ASYNCHRONOUS DYNAMICAL SYSTEMS
PART I: DETERMINISTIC DYNAMICS

by

Kemal Inan

Memorandum No. UCB/ERL M89/59

17 May 1989
(Revised June 23, 1989)

ASYNCHRONOUS DYNAMICAL SYSTEMS
PART I: DETERMINISTIC DYNAMICS

by

Kemal Inan

Memorandum No. UCB/ERL M89/59

17 May 1989
(Revised June 23, 1989)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ASYNCHRONOUS DYNAMICAL SYSTEMS
PART I: DETERMINISTIC DYNAMICS

by

Kemal Inan

Memorandum No. UCB/ERL M89/59

17 May 1989
(Revised June 23, 1989)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ASYNCHRONOUS DYNAMICAL SYSTEMS PART I : DETERMINISTIC DYNAMICS [1]

Kemal Inan

Department of Electrical Engineering and Computer Sciences
and Electronics Research Laboratory
University of California, Berkeley, CA 94720

ABSTRACT

This is the first part of a three part paper on symbolic asynchronous dynamics in which the underlying deterministic dynamical system formalism is developed . An asynchronous dynamical system (ADS) is an abstract model for a symbolic event driven system. It is based on a functional input-output representation and it differs from the existing models of discrete event systems in this respect . Input , output and state signals in an ADS are represented by 'marked processes' , a concept recently introduced as a unifying model for discrete event processes. Unlike conventional dynamical systems that operate arithmetically on synchronous collection of signals ADS operates on signals with interleaved sequences of symbolic events .

In this paper the basic concepts of ADS are derived. Algebraic operators that model a heterogenous deterministic environment of interacting discrete event processes are developed , definition and basic properties of ADS are presented and illustrated with examples.

[1] Research supported in part by NSF Grant ECS 8719779, by the MICRO program of the State of California and a grant from Pacific Bell .

1. Introduction

A discrete event process is a sequence of events each of which signify a qualitative change in the state of a system. A digital communication network , a flexible manufacturing environment or an operating system of a computer are examples that incorporate discrete event processes .

Discrete event process models can be timed or logical . In logical models discrete event processes are defined by constraining the order of events in sequences . A logical discrete event signal is represented by a collection of feasible event sequences . Models for discrete event processes include finite state machines , petri-nets [10], extended state machines [11], , finitely recursive processes [1] and many others . These models are finite symbol representations for processes just as the set of equations $\{\dot{x}=-x \ ; x(0)=1\}$, is a finite symbol representation for the signal e^{-t}.

A discrete event system operates on discrete event processes and generates new discrete event processes. For example a computer operating system has to respond to external events such as interrupt signals that require immmediate servicing or explicit user demands for compiling or running programs entered through user terminals. In general the external events may represent a multiplicity of demands that await servicing from the operating system which responds to these demands by servicing the interrupt signal , running a program etc. in a logically pre-specified order. By responding it generates new (output) events corresponding to external (input) demands. Since all this happens in real time such a system can be conceptualized as a dynamical system operating on some abstract space of input signals to generate output signals. Therefore it makes good sense for an abstraction of a discrete event system to recognize , either implicitly or explicitly , the functional nature of input-output dynamics. Here we are pointing to a difference of emphasis between a discrete event *process* and a discrete event *system* model . Existing models in literature have a strong *process* bias whereby recognition of input-output dynamics is implicit , usually with little , if any , analytical use.

The driving force for discrete event formalisms has come from problems of computer science. Exploiting parallelism in computation gave rise to new formalisms for expressing complex execution sequences . Models for discrete event processes have since been used as semantic models for parallel languages . CSP [3] , CCS [4] , are among the best known examples of such semantic models.

In actual practice input-output classification of events has a descriptive value for discrete event systems. For example in a relatively complex protocol specification [6] , the inputs - messages at the input channel- are described as external events and the outputs are described as the sequence of actions to be taken upon receiving an external input. Clearly the descriptive value of this specification is useful. On the other hand underlying models of such descriptions - in this case an event driven extended state machine - do not treat input-output relations at the functional level of *processes* and therefore do not generate tools of analysis in the spirit of control systems . As another example take CSP or CCS both of which differentiate between immediate inputs and outputs , and has special notations for the execution , synchronization and hiding of input and output events . There are input-output operators in CSP such as the "piping" operator of UNIX that describe the restricted parallelism in file transfers. In short notwithstanding the abundance of operators that mimic well-known multi-programming features , concern and use for a functional input-output representation is nonexistent.

In general semantic models for languages do not recognize control action as a theoretical category . This category is presumably aggregated in the actual practice of (real-time) programming. Therefore dynamic input-output representation , which is of central interest to control theory is alien to semantic models of computer science. In contrast , the supervisory control framework [5] is a control oriented approach to discrete event systems in which the control action is a central category. However , the recognition of inputs and outputs in supervisory control is implicit. Instead of input or output events or

processes there are control actions that block or initiate events. The theory has developed by viewing control as a mechanism of restricting trajectories, much in the spirit suggested by Willems [7] . Among other approaches only [8] and [9] have explicitly used input-output formulations and have pursued this to the extent of exploiting some control-theoretic advantage of their respective formulations.

We suggest and partially demonstrate in these papers that functional input-output representation , which is the main thrust of ADS , is a useful formalism to model and solve problems for discrete event systems . In the second and third parts of this series of papers we extend these results to an operationally feasible non-deterministic environment and formulate the interconnection problem of ADS's by a block diagram algebra and an associated set of topological conventions - not unlike the block diagrams of control systems - and demonstrate that formal specification and verification problems for evaluating the logical performance of complex discrete event systems can be reduced to block diagram simplification procedures . In particular it will be shown that the complexity of computing a response function is dependent upon the largest nested loop in a system of interconnected ADSs and does not necessarily grow exponentially with the size and the number of the component ADSs . These results , which are encouraging in themselves , point to the potential in applying input-output analysis techniques to discrete event systems in the area of supervisory control , an endeavor to be undertaken as an independent task in future .

The way we define an input-output map in an asynchronous discrete event environment is related to the projection of signals onto signal spaces. We first explain the situation in a conventional discrete time system .

Suppose a system is described by the following equations :

$$f(x(k+1),x(k)) = 0$$

where the vector signal x has n components and f has $m < n$ components. In other words there is $n-m$ degree of freedom in these equations. If f satisfies certain requirements then the specification of $n-m$ components of the time sequence $x(k)$ as the *input* signal , together with some boundary conditions , will uniquely determine the the remaining components of $x(k)$ a subset of which may be designated as the *output* . This can be viewed as projecting any feasible trajectory on the input and output signal spaces and then observing that projection on the input space uniquely determines the projection on the output space. Note that the projections are straightforward operations since each component of $x(k)$ is related to the others as a synchronous [2] component of a vector valued discrete-time signal. Each choice of components for the input and output signals satisfying a unique solvability requirement will induce a well-defined input-output map . Among these choices there is a distinguished one of practical significance which we usually call the response function of the system.

In discrete event systems the signals are not the components of a synchronous vector. Instead , all events are intertwined with each other according to the logic of event sequences in every trace of the system. Therefore the projection operator required to filter the input and the output signals is less obvious than the conventional synchronous case. This is one of the reasons why discrete event formulations have resisted , at least so far , functional input-output descriptions.

In this paper we develop a new model called an *asynchronous dynamical system* (ADS) for event driven systems. ADS is an abstract model that supports a heterogenous environment of input ,output and state processes based on *marked process* theory . In a recent paper [2] it was shown that different

[2] By a "synchronous component " we mean the self-evident fact that the components of a vector are composed of scalar signals evaluated at a common instant of time. This self-evident fact ceases to be true in an asynchronous environment.

representations for discrete event processes have a common underlying abstraction with useful algebraic properties. In this paper we use this unifying abstraction , namely marked processes , for the modelling of signals in ADS's. In section 2 we summarize the relevant results on marked processes . In section 3 we define and state the properties of the basic operators of ADS , namely the projection operator and the two sum operators. In section 4 we define the ADS and illustrate the concept with examples . We also discuss and derive results for extending the basic calculus operators of marked process spaces to ADS representations. Finally in section 5 we discuss our results .

2. Marked Processes

We review below the relevant results for marked processes. The reader is referred [1] and [2] for a detailed treatment.

Throughout, A is a fixed, finite set of events, A^* is the set of all finite sequences of events, called **traces**, including the empty trace $\langle\rangle$. $\langle a\rangle$ is the trace consisting of the single event a and $s\hat{}t$ is the trace obtaining by concatenating the traces s and t. $T \subseteq A^*$ is (prefix)-*closed* if $s\hat{}t \in T$ implies $s \in T$. Let $C(A^*)$ be the family of all closed sets. Let M be a (possibly infinite) fixed set of **marks** and let Ψ be a fixed family of functions from A^* into M such that if $\mu \in \Psi$ and $s \in A^*$, then the function $\mu/s \in \Psi$, where

$$\mu/s(t) := \mu(s\hat{}t)\ ,\ t \in A^*$$

We call the cartesian product $W := C(A^*) \times \Psi$ an **embedding** set. An element $w \in W$ is called a **(marked) process** [3] and is denoted as $w = (trw, \mu w)$; $trw \in C(A^*)$ is the set of traces of w and $\mu w \in \Psi$, regarded as a mapping $\mu w: trw \to M$, is called its marking function; finally, $\mu w(s)$ is the **marking** of the trace s.

If $w \in W$ and $s \in trw$, the **post-process** of w after s is the process $w/s \in W$ defined by

$$tr(w/s) := \{t \mid s\hat{}t \in trw\}, \quad \mu(w/s)(t) = \mu w/s(t) := \mu w(s\hat{}t)$$

The **choice function** combines arbitrary processes $w_1, \cdots, w_k$ in W, distinct events $a_1, \cdots, a_k$ in A and any mark m in M to give the process w denoted

$$w = [a_1 \to w_1 \mid \cdots \mid a_k \to w_k]_m$$

and defined by

$$trw := \{\langle\rangle\} \cup \bigcup_i \{\langle a_i\rangle\hat{}s \mid s \in trw_i\}$$

$$\mu w(\langle\rangle) := m\ ,\quad \mu w(\langle a_i\rangle\hat{}s) := \mu w_i(s)$$

The post-process and choice functions are the counterparts , in discrete event calculus , of differential and integral operators of ordinary calculus. They satisfy the familiar fundamental calculus relation :

$$w = [a_1 \to w/\langle a_1\rangle \mid \cdots \mid a_k \to w/\langle a_k\rangle]_m$$

where

$$\{\langle a_1\rangle, \cdots, \langle a_k\rangle\} = \{s \in trw \mid \#s = 1\}, \quad m = \mu w(\langle\rangle)$$

and $\#s$ is the number of events in trace s.

---

[3] Strictly speaking a marked process is a member of a marked process space. For a definition of the latter see below.

We call v a **subprocess** of w if $trv \subseteq trw$ and

$$\mu v(s) = \mu w(s) \quad \text{for } s \in trv$$

We shall use the notation $v \leq w$ to denote that v is a subprocess of w. It is easy to check that $\leq$ is a complete partial order on W ,i.e. every nondecreasing sequence (chain) v_i in W has a least upper bound (limit) $v \in W$ where

$$trv = \bigcup_i trv_i$$

and $\mu v(s) := \mu v_j(s)$ for any j for which $s \in trv_j$. We shall use the notation $v = \bigsqcup_i v_i$ to denote the limit of a chain v_i.

Finally for any non-negative integer n , define the trace projection operator $\uparrow n$ mapping W into itself by letting

$$tr(v \uparrow n) := (s \in trv \mid \#s \leq n)$$

and

$$\mu(v \uparrow n)(s) := \mu v(s) \quad \text{for } s \in tr(v \uparrow n)$$

The triple $(W, \leq, \{\uparrow n\})$ is called an **embedding space** [4] .

A (marked) **process space** $\Pi = (\Pi, \leq, \{\uparrow n\})$ is any subset of an embedding space $(W, \leq, \{\uparrow n\})$ satisfying the following four axioms.

Axiom of projection

$$P \in \Pi \Rightarrow P \uparrow n \in \Pi , \ n \geq 0$$

Axiom of post-processes

$$P \in \Pi \text{ and } s \in trP \Rightarrow P/s \in \Pi$$

Axiom of prefixing

Let $R, P_1, \cdots, P_k$ in Π be such that

$$R = [a_1 \rightarrow R/\langle a_1 \rangle \mid \cdots \mid a_k \rightarrow R/\langle a_k \rangle]_m , \quad R/\langle a_i \rangle \uparrow 0 = P_i \uparrow 0 , \ i = 1, \cdots, k$$

Then

$$[a_1 \rightarrow P_1 \mid \cdots \mid a_k \rightarrow P_k]_m \in \Pi$$

Axiom of completeness

If the chain $P_1 \leq P_2 \cdots$ in Π converges to P in W, then $P \in \Pi$

Elements of Π are called (marked) **processes**. Any subset of Π which also satisfies these axioms is called a process *subspace*. It is a fact that arbitrary intersection of process spaces in W is also a process space.

