MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 123

A New Approach to Petri Nets

by

Frederick Furtek

(This material originally appeared in the author's doctoral thesis proposal, submitted January 29, 1975, and entitled "The Logic of Systems".)

This work was supported in part by the National Science Foundation under research grant DCR 74-21822.

April 1975

Table of Contents

1. System Models

The term system is taken to mean an assemblage of interacting elements. But that is not enough. Because the behavior of a system is always, to some extent, 'systematic', or 'organized', there must exist rules that govern, at least partially, the interactions among elements. In many cases, these rules are only implicitly understood, but when made explicit, they form a model[†] of the system. Some of the more common types of models are:

(a) a set of differential equations (where the interacting elements are 'infinitesimal')

(b) a collection of sequential circuits

(c) a set of difference equations relating system levels and rates of flow (the models of 'System Dynamics')

(d) a description in plain English

Each of these has a domain to which it is particularly suited. The type of model we'll be working with is especially appropriate for those systems in which information transmission and transformation are the chief interests.

[†] 'Model' is used in two slightly different senses. It can refer to a detailed set of rules applicable to a particular system, or to a general set of rules applicable to a whole range of systems. A set of differential equations relating the electric and magnetic field intensity in a specific region of space is an example of the former sense, while Maxwell's equations are an example of the latter. Where the distinction is important, context will make clear the usage intended.

2. Actors and Parts[†]

A very natural way of modelling a system is to view the system elements as actors playing parts.

The term 'part' is here used in the sense of a role, or pattern of activity, that someone or something follows. Delivery boy, secretary, aviator, policeman, 'Hamlet', are examples of roles played by humans, while clock, power source, transmission line are examples of roles usually played by mechanical devices.

An actor is anybody or anything that is seen to play a part - 'participant' is another term that would serve equally as well. Under appropriate circumstances, a parson, a chair, a flip-flop, a displacement, or a voltage level might each be considered an actor.

Now, among all the properties an actor may be thought to have, only certain ones will be pertinent in a given system context. It is the part[††] an actor plays that abstracts just those properties that are deemed relevant to system operation. For this reason, 'part' will be an important concept in what follows.

In our theory, there is just one requirement a pattern of activity must satisfy for it to define a part: it must be totally sequential. That is, a part must require an actor to do at most 'only one thing at a time'. This is because each part of a system will be identified with a 'state machine'. State machines are ideally suited for describing sequential behavior, but, for our purposes, they are completely unsuited for describing nonsequential behavior.

[†] The ideas in this section are due to Anatol Holt.

[††] It may happen that an actor plays several parts simultaneously or switches between parts. Although questions relating to actor identity are important, they are, unfortunately, outside the scope of this work.

3. State Machines

A state machine $\mathcal{M}$ is a quadruple $\langle S,E,F,s_0 \rangle$ where,

S is a finite set of states
E is a finite set of events
$F \subseteq S \times E \times S$ is the flow relation
$s_0 \in S$ is the initial state

Furthermore, each event appears in exactly one triple of F. Stated in logical notation,

$$\forall e \in E: \quad |F \cap (S \times \{e\} \times S)| \; = 1$$

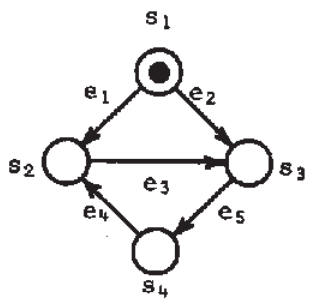
If $\langle s_1,e,s_2 \rangle \in F$, then s_1 is said to be a precondition of e and s_2 a postcondition of e. The preceding requirement means that within a single state machine each event has a unique precondition and a unique postcondition.

In the graphical representation of a state machine, each state is depicted as a circle and each event as an arc directed from the event's precondition to its postcondition. The initial state is designated by placing a 'token' on the appropriate state.

Example:

$$\mathcal{M} = \langle \{s_1,s_2,s_3,s_4\}, \; \{e_1,e_2,e_3,e_4,e_5\} \; \{\langle s_1,e_1,s_2 \rangle, \langle s_1,e_2,s_3 \rangle, \langle s_2,e_3,s_3 \rangle, \langle s_4,e_4,s_2 \rangle, \langle s_3,e_5,s_4 \rangle\}, \; s_1 \rangle$$

(a) formal representation



(b) graphical representation

Figure 1. A State Machine

A process in a state machine is any finite path originating at the initial state. Thus, $s_1e_2s_3e_5s_4$ and $s_1e_1s_2e_3s_3$ are two of the infinitely many processes in our example. In a process, each instance of a state is called a holding of that state, and each instance of an event an occurrence of that event. A holding is said to be initiated by the immediately preceding occurrence, if any, and terminated by the immediately following occurrence, if any.

From these definitions we see that:

(1) The initial state 'holds' initially.

(2) An event may 'occur' when its precondition holds.

(3) The occurrence of an event terminates a holding of the event's precondition and initiates a holding of its postcondition.

(4) A holding is terminated by no more than one occurrence.

(5) Except as provided for in (1), holdings are initiated and terminated only by occurrences.

