Tutorial on Petri Nets

Gregory J. D'Angelo
Bell Laboratories
Crawfords Corner Road
Holmdel, N. J. 07733

1. INTRODUCTION

Petri nets are a flowcharting technique used to model asynchronous and concurrent processes. There have been many articles written on them and various other modifications of the Petri net language. This paper cannot describe all aspects of Petri nets and latter spinoffs in detail, but is intended to give a general overview of their functionality and uses.

2. OVERVIEW [5]

2.1 Elements of Petri Nets

Petri nets are composed of four symbols: circles, bars, arcs, and dots. A circle represents a place which models a condition in the graph. The bars are transitions which are actions that occur and the arcs are directional connections between places and transitions. A place can only connect to a transition, and a transition can only connect to a place. This rule reduces the Petri net into a bipartite directed graph. The dots are called tokens and are used to control the flow of the net. The tokens reside in places and are moved through the net by a series of rules similar to a game.

Petri nets have the ability to hierarchically model a problem. One could look at a problem globally by having places and transitions represent a series or a group of actions. When the problem must be investigated further, the places and transitions can be replaced by subnets of greater detail.

2.2 Rules for Petri Nets

If node "i" is connected to node "j" by an arc, then node "i" is considered to be an input to node "j" and node "j" is considered to be an output of node "i". The tokens of the Petri net are moved by the action of a transition firing. A transition can fire when it becomes enabled, by having all of its input places occupied by a token. When the transition fires, a token from each of the input places is removed, and new tokens are created and one is placed in each of its output places.

Figure 1 shows a Petri net with an initial marking. In this example, "t2" is enabled since its input place has a token in it. Figure 2 shows the resulting marking after "t2" fires. After "t2" fired, transactions "t1", "t3", and "t5" are all enabled. In this condition, transactions "t3" and "t5" are in conflict, that is if one of them fires, it will disable the other. Figures 3 - 5 show the resulting markings of the Petri net from each of the transitions firing.

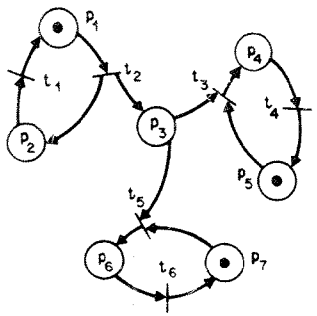


FIGURE 1 A MARKED PETRI NET

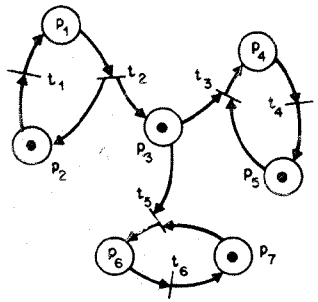


FIGURE 2 THE MARKING RESULTING FROM FIRING TRANSACTION "t2" IN FIGURE 1. NOTE THAT THE TOKEN IN "p1" WAS REMOVED AND TOKENS WERE ADDED TO "p2" AND "p3"

MARKINGS RESULTING FROM THE FIRING OF DIFFERENT TRANSITIONS IN THE NET OF FIGURE 2.

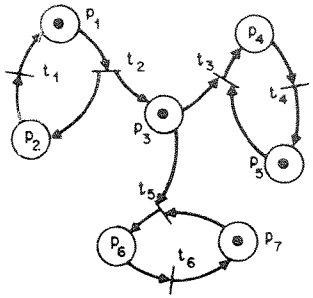


FIGURE 3 RESULT OF FIRING TRANSITION "t1"

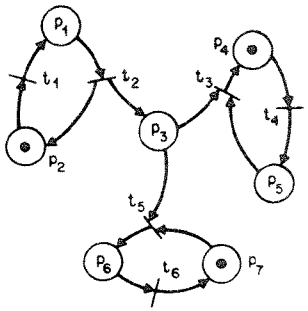


FIGURE 4 RESULT OF FIRING TRANSITION "t3"

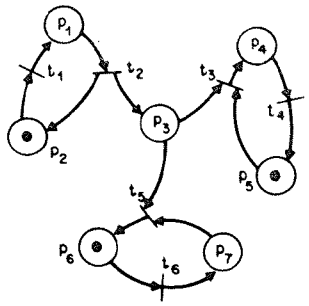


FIGURE 5 RESULT OF FIRING TRANSITION "t5"

The Petri net language has no innate way of resolving a conflict situation; this problem must be resolved by the user in some higher level. In its standard form, there is no passage of time in Petri nets, all actions occur instantaneously, i.e. take zero time. These two properties of Petri nets add to their flexibility since they do not impose any predetermined restrictions on the method of solving conflicts or accounting for the passage of time.

