develop formal means for reasoning about the behaviour of distributed systems. This is important because one would like to ensure that a specification is in some sense consistent before one attempts an analysis or an implementation. Even more importantly, one would like to guarantee that a proposed implementation indeed meets the requirements of a specification.

In this paper, we present some of our work in the last few years on modelling distributed systems with true concurrency and using logic to reason about these models. The emphasis is on motivating definitions through examples and on presenting major results. No attempt will be made to go into formal details; we shall provide pointers to the literature where these details can be found.

In the first part of the paper, we introduce three models called distributed transition systems, elementary net systems and event structures. Using these models, we illustrate some of the fundamental features of distributed systems, such as causality, choice and concurrency.

In the second half of the paper, we develop a family of logics to specify and reason about the behavioural properties of the models considered in the first half of the paper.

A Models for True Concurrency

Typically, a distributed system consists of spatially separated processes or agents performing a joint task. The agents function largely autonomously and coördinate their actions by communicating with each other. In such a situation, actions may be performed by different agents of the system locally, in a concurrent manner.

Informally, we say that two events are concurrent if they occur with no *a priori* ordering over their occurrences. This is in contrast to a sequential system in which any two events that occur in a computation must be ordered.

In addition to concurrency, two other aspects are of interest in the theory of distributed systems – causality and choice. Causality refers to the fact that certain events in a distributed system can only occur in a fixed order; for example, a message can be received only after it has been sent. The receipt of a message is said to be causally dependent on the sending of the message.

Choice captures the fact that systems can behave in an indeterminate fashion. In other words, at certain points of the computation, the system may choose between alternative events, leading to different behaviours.

As we shall see, labelled transition systems are simple and convenient models of sequential systems which can explicitly describe causality and choice, but which do not have a natural way of representing concurrency. One way of describing concurrency in the framework of transition systems is in terms of indeterminacy. In this approach, the fact that a set of actions may be performed concurrently is represented by permitting the system to choose between all possible sequentializations of the actions. This approximation of concurrency by *interleaving* is used in various algebraic approaches to the theory of distributed systems such as CCS [Milner 1989], CSP [Hoare 1984] and ACP [Bergstra et al 1984].

Such an implicit representation of concurrency leads to problems in analyzing system behaviour, due to the combinatorial explosion in the number of possible interleavings. We follow an alternative approach, called "true concurrency", where concurrency is represented explicitly in the models.

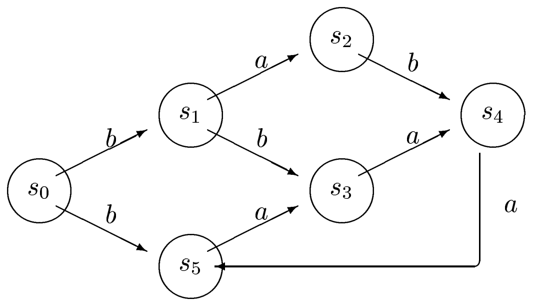

Figure 1: A transition system

Many abstract models of distributed systems have been suggested which explicitly deal with the phenomena of causality, choice and concurrency. Here, we shall consider three of these models — distributed transition systems, elementary net systems and event structures. We shall also discuss a model called communicating sequential agents. This model, based on a restricted class of event structures, captures in a natural way the intuitive picture of a distributed system as a collection of sequential agents coördinating their actions through communication.

1 Distributed Transition Systems

Before discussing models of concurrent systems, let us briefly look at sequential systems. Transition systems are a basic model of sequential systems.

Definition 1.1 *A* (Σ-labelled) transition system *is a triple* $TS = (S, \Sigma, \rightarrow)$ *where*

(i). S *is a set of states.*

(ii). Σ *is a set of actions.*

(iii). $\rightarrow \subseteq S \times \Sigma \times S$ *is the transition relation.*

If $(s, a, s') \in \rightarrow$, then the idea is that the action a can occur at state s and after the execution of a the system assumes the state s'. We shall often write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in \rightarrow$.

Figure 1 is a graphical representation of a transition system. The nodes of the graph represent the states of the system. The edges, labelled by actions from Σ, reflect the transition relation $\rightarrow$.

Clearly the structure of a transition system captures both the basic phenomena present in sequential systems — causality and choice. The transition relation can be used to determine the causal dependancies between system states. Choice is specified by branching in the transition system. In other words, if $s \xrightarrow{a} s'$ and $s \xrightarrow{b} s''$ both belong to the transition relation, then the system at state s can choose between the actions a and b. For example, at s_1 the system shown in Figure 1 can either move by an a to s_2 or move by a b to s_3. In general, different choices available to the system at a state may be labelled by the *same* action. In other words, the behaviour could be nondeterministic. For instance, at s_0 this system can move on b either to s_5 or to s_1.

In this example, starting at s_1, either the action a can occur followed by the action b or the action b can occur followed by the action a. In the interleaving approach to concurrency, this situation often amounts to saying that a and b can occur concurrently at s_1.

However, we would like to maintain a clear distinction between nondeterminism and concurrency. Hence, to describe concurrency in a transition system, we enrich the relation $\rightarrow$ by permitting a transition to be labelled by a finite set of actions from Σ, rather than just by a single action. Thus, we will now have elements in $\rightarrow$ of the form $s \xrightarrow{u} s'$, where u is a finite subset of Σ. The idea is that the actions in u can occur at s with no order over their occurrences. When they have all occurred, the resulting state is s'. The set of actions u is termed a *concurrent step*.

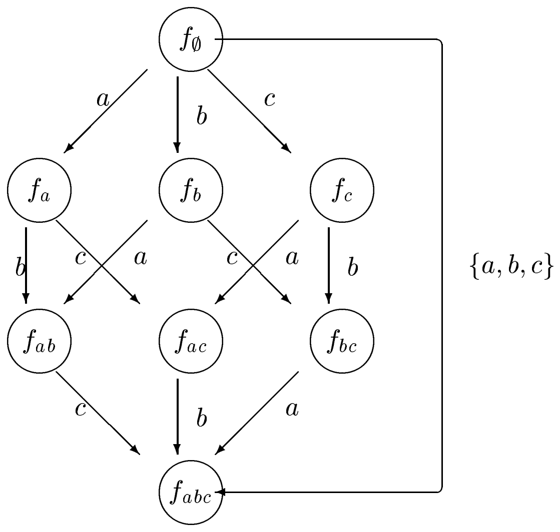

Figure 2: A "cube" generated by a concurrent step

Henceforth, given a set X, $\wp(X)$ denotes the set of subsets of X and $\wp_{fin}(X)$ denotes the set of *finite* subsets of X. We can now formally define distributed transition systems as follows.

Definition 1.2 *A* distributed transition system (dts) *is a triple* $DTS = (S, \Sigma, \rightarrow)$ *where*

(i). S *is a set of states.*

(ii). Σ *is a set of actions.*

(iii). $\rightarrow\ \subseteq S \times \wp_{fin}(\Sigma) \times S$ *is the step transition relation satisfying for all* s, s' *in* S*:*

- *(a)* $s \xrightarrow{\emptyset} s'$ *iff* $s = s'$.
- *(b) for all* $u \in \wp_{fin}(\Sigma)$, *if* $s \xrightarrow{u} s'$ *then there exists a function* $f\colon \wp(u) \rightarrow S$ *such that* $f(\emptyset) = s$, $f(u) = s'$ *and for every* $v_1, v_2 \in \wp(u)$ *with* $v_1 \subseteq v_2$, *it is the case that* $f(v_1) \xrightarrow{v_2 - v_1} f(v_2)$.

We often say that $DTS = (S, \Sigma, \rightarrow)$ is a *dts over* Σ. For convenience, we write $s \xrightarrow{a} s'$ instead of $s \xrightarrow{\{a\}} s'$.

The new definition of $\rightarrow$ is a bit involved because we have to ensure that any non-trivial "substep" of a concurrent step is also performed as a concurrent step. The function f in clause (3.b) is said to define a *u-cube* (from s to s'). The existence of the u-cube guarantees that the mutual independence of the actions in u holds for all the substeps as well. For example, Figure 2 shows the cube generated by a concurrent step consisting of three events. To avoid cluttering up the figure, "interior" arrows such as $f_\emptyset \xrightarrow{\{a,b\}} f_{ab}$ and $f_b \xrightarrow{\{a,c\}} f_{abc}$ have not been drawn.

Figure 3 is an example of a distributed transition system modelling the allocation of a shared resource to different processes within a system. In the example, we have 3 processes P_1, P_2 and P_3 functioning in an operating environment that supports multiprocessing. The resource — say, for example, blocks of memory — is available in "units". There are totally 5 units available. The 3 processes require 2, 3 and 5 units of the resource at a time respectively. In this dts, $\Sigma = \{a_1, a_2, a_3, r_1, r_2, r_3\}$, where a_i denotes that process P_i has been allocated the entire amount of the resource that it needs and r_i denotes that P_i has released the resource it has been allocated. The states of the dts are ordered pairs consisting of the number of unallocated units of the resource available in the system along with the set of processes currently in possession of their required quota of the shared resource.

Thus, at the state $(5, \emptyset)$, no processes are active and all 5 units of the resource are available. At this state, the system can either allocate units of the resource to one of the three processes, or perform a concurrent step allocating resources to both P_1 and P_2. Notice that the transition from $(3, \{P_1\})$ to $(2, \{P_2\})$ can either be performed as a concurrent step $\{a_2, r_1\}$ or by interleaving the two actions. In

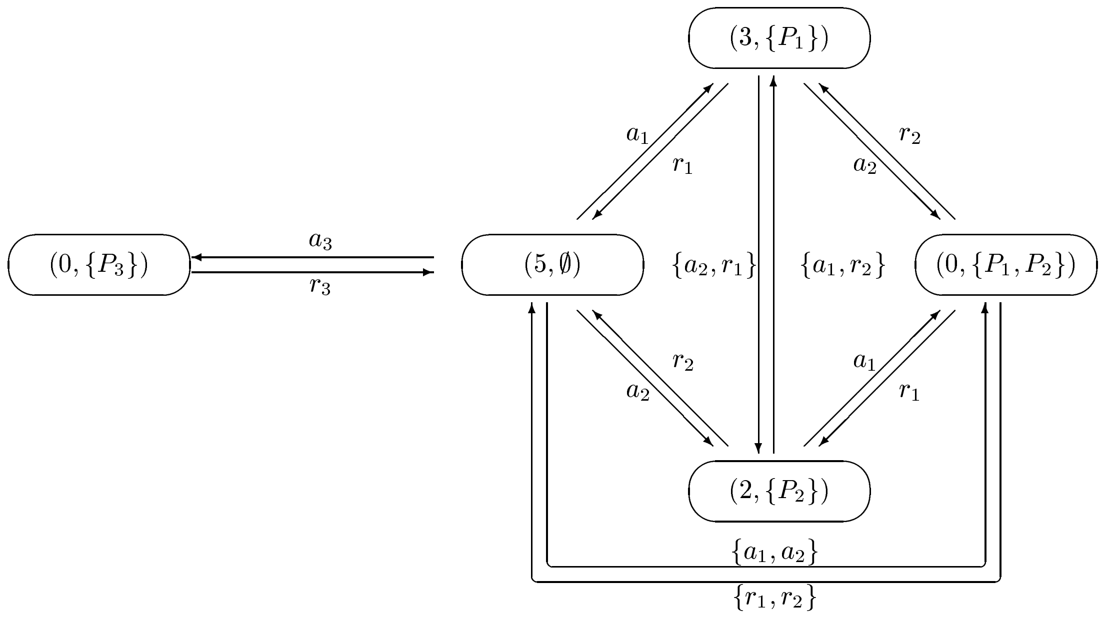

Figure 3: A distributed transition system

one interleaving, however, $(5, \emptyset)$ is reached as an intermediate state, at which point the resource can be allocated to P_3 instead of P_2. Thus, in this case, the effect of the interleavings is not quite the same as that of the concurrent step.

In general, it is important to note that clause (3.b) in Definition 1.2 is merely an implication. The existence of a function from $\wp(u)$ into S which fulfills the stated requirements does not guarantee the existence of a concurrent step. This is line with our philosophy that concurrency should be clearly differentiated from interleaving. As we have seen above, interleavings may permit unintended deviations from the behaviour expected of a concurrent step. In fact, it is possible to have a concurrent step as well as an interleaving of the step performed at a state but leading to two different states.

Finally, we introduce the important notion of reachability in a transition system. Given $TS = (S, \Sigma, \rightarrow)$ we define $\mathcal{R}(s_0)$, the *reachability set* of $s_0 \in S$, as the least subset of S containing s_0 satisfying:

$$\text{If } s \in \mathcal{R}(s_0),\ a \in \Sigma \text{ and } s \xrightarrow{a} s',\ \text{ then } s' \in \mathcal{R}(s_0).$$

Thus, $\mathcal{R}(s_0)$ is the set of states reachable from s_0 in a finite number of steps using $\rightarrow$.

2 Elementary Net Systems

In a distributed transition system, concurrency is explicitly introduced into a transition system by permitting transitions between states via finite sets of actions called concurrent steps. In effect, the notion of a state is left unchanged and the transition relation is enriched to model concurrency.

An alternative way of introducing concurrency into a transition system is to "distribute" the states of the system. The states of a dts correspond to the global states of the concurrent system being modelled by the dts. Rather than regard these global states as indivisible entities, we can break them up into atomic components which can be regarded as the local states of the different processes within the system. The global states of the system can then be characterized in terms of the local states.

By distributing the states of the model in this manner, we can clearly distinguish concurrency from choice without having to define a transition relation involving sets of actions as in a dts. Instead, the transition relation is designed to capture the fact that the change of state accompanying each event occurrence in the system is "localized" to those processes that actually participate in the event. As a result, when an event occurs, only specific local components of the global state are affected, leaving the rest of the components untouched. Thus, two events that are enabled at a global state of the system can occur concurrently if the local states that they affect are disjoint. On the other hand, if the local states

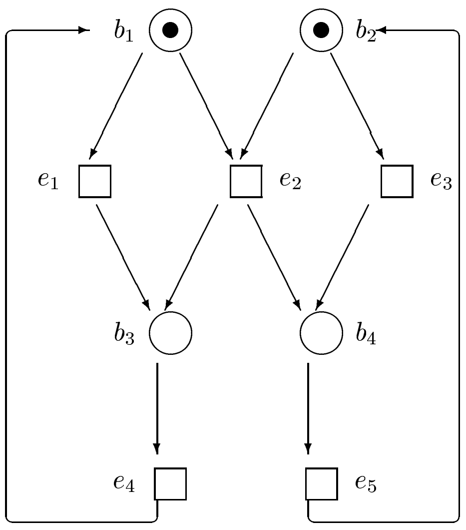

Figure 4: An elementary net system

affected by the two events overlap, they cannot both occur in the same computation at that state and so a choice must be made between them.

