COO-3288-26

HOPKINS COMPUTER RESEARCH REPORTS
REPORT # 26
JULY 1973

COMMENTS ON CAPABILITIES, LIMITATIONS AND "CORRECTNESS" OF PETRI NETS

BY

TILAK AGERWALA

RESEARCH PROGRAM IN COMPUTER SYSTEMS ARCHITECTURE
COMPUTER SCIENCE PROGRAM
THE JOHNS HOPKINS UNIVERSITY
BALTIMORE, MARYLAND

DISCLAIMER

COMMENTS ON CAPABILITIES, LIMITATIONS AND "CORRECTNESS" OF PETRI NETS *

by Tilak Agerwala

*This work was supported by the U. S. Atomic Energy Commission under contract AT(11-1 3288)

ERRATA

1. Page 3, line 6, replace "$a_1, \ldots, a_k$" by "$\{a_1, \ldots, a_k\}$

2. Page 3, line 6, replace "x,y" by "<x,y>"

3. Page 19, line 1, replace "p" by "p"

4. Page 25, Figure 2.17, place a "○" in the right most arc between td and td

5. Page 27, Figure 2.20, place "p2 near the circle.

6. Page 33, top right diagram, replace "  t_2 " by "  t_2 "

7. Page 49, Line 2, replace "n-triple" by "n-tuple".

CONTENTS

Comments on Capabilities, Limitations and "Correctness" of Petri Nets

CHAPTER I

Introduction

1.1 Why Petri nets?

In recent years there have been numerous studies relating to the theoretical aspects of parallel computations. Various models have been proposed in an attempt to study the properties of parallel systems. Some of the well known ones are those developed by Estrin and Martin and others (the U.C.L.A. model), Rodriguez, Luconi, Karp and Miller, Adams, etc. For a comprehensive bibliography the reader is referred to [1]. These models differ in generality and scope according to the properties one is interested in studying. Some are very powerful: Adams proves that every computable function can be represented in his model. The models basically have two parts--the data flow structure and control. However, since the emphasis is on representation of computations, the overall coordination scheme is obscured. The models, for all their power and generality, are thus not suitable if one is interested in studying problems involving coordination of events and representation of such coordination. Petri nets [15], one of the earliest contributions to the theory of parallel computations, appear to be a natural way to represent the coordination of asynchronous events[1]. Others have also recognized the suitability of Petri nets in this respect. For example, the Computation Structures Group at MIT states [3]: "...we have found Petri nets to be an elegant formalism for representation of concurrency in processes and for studying asynchronous systems. Petri nets stand out in relation to other schemes because of the preciseness and ease with which they can express parallel actions, resolution of conflicts, and interaction among processes".

It should be pointed out that although we are referring to parallel computations, Petri nets are not restricted to modelling coordinations in computer systems. Any system where there are "loosely connected" essentially independant processes which proceed in an asynchronous manner can be modelled using Petri nets. Patil [13] says that they should be useful in modelling business systems and biological systems as well.

1.2 Definitions concerning Petri nets

In the next section we will present a brief survey of the work already done on Petri nets. This will give the reader further insight into the usefulness of Petri nets per se and will also provide a justification for the research presented in this paper. Before that, however, we will have to explain what a Petri net is, define some terms and give the simulation rules explicitly.

---

[1] When we use the term Petri net we refer to the modified Petri nets used by Holt [8].

Definition 1.1. A Petri net N is a directed graph defined as a quadruplet

$\langle T, P, A, B^{\circ} \rangle$

$T = \{t_1, \ldots\ldots, t_m\}$ is a finite set of transitions

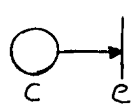
$P = \{p_1, \ldots\ldots, p_n\}$ is a finite set of places

(T, P form the nodes of the graph)

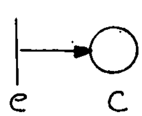
$A =\ a_1, \ldots\ldots, a_k$ is a finite set of directed arcs of the form x, y which either connect a transition to a place or a place to a transition. Each place may have one or more markers in it or it may be empty. A place is full if it has at least one marker.

$B^{\circ} = \{\langle p, n \rangle \mid p \in P \text{ and } n \in N\}$ is the initial marker distribution (initial marking).

The places are denoted by circles and represent conditions, the transitions are denoted by bars and represent events or processes.



means: Every occurrence of event e ends one holding of condition c.



means: Every occurrence of event e begins one holding condition c.

Definition 1.2. The input places of a transition t_i,

$I_i = \{p_j \mid \langle p_j, t_i \rangle \in A\}$ i.e. the set of all places from which arcs are incident on t_i.

Definition 1.3. The output places of a transition t_i,

$O_i = \{p_j \mid \langle t_i, p_j \rangle \in A\}$ i.e. the set of all places onto which arcs are incident from t_i.

Definition 1.4. A transition t_i is said to be enabled if $p_k \in O_i \Rightarrow P_k = 1$, i.e., if each input place of t_i is full. (P_i = No. of markers in p_i)

Definition 1.5. Two transitions t_i and t_j are said to be in conflict if during the simulation the net reaches a certain marking where both t_i and t_j are enabled and $I_i \cap I_j \neq \phi$, i.e., they share an input place.

Simulation rules: Whenever a transition t_i is enabled it may at some later time (finite, a priori unknown and unbounded) decide to fire. At such a time it reserves one stone[1] in each input place and begins firing. No other transition which shares input places with t_j can claim such a stone. In fact, a reserved stone is invisible to all other transitions. At the completion of firing (again the time is finite, a priori unknown and unbounded) the transition removes the reserved stones and places one stone in each of its output places. (The reasons for this particular scheme will become obvious when we give the proof techniques). If at any instant, two transitions are in conflict, the decision as to which one will fire is absolutely arbitrary and nondeterministic.

Definition 1.6. A place p_i in a Petri net is said to be safe with respect to a marking M if no simulation of the net starting from M causes more than one stone to be placed in p_i. A marking M is safe if all the places in the net are safe with respect to M.

Definition 1.7. A marking of a Petri net is said to be live if for any marking reachable from the given marking, there is a firing sequence that will enable any transition of the net.

---

[1] From this point on, we use "stone" and "marker" interchangeably.

Example 1.1. from [4]. A Petri net: $N = \langle T, P, A, B^\circ \rangle$

$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$

$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$

$A = \{\langle p_1,t_1\rangle, \langle p_1,t_2\rangle, \langle p_3,t_3\rangle, \langle p_4,t_4\rangle,$
$\langle p_2,t_3\rangle, \langle p_2,t_4\rangle, \langle p_5,t_5\rangle, \langle p_6,t_6\rangle,$
$\langle t_1,p_3\rangle, \langle t_2,p_4\rangle, \langle t_3,p_5\rangle, \langle t_3,p_6\rangle,$
$\langle t_4,p_5\rangle, \langle t_4,p_6\rangle, \langle t_5,p_1\rangle, \langle t_6,p_2\rangle\}$

$B^\circ = \{\langle p_1,1\rangle, \langle p_2,1\rangle\}$

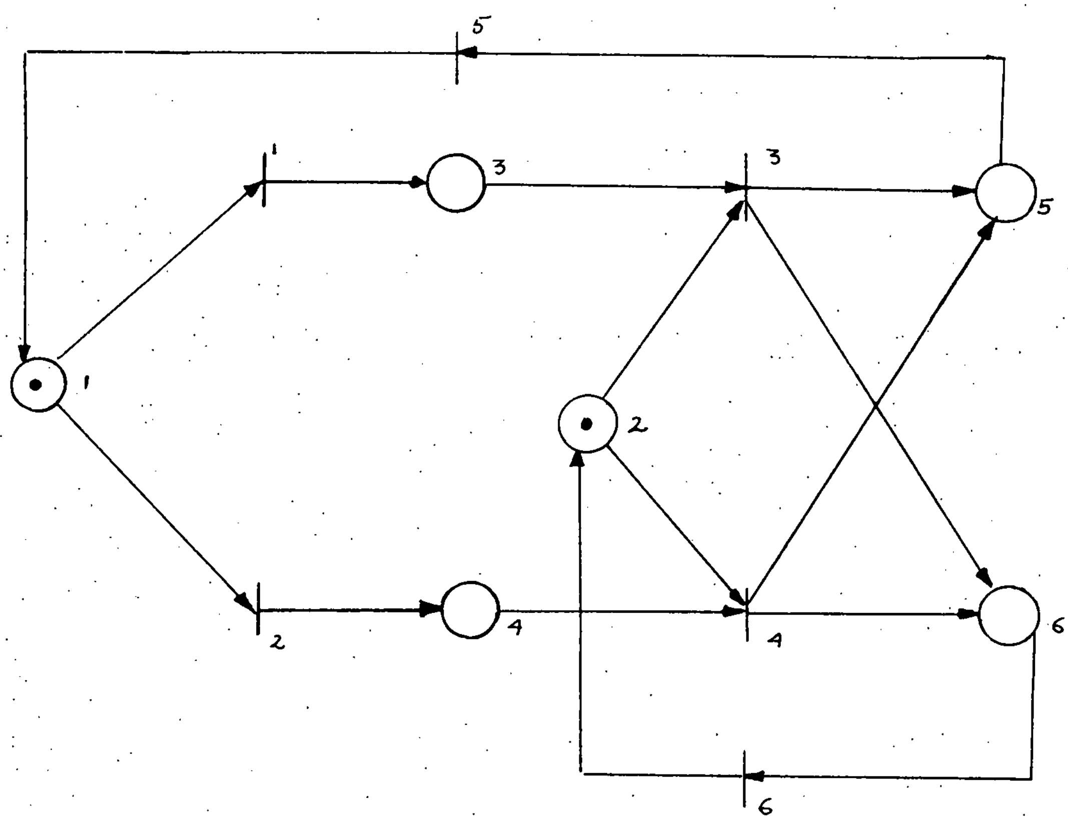


Fig. 1.1: The graph.

Simulation: An examination of the graph indicates that the only transitions initially enabled are t_1 and t_2. Also, these two transitions are in conflict. Assume that t_1 fires. As a result, t_3 is enabled, firing of t_3 enables t_5 and t_6. If t_6 fires first and then t_5, the net returns to its original condition. If t_5 fires first t_1 and t_2 are enabled. Assume t_2 fires. The net cannot proceed now until t_6 fires. If it does, t_4 is enabled which again enables t_5 and t_6 and the simulation continues.

This Petri net is live and safe with respect to $B°$.

1.3 Work related to Petri nets.

Petri nets are extremely general and thus it is difficult to study their properties. However, properties of subclasses have been examined. These subclasses are [3]:

1. Simple nets: Every transition has at most one shared input place.
2. Free choice nets: Every arc from a place to a transition is either the only output of the place or the only input to the transition.
3. Marked graphs: Every place has exactly one input transition and one output transition.
4. State machines: Every transition has exactly one input place and one output place.

Genrich [7] and Holt and Commoner [8] have studied properties such as liveness and safety of marked graphs. In [8] the concept of information flow through a system has been introduced and studied in the context of state machines. Patil [14] has used simple Petri nets to establish the correspondence between cooperating sequential processes [5] and Petri nets. Simple Petri nets represent the flow of control in processes where the processes do not use conditional statements and the only synchronizing primitives are Dijkstra's P and V operators [5]. Patil presents a coordination problem that cannot be solved using Simple Petri nets. In this context we would also like to mention that Kosaraju [9] has shown that there exist problems that cannot be solved using even general Petri nets.