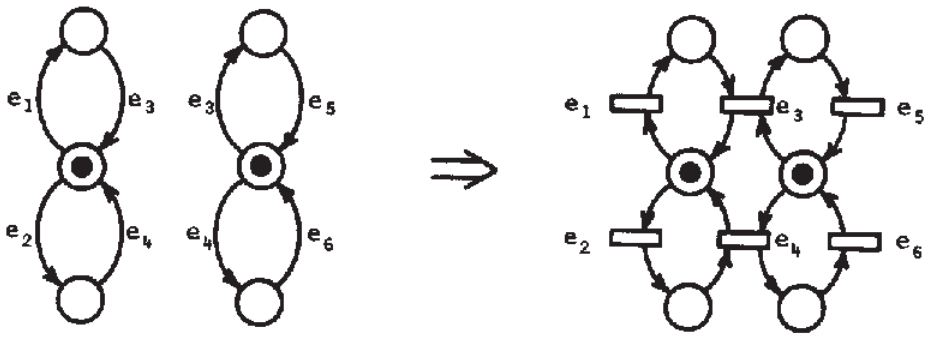
These five rules completely characterize the way in which a state machine defines a pattern of behavior. Notice that when several events have a common precondition, a holding of that precondition may be terminated by an occurrence of any one of those events. This is how the notion of alternativeness is represented.

4. Interconnection of State Machines

If I hand you a letter, then the two of us are involved in one and the same event (although it will be viewed in two different ways: by me as a giving and by you as a taking). This motivates the following.

Interaction between parts is accommodated through the 'synchronization' of events belonging to different state machines, which amounts to nothing more than allowing state machines to have events in common. To express this added relationship, a new type of structure is introduced: a network of interconnected state machines.

In the graphical representation of a network, events are drawn as rectangles so that the sharing of events can be explicitly shown. Thus, instead of having the same event appear as several arcs, as in Figure 2(a), it appears as a single rectangle, as in Figure 2(b).



(a) (b)

Figure 2. Representing Shared Events

The 'simulation rule' for networks (defined formally in Section 8) is completely natural. It simply incorporates the rules of the individual state machines without adding any extra constraints. In other words, the new rule is just the conjunction of the rules for the component state-machines:

(1) Each initial state 'holds' initially.

(2) An event may 'occur' when all of its preconditions hold.

(3) The occurrence of an event terminates a holding of each precondition and initiates a holding of each postcondition.

(4) A holding is terminated by no more than one occurrence.

(5) Except as provided for in (1), holdings are initiated and terminated only by occurrences.

From these rules, we see that states are 'disjunctive' while events are 'conjunctive'.

As it turns out, 'networks' are a subclass of a more general structure, the elements of which are known as Petri nets. The members of the subclass are, naturally enough, known as state-machine-decomposable (SMD) Petri nets. The simulation rule just described holds not only for SMD nets but for general nets as well.

5. Petri Nets

The story of Petri nets begins with the work of Carl Adam Petri who published the seminal paper Kommunikation mit Automaten in 1962. The model introduced there was later refined by Anatol Holt[9,10], and the amended structures were given the name 'Petri nets'. With Holt's work there has been a steadily increasing interest in nets, the key attraction being an ability to represent both concurrency and nondeterminacy. SMD nets have long been recognized as an important subclass of Petri nets, but because of the difficulties involved, there have not yet been any significant results for this subclass. The results that have been obtained are for much more severely restricted classes of nets: state machines [12],

marked graphs [1,4,7,11,13], free-choice nets [3,8], simple nets [3,14], and determinate† nets [17]. It is with this background that we approach the study of SMD nets.

We present now the formal definition of a Petri net and definitions of those subclasses that are characterized in terms of structure. (The subclasses characterized in terms of behavior will be defined after the simulation rule has been given.)

A Petri net is a quadruple $<S,E,F,I>$, where,

S is a finite set of states
E is a finite set of events
$F \subseteq (S \times E) \cup (E \times S)$ is the flow relation
$I \subseteq S$ is the set of initial states

Nets are represented graphically as in Section 4, the arcs corresponding to the ordered pairs in F. We write $x \cdot y$ to mean $\langle x,y \rangle \in F$. For $x \in S \cup E$,

$$^{\bullet}x = \{y | y \cdot x\}$$
$$x^{\bullet} = \{y | x \cdot y\}$$

We refer to $^{\bullet}x$ as the backdot of x, and to $x^{\bullet}$ as the foredot of x. If x is a state, the elements of $^{\bullet}x$ are called the input events of x, and the members of $x^{\bullet}$ the output events of x. If x is an event, the elements of $^{\bullet}x$ are called the input states, or preconditions, of x, and the members of $x^{\bullet}$ the output states, or postconditions, of x. For $X \subseteq S \cup E$,

$$^{\bullet}X = \{y | \exists x \in X: \ y \cdot x\}$$
$$X^{\bullet} = \{y | \exists x \in X: \ x \cdot y\}$$

The preceding terminology carries over.

---

†Determinate nets are also known as persistent nets and conflict-free nets.

State machines,[†] marked graphs, free-choice nets, and simple nets are all defined on Table 1. For marked graphs there is an abbreviated representation comparable to that given for state machines in Section 3. Each event is drawn as a point, and each state as an arc directed from the state's (unique) input event to its (unique) output event. Example:

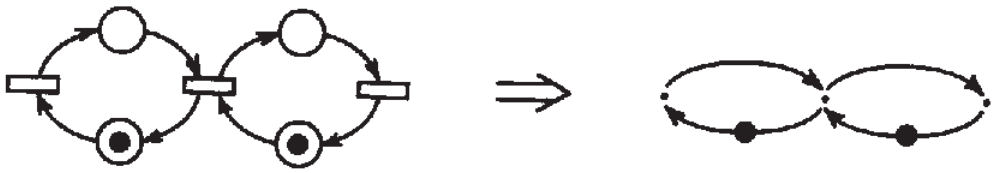


Figure 3. Abbreviated Representation for Marked Graphs

A <u>state-machine decomposable Petri net</u> is a net in which,

$\exists D \subseteq \mathcal{P}(S)$ such that:

(1) $$\cup D = S$$
and
(2) $$S^{\cdot} = E$$
and
(3) $$(\forall \delta \in D)(\forall s_1, s_2 \in \delta)(s_1 \neq s_2 \Rightarrow (s_1{}^{\cdot} \cap s_2{}^{\cdot} = \phi \wedge {}^{\cdot}s_1 \cap {}^{\cdot}s_2 = \phi))$$
and
(4) $$(\forall \delta \in D)({}^{\cdot}\delta = \delta^{\cdot})$$
and
(5) $$(\forall \delta \in D)(|\delta \cap I| = 1)$$

Each subset δ defines a state machine as follows:

$$M_\delta = \langle S_\delta, E_\delta, F_\delta, I_\delta \rangle = \langle \delta, \delta^{\cdot}, F \cap (\delta \times E \cup E \times \delta), \delta \cap I \rangle$$

Conditions (1) and (2) require that every state and every event belong to a state machine. Conditions (3) and (4) insure that each state machine statisfies the 'one-in-one-out' property for events. Condition (5) says that each state machine has exactly one initial state. Notice that D doesn't necessarily

[†]The definition of a state machine given here is slightly different from the one given earlier.

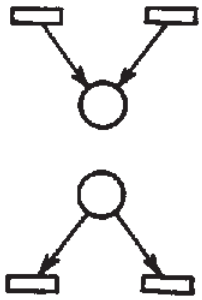
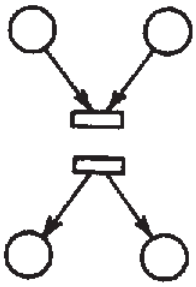
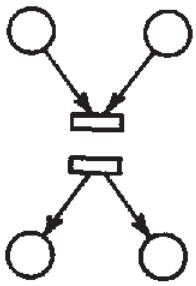
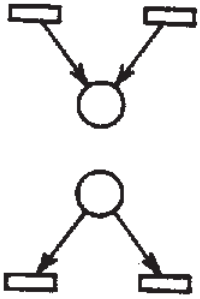
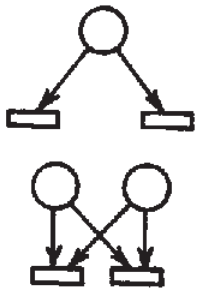
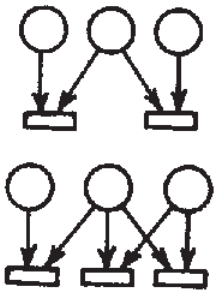
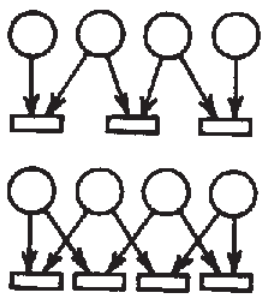
Definition of Subclass	Local Configuration: yes	Local Configuration: no
state machines - every event has exactly one precondition and one postcondition: $\forall e \in E: \lvert {}^\bullet e \rvert = \lvert e^\bullet \rvert = 1$		
marked graphs - every state has exactly one input event and one output event: $\forall s \in S: \lvert {}^\bullet s \rvert = \lvert s^\bullet \rvert = 1$		
free-choice nets - if the foredots of two states are not mutually exclusive, then they are equal: $\forall s_1, s_2 \in S: s_1{}^\bullet \cap s_2{}^\bullet \neq \phi \Rightarrow s_1{}^\bullet = s_2{}^\bullet$		
simple nets - if the foredots of two states are not mutually exclusive, then one foredot contains the other: $\forall s_1, s_2 \in S:$ $s_1{}^\bullet \cap s_2{}^\bullet \neq \phi \Rightarrow s_1{}^\bullet \subseteq s_2{}^\bullet \vee s_2{}^\bullet \subseteq s_1{}^\bullet$		

Table 1. Structural Subclasses of Petri Nets

partition S, thus allowing state machines to share states as well as events. This is done only to keep the nets as simple as possible. Shared states can always be duplicated, thereby permitting different state machines to have mutually-exclusive state sets. (A state-machine partitionable net results.) The technique is illustrated in Figure 4.

Figure 5 shows the inclusion relations among most of the sublclasses.

6. A Petri-Net Example

The net in Figure 6 demonstrates the ability of Petri nets to model computational devices. The net, which is SMD, is a model of a 'half adder'.†

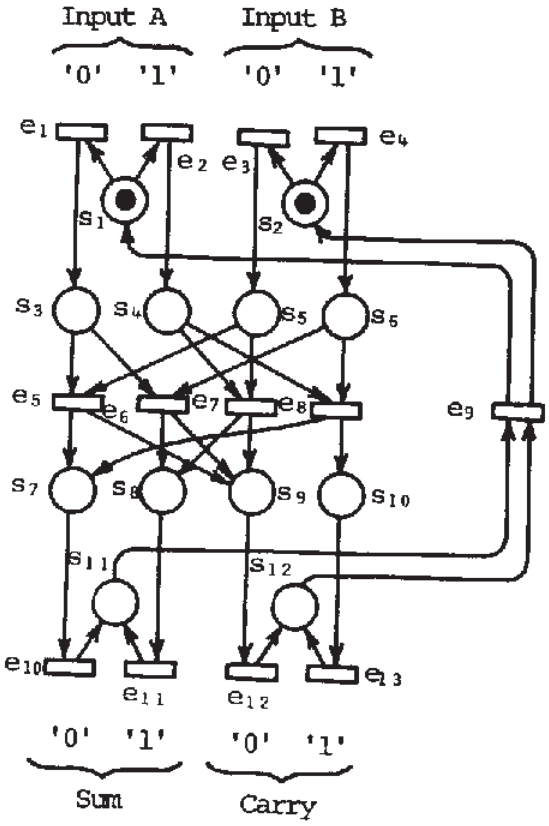


Figure 6. A 'Half Adder'

†More complex examples can be found in [6].