One extension to the Petri net language is the use of the inhibitor arc. This extension models what is known as zero-testing. An arc from place "p" to transition "t" will allow the transition to fire only if there are zero tokens in place "p". This arc is designated in the net by using a small circle at the tip of the arc instead of an arrow head. The inhibitor arc is used to give priority to one transaction over another, or to perform a task if a condition is false, i.e. there are zero tokens in its input place. An example of the use of the inhibitor arc in giving priority is shown in figure 6. In this example, transition "c1" has priority over "c2", because if places "b1" and "b2" both have tokens in them, only transaction "c1" is enabled, since transaction "c2" is only enabled if there is a token in "b2" and no tokens in "b1".

[6] The use of the inhibitor arc tends to make the net less readable since the reader must be familiar with the meaning and use of this arc. It is possible, however, to model the action of the inhibitor arc using the basic Petri net language, as shown in figure 7. The addition of place "c1′" and its associated arcs replace the inhibitor arc. When transition "t0" fires, it removes the token from "c1′" thereby preventing "t2" from firing. When transition "t1" fires, the token removed from "c1′" is replaced, thereby allowing "t2" to fire when it becomes enabled. When "t2" fires, the token is removed from "c1′" and is replaced when "t2" finishes firing.

2.3 Markings of a Petri Net

A Petri net with tokens is said to be a marked Petri net. There are different names for Petri nets that possess unique qualities in the handling of tokens. If the Petri net is so constructed that there can be at most "k" tokens in a single place at one time, then the net is said to be "k-bounded". If the exact value of "k" is unknown, but is known to be some finite number, then the net is just referred to as being "bounded". In the special case where "k" is equal to one, the net is referred to as being a "safe" net. When the tokens in the net represent a resource, which cannot be created nor destroyed, the total number of tokens in the net must remain constant. This particular type of net is referred to as a "conservative" net.

Transitions can be in any one of three states: "dead", "potentially firable", or "live". A transition is "dead" if there exists no sequence of firings, from some initial marking, which will enable the transition. If

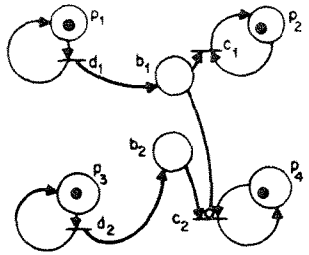


FIGURE 6 USE OF THE INHIBITOR ARC

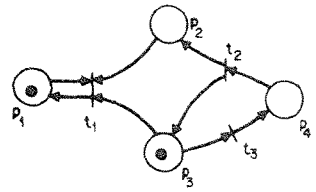


FIGURE 8 A PETRI NET WITH MARKING (1,0,1,0) AND INFINITE REACHABLE STATE-SPACE

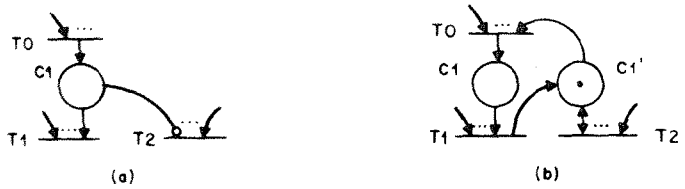


FIGURE 7 REPLACING AN INHIBITOR ARC FROM A 1-BOUNDED PLACE AS SHOWN IN (a) AND WITH ORDINARY PETRI NET GRAPH CONSTRUCTS AS SHOWN IN (b). [THE ARC ⟷ IS SHORTHAND FOR ⇄ , MEANING THAT FIRING OF A TRANSITION RESTORES A TOKEN TO THE PLACE WHERE IT WAS BEFORE THE FIRING]

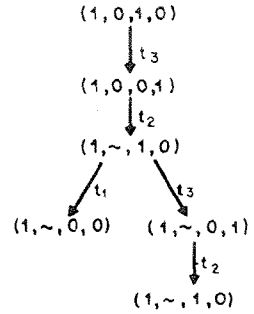


FIGURE 9 THE REACHABILITY TREE OF THE PETRI NET OF FIGURE 8

there exists a sequence of firings which can enable a transition, then it is "potentially firable". A transition is "live", if for every possible marking that can result from some initial marking, the transition is always enabled.