In [13] Patil presents another scheme for representing coordinations and claims that it leads to reduction in details and simplification of representation as compared to Petri nets. The nets called coordination nets are a generalization of Petri nets but do not add more variety to the class of coordinations represented by Petri nets. He also presents asynchronous control structures for implementing coordination nets and shows how coordination structures can be derived systematically from the nets. He believes that these modules can be implemented in hardware systematically.

Dennis [2] has used Petri nets to describe the control mechanism of a computer with multiple functional units. For each of the six major units in the machine, Petri nets and modular control structures are presented. The control structures are constructed from primitive modules whose behavior is specified in terms of p-nets which are abbreviated representations of Petri nets. Dennis points out that the ultimate aim of studies such as this is to understand how to translate a Petri net specification into an efficient digital system.

Seitz [16] provides an analogy between Huffman primitive flow tables for asynchronous sequential machines and Petri nets. He then generalizes the Huffman flow table as a special form of Petri net called an m-net. The m-net can now be used to design an asynchronous machine exhibiting concurrency in much the same way as Huffman tables are used in designing asynchronous sequential machines. The author emphasizes that the m-net representation is a very practical one which permits orderly design of machines which would be difficult to design by other methods.

Noe [12] is concerned with measurement and evaluation of computer systems. He introduces Petri nets with EOR input, EOR output and Inclusive OR input logics for the description of operating systems at different levels of detail. The paper describes a multiprocessor, multiprogramming system, the CDC 6400, in terms of Petri nets and shows how this type of representation lends itself to planning system measurements.

Shapiro and Saint [17] use Petri nets and o-systems for the solution of an optimization problem. The problem they focus attention on is that of generating efficient programs to run on a parallel machine starting with an algorithm specified in a high level language. Many different sequences of operations may be representations of a given

I/O mapping. If the target machine is capable of parallel operation (e.g., CDC 6600 or IBM 360/91) efficiency of execution may vary greatly depending on the particular sequencing chosen. Petri nets are used to express the algorithm in a form where incidental sequencing constraints imposed by the algorithmic language are removed. This process is called decompilation and the resulting net presents maximum asynchrony. The sequencing constraints required by the target hardware are then introduced into the net. All sequences of which this net is capable, are realizable on the target equipment and perform the correct mapping.

1.4 Motivations for current research

In the previous sections we have tried to establish that Petri nets are a neat and convenient way of representing coordinations and can be used as tools for the specification, design and evaluation of complex computer systems. If we are going to use Petri nets to represent coordinations we must, first and foremost, be aware of the capabilities and limitations of Petri nets; else we may end up trying to represent a coordination for which there is no Petri net representation. For a while it was felt that Petri nets were all-powerful, i.e., all coordinations could be represented using Petri nets. We know now that this is not true. It would be interesting to see if the power of Petri nets can be increased by suitable modifications. This will further improve our understanding of the capabilities of Petri nets. These ideas will be discussed in Chapter II.

We have seen that Petri nets can be used in the specification of computer systems. The first question that arises is: given a humanly statable computing problem and a Petri net, how does one show that the Petri net "correctly" represents the desired coordination? We have not seen anywhere an attempt to "prove the correctness" of a given Petri net and we tackle this problem in Chapter III[1]. Another motivation for correctness of Petri nets is that proving correctness of parallel programs in general and cooperating sequential processes in particular is extremely difficult and suitable simple techniques do not exist. Developing proof techniques for Petri nets may simplify matters in two ways:

1. If it is possible to mechanically translate a program P into a Petri net N, one may be able to prove properties of P more easily in the framework of N.
2. Having developed techniques for proving properties of Petri nets, it may be possible to suitably modify these techniques for application to proving correctness of parallel programs or at least to provide some insight into the techniques to be used.

---

[1] We use quotes because terms such as correctness and proof can have many different meanings; In Chapter III we will make it clear what is meant.

CHAPTER II

Capabilities and Limitations of Petri Nets

2.1 Introduction

In his thesis [13] Patil states: "The author has found Petri nets to be adequate in representing coordination of events, but it appears that a claim that Petri nets provide a satisfactory formal counterpart to vague notions about coordination of asynchronous events cannot be proved just as the claim that Turing machines provide a satisfactory formal counterpart to the vague concept of algorithm cannot be proved. The claim must be accepted or rejected on the basis of experience and the experience of the author and that of others indicates that Petri nets provide a satisfactory formalism for the study of coordination of asynchronous events". Patil seems to feel that any coordination problem can be represented as a Petri net. This, however, is not true. In what follows, we will further discuss this point. We will try to modify Petri nets by introducing some transitions and places with different properties and see whether the overall power is increased.

2.2 Interpretation and equivalence

The transitions in a Petri net are labelled, by definition, t_1, t_2, ..., t_n. Each transition has a distinct label. However, these transitions can be interpreted in any way we choose. For example, two transitions with different labels may refer to the same event or process. This is allowed even if the two transitions can fire at the same time. The same transition cannot, however, refer to two different processes. Firing of a transition corresponds to the occurrence of an event or the initiation and completion of some process. Thus a net is interpreted if process names are attached to some transitions. For an interpreted net, we are interested only in the manner in which the named processes interact. We can now define two kinds of equivalence:

1. Strong Equivalence:

   N_1 and N_2 are equivalent with respect to a set of transitions, T, iff $T \subseteq T_1$ and $T \subseteq T_2$ and the same firing sequences of transitions in T can be achieved in both N_1 and N_2. N_1 and N_2 are strongly equivalent if they are equivalent with respect to T and $T = T_1$ or $T = T_2$.

2. Weak Equivalence:

   Two nets are equivalent with respect to an interpretation iff

   (a) The processes named in one are the same as those named in the other, and,

   (b) The sequences of process initiation and completion achievable in the nets are identical.

   Two nets are weakly equivalent if they are equivalent with respect to at least one interpretation.

Note: Thus the same Petri net can refer to different coordinations among processes depending on the interpretation given to it.

2.3 Classes of coordination problems and nets

1. Let the class of regular[1] Petri nets be TN and the coordinations representable by this class, PN.

2. Regular Petri nets have only AND-input logic, i.e., a transition is enabled iff all the input places are full. Let us consider Petri nets that have Inclusive OR-input logic as well. For example:

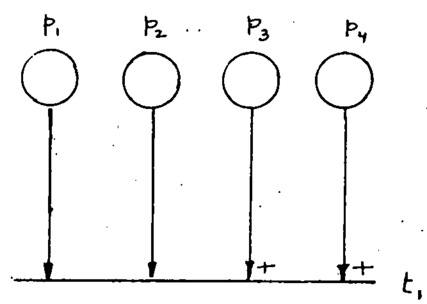


Fig. 2.1

(Let P_i = No. of stones in p_i)

In Fig. 2.1 t_i is enabled iff $[(P_1 > 0) \wedge (P_2 > 0)] \wedge [(P_3 > 0) \vee (P_4 > 0)]$. We should explain clearly the simulation of such nets. Consider the net below:

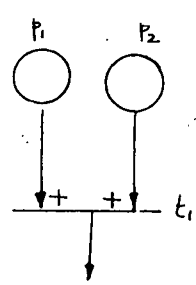


Fig. 2.2

[1] Regular Petri nets refers to the nets described in section 1.2.

It should be clear to the reader that the final state of the net will depend on the state at the instant t_1 decides to fire. Let $P_1 = 1$ and $P_2 = 0$ at some instant. Then t_1 is enabled and let it decide to fire. Immediately thereafter let a stone appear in p_2. Then, when t_2 finishes firing, $P_1 = 0$ and $P_2 = 1$. If t_1 decided to fire after the stone appeared in p_2 the final situation would be $P_1 = 0$ and $P_2 = 0$.

Let TN_{log} be the class of nets and PN_{log} the class of problems where the desired coordination can be represented using these nets.

3. In addition to the regular arc between transitions and places we allow the following type of arc:

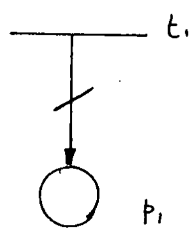


Fig. 2.3

When t_1 fires, a marker is placed in p_1 iff $P_1 > 0$. Let TN_{out} be the class of nets and PN_{out}, the class of coordinations representable by these nets.

4. We introduce a special place ◌, (say p_1). A transition will place a stone in p_1 iff $P_1 = 0$. Let us call this class of net $TN_{\overline{out}}$ and the corresponding class of coordination problems, $PN_{\overline{out}}$.

5. In addition to the regular arcs between places and transitions, we allow a special arc.

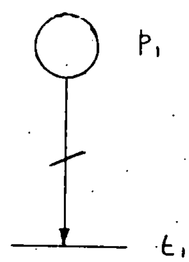


Fig. 2.4

t_1 is enabled iff $P_1 = 0$. Let this class of nets be $\overline{TN}$ and the class of coordination problems $\overline{PN}$.

2.4 <u>Results</u>

1. Obviously,

$$PN \subseteq PN_{log} \quad (a)$$

$$PN \subseteq PN_{out} \quad (b)$$

$$PN \subseteq PN_{\overline{out}} \quad (c)$$

$$PN \subseteq \overline{PN} \quad (d)$$

2. PN $\subset$ $\overline{PN}$

Proof: Kosaraju, in [9], describes a coordination problem and proves that it falls outside PN. The problem is as follows: There are two producers, P_1 and P_2, two consumers, C_1 and C_2, and two buffers, B_1 and B_2. If P_i is activated, it produces an item, deposits it on top of B_i and deactivates itself. If C_i is activated, it consumes the bottom item from B_i and deactivates itself. Another constraint is added. C_1 and C_2 cannot be active simultaneously; C_1 has priority over C_2, i.e., if both C_1 and C_2 are inactive and buffer B_1 is not empty, then C_2 cannot consume from B_2 (since C_1 can be activated) at that instant. To prove that PN $\subset$ $\overline{PN}$ we will show that the coordination desired in the above example can be represented using a net N $\in$ $\overline{TN}$.