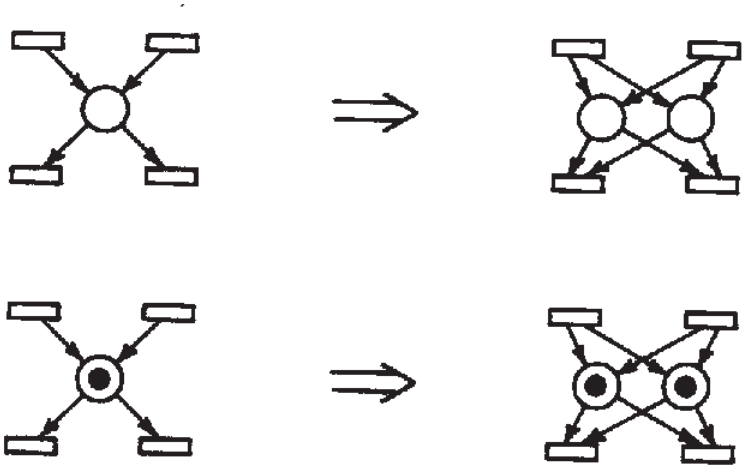


Figure 4. Duplicating a State

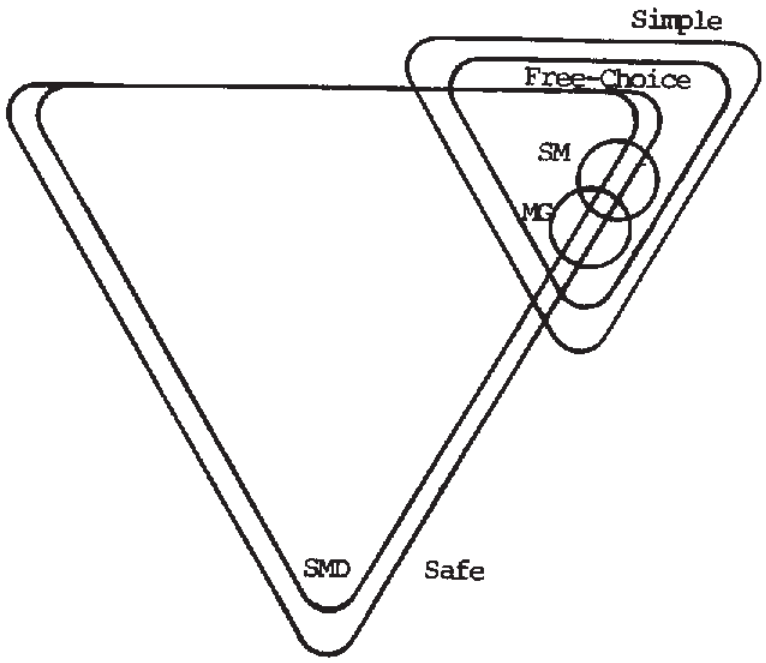


Figure 5. Inclusion Relations among Subclasses

The two 'free-choice' structures consisting of arcs $\langle s_1,e_1\rangle$ and $\langle s_1,e_2\rangle$ and of arcs $\langle s_2,e_3\rangle$ and $\langle s_2,e_4\rangle$ are the means by which 'information' enters the system from the environment. It is 'indeterminate' whether e_1 or e_2 will terminate a holding of s_1, and, likewise, it is 'indeterminate' whether e_3 or e_4 will terminate a holding of s_2. 'Information' supplied by the environment resolves this indeterminacy. For example, an occurrence of e_1 means that a '0' has been received on Input A, and an occurrence of e_4 means that '1' has been received on Input B.

Occurrences of e_1 and e_4 lead to holdings of s_3 and s_6. This allows e_6 to occur, thereby producing holdings of s_8 and s_9. We now reach two situations in which 'information' is lost by the system - and presumably acquired by the environment. After e_{11} occurs and initiates a holding of s_{11}, the system no longer 'remembers' - i.e., it 'forgets' - whether e_{10} or e_{11} occurred. The 'information' transferred to the environment is the 'sum' of the last two inputs - in this case, '1'. An occurrence of e_{12} followed by a holding of s_{12} represents a 'carry' of '0'.

With an occurrence of e_9, states s_1 and s_2 once again hold, and the system is ready to receive another pair of inputs.

7. The Ordering of Occurrences

The simulation rule is the means by which a Petri net defines a pattern, or patterns, of behavior. It is analogous to the definition of a 'process' for state machines. But whereas a state machine requires any two event occurrences (in a process) to be ordered, that is not the case for nets. And therein lies a problem.

It is my feeling that a theory based on Petri nets will be truly productive only if the simulation rule meets a crucial requirement:

> For any Petri net, two event occurrences are not to be ordered, either explicitly or implicitly, unless so required by the structure of the net.

As simple and natural as this requirement is, there has not been, to my knowledge, a definition of the simulation rule that satisfies it.

What has been customary is to express the rule in terms of transformations between 'markings'. This, however, leads directly to the notion of a 'firing sequence', which is a total ordering of all occurrences. (Commutativity of occurrences - a follows b or b follows a - is not the same as being unordered.)