Generally, a process space is specified by an embedding space W and **marking axioms** that define a subset Π of W and which imply the four axioms above. The next result is quite important since it shows that marking axioms only need specify "local" behavior.

---

[4] For a more general definition of embedding space see [2].

Proposition 2.1

Let W be an embedding space. Let $W_0 \subseteq W\uparrow 0$ and $W_1 \subseteq W\uparrow 1$ be subsets satisfying the following consistency conditions:

(1) $W_1\uparrow 0 = W_0$

(2) If $w \in W_1$ and $<a> \in trw$, then $w/<a> \in W_0$

Then there exists a unique marked process space Π with $\Pi\uparrow 0 = W_0$ and $\Pi\uparrow 1 = W_1$. Conversely, if $\Pi \subset W$ is a marked process space, then $W_0 := \Pi\uparrow 0$ and $W_1 := \Pi\uparrow 1$ satisfy the consistency conditions.

Proposition 2.1 states that the marked process space Π is determined by specifying all "one-shot" processes of the type

$$w = [a_1 \rightarrow w_{01} \mid \cdots \mid a_k \rightarrow w_{0k}]_m$$

where $m \in M$, $a_i \in A$, $w_{0i} \in W_0$. Other processes in Π are obtained from one-shot processes using the first three axioms of a marked process space. A simple pictorial description of a marked process consists of an infinite tree where each branch represents an event transition and each node represents the mark associated with the unique trace from the root to the node in question. In terms of this description Proposition 2.1 states that the any marked process space is completely defined by its building blocks of depth one , i.e. trees of depth one corresponding to each element of W_1 . The construction of arbitrary processes in this space is accomplished by pasting any one of these trees to any leaf of the tree inductively.

Examples of marked process spaces are state machines, Petri Nets, CSP , and various extensions of these models . A function $F: \Pi \rightarrow \Pi$ is **continuous** if for every chain $X_1 \leq X_2 \leq \cdots$ in Π, $F(X_1) \leq F(X_2) \leq \cdots$ is also a chain and

$$\bigsqcup_n F(X_n) = F(\bigsqcup_n X_n)$$

F is **constructive** (*con*) if for every $X \in \Pi$ and $n \geq 0$

$$F(X)\uparrow n+1 = F(X\uparrow n)\uparrow n+1$$

F is **non-destructive** (*ndes*) if for every $X \in \Pi$ and $n \geq 0$

$$F(X)\uparrow n = F(X\uparrow n)\uparrow n$$

Intuitively, F is *con* if the $(n + 1)$st event executed by $F(X)$ is determined by the first n events that X executes. Evidently *con* implies *ndes*. If F has several arguments these definitions apply if they apply to each argument when others are fixed. Extension of these concepts to functions with domain and range consisting of different marked process spaces are straightforward. The properties of continuity, *con* and *ndes* are preserved under composition of functions. Furthermore, if F is *con* and G is *ndes* then $F \circ G$ and $G \circ F$ both are *con*.

The choice function is continuous and *con*; every projection operator $\uparrow n$ is continuous and *ndes*; finally, the post-process is a continuous partial function in the sense that if $X_1 \leq X_2 \cdots$ is a chain converging to X and $s \in trX$, then there is an integer I such that $X_I/s \leq X_{I+1}/s \leq \cdots$ is a chain converging to X/s.

The following existence result is fundamental to recursive representation of processes.

Theorem 2.1

Let Π be a process space and consider the recursive equation

$$X = F(X) \tag{2.1}$$

where $F = (F_1, \cdots, F_n)$: $\Pi^n \rightarrow \Pi^n$. Also given is a set of initial conditions

$$X_i \uparrow 0 = Z_{0i} \ , \ i = 1, \cdots, n \tag{2.2}$$

that is consistent, i.e.

$$Z_{0i} = F_i(Z_{01}, \cdots, Z_{0n}, U) \uparrow 0 \ , \ i = 1, \cdots, n$$

If each F_i is continuous in X then the process $Z = \bigsqcup Z^k$ is well-defined where

$$Z^0 := (Z_{01}, \cdots, Z_{0n}) \ , \quad Z^{k+1} := F(Z^k, U) \ , k \geq 0$$

and Z is the minimal solution of (2.1), (2.2), i.e. if X satisfies (2.1), (2.2), then $Z_i \leq X_i$, all i.
If in addition F is *con* in X then Z is the unique solution of (2.1) and (2.2) .

A *finitely recursive process* (FRP) P is a process represented by the equations

$$X = F(X)$$
$$P = G(X)$$

where the function F satisfies both the existence and uniqueness requirements of the theorem above and both F and G are composed of members of a family of functions that determine the algebraic structure of the representation. The reader is referred to [2] for a precise and general definition of FRP .

3. Projection and Sum Operators

In order to express and formulate the relationship of input,state and output signals of discrete event dynamical systems we rely on two operators called *projection* and *sum* operators . A third operator we use is the *internal sum* operator which is a restrictive version of the sum operator. In this section these operators are defined and their properties are summarized. The projection and sum operators are tracewise equivalent to the 'projection' and 'disjunction' operators defined in [12] and extend these to marked spaces.

The projection operator projects the traces of a process in a given marked process space on another process given in another marked process space . This is done by collapsing each trace of the original process by removing events from it . The events to be removed are determined dynamically by the target process onto which the projection is being performed in such a way that the resulting trace belongs to this target process .

We start by defining the projection of an arbitrary trace s on a (prefix-closed) set of traces . If $K \in C(A^*)$, we define $s\downarrow_K$, inductively as follows :

(1) $< >\downarrow_K := < >$

(2)

$$<a>\downarrow_K := \begin{cases} <a> & \text{if } <a> \in K \\ < > & \text{otherwise} \end{cases}$$

(3)

$$s\downarrow_K = (r\hat{}<a>)\downarrow_K := r\downarrow_K \hat{} <a>\downarrow_{K/(r\downarrow_K)}$$

where for $K \in C(A^*)$ we use the notation K/s to denote the post-process traces , namely

$$K/s := (t \in A^* \mid s^\wedge t \in K)$$

As an example take $s := abaac$ and $K := \{<>, <b>, ba, bc, bac\}$, then $s\downarrow_K = bac$.

Fact 3.1

(1)

$$\begin{aligned} &L\downarrow_K \subseteq K \\ &K \subseteq L \Rightarrow L\downarrow_K = K \end{aligned} \tag{3.1}$$

where $L\downarrow_K$ denotes the set of all $s\downarrow_K$ for $s \in L$.

(2) The trace projection distributes over the trace concatenation according to the following formula

$$(u^\wedge v)\downarrow_K = u\downarrow_K{}^\wedge v\downarrow_{K/(u\downarrow_K)} \tag{3.2}$$

Definition 3.1

Let $\Pi = \Pi(A, M)$ and $\hat{\Pi} = \hat{\Pi}(A, \hat{M})$ be two process spaces over the same event alphabet A and let P and $\hat{P}$ be processes in Π and $\hat{\Pi}$ respectively . The projection of P on $\hat{P}$ denoted $P.\hat{P}$ is defined as

$$tr(P.\hat{P}) := trP\downarrow_{tr\hat{P}}$$
$$\mu(P.\hat{P})(s) := \mu\hat{P}(s) \quad \text{for } s \in tr(P.\hat{P}) \subseteq tr\hat{P}$$

Fact 3.2

The projection map

$$(P, \hat{P}) \in \Pi \times \hat{\Pi} \rightarrow P.\hat{P} \in \hat{\Pi}$$

has the properties stated below :

(1) $P.\hat{P} \leq \hat{P}$.

(2) It is continuous in P but not necessarily [5] continuous in $\hat{P}$. A limited version of continuity is given by the following formulas where n and m are non-negative integers .

$$m \leq n \Rightarrow P.(\hat{P}\uparrow m) \leq P.(\hat{P}\uparrow n) \tag{3.3}$$

and

$$\bigsqcup_n (P.(\hat{P}\uparrow n)) = P.(\bigsqcup_n (\hat{P}\uparrow n)) = P.\hat{P} \tag{3.4}$$

(3) The following relations hold for arbitrary n :

$$P.(\hat{P}\uparrow n) = (P.(\hat{P}\uparrow n))\uparrow n = (P.\hat{P})\uparrow n \tag{3.5}$$

$$(P\uparrow n).\hat{P} \leq (P.\hat{P})\uparrow n \tag{3.6}$$

In particular the operator is *ndes* in $\hat{P}$ but not necessarily so in P .

---

[5] We use the word "necessarily" for a property , to remind the reader that in some special marked process space the property may well hold , but in general it does not. In all the counter-examples given in the appendix we use the simplest of process spaces , namely that without marks.

(4) The projection operator is not necessarily associative in its arguments , that is , in general :

$$(P.Q).R \neq P.(Q.R)$$

yet it satisfies the relation : [6]

$$s \in trP \Rightarrow s\downarrow_{(P.Q)} = s\downarrow_Q$$

and therefore the following idempotency condition :

$$P.(P.Q) = P.Q$$

For simpler notation we shall use left-to-right evaluation convention , that is we write $P.Q.R$ to mean $((P.Q).R)$.

(5) The post-process of a projected process is given by the following formula

$$(P.\hat{P})/s := \bigcup_{t \in T_s} ((P/t).(\hat{P}/s)) \tag{3.7}$$

where

$$T_s := (t \in trP \mid t\downarrow_{\hat{P}} = s)$$

and if $\{Q_t\}_{t \in T}$ is a family of processes with consistent marks (see Remark 3.1 below) then we define the union $\bigcup_{t \in T} Q_t$ as

$$tr(\bigcup_{t \in T} Q_t) := \bigcup_{t \in T} trQ_t$$

and the mark for each trace is uniquely given by the consistency requirement. If the relation $trP \subseteq tr\hat{P}$ holds then (3.7) reduces to the following important special case

$$(P.\hat{P})/s = (P/s).(\hat{P}/s) \tag{3.8}$$

(6) If

$$\hat{P} := (a_1 \rightarrow \hat{P}_1 \mid \cdots \mid a_n \rightarrow \hat{P}_n)_m$$

then

$$P.\hat{P} = (a_{k_1} \rightarrow \bigcup_{t \in T_{k_1}} [(P/(t^\wedge <a_{k_1}>)).\hat{P}_{k_1}] \mid \cdots \mid a_{k_i} \rightarrow \bigcup_{t \in T_{k_i}} [(P/(t^\wedge <a_{k_i}>)).\hat{P}_{k_i}])_m \tag{3.9}$$

where T_j is the set of all traces t of trP such that

(i) for any k , a_k is not in the trace t

(ii) $t^\wedge <a_j> \in trP$.

and the indices k_j correspond to non-empty T_{k_j}.

Remark 3.1

1 - The definition of process projection disregards the marks of the projected process P . Therefore , what is projected is in fact the traces of P .

---

[6] By a slight abuse of notation we write $s\downarrow_Q$ to mean $s\downarrow_{trQ}$ in terms of the previous notation .

2 - The relation (3.7) deserves explanation since for marked (deterministic) processes [7] union of processes is not always defined . This is because the two processes may share a trace with different assignment of marks giving rise to either inconsistency or the necessity to allow marking functions to be point-to-set mappings. On the other hand if the collection of processes assign consistent marks to shared traces this problem does not arise. For example let $[P]$ denote the set of all subprocesses of P [8] then the collection $[P]$ has the consistency property since $s \in trX \cap trY$ and $X,Y \in [P]$ implies that $\mu X(s) = \mu Y(s)$. This condition is satisfied in (3.7) since all the processes under the union symbol are in $[\hat{P}/s]$.

3- When both P and $\hat{P}$ are generated by state machines , the state machine for the process $P.\hat{P}$ can be constructed as described below :

> Let X and Y denote the set of states for P and $\hat{P}$ respectively. We choose the cartesian product $\tilde{X} \times Y$ for the state set of the process $P.\hat{P}$ where $\tilde{X}$ denotes the set of all subsets of X. The initial state is $(\tilde{x}_0, x_0)$ where x_0 is the initial state of $\hat{P}$ and $\tilde{x}_0$ is the set of all the states in P that can be reached from the initial state of P via transitions corresponding to events that are distinct from all the events of the single step transitions of $\hat{P}$ emanating from x_0. Given any state $(\tilde{x}, x)$ there is a transition of event $<a>$ from it if and only if
>
> (1) $<a>$ is a transition of $\hat{P}$ from x , and
>
> (2) $<a>$ is a transition from some $y \in \tilde{x}$ of P.
>
> The next state corresponding to the transition $<a>$ is $(\tilde{z}, z)$, where z is the next state in $\hat{P}$ corresponding to $<a>$ and $\tilde{z}$ is the set of all states that can be reached from any state in $\tilde{x}$ via a sequence of transitions distinct from all the transitions of $\hat{P}$ from x , except for a single $<a>$ within the sequence.