One method of understanding the flow of a Petri net is by the use of the "reachability tree". This tree shows all states a particular net can be in, resulting from some initial marking. It is composed of a root which is the initial marking and the nodes which are the resulting markings from transitions firing. Often the condition exists where a transition fires repeatedly and keeps increasing the number of tokens in a place but leaves the marking of the rest of the net unchanged. When the number of tokens in this place becomes "too large", the set of nodes which represent the sequence of firings are reduced to one node, and a special symbol is used to designate the number of tokens. The symbol "~" is used to represent this condition *. The symbol "~" must represent a value which is arbitrarily large, and the operations of addition, subtraction, and comparison are interpreted as:

$$\sim + a = \sim$$
$$\sim - a = \sim$$
$$a < \sim$$

for any natural number "a". Therefore ~ can be thought of as representing infinity.

Figures 8 and 9 show an example of a Petri net and its corresponding reachability tree. The marking (1,0,1,0) is chosen as the initial marking or root. A "1" represents a token and a "0" represents an empty place, and the order of the 1's and 0's correspond to the place numbers. Transition "t3" is enabled and fires to give the marking (1,0,0,1). After "t2" fires, the marking (1,1,1,0) results, and we now have two transitions enabled, "t1" and "t3". Firing "t1" results in (1,1,0,0), and firing "t3" generates (1,1,0,1). Transition "t3" has now fired twice since the initial marking and has resulted in two different markings. The second marking (1,1,0,1) is greater than the first (1,0,0,1) and therefore the second component is replaced with "~", which means that the series of firings, "t3 t2", can continue as many times as desired to make the value of the second component as large as desired. Since (1,1,0,1) was replaced by (1,~,0,1) all of the resulting markings from this marking contain a "~" in position two, which changes the earlier markings into (1,~,0,0) and (1,~,0,1). The first of these two markings has no successor since only two of the three places needed to enable the transition contain tokens. The second marking will result in (1,~,1,0) which previously existed in the tree, and therefore, the entire reachability is shown in figure 9.

---

* In [5] a lower case omega is used.

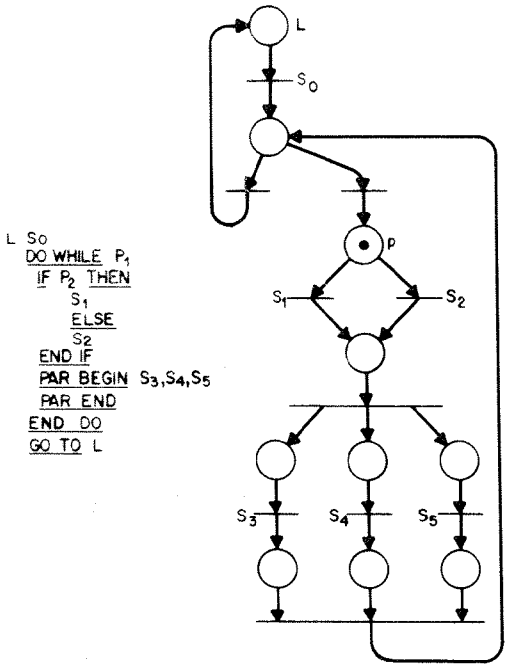


FIGURE 10 EXAMPLE OF A PETRI NET USED TO REPRESENT THE FLOW OF CONTROL IN PROGRAMS CONTAINING CERTAIN KINDS OF CONSTRUCTS

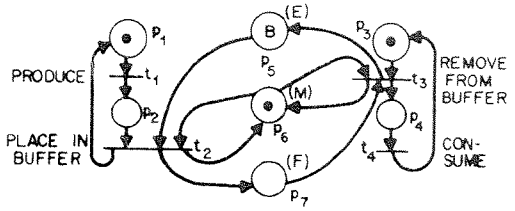


FIGURE 11 A COMPACT NET FOR THE TWO-PROCESS SYSTEM

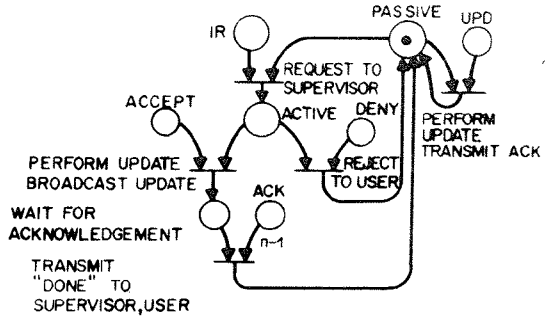


FIGURE 12 PETRI NET REPRESENTING THE LOCAL CONTROLLER IN A DISTRIBUTED SYSTEM

3. USES OF PETRI NETS

The aforementioned properties of Petri nets make them a viable tool for use in the modeling of computer systems. They have applications in software design, as well as hardware.