Occurrence graphs (o-graphs) [9] are a step in the right direction since they are a way of expressing precisely the partial order that exists among holdings and occurrences. The trouble is that no one has shown how to suitably generate the o-graphs of a net. The obvious approach (the one given in [9]) is an inductive definition in which there is one initial o-graph, determined by the initial marking, and in which a new o-graph is constructed from an old one by adding one more event occurrence. Implicit in this definition is the fact that each o-graph is generated sequentially, and thus the existence of an o-graph presupposes the existence of a firing sequence.

In the next section we give a definition of the simulation rule that I believe satisfies completely the criterion stated above. The representations of behavior are called 'traces', and they are equivalent to a special type of occurrence graph. What makes my approach novel is the manner in which

traces are generated. Instead of using just one previously-assembled trace to construct a new one, multiple traces will, in general, be used.

8. The Simulation Rule

The simulation rule is embodied in the definition of a 'trace'. (This parallels the definition of a 'process' for state machines.) Before showing how the traces of a net are formally generated, we describe their structure and introduce some terminology.

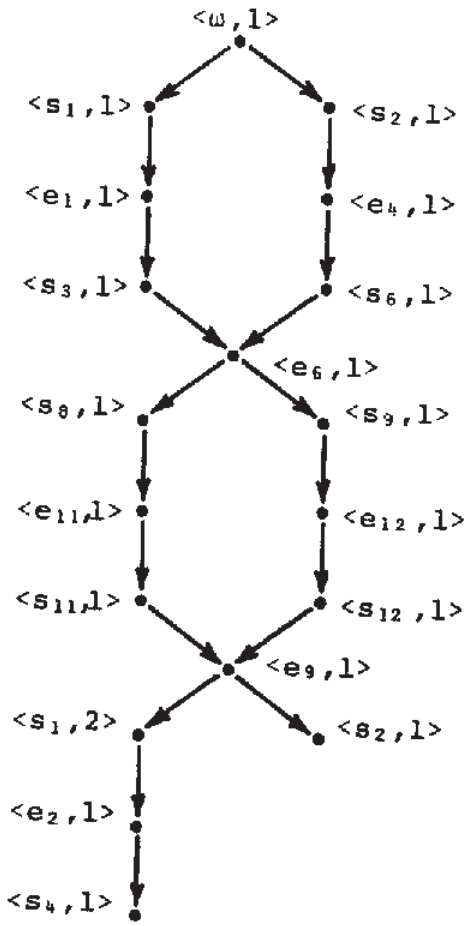
Each trace of a Petri net $\langle S,E,F,I\rangle$ is a binary relation viewed as a directed graph.† The vertices, called instances, are of the form $\langle x,n\rangle$, where x, the instance type, is either a state or an event, and n, the instance number, is a natural number. By a 'state' we mean simply a member of S, but by an 'event' we mean a member of E', where E' is E with the special symbol ω added. That is, $E' = E \cup \{\omega\}$. An instance $\langle x,n\rangle$ is a holding - the n'th holding of x - if $x \in S$, and an occurrence - the n'th occurrence of x - if $x \in E'$. The special event ω appears only in arcs of the form $\langle\langle\omega,1\rangle, \langle s,1\rangle\rangle$, where $s \in I$. This is to be interpreted as saying that what precedes the first holding of s, if anything, is unknown.

In Figure 7 is one of the traces for the net in Figure 6. It exhibits six properties common to all traces:

---

† The source of our terminology relating to graphs is Berge, Graphs and Hypergraphs.

T:



holdings of T = {<s_1,1>,<s_2,1>,<s_3,1>,<s_6,1>,<s_8,1>,<s_9,1>,<s_{11},1>,<s_{12},1>,
<s_1,2>,<s_2,1>,<s_4,1>}

occurrences of T = {<ω,1>,<e_1,1>,<e_4,1>,<e_6,1>,<e_{11},1>,<e_{12},1>,<e_9,1>,<e_2,1>}

boundary of T = {<s_2,1>,<s_4,1>}

focus of T = <e_2,1>

Figure 7. A Trace of the Net in Figure 6

(1) The trace is bipartite with holdings and occurrences forming the partition.

(2) $\langle\omega,1\rangle$ is the only occurrence of ω and is a root of the trace.

(3) The out-degree of a holding is at most 1. (The holdings whose out-degrees are 0 comprise the 'boundary' of the trace.)

(4) There are no circuits. (But there may be cycles.)

(5) All instances of a given type lie on an elementary path, the instance numbers appearing consecutively beginning with 1.

(6) There exists a unique occurrence, (the 'focus') that is reachable from all other occurrences.

Property (1) means that each arc of a trace is either of the form <occurrence, holding> or of the form <holding, occurrence> . In the first case, we say that the occurrence initiates the holding, and in the second, that it terminates the holding. Because of Property (2), an initiating occurrence is specified for each holding.† A terminating occurrence, however, is not necessarily specified, but when it is, it must be unique because of Property(3).

We now define a special relationship among traces. It will be used in the simulation rule and will contribute to producing Property (3). Traces $T_1,T_2,\cdots,T_n$ are said to be compatible if and only if their union contains no more than one terminating occurrence for each holding. In other words, if holding h is terminated by occurrence x in one trace and by occurrence y in another, then x and y are the same.

†The question of whether a holding can be initiated by more than one occurrence is discussed in the next section.

In formal terms,

$T_1, T_2, \cdots, T_n$ are compatible $\Leftrightarrow$

$\forall h \in S \times N$: $\langle h,x \rangle \in T_1 \cup T_2 \cup \cdots T_n \wedge \langle h,y \rangle \in T_1 \cup T_2 \cup \cdots \cup T_n \Rightarrow x=y$

In Figure 8, T_1 and T_2 are compatible, as are T_1 and T_3. However, T_2 and T_3 are not compatible. If T is the union of a compatible set of traces, then the boundary of T is the set of those holdings for which no terminating occurrence is given - those holdings whose out-degree is 0.