Net theory deals with models of concurrent systems based on this approach. Here we describe elementary net systems, which are a basic model in this theory. We begin with the definition of a net.

Definition 2.1 *A* net *is a triple* $N = (B, E, F)$ *where* B *and* E *are disjoint sets and* $F \subseteq (B \times E) \cup (E \times B)$ *satisfies:*

$$\forall x \in B \cup E : \exists y \in B \cup E : (x, y) \in F \text{ or } (y, x) \in F.$$

The elements of B are called *conditions* and are used to denote atomic local states. The elements of E are called *events* and are used to represent atomic actions. The *flow relation* F models a *fixed* neighbourhood relation between the conditions and events of a system. This flow relation determines the way in which the atomic actions affect the atomic local states. The restriction on F in the definition of a net ensures that there are no isolated conditions or events.

We can now define an elementary net system as follows.

Definition 2.2 *An* elementary net system *is a quadruple* $\mathcal{N} = (B, E, F, c_{in})$ *where*

(i). $N_{\mathcal{N}} = (B, E, F)$ *is a net called the underlying net of* $\mathcal{N}$.

(ii). $c_{in} \subseteq B$ *is the* initial case.

Figure 4 is an example of an elementary net system. We have used the conventional graphical notation for nets — conditions are represented by circles, events by boxes and the flow relation by directed arcs. The "marked" conditions denote the initial case c_{in}.

For e in E the conditions "pointing into" e via F are called the *pre-conditions of* e and are denoted by ${}^\bullet e$. Similarly, the conditions "pointing away" from e via F are called the *post-conditions of* e and are denoted by $e^\bullet$. More formally we have

$$\begin{aligned} {}^\bullet e &\stackrel{\text{def}}{=} \{b \mid (b, e) \in F\} \\ e^\bullet &\stackrel{\text{def}}{=} \{b \mid (e, b) \in F\} \end{aligned}$$

A state of a net system, called a *case*, consists of a set of conditions $c \subseteq B$. The conditions in c are said to *hold* when the system is at the case c. Thus, c_{in} is the set of conditions that hold when the system starts up.

The system moves from one case to another through the occurrence of events from E. An event can occur at a case iff all its pre-conditions hold and none of its post-conditions do at the case. When an event occurs all its pre-conditions cease to hold and all its post-conditions begin to hold.

In graphical terms, an event e can occur at a case c iff all the conditions pointing into e are "marked" at c and none of the conditions pointing away from e are. For example, in Figure 4, the event e_1 can occur at the initial case $c_{in} = \{b_1, b_2\}$. When e_1 occurs, we "unmark" all the pre-conditions of e_1 and "mark" all its post-conditions, leaving the other conditions in c_{in} untouched. Thus, after the occurrence of e_1, the system is at the case $\{b_2, b_3\}$.

We can formalise this by defining $\rightarrow_N \subseteq \wp(B) \times E \times \wp(B)$, the *(elementary) transition relation generated by the net* $N = (B, E, F)$ as follows.

$$\rightarrow_N \;\; = \{(x, e, x') \mid x - x' = {}^\bullet e, \; x' - x = e^\bullet\}$$

Using this transition relation, we can associate a transition system with an elementary net system as follows

Definition 2.3 *Let $\mathcal{N} = (B, E, F, c_{in})$ be a net system.*

(i). $C_\mathcal{N}$, the state space *of $\mathcal{N}$, is the least subset of $\wp(B)$ containing c_{in} such that if $c \in C_\mathcal{N}$ and $(c, e, c') \in \;\rightarrow_{N_\mathcal{N}}$ then $c' \in C_\mathcal{N}$.*

(ii). $TS_\mathcal{N} = (C_\mathcal{N}, E, \rightarrow_\mathcal{N})$ is the transition system associated with *$\mathcal{N}$, where $\rightarrow_\mathcal{N}$ is $\rightarrow_{N_\mathcal{N}}$ restricted to $C_\mathcal{N} \times E \times C_\mathcal{N}$.*

For the net system shown in Figure 4, $\{\{b_1, b_2\}, \{b_1, b_4\}, \{b_2, b_3\}, \{b_3, b_4\}\}$ is its state space.

Let $\mathcal{N} = (B, E, F, c_{in})$ be a net system with $c \in C_\mathcal{N}$ and $e \in E$. Then e is said to be *enabled at* c — denoted $c[e\rangle$ — iff there exists $c' \in C_\mathcal{N}$ such that $c \xrightarrow{e} c'$, where as usual $c \xrightarrow{e} c'$ abbreviates $(c, e, c') \in \;\rightarrow_\mathcal{N}$.

As we had mentioned at the beginning of this section, we can clearly separate concurrency from choice once we have distributed the global states of a transition system into local components.

Let $\mathcal{N} = (B, E, F, c_{in})$ be a net system and $e, e' \in E$. We say that e and e' can occur concurrently at a case c — denoted $c[\{e, e'\}\rangle$ — iff $c[e\rangle$ and $c[e'\rangle$ and $({}^\bullet e \cup e^\bullet) \cap ({}^\bullet e' \cup e'^\bullet) = \emptyset$. Thus, e and e' can occur concurrently at a case if they can occur individually and their "neighbourhoods" are disjoint.

Similarly we can define the notion of *conflict*. Let $\mathcal{N}$ be a net system as above with $e, e' \in E$. e and e' are said to be in conflict at a case c iff $c[e\rangle$ and $c[e'\rangle$ but *not* $c[\{e, e'\}\rangle$. Thus, if e and e' are in conflict at c, it means that they are both individually enabled at c, but they cannot occur *together* at c. For the computation to proceed, the conflict must be *resolved* by making a (nondeterministic) choice between the two events.

The definition of $\rightarrow_\mathcal{N}$ is designed to ensure that the notion of change of state in an elementary net system is fairly restricted.

First, notice that an event must cause the same change in the system state whenever it occurs; its pre-conditions cease to hold and its post-conditions begin to hold. Thus, if $c_1 \xrightarrow{e} c_2$ and $c_3 \xrightarrow{e} c_4$ are both possible in a net system, then it must be the case that $c_1 - c_2 = c_3 - c_4 = {}^\bullet e$ and $c_2 - c_1 = c_4 - c_3 = e^\bullet$.

Further, to determine whether an event e is enabled at a case c, it is sufficient to look at the conditions contained in ${}^\bullet e$ and $e^\bullet$. e is enabled at c iff ${}^\bullet e \subseteq c$ and $e^\bullet \cap c = \emptyset$ — no "side-conditions" are involved in the enabling of an event.

Finally, it turns out that the transition system $TS_\mathcal{N}$ associated with a net system $\mathcal{N}$ is *deterministic*; that is, $c \xrightarrow{e} c'$ and $c \xrightarrow{e} c''$ implies that $c' = c''$. To connect up with other approaches to the theory of distributed systems, nondeterminism can be introduced into $TS_\mathcal{N}$ by *labelling* the events in E. We shall come back to this point later in this section.

Let us consider an example of modelling a distributed system using an unlabelled elementary net system. Consider the problem of sharing resources in a distributed system. Suppose that there are two processes P_1 and P_2 in the system which require access to a common resource r. Suppose that r can be used by only one process at a time – r could, for instance, be a printer. Then, when one of the processes is granted access to r, the other process should be prevented from accessing r till the first process releases it. This will ensure that at any state during a computation of the system, at most one process can actually be using that resource.

Figure 5 models a solution to this problem of *mutual exclusion*. In this net system the process P_i, $i = 1, 2$, is represented by the conditions $\{b_0^i, b_1^i, b_2^i, b_3^i\}$ and the events $\{e_0^i, e_1^i, e_2^i, e_3^i\}$. Each process is modelled as a simple loop consisting of four events — getting access to r (e_0^i), utilizing r (e_1^i), releasing r (e_2^i) and performing some internal computations not involving r (e_3^i). At the initial case, both processes are waiting for access to r. The additional condition a functions as an arbitrator which enforces mutual

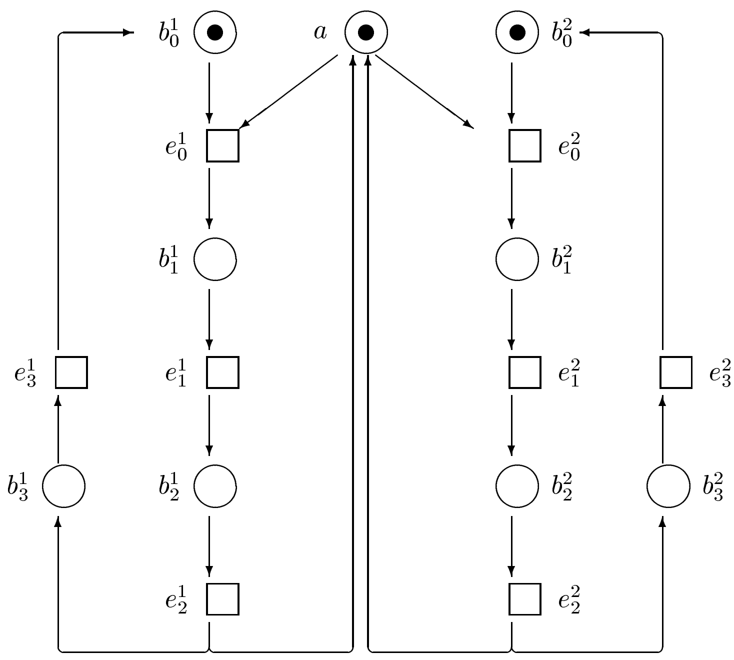

Figure 5: Mutual exclusion

exclusion of access to r. For example, suppose that e_0^2 occurs initially, giving P_2 access to r. Since a ceases to hold e_0^1 is no longer enabled. Thus, P_1 can gain access to r only after P_2 releases r by the occurrence of e_2^2. It is easy to check that b_1^1 and b_1^2 can never hold together in this net system. On the other hand, the conditions b_3^1 and b_3^2 can hold at the same case — that is, the events e_3^1 and e_3^2 which do not involve the use of r can occur concurrently in this system.

Finally, we show that we can describe the behaviour of elementary net systems in terms of distributed transition systems. Consider an elementary net system $\mathcal{N} = (B, E, F, c_{in})$. The transition system $TS_{\mathcal{N}}$ contains information about the causality and conflict present in $\mathcal{N}$. To describe the concurrency present in $\mathcal{N}$, it is sufficient to augment $TS_{\mathcal{N}}$ with additional transitions labelled by concurrent steps, as follows.

We first extend the notion of a pair of events being concurrently enabled at a case to a set of events. Let $u = \{e_1, e_2, \ldots, e_n\}$ be a finite subset of E. We say that u is concurrently enabled at a case $c \in C_{\mathcal{N}}$ — denoted $c[u\rangle$ — iff $c[e_i\rangle$ for each $e_i \in u$ and, further, $c[\{e_1, e_2\}\rangle$ for every pair of distinct events $e_1, e_2 \in u$.

We can then define the step transition relation $\Rightarrow_{\mathcal{N}}$ as follows.

$$\Rightarrow_{\mathcal{N}} \;=\; \{(c, u, c') \mid c, c' \in C_{\mathcal{N}}, c[u\rangle \text{ and } c - c' = {}^\bullet u, c' - c = u^\bullet\}$$

Here ${}^\bullet u$ and $u^\bullet$ denote the unions of the pre-conditions and post-conditions of the events contained in u. Note that $\rightarrow_{\mathcal{N}}$ is "included" in $\Rightarrow_{\mathcal{N}}$ in the sense that if $(c, e, c') \in\ \rightarrow_{\mathcal{N}}$ then $(c, \{e\}, c') \in\ \Rightarrow_{\mathcal{N}}$. We can then immediately establish the following.

Proposition 2.4 *$DTS_{\mathcal{N}} = (C_{\mathcal{N}}, E, \Rightarrow_{\mathcal{N}})$ is a distributed transition system over E.*

It is easy to verify that the concurrency and choice present in $\mathcal{N}$ is precisely captured by the dts $DTS_{\mathcal{N}}$.

However, notice that this dts is deterministic, for the same reason that the transition system $TS_{\mathcal{N}}$ is. As we had mentioned earlier, we can introduce nondeterminism by labelling the events.

Definition 2.5 *A* Σ-labelled elementary net system *is a pair $\mathcal{N}_\Sigma = (\mathcal{N}, \phi)$, where $\mathcal{N} = (B, E, F, c_{in})$ is an elementary net system, called the underlying net system of $\mathcal{N}_\Sigma$, Σ is a set of labels and $\phi: E \rightarrow \Sigma$ is the labelling function.*

The notions we have developed for net systems can be transported to labelled net systems in the obvious way. To represent the behaviour of a labelled net system $\mathcal{N}_\Sigma$ as a dts, we can define $DTS_{\mathcal{N}_\Sigma}$ to be the dts over Σ obtained by using the labelling function ϕ to rename the actions in $DTS_{\mathcal{N}}$, the dts over E generated by the underlying net system $\mathcal{N}$.

However, in general we need to place a restriction on the labelling function in order to get a neat translation from labelled net systems to dts's. In a dts, we have restricted concurrent steps to be *sets* of actions. On the other hand, a labelled net system $\mathcal{N}_\Sigma$ may generate a concurrent step in $DTS_{\mathcal{N}_\Sigma}$ where two distinct events in the step have the *same* label. To avoid dealing with multisets in concurrent steps that arise in this fashion, we require that events which can occur concurrently in the underlying net system $\mathcal{N}$ have distinct labels.