3.1 Modeling of Software [1]

Petri nets can give the user a graphical representation of the code he wishes to model. Program structures such as: IF-THEN-ELSE, DO-WHILE, GOTO, and PARBEGIN-PAREND can all be easily represented in Petri nets. All these structures are shown in the example of figure 10. As was mentioned earlier, the Petri net will not be able to show which path will be taken in a decision situation, but is intended to show the structure of the code.

One application that is easily modeled by Petri nets is P and V operations on semaphores. P and V operations can be described as follows:

```
P(S) : if S>0 then S = S-1 else
       wait
V(S) : S = S+1
```

The waiting process can be scheduled at some later time when S>0. At this time S is decremented and the process continues. An example of this is shown in figure 11 as a producer-consumer system. In this system, the producer is an input process or device that received data and places it in a buffer of fixed sized "B". The consumer is a computing process or device which will retrieve the data from the buffer and operates on it. There are three problems which must be prevented from occurring in such a process. They are:

1. Buffer overflow
2. Buffer underflow
3. Both processes accessing the buffer simultaneously

Positions "E" and "F" represent the number of empty and full buffer positions respectively, and "M" is used for mutual exclusion. Since in the initial marking, "B" was some fixed amount, buffer overflow and underflow are prevented because there would not be a token available to allow the transition (which would cause overflow or underflow) to fire.

Another application of Petri nets is in the study of protocols. Figure 12 represents a local controller in a distributed system where each of "n" locations must keep an exact copy of a data base. An update request is noted by a token appearing in place "IR", and it is accepted or rejected by the supervisor by a token being placed in "ACCEPT" or "DENY". If the request is accepted, the broadcast update causes a token to appear in place "UPD" in each of the other controllers nets. When each controller performs the update, an acknowledgement is sent to the updating controller and a token is added to place "ACK". Once there are n-1 tokens in "ACK", a "DONE" command is sent to the supervisor and to the updating controller. The "ACTIVE" and "PASSIVE" places prevent two requests from being processed at

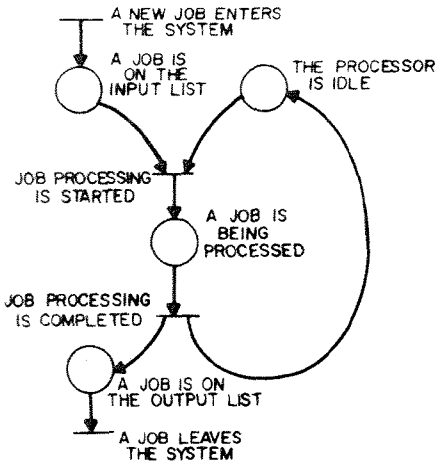


FIGURE 13 MODELING OF A SIMPLE COMPUTER SYSTEM

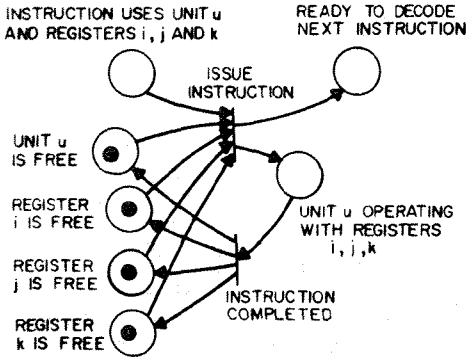


FIGURE 14 A PORTION OF A PETRI NET MODELING A CONTROL UNIT FOR A COMPUTER WITH MULTIPLE REGISTERS AND MULTIPLE FUNCTIONAL UNITS