As an application of the procedure outlined above consider the simple marked processes P and Q given by the state machine diagrams in Fig. 3.1.(a) . The marking used is an assignment of a 0 or a 1 to each state as shown in the figure . Applying the above procedure to this example we obtain the processes $P.Q$ and $Q.P$ given in Fig. 3.1.(c) . In Fig. 3.1.(c) the states of the resulting processes are tagged in terms of the states of its argument processes . For example the initial state of $P.Q$ is the pair $(\{0,1\},0)$ where $\{0,1\}$ signifies the subset of states 0 and 1 of P and 0 signifies the corresponding state of Q . Observe that the states 1 and 3 of $P.Q$ are process-equivalent (i.e. they generate the same process starting from these initial states) and therefore we have $P.Q = Q$ for this example.

4 - It is a fact that if P and $\hat{P}$ are two FRP in different process spaces , then , in general , there is no guarantee that $P.\hat{P}$ is an FRP. This is because the complexity of the marked space that contains $\hat{P}$ may not be expressive enough to represent the projection of P in a finitely recursive way. As an example take P as the infinite state process represented by the finite recursion

$$Q = (a \rightarrow Q;P \mid c \rightarrow SKIP)$$
$$P = (b \rightarrow SKIP)$$

where ';' denotes the sequential operator which signifies that the process $Q;P$ should continue with Q when P terminates through the process $SKIP$ (for an exact definition of the sequential operator see [1]) and P generates the traces $a^n c b^n$ for arbitrary non-negative integer n . It is well known that the traces of P constitute a non-regular language and therefore cannot be generated by a finite state machine. Now if $\hat{P}$ is an element of the unmarked space Π where every FRP is also a finite state machine then

---

[7] For a discussion of non-deterministic marked processes see [2] .

[8] For further properties of $[P]$ see Fact 3.5 .

$P.\hat{P}$ may not possess an FRP representation in Π. In particular if $\hat{P}$ is defined by $tr\hat{P} := \{a,b,c\}^*$, then $P.\hat{P}$ does *not* have an FRP description since $trP = tr(P.\hat{P})$ and the former is an infinite state process.

The sum operation is a converse of the projection operation . It combines a given set of processes in a new sum process such that each component process P_k is recovered by projecting the sum process on P_k . The novelty of this operator is its heterogenuity. The arguments of a sum operator may belong to different marked process spaces such as Petri-Nets , finite state machines etc. , provided that they share the same event alphabet.

In order to define the sum operator in a general setting of marked process spaces certain technicalities have to be resolved . In particular we make sure that both at the embedding space level and the process space level we have well-defined *sum spaces* to incorporate sum processes. First we define a sum embedding space using two given embedding spaces. Let $W = W(A, M, \Psi)$ and $\hat{W} = \hat{W}(A, \hat{M}, \hat{\Psi})$ denote two embedding spaces both with the usual partial order '$\leq$' and the usual length projection operator '$\uparrow n$' . We define the **sum embedding space** , denoted $W \oplus \hat{W}$, as the one generated by the triple (A , $M \times \hat{M}$, Ψ^p) where '$\leq$' and '$\uparrow n$' are as before and Ψ^p is defined as the following subset of $\{(\phi \times \hat{\phi}) : C(A^*) \to M \times \hat{M}\}$:

$$\phi(s) := \psi(s\downarrow_w) \text{ and } \hat{\phi}(s) := \hat{\psi}(s\downarrow_v)$$
$$\text{for some } \psi \in \Psi , \hat{\psi} \in \hat{\Psi} \text{ and } w \in W , v \in \hat{W}$$

In this setting we inductively (on the length of its traces) define the sum operator below .

Definition 3.2

Let $w \in W$ and $v \in \hat{W}$ be given. Define the **sum** of w and v , denoted by $w \oplus v \in W \oplus \hat{W}$, inductively by :

$$s \in tr(w \oplus v) \Rightarrow \left[s\hat{}\langle a \rangle \in tr(w \oplus v) \iff (s\hat{}\langle a \rangle)\downarrow_w = s\downarrow_w \hat{}\langle a \rangle \text{ or } (s\hat{}\langle a \rangle)\downarrow_v = s\downarrow_v \hat{}\langle a \rangle \right]$$

$$\mu(w \oplus v)(s) := (\mu w(s\downarrow_w) , \mu v(s\downarrow_v)) \in M \times \hat{M}$$

When components are finite state machines the result of the sum operator is one version of the usual product state machine where each state is a pair of states from the component processes. As an example consider the processes P and Q given in Fig. 3.1.(a) . The sum process $P \oplus Q$ is given in Fig. 3.1.(b) . Each state of $P \oplus Q$ is labeled by ij corresponding to the states i and j in P and Q and the superscript of each event transition indicates whether the transition is a joint one or an isolated one belonging to only P or Q . In the figure each state is also marked with the appropriate pair of 0's and 1's in accordance with the product definition of marks .

The relevant properties of the sum operator are stated below.

Fact 3.3

(1) The formula for evaluating a post-process is given by :

$$(w \oplus v)/s = (w/(s\downarrow_w)) \oplus (v/(s\downarrow_v)) \tag{3.10}$$

provided that $s \in tr(w \oplus v)$ (otherwise the formula is not valid) and traces of a sum is larger than each individual trace in the sense given below

$$trw \bigcup trv \subseteq tr(w \oplus v) \tag{3.11}$$

(2) The sum operator is an associative and *ndes* operator of its arguments . However it is not necessarily a continuous function of its arguments .

(3) The sum operator is related to the projection operator by the following formulas :

$$(w \oplus v).w = w \tag{3.12}$$

$$u.(w \oplus v).w = u.w \tag{3.13}$$

and

$$u.(w \oplus v) \leq u.w \oplus u.v \tag{3.14}$$

So far the definition of sum operator is restricted to embedding spaces . In order to extend this definition to marked process spaces we need the notion of a sum process space . Let $\Pi\,(A\ ,M)$ and $\hat{\Pi}\,(A\ ,\hat{M})$ be two marked process spaces , embedded in W and $\hat{W}$ respectively . Define the sum marked process space $\Pi \oplus \hat{\Pi}$ as the unique process space generated by the pair (W_0^p , W_1^p) (see Proposition 2.1) where :

$$W_0^p := ((\Pi\uparrow 0)\oplus(\hat{\Pi}\uparrow 0))$$
$$W_1^p := ((\Pi\uparrow 1)\oplus(\hat{\Pi}\uparrow 1))\uparrow 1$$

where we have used the notation $A \oplus B$ to denote the set of all $a \oplus b$ with $a \in A$, $b \in B$.

Fact 3.4

The sum space is well-defined , that is :

(1) The generators (W_0^p , W_1^p) satisfy the requirements of Proposition 2.1 .

(2) $w \in \Pi$, $v \in \hat{\Pi} \Rightarrow w \oplus v \in \Pi \oplus \hat{\Pi}$ (i.e. $\Pi \oplus \hat{\Pi} \subseteq \Pi \oplus \hat{\Pi}$)

Let Π be a given process space. If P is a process in Π then it does not necessarily follow that the process that inherits an arbitrary subset of the traces together with their markings of P is itself in Π. If , however , the process space has the property that it includes all such elements we call it a solid space. All the commonly used marked process spaces are solid in this sense. Similarly given a solid process space Π we define any set $\Gamma \subseteq \Pi$ to be a solid set if all the subprocesses of the processes in Γ are also in Γ .

Example 3.1

Consider the embedding space W with no marks (i.e M is a singletone) and $A := \{a\ , b\}$. Define the generators (see Proposition 2.1)

$$W_0 := \{HALT\}$$
$$W_1 := \{(a \rightarrow HALT \mid b \rightarrow HALT)\}$$

where $HALT$ is the process with the empty trace. If Π is the process space generated by W_0 , W_1 then any process P in Π has the property that

$$<a> \in tr\,(P/s) \iff <b> \in tr\,(P/s)$$

for each $s \in trP$. Therefore although the process $(a \rightarrow HALT) \in W$ inherits a trace of $(a \rightarrow HALT \mid b \rightarrow HALT) \in \Pi$, it is not in Π and therefore Π is not a solid process space. To make it a solid space we replace W_1 by the version given below :

$$W_1 := \{(a \rightarrow HALT)\ , (b \rightarrow HALT)\ , (a \rightarrow HALT \mid b \rightarrow HALT)\}$$

We shall encounter subsets denoted by $[P]$, where $P \in \Pi$ and $[P]$ is defined as the set of all the subprocesses of P. The next result shows that the property of being a solid space is a local one. It also summarizes the properties of the set $[P]$.

Fact 3.5

(1) Π is a solid marked process space if and only if $\Pi\uparrow 1$ is a solid set in the embedding space W.

(2) If Π is a solid process space then for any $P \in \Pi$, $[P]$ is a solid and closed (i.e. every chain in $[P]$ has a limit in $[P]$) subset of Π that satisfies the property

$$s \in trX \cap trY \quad \text{and} \quad X,Y \in [P] \Rightarrow \mu X(s) = \mu Y(s)$$

Henceforth we assume that all process spaces are solid.

The property (2) above states that for all the processes within a set $[P]$ any trace uniquely determines the mark for that trace. Therefore within $[P]$ we can apply unions intersections or any other set operations to processes since every operation is uniquely identified by its traces.

The sum operator is the most liberal interleaving operator respecting determinism. It allows all possible interleavings of the transitions of its argument processes within bounds of determinism. In particular processes are not allowed to block transitions of each other. The following restrictive sum operator gives processes limited blocking power .

Definition 3.3

Let $X_1 \in \Pi_1$ and $X_2 \in \Pi_2$ then for any $x_1 \in [X_1]$ and $x_2 \in [X_2]$ we define the **internal sum**

$$x_1 + x_2 \in [X_1 \oplus X_2]$$

of these processes , relative to the cover sets $[X_1]$ and $[X_2]$ inductively as follows :

(1) $\mu(x_1 + x_2)(\langle\rangle) := \mu(x_1 \oplus x_2)(\langle\rangle)$

(2) $\langle a\rangle \in tr(x_1 + x_2)$ if and only if

$$\left\{\begin{array}{l} \langle a\rangle \in tr(x_1 \oplus x_2) \text{ and} \\ \langle a\rangle \notin trx_1 \Rightarrow \langle a\rangle \notin trX_1 \text{ and} \\ \langle a\rangle \notin trx_2 \Rightarrow \langle a\rangle \notin trX_2 \end{array}\right\}$$

(3) $(x_1 + x_2)/\langle a\rangle := x_1/(\langle a\rangle\downarrow_{X_1}) + x_2/(\langle a\rangle\downarrow_{X_2})$

where the internal sum on the right hand side is with respect to $[X_1/(\langle a\rangle\downarrow_{X_1})]$ and $[X_2/(\langle a\rangle\downarrow_{X_2})]$ respectively.

According to the definition above a process can block a transition $\langle a\rangle$ in the environment of summands if it does <u>not</u> make that transition itself but its cover process makes that transition . As an example consider the processes $p \in [P]$ and $q \in [Q]$ given by the state diagram in Fig. 3.1.(d) . The internal sum $p + q$ with respect to cover sets $[P]$ and $[Q]$ is given in Fig. 3.1.(e) . For example p blocks the transition $\langle a\rangle$ for q at the joint state (2,1) since at 2 the cover process P <u>makes</u> the transition $\langle a\rangle$ whereàs p does not. Note that in order to evaluate the traces of the process $p + q$ the traces of all four processes P ,Q ,p , and q must be evaluated jointly since cover processes dynamically progress with the original processes according to the definition.

Fact 3.6

(1) The internal sum operator is an associative ,*ndes* and **continuous** operator with the post-process formula

$$(w + v)/s = w/(s\downarrow_W) + v/(s\downarrow_V) \tag{3.15}$$

replacing that of (3.10) , where the cover sets for the internal sum on the left side are $[W]$ and $[V]$, and on the right side are $[W/(s\downarrow_W)]$ and $[V/(s\downarrow_V)]$ respectively .

(2) The internal sum operator is related to the projection and sum operators by the formulas

$$v = z_1.v_1 + \cdots + z_n.v_n \tag{3.16}$$

Each process z_j is arbitrary except for the trace constraint

$$trv \subseteq tr(z_j)$$

where $v := v_1 + \cdots + v_n$ and each v_j belongs to the cover set $[V_j]$.

$$(w + v).V \leq v \tag{3.17}$$

$$u.(W \oplus V) \leq u.W + u.V \tag{3.18}$$

where the cover sets are W and V in the last two cases .