Let $\mathcal{N}_\Sigma = (B, E, F, c_{in}, \phi)$ be a Σ-labelled net system. The labelling function ϕ is said to be *co-injective* if it satisfies the following condition.

$$\forall e_1, e_2 \in E : (\exists c \in C_{\mathcal{N}} : c[\{e_1, e_2\}\rangle) \text{ implies } \phi(e_1) \neq \phi(e_2).$$

Proposition 2.6 *Let $\mathcal{N}_\Sigma = (\mathcal{N}, \phi)$ be a Σ-labelled elementary net system, where $\mathcal{N} = (B, E, F, c_{in})$, such that ϕ is co-injective. Then $DTS_{\mathcal{N}_\Sigma} = (C_{\mathcal{N}}, \Sigma, \Rightarrow_{\mathcal{N}_\Sigma})$ is a dts over Σ, where*

$$\Rightarrow_{\mathcal{N}_\Sigma} = \{(c, \phi(u), c') \mid (c, u, c') \in \Rightarrow_{\mathcal{N}}\}.$$

3 Event Structures

To reason about the behaviour of a distributed transition system or an elementary net system, we have to examine all the computations of the underlying "machines" defined by the model. For this, it is convenient to work with an abstract representation of the *entire* behaviour of the system. This behavioural description should include information about all the computations of the system, explicitly identifying the causal dependancies and concurrency present within each computation. In addition, it should also have a way of describing the branching points in the system behaviour.

Before discussing behavioural representations of concurrent systems, let us first go back to sequential transition systems. A computation of a sequential transition system $TS = (S, \Sigma, \rightarrow)$ starting at some state $s_0 \in S$ is an alternating sequence of actions and states which obeys the transition relation $\rightarrow$. We shall restrict our attention to the *maximal* computations of the system — those that cannot be extended by performing any more actions. Thus, a maximal computation is a finite sequence just in case a state is reached at the end of the sequence from which no transition is possible; otherwise, it is an infinite sequence.

A natural way to group together the sequences which correspond to computations of $TS = (S, \Sigma, \rightarrow)$ starting from s_0 is in the form of a tree. The nodes of the tree are labelled by states from S and the edges are labelled by actions from Σ. The root node is labelled by the initial state s_0. Each maximal path in the tree now corresponds to a computation of the system. The branching points in the tree are the states where the system makes choices between different possible actions.

In the case of models exhibiting concurrency, the situation is more complicated. A computation of such a system is a partially ordered set of actions, not a simple sequence, so we need a more sophisticated method of collecting all the computations together in a single structure. An elegant way of achieving this is to use event structures. Event structures are behavioural models of distributed systems in which causality, concurrency and choice (conflict) are represented explicitly.

Prime event structures, introduced in [Nielsen et al 1980], are the simplest type of event structures. They have a rich theory and are closely related to both net systems and domains. Since we deal only with prime event structures in this paper, henceforth we shall simply call them event structures.

Definition 3.1 *An* event structure *is a triple $ES = (E, \leq, \#)$ where*

(i). E is a set of event occurrences.

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

(iii). $\# \subseteq E \times E$ is an irreflexive and symmetric conflict relation.

(iv). $\#$ is inherited via $\leq$ in the sense that $e_1 \# e_2 \leq e_3$ implies that $e_1 \# e_3$ for every e_1, e_2, e_3 in E.

An element of E represents the occurrence of an event within a specific context. Thus, if the same event can occur in different contexts, "copies" of it will be present in the event structure. This is why we have called the elements of E event occurrences rather than events.

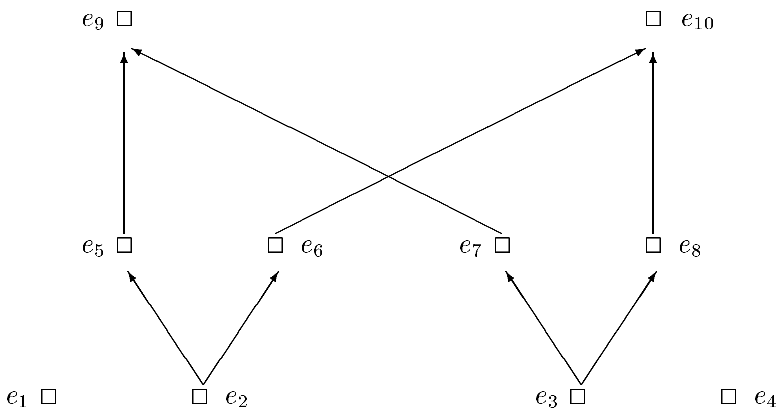

Figure 6: An event structure

If $e_1 \leq e_2$, then e_2 is causally dependent on e_1. Thus, in any computation of the system, e_2 can occur only if e_1 has already occurred. As usual we let $\geq$ stand for $\leq^{-1}$.

The # relation identifies pairs of events which are inconsistent with each other and hence cannot both occur during the same computation. The last clause of the Definition 3.1 ensures that if $e_1 \# e_2$ then events that are causally dependent on e_1 are in conflict with events that are causally dependent on e_2 — in other words, the inconsistency of e_1 and e_2 is inherited by events that follow these two events.

Two events that are neither causally related nor in conflict with each other can both occur within a computation with no order over their occurrence. We can thus define the concurrency relation *co* in an event structure $ES = (E, \leq, \#)$ in terms of $\leq$ and # as follows.

$$co \stackrel{\text{def}}{=} E \times E - (\leq \cup \geq \cup \#).$$

Notice that *co*, like #, is irreflexive and symmetric. Clearly, every pair of *distinct* events in an event structure belongs to exactly one of the four relations $\{\leq, \geq, \#, co\}$.

It is useful to define one more auxiliary relation. Let $ES = (E, \leq, \#)$ be an event structure and $e, e' \in E$. Then

$$e \#_\mu e' \stackrel{\text{def}}{=} e \# e' \text{ and } \forall e_1, e_1' \in E : [\, e_1 \leq e \text{ and } e_1' \leq e' \text{ and } e_1 \# e_1' \\ \text{implies } e_1 = e \text{ and } e_1' = e' \,].$$

$\#_\mu$ identifies the minimal elements (under $\leq$) of the # relation and is hence called the *minimal conflict* relation. $\#_\mu$ identifies the actual branching points in the behaviour where choices are made between conflicting events. This "basic" conflict then propagates to causally related events and "generates" other conflicts.

Figure 6 is an example of an event structure. The squiggly lines represent the $\#_\mu$ relation. The causality relation is shown in the form of the associated Hasse diagram. The # relation is then uniquely determined by the last part of Definition 3.1. In this event structure, $e_1 \# e_6$ because $e_1 \#_\mu e_2 \leq e_6$. It is also easy to see that $e_6 \; co \; e_7$.

The states of an event structure are called configurations. A configuration identifies a set of events that have occurred "so far". An event can occur only if all the events in its past have occurred. Two events that are in conflict can never both occur in the same stretch of behaviour. Before formalizing these notions it will be convenient to adopt the following notation.

Let $ES = (E, \leq, \#)$ be an event structure and $X \subseteq E$. Then $\downarrow X = \{e' \mid \exists e \in X : e' \leq e\}$. For the singleton $\{e\}$, we shall write $\downarrow e$ instead of $\downarrow \{e\}$.

Definition 3.2 *Let $ES = (E, \leq, \#)$ be an event structure and $c \subseteq E$. Then c is a* configuration *iff*

(i). $c = \downarrow c$ *(left-closed)*

(ii). $(c \times c) \cap \# = \emptyset$ *(conflict-free)*

For the event structure shown in Figure 6, $\{e_2, e_5, e_6\}$ is a configuration. $\{e_2, e_5, e_{10}\}$ is not a configuration because it is not left-closed and $\{e_3, e_7, e_8\}$ is not a configuration because it is not conflict-free.

We are particularly interested in a restricted subset of configurations called local configurations. The notion of a local configuration is based on a simple but crucial observation which lies at the heart of the theory of event structures [Nielsen et al 1980].

Proposition 3.3 *Let $ES = (E, \leq, \#)$ be an event structure and $e \in E$. Then $\downarrow\!e$ is a configuration.*

We now define $LC_{ES} = \{\downarrow\!e \mid e \in E\}$ to be the set of *local configurations* of the event structure $ES = (E, \leq, \#)$.

We do so because a (general) configuration $c \subseteq E$ can be viewed as a global state of the system. Parts of a global configuration may change independent of each other, due to the spatial separation and the partial autonomy of the individual agents in the system being modelled by the event structure. A finite global configuration c is completely characterized by specifying the maximal events (with respect to $\leq$) which belong to c. Each local configuration $\downarrow\!e$ corresponding to a maximal event $e \in c$ can be regarded as a local state which contributes to the global state at c.

When we reason about the behaviour of an event structure, we would like to make assertions about properties that are satisfied by the global configurations — that is, properties that hold at the global states of the system. However, a global state can be *completely* described in terms of all the local states that are part of that global state. Thus, we shall restrict ourselves to specifying properties at the local configurations. Using combinations of these assertions, we can describe global configurations of the event structure. Further, the assertions that we can make about a global configuration are tied down to the assertions that we can make about the local configurations that constitute the global configuration. This will become clearer in the second part of the paper where we discuss how to specify properties of distributed systems.

As we had mentioned at the beginning of this section, an event structure is a single entity which describes all the computations of a distributed system. Thus, we need a means of "extracting" individual computations from an event structure. Since a configuration represents a set of events that have happened so far, in general an arbitrary configuration represents a partial computation of the system. If we consider configurations which are maximal (with respect to inclusion) we obtain the maximal computations of the event structure. We call these the *runs* of the event structure. It is easy to verify the following characterization of runs. Let $r \subseteq E$. Then r is a run iff

$$\forall e \in E : e \in r \text{ iff } \forall e' \in E : e \ \# \ e' \text{ implies } e' \notin r$$

Next, let us look at some useful restrictions on event structures. We begin with the auxiliary relation $\#_\mu$. In general, there may be events in $\#$ whose inconsistency cannot be traced back to a pair of events in $\#_\mu$ — a typical example consists of two infinite descending chains of events in $\#$ with each other. We would like to rule out such structures, since they model behaviours which are intuitively infeasible. We can therefore restrict our attention to well branching event structures.

Definition 3.4 *Let $ES = (E, \leq, \#)$ be an event structure. ES is* well branching *iff*

$$\forall e, e' \in E : e \ \# \ e' \text{ implies } \exists e_1, e_1' \in E : e_1 \leq e \text{ and } e_1' \leq e' \text{ and } e_1 \ \#_\mu \ e_1'.$$

Well branching is a fairly weak restriction. A stronger and more useful restriction is that of *finitariness*. An event structure $ES = (E, \leq, \#)$ is said to be finitary in case $\downarrow\!e$ is a finite set for every $e \in E$. Finitariness captures the important fact that in any realizable system, an event can be causally dependent on only a finite set of events. An event with an infinite past can never actually occur.

There is a systematic way of describing the behaviour of elementary net systems using finitary event structures. To do this, we require *labelled* event structures. A labelled event structure is a pair $ES_\Sigma = (ES, \phi)$ where $ES = (E, \leq, \#)$ is an event structure and $\phi : E \to \Sigma$ is a labelling function.

Constructing a labelled finitary event structure describing the behaviour of a net system involves an intermediate stage where the net system is "unfolded" to generate an acyclic structure. The details are a bit involved and can be found in [Nielsen et al 1980], [Thiagarajan 1990]. We shall merely present an example.

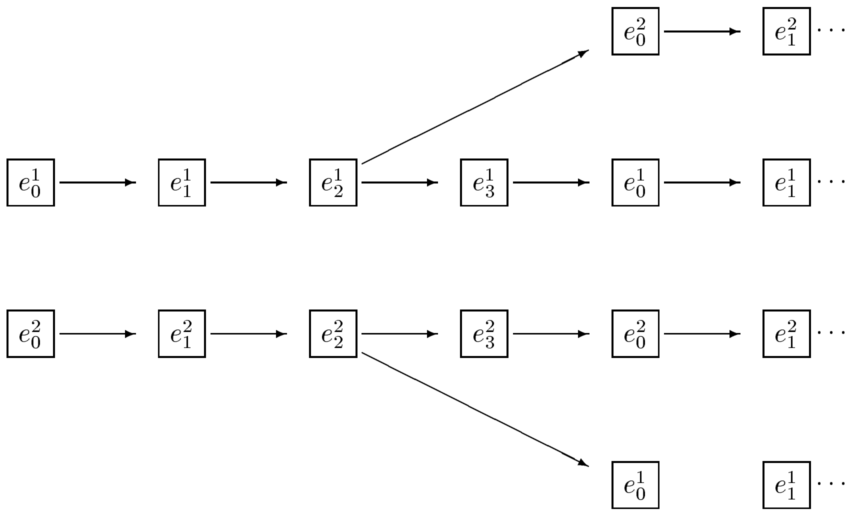

Figure 7: A labelled event structure

Consider the elementary net system in Figure 5 modelling mutual exclusion. The labelled event structure in Figure 7 describes the behaviour of this system. In this case, the event occurrences in the event structure are labelled by the events of the net system.

Given a finitary event structure ES, we can construct a dts DTS_{ES} which exhibits the same behaviour as ES. Let $\mathcal{C}_{ES}^{fin}$ denote the set of *finite* configurations of the *finitary* event structure $ES = (E, \leq, \#)$. We can define the step transition relation $\rightarrow_{ES} \subseteq \mathcal{C}_{ES}^{fin} \times \wp_{fin}(E) \times \mathcal{C}_{ES}^{fin}$ as follows:

$$\begin{aligned}\rightarrow_{ES} \;= \{(c, u, c') \mid\, & c \cap u = \emptyset \text{ and } c \cup u = c' \text{ and} \\ & \forall e_1, e_2 \in u : e_1 \neq e_2 \text{ implies } e_1 \;co\; e_2\}\end{aligned}$$

Proposition 3.5 *$DTS_{ES} = (\mathcal{C}_{ES}^{fin}, E, \rightarrow_{ES})$ is a dts over E.*

As in the case of elementary net systems, it turns out that DTS_{ES} is always deterministic. Once again, we can use labelled event structures to permit nondeterminism in this dts. As before, we have to restrict the labelling to be *co-injective* to rule out multisets in concurrent steps. In other words, given $ES_\Sigma = (E, \leq, \#, \phi)$, we require that for every $e_1, e_2 \in E : e_1 \;co\; e_2$ implies $\phi(e_1) \neq \phi(e_2)$. We then have the following result.

Proposition 3.6 *Let $ES_\Sigma = (ES, \phi)$ be a Σ-labelled event structure where ϕ is a co-injective labelling function. Then $DTS_{ES_\Sigma} = (\mathcal{C}_{ES}^{fin}, \Sigma, \Rightarrow_{ES_\Sigma})$ is a dts over Σ where*

$$\Rightarrow_{ES_\Sigma} \;= \{(c, \phi(u), c') \mid (c, u, c') \in \;\rightarrow_{ES}\}.$$

4 Communicating Sequential Agents

In an event structure, the entire behaviour of a distributed system is specified as a single entity. Individual computations of the system can be identified using the notion of a run. However, no further information is provided about the structure of the system.