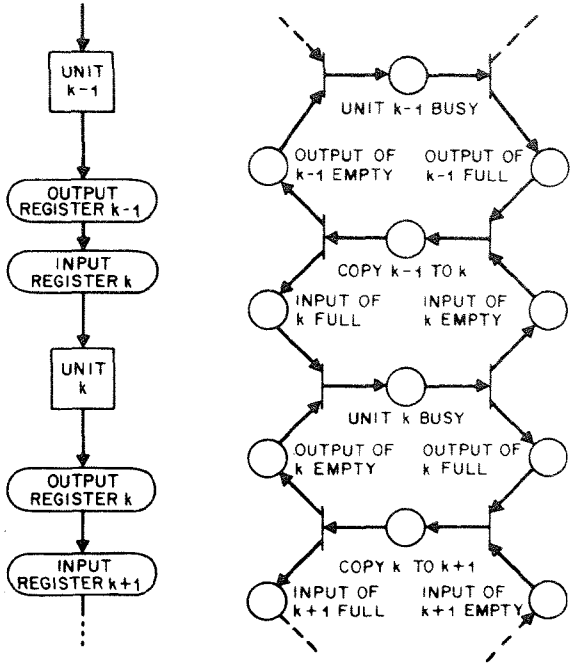


FIGURE 15 REPRESENTATION OF AN ASYNCHRONOUS PIPELINED CONTROL UNIT. THE BLOCK DIAGRAM ON THE LEFT IS MODELED BY THE PETRI NET ON THE RIGHT

the same time. A Petri net methodology was used in [2] to study a telephone signaling protocol.

3.2 Modeling of Hardware [5]

Besides their ability to model software, Petri nets can also model the actions of hardware. A simplified example of this is shown in figure 13. In this example, jobs enter the system and are queued in the input list. When the processor is idle, the job will be executed by the processor, and any new jobs entering the system will have to wait for the processor to become idle. When the job is completed, it leaves the system, and the processor becomes idle and waits for another job.

Petri nets can also model hardware at a much lower level. In figure 14, they are used to show the interaction of a functional unit and the three registers it must use to execute an instruction. When an instruction that uses unit "u" is issued, unit "u" and registers "i", "j", and "k" must all be free in order for the instruction to execute. When the instruction executes, the unit and its registers are utilized, and are not available for any other use. Once the instruction is complete, the unit and the registers are freed and another instruction can be executed.

Another hardware aspect that can be modeled in Petri nets is the use of a pipeline. The pipeline is composed of a number of stages that can execute simultaneously. It operates by having each stage perform a certain task, and then passing on the results to another stage. Before the results can be passed on, the next stage in the pipeline must be free to accept new data. The Petri net in figure 15 models this type of operation. The controller of the pipeline must know when each of the following conditions occurs:

- input register full
- input register empty
- output register full
- output register empty
- unit busy
- unit idle
- copying taking place

As the six previous examples have shown, Petri nets can provide a good graphical picture of the execution process of software as well as hardware.

4. OTHER RELATED NETS

4.1 Token Machine [2]

A token machine is a state machine of a Petri net, and is a graphical representation of the reachability tree. It shows all possible states a Petri net can be in, from some initial marking. The markings at each state are shown in the graph by listing the place numbers with tokens in them in a circle. The circles are connected by an arc which is labeled with the transition name which can fire from that marking. Figures 16 and 17 show a Petri net with its corresponding token

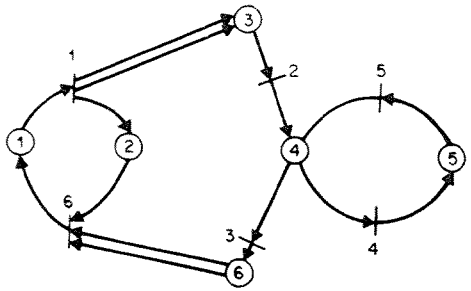


FIGURE 16 PETRI NET

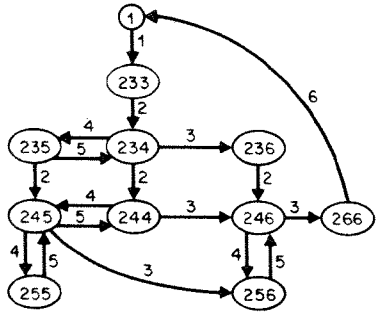


FIGURE 17 TOKEN MACHINE FOR THE PETRI NET OF FIGURE 16

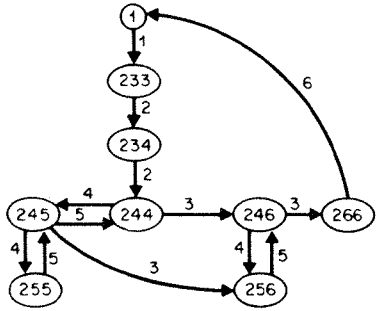
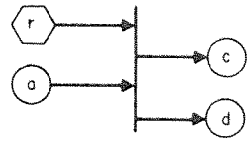