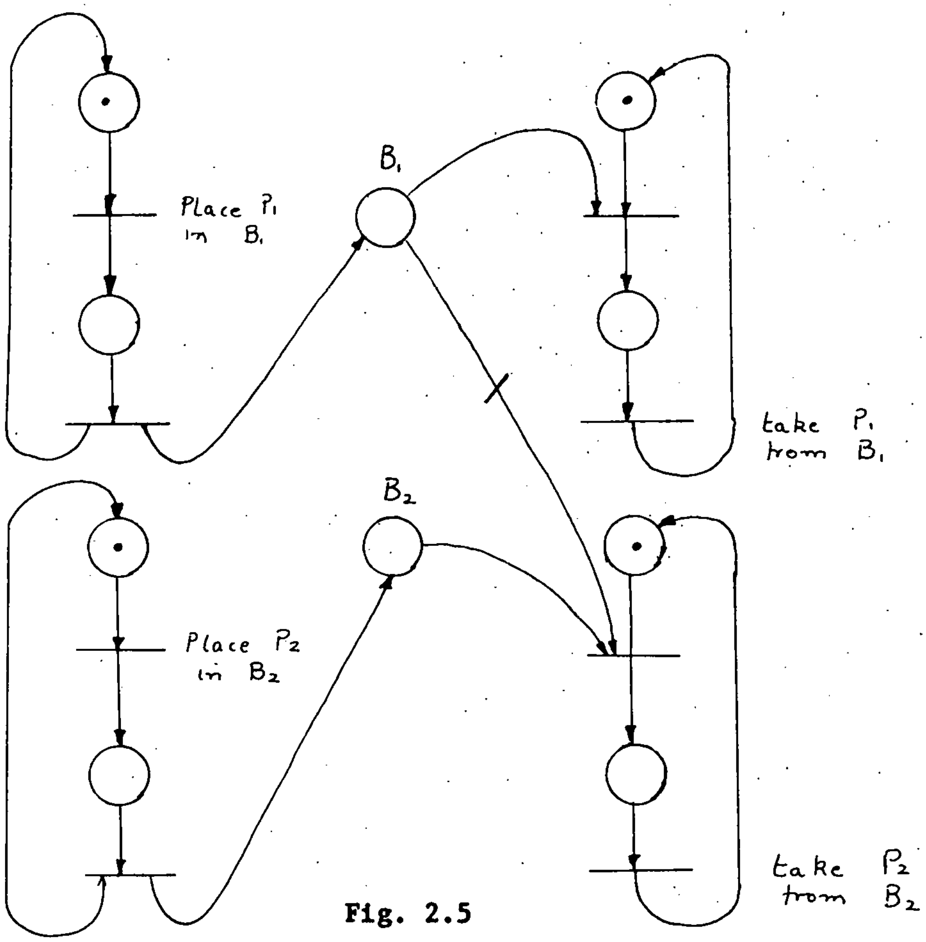


Fig. 2.5

To give a better understanding of why regular Petri nets are not all powerful we will consider another example. Kosaraju's proof will go through for this case also, but here we are more interested in an intuitive discussion. Consider the following simple net:

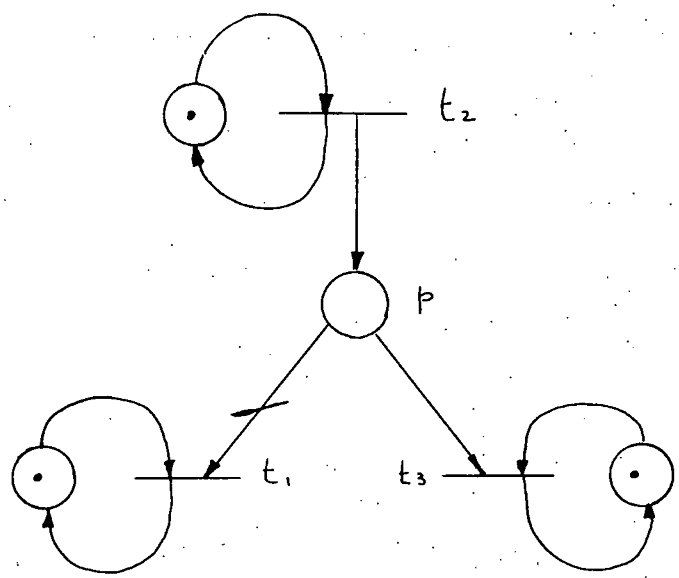


Fig. 2.6

This net has an interesting interpretation. In fact, it is very similar to the control structure that implements "cycle stealing" in a multi-programmed computer. t_1 can be interpreted as the main process PROC 1 to which the CPU is allocated. t_2 is an input process PROC 2, acting asynchronously and reads (say card images) into a buffer. t_3 is the process PROC 3 that reads from the buffer into main memory. If the buffer is not empty, the main process PROC 1 is halted until PROC 3 reads all the information into core. Only at this instant, when the buffer becomes empty again, is PROC 1 allowed to continue. We said "very similar" in the beginning because usually the buffer is bounded and consumption takes place instantaneously.

We will now give an intuitive argument to show that the desired coordination cannot be achieved using a regular Petri net.

Initially, t_1 is capable of firing an unlimited number of times until t_2 chooses to fire. Since places cannot initially hold an unbounded number of stones, a part of the net has to be equivalent to Fig. 2.7 with respect to t_1 .

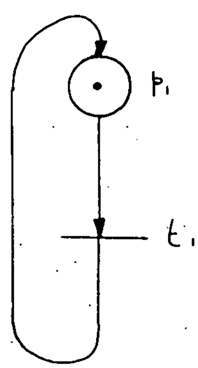


Fig. 2.7

Let T_1 = No. of times t_1 has fired till any given instant.
If $T_2 > T_3$, t_1 has to be disabled, and obviously, this can only be done by removing the stone from p_1. Also, when T_2=T_3, t_2 fires independant of the rest of the net. Therefore a part of the net must be equivalent to:

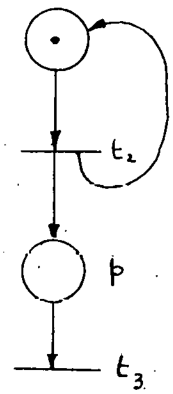


Fig. 2.8

Placing a stone in P must cause the stone in p_1 to be removed. Therefore, p_1 must be an input place of t_2 or some other transition that causes a stone to be placed in P. Thus, the coupling may be as follows:

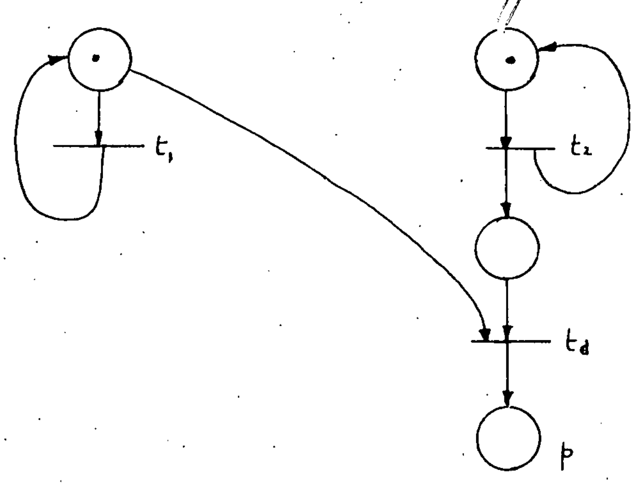


Fig. 2.9

Assume that in the original net t_2 fires twice and t_3 does not fire. Then, t_1 is disabled as soon as the first stone is placed in p and will remain so until t_3 fires twice, which it is capable of doing since p contains two stones. The constructed net works correctly to a point. When t_2 fires the first time a stone appears in p and t_1 is disabled (Assume dummy transition t_d fires instantaneously). However, when t_2 fires a second time, no stone appears in p since t_d is not enabled. Therefore, t_1 has to be enabled in order to cause the second stone to be placed in p. But the coordination desired requires that t_1 not be enabled until t_3 has fired twice. Even if this problem can be taken care of, there is no way of constructing the net so that t_1 is reenabled only when t_3 has fired the same number of times as t_2. We thus argue that there is no regular Petri net strongly equivalent to the net in Fig. 2.6.

The closest we can get to the desired coordination using regular Petri nets is the net in fig 2.10:

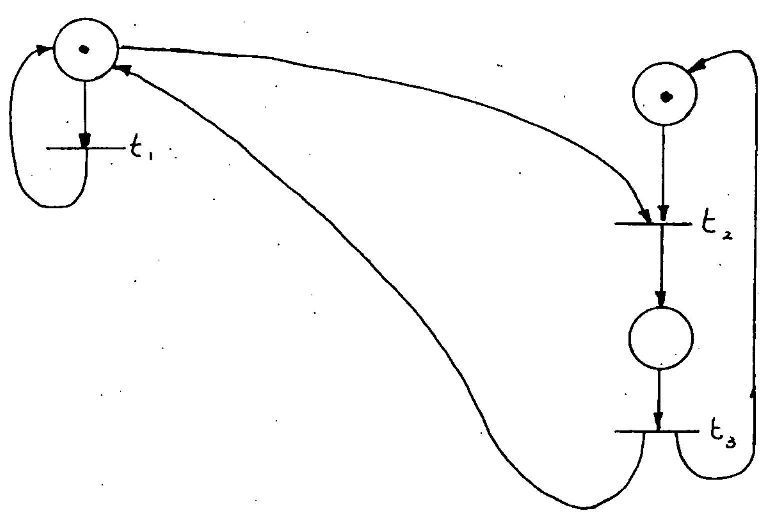


Fig. 2.10

Here the only sequences allowed are

$$t_1t_1 \ldots\ldots t_1t_2t_3t_2t_3\ldots\ldots\ldots\ldots t_2t_3t_1t_1\ldots\ldots\ldots$$

In the original problem we also allowed sequences like

$$t_1 \ldots\ldots t_1 \underbrace{t_2t_2\ldots t_2}_{n} \underbrace{t_3t_3\ldots\ldots t_3}_{n} t_1t_1\ldots\ldots \quad \text{etc.}$$

OR

$$t_1\ldots.t_1 \underbrace{t_2\ldots.t_2}_{a} \underbrace{t_3\ldots.t_3}_{b} \underbrace{t_2\ldots.t_2}_{c} \underbrace{t_3\ldots.t_3}_{d} t_1\ldots.t_1$$

where $a + c = b + d$.

3. $PN \subset PN_{log} \subset \overline{PN}$

(a) $PN_{log} \subseteq \overline{PN}$.

Proof: The net in Fig 2.11 can be simulated using a net $N' \in \overline{TN}$.

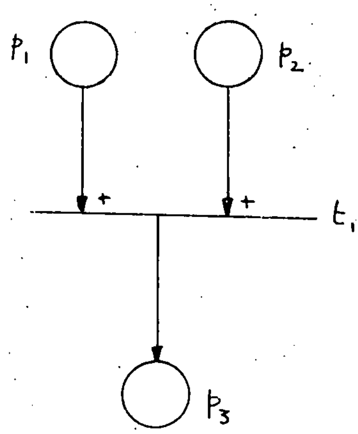


Fig. 2.11

The simulation is:

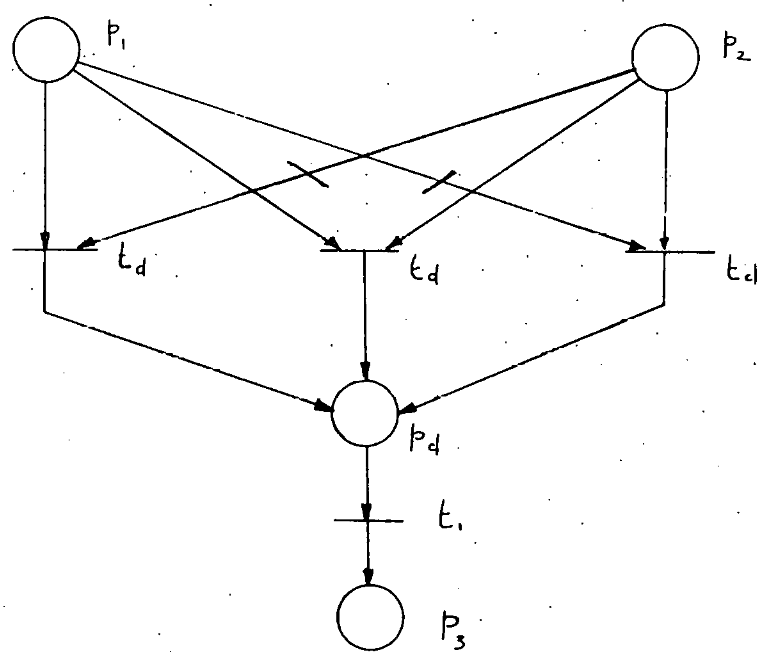


Fig. 2.12

Where t_d and p_d are dummy transitions and places respectively. Therefore, for any $N = \langle T, P, A, B° \rangle \in TN_{log}$ $\exists$ a net $N' \in \overline{TN} \ni N'$ is equivalent to N with respect to T, i.e., N' and N are strongly equivalent.