Consider a distributed system consisting of a finite set of sequential agents performing a joint task, using communication to coördinate their activities. When reasoning about the behaviour of such a system, it is convenient to associate the events occurring in the system with the agents involved in the events. This can be captured by restricting event structures to a model called communicating sequential agents (csa's).

Let $\mathbf{N}$ denote the set of natural numbers $\{1,2,3 \ldots\}$. We shall use elements of $\mathbf{N}$ as names for the agents in our system.

Definition 4.1 *A system of* communicating sequential agents (csa) *is a triple $CSA = (E, \leq, \eta)$, where*

(i). E is a non-empty set of event occurrences.

(ii). $\leq$ is a partial order on E called the causality relation.

(iii). $\eta : E \to \wp_{fin}(\mathbf{N})$ is a naming function assigning to each e in E a non-empty finite subset of $\mathbf{N}$.

(iv). Let $E_j = \{e \mid e \in E$ and $j \in \eta(e)\}$. Then, for every e in E:

$$\forall j \in \mathbf{N} : {\downarrow}e \cap E_j \textit{ is totally ordered by } \leq .$$

We interpret $j \in \eta(e)$ as the agent j participating in the event e. Thus $\eta(e) = \{1, 2\}$ can stand for a synchronization "handshake" between agents 1 and 2.

The poset $(E_j, \leq_j)$, where $\leq_j$ is $\leq$ restricted to $E_j \times E_j$, represents the local behaviour of agent j in CSA. Usually, we say "agent j" to denote this poset.

As in an event structure, if $e_1 \leq e_2$ then e_2 causally depends on e_1; in no run of CSA can e_2 occur without e_1 having occurred earlier.

To separate concurrency from conflict, both the causality relation $\leq$ and the naming function η are used. In a csa, each agent is defined to be sequential. Thus, given any two events e and e' which both involve the same agent – that is $\eta(e)$ and $\eta(e')$ are *not* disjoint — e and e' must either be causally related or in conflict. So if e and e' are incomparable with respect to $\leq$ and $\eta(e) \cap \eta(e') \neq \emptyset$, then e and e' are in conflict.

The motivation for the last condition in Definition 4.1 should now be clear: we do not wish an event occurrence to causally depend upon conflicting event occurrences. This condition also implicitly ensures that the basic conflict in the system is generated within agents — in effect, choices are made locally by individual agents and then propogated across agents via $\leq$.

On the other hand, if two events e and e' are unordered and their combined past does not contain any conflicting events then they must be concurrent. Since choices are assumed to be made locally, it is sufficient to check that for each agent j, the combined past of e and e' does not have incomparable events involving j. In other words, if $({\downarrow}e \cup {\downarrow}e') \cap E_j$ is totally ordered by $\leq$ for every j, then the two events e and e' are concurrent.

If $e \in E_j$, the local state ${\downarrow}e$ includes the local history of agent j as well as the "latest" local histories of all other agents with which j has communicated upto this state. Let $LC_{CSA} = \{{\downarrow}e \mid e \in E\}$ be the set of local states of CSA.

By suitably restricting the naming function η, we can capture interesting subclasses of csa's.

The first restriction is on the number of agents. In a general csa, we may have an unbounded number of agents in the system. By restricting the range of η to a finite subset $\{1, 2, \ldots, n\}$ of $\mathbf{N}$, we obtain csa's which may have upto n agents, which we call n-csa's.

As we had mentioned earlier, if $\eta(e)$ is not a singleton, the interpretation is that the event e is performed jointly by the agents mentioned by $\eta(e)$. This intuitively corresponds to "handshaking" or synchronous communication between agents. By restricting η so that $\mid \eta(e) \mid \ = 1$ for every event e in E, we effectively rule out this type of synchronous communication. Instead, in such an *asynchronous* csa, the agents communicate by sending messages to each other. The sending and receiving of a message are regarded as two distinct actions, each involving only one agent at a time.

Finally, we say that a csa is *finitary* in case ${\downarrow}e$ is a finite set for every e in E. The motivation for defining finitary csa's is the same as the motivation for defining finitary event structures — any computation of a real system can be traced back to some starting point, so the past of any event occurring during the computation must be finite.

Figure 8 is an example of an asynchronous csa consisting of two agents, a *producer* and a *consumer*, communicating via an unbounded buffer. The producer can produce zero or more items and then quit. The consumer can consume items produced by the producer as long as the items are available in the buffer. The events in the csa are labelled p, q and c to denote these three types of actions.

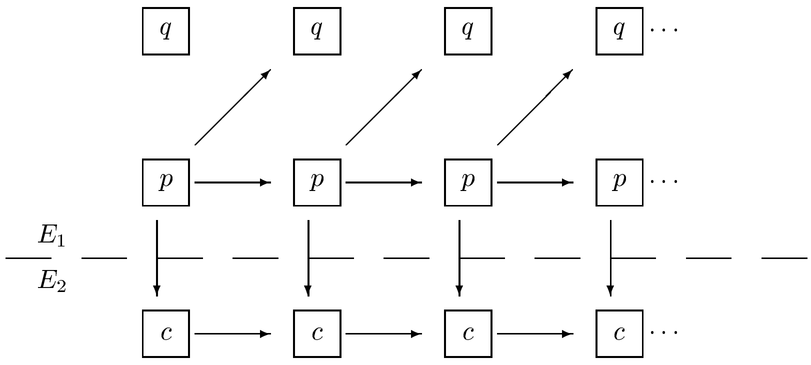

Figure 8: An asynchronous csa

B Logics for Concurrency

We now turn our attention to the problem of reasoning about the behaviour of distributed systems.

A *specification language* is simply a formalism in which one specifies behaviours of systems under study. Thus, a specification language for distributed systems is one in which we can describe behavioural properties of distributed systems.

The specification language should permit us to combine simple specifications together to construct more complex specifications, reflecting the intuition that large systems can be broken down into more manageable subsystems. This calls for disjunctive and conjunctive abilities in the language.

In addition, since we are dealing with distributed systems we expect to describe properties like causality, choice and concurrency. For this, we will need to be able to specify the relationships that hold between system states as the computation proceeds.

Our requirements suggest the use of a formal logic with boolean connectives and temporal modalities as our specification language. Temporal logic is a branch of modal logic which is used to study structures of states varying with time. We will design a variety of modal logics which are extensions of temporal logic to deal with the models of distributed systems developed in Part A.

We begin with a quick sketch of classical propositional modal logic. We assume the existence of $\mathcal{P}$, a countable set of atomic propositions $\{p_0, p_1, \ldots\}$. The well-formed formulas of our logic $\mathcal{L}_0$ are defined inductively:

- Every $p \in \mathcal{P}$ is a formula of $\mathcal{L}_0$.
- If α and β are formulas of $\mathcal{L}_0$, then so are $\neg\alpha, \alpha \vee \beta$ and $\Diamond\alpha$.

$\neg\alpha$ is to be read as "not α", $\alpha \vee \beta$ is to be read as "α or β", and $\Diamond\alpha$ is to be read as "Diamond α". The intended meaning of $\Diamond\alpha$ is "α becomes true eventually".

Formulas are to be interpreted over *frames*. In our set-up, a *frame* is a transition system $TS = (S, \Sigma, \rightarrow)$. A *model* M is a frame with a *valuation* function; i.e $M = (TS, V)$, where $TS = (S, \Sigma, \rightarrow)$ is a transition system and $V : S \rightarrow \wp(\mathcal{P})$. For example, if $V(s) = \{p_1, p_3\}$, we interpret this to mean that propositions p_1 and p_3 are true at state s and, further, that no other proposition is true at s.

The notion of a formula α being true at a state s in a model $M = (TS, V)$ where $TS = (S, \Sigma, \rightarrow)$, denoted as $M, s \models \alpha$, is defined inductively as follows:

(i) $M, s \models p$ iff $p \in V(s)$, for $p \in \mathcal{P}$.
(ii) $M, s \models \neg\alpha$ iff $M, s \not\models \alpha$.
(The notation $M, s \not\models \alpha$ stands for "It is not the case that $M, s \models \alpha$")
(iii) $M, s \models \alpha \vee \beta$ iff $M, s \models \alpha$ or $M, s \models \beta$.
(iv) $M, s \models \Diamond\alpha$ iff $\exists s' \in \mathcal{R}(s) : M, s' \models \alpha$.
(Recall that $\mathcal{R}(s)$ is the set of states reachable from s via $\rightarrow$)

$M, s \models \alpha$ can be interpreted as the assertion that the model M at state s is an implementation of the specification α. We say α is *satisfiable* if there exists a model $M = (TS, V)$, where $TS = (S, \Sigma, \rightarrow)$, and there exists a state $s \in S$ such that $M, s \models \alpha$. We say that α is *M-valid* if $M, s \models \alpha$ for every $s \in S$. We say that α is *valid* — and denote this by $\models \alpha$ — if α is M-valid for every model M. It is easy to see that α is valid iff $\neg\alpha$ is not satisfiable.

The following derived formulas are useful.

$\alpha \wedge \beta$	$\stackrel{\text{def}}{=}$	$\neg(\neg\alpha \vee \neg\beta)$	the conjunction of α and β.
$\alpha \supset \beta$	$\stackrel{\text{def}}{=}$	$\neg\alpha \vee \beta$	α implies β
$\alpha \equiv \beta$	$\stackrel{\text{def}}{=}$	$(\alpha \supset \beta) \wedge (\beta \supset \alpha)$	logical equivalence of α and β
$\Box\alpha$	$\stackrel{\text{def}}{=}$	$\neg(\Diamond\neg\alpha)$	"Henceforth" α
True	$\stackrel{\text{def}}{=}$	$p_0 \vee \neg p_0$	
False	$\stackrel{\text{def}}{=}$	$\neg True$	

It can easily be verified that for any model $M = ((S, \rightarrow), V)$ and $s \in S$,

$$M, s \models \Box\alpha \text{ iff } \forall s' \in \mathcal{R}(s) : M, s' \models \alpha.$$

A number of interesting properties of transition systems can be expressed using this logic. Suppose that we are using transition systems to model a distributed system consisting of n processes which can compete for a shared resource r. Let the atomic proposition c_i stand for "Process i has access to the resource r". Then

$$\Box \bigwedge_{i \in \{1,2,\ldots,n\}} (c_i \supset \bigwedge_{i \neq j} \neg c_j)$$

expresses a so-called *safety property*. It says that at any system state, at most one process has control of the shared resource r. This will ensure, for instance, that in case r is a shared piece of data then the sequence of values assumed by r during the history of the system will be well-defined. Broadly speaking, safety properties assert that "bad" situations never arise in the system.

Similarly, if we let the proposition rq_i stand for "Process i requires access to resource r", the formula

$$\Box \bigwedge_{i \in \{1,2,\ldots,n\}} (rq_i \supset \Diamond c_i)$$

expresses a *liveness property*. It says that any request made by a process for the shared resource is eventually granted by the system. In general, liveness properties specify that something "good" occurs eventually.

This logical framework is very simple, but for that reason is also not as expressive as we would wish. In particular, we would like to devise logics to reason about models with true concurrency. In the rest of this section, we shall show how such logics can be defined for the formal models presented in Part A.

1 Logic for Distributed Transition Systems

Recall that in a dts, a concurrent step consists of a transition labelled by a finite set of actions. This leads us to augment the simple modal logic considered earlier with one additional modality, $\langle u \rangle$, where u is a finite subset of Σ, the set of actions.

Let $\mathcal{L}_{DTS}$ be the language whose well formed formulas are given by:

- Every $p \in \mathcal{P}$ is a formula of $\mathcal{L}_{DTS}$.
- If α and β are formulas of $\mathcal{L}_{DTS}$ then so are $\neg\alpha$, $\alpha \vee \beta$, $\Diamond\alpha$ and $\langle u \rangle\alpha$, where u is a finite subset of Σ.

Thus, the logic $\mathcal{L}_{DTS}$ is parametrized by Σ. To emphasize this, we will write $\mathcal{L}_{DTS}^{\Sigma}$ instead of $\mathcal{L}_{DTS}$.

As one may expect, the frames for our logic are dts's over Σ. A model is a pair $M = (DTS, V)$, where $DTS = (S, \Sigma, \rightarrow)$ is a dts over Σ and $V : S \rightarrow \wp(\mathcal{P})$ is the valuation function. Given $s \in S$, the notion $M, s \models \alpha$ is defined as before for the atomic propositions and for the connectives $\neg$ and $\vee$ and the modality $\Diamond$. For the new modality we define:

$$M, s \models \langle u \rangle\alpha \text{ iff } \exists s' \in S : s \xrightarrow{u} s' \text{ and } M, s' \models \alpha.$$

Figure 9: Varieties of branching in transition systems

Relative to the new notion of models, satisfiability and validity are defined as before. We will write $\models^{\Sigma}_{DTS} \alpha$ to denote that α is a valid formula in this logic. Let SAT^{Σ}_{DTS} denote the set of all satisfiable formulas from $\mathcal{L}^{\Sigma}_{DTS}$.

Before considering an example, we introduce some notational conventions. The derived modality $[u]$ is defined as:

$$[u]\alpha \stackrel{\text{def}}{=} \neg\langle u\rangle\neg\alpha$$

Where u is a singleton $\{a\}$, we will write $\langle a\rangle\alpha$ instead of $\langle\{a\}\rangle\alpha$. For the empty step, we write $\langle\emptyset\rangle\alpha$.

Now that the modalities are indexed by steps, we can clearly identify the branching points in a transition system. For example, consider the transition systems shown in Figure 9. In the first system, starting at s_0 we can perform a and then choose between b and c whereas in the second system, at s'_0 we have to decide right away whether we are going to execute a followed by b or a followed by c. The first situation is captured by the formula $\langle a\rangle(\langle b\rangle\mathit{True} \wedge \langle c\rangle\mathit{True})$ while the second can be expressed as $\langle a\rangle(\langle b\rangle\mathit{True} \wedge [c]\mathit{False}) \wedge \langle a\rangle(\langle c\rangle\mathit{True} \wedge [b]\mathit{False})$.

In this logic, we can distinguish between interleavings and true concurrency. For instance, the formula $\langle a\rangle\langle b\rangle\mathit{True} \wedge \langle b\rangle\langle a\rangle\mathit{True} \wedge [\{a,b\}]\mathit{False}$ is satisfiable. At the state where this formula is true, both the interleavings ab and ba can occur, but the corresponding concurrent step $\{a,b\}$ is not enabled. On the other hand, it is easy to see that the formula $\langle\{a,b\}\rangle\alpha \supset \langle a\rangle\langle b\rangle\alpha$ is a valid formula, because the definition of a dts guarantees the existence of a function f associated with each step, breaking it up into substeps.