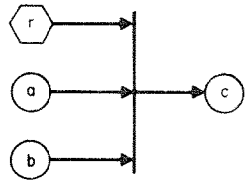
FIGURE 18 TOKEN MACHINE FOR THE TIME PETRI NET OF FIGURE 16
IN WHICH t**2 < t*3 AND t**2 < t*4



X TRANSITION

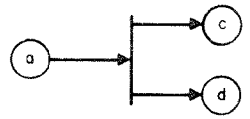
X (r,a,c,d): (0,1,0,0) → (e,0,1,0)
(0,1,0,1) → (e,0,1,1)
(1,1,0,0) → (e,0,0,1)
(1,1,1,0) → (e,0,1,1)

NOTE: "e" DENOTES "0" IF r IS AN INNER LOCATION; DENOTE "ø" (UNDEFINED) IF r IS A PERIPHERAL LOCATION.



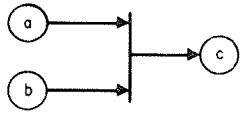
Y TRANSITION

Y (r,a,b,c): (0,1,1,0) → (e,0,1,1)
(0,1,0,0) → (e,0,0,1)
(0,0,1,0) → (e,0,0,1)
(1,1,1,0) → (e,1,0,1)
(1,1,0,0) → (e,0,0,1)
(1,0,1,0) → (e,0,0,1)



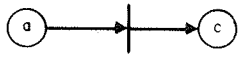
F TRANSITION

F (a,c,d): (1,0,0) → (0,1,1)



J TRANSITION

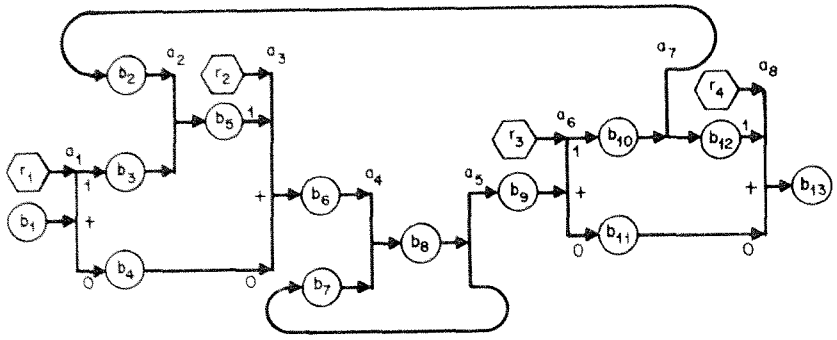
J (a,b,c): (1,1,0) → (0,1,1)



T TRANSITION

T (a,c): (1,0) → (0,1)

FIGURE 19 THE FIVE PRIMITIVE E-NET TRANSITIONS



b_1: JOB READY TO ENTER MIX
b_2: TAPE DRIVE IS AVAILABLE
b_3: JOB REQUIRES TAPE DRIVE
b_4: JOB DOES NOT REQUIRE TAPE DRIVE
b_5: TAPE JOB HAS DRIVE ALLOCATED
b_6: JOB REQUESTING CP
b_7: CP IS IDLE
b_8: CP IS BUSY
b_9: JOB IS THROUGH WITH CP
b_{10}: TAPE JOB READY TO RELEASE DRIVE
b_{11}: NON-TAPE JOB READY TO VACATE
b_{12}: TAPE JOB READY TO VACATE
b_{13}: JOB IS COMPLETE
r_1: ROUTES TAPE JOB TO b_3; NON-TAPE TO b_4
r_2: CHOOSES JOB FROM b_5 OR b_4
r_3: ROUTES TAPE JOB TO b_{10}; NON-TAPE TO b_{11}
r_4: CHOOSES JOB FROM b_{12} OR b_{11}