(3) If $w + v \in [W \oplus V]$ where $W = V$ then

$$tr(w + v) = tr(w) \cap tr(v)$$

In other words every transition must be shared by both processes. Therefore by equating the cover sets one enforces total synchronization through the internal sum operation.

4. Asynchronous Dynamical Systems

In this section we introduce a new formalism called an *asynchronous dynamical system* (ADS). ADS is a discrete event system formalism using a functional input-output description. ADS differs from conventional dynamical systems that operate arithmetically on synchronous instants of signals . It operates on signals that are *symbolic* and *asynchronous* .

Definition 4.1

An **Asynchronous Dynamical System Representation** R is an ordered triple $R := (U,X,Y)$ where U,X and Y are marked processes in the process spaces Π_U , Π_X and Π_Y respectively. The processes U ,X and Y are called the **input** , **state** and **output** processes and the sets $[U]$,$[X]$ and $[Y]$ are called the corresponding **signal spaces** . Processes in either of these spaces are called **signals** . The **state function** of the representation R , $S_R : [U] \rightarrow [X]$, is defined by :

$$\begin{aligned} trS_R(u) &:= (s \in trX \mid s\downarrow_U \in tr(u)) \\ \mu S_R(u)(s) &:= \mu X(s) \; ; \; s \in trS_R(u) \subseteq trX \end{aligned} \tag{4.1}$$

and the **response function** $H : [U] \rightarrow [Y]$ is defined by

$$H_R(u) := S_R(u).Y \tag{4.2}$$

For simplicity in notation we shall drop the subscripts of the state and response functions whenever there is no ambiguity. The basic properties of state and response fuctions are summarized by the following fact.

Fact 4.1

(1) The state function S is well-defined and is the inverse projection of the input space on the state space . More precisely the set $trS(u)$ given by (4.1) is prefix-closed and therefore $S(u) \in [X]$ is well-defined on its domain $[U]$. Furthermore $S(u)$ satisfies

$$S(u).U \begin{cases} = u & \text{if } u \in [X.U] \\ \lesssim u & \text{if } u \in [U] \end{cases} \tag{4.3}$$

The set $[X.U]$ is called the **projected domain** of both the state and response functions . The characterizing property of the projected domain is given by

$$\begin{aligned} S(u) &= S(\hat{u}) \\ H(u) &= H(\hat{u}) \end{aligned} \tag{4.4}$$

where

$$\hat{u} := S(u).U$$

(2) The state function can be expressed in terms of the internal sum and projection operators as

$$S(u) = (X.(X \oplus U) + u).X \tag{4.5}$$

where the internal sum is defined with respect to the cover sets $[X \oplus U]$ and $[U]$ respectively .

(3) $S(.)$ is a continuous and *ndes* function and $H(.)$ is a continuous function on $[U]$. Moreover restriction of S to its projected domain $[X.U]$ is a one-to-one function.

Example 4.1

The following equations are an FRP description of an ADS where all spaces are taken as the unmarked space.

State Process X :

$$\begin{aligned} X_1 &= (n \rightarrow X_2) \\ X_2 &= (e \rightarrow X_3) \\ X_3 &= (f \rightarrow X_4 \mid c \rightarrow X_1) \\ X_4 &= (a \rightarrow X_1) \\ X &= X_1 \end{aligned}$$

where the e, f, c, n and a stand for the discrete events 'enter shop' , 'failed' , 'completed' , 'next job accepted ' , 'acknowledge failure ' respectively in a job shop managing process. The input process U is given by

$$\begin{aligned} U_1 &= (n \rightarrow U_2) \\ U_2 &= (f \rightarrow U_1 \mid c \rightarrow U_1) \\ U &= U_1 \end{aligned}$$

where the inputs f and c are feedback inputs coming from the actual machine shop and n is an input coming from the user. The output process Y is given by

$$Y = (e \rightarrow Y \mid a \rightarrow Y)$$

The triple $R = (U, X, Y)$ is an ADS representation. We compute the state function and the response function for the specific input signal u - representing success cycles - described below

$$u_1 = (n \rightarrow u_2)$$
$$u_2 = (c \rightarrow u_1)$$
$$u = u_1$$

The corresponding state and response functions can be computed by inspection as

$$s_1 = (n \rightarrow s_2)$$
$$s_2 = (e \rightarrow s_3)$$
$$s_3 = (c \rightarrow s_1)$$
$$S(u) = s_1$$

and

$$H(u) = (e \rightarrow H(u))$$

We can remodel the external complexity of the system by internalizing some of the input and output transitions to private state transitions. The new ADS representation $\hat{R} = (\hat{U}, X, \hat{Y})$ has the modified version of the input and output spaces where events that are relevant only to direct machine shop operations are eliminated as shown below.

$$\hat{U} = (n \rightarrow \hat{U})$$
$$\hat{Y} = (a \rightarrow \hat{Y})$$

For this representation the events e , f and c are pure state events invisible at the input and the output. The state function evaluated at the input signal $\hat{u} := (n \rightarrow HALT)$ is given by

$$s_1 = (n \rightarrow s_2)$$
$$s_2 = (e \rightarrow s_3)$$
$$s_3 = (f \rightarrow s_4 \mid c \rightarrow HALT)$$
$$s_4 = (a \rightarrow HALT)$$
$$S(\hat{u}) = s_1$$

and the response function is $H(\hat{u}) = (a \rightarrow HALT)$.

Example 4.2

Consider the ADS defined by the processes given in Fig. 4.1 . which describe a simple flow control mechanism using a Petri-Net and two finite state machines. The input and state processes are given by the state machines of Fig. 4.1.(a) and Fig. 4.1.(b) , and the output process is given by the Petri-Net described by Fig 4.1.(c) . The input transitions "+" and "-" adjust the flow into the place 4 of the output net by controlling the transitions <b> and <c> by either blocking or releasing them .

The state signals $S(u_1)$ and $S(u_2)$ for the two inputs u_1 and u_2 where $tru_1 := <+>$ and $tru_2 := +-$ respectively are given in Fig. 4.2.(a) and Fig. 4.2.(b) . The corresponding outputs are identical , i.e. $y := H(u_1) = H_2(u_2)$ where y is given by the Petri-Net in Fig 4.2.(c) . Note that the reason the inputs generate the same outputs is because the effect of the transition '+' is rich enough to contaminate the output signal even though '-' again restricts the $<b>$ transitions. This is one peculiarity of untimed models when the response to an earlier input - in this case to input transition + - unleashes an unbounded set of trajectories as in this example. In order to observe the output restricting effect of the input transition – after + , we need a model after a specific state trace . For example the representation $\hat{R} := (U/(+-), X/s, Y/(s\downarrow_Y))$ where s is any state trace such that $s\downarrow_U = +-$ will capture this effect and the output $H_{\hat{R}}(\phi)$ then corresponds to the Petri-Net given by Fig 4.2.(d) - with an initial token adjustment to account for the trace s - .

Also observe that when the state signal makes any of the the transitions of the output Petri-Net then the Petri-Net follows this only if its token condition is met as required by the definition of projection . Otherwise this becomes a private state transition without causing an output transition. Therefore in this model the *same* event can generate both output and private state transitions .

Let us now modify this model by interchanging the role of the state and output spaces. More precisely suppose that we choose the input and output spaces as simple as possible , namely those given in Fig.4.3.(a) and Fig.4.3.(b) . Can we choose the state space in such a way that the resulting input-output relation will behave as before as far as the traces of the responses are involved ? The answer is choosing a variable topology Petri-Net such that the inputs switch the Petri-Net from one topology to the other as suggested in Fig.4.3.(c) . In general topologically switched Petri-Nets are *not* Petri-Nets. Nevertheless they can be formulated as extended FRP in appropriate process spaces (see [2] p. 33 , example 4 for finitely recursive representation of Petri-Nets). Apart from the representation problem of state space the output space becomes too small in expressibility of finite representations of the signals $H(u)$ [9] . In this sense the initial model of this example looks , at least formally , easier to handle .

Dynamical systems are interconnected with others to form more complex systems. Therefore we need a compact representation for an ADS that has two capabilities : (i) it should characterize its internal input-state-output dynamics ; (ii) it should define the interacting capacity of the system with other ADS through input and output synchronization. We have already seen that the autonomous dynamics of an ADS is completely characterized by its state process X. Once X is given the state or output signals can be expressed for a given input signal u using (4.5). On the other hand interaction with other ADS can be modeled by making use of the internal sum operations and cover sets. For this we characterize ADS's as *signals* in appropriate cover sets that completely determine the environment of synchronization [10] . These considerations are formally elaborated below .

Definition 4.2

For an ADS representation $R=(U,X,Y)$ the **representation signal** $r(u)$ is defined as

$$r(u) := S(u).(X \oplus U \oplus Y) \tag{4.7a}$$

with the cover set $[X \oplus U \oplus Y]$ and the **system signal** $d(u)$ is defined as

$$d(u) := S(u).(U \oplus Y) \tag{4.7b}$$

with the cover set $[U \oplus Y]$.

The representation signal $r(u)$ simultaneously generates the traces of the corresponding state signal ,i.e.

$$tr(r(u)) = tr(S(u)) \tag{4.8}$$

and represents the blocking power of the system interconnected to other systems using the internal sum operation. For example if $\hat{R}$ is another representation with an input process $\hat{U}=Y$, then the signal $r(u) + \hat{r}(\hat{U})$ generates the traces of the interconnected process where the output of R is connected to (synchronized by) the input of $\hat{R}$.

---

[9] See 4 - of Remark 3.1 .

[10] For an application of this construct to the general interconnection algebra of ADS's see part III of this series of papers .

Every transition in $S(u)$ is visible in $r(u)$ since $S(u)$ also appears on the right side of the projection operator (cf. (3.8)). On the other hand the system signal $d(u)$ is the externally visible version of $r(u)$. The difference between $r(u)$ and $d(u)$ is that in $r(u)$ "pure" state transitions (i.e. transitions that are neither input nor output) are allowed to appear whereas in $d(u)$ they are masked out. Both signals can explicitly be expressed as functions of u using internal sums

$$\begin{aligned} r(u) &= (r + u).r \\ d(u) &= (d + u).d \end{aligned} \tag{4.9}$$

where

$$\begin{aligned} r &:= X.(X \oplus U \oplus Y) \\ d &:= X.(U \oplus Y) \end{aligned} \tag{4.10}$$

We now relate the representation signal to the basic calculus of processes. First we express its evolution in terms of post-processes . Using (3.8) and (3.10) we have

$$r(u)/s = S(u)/s.(S(u)/s \oplus U/(s\downarrow_U) \oplus Y/(s\downarrow_Y))$$

Motivated by the *post-representation signal* above we define the **post-representation** after the execution of the state trace s as $R/s := (U/(s\downarrow_U), X/s, Y/(s\downarrow_Y))$. Similarly for the length projection operator using (3.5) we have

$$r(u)\uparrow n = (S(u)\uparrow n).(S(u)\uparrow n \oplus U\uparrow n \oplus Y\uparrow n)$$

and as above we define the **n-projected representation** as $R\uparrow n := (U\uparrow n, X\uparrow n, Y\uparrow n)$

Fact 4.2

(1) The state and the response functions for the representations R and R/s are related as follows

$$\begin{aligned} S_{R/s}(u) &= S_R(\hat{u}(u))/s \\ H_{R/s}(u) &= H_R(\hat{u}(u))/(s\downarrow_Y) \end{aligned} \tag{4.11}$$

for all $u \in [U/(s\downarrow_U)]$, where $\hat{u}(u) \in [U]$ is defined by :

$$\begin{aligned} tr\hat{u}(u) &:= (s\downarrow_U \hat{}\, t \mid t \in tr(u)) \\ \mu\hat{u}(u)(t) &:= \mu U(t)\ ;\ t \in tr\hat{u}(u) \end{aligned} \tag{4.12}$$

(2) Similarly the state and response functions of R and $R\uparrow n$ are related by

$$\begin{aligned} S_{R\uparrow n}(u) &= S_R(u)\uparrow n \\ H_{R\uparrow n}(u) &= S_{R\uparrow n}(u).(Y\uparrow n) \end{aligned} \tag{4.13}$$

Finally we apply the choice operator to signals . Let $R_j = (U_j , X_j , Y_j)$ be a collection of J ADS representations such that for each j $U_j \in \Pi_U$, $X_j \in \Pi_X$ and $Y_j \in \Pi_Y$, that is , all input processes belong to the same marked process space and similiar conditions hold for state and output processes. Now define a process r via the choice function as :

$$r := (a_1 \rightarrow r_1 \mid \cdots \mid a_J \rightarrow r_J)_{(m_X, m_U, m_Y)} \in \Pi_X \oplus \Pi_U \oplus \Pi_Y \tag{4.14}$$

where each r_j is given by

$$r_j = X_j.(X_j \oplus U_j \oplus Y_j)$$

Unfortunately this definition does not uniquely identify the signal r with a representation triple (U, X, Y) as is the case for the signals r_j . This is because the input and output processes U and Y are

unspecified . On the other hand if U and Y are fully specified then r may be inconsistent with (4.14) in the sense that the inputs U and Y are not connected to individual U_j and Y_j in a meaningful way . In order to resolve this problem of partial specification we define the concept of integration below.