Returning briefly to the system of n processes considered earlier, assume that the shared resource r represents a data item in a shared block of memory. Let ud_i denote the act of process i updating the value of r. Then, the specification

$$\Box \bigwedge_{i\neq j}[\{ud_i, ud_j\}]\mathit{False}$$

requires that the memory manager never permit two distinct processes to concurrently update r.

Let us consider another example. The writing of a paper can be seen as a sequential activity: work out what you want to say, write it out, get it typed. In the case of a joint paper, the work may be divided up in terms of sections. One policy the authors may follow is to work out all the sections before preparing a typescript, with meetings for discussion and correction in between. That is, the authors satisfy

$$\langle WK\rangle(\mathit{worked} \wedge \langle WR\rangle(\mathit{written} \wedge \langle TY\rangle\mathit{typed}))$$

where $WK = \{$work out §1, work out §2, work out §3 $\}$
$WR = \{$write §1, write §2, write §3 $\}$
$TY = \{$type §1, type §2, type §3 $\}$

and *worked*, *written* and *typed* are atomic propositions indicating the end of the working out, writing and typing steps respectively. Here we have assumed that there are three authors each of whom is responsible for one section.

The concurrent steps are necessary, since they express the fact that this is a *joint* paper; if the interleaving of the actions required for the three sections were present, we could not rule out the possibility that the three authors were separately writing three (single-section) papers.

The states we are using are global states. The person working out §2 may refer to a lemma in §1; the person doing the word processing for §1 may use the macros defined in §3.

It becomes necessary to use sequentializations when a complete record of the writing of the paper is required. For example, a mistake pointed out by the referee in §2 may be traced to the lemma in §1, which may be just a case of wrong typing thanks to a misapplication of the macro from §3.

This sort of mixture of independent actions and synchronization is well described in a dts framework.

We now turn to the formal theory of the language $\mathcal{L}^{\Sigma}_{DTS}$. Typical questions one asks of such a logic include:

- Is the set of valid formulas axiomatizable?
- Is the satisfiability problem decidable?

The answers to these questions provide a good deal of insight into the strengths and weaknesses of the logic and, most importantly, into the *expressive power* of the logic.

It turns out that both these questions have positive answers for $\mathcal{L}^{\Sigma}_{DTS}$. Consider the following logical system *ND*.

The System *ND*

AXIOM SCHEMES

(A0)	All the substitutional instances of the tautologies of Propositional Calculus.	
(A1)	(a) $\Box(\alpha \supset \beta) \supset (\Box\alpha \supset \Box\beta)$	(Deductive Closure)
	(b) $[u](\alpha \supset \beta) \supset ([u]\alpha \supset [u]\beta)$	
(A2)	$\Box\alpha \supset [u]\alpha \wedge \Box\Box\alpha$	(Reachability)
(A3)	$\alpha \equiv \langle\emptyset\rangle\alpha$	(Empty Step)
(A4,k)	(for $k \geq 1$)	(Step Axiom)

$$\langle u\rangle\alpha \wedge \bigwedge_{v\subseteq u}[v]\bigvee_{i=1}^{k}\beta_v^i \supset \bigvee_{f\in F(u,k)}\ \bigwedge_{v_1\subseteq u}\langle v_1\rangle(\gamma_{v_1} \wedge \bigwedge_{v_1\subseteq v_2\subseteq u}\langle v_2 - v_1\rangle\gamma_{v_2})$$

where $F(u,k)$ is the set of all functions $\{f \mid f\colon \wp(u) \to \{1,2,\ldots,k\}\}$ and

$$\gamma_v = \begin{cases} \beta_v^{f(v)} \wedge \alpha & \textit{if } v = u \\ \beta_v^{f(v)} & \textit{if } v \subset u \end{cases}$$

INFERENCE RULES

(MP) $\dfrac{\alpha, \alpha \supset \beta}{\beta}$ (TG) $\dfrac{\alpha}{\Box\alpha}$

Axioms A0 to A2 and the rules MP and TG are standard. The characteristic axioms of dts's are A3 and A4,k. A3 captures the fact that the empty step cannot change the state of the system. A4,k is actually an infinite set of axioms, finitely presented. The complicated formulation of A4,k is necessary to describe the fact that each concurrent step u in a dts can be broken up into concurrent substeps which are specified by the associated function $f\colon \wp(u) \to S$.

A formula α is called a *thesis* of the system ND — denoted $\vdash_{ND} \alpha$ — iff α can be derived in a finite number of steps using the axioms and inference rules of ND.

Theorem 1.1

(i). ND is a sound and complete axiomatization of the valid formulas in $\mathcal{L}^{\Sigma}_{DTS}$. In other words, $\vdash_{ND} \alpha$ iff $\models^{\Sigma}_{DTS} \alpha$ for every $\alpha \in \mathcal{L}^{\Sigma}_{DTS}$.

(ii). The satisfiability problem for this logic (i.e. the membership problem for SAT^{Σ}_{DTS}) is decidable in nondeterministic exponential time.

It turns out that combining concurrency, captured by the step notion, with determinacy leads to a very expressive class of models. The frame $TS = (S, \Sigma, \to)$ is said to be *deterministic* if for every $s \in S$ and every $u \in \wp_{fin}(\Sigma)$ there exists at most one $s' \in S$ such that $s \xrightarrow{u} s'$. A model is deterministic if its underlying frame is.

The formula α is said to be deterministically satisfiable if there exists a deterministic model for α. Similarly, α is said to be deterministically valid if α is valid over the class of deterministic models. Let $\models^{\Sigma}_{Det} \alpha$ denote that α is deterministically valid and let $DSAT^{\Sigma}_{DTS}$ denote the set of deterministically satisfiable formulas in $\mathcal{L}^{\Sigma}_{DTS}$.

It turns out that the deterministically valid formulas in $\mathcal{L}^{\Sigma}_{DTS}$ are axiomatizable. Thanks to determinacy, one obtains a much simpler axiomatization than for the general case. Let D denote the logical system obtained from ND by dropping the infinitary set of axioms A4,k ($k \geq 1$) and adding two new axioms:

(A5)	$\langle u\rangle\alpha \supset \langle v\rangle\langle u-v\rangle\alpha$	$(v \subseteq u)$	(Weak Step Axiom)
(A6)	$\langle u\rangle\alpha \supset [u]\alpha$		(Determinacy)

Let $\vdash_D \alpha$ denote that α is derivable in D.

Theorem 1.2

(i). D is a sound and complete axiomatization of the deterministically valid formulas in $\mathcal{L}^{\Sigma}_{DTS}$. In other words, $\vdash_D \alpha$ iff $\models^{\Sigma}_{Det} \alpha$ for every $\alpha \in \mathcal{L}^{\Sigma}_{DTS}$.

(ii). The membership problem for $DSAT^{\Sigma}_{DTS}$ is undecidable.

The surprise here is that determinacy adds a sufficient amount of expressive power to make the satisfiability problem undecidable. By combining concurrent steps in a deterministic fashion, it turns out that we can encode the two-dimensional grid of natural numbers $\mathbf{N} \times \mathbf{N}$. We can then use this encoding to reduce some undecidable tiling problems described by Wang [Wang 1961] and Harel [Harel 1985] to the problem of deterministic satisfiability in our logic. This negative result was shown by Parikh [Parikh 1989].

A variety of positive and negative results can be obtained in this logical framework by studying the effect of placing suitable restrictions on dts's. For instance, we can restrict the set of actions Σ to be finite. Alternatively, we can demand the dts as a whole be finite — that is, the set of states and the set of transitions are both finite. We can also incorporate ideas from trace theory, arising out of the work of Mazurkiewicz [Mazurkiewicz 1989], and define trace transition systems, which permit both local and global specifications of concurrency. Finally, we can also study a smooth generalization of Propositional Dynamic Logic [Harel 1984] obtained by extending the notion of a regular program to permit concurrent steps as atomic actions. The details can be found in a forthcoming paper [Lodaya et al 1991].

The logical language $\mathcal{L}^{\Sigma}_{DTS}$ can also be interpreted over Σ-labelled elementary net systems and Σ-labelled event structures, where the labelling function is co-injective. The frames that we use are the corresponding dts's, as defined in Part A. Thus, a Σ-labelled elementary net system $\mathcal{N}_{\Sigma} = (\mathcal{N}, \phi)$, where $\mathcal{N} = (B, E, F, c_{in})$, gives rise to a model $(DTS_{\mathcal{N}_{\Sigma}}, V)$, where $V : C_{\mathcal{N}} \to \wp(\mathcal{P})$. Similarly, a Σ-labelled event structure $ES_{\Sigma} = (ES, \phi)$, where $ES = (E, \leq, \#)$, defines a model $(DTS_{ES_{\Sigma}}, V)$, where $V : \mathcal{C}^{fin}_{ES} \to \wp(\mathcal{P})$.

Let $SAT^{\Sigma}_{\mathcal{N}}$ and SAT^{Σ}_{ES} denote the set of formulas from $\mathcal{L}^{\Sigma}_{DTS}$ satisfiable in models generated by Σ-labelled elementary net systems and Σ-labelled event structures respectively.

Theorem 1.3 $SAT^{\Sigma}_{DTS} = SAT^{\Sigma}_{\mathcal{N}} = SAT^{\Sigma}_{ES}$.

In other words, this logic cannot discriminate between these classes of models.

2 Logic for Event Structures

We now turn from dts's to event structures as frames for our logic. In the logic for dts's, we used the global state approach to reasoning about the behaviour of the system. In this approach, assertions are made by a "global" observer of the system who can "see" the distributed system in its entirety in any given state. This is appropriate for dts's, since the states of a dts do in fact correspond to the global states of the system being modelled.

Alternatively, we can reason about the system from the point of view of the local states of the system. Here, assertions are made by individual *agents* in the system and hence the nature of the assertion is determined by the "visibility" of the system state from that agent's point of view. This approach is more suitable for reasoning based on event structures, where we can use a local configuration $\downarrow e$ to represent the local state of the system at the point where the event e has just occurred.

Another feature of the dts logic is that concurrency is described by explicitly specifying the actions which are to be performed concurrently and describing the effect of such actions. This approach is natural for dts's because the models themselves are action-based. On the other hand, in an event structure it is more convenient to specify concurrency in an abstract manner by simply asserting facts about concurrent events without specifying which actions are to be performed concurrently.

The key notions in the theory of event structures are those of causality, conflict and concurrency. This leads us to extend the language $\mathcal{L}_0$ by adding modalities to capture these notions. It turns out to be fruitful to split up causality into two parts, allowing us to specify both "past" and "future" behaviour.

The logic $\mathcal{L}_{ES}$ is built up as follows: again fix $\mathcal{P} = \{p_0, p_1, \ldots\}$, a countable set of atomic propositions. Then the well-formed formulas of $\mathcal{L}_{ES}$ are given by:

- Every $p \in \mathcal{P}$ is a formula of $\mathcal{L}_{ES}$.
- If α and β are formulas of $\mathcal{L}_{ES}$, then so are $\neg\alpha$, $\alpha \vee \beta$, $\Diamond\alpha$, $\diamondminus\alpha$, $\triangle\alpha$ and $\triangledown\alpha$.

Here, the modalities $\Diamond$ and $\diamondminus$ denote the future and past respectively. $\triangle$ will be used to describe concurrency and $\triangledown$ will be used to capture conflict.

Frames for this logic are event structures, or rather the local configurations of event structures. More precisely, a frame is a pair (ES, LC_{ES}), where $ES = (E, \leq, \#)$ is an event structure and LC_{ES} is the set of local configurations of ES.

A model is a pair $M = ((ES, LC_{ES}), V)$ where ES is frame and $V : LC_{ES} \to \wp(\mathcal{P})$ is a valuation function. If $p \in V(\downarrow e)$ then this is taken to mean that p is true at the local state $\downarrow e$ in the model M.

The notion of a formula α being true at a local state $\downarrow e$ in the model $M = ((ES, LC_{ES}), V)$ is denoted as $M, \downarrow e \models \alpha$ and is defined inductively as follows:

(i) $M, \downarrow e \models p$ iff $p \in V(\downarrow e)$, for $p \in \mathcal{P}$.
(ii) $M, \downarrow e \models \neg\alpha$ iff $M, \downarrow e \not\models \alpha$.
(iii) $M, \downarrow e \models \alpha \vee \beta$ iff $M, \downarrow e \models \alpha$ or $M, \downarrow e \models \beta$.
(iv) $M, \downarrow e \models \Diamond\alpha$ iff $\exists e' : e < e'$ and $M, \downarrow e' \models \alpha$.
(v) $M, \downarrow e \models \diamondminus\alpha$ iff $\exists e' : e' < e$ and $M, \downarrow e' \models \alpha$.
(vi) $M, \downarrow e \models \triangledown\alpha$ iff $\exists e' : e \# e'$ and $M, \downarrow e' \models \alpha$.
(vii) $M, \downarrow e \models \triangle\alpha$ iff $\exists e' : e \; co \; e'$ and $M, \downarrow e' \models \alpha$.

Notice that we have defined the modalities $\Diamond$ and $\diamondminus$ in an *irreflexive* manner. This is necessary for the axiomatization which follows.

The notions of satisfiability and validity are defined as usual. $\models_{ES} \alpha$ will denote that α is a valid formula in $\mathcal{L}_{ES}$.

The derived connectives $\wedge$, $\supset$, $\equiv$, $\Box$ are defined as before. In addition, we set

$$\boxminus\alpha \stackrel{\text{def}}{=} \neg\diamondminus\neg\alpha, \quad \underline{\triangledown}\alpha \stackrel{\text{def}}{=} \neg\triangledown\neg\alpha, \quad \underline{\triangle}\alpha \stackrel{\text{def}}{=} \neg\triangle\neg\alpha$$

We can also define a useful derived modality as follows:

$$\mathcal{S}\alpha \stackrel{\text{def}}{=} \alpha \vee \Diamond\alpha \vee \diamondminus\alpha \vee \triangledown\alpha \vee \triangle\alpha$$

$\mathcal{S}\alpha$ is to be read as "Somewhere α". Its dual $\mathcal{E}\alpha \stackrel{\text{def}}{=} \neg\mathcal{S}\neg\alpha$, read as "Everywhere α" expands as follows:

$$\mathcal{E}\alpha \stackrel{\text{def}}{=} \alpha \wedge \Box\alpha \wedge \boxminus\alpha \wedge \underline{\triangledown}\alpha \wedge \underline{\triangle}\alpha$$

Thus $\mathcal{E}\alpha$ describes a property invariant over the entire model.