(b) Kosaraju's problem 1 and proof can be used to show that $PN_{log} \subset \overline{PN}$. Informally, Fig. 2.6 can be used to show the same thing. The conditions under which t_1, t_2 and t_3 are enabled are:

t_2	:	always	(1.)
t_1	:	iff $T_2 = T_3$	(2.)
t_3	:	iff $T_2 > T_3$	(3.)

The conditions (2) and (3) are single conditions. We cannot really take advantage of the fact that OR-input logic is allowed. We may replace $T_2 = T_3$ by $(T_2 = T_3 \wedge X) \vee (T_2 = T_3 \wedge \overline{X})$ where X is some condition but this will not help because if we can test X we cannot test $\overline{X}$ and vice versa.

(c) Also, informally, we can conclude that $PN \subset PN_{log}$. Consider the net below:

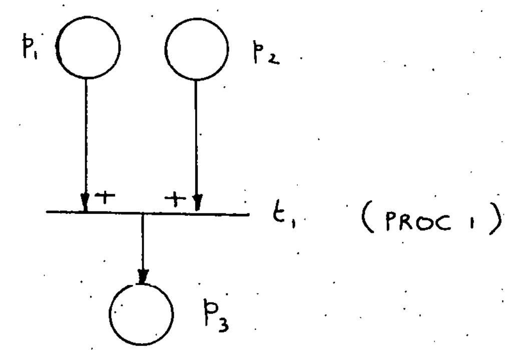


Fig. 2.13

t_1 represents some process PROC 1. PROC 1 is activated only if $P_1 > 0 \vee P_2 > 0$. Since in the original net PROC 1 will be enabled if $P_1 > 0$ a part of the net $N \in TN$ must be equivalent to:

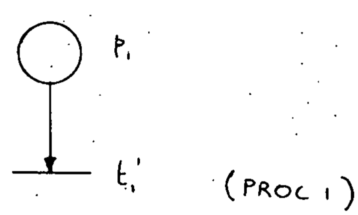


Fig. 2.14

Similarly, a part of N must be equivalent to

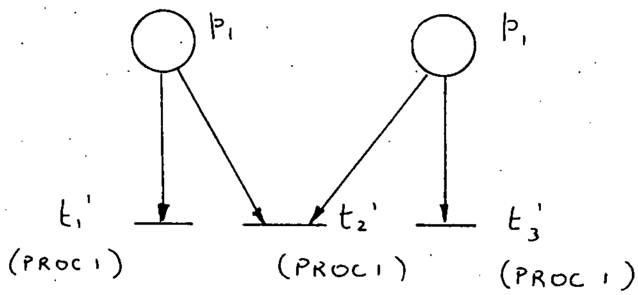


Fig. 2.15

In the original net, if stones appear in p_1 and p_2 at the same time, PROC 1 will be activated only once. In Fig. 2.15 PROC 1 can be activated twice if t_1' and t_3' fire instead of t_2'.

4. $\underline{PN \subset PN_{\bar{out}} \subset \overline{PN}}$

(a) $PN_{\bar{out}} \subseteq \overline{PN}$

Proof: The net in Fig. 2.16 can be simulated using a net $N' \in \overline{TN}$.

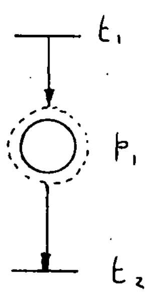


Fig. 2.16

The simulation is:

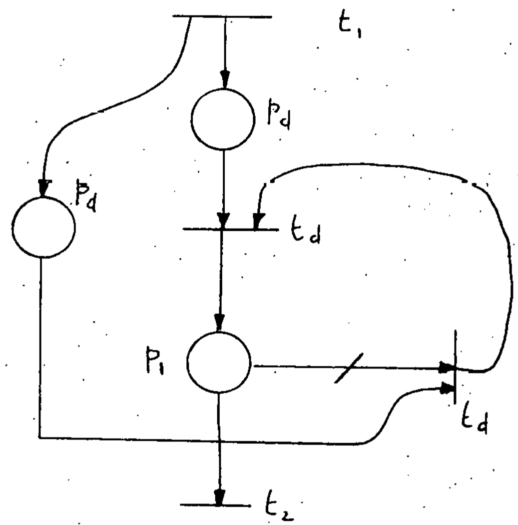


Fig. 2.17

Where t_d, P_d are dummy transitions and places. Therefore, for any $N = \langle T, P, A, B^\circ \rangle \in TN_{\overline{out}}$ $\exists$ $N' \in \overline{\overline{TN}}$ $\ni$ N and N' are equivalent with respect to T, i.e., N and N' are strongly equivalent.

(b) Kosaraju's problem 1 and proof can be used to prove that $PN_{\overline{out}} \subset \overline{PN}$.

(c) We will demonstrate, with an example that $PN \subset PN_{\overline{out}}$. Consider the simple net in Fig. 2.18

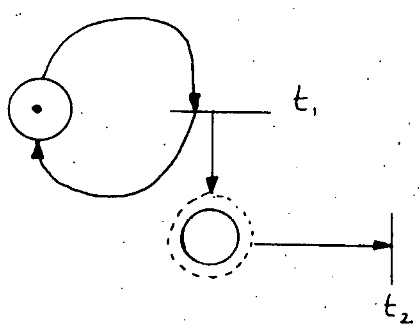


Fig. 2.18

Consider the following interpretation. t_1 is a producer, PROD, which produces items one at a time and deposits them in a buffer. t_2 is a consumer CONS which, when enabled, consumes everything in the buffer. Let us assume that t_1 is firing at a fixed rate and t_2 fires whenever it is enabled. Therefore, according to the figure, the number of times CONS is activated depends on the time it takes for consumption. It should be intuitively obvious to the reader that the coordination specified cannot be achieved using regular Petri nets.

(d) We will discuss an example here, that establishes the need for two notions of equivalence. Consider the net N' of Fig. 2.19.

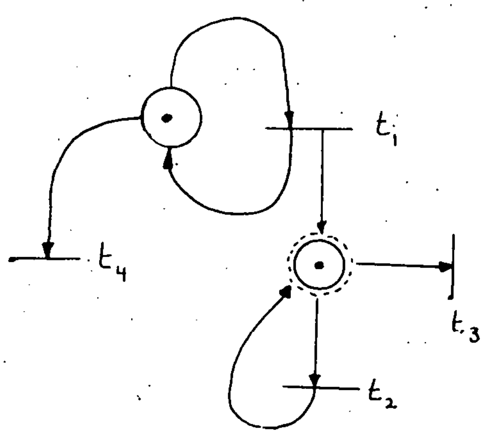


Fig. 2.19

We will show that there is no net $N \in TN \ \exists \ N'$ and N are equivalent with respect to $\{t_1, t_2, t_3, t_4\}$. We will then give the net in Fig. 2.19 an interesting interpretation and show that there is a net $N \in TN$ which is equivalent to N' with respect to this interpretation.

In attempting to construct a regular Petri net equivalent to N' with respect to $\{t_1, t_2, t_3, t_4\}$, we go through the following stages.

t_2 must be capable of being fired an unbounded number of times independantly until t_3 is fired. Therefore, a part of the net has to be equivalent to:

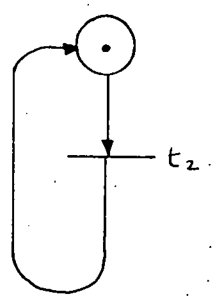


Fig. 2.20

In the original net we had a transition t_1 firing continuously, which kept t_2 enabled, even if t_3 caused it to be disabled temporarily. If we are using regular Petri nets this will not be possible since a large number of stones may accumulate in p_2. If this happens, then t_2 cannot be disabled by one firing of t_3. Thus we will have to use a mechanism whereby t_3 itself causes t_2 to be enabled:

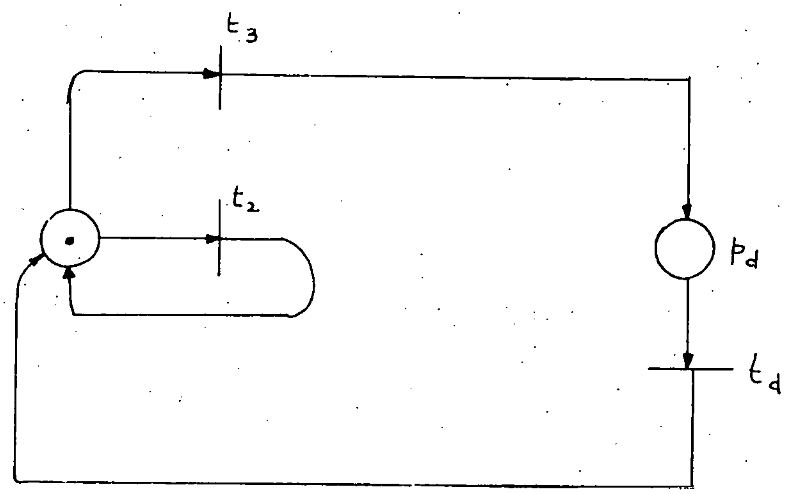


Fig. 2.21

The problem now is that if t_4 has previously fired, t_d should be prevented from firing and there is no way to do this.

Note: The net in Fig. 2.22 would not be valid because, in the original net, t_4 can fire in the beginning

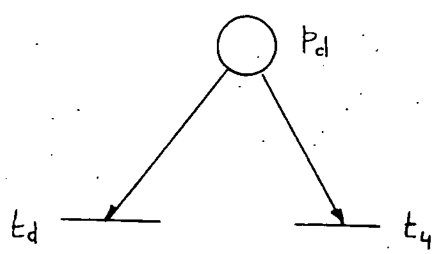


Fig. 2.22

Consider now the following interpretation of the net in Figure 2.19. t_2 represents some process PROC 1 which is enabled and can proceed independantly. t_3 is some "interrupt process." t_4 is an "unmask process." If the "unmask process" is not activated, and the "interrupt process" is activated, PROC 1 is temporarily disabled. Temporarily, because there exists a way of enabling it immediately. If, however, the "unmask process" is activated, subsequent, activation of the "interrupt process" causes PROC 1 to be permanently disabled.

The interpreted net of Fig. 2.23 is a regular Petri net and correctly represents the coordination specified in the above interpretation.

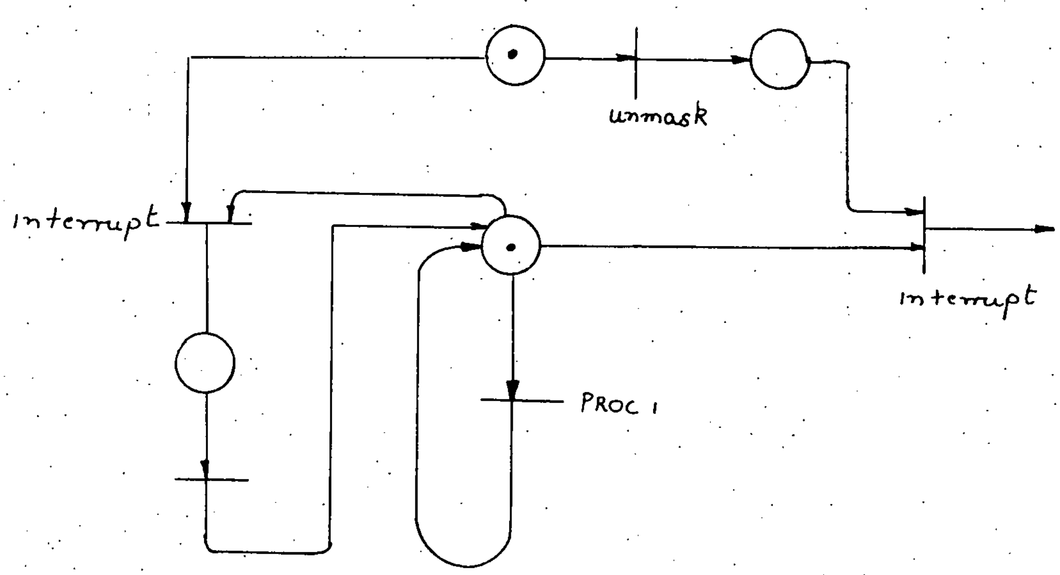


Fig. 2.23

5. <u>$PN_{out} = PN$</u>

<u>Proof</u>: We will show that the net of Fig. 2.24 can be simulated using a regular Petri net.