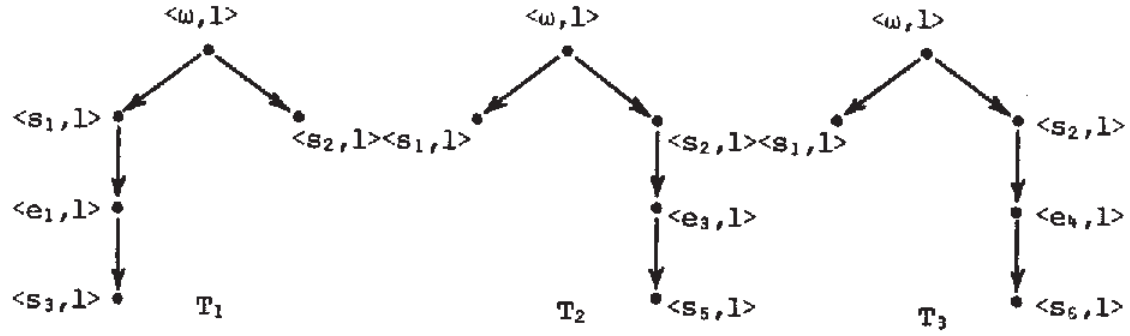


Figure 8. Three Traces of the Net in Figure 6

Consider now the result of taking the transitive closure of a trace and adding a self-loop for each vertex. The new relation is transitive, reflexive, and - because of Property (4) - antisymmetric. In short, it is a partial order. We write $x \leq y$ to indicate that x is related to y by this partial order and $x < y$ to indicate that $x \leq y$ but $x \neq y$. We adopt the following terminology:

$x = y$ -	x and y are coincident	
$x < y$ -	x is before y y is after x x precedes y y follows x	x and y are ordered
$x \not\leq y \wedge y \not\leq x$ -	x and y are concurrent	

A consequence of Property (5) is that any two instances of the same type are ordered. Furthermore, for instances $\langle x,m\rangle$ and $\langle x,n\rangle$ appearing in a trace, $\langle x,m\rangle \leq \langle x,n\rangle$ if and only if $m \leq n$.

Having defined the notion of ordering, we can now describe the special significance of a trace. Property (6) says, in effect, that each trace has a unique occurrence that follows all other occurrences. In other words, a trace describes just that acitivity that precedes an occurrence of some event. Or put a little differently, a trace contains an occurrence and all the 'activity' that leads up to it. The unique occurrence is called the focus of the trace.

We are now ready for the simulation rule.

Definition: Given a Petri net $\eta = \langle S,E,F,I\rangle$, we define the traces of η as follows:

(1) $\{\langle\omega,1\rangle\} \times (I\times\{1\})$ is a trace.

($I\times\{1\}$ is the set of initial holdings.)

(2) If: (a) T is the union of a compatible set of existing traces,

(b) e is an event whose preconditions all have holdings in the boundary of T, and A is that set of holdings,

(c) each occurrence in T has a descendant in A,

then $T \cup A\times\{\mu(e,T)\} \cup \{\mu(e,T)\}\times\mu(e^{\bullet},T)$ is a trace,

where $\mu(q,T) = \langle q, \left|\{y \mid \exists\, n,x:\ y = \langle q,n\rangle \wedge \langle x,y\rangle \in T\}\right| + 1\rangle$

($\mu(e,T)$ is the k+1'st occurrence of e, where k is the number of occurrences of e in T. $\mu(e^{\bullet},T)$ is the set of new holdings initiated by $\mu(e,T)$.)

(3) The only traces of η are those given by (1) and (2).

Figure 9 illustrates how several of the traces for the net in Figure 6 are generated.

We now look at the simulation rule in the light of the five rules given in Section 4 and the requirement of the preceding section. We first consider the simulation rule without Condition 2(c), and then consider the effect of adding that condition.

With Condition 2(c) left out, the simulation rule exactly implements the five rules of Section 4. Nothing is added and nothing taken away. The resulting structures, called simulations, are equivalent to occurrence graphs.

The addition of Condition 2(c) has the effect of requiring each trace to have a 'focus'. It also means that if two occurrences are ordered (as defined above), then the two occurrences are created in that order, and if two occurrences are concurrent, then the acts of creating the two are logically independent and thus unordered. In other words, the requirement of the preceding section is satisfied. Condition 2(c) does not impose any fundamental limitations since any simulation, with or without a 'focus', can be generated, in accordance with our requirement, by taking the union of a compatible set of traces.

9. Safety, Determinacy, and Liveness

In this section, we define some of the concepts used in characterizing the behavior of nets.

A net is said to be safe if and only if no simulation of the net contains a holding initiated by more than one occurrence. The net in Figure 10(a) is an example of an unsafe net. Notice that the first holding of s_3 can be initiated