Many interesting features of event structures can be expressed in this logic. Recall that the maximal computations of event structures are termed runs. We can use an atomic proposition ρ to mark out a run with the formula $\rho \equiv \underline{\triangledown}\neg\rho$. For any model $M = ((ES, LC_{ES}), V)$, if the formula $\rho \equiv \underline{\triangledown}\neg\rho$ is M-valid, then $\{e \mid M, \downarrow e \models \rho\}$ constitutes a run of ES. Using this method of marking out runs, we can express liveness and safety properties in event structures. Let α represent a liveness property. Then $\mathcal{S}(\rho \wedge \alpha)$ is M-valid for a model M just in case every computation of the underlying event structure contains a local state where α is true. Similarly, if β represents an undesirable situation, the formula $\mathcal{E}(\rho \supset \neg\beta)$ expresses the safety property that β does occur at any state of the run marked by ρ.

In a similar spirit the formula $\chi \equiv \Box\neg\chi \wedge \boxminus\neg\chi$ can be used to capture the notion of a *cut* — a maximal set of pair-wise incomparable events. Within a computation, a cut corresponds to a global state. Thus we can use the notion of a cut in conjunction with that of a run to look "sideways" from a local state and make assertions about the current global state.

The formula $\triangledown\alpha \supset \Box\triangledown\alpha$ describes the fact that conflict is inherited in a prime event structure. The formula $\triangle\alpha \supset \boxminus(\triangle\alpha \vee \Diamond\alpha)$ expresses the fact that the configurations of an event structure are "consistent" by asserting that the unified past of any pair of events in *co* is conflict-free.

Due to lack of space, we will not provide a separate detailed example for this logic. The logic presented in the next section, called $\mathcal{L}_{CSA}$, is also based on event structures. We shall provide a detailed example for that logic. It will not be difficult to see how that example can be translated into the present framework.

Consider the logical system E.

The System E

AXIOM SCHEMES

(A0) All the substitutional instances of the tautologies of Propositional Calculus.

(A1) (i) $\Box(\alpha \supset \beta) \supset (\Box\alpha \supset \Box\beta)$ (Deductive Closure)
(ii) $\boxminus(\alpha \supset \beta) \supset (\boxminus\alpha \supset \boxminus\beta)$
(iii) $\dot{\triangledown}(\alpha \supset \beta) \supset (\dot{\triangledown}\alpha \supset \dot{\triangledown}\beta)$
(iv) $\dot{\triangle}(\alpha \supset \beta) \supset (\dot{\triangle}\alpha \supset \dot{\triangle}\beta)$

(A2) (i) $\Box\alpha \supset \Box\Box\alpha$ (Transitivity of $<$)
(ii) $\boxminus\alpha \supset \boxminus\boxminus\alpha$

(A3) (i) $\alpha \supset \dot{\triangledown}\dot{\triangledown}\alpha$ (Symmetry of $\#$ and *co*)
(ii) $\alpha \supset \dot{\triangle}\dot{\triangle}\alpha$

(A4) (i) $\alpha \supset \Box\diamondminus\alpha$ (Relating past and future)
(ii) $\alpha \supset \boxminus\Diamond\alpha$

(A5) $\triangledown\alpha \supset \Box\triangledown\alpha$ (Conflict inheritance)

(A6) $\triangle\alpha \supset \boxminus(\Diamond\alpha \vee \triangle\alpha)$ (Conflict-free past)

(A7) (i) $\Diamond\alpha \supset \Box(\alpha \vee \Diamond\alpha \vee \diamondminus\alpha \vee \triangledown\alpha \vee \triangle\alpha)$ (Relating $<$,$\#$ and *co*)
(ii) $\triangledown\alpha \supset \dot{\triangledown}(\alpha \vee \Diamond\alpha \vee \diamondminus\alpha \vee \triangledown\alpha \vee \triangle\alpha)$
(iii) $\triangle\alpha \supset \dot{\triangle}(\alpha \vee \Diamond\alpha \vee \diamondminus\alpha \vee \triangledown\alpha \vee \triangle\alpha)$
(iv) $\diamondminus\alpha \supset \boxminus(\alpha \vee \Diamond\alpha \vee \diamondminus\alpha \vee \triangle\alpha)$
(v) $\triangledown\alpha \supset \dot{\triangle}(\Diamond\alpha \vee \triangledown\alpha \vee \triangle\alpha)$
(vi) $\triangle\alpha \supset \Box(\diamondminus\alpha \vee \triangledown\alpha \vee \triangle\alpha)$

INFERENCE RULES

(MP) $\dfrac{\alpha}{\alpha \supset \beta}$

(TG) (i) $\dfrac{\alpha}{\Box\alpha}$ (ii) $\dfrac{\alpha}{\boxminus\alpha}$ (iii) $\dfrac{\alpha}{\dot{\triangle}\alpha}$ (iv) $\dfrac{\alpha}{\dot{\triangledown}\alpha}$

(UNIQ) $\dfrac{\hat{p} \supset \alpha}{\alpha}$ where p is an atomic proposition not appearing in α
and $\hat{p} \stackrel{\text{def}}{=} p \wedge \Box{\sim}p \wedge \boxminus{\sim}p \wedge \dot{\triangle}{\sim}p \wedge \dot{\triangledown}{\sim}p$

Axioms A0 to A4 and inference rules MP and TG are standard. A5 expresses the fact that conflict is inherited via $\leq$. A6 ensures that any two events related by *co* have consistent (i.e. conflict-free) pasts. The remaining axioms are necessary to capture the fact that the relations $\leq, \geq, \#$ and *co* "cover" the event structure – i.e., any two distinct events are related by one of these relations.

The rule UNIQ is adapted from [Burgess 1980]. Given a proposition p, the definition of $\hat{p}$ ensures that it can be true in at most one local configuration. Hence, we can label each local configuration $\downarrow e$ by a distinct formula $\hat{p}_e$. The rule UNIQ allows us to construct this labelling, which is crucial in demonstrating the completeness of the axiomatization.

Let $\vdash_E \alpha$ denote that α is a thesis of the system E.

Theorem 2.1 *E is a sound and complete axiomatization of the valid formulas in $\mathcal{L}_{ES}$. In other words, $\vdash_E \alpha$ iff $\models_{ES} \alpha$.*

Recall that we had defined an auxiliary relation $\#_\mu$ in an event structure, called the minimal conflict relation. We can define a modality $\triangledown_\mu$ to capture the relation $\#_\mu$.

It is possible to strengthen $\mathcal{L}_{ES}$ by replacing the modality $\triangledown$ by the modality $\triangledown_\mu$. Let us call this new language $\mathcal{L}^\mu_{ES}$. To obtain a useful comparison with $\mathcal{L}_{ES}$, and also to obtain an axiomatization, we must change the notion of a frame. For this language, we define a frame to be a pair (ES, LC_{ES}) where ES is a *well branching* event structure. Recall that a well branching event structure is one in which the $\#$ relation can be completely specified using the relations $\#_\mu$ and $\leq$. As usual, a model is a frame together with a valuation function. Models based on well branching frames are called well branching models.

The semantics of $\mathcal{L}^\mu_{ES}$ is the same as that of $\mathcal{L}_{ES}$ except that the clause for $\triangledown$ is replaced by:

$$M, {\downarrow}e \models \triangledown_\mu \alpha \text{ iff } \exists e' : e \;\#_\mu\; e' \text{ and } M, {\downarrow}e' \models \alpha.$$

In $\mathcal{L}^\mu_{ES}$, we can obtain $\triangledown$ as a derived modality:

$$\triangledown \alpha \stackrel{\text{def}}{=} \triangledown_\mu \alpha \vee \triangledown_\mu \Diamond \alpha \vee \diamondminus \triangledown_\mu \alpha \vee \diamondminus \triangledown_\mu \Diamond \alpha$$

As before, $\dot{\triangledown} \alpha$ denotes the formula $\neg \triangledown \neg \alpha$. It is easy to verify that $\dot{\triangledown} \alpha$ can be expressed as follows:

$$\dot{\triangledown} \alpha \stackrel{\text{def}}{=} \dot{\triangledown}_\mu \alpha \wedge \dot{\triangledown}_\mu \Box \alpha \wedge \boxminus \dot{\triangledown}_\mu \alpha \wedge \boxminus \dot{\triangledown}_\mu \Box \alpha$$

In a well branching model, the derived modalities $\triangledown$ and $\dot{\triangledown}$ have precisely the same interpretation as the corresponding modalities of $\mathcal{L}_{ES}$. On the other hand, there is no obvious way to characterize the minimal conflict relation $\#_\mu$ using the modality $\triangledown$. In this connection, we can establish the following result.

Theorem 2.2 *For well branching models, the language $\mathcal{L}^\mu_{ES}$ is strictly more expressive than $\mathcal{L}_{ES}$.*

Informally, this result says that we can use formulas from $\mathcal{L}^\mu_{ES}$ to differentiate models which are indistinguishable using the language $\mathcal{L}_{ES}$.

An example of the use of $\triangledown_\mu$ is in systems where agents have names, like csa's. For each event e that process i participates in, we can assign an atomic proposition τ_i to the local configuration ${\downarrow}e$. Suppose that there are n agents in the system, with "names" $\tau_1, \tau_2, \ldots, \tau_n$. Then the formula $\bigwedge_{1 \leq i \leq n} (\tau_i \supset \dot{\triangledown}_\mu \tau_i)$ expresses the fact that all choices in behaviour are made locally by individual agents.

The axiom system E_μ is obtained by adding the following axiom schemes to the system E.

(A1)	(v) $\dot{\triangledown}_\mu(\alpha \supset \beta) \supset (\dot{\triangledown}_\mu \alpha \supset \dot{\triangledown}_\mu \beta)$	(Deductive Closure)
(A3)	(iii)$\alpha \supset \dot{\triangledown}_\mu \triangledown_\mu \alpha$	(Symmetry of $\#_\mu$)
(A6)	(ii) $\triangledown_\mu \alpha \supset \boxminus(\Diamond \alpha \vee \triangle \alpha)$	(Minimal Conflict)

A1(v) and A3(iii) are standard. A6(ii) is the characteristic axiom describing the $\#_\mu$ relation as the *minimal* conflict relation.

Let $\vdash^\mu_E \alpha$ denote that α is a thesis of the system E_μ and let $\models^\mu_{ES} \alpha$ denote that α is valid over the class of well branching models. Then we get:

Theorem 2.3 *E_μ is a sound and complete axiomatization of the valid formulas in $\mathcal{L}^\mu_{ES}$. In other words, $\vdash^\mu_E \alpha$ iff $\models^\mu_{ES} \alpha$.*

3 Logic for Communicating Sequential Agents

We now wish to study a means of talking about a central feature of many distributed systems — the communication pattern between the components of the system that ensure coördination. For this, we shall define a logic that is to be interpreted over csa's.

Let $\mathcal{P} = \{p_0, p_1, \ldots\}$ be a countable set of atomic propositions, and $\mathcal{T} = \{\tau_0, \tau_1, \ldots\}$, a countable set of *type* propositions disjoint from $\mathcal{P}$. The formulas of $\mathcal{L}_{CSA}$ are built up as follows:

- Every member of $\mathcal{P} \cup \mathcal{T}$ is a formula of $\mathcal{L}_{CSA}$.

- If α and β are formulas of $\mathcal{L}_{CSA}$, then so are $\neg\alpha$, $\alpha \vee \beta$, $\Diamond_i\alpha$ and $\diamondminus_i\alpha$.

The formula τ_i asserts that the observer is located in agent i. $\Diamond_i$ and $\diamondminus_i$ capture the "visible" future and past of agent i. This will become clearer when we define the formal semantics of these modalities.

A frame for $\mathcal{L}_{CSA}$ is a pair (CSA, LC_{CSA}), where $CSA = (E, \leq, \eta)$ is a system of communicating sequential agents and LC_{CSA} is the set of local states of CSA. A model is a pair $M = ((CSA, LC_{CSA}), V)$ where (CSA, LC_{CSA}) is a frame and $V\colon LC_{CSA} \to \wp(\mathcal{P} \cup \mathcal{T})$ is a valuation function such that

$$\tau_i \in V(\downarrow e) \text{ iff } i \in \eta(e).$$

The notion $M, \downarrow e \models \alpha$ can be defined inductively as follows:

(i) $M, \downarrow e \models \alpha$ iff $\alpha \in V(\downarrow e)$, for $\alpha \in \mathcal{P} \cup \mathcal{T}$.
(ii) $M, \downarrow e \models \neg\alpha$ iff $M, \downarrow e \not\models \alpha$.
(iii) $M, \downarrow e \models \alpha \vee \beta$ iff $M, \downarrow e \models \alpha$ or $M, \downarrow e \models \beta$.
(iv) $M, \downarrow e \models \diamondminus_i\alpha$ iff $\exists e' \in E_i : e' \leq e$ and $M, \downarrow e' \models \alpha$.
(v) $M, \downarrow e \models \Diamond_i\alpha$ iff $\begin{cases} (e \in E_i): & \exists e' \in E_i : e \leq e' \text{ and } M, \downarrow e' \models \alpha. \\ (e \notin E_i): & \forall e' \in E_i : \text{ if } e' \leq e \text{ then } M, \downarrow e' \models \Diamond_i\alpha. \end{cases}$

Note that $\diamondminus_i$ behaves like a normal past modality — it covers all events that lie in the i-past of e. However $\Diamond_i\alpha$ is different: in agent j, $j \neq i$, it asserts that upto the last communication from i, there is a future for agent i satisfying α. In case there is no communication from agent i at all, agent j can assert $\Diamond_i\alpha$ for *any* formula α.

Define $\boxminus_i\alpha \stackrel{\text{def}}{=} \neg\diamondminus_i\neg\alpha$ and $\Box_i\alpha \stackrel{\text{def}}{=} \neg\Diamond_i\neg\alpha$. It can be verified that $\Box_i\alpha \supset \diamondminus_i\Box_i\alpha$ is a valid formula over csa's. It asserts that an invariant formula about an agent must be supported by a communication from that agent. Thus $\Box_i$ is a "strong" modality whereas $\Diamond_i$ is "weak" unlike in standard modal logic. This asymmetry arises from the fact that in distributed systems, the past of other agents can be completely obtained by messages, while the possibilities for the future are only locally known.