Definition 4.3

Given the collection of representations R_j above , a representation $R = (U,X,Y)$ is said to **integrate** this collection if the following equations are satisfied :

$$\begin{aligned} &X = (a_1 \to X_1 \mid \cdots \mid a_J \to X_J\ )_{m_x} \\ &U/(<a_j>\downarrow_U) = U_j \\ &Y/(<a_j>\downarrow_Y) = Y_j \end{aligned} \tag{4.15}$$

for events a_j ; $j = 1, \cdots, J$. Under these conditions r given by (4.14) is the representation signal $r(u)$ evaluated at $u=U$ for the integral representation R .

According to this definition , integration of a given set of representations corresponds to some prefixing of the state processes of the collection in a choice function such that by an incremental specification of the status - input , output or internal state - of each transition of the choice function the *minimal* [11] input and output processes U and Y are determined if they are to satisfy the integration requirements given by (4.15). Equations (4.15) are important in recovering the functional description of the input and output processes from the usual form of supplied data on the status of individual transitions , that is , whether a transition is input or output or neither (internal state) [12] . The pointwise data is *integrated* in a recursive manner using the FRP description of the state process X. This is illustrated by the following example.

Example 4.2

Consider the following process X expressed in FRP formulation

$$\begin{aligned} X_1 &= (a_1 \to X_2) \\ X_2 &= (b_1 \to X_1 \mid a_2 \to X_3) \\ X_3 &= (b_1 \to X_4) \\ X_4 &= (a_1 \to X_5) \\ X_5 &= (b_2 \to X_2) \\ X &= X_1 \end{aligned}$$

We are given the transition status information that in each context a_1,a_2 and b_1,b_2 are the input and output transitions respectively. We are asked to find the minimal input space $[U]$ and the output space $[Y]$ consistent with this data by using (4.15) recursively. We take all the processes to be unmarked . Let U_j and Y_j be the variables at each stage j corresponding to X_j. Using the integration definition and assuming that each FRP equation integrates the collection of the processes on its right side we have

$$U_1 \geq (a_1 \to U_2)$$

which is obtained from (4.15) by applying it to the first equation of the FRP description and observing that

---

[11] By minimal we mean the smallest processes according to the partial order on process spaces .

[12] For example the IEEE 800 series data link protocol [6] is described by an implied state machine model , listing the sequence of actions to be taken (outputs) after each specific external input . The procedure of Example 4.2 using (4.15) can be used to compute the " functional " input and output processes from this pointwise data.

$$U_1/(<a_1>\downarrow_{U_1}) = U_1/<a_1> = U_2$$

where we used the given data that $<a_1>$ is an input transition. The reason for the inequality is because (4.15) is a necessary condition and the input could well be larger. Although we are after the minimal input process for reasons of consistency we impose this at the end. Similarly we write for the output process the following equations

$$Y_1 = Y_2$$

since again by (4.15)

$$Y_1/(<a_1>\downarrow_{Y_1}) = Y_1$$

where the projection is reduced to the null event .

The rest of the equations are given below :

$$U_2 \geq (a_2 \rightarrow U_3) \; ; \; U_2 = U_1$$
$$Y_2 \geq (b_1 \rightarrow Y_1) \; ; \; Y_2 = Y_3$$
$$U_3 = U_4 \; ; \; Y_3 \geq (b_1 \rightarrow Y_4)$$
$$U_4 \geq (a_1 \rightarrow U_5) \; ; \; Y_4 = Y_5$$
$$U_5 = U_2 \; ; \; Y_5 \geq (b_2 \rightarrow Y_2)$$

The minimal solution to these equations is given below

$$U_1 = U_2 = U_5 \; ; \; U_3 = U_4$$
$$Y_1 = Y_2 = Y_3 \; ; \; Y_4 = Y_5$$

and

$$U_1 = (a_1 \rightarrow U_1 \mid a_2 \rightarrow U_3)$$
$$U_3 = (a_1 \rightarrow U_1)$$

In computing the output process unless we assume that $Y_1 = Y_4$ we may end up in a non-deterministic solution (transition $<b_1>$ takes $Y_2 = Y_3$ into both Y_1 and Y_4). Therefore the minimal (deterministic) output solution is

$$Y = (b_1 \rightarrow Y \mid b_2 \rightarrow Y)$$

where Y is the process that equals all Y_j .

Remarks

(1) In the context of a state machine the graphical interpretation of solving (4.15) is simple : tag each transition as an input or an output (pointwise information) and obtain the input process by hiding all output transitions and vice-versa. If non-determinacy arises then further merge all the target states of that non-deterministic transition.

(2) The procedure above computes the minimal solution. In instances where inter-process blocking through internal sums is desired larger solutions may be selected .

The concept of integration is also important in checking properties of response fuctions. For example one can attribute an ADS "maximum causality" if its response function maps trajectories (i.e. signals with a single trace and its prefixes) in $[U]$ into trajectories in $[Y]$. The conditions under which a given transition status in the FRP formulated state process of an ADS satisfies the maximum causality

requirement is not immediate. Deriving these conditions in a more general setting makes use of the concept of integration. For this reason we present the following properties of the integrated process without proof [13] .

Fact 4.3

(1) The representation signal for the integrated process is given by

$$r(u)= \{[j \in J_1:(a_j \rightarrow r_j(u/<a_j>))] \mid [j \in J_2:(a_j \rightarrow r_j(u))]\}_{(m_X,m_U,m_Y)} \tag{4.16}$$

where J_1,J_2 are subsets of $\{1,...,J\}$ defined by

$$j \in J_1 \iff <a_j> \in tr(u)$$
$$j \in J_2 \iff <a_j> \notin trU$$

and we used an indexed notation for the choice function.

(2) Similarly the integrated state function is given by

$$S(u)= \{[j \in J_1:(a_j \rightarrow S_{R_j}(u/<a_j>))] \mid [j \in J_2:(a_j \rightarrow S_{R_j}(u))]\}_{m_X} \tag{4.17}$$

(3) The expression for the response function is more complex and is given by :

$$H(u)= \{[j \in I_1:(a_j \rightarrow H_{R_j}(u/<a_j>))] \mid [j \in I_2:(a_j \rightarrow H_{R_j}(u))]\}_{m_Y} \bigcup (\bigcup_{j \in J} (H_{R_j}(u_j))) \tag{4.18}$$

where the index sets are defined by

$$\begin{aligned} j \in I_1 &\iff <a_j> \in tr(u) \cap trY \\ j \in I_2 &\iff <a_j> \notin trU \text{ and } <a_j> \in trY \\ j \in J &\iff <a_j> \notin trY \text{ and } j \in J_1 \bigcup J_2 \end{aligned} \tag{4.19}$$

and

$$u_j := \begin{cases} u/<a_j> & \text{if } j \in J_1 \\ u & \text{otherwise} \end{cases} \tag{4.20}$$

Note that the union in (4.18) is well-defined since all the processes are subprocesses of Y and therefore have consistent marks.

5. Conclusions

We presented a new logical model called an asynchronous dynamical system (ADS) for discrete event systems. The model is distinguished by its functional input-state and input-output maps derived from projection and sum operators defined on spaces of marked processes. These operators allow for a variety of discrete event models to participate in a heterogenous synchronization environment. In the second part of this series of papers we generalize these ideas to nondeterministic dynamics and in the third part we utilize these ideas to construct an interconnection algebra and demonstrate that formal verification problems can be reduced to methods of block diagram reduction.

We have not touched upon problems of nondeterminism arising out of our definition , inparticular out of the projection operator. Clearly the implicit assumption of determinism which states that a process P will progress after a trace s if and only if the process P/s is not the null process (i.e. the process with only the empty trace $<>$)is not valid for the output signals in ADS formulation even if the

[13] We deal with problems causality in ADS in a forthcoming paper.

input and state signals are assumed to behave deterministically . We shall deal with this issue in part II of this paper by developing an appropriate equivalence definition that captures the semantics of non-determinism at a reasonable operational level .

It may be fruitful to investigate the conditions under which a finitely recursive process algebra couples to the algebra of ADS. By this we mean the investigation of how process operators such as sequential or parallel composition [1] , or others , interact with the projection and sum operators of ADS. Preservation of properties of algebraic structures under ADS operations may yield useful insights and simplifications in the analysis and design of discrete event systems .

APPENDIX

Proof of Fact 3.1

The proof of (3.1) is routine . To prove (3.2) we use induction on the length of v . By definition

$$(u\hat{}v\hat{}<a>)\downarrow_K = (u\hat{}v)\downarrow_K \hat{} <a>\downarrow_{K/(u\hat{}v\downarrow_K)}$$

Applying the induction hypothesis

$$u\hat{}v\hat{}<a>\downarrow_K = u\downarrow_K \hat{} v\downarrow_{K/(u\downarrow_K)} \hat{} <a>\downarrow_{K/(u\hat{}v\downarrow_K)}$$

$$= u\downarrow_K \hat{} (v \hat{} <a>)\downarrow_{K/(u\downarrow_K)}$$

where we used the transitivity relation

$$(K/(u\downarrow_K))/(v\downarrow_{K/(u\downarrow_K)}) = K/((u\downarrow_K) \hat{} (v\downarrow_{K/(u\downarrow_K)})) = K/(u \hat{} v)\downarrow_K$$

which completes the proof .

Proof of Fact 3.2

(1) Follows by definition of the projection operator .

(2) First consider the following counter-example to show that the projection operator is not necessarily continuous in $\hat{P}$:

Take

$trP = \{<>,<b>,ba\}$
$tr\hat{P} = \{<>,<b>,<a>\}$
$tr\tilde{P} = \{<>,<a>\}$

then although $\tilde{P} \leq \hat{P}$ we have

$tr(P.\tilde{P}) = \{< >,<a>\}$
$tr(P.\hat{P}) = \{< >,<b>\}$

which violates the monotonocity requirement of continuity.

Next we claim that

$$\#(s\downarrow_P) \leq n \Rightarrow [s\downarrow_P = s\downarrow_{P\uparrow n}] \qquad \text{(A1)}$$

Proof of Claim :

We use induction on the length of s in (A1) . Let $s = v\hat{}<a>$ then by (3.2)

$$s\downarrow_P = (v\hat{}<a>)\downarrow_P = v\downarrow_P \hat{} <a>\downarrow_{P/(v\downarrow_P)}$$

$$s\downarrow_{P\uparrow n} = (v\hat{}<a>)\downarrow_{P\uparrow n} = v\downarrow_{P\uparrow n} \hat{} <a>\downarrow_{(P\uparrow n)/(v\downarrow_{P\uparrow n})}$$

and using the induction hypothesis , namely ,

$$w := v\downarrow_P = v\downarrow_{P\uparrow n}$$

it remains to prove that

$$<a>\downarrow_{P/w} = <a>\downarrow_{(P\uparrow n)/w}$$

which can equivalently be formulated as

$$w \hat{} <a> \in trP \iff w \hat{} <a> \in tr(P\uparrow n)$$

To prove the relation above it is enough to show that

$$w \hat{} <a> \in trP \Rightarrow w \hat{} <a> \in tr(P\uparrow n) \qquad \text{(A2)}$$

since the reverse implication is obvious. But (A2) follows by observing that

$$w \hat{} <a> \in trP \Rightarrow v\downarrow_P \hat{} <a> = (v \hat{} <a>)\downarrow_P$$

therefore by definition of s

$$\#(w \hat{} <a>) = \#(s\downarrow_P) \leq n \Rightarrow w \hat{} <a> \in tr(P\uparrow n)$$

This proves the claim (A1) . To prove (3.5) let $s \in tr((P.\hat{P})\uparrow n)$ then there exists $t \in trP$ such that $s=t\downarrow_{\hat{P}}$ and $\#s \leq n$. But by the claim proved above we have $t\downarrow_{\hat{P}} = t\downarrow_{\hat{P}\uparrow n}$ and therefore $s \in tr(P.(\hat{P}\uparrow n))\uparrow n$. This proves that $(P.\hat{P})\uparrow n \leq (P.(\hat{P}\uparrow n))\uparrow n$. Conversely let $s \in tr(P.(\hat{P}\uparrow n))\uparrow n$ then there exists $t \in trP$ such that $s = t\downarrow_{\hat{P}\uparrow n}$. But by definition of projection $s \in tr(\hat{P}\uparrow n)$ and therefore $\#s \leq n$ which implies that $s \in tr((P.\hat{P})\uparrow n)$. We have thus proved that

$$(P.\hat{P})\uparrow n = (P.(\hat{P}\uparrow n)\uparrow n$$

and the rest of (3.5) follows easily. Proofs for (3.3) and (3.4) follows using (3.5).