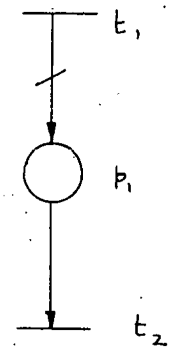


Fig. 2.24

The simulation is:

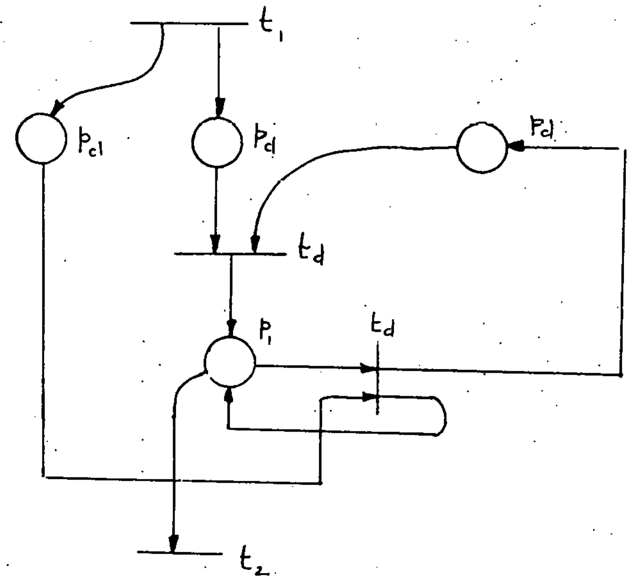


Fig. 2.25

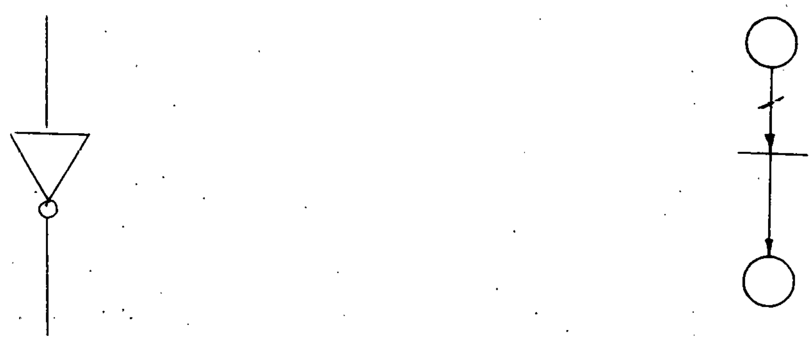
Therefore, for every net $N' = \langle T, P, A, B^\circ \rangle \in TN_{out}$ $\exists$ a net $N \in TN$, $\exists$ N' and N are equivalent with respect to T. This proves that $PN_{out} \subseteq PN$ and from result 1b we conclude that $PN_{out} = PN$.

2.5 Analogy with switching circuits and functional completeness

In switching theory, the notion of functional completeness is a well defined one, i.e., the AND and NOT functions, for example, form a "basis" and are capable of realizing all logical functions of 2 (and hence n) variables. We have seen, in this chapter, that $PN \subset \overline{PN}$. In TN we only allowed transitions with AND-input logic; in $\overline{TN}$, we allow NOT-input logic as well. In some sense at least, $\overline{TN}$ seems to be functionally complete with respect to the class of coordination problems. However, the last statement should be made carefully because though a "logical function" is a very well defined concept, a "coordination problem" is not.

Fig. 2.26

The output of the AND gate is high when all the inputs are high. The Petri net N fires when all input places are full, a stone is then put in the output place.



NOT gate Petri net N'

Fig. 2.27

The analogy between the NOT gate and N' is also strong. The question that immediately arises is: How far can we push this analogy? Can we argue that switching circuits are completely analogous to Petri nets and conclude from here independantly that there are coordination problems outside PN (since TN, by analogy, is not a basis)? We tend to think that this is not the case. Consider safe Petri nets. Since each place can contain at most one stone, this appears to be "more analogous" to switching nets where inputs are either Hi or Lo. If this is the case, we would tend to conclude that

$$TN\Big|_{safe} \subset \overline{TN}\Big|_{safe}$$

where $X\Big|_{safe}$ is the class of problems where the desired coordination can be represented using safe nets belonging to X. However, this is false as the simulations below indicate.

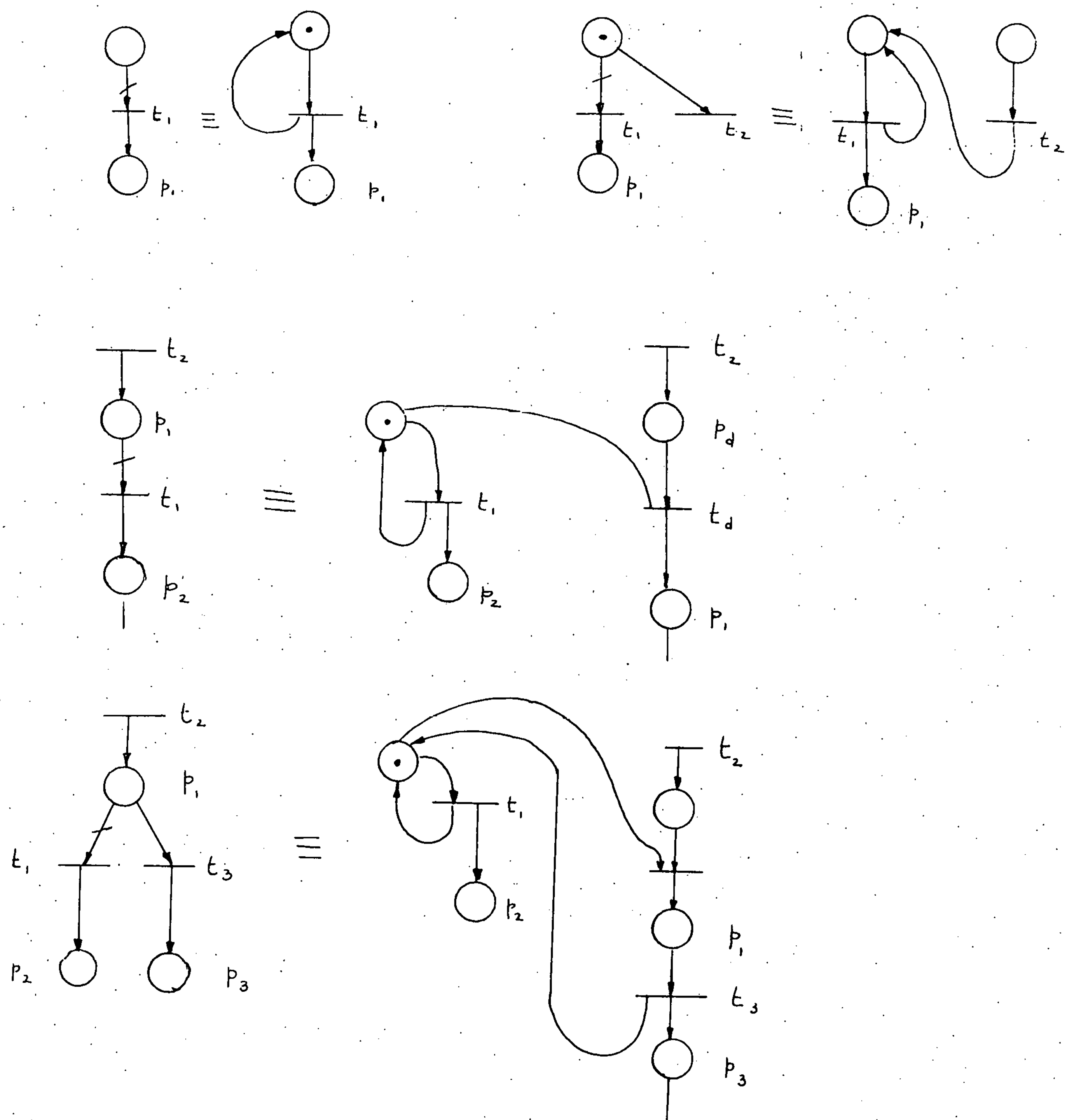


Fig. 2.28

$$\therefore \quad TN \Big|_{safe} \equiv T\overline{N} \Big|_{safe}$$

At this stage, we would only like to state that one should be careful in drawing the analogy between switching circuits and Petri nets. Also, the notion of functional completeness seems to apply to Petri nets as well. In a later report we will discuss this in greater depth.

CHAPTER III

Correctness

3.1 Introduction

Let us now come to the question of correctness of Petri nets. When we say "Petri net N is correct", intuitively what is meant is that the Petri net does what the designer intended it to do. Given a particular problem, a Petri net is constructed which represents the desired coordination. First and foremost, we are not at all interested in whether the constructed Petri net is the best one for the given problem. In fact we will not even try to prove that the Petri net effectively represents the desired coordination. We shall, however, try to prove a very restrictive kind of statement about the net which we will ask the designer to provide. The kinds of statements we will attempt to prove for a given Petri net are:

1. At any given time, only one of the transitions from the set $t_1,\ldots\ldots t_k$ may be firing.

2. Statements about termination:

   Termination may occur in two ways:

   (a) Natural termination: The net terminates because it has completed its actions according to the design specifications.

(b) Abnormal termination: The net terminates as a result of conditions not specified in the design. This kind of termination is usually called deadlock. Proving that a particular net is free from deadlock may be the most difficult aspect. The problem arises because it is difficult to recognize in general all the possible conditions under which deadlock may occur. For example, if we show that at every stage some transition in the net is live we cannot conclude from this that the net is deadlock free. The transition which is live at every stage may be the same one and useless as far as the actual operation of the net is concerned. If we prove that at every stage the marking is live then will show that the net is deadlock free but this condition is too strong and may be difficult to prove. A certain part of the net may become 'dead' after the initial stages but the rest of the net may be deadlock free and operating correctly. In this case after the initial stages every marking will not be live. Thus it will be difficult to formulate conditions which can be proved to hold for any Petri net and which will ensure deadlock free operation. For a given net, given a description of the way it is supposed to behave, it may become apparent under what conditions the net can become deadlocked. For example, by showing that every transition in a particular cycle is live at every stage, one may be able to conclude that the net cannot terminate abnormally. The above discussion seems to indicate that we should not try to prove the statement "Net N is deadlock free" in general but should ask the designer to give us simpler statements to prove from which he can reasonably conclude that the net will never become deadlocked.

3. Two given transitions will never conflict.
4. A given place is safe with respect to a particular marking or a given marking is safe.
5. A given transition is live.
6. A given marking is reachable from another.
7. A given transition will fire N times or less than N times.