Notice that the formula $\tau_i \wedge \tau_j$ is satisfied at a local state $\downarrow e$ only if $\{i, j\} \subseteq \eta(e)$ and thus specifies a synchronization between agents i and j. The infinite set of formulas $\{\tau_i \supset \neg\tau_j \mid i \neq j\}$ together specify that each event is in at most one agent and hence can specify *asynchronous* csa's.

Consider the formula $\diamondminus_i\alpha \wedge \diamondminus_i\beta \supset \diamondminus_i(\alpha \wedge \diamondminus_i\beta) \vee \diamondminus_i(\beta \wedge \diamondminus_i\alpha)$. This specifies that agent i is *backwards linear* – during a computation if we look back at any two events involving agent i, then they must be ordered. This captures the fact that agents in a csa are sequential.

Similarly, the formula $\diamondminus_i\alpha \supset \diamondminus_i(\alpha \wedge \boxminus_i(\neg\alpha \supset \boxminus_i\neg\alpha))$ can be used to specify *finitary* csa's, i.e those where each event has a finite past. This formula asserts that if α is true somewhere in the past, then we can find an "earliest" point where α is true.

The principal advantage of this logic is that communication between agents in a distributed system can be easily expressed: $\neg\tau_i \wedge \diamondminus_i\alpha \wedge \tau_j$ can be used to specify that i has communicated the truth of α to j sometime in the past.

We shall present a detailed example of reasoning with this logic at the end of this section. First, we present our main technical results for this logic.

We begin with logical system C defined below.

The System C

AXIOM SCHEMES

(A0) All the substitutional instances of the tautologies of Propositional Calculus.

(A1) (a) $\boxminus_i(\alpha \supset \beta) \supset (\boxminus_i\alpha \supset \boxminus_i\beta)$ (Deductive closure)
(b) $\Box_i(\alpha \supset \beta) \supset (\Box_i\alpha \supset \Box_i\beta)$

(A2) (a) $\tau_i \supset (\boxminus_i\alpha \supset \alpha)$ (Local reflexivity)
(b) $\tau_i \supset (\Box_i\alpha \supset \alpha)$

(A3) $\diamondminus_i\diamondminus_j\alpha \supset \diamondminus_j\alpha$ (Transitivity)

(A4) (a) $\diamondminus_i\alpha \supset \Box_i\diamondminus_i\alpha$ (Relating past and future)
(b) $\Diamond_i\alpha \supset \boxminus_i\Diamond_i\alpha$

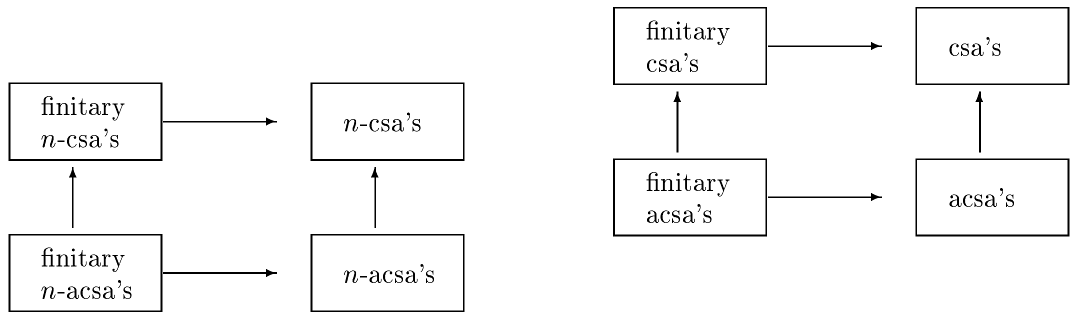

Figure 10: Subclasses of csa's

(A5) $\quad \diamondminus_i \alpha \wedge \diamondminus_i \beta \supset \diamondminus_i(\alpha \wedge \diamondminus_i \beta) \vee \diamondminus_i(\beta \wedge \diamondminus_i \alpha).$ (Backward linearity)

(A6) $\quad \Box_i \alpha \supset \diamondminus_i \Box_i \alpha$ (Communication)

(A7) (a) $\boxminus_i \tau_i$ (Type axioms)
(b) $\tau_i \supset \Box_i \tau_i$

INFERENCE RULES

(MP) $\dfrac{\alpha, \alpha \supset \beta}{\beta}$ $\qquad$ (TG$\boxminus_i$) $\dfrac{\alpha}{\boxminus_i \alpha}$ $\qquad$ (TG$\Box_i$) $\dfrac{\alpha}{\tau_i \supset \Box_i \alpha}$

Axioms A0 to A4 are standard axioms suitably modified to reflect the special interpretation of $\Diamond_i$. A5 asserts that individual agents are sequential. A6 captures that fact that knowledge about another agent's future can only be obtained via communication. A7 ensures that the type propositions from $\mathcal{T}$ are assigned consistently. The rules MP and TG$\boxminus_i$ are standard. The standard form of the rule TG$\Box_i$ will not preserve validity because of the communication requirement imposed by the semantics of $\Box_i$.

Let $\vdash_C \alpha$ denote that α is a thesis of the system C. Let $\models_{CSA} \alpha$ denote that α is valid over the class of models based on csa's. We then have the following result.

Theorem 3.1 *C is a sound and complete axiomatization of the valid formulas of $\mathcal{L}_{CSA}$. In other words $\vdash_C \alpha$ iff $\models_{CSA} \alpha$ for every $\alpha \in \mathcal{L}_{CSA}$.*

When we introduced csa's in Part A, we had defined various subclasses of csa's. Let $CSA = (E, \leq, \eta)$ be a csa. Recall that CSA is an n-csa if $\eta(E) \subseteq \{1, 2, \ldots, n\}$ — that is, there are at most n agents in the system. CSA is an *asynchronous*-csa (acsa) if $\forall e \in E : \ \mid \eta(e) \mid \ = 1$. CSA is *finitary* if $\forall e \in E : \downarrow e$ is a finite set. We can combine these notions; for example, an n-acsa is an acsa with a bounded number of agents. Similarly, we can have finitary n-csa's, finitary acsa's and, finally, finitary n-acsa's. Figure 10 pictorially represents the relationships between these various classes. The arrows in the figure indicate inclusion.

Let $\mathcal{C}$ denote one of the subclasses of csa's mentioned above. Then we can define the notions of satisfiability and validity relative to $\mathcal{C}$. Thus, a formula α is $\mathcal{C}$-satisfiable if we can find a model $M = ((CSA, LC_{CSA}), V)$ for α such that $CSA \in \mathcal{C}$. We let $SAT_{\mathcal{C}}$ denote the set of $\mathcal{C}$-satisfiable formulas in $\mathcal{L}_{CSA}$. α is $\mathcal{C}$-valid if it is valid over the class of models based on frames in $\mathcal{C}$.

We can axiomatize the $\mathcal{C}$-valid formulas for all these subclasses. The required axiomatizations are obtained by suitably combining the system C with the following axiom schemes.

AUXILIARY AXIOM SCHEMES AND INFERENCE RULES

(A8) $\quad \tau_1 \vee \tau_2 \vee \ldots \vee \tau_n$ (n agents)

(A9) $\quad \tau_i \supset \neg \tau_j$, for $i \neq j$ (disjoint agents)

(A10) (a) $\diamondminus_i \alpha \supset \diamondminus_i(\alpha \wedge \boxminus_i(\neg \alpha \supset \boxminus_i \neg \alpha))$ (well-founded agents and communications)
(b) $\diamondminus_i \alpha \supset \diamondminus_i(\alpha \wedge \boxminus_j \boxminus_i \neg \alpha)$, for $i \neq j$

Theorem 3.2

(i). The logical system $C_A \stackrel{\rm def}{=} C + (A9)$ *is sound and complete for the class of models based on acsa's.*

(ii). The logical system $C_F \stackrel{\rm def}{=} C + (A10)$ *is sound and complete for the class of models based on finitary csa's.*

(iii). The logical system $C_{FA} \stackrel{\rm def}{=} C_A + (A10)$ *is sound and complete for the class of models based on finitary acsa's.*

(iv). The logical system $C_n \stackrel{\rm def}{=} C + (A8)$, $n \in \mathbf{N}$, *is sound and complete for the class of models based on* n*-csa's.*

(v). The logical system $C_{nA} \stackrel{\rm def}{=} C_A + (A8)$, $n \in \mathbf{N}$, *is sound and complete for the class of models based on* n*-acsa's.*

(vi). The logical system $C_{nF} \stackrel{\rm def}{=} C_F + (A8)$, $n \in \mathbf{N}$, *is sound and complete for the class of models based on finitary* n*-csa's.*

(vii). The logical system $C_{nFA} \stackrel{\rm def}{=} C_{FA} + (A8)$, $n \in \mathbf{N}$, *is sound and complete for the class of models based on finitary* n*-acsa's.*

We also have the following relationship between satisfiability in subclasses with an unbounded number of agents and the corresponding subclasses with only a bounded number of agents.

Theorem 3.3 *Let* $\mathcal{C}$ *range over csa's, acsa's, finitary csa's and finitary acsa's. Let* $n\mathcal{C}$, $n \in \mathbf{N}$, *denote the corresponding class with a bounded number of agents* n*. Then* $SAT_{\mathcal{C}} = \bigcup_n SAT_{n\mathcal{C}}$.

We now give a detailed example of how communication between agents can be specified in $\mathcal{L}_{CSA}$. Consider a distributed database accessed by n processes which communicate with each other by exchanging messages. A protocol is needed whereby the processes can *commit* to a distributed transaction. When each committed process knows that all the others have also committed it can go ahead and perform its local share of the distributed transaction. For this, the following requirement must be met.

> If any process commits to the transaction then it eventually knows that all processes in the system have also committed.

Such *distributed transaction commit protocols* commonly arise in the design of distributed systems [Pinter et al 1984].

We now specify the protocol requirement in our logical language. Let $\{c_1, \ldots, c_n\}$ be a set of atomic propositions, where c_j is read to mean "process j has committed to the transaction". The formula

$$\bigwedge_i (\tau_i \wedge c_i \supset \Diamond_i (\bigwedge_j \diamondminus_i c_j)) \tag{1}$$

expresses the requirement above.

A two-stage implementation of this protocol may use two local boolean variables in each process P_i:

- a variable l_i in which process P_i records whether it can participate in the transaction or not, and
- a variable, which we also call c_i, to record the commitment of the process to the transaction.

The implementation can perhaps run as follows:

Process P_i:

(i). As soon as a local decision l_i is made, broadcast l_i to all other processes;

(ii). When l_j is heard from all j, set c_i to $True$;

(iii). As soon as c_i is set, broadcast it to all other processes;

(iv). When c_j is heard from all j, perform transaction;

(v). Acknowledge all incoming messages.

All processes follow the same protocol in a symmetric manner. This is, of course, a naïve protocol. However, our aim here is to merely illustrate the use of our logical language. Let us again, by abuse of notation, use $\{l_1, \ldots, l_n\}$ to denote another set of atomic propositions. Consider now the following formulas:

$$\bigwedge_i (\tau_i \supset (c_i \equiv \bigwedge_j \diamondminus_j l_j)) \tag{2}$$

"c_i is set $True$ only when l_j is heard from all other processes P_j"

$$\bigwedge_i (\tau_i \wedge c_i \supset \Diamond_i \bigwedge_j \diamondminus_j \diamondminus_i c_i) \tag{3}$$

"if c_i is set, then it will be broadcast and acknowledged"

Note that here an agent has to assert something about the state of other agents and this can be done only using messages from them. The formula $\Diamond_i \diamondminus_j \diamondminus_i c_i$ says that agent i has received an acknowledgment from agent j of the message c_i sent from i to j. This is necessary because we assume that messages may be lost in this network.

It is easy to verify that the formulas 2 and 3 together imply the requirement 1 above. In fact, we can use the axiom system C and logically deduce the requirement from 2 and 3. This verifies that the simple protocol above meets its specification.

Note that the protocol above works for only one transaction, in the sense that the commitment is *stable*; once a process commits to the transaction, it stays committed. When a protocol is needed for several transactions, we can index the transactions by sequence numbers and modify the specification above appropriately.

While the preceding example illustrates the specification of a protocol which assumes complete connectivity in the network of communicating agents, we can also specify protocols which demand specific patterns of connectivity. Since agents are syntactically mentioned in formulas, this logic is particularly suited for describing communications which name specific agents. We illustrate this point with another detailed example.

Assume that processes $P_0, P_1, \ldots, P_{n-1}$ are connected in a ring and communicate with each other only by exchanging messages. A process P_i can communicate only with its neighbours P_{i-1} and P_{i+1} on the ring. Here and in the sequel, addition and subtraction are assumed to be *modulo* n.

Assume that each process P_i maintains a variable x_i taking values in $\mathbf{N}$ and whose value initially is v_i, for $0 \leq i \leq n-1$. It is desired to specify a distributed protocol which computes the greatest common divisor of the values $v_0, \ldots, v_{n-1}$. Let *result* denote the value of the constant $gcd(v_0, v_1, \ldots, v_{n-1})$. When the computation terminates, the variables $x_i, i \in \{0, \ldots, n-1\}$ should satisfy

$$x_0 = x_1 = \ldots = x_{n-1} = result$$

Since our logical language is propositional in nature we cannot express values of variables and hence assume countably many propositions $X_i^k, k \in \mathbf{N}$, to denote "$x_i = k$". With this understanding we write such propositions as equalities. Similarly we assume propositions to denote "$k < l$", "$k = i - j$" etc. The protocol requirement is then specified by

$$\bigwedge_i (\tau_i \wedge (x_i = v_i) \supset \Diamond_i \bigwedge_k \diamondminus_k (x_k = result))$$

An algorithm for computing the gcd can be descibed as follows: process P_i, at any state, compares the current value of x_i, with the current values of its neighbours, x_{i-1} and x_{i+1}. In case x_i is smaller, nothing needs to be done; if x_{i-1} is smaller, x_i is updated to be $x_i - x_{i-1}$; similarly, if x_{i+1} is smaller, x_i is updated to be $x_i - x_{i+1}$. Whenever the value of x_i changes, this is communicated to the neighbouring processes. Eventually, all values stabilize at the greatest common divisor.