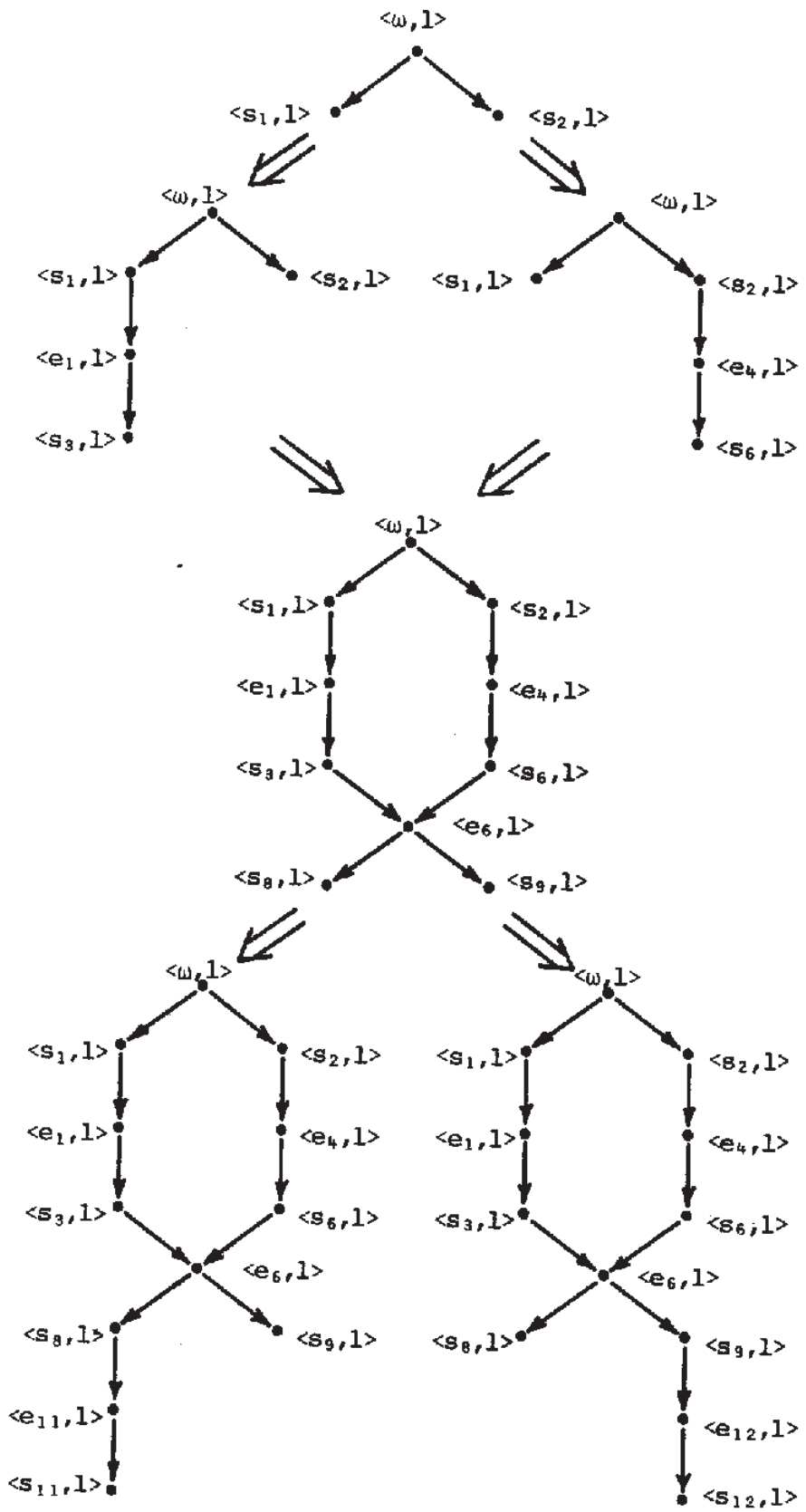
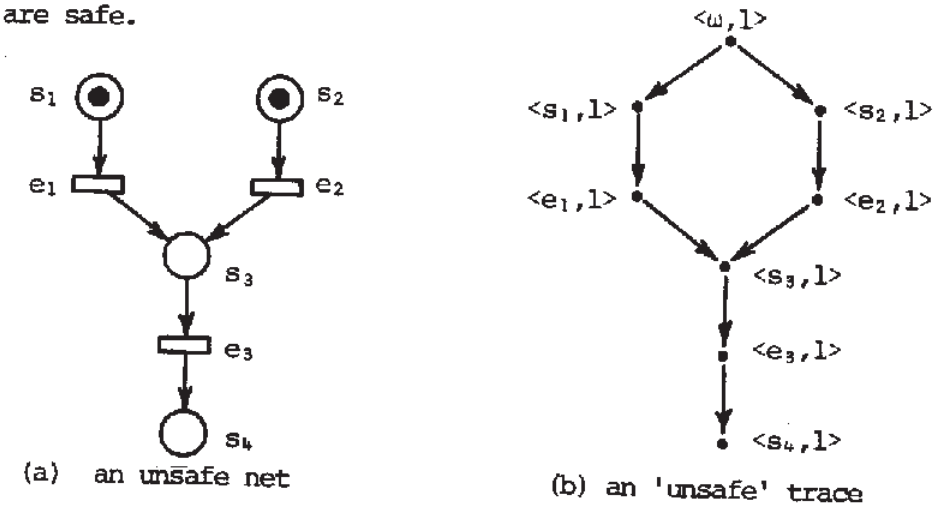


Figure 9. Generating the Traces of the Net in Figure 6.

by both the first occurrence of e_1 and the first occurrence of e_2. This is reflected in the trace of Figure 10(b). For the most part, our work will deal only with safe nets. This is not a serious limitation since all SMD nets are safe.



(a) an unsafe net

(b) an 'unsafe' trace

Figure 10. An Example of Nonsafety

Determinate nets are what their name implies: nets whose behavior does not involve arbitrary choices. More precisely, a net is <u>determinate</u> if and only if all traces associated with the net are compatible. The net in Figure 6 is an example of an indeterminate net, while the net in Figure 11 is an example of a determinate net.

<u>Liveness</u> has to do with the ability of states to hold and events to occur. We distinguish between five degrees of liveness for a state or event, x:†

† Except for the definition of L_3, these definitions are equivalent to the ones given by Commoner [3], p. 8.