FIGURE 20 GRAPH OF EVALUATION NET

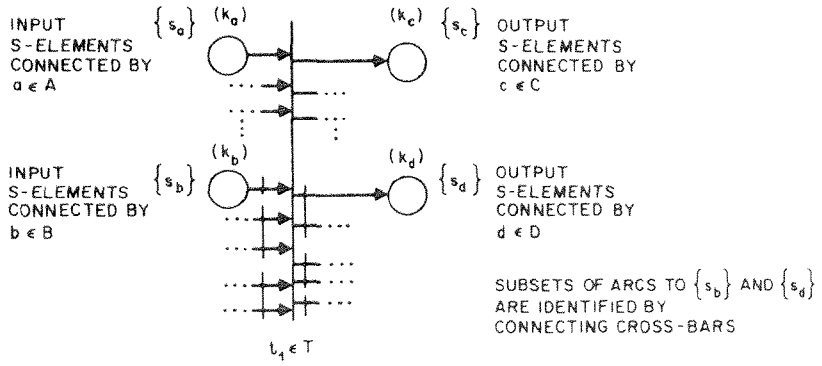


FIGURE 21 A PRIMITIVE PRO-NET ELEMENT

machine. The token machine is more powerful than the reachability tree since if there are two nodes with the same marking, the tree will have the two nodes in different places, whereas the token machine will reduce the two nodes into one. This displaying of the firing cycles may bring out some otherwise hidden aspects of the Petri net that is under study. One of these hidden aspects that comes to mind is the possibility of reaching a deadlock state. This means that there may a exist a sequence of firings that will result in a marking that will not enable any transitions. This condition will be shown by a node with an input and no output.

4.2 Time Petri Nets [2]

Time Petri nets were created to study timing aspects of Petri net models. They retain all aspects of a standard Petri net but contain the following additions. Each transition has associated with it two times, a minimum and maximum wait time. The first is the minimum time that a transition must wait to fire after being enabled, and the second is the maximum time the transition can possible wait to fire. The following notation is used to designate these times:

1. t*i denotes the minimal time for transition "i".
2. t**i denotes the maximal time for transition "i".
3. When t*i = t**i, the timing for transition "i" is denoted as ti.

In the special case where t*i = 0 and t**i approach infinity, the values of t*i and t**i are not shown in the net, and are understood to be zero and undefined respectively.

The use of the timing aspects of a time Petri net is usually used to prevent some of the transitions in the corresponding token machine from firing. For example, in the Petri net of figure 15 if we convert it into a token machine where t**2 < t*3 and t**2 < t*4, then in state 234 of figure 16 transition 2 will always fire before transitions 3 or 4. This will prevent states 235 and 236 from ever being reached and the token machine reduces to the one shown in figure 18.

4.3 Evaluation Nets [4]

Evaluation nets are similar to Petri nets except that they are more restrictive in their execution. One of the basic restrictions is that the output place of a transition must be empty in order for it to fire. There are five primitives each with their own set of firing conditions which are based upon certain definitions. A new symbol is introduced in Evaluation nets, the resolution location, designated by a hexagon. A location can be either an inner location, containing an input and an output, or a peripheral location, being a sink or a source. Inner locations can be either full, i.e. containing a token, or empty. A peripheral location can be undefined, when it is not known if it contains a token. Figure 19 shows the five primitives along with their