(3) In order to prove (3.6) let $s \in tr((P\uparrow n).\hat{P})$, then there exists $t \in tr(P\uparrow n)$ such that $\#t \leq n$ and

$$s = t\downarrow_{\hat{P}}$$

Using the same $t \in trP$ we show that $s \in (P.\hat{P})$ and the result follows by observing that

$$\#s = \#(t\downarrow_{\hat{P}}) \leq \#t$$

To show that the projection operator is not *ndes* in P consider the counter-example :

Take

$$trP = \{<>,<b>,ba\}$$
$$tr\hat{P} = \{<>,<a>\}$$

then

$$tr(P.\hat{P})\uparrow 1 = \{<>,<a>\}$$

whereas

$$tr((P\uparrow 1).\hat{P})\uparrow 1 = \{<>\}$$

(4) First consider the following counter-example :

Take P ,Q ,R such that

$$trP = \{<>,<c>,cb\}$$
$$trQ = \{<>,<b>,<a>,ac\}$$
$$trR = \{<>,<b>,<c>\}$$

then we have

$$tr(P.(Q.R)) = \{<>,<c>\}$$

whereas

$$tr((P.Q).R) = \{<>,<b>\}$$

which proves that the operator is not necessarily associative. We prove the rest by using induction on the length of s. Therefore we let $s = t \hat{} <a> \in trP$ and show that

$$t \hat{} <a>\downarrow_Q = t \hat{} <a>\downarrow_{P.Q} \tag{A3}$$

assuming by induction hypothesis that

$$t\downarrow_Q = t\downarrow_{P.Q} \tag{A4}$$

Applying (3.2) and using the induction hypothesis (A4)

$$t \hat{} <a>\downarrow_Q = t\downarrow_Q \hat{} <a>\downarrow_{Q/(t\downarrow_Q)}$$

$$t \hat{} <a>\downarrow_{P.Q} = t\downarrow_Q \hat{} <a>\downarrow_{(P.Q)/(t\downarrow_Q)}$$

Therefore it remains to show that

$$<a>\downarrow_{Q/(t\downarrow_Q)} = <a>\downarrow_{(P.Q)/(t\downarrow_Q)}$$

which is equivalent to showing that

$$<a> \in tr(Q/(t\downarrow_Q)) \iff <a> \in tr((P.Q)/(t\downarrow_Q))$$

and it is enough to show that

$$<a> \in tr(Q/(t\downarrow_Q)) \Rightarrow <a> \in tr((P.Q)/(t\downarrow_Q))$$

where the reverse implication follows from the property $P.Q \lesssim Q$. But since $t\hat{}<a> \in trP$ we have

$$<a> \in tr(P/t) \text{ and } <a> \in tr(Q/(t\downarrow_Q))$$

which implies that

$$<a> \in tr((P/t).(Q/(t\downarrow_Q))) \subseteq tr((P.Q)/(t\downarrow_Q))$$

where we have used (3.7) which is proved next . This proves (A3) . The idempotency condition is an immediate consequence .

(5) By the definition of union operation it is enough to prove the equality of the traces of the processes. Let $v \in tr(P.\hat{P})/s$ then

$$s \hat{} v \in tr(P.\hat{P})$$

and by definition there exists $t \in trP$ such that

$$t\downarrow_{\hat{P}} = s\hat{}v$$

By partitioning t appropriately as $t = t_1 \hat{} t_2$ and using (3.2) we obtain

$$t\downarrow_{\hat{P}} = t_1\downarrow_{\hat{P}} \hat{} t_2\downarrow_{\hat{P}/s}$$

where

$$t_1\downarrow_{\hat{P}} = s \text{ and } t_2\downarrow_{\hat{P}/s} = v$$

therefore

$$v = t_2\downarrow_{\hat{P}/s} \in tr((P/t_1).(\hat{P}/s)) \subseteq \bigcup_{t \in T_s} tr((P/t).(\hat{P}/s))$$

Conversely let

$$v \in \bigcup_{t \in T_s} tr((P/t).(\hat{P}/s))$$

then there exists $t_1 \hat{} t_2 \in trP$ such that

$$t_1\downarrow_{\hat{P}} = s$$
$$t_2 \in tr(P/t_1)$$
$$t_2\downarrow_{\hat{P}/s} = v$$

therefore as before

$$(t_1 \hat{} t_2)\downarrow_{\hat{P}} = s \hat{} v \in tr(P.\hat{P})$$

and

$$v \in (P.\hat{P})/s$$

(6) The proof uses similar approach as in (5) above and is omitted .

Proof of Fact 3.3

(1) (3.10) follows from the definition using induction , (3.11) is a consequence of the property that argument processes of the sum operation cannot block each other.

(2) To prove that the sum operator is associative an *ndes* is also simple and omitted . The following counter-example shows that the sum operator is not necessarily continuous.

Take

$$trw = \{<>,<b>,<a>\}$$
$$trv = \{<>,<b>\}$$
$$tr\hat{w}\{<>,<a>\}$$

so that $\hat{w} \leq w$. But

$$tr(\hat{w} \oplus v) = \{<>,<a>,<b>,ab,ba\}$$
$$tr(w \oplus v) = \{<>,<a>,<b>,ab\}$$

which violates the monotonicity requirement for continuity .

(3) (3.12) follows from (3.11) and the definition of projection. We prove (3.13) by using induction on the length of $s \in tr(u)$ to prove the formula :

$$(s\downarrow_{(w \oplus v)})\downarrow_w = s\downarrow_w \qquad \text{(A5)}$$

Given $s\hat{}<a>$ we evaluate the left side of (A5) using (3.2) twice

$$(s\hat{}<a>\downarrow_{w \oplus v})\downarrow_w = (s\downarrow_{w \oplus v}\hat{}<a>\downarrow_{(w \oplus v)/(s\downarrow_{w \oplus v})})\downarrow_w$$
$$= (s\downarrow_w\hat{}((<a>\downarrow_{(w \oplus v)/(s\downarrow_{w \oplus v})})\downarrow_{w/(s\downarrow_w)}$$

where we already used the induction hypothesis on s given by (A5). It remains to show that

$$((<a>\downarrow_{(w \oplus v)/(s\downarrow_{w \oplus v})})\downarrow_{v/(s\downarrow_v)} = <a>\downarrow_{v/(s\downarrow_v)} \qquad \text{(A6)}$$

We prove (A6) by first claiming that

$$<a> \in tr(v/s\downarrow_v) \Rightarrow <a> \in tr((w \oplus v)/(s\downarrow_{w\oplus v}))$$

After using (3.10) and the induction hypothesis to evaluate

$$(w \oplus v)/(s\downarrow_{w\oplus v}) = w/((s\downarrow_{w\oplus v})\downarrow_w) \oplus v/((s\downarrow_{w\oplus v})\downarrow_v)$$
$$= w/(s\downarrow_w) \oplus v/(s\downarrow_v)$$

the claim follows by definition of the sum operator. (A6) then follows from the claim and the general observation that

$$(<b>\downarrow_P)\downarrow_Q = <b> \iff <b> \in trP \cap trQ$$

We omit the proof of (3.14).

Proof of Fact 3.4

Proof of (1) is obvious. Proof of (2) makes use of the inductive proof of Proposition 2.1. The details are omitted. The reader is referred to [2] for a proof of Proposition 2.1 .

Proof of Fact 3.5

(1) If $\Pi\uparrow 1$ is solid then the generators of Π are solid subsets in W and the proof of Proposition 2.1 [2] will reveal (again details are omitted here) by induction that this property extends to the entire space Π. Proof in the reverse direction is trivial.

(2) If Y inherits a subset of traces of P together with the marks then it is a process in Π since Π is solid , and therefore it is a subprocess of P and is a member of $[P]$ by definition. This proves that $[P]$ is solid. If Y^n is a chain in $[P]$ then

$$tr(\bigsqcup_n Y^n) = \bigcup tr(Y^n) \subseteq trP$$

defines the limit which proves that $[P]$ is closed. Finally let $s \in trX \cap trY$ and $X, Y \in [P]$, then every trace of X and Y must belong to P and therefore

$$\mu Y(s) = \mu X(s) := \mu P(s)$$

Proof of Fact 3.6

(1) The proof is omitted.

(2) We prove (3.16) for the special case when n=2 . Generalization should be straightforward. We first prove (3.16) with v replacing z_1 and z_2 by using induction on the length of traces. Let

$$s \in trv \quad \text{and} \quad s \in tr(v.v_1 + v.v_2)$$

and assume that the trace equality holds for all traces with length $\leq \#s$. We compute the post-process after s of both sides using (3.7) and (3.15) as below :

$$v/s = v_1/(s\downarrow_{v_1}) + v_2/(s\downarrow_{v_2})$$

whereas

$$(v.v_1 + v.v_2)/s = \bigcup_{t\downarrow_{v_1} = s\downarrow_{v_1}} ((v_1/(s\downarrow_{v_1}) + v_2/(t\downarrow_{v_2})).v_1/(s\downarrow_{v_1})$$
$$+ \bigcup_{t\downarrow_{v_2} = s\downarrow_{v_2}} ((v_1/(t\downarrow_{v_1}) + v_2/(s\downarrow_{v_2})).v_2/(s\downarrow_{v_2})$$

The proof is complete if we show that for the processes given by the above two equations a transition $<a>$ belongs to one if and only if it belongs to the other. This is straightforward and we omit the de-

$$v.v_j \leq z_j.v_j \leq v_j$$

for j=1,2 , which is valid by the continuity of the internal sum operator.

To prove (3.17) we proceed by induction similarly and compute

$$((w + v).V)/s = \bigcup_{t\downarrow_V = s} (w/(t\downarrow_W) + v/s).V/s$$

and therefore if

$$<a> \in tr(\bigcup_{t\downarrow_V = s} (w/(t\downarrow_W) + v/s).V/s)$$

then $<a> \in tr(v/s)$ which completes the induction argument.

Finally for (3.18) we compute the post-processes as below

$$u.(W \oplus V)/s = \bigcup_{t\downarrow_{W\oplus V} = s} (u/t).(W/(s\downarrow_W) \oplus V/(s\downarrow_V))$$

$$(u.W + u.V)/s = \bigcup_{t\downarrow_W = s\downarrow_W} (u/t).(W/s\downarrow_W) + \bigcup_{t\downarrow_V = s\downarrow_V} (u/t).(V/s\downarrow_V)$$

and prove the result by demonstrating that if a transition $<a>$ belongs to the first process above then it belongs to the second one. We omit the details.

The proof of (3) is simple and omitted.

Proof of Fact 4.1

(1) Let $s\hat{}t \in trS(u)$ then using (3.2)

$$(s\hat{}t)\downarrow_U := s\downarrow_U \hat{} t\downarrow_{U/(s\downarrow_U)} \in tr(u) \qquad \text{(A7)}$$

by definition of $S(u)$. We must show that $s \in trS(u)$ or

$$s\downarrow_U \in tr(u)$$

which follows from (A7) by noting that it is the prefix of a trace of the process u . To prove (4.3) let $t \in tr(u)$ then since $u \in [X.U]$ there exists $s \in trX$ such that $s\downarrow_U = t$ by definition of $[X.U]$. But this implies that $s \in S(u)$ by definition and therefore $t \in tr(S(u).U)$. This proves that $u \leq S(u).U$. The reverse inequality is proved similarly.

(2) Proof is straightforward and omitted.

(3) Let $S(u) = S(v)$ then $S(u).U = S(v).U = u = v$ by (4.3). Continuity of S follows from (4.4) using the continuity of the projection and internal sum operators and the rule of composition. S is *ndes* in u since no transitions in u can be made invisible in X by (4.4) . Continuity of the response function follows from the continuity of the state and projection functions and the composition rule.

The proofs of Fact 4.2 and Fact 4.3 are omitted.

REFERENCES

[1] K. Inan , P. Varaiya , " Finitely Recursive Process Models for Discrete Event Systems " , *IEEE Trans. Automat. Contr.* , vol.AC-33, No.7, pp. 626-639 , July 1988.

[2] K. Inan , P. Varaiya , "Algebras of Discrete Event Models " , *IEEE Proc.* vol.77 , No.1 , pp. 24-38 , Jan. 1989 .

[3] C.A.R. Hoare , *Communicating Sequential Processes* , Herts, England: Prentice-Hall International, 1985 .