As before, we assume that messages may fail and hence received messages are always acknowledged. Let $\Diamond_{i \to j} \alpha$ abbreviate the formula $\tau_i \wedge \Diamond_i \diamondminus_j \diamondminus_i \alpha$. (In some sense, this stands for "i sends the message α to j and receives an acknowledgment")

Our protocol can now be specified as

$$\bigwedge_i (\tau_i \supset \Box_i \delta \wedge \boxminus_i \delta)$$

where $\delta \stackrel{\text{def}}{=} \delta_0 \wedge \delta_1 \wedge \delta_2 \wedge \delta_3$ is given by:

$$\delta_0 : \ (x_i = v \supset \bigwedge_{j \,\in\, \{i-1,\ i+1\}} \Diamond_{i \to j}(x_i = v))$$

"neighbours are always kept informed of current x_i value"

$$\delta_1 : \ (x_i = v \supset \Box_i(x_i = v' \supset v' \leq v))$$

"values are never increased"

$$\delta_2 : \ (x_i = v \wedge \diamondminus_{i-1}(x_{i-1} = v') \wedge v' < v \supset \Diamond_i(x_i = v'' \wedge v'' = v - v'))$$

"if $x_{i-1} < x_i$ then $x_i := x_i - x_{i-1}$"

$$\delta_3 : \ (x_i = v \wedge \diamondminus_{i+1}(x_{i+1} = v') \wedge v' < v \supset \Diamond_i(x_i = v'' \wedge v'' = v - v'))$$

"if $x_{i+1} < x_i$ then $x_i := x_i - x_{i+1}$"

It is easy to see that this specifies a distributed implementation of Euclid's algorithm for computing the gcd.

Discussion

In this paper, we have looked at models for distributed systems which emphasize their non-sequential behaviour and considered their logical characterization using an assortment of modal logics.

A fair amount of theory has been developed for the models we have considered. Our notion of a distributed transition system is only one of several that have been considered; alternative formulations include those of Degano and Montanari [Degano et al 1987] and Boudol and Castellani [Boudol et al 1988]. Stark has defined a related class of model called concurrent transition systems [Stark 1989]. In net theory, more general net systems include Petri nets, Predicate/transition nets and coloured nets [Brauer et al 1987]. As far as event structures are concerned, we have only considered prime event structures in this paper; other classes of event structures include stable event structures and general event structures [Winskel 1987] as well as flow event structures [Boudol 1990]. Systems of communicating sequential agents were introduced in [Lodaya et al 1989b], as a generalization of the n-agent event structures described in [Lodaya et al 1987].

The models that we have dealt with in this paper are closely related to each other. We have described how labelled net systems and labelled event structures give rise to dts's in a natural way. A strong relationship also exists between elementary net systems and prime event structures ([Nielsen et al 1980], [Nielsen et al 1990]). The connection between csa's and event structures is described in [Lodaya et al 1989b]. By establishing formal connections between models in this manner, we can translate results obtained using one class of models to other classes.

As for the logics that we have described here, the main results that we have are sound and complete axiomatizations for different classes of models (see [Lodaya et al 1991], [Lodaya et al 1989a], [Lodaya et al 1989b], [Lodaya et al 1987], [Mukund 1990], [Mukund et al 1989] and [Mukund et al 1991]). For the logic for distributed transition systems, we also have various decidability and undecidability results [Lodaya et al 1991]. However, for the logics for event structures and csa's, the decidability question remains open.

Several attempts have been made to use logics to characterize the behaviour of distributed programs. Temporal modalities have been traditionally interpreted over different types of tense structures ([Burgess 1984], [Burgess 1980]). Using the interleaving approach to modelling concurrency, various authors have used temporal logics defined on sequences and trees to describe concurrent computations (see e.g. [Clarke et al 1986], [Gabbay et al 1980], [Pnueli 1977]). Pinter and Wolper have extended this work to true concurrency by explicitly using partial orders to represent concurrent computations [Pinter et al 1984]. Katz and Peled have defined a first-order temporal logic over sets of partial orders [Katz et al 1989].

However, the use of classes of behavioural structures for distributed systems as frames for logics seems to be relatively new. Penczek has used event structures as frames [Penczek 1988]. He was the first to use an explicit modality to represent conflict. Reisig [Reisig 1986] is working on logics which directly use elementary net systems as frames. Christiansen [Christiansen 1989] has worked with csa-like frames; he uses an indexed $\triangle$ modality in his logic to describe concurrency across agents.

Trace theory is a language theoretic approach to describing concurrency which we have not considered. This formalism also gives rise to models of distributed systems with true concurrency. Here, along with an alphabet of actions, one is given an *independence relation* declaring which actions in the system are concurrent. Instead of viewing a computation as a string of symbols from the alphabet, one now considers sequences made up of sets of concurrent actions (sequences of concurrent steps, in our framework), which are called *traces*. Like strings, traces form a monoid, called a partially commutative monoid, and so one can meaningfully talk about trace languages. A syntactic Kleene-like characterization of regular trace languages has been given by Ochmanski [Ochmanski 1985], while a characterization in terms of automata has been obtained by Zielonka [Zielonka 1987]. The *pomsets* of Gischer and Pratt [Pratt 1986] are similar to traces.

Logics for trace theory have not been considered in the literature. We believe that results like the ones in Part B, Section 1 can be obtained [Lodaya et al 1991].

Another widely prevalent approach to modelling concurrency is *algebraic*. One way of describing sequential nondeterministic programs is through regular expressions, by interpreting the operators $\bullet$, $+$ and $*$ as sequential composition, choice and iteration. Similarly, in the algebraic approach to concurrency, one introduces an operator to denote the parallel composition of programs. Program behaviour is specified by modelling the language operators in an appropriate semantic domain. Popular languages for concurrency include CSP [Hoare 1984], CCS [Milner 1989] and ACP [Bergstra et al 1984], and the models most often used are transition systems [Plotkin 1981] and equational algebras [Bergstra et al 1984]. Most of this work has been based on interleaving models and only recently have attempts been made to give a "truly concurrent" semantics to these languages ([Degano et al 1989], [van Glabbeek et al 1987], [Olderog 1987]). An earlier denotational semantics using event structures as domains was given in [Winskel 1982].

In this framework, Hennessy and Milner [Hennessy et al 1985] have used action-indexed logics to characterize computations of sequential nondeterministic systems. Assuming an interleaving model of concurrency, this characterization extends to the computations of distributed systems. This work has been considerably extended by Stirling [Stirling 1987]. However, the emphasis here is on axiomatizing *program equivalences* using equational logic. Our use of action-indexed logics for models exhibiting true concurrency is inspired by this work, but we have concentrated on axiomatizing the valid formulas, as is traditional in logic.

Logics in which the modalities are indexed by programs, rather than just actions, arose in the framework of program verification [Hoare 1969]. Programs with parallel composition operators have been considered by several authors (e.g [Apt et al 1980]). Dynamic logics, originally defined over sequential programs [Harel 1984], have been extended with an operator for intersection to model synchronization [Peleg 1987]. However, a lot of work remains to be done on characterizing models for true concurrency using program-indexed logics.

References

[Apt et al 1980] Apt K R, Francez N, de Roever W P 1980 A proof system for communicating sequential processes, *ACM Trans. Prog. Lang. and Sys.* **2**,3:359-385. [Also Moitra A 1983 Letter, *same journal* **5**,3:500-501.]

[Bergstra et al 1984] Bergstra J A, Klop J W 1984 Process algebra for synchronous communication, *Inf. Control* **60**,1-3:109-137.

[Boudol 1990] Boudol G 1990 Flow event structures and flow nets, *LNCS* **469**:62-95.

[Boudol et al 1988] Boudol G, Castellani I 1988 A non-interleaving semantics for CCS based on proved transitions, *Fund. Inf.* **XI**:433-452.

[Brauer et al 1987] Brauer W, Reisig W, Rozenberg G (eds.) 1987 *Petri nets: central models and their properties, LNCS* **254**.

[Burgess 1984] Burgess J P 1984 Basic tense logic, in Gabbay D, Guenthner F (eds.): *Handbook of philosophical logic II* (Dordrecht: D Reidel) pp. 89-133.

[Burgess 1980] Burgess J P 1980 Decidability for branching time, *Studia Logica* **XXXIX**,2/3:203-218.

[Christiansen 1989] Christiansen S 1989 A logical characterization of linear n-agent event structures, M.Sc. thesis, Computer Science Dept, Århus Univ, Århus, Denmark.

[Clarke et al 1986] Clarke E M, Emerson E A, Sistla A P 1986 Automatic verification of finite-state concurrent programs using temporal logic specifications, *ACM Trans. Prog. Lang. and Sys.* **8**,2:244-263.

[Degano et al 1987] Degano P, Montanari U 1987 Concurrent histories: A basis for observing distributed systems, *J. Comput. Sys. Sci.* **34**:422-461.

[Degano et al 1989] Degano P, de Nicola R, Montanari U 1989 Partial orderings descriptions and observations of nondeterministic concurrent processes, *LNCS* **354**:438-466.

[Gabbay et al 1980] Gabbay D, Pnueli A, Shelah S, Stavi J 1980 On the temporal analysis of fairness, *Proc. 7th ACM Conf. Princ. of Prog. Lang.* pp. 163-173.

[van Glabbeek et al 1987] van Glabbeek R, Vaandrager F 1987 Petri net models for algebraic theories of concurrency, *LNCS* **259**:224-242.

[Harel 1984] Harel D 1984 Dynamic logic, in Gabbay D, Guenthner F (eds.): *Handbook of philosophical logic II* (Dordrecht: D Reidel, 1984) pp. 497-604.

[Harel 1985] Harel D 1985 Recurring dominoes: making the highly undecidable highly understandable, *Ann. Disc. Math.* **24**:51-72.

[Hennessy et al 1985] Hennessy M, Milner R 1985 Algebraic laws for nondeterminism and concurrency, *J. ACM* **32**:137-161.

[Hoare 1969] Hoare C A R 1969 An axiomatic basis for computer programming, *Comm. ACM* **12**,10:576-580, 583.

[Hoare 1984] Hoare C A R 1984 *Communicating sequential processes* (New York:Prentice-Hall).

[Katz et al 1989] Katz S, Peled D 1989 An efficient verification method for parallel and distributed programs, *LNCS* **354**:489-507.

[Lodaya et al 1991] Lodaya K, Parikh R, Ramanujam R, Thiagarajan P S 1991 Logics for distributed transition systems (in preparation).

[Lodaya et al 1989a] Lodaya K, Ramanujam R, Thiagarajan P S 1989a A logic for distributed transition systems, *LNCS* **354**:508-522.

[Lodaya et al 1989b] Lodaya K, Ramanujam R, Thiagarajan P S 1989b Temporal logics for communicating sequential agents: I, *Report* **IMSc/89/15**, Inst. Math. Sc., Madras, India (to appear in *Int. J. Found. of Comput. Sci.*).

[Lodaya et al 1987] Lodaya K, Thiagarajan P S 1987 A modal logic for a subclass of event structures, *LNCS* **267**:290-303. [Also Lodaya K, Thiagarajan P S 1989 A correction to "A modal logic for a subclass of event structures," *Report* **DAIMI-PB-275**, Computer Science Dept, Århus Univ, Århus, Denmark.]

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

[Milner 1989] Milner R 1989 *Communication and concurrency* (New York:Prentice-Hall).

[Mukund 1990] Mukund M 1990 Expressiveness and completeness of a logic for well branching prime event structures, *Report* **TCS-90-1**, School of Math., SPIC Science Foundation, Madras, India.

[Mukund et al 1989] Mukund M, Thiagarajan P S 1989 An axiomatization of event structures, *LNCS* **405**:143-160.

[Mukund et al 1991] Mukund M, Thiagarajan P S 1991 A logical characterization of well branching event structures (to appear in *Theor. Comput. Sci.*).

[Nielsen et al 1980] Nielsen M, Plotkin G, Winskel G 1980 Petri nets, event structures and domains I, *Theor. Comput. Sci.* **13**,1:86-108.

[Nielsen et al 1990] Nielsen M, Rozenberg G, Thiagarajan P S 1990 Behavioural notions for elementary net systems, *Distr. Comput.* **4**,1:45-57.

[Ochmanski 1985] Ochmanski E 1985 Regular trace languages, Ph.D. thesis, Warsaw, Poland [summary in 1985 *Bull. EATCS* **27**.]

[Olderog 1987] Olderog E-R 1987 Operational Petri net semantics for CCSP, *LNCS* **266**:196-223.

[Parikh 1989] Parikh R 1989 Decidability and undecidability in distributed transition systems, in Narasimhan R (ed.): *A perspective in theoretical computer science — commemorative volume for Gift Siromoney* (Singapore:World Scientific) pp. 165-198.

[Peleg 1987] Peleg D 1987 Concurrent dynamic logic, *J. ACM* **34**,2:450-479.

[Penczek 1988] Penczek W 1988 A temporal logic for event structures, *Fund. Inf.* **XI**:297-326.

[Pinter et al 1984] Pinter S, Wolper P 1984 A temporal logic for reasoning about partially ordered computations, *Proc. 3rd ACM Conf. Princ. of Dist. Comput.* pp. 28-37.

[Plotkin 1981] Plotkin G 1981 A structural approach to operational semantics, *Report* **DAIMI FN-19**, Computer Science Dept, Århus Univ, Århus, Denmark.

[Pnueli 1977] Pnueli A 1977 The temporal logic of programs, *Proc. 18th IEEE Conf. Found. of Comput. Sci.* pp 46-57.

[Pratt 1986] Pratt V 1986 Modelling concurrency with partial orders, *Int. J. Parallel Programming* **15**,1:33-71.

[Reisig 1986] Reisig W 1986 Towards a temporal logic for causality and choice in distributed systems, *LNCS* **354**:603-627.

[Stark 1989] Stark E W 1989 Concurrent transition systems, *Theor. Comput. Sci.* **64**:221-269.

[Stirling 1987] Stirling C 1987 Modal logics for communicating systems, *Theor. Comput. Sci.* **49**:311-347.

[Thiagarajan 1990] Thiagarajan P S 1990 Some behavioural aspects of net theory, *Theor. Comput. Sci.* **71**:133-153.

[Wang 1961] Wang H 1961 Proving theorems by pattern recognition II, *Bell Syst. Tech. J.* **40**:1-41.

[Winskel 1982] Winskel G 1982 Event structure semantics for CCS and related languages, *LNCS* **140**:561-577.

[Winskel 1987] Winskel G 1987 Event structures, *LNCS* **255**:325-392.

[Zielonka 1987] Zielonka W 1987 Notes on finite asynchronous automata, *RAIRO Inf. Th. et Appl.* **21**,2:99-135.