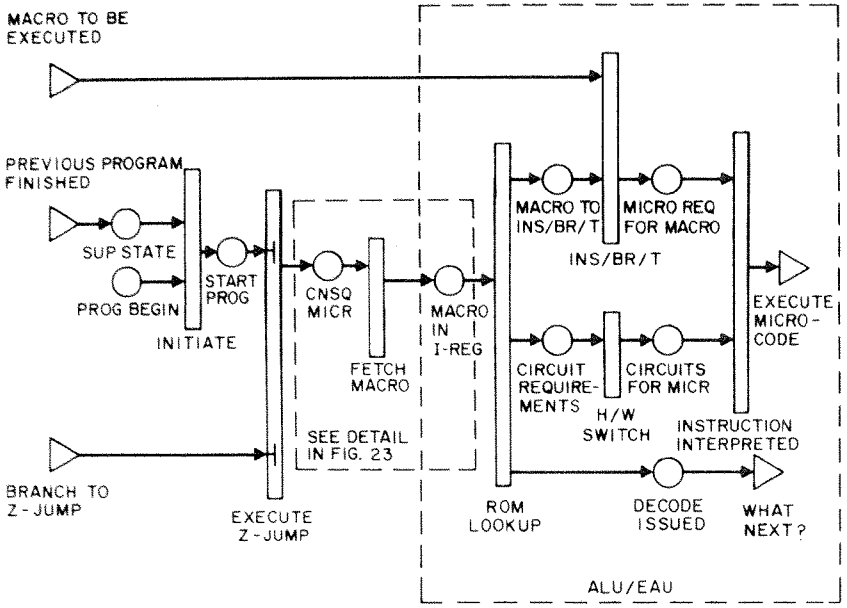


FIGURE 22 PRO-NET EXAMPLE: ACTION OF ALU/EFFECTIVE ADDRESS UNIT

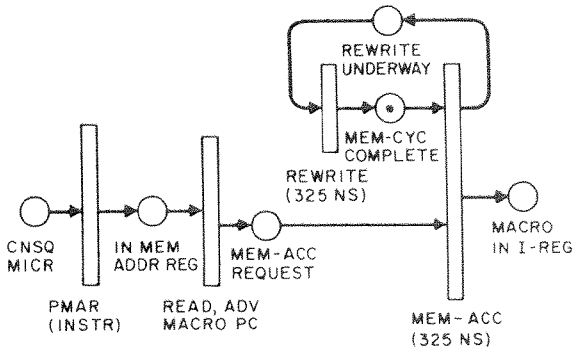


FIGURE 23 PRO-NET EXAMPLE: MEMORY FUNCTIONAL UNIT

possible firing conditions.

In the "X" Transition, the resolution location acts as a switch in that if "a" is full and "r" and "c" are empty, "a" will fire into "c" regardless of the condition of "d". In the opposite condition, if "a" and "r" are full and "d" is empty, "a" will fire into "d" regardless of the condition of "c". In the "Y" Transition, "r" is used as a tie breaker. If "a" or "b" is full, "r" can be either full or empty and the transition will fire. The resolution location is only used in the case where both "a" and "b" are full. When this occurs, if "r" is empty, "a" yields a token, and if "r" is full, "b" yields a token. The "F", "J", and "T" Transitions, operate whenever all of the input locations contain a token, and the output locations are empty. Since the use of the primitives must be strictly adhered to, if a condition exists which does not follow one of the primitives, the user must combine primitives together to model this condition.

An additional feature that Evaluation nets possess is the ability of the tokens to contain attributes. Since Evaluation nets are safe, the tokens can be given names, and values. This type of feature can be helpful when one is doing a simulation and needs to keep track of certain attributes of the objects that are flowing through the model. Figure 20 shows a simplified computer system modeled with Evaluation nets.

4.4 Pro-Nets [3]

Pro-Nets are similar to Petri nets but they have a series of other attributes associated with them. There are two definitions which must be specified:

1. *T-element* - is a transition and is displayed as a line segment if it represents a primitive, and is displayed as a rectangle otherwise.

2. *S-element* - is a place and may contain zero or more tokens and may contain multiple inputs and outputs.

These attributes are best shown by referring to figure 21 and reading the rules below *:

a. the T-element is enabled if all of the following hold:

   1. a token exists on each of the "AND" inputs in set {Sa}.

   2. a token exists on every S-element place in at least one of the subsets of the "selection" inputs in {Sb} (denoted by small bars across the arcs; each spans a subset).

---

* The following rules were taken directly from the reference.

3. the number of tokens on each of the "AND" outputs in set {Sc} is less than the bound.

4. the token count on each of the *selected* output subsets in {Sd} is less than the bound.

5. state variables that have been declared to control initiation are found to have permissive values.

b. After enabling, the action of a Pro-Net-element will begin at a time t, such that d_min <= t <= d_max where 0 <= d_min <= d_max. This feature is adapted from the work of P. M. Merlin. When action is initiated, it will endure for time t, t >= 0, and during that period, required tokens will be marked "reserved".

c. At the conclusion of the action, a token will have been:

1. removed from each S-element in {Sa}.

2. removed from each S-element in the selected subset in {Sb}.

3. added to each S-element in {Sc}.

4. added to each S-element in the selected subset in {Sd}.

As in Evaluation nets, tokens in Pro-Nets can have attributes associated with them. An example of a Pro-Net is shown in figures 22 - 23. As was mentioned earlier, the transitions are shown as rectangles since they may be expanded into greater detail.

Pro-Nets can be treated as Petri nets if the following restrictions are implemented:

1. the selector arcs are not used.

2. bounds on the amount of tokens in a place is infinity.

3. there are no transition procedures, global variables, or token attributes.

4. the elapsed firing time is instantaneous.

5. the firing bounds (d_min d_max) are not specified.

5. ACKNOWLEDGEMENT

The author gratefully wishes to acknowledge Dr. Julian Scher of the Department of Computer and Information Science at New Jersey Institute of Technology for his aid in locating references for this paper.

REFERENCES

1. Agerwala, T., *Putting Petri Nets to Work*, Computer, IEEE, December 1979.

2. Merlin, P. M., *A Methodology for the Design and Implementation of Communication Protocols*, IEEE Transactions on Communications, Volume COM-24, Number 6 June 1976.

3. Noe, J. D., *Nets in Modeling and