[4] R. Milner , *Calculus of Communicating Systems* , New York NY : Springer , 1980 .

[5] Ramadge P., Wonham M., " Supervisory Control of a Class of Discrete-Event Processes " , *SIAM J. Contr. Optimization* , vol.25 ,no.1, pp. 206-230 , Jan. 1987 .

[6] ANSI/IEEE Std. 802.2-1 85 , approved Dec. 1984.

[7] A. Benveniste , P. Le Guernic , "Hybrid Dynamical Systems Theory and Nonlinear Dynamical Systems over Finite Fields ", *Proc. 27th CDC* , pp. 209-213 , Austin , Texas , Dec. 1988.

[8] M. Tuttle , *Hierarchical Correctness Proofs for Distributed Algorithms* , MSc. Thesis , MIT ,Apr. 1987 .

[9] Willems J., "From Time Series to Linear Systems - Part I. Finite Dimensional Linear Time Invariant Systems" , *Automatica*, vol. 22 , No.5 pp. 561-580 .

[10] J.L. Peterson, *Petri Net Theory and the Modelling of Systems*, Englewood Cliffs, NJ : Prentice Hall, 1981 .

[11] J.S. Ostroff , "Real Time Computer Control of Discrete Event Systems Modelled by Extended State Machines : A Temporal Logic Approach", Rep. 8618, Dept. Electrical Engineering, University of Toronto, Sept. 1986.

[12] Xi-Ren Cao , " The Predictability of Discrete Event Systems " , *Proc. 27th Conference on Decision and Control* , Austin , Dec. 1988 , pp. 198-202 .

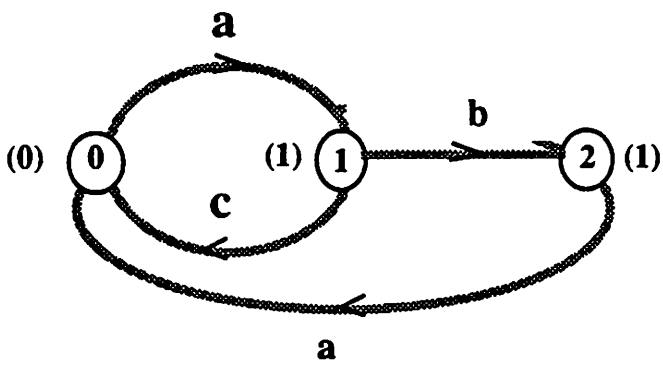
Figure Captions

Figure 3.1 Sum and projection of processes : $P \oplus Q$, $P.Q$, $Q.P$, and $p+q$

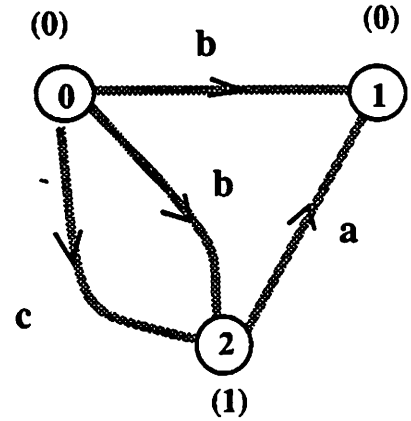
Figure 4.1 Flow controlled Petri-Net example of ADS

Figure 4.2 State and output signals for Example 4.2

Figure 4.3 Alternative switched Petri-Net model for Example 4.2

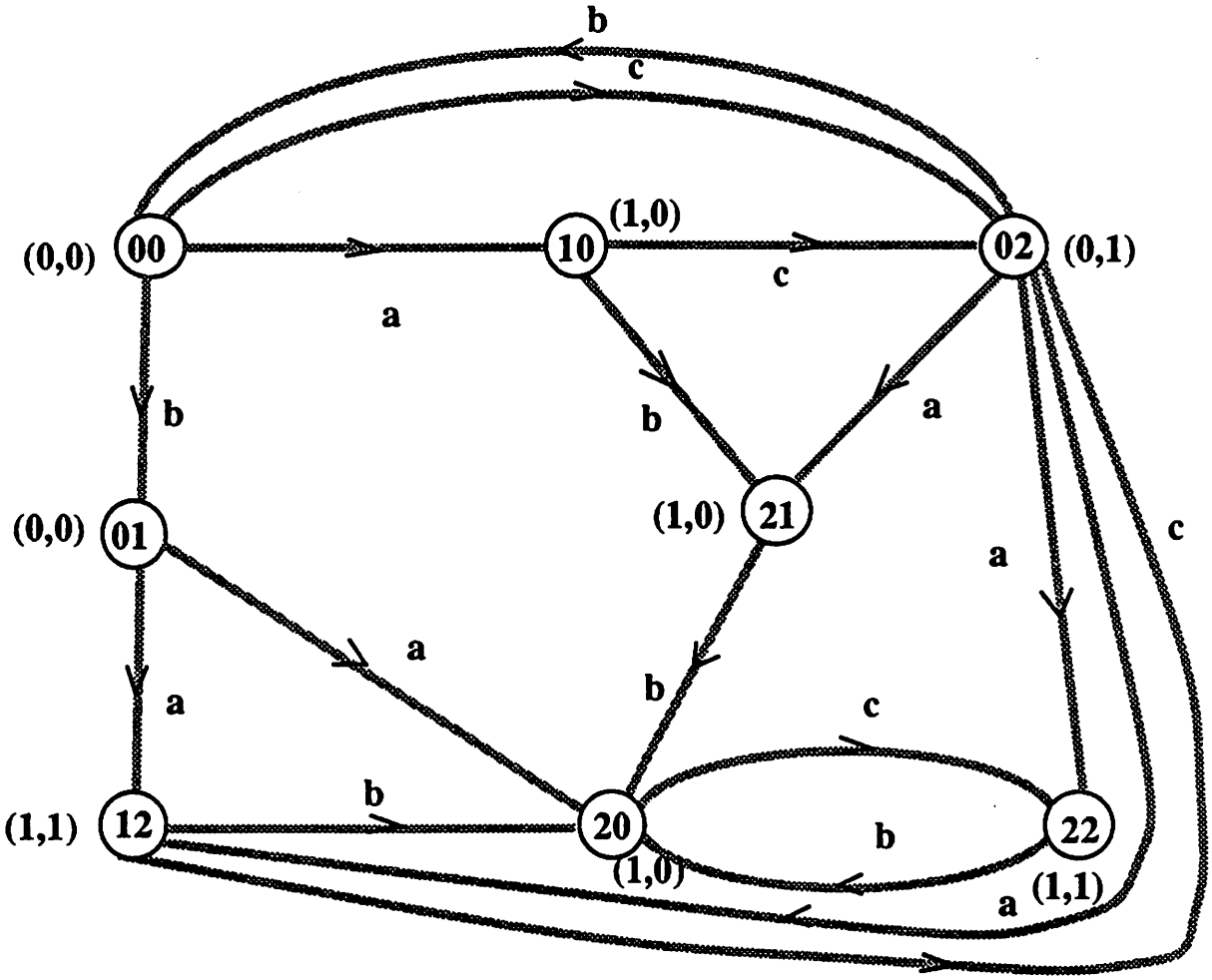


Process P



Process Q

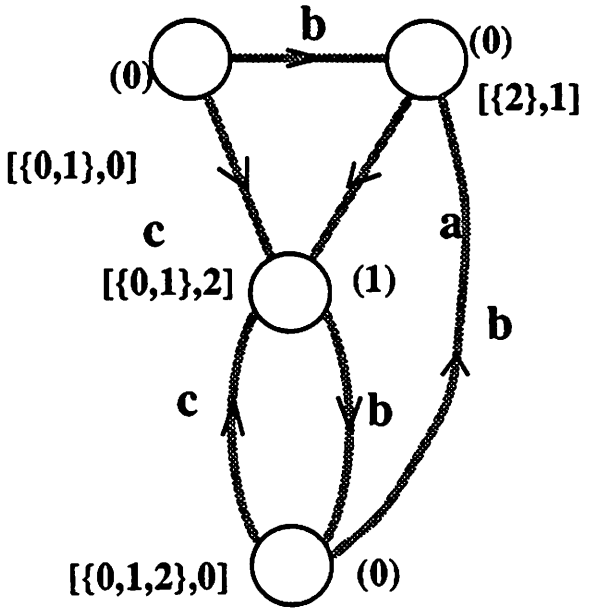
(a)



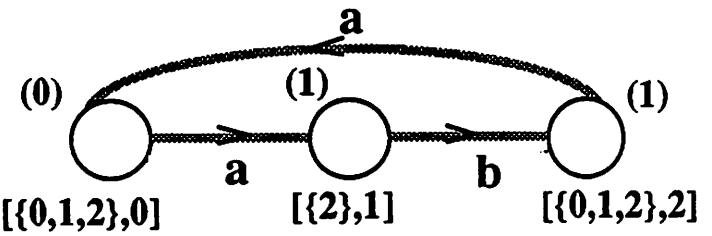
Process P⊕Q

(b)

Figure 3.1 (a) & (b)

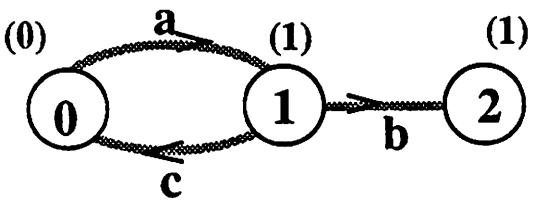


Process P.Q

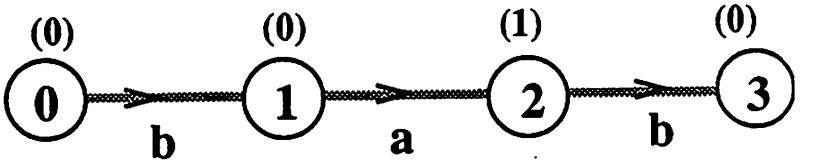


Process Q.P

(c)

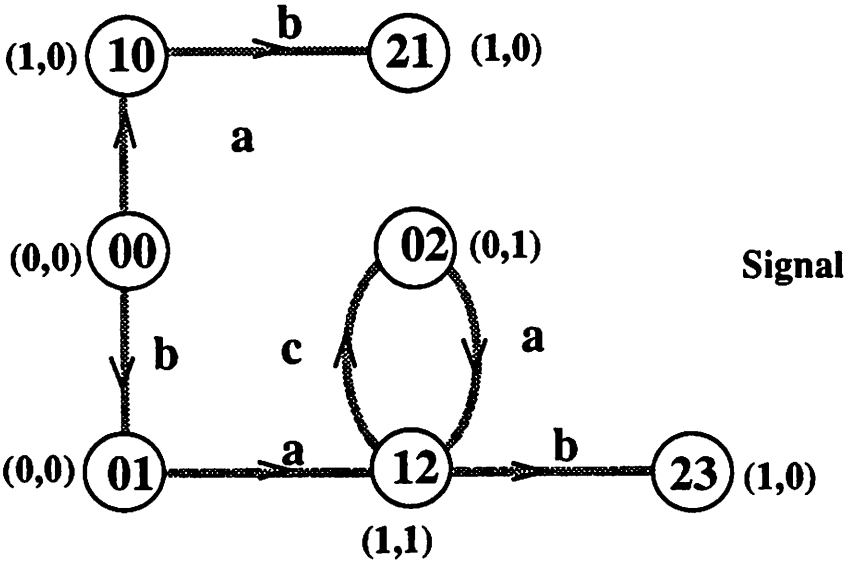


Signal p ε [P]



Signal q ε [Q]

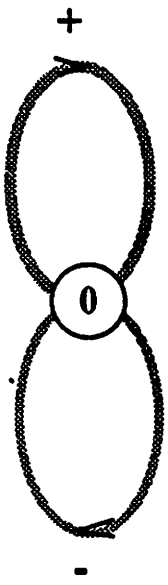
(d)



Signal p+q ε [P + Q]