Thus, our aim is not to prove that a Petri net is correct but to prove that it is correct with respect to an assertion that is made up of statements of the above type. In the above, we have used the term "prove" frequently. In [11] Mills states in connection with proofs of correctness for sequential programs: "There is no such thing as an absolute proof of logical correctness. There are only degrees of rigor,..." and "It is clear that a whole spectrum of rigor will be useful in correctness proofs." We agree with him that formality and brevity do not cooperate and have often sacrificed the former for the sake of the latter.

We would also like to comment briefly on the effects of a transition firing. There are two possible effects.

(a) A change in marker distribution.

(b) A change in the value of some variable not part of the net as a result of the occurrence of the event which the transition represents.

The Petri net may be the representation of a complicated parallel computation. We will never make statements about the values of variables which the transition firings may effect. In a sense, we are interested only in the control structure and not the actual mathematical computation.

3.2 Proof techniques

1. Computational Induction: In this method we will develop certain relations which remain invariant during the simulation of the net. By using these relations suitably, we will be able to prove certain properties about the net. When we say a relation is invariant we mean it holds whenever a change in marker distribution takes place or a transition fires. The relations follow trivially from the simulation rules.

The places are labelled $p_1, p_2, \ldots\ldots p_m$

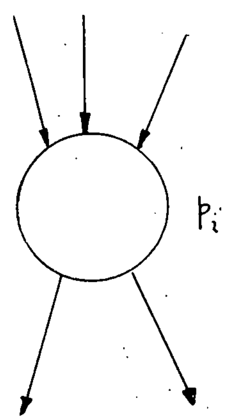
The transitions are labelled $t_1, t_2, \ldots\ldots t_n$

M_i : No. of stones in p_i initially

P_i : No. of stones in p_i

T_i : No. of times t_i has fired.

Relation 1



$\{I_i\}$: set of transitions with p_i as output place

$\{O_i\}$: set of transitions with P_i as an input place

Fig. 3.1

$$P_i = \sum_{t_k \in I_i} T_k - \sum_{t_j \in O_i} T_j + M_i \geq 0$$

<u>Example 3.1</u>

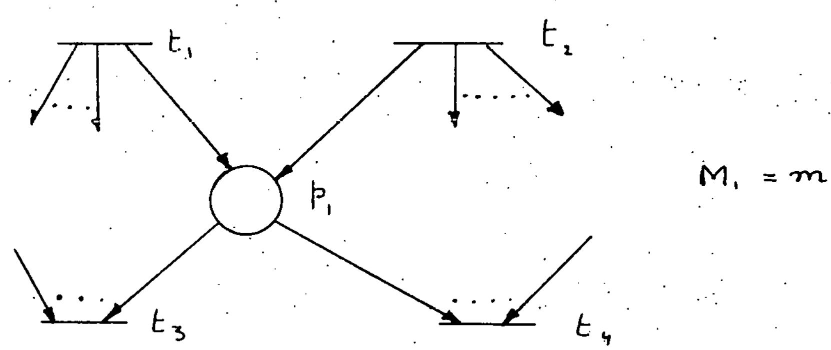


Fig. 3.2

For the net in Fig. 3.2, the relation below holds

$$T_3 + T_4 \leq T_1 + T_2 + m$$

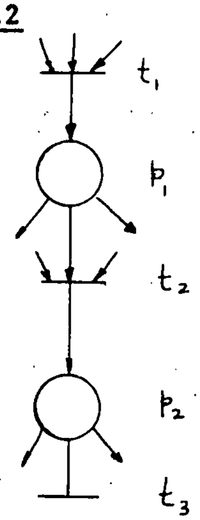
<u>Relation 2</u>

Starting from transition t_{a1} let us trace down the net along any path to transition t_{ak}. Let the path be

$$t_{a_1}\ p_{b_1}\ t_{a_2}\ p_{b_2}\ \cdots\cdots\ p_{bk-1}\ t_{ak}$$

If $a_1 \neq a_2 \neq \cdots\cdots \neq a_k$ and $I_{bi} = \{t_{ai}\}$ we call such a path a simple path.

<u>Example 3.2</u>



$t_1p_1t_2p_2t_3$ is a simple path.

Fig. 3.3

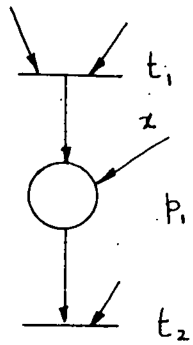


Fig. 3.4

$t_1p_1t_2$ is not a simple path because of the arc marked x coming into p_1.

$$T_{ak} \leq T_{a_1} + \sum_{i=1}^{k-1} M_{bi} \quad \text{for any simple path } t_{a_1} \; p_{b_1} \ldots t_{ak}$$

Relation 3

Given a simple path S_1 from t_i to t_j and also a simple path S_2 from t_j to t_i, then S_1S_2 forms a simple cycle.

If in addition, every place in a simple cycle has only one input and one output arc we have a pure cycle.

Let S be a pure cycle

$$\sum_{p_i \text{ in } S} P_i = \sum_{p_i \text{ in } S} M_i = N_s \quad \text{(say)}$$

In this context we can also state another simple property. If there exists a simple cycle in the net and all places in it are initially empty, then no transition on that cycle can ever fire.

Let us see how we can use these simple relations to prove properties about Petri nets.

Example 3.3

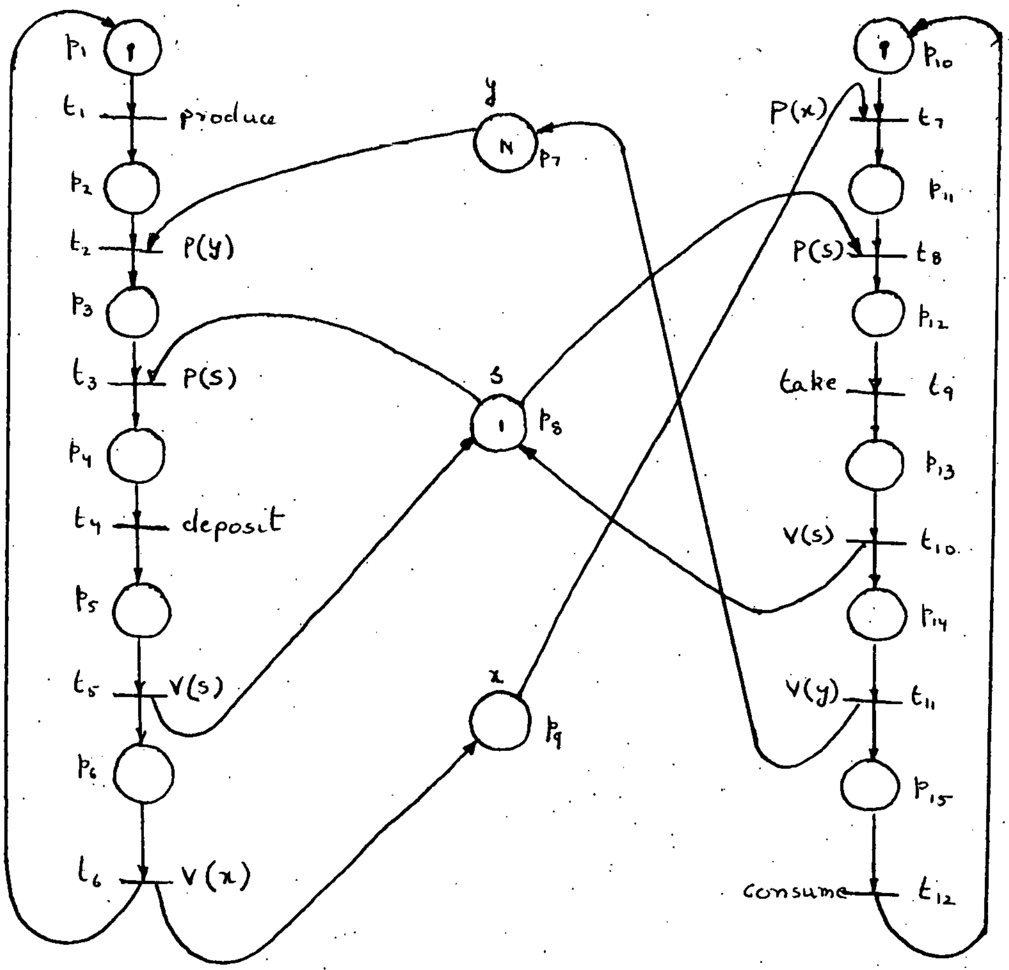
Consider the producer-consumer problem with bounded buffer. We have one producer and one consumer. The producer places items in the buffer (length N) and the consumer consumes the items. The problem is to coordinate these two essentially independant processes so that the consumer does not try to take an item from the buffer when it is empty and the producer does not place an item when the buffer is full[1]. The solution, using Dijkstra's P and V operations, is as follows:

Producer:		Consumer:	
	Produce		P(x)
	P(y)		P(s)
	P(s)		take
	deposit		V(s)
	V(s)		V(y)
	V(x)		consume
	go to producer		go to consumer

A direct mechanical translation gives the following Petri net:

---

[1] This is a famous problem solved by Dijkstra [5] and is quite different from the problems suggested by Kosaraju.



p1
t1
produce
p2
t2
P(y)
p3
t3
P(s)
p4
t4
deposit
p5
t5
V(s)
p6
t6
V(x)
y
N
p7
s
1
p8
x
p9
p10
P(x)
t7
p11
P(s)
t8
p12
take
t9
p13
V(s)
t10
p14
V(y)
t11
p15
consume
t12

Fig. 3.5

The numbers inside the places represent the initial number of markers.

We are interested in proving the following properties for this net.

(1) t_4 and t_9 cannot be firing at the same time, i.e., producer P and consumer C do not try to access the buffer at the same time.

(2) $0 \leq T_4 - T_9 \leq N$, i.e., no buffer overflow or underflow.

(3) No deadlock.

<u>Proof 1.</u>

$I_8 = \{t_{10}, t_5\}$

$O_8 = \{t_8, t_3\}$

$M_8 = 1$

Therefore $T_8 + T_3 \leq 1 + T_{10} + T_5$	(1)	by R_1
$P_4 = T_3 - T_4 + 0$	(2)	by R_1
$T_5 \leq T_4$	(3)	by R_2
$P_4 \leq T_3 - T_5$	(4)	(2) & (3)
Similarly $P_{12} \leq T_8 - T_{10}$	(5)	
Therefore $P_4 + P_{12} \leq 1$	(6)	(4), (5) and (1)

From 6 we can conclude that either P_4 contains a stone or P_{12} or neither.

From our simulation rules we can conclude directly that T_4 and T_9 cannot be firing at the same time.

<u>Proof 2:</u>

$t_9\ p_{13}\ t_{10}\ p_{14}\ t_{11}\ p_7\ t_2\ p_3\ t_3\ p_4\ t_4$ is a simple path.

Therefore, $T_4 \leq T_9 + N$ $\quad R_3$

Therefore, $T_4 - T_9 \leq N$

i.e., No. of deposits minus no. of removals $\leq N$

Therefore, No overflow of buffer

Similarly:

$t_4\ p_5\ t_5\ p_6\ t_6\ p_9\ t_7\ p_{11}\ t_8\ p_{12}\ t_9$ forms a simple path.

Therefore, $T_9 \leq T_4$

Therefore, $T_4 - T_9 \geq 0$

Therefore, No. of deposits minus no. of removals ≥ 0

Therefore, No buffer underflow.

<u>Proof 3</u>:

For this particular problem it is easy to see that deadlock can occur only if $P_7 = P_9 = 0$ and there is no way to change this situation. (Pure cycles can be represented by the subscripts of the places only. The transitions can be left out because there is no ambiguity).