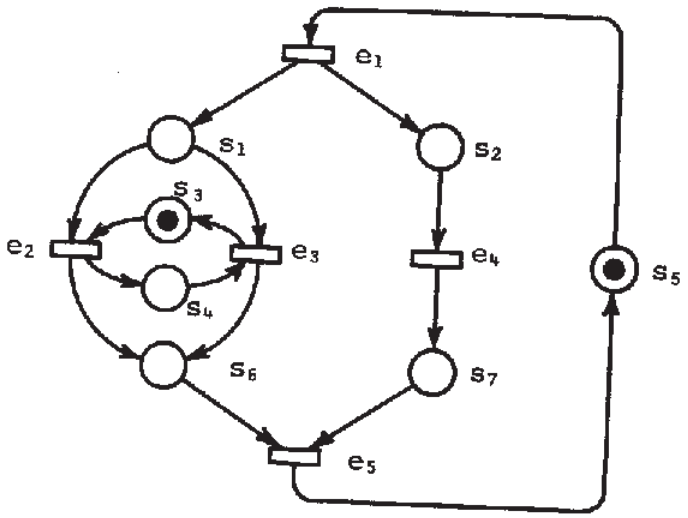


Figure 11. A Determinate Net

L_0 - no trace contains <x,1>.

L_1 - there exists a trace containing <x,1>.

L_2 - for every natural number n, there exists a trace containing <x,n>.

L_3 - there exists a simulation T such that for every simulation T' containing T and for every $n \in N$, there exists a trace compatible with T' and containing <x,n>.

L_4 - for every simulation and for every $n \in N$, there exists a trace compatible with the simulation and containing <x,n>.

It follows that $L_4 \Rightarrow L_3 \Rightarrow L_2 \Rightarrow L_1$. For $1 \leq i \leq 3$, we say that a state or event is strictly L_i iff it is L_i but not L_{i+1}. A net is said to be L_i iff all states and events are L_i. The net in Figure 12 illustrates the different degrees of liveness.

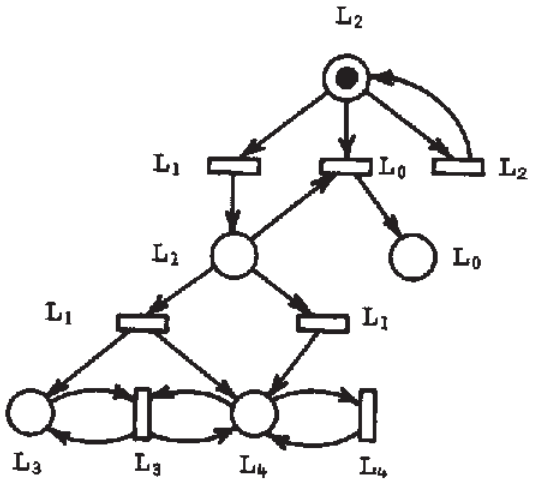


Figure 12. The Degrees of Liveness.

REFERENCES

1. Baker, H. G., Equivalence Problems of Petri Nets, S.M. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, June 1973.

2. Berge, C., Graphs and Hypergraphs, North-Holland Publishing Co., 1973.

3. Commoner, F. G., Deadlocks in Petri Nets, Report CA-7206-2311, Applied Data Research, Inc., Wakefield, Mass., June 1972.

4. Commoner, F., A. W. Holt, S. Even, and A. Pnueli, "Marked Directed Graphs", Journal of Computer and System Sciences, Vol. 5, October 1971, pp. 511-523.

5. Furtek, F. C., Modular Implementation of Petri Nets, S.M. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, September 1971.

6. Furtek, F. C., Asynchronous Push-Down Stacks, Computation Structures Group Memo 86, Project MAC, Massachusetts Institute of Technology, August 1973.

7. Genrich, H. J., Einfache Nicht-sequentielle Prozesse (Simple Nonsequential Processes). Bericht Nr. 37, Gesellschaft für Mathematik und Datenverarbeitung, Bonn. 1971.

8. Hack, M. H. T., Analysis of Production Schemata by Petri Nets, Technical Report MAC-TR-94, Project MAC, Massachusetts Institute of Technology, February 1972.

9. Holt, A. W., et. al., Final Report of the Information System Theory Project, Technical Report No. RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, New York, September 1968.

10. Holt, A. W. and F. G. Commoner, Events and Conditions, Part 1, Applied Data Research, Inc., New York, 1970. (Chapter I, II, and IV appear in Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York, 1970, pp. 3-31.)

11. Holt, A. W., Events and Conditions, Part 2.

12. Holt, A. W., Events and Conditions, Part 3. (Reprinted in Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, pp. 33-52.)

13. Jump, J. R., and P. S. Thiagarajan, "On the Equivalence of Asynchronous Control Structures," SIAM Journal of Computation, Vol. 2, No. 2, June 1973, pp. 67-87.

14. Patil, S. S., Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes, Computation Structures Group Memo 57, Project MAC, Massachusetts Institute of Technology, February 1971.

15. Petri, C. A., Communication with Automata, Supplement 1 to Technical Report RADC-TR-65-377, Vol. 1, Rome Air Development Center, Griffiss Air Force Base, New York, 1966. [Originally published in German: Kommunikation mit Automaten, Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn, Hft. 2, Bonn, 1962.]

16. Petri, C. A., Series of talks given at Applied Data Research, Inc., Wakefield, Mass., August 1973.

17. Ramchandani, C., Analysis of Asynchronous Concurrent Systems by Petri Nets, Technical Report MAC-TR-120, Project MAC, Massachusetts Institute of Technology, February 1974.