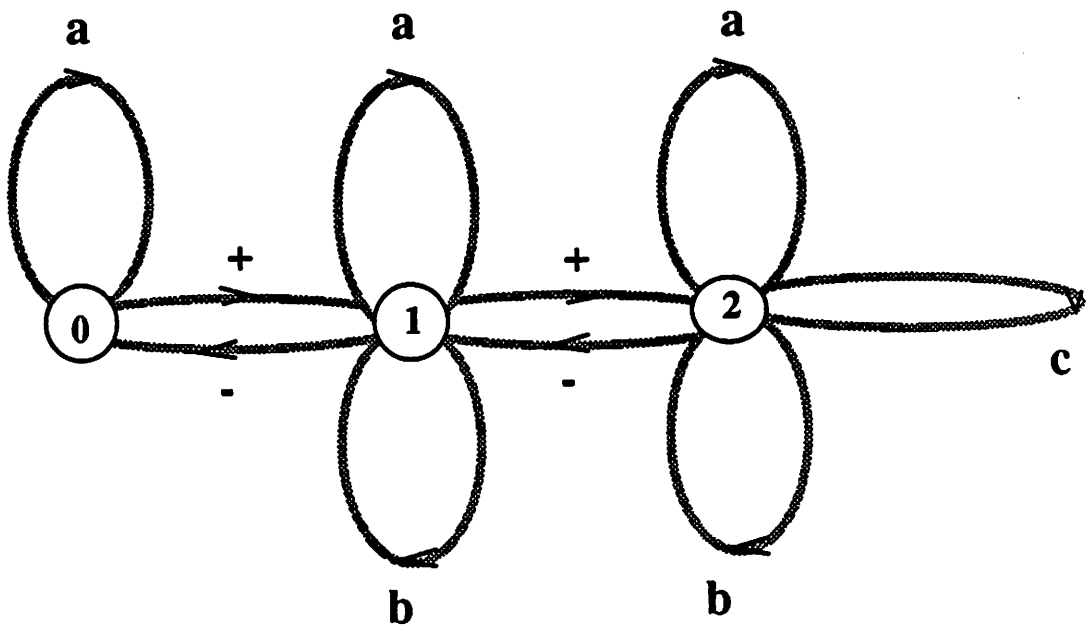
(e)

Figure 3.1 (c) (d) & (e)



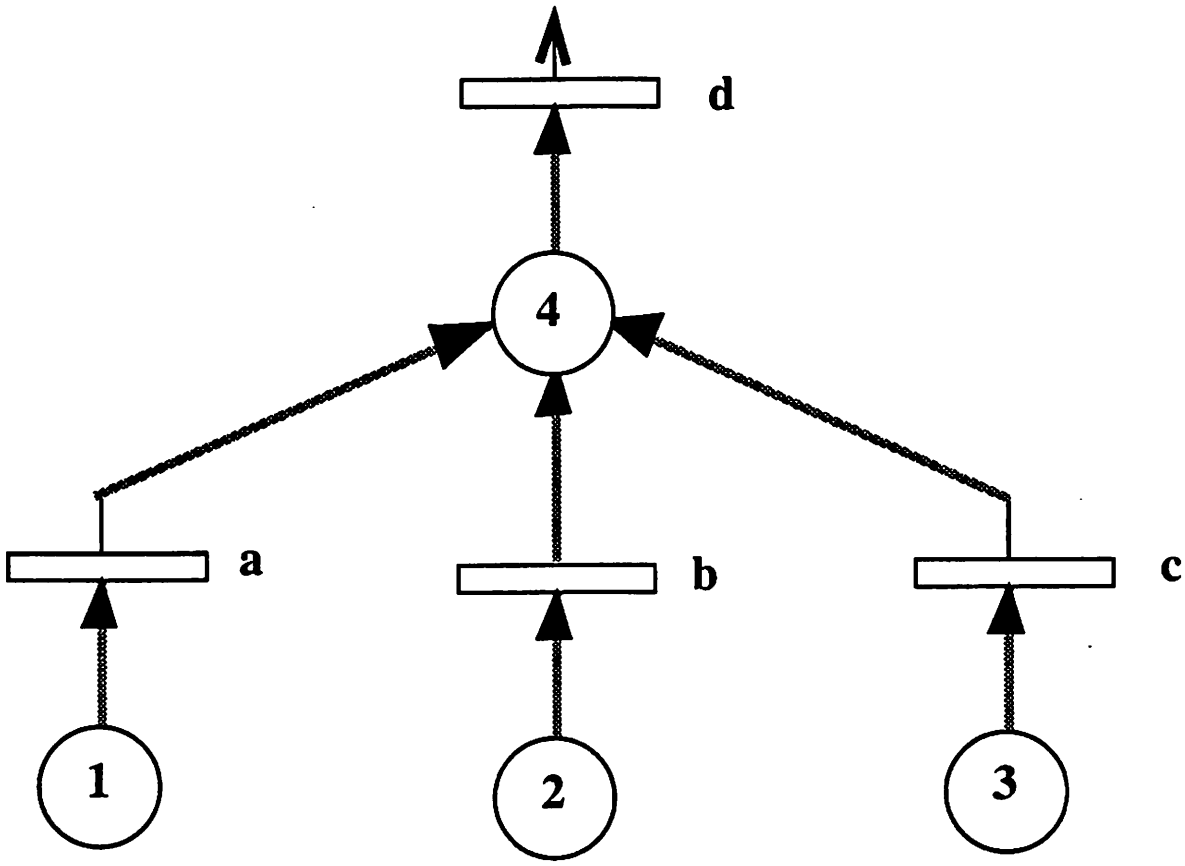
Input U

(a)



State X

(b)



Output Y

(c)

Figure 4.1

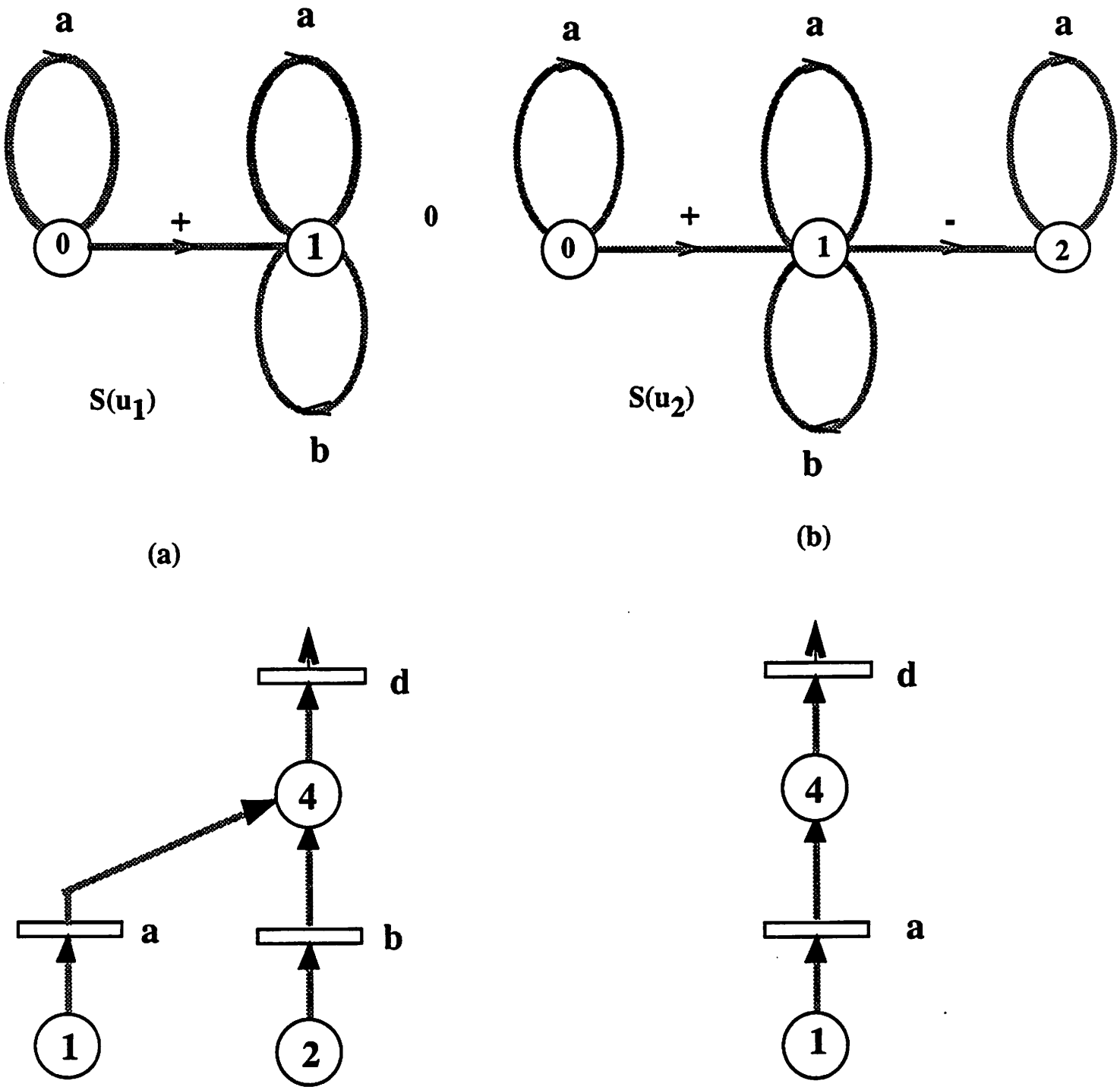
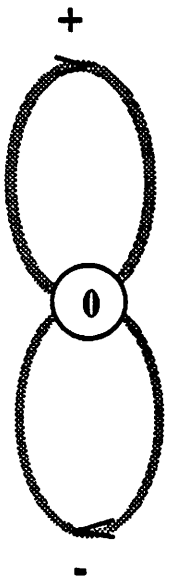
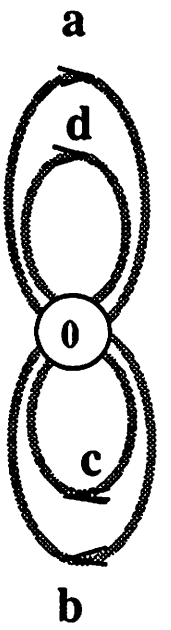


Figure 4.2



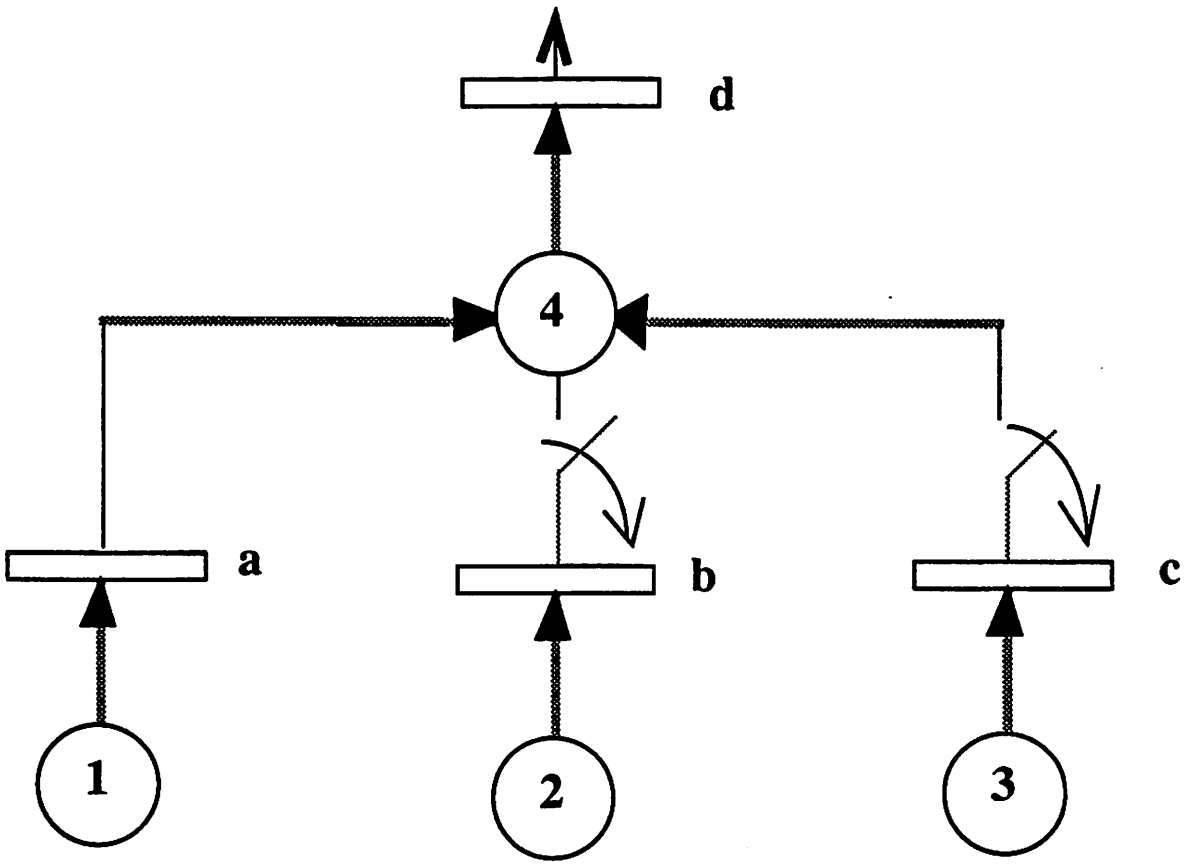
Input U

(a)



Output Y

(b)



State X

(c)

Figure 4.3