$S_1 = 1, 2, 3, 4, 5, 6, 7, 1$ is a pure cycle.

$S_2 = 10, 11, 12, 13, 14, 15, 10$ is a pure cycle.

$S_3 = 3, 4, 5, 6, 9, 11, 12, 13, 14, 7, 3$ is a pure cycle.

$N_{s_1} = 1$ (1)

$N_{s_2} = 1$ (2)

$N_{s_3} = N$ (3)

$a = P_3 + P_4 + P_5 + P_6 \leq 1$ **from (1)**

$b = P_{11} + P_{12} + P_{13} + P_{14} \leq 1$ **from (2)**

Therefore, $a + b \leq 2$

But if $N > 2$ and $P_7 = P_9 = 0$ then $a + b = N > 2$ **from (3)**

Therefore Contradiction.

Therefore, at no stage can both P_7 and P_9 be zero, for $N > 2$.

For N = 1, if $P_7 = P_9 = 0$ then

(a) one of p_3, p_4, p_5, p_6 contains a stone or (exclusive).

(b) one of p_{11}, p_{12}, p_{13}, p_{14} contains a stone.

Case a: t_6 will eventually fire causing a stone to be placed in P_7. Therefore, $P_7 + P_9 \neq 0$.

Similarly for N = 2 it can be shown that $P_7 + P_9 = 0$ cannot exist forever.

Therefore no deadlock is possible.

Example 3.4.

Consider the problem of two cars passing through a gate [13]. There is a button. Pressing the button causes the gate to open if it is closed and closed if it is open. The problem is to coordinate the activity so that both cars may pass through the gate irrespective of their times of arrival at the gate. The desired coordination is represented by the following net:

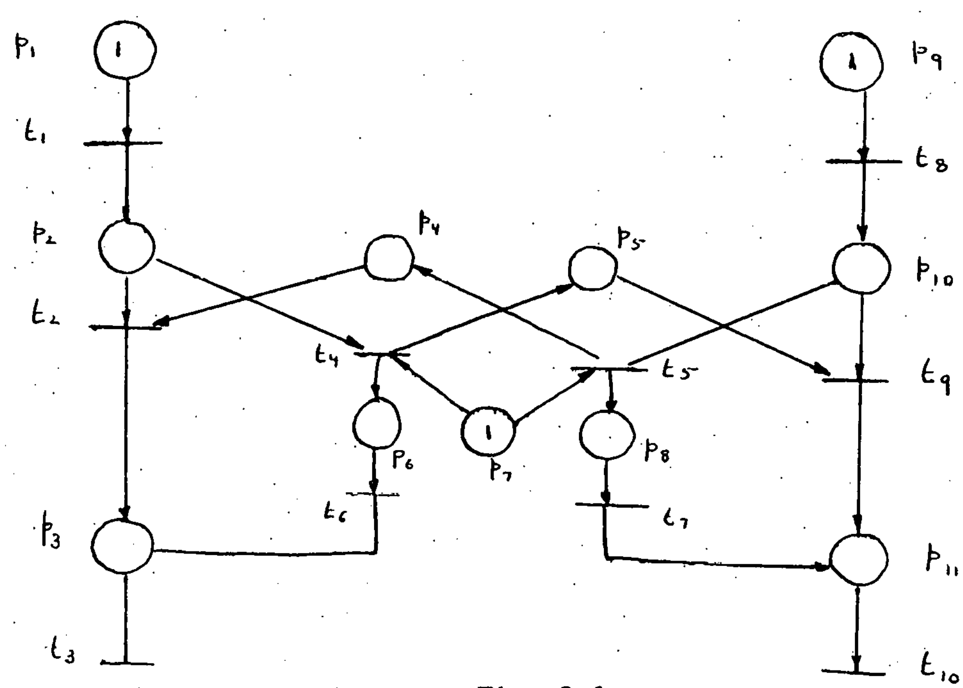


Fig. 3.6

t_1 : car A comes to gate

t_4 : car A presses button

t_3 : car A passes through gate

t_8 : car B comes to gate

t_5 : car B presses button

t_{10}: car B passes through gate.

Gate is initially closed.

We want to prove the following:

(1) t_4 fires $\Longrightarrow t_5$ does not

t_5 fires $\Longrightarrow t_4$ does not.

i.e., if car A presses the button then car B does not and vice versa.

(2) t_3 and t_{10} will both eventually fire irrespective of whether t_1 fires first or t_8 or both together. That is, both cars will eventually get through the gate irrespective of the order in which they arrive.

(3) Ultimately, $T_3 \leq 1$, $T_{10} \leq 1$

<u>Proof</u>:

(1) $T_4 + T_5 \leq 1$ from R_1 ......(1)

Therefore $T_4 = 1 \Rightarrow T_5 = 0$

$T_5 = 1 \Rightarrow T_4 = 0$

Therefore only one car presses the button.

(2) From R_2

$P_5 = T_4 - T_9$

$P_4 = T_5 - T_2$

Therefore $0 \leq P_5 + P_4 = T_4 + T_5 - (T_9 + T_2)$

$\leq 1 - (T_9 + T_2)$ from (1)

Therefore $T_9 + T_2 \leq 1$

Therefore either T_2 fires or T_9 fires.

t_1 and t_8 are live for the initial marking. If both fire, then $P_2 = P_{10} = 1$. At this stage t_4 and t_5 are enabled. $T_5 = 1 \Longrightarrow T_4 = 0$, t_2 can fire, t_9 cannot fire. t_3 is live. $T_5 = 1 \Longrightarrow t_{10}$ is live. Similarly $T_4 = 1 \Longrightarrow T_5 = 0 \wedge t_3$ is live $\wedge$ t_{10} is live. Therefore, for any simulation, before t_3 and t_{10} fire, they are live with respect to every intermediate marking.

(3) Follows directly from R_2.

The net has a slight problem in that t_3 and t_{10} may be firing at the same time, i.e., both cars may try to pass through the gate at the same time.

2. Method of Inductive Assertions

This method was introduced by Floyd [6] to prove the correctness of sequential programs. We will apply this method to prove that a Petri net is correct with respect to a particular, given assertion A. The basic ideas are taken from [10] where the technique is applied for proving parallel programs correct. The procedure is as follows: With each transition in the Petri net we associate an assertion. Our aim is to prove that every time a transition in a Petri net is enabled, the corresponding assertion is true irrespective of the particular simulation which caused this transition to be enabled and irrespective of the state of the rest of the net. Once this has been established, we will try to deduce that the Petri net is correct with respect to A. As we have already stated previously, A will be a statement about the flow of control in the net and not about the actual computation achieved. Thus the assertions at the transitions will in all probability be statements about the number of stones in a particular place or the number of times a particular transition has fired.

Definition 3.1. Let $N = \langle T, P, A, B° \rangle$ be a Petri net. An assertion α_i asserted with a transition $t_i \in T$ is a predicate on the values of P_k, T_k where $p_k \in P$ and $t_k \in T$. The Petri net N is correct with respect to the assertion α_i if and only if for each simulation of the net that enables t_i, α_i is true when t_i is enabled. The net N is correct with respect to a set of assertions if and only if it is correct with respect to each assertion in the set.

Induction Theorem:

To prove that a Petri net $N = \langle T, P, A, B° \rangle$ is correct with respect to a set of assertions $\{\alpha_i \mid t_i \in T\}$ it is sufficient to prove the following.

(1) α_i is true for all t_i that are enabled in B°.

(2) For each $t_i \in T$

Let $P_i = \{p \mid p \in I_i \wedge \langle p, o \rangle \in B°\}$

i.e., the set of all initially unmarked input places of t_i.

Let $P_i = \{q_1, q_2, \ldots\ldots q_n\}$

Let $T_j = \{t_k \mid q_j \in O_k\}$ $1 = j = n$

i.e., the set of all transitions of which q_j is an output place.

Let $B_i = \{\langle b_1, b_2, \ldots\ldots, b_n\rangle \mid t_{bj} \in T_j\}$

Each n-triple in B_i, gives the subscripts of the transitions which when fired will cause stones to be placed in the initially unmarked input places of t_i.

Let Fire $(b_1, \ldots\ldots b_n)$ denote the fact that the transitions $t_{b_1}, \ldots\ldots, t_{bn}$ fire.

Then for each $t_i \in T$

$$\alpha_{b_1} \wedge \alpha_{b_2} \ \ldots\ldots\ \alpha_{bn} \ \wedge \text{Fire}\ (b_1, \ldots\ldots, b_n) \Rightarrow \alpha_i \qquad (1)$$

for all $\langle b_1, \ldots\ldots, b_n\rangle \in B_i$.

The proof is similar to that presented by Lawer in [10] and will be omitted here.

Each equation of the form (1) is called a verification condition:

It should be obvious to the reader that what we are trying to prove in the second part of the induction theorem is really a very strong condition. It is sufficient but not necessary. The converse of the Induction theorem is not true in general. In a later paper we would like to get weaker verification conditions and our results here are just a first attempt. In general, the stronger the verification conditions, the less is the "information content" of the assertions. That is, if we have very strong conditions to verify, the assertions will tend to be of a trivial nature and we may not be able to conclude the main assertion A about the net.

Thus the method is

(1) Formulate the assertions for each transition.

(2) Prove that all assertions associated with transitions that are initially enabled are true.

(3) Prove all the pertinent verification conditions hold.

(4) Deduce that the net operates correctly with respect to the main, overall assertion.

Note:

(1) Floyd, when proposing this method for sequential programs showed that it was not necessary to have an assertion at every point. One assertion in each loop and one assertion at each termination point are sufficient. In the above formulation for Petri nets we have applied an assertion at every transition. Analogous to the procedure for sequential programs we do not think it is necessary to have an assertion at every transition. However, we have not as yet been able to find the concept in a Petri net that is equivalent to a loop in a sequential program and which fits into the framework of our induction theorem.

We will again give the simple producer-consumer problem and show that it is correct with respect to an assertion using the inductive assertion method.[1] However, to do this we have to introduce the concept of an augmented Petri net. A simulation S of a net N can be represented as follows:

$$S = B^{\circ} \{T_o\} M_1 \{T_1\} M_2 \{T_2\} \ldots\ldots\ldots M_i \ \{T_i\} \ldots\ldots.$$

B° is the initial marking.

M_i is some subsequent marking of N and T_i represents the set of transitions that finish firing at the same time starting from marking M_i and ending with marking M_{i+1}. Let $N = (P, T, A, B^{\circ})$ be a Petri net.

[1] The proof presented here is taken from [10] and modified to be applicable in the context of Petri nets.

Example: 3.5 (Producer - Consumer problem)

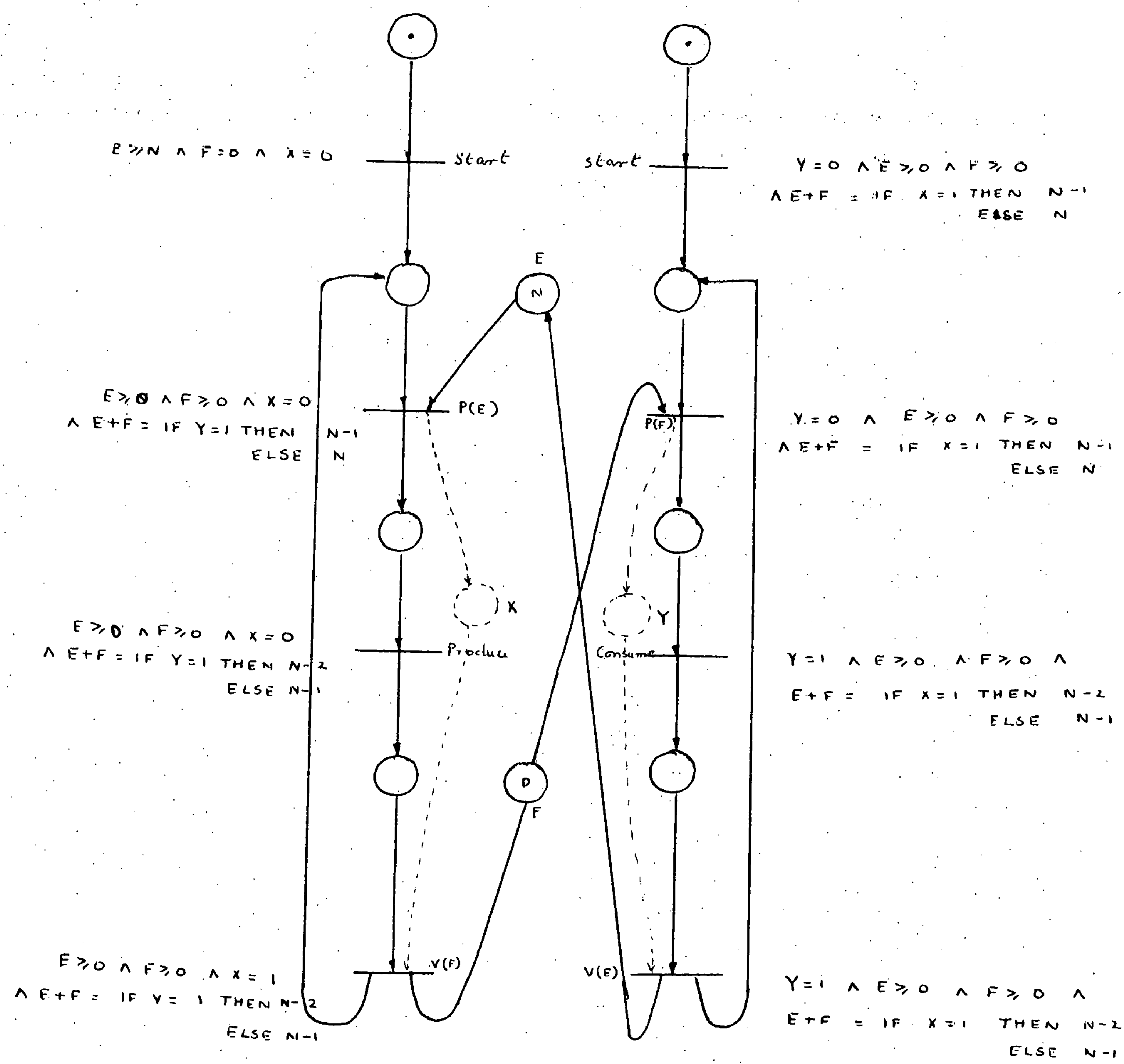


Fig. 3.7

Then, $N' = (P', T', A', B^{\circ}{}')$ is an augmentation of N if and only if $P \subset P'$, $T \subset T'$, $A \subset A'$, $B^{\circ} \subset B^{\circ}{}'$ and for each possible simulation S of N there exists a simulation S' of N' and vice versa such that $M_i \subset M_i'$, $\{T_i\} \subset \{T_i'\}$.

In a later report we will develop "local" conditions under which places, arcs and transitions can be added to a net N to form an augmentation N'. For the present we will only state the following:

Theorem 1. If $t_{a_1}\ p_{b_1}\ t_{a_2}\ p_{b_2}\ ....t_{ak}$ is any simple path through a net N and if a place P is added so that it is an output place of t_{a_1} and an input place of t_{ak} then the resulting net is an augmentation of the original net.

Proof: Obvious

Theorem 2. If N' is an augmentation of N then N' is correct with respect to α_i if and only if N is correct with respect to α_i.

Proof: Obvious

In Fig. 3.1 the solid lines represent the original Petri net for the problem and X and Y are places added to give the net N'. Theorem one guarantees that N' is an augmentation of N. The assertions at each transition are given, E, F, X, Y represent the number of stones in the respective places.

To prove N is correct with respect to

$E + F = N - 2 \wedge 0 \leq F \leq N$ ....A

If N' is correct with respect to $\{\alpha_1 .... \alpha_8\}$ and we can deduce A from $\{\alpha_1, ... \alpha_8\}$ then N' is correct with respect to A and Theorem 2 guarantees that N is correct with respect to A.

To prove that N' is correct with respect to $\{\alpha_1 ,\ldots\ldots\alpha_8\}$:

(1) The only transitions which are initially enabled are t_1 and t_5. α_1 and α_5 are both true trivially.

(2) We will only prove the verification conditions for transition 2. The rest is left for the reader.

(a) $\alpha_1 \wedge t_1$ fires $\Rightarrow \alpha_2$

$E = N \wedge F = 0 \wedge X = 0 \wedge t_1$ fires $\Rightarrow$

$E \geq 0 \wedge F \geq 0 \wedge X = 0 \wedge E + F =$ if $Y = 1$ then $N - 1$ else N.

Proof.

$E = N \Rightarrow E \geq 0$

$F = 0 \Rightarrow F \geq 0$

$X = 0 \Rightarrow X = 0$

To prove that $E + F =$ if $Y = 1$ then $N-1$ else N

$X = 0$ and examination of the net indicates that

$Y = 1$ at t_7 and t_8. $\quad X = 0 \wedge \alpha_7 \Rightarrow E + F = N - 1$

$X = 0 \wedge \alpha_8 \Rightarrow E + F + N - 1$

Similarly $Y = 0$ at t_5 and t_6 and

$X = 0 \wedge \alpha_5 \Rightarrow E + F = N$

$X = 0 \wedge \alpha_6 \Rightarrow E + F = N$

Therefore, $X = 0 \Rightarrow$ if $Y = 1$ then $N - 1$ else N.

Q.E.D.

(b) $\alpha_4 \wedge t_4$ fires $\Rightarrow \alpha_2$

i.e.

$E \geq 0 \wedge F \geq 0 \wedge X = 1 \wedge E + F =$ if $Y = 1$ Then $N - 2$ else $N - 1$

$\wedge\ t_4$ fires $\Rightarrow$

$E \geq 0 \wedge F \geq 0 \wedge X = 0 \wedge E + F =$ if $Y = 1$ then $N - 1$, else N.

<u>Proof</u>:

t_4 fires $\Rightarrow$ $X \leftarrow X - 1 \wedge F \leftarrow F + 1$

Therefore, $X = 1 \wedge t_4$ fires $\Rightarrow$ $X = 0$

$E + F =$ if $Y = 1$ then $N - 2$ else $N - 1 \wedge t_4$ fires

$\Rightarrow$ $E + F =$ if $Y = 1$ then $N - 1$ else N

Q.E.D.

Therefore, N' is correct with respect to α_2.

Similarly, N' is correct with respect to $\{\alpha_1, \alpha_2, \ldots\ldots, \alpha_8\}$

Therefore by Theorem 2 N is correct with respect to $_1, \ldots\ldots\ _8$.

From the assertions we can conclude

$E + F \geq N - 2$ (1)

$E + F \leq N$ (2)

$E \geq 0$ (3)

Therefore, $F \leq N$ (4) from (2) and (3)

$F \geq 0$ (5)

Therefore, N is correct with respect to A.

Also, from (1) conclude: no deadlock for $N > 2$

From (4) conclude: no buffer overflow.

From (5) conclude: no buffer underflow.

CHAPTER IV

Conclusions

In this report we presented a general discussion of Petri nets. We demonstrated that Petri nets were being used in the specification, design and evaluation of complex computer systems, thus establishing the need for a study of the capabilities of Petri nets and proofs of their correctness. In Chapter II we showed how Petri nets could be modified so as to obtain different classes of representable coordinations. In Chapter III we first discussed what was meant by the statement "Petri net N is correct" and then established the feasibility of using the methods of Computational Induction and Inductive Assertions to prove that "Petri net N is correct with respect to assertion A".

In a subsequent report we will further examine some of the ideas introduced here.

ACKNOWLEDGEMENTS

I am grateful to my advisor Professor M.J. Flynn for his continued guidance, encouragement and support. I would also like to thank my colleague, Joe Davison, for his helpful comments.

REFERENCES

1. BAER, J.L. A Survey of Multiprocessing. Technical Report No. 72-05-01. Computer Science Group, University of Washington, Seattle, Washington 98195, May 1972.
2. DENNIS, J.B. Modular, Asynchronous Control Structures for a High Performance Processor. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation. ACM, New York, 1970, pp. 55-80.
3. DENNIS, J.B. Computation Structures. Project MAC Progress Report VIII July 1970-July 1971. MIT, July 1971.
4. DENNIS, J.B. Concurrency in Softwear Systems. Advanced Course in Softwear Engineering, Lecture Notes in Economics and Mathematical Systems. Ed. M. Beckmann et. al. June 1972, pp. 111-127.
5. DIJKSTRA, E.W. Cooperating Sequential Processes. Programming Languages. Ed. F. Genuys. Academic Press, New York, 1968, pp. 43-112.
6. FLOYD, R.W. Assigning Meanings to Programs. Proceedings of a Symposium in Applied Mathematics Vol. 19, Mathematical Aspects of Computer Science, American Mathematical Society. 1967, pp. 19-32.
7. GENRICH, H. Einfache Nicht-sequentielle Prozesse. Doctoral Dissertation. Gesellschaft für Mathematik und Datenverarbeitung, 5201 Birlinghoven.
8. HOLT, A.W. and COMMONER, F. "Events and Conditions". Record of the Project MAC Conference on Concurrent Systems and Parallel Computation. Association for Computing Machinery, Ney York, 1970, pp. 3-52.

9. KOSARAJU, S. RAO. Limitations of Dijkstra's Semaphore Primitives and Petri Nets. Hopkins Computer Research Reports No. 25, Research Program in Computer Systems Architecture. The Johns Hopkins University, May 1973.

10. LAWER, H.C. Correctness in Operating Systems. Ph.D. Thesis, Carnegie-Mellon University, September 1972. AFOSR-TR-72-2361 Contract F44620-70-C-0107.

11. MILLS, H.D. Mathematical Foundations for Structured Programming. FSC 62-6012. Federal Systems Division, IBM Corp., Gaithersburg, Maryland, 1972.

12. NOE, J.D. A Petri Net Model of the CDC 6400. Proceedings of the ACM/SIGOPS Workshop on Systems Performance Evaluation. April 1971, pp. 362-378.

13. PATIL, S.S. "Coordination of Asynchronous Events." Ph.D. Thesis, E.E. Dept., Project MAC, MIT. MAC TR-72, June 1970, AD-711-763.

14. PATIL, S.S. "Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes". Project MAC, Computational Structures Group Memo, 57. February 1971, pp. 1-18.

15. PETRI, C.A. Communication with Automata. Supplement 1 to Technical Report RADC-TR-65-377, Vol. 1, Griffiss Air Force Base, New York, 1966.

16. SEITZ, C.L. Asynchronous Machines Exhibiting Concurrency. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation. ACM, New York, 1970, pp. 93-106.

17. SHAPIRO, R.M. and SAINT, H. A New Approach to Optimization of Sequencing Decisions. Annual Review in Automatic Programming, Vol. 6, Part 5, 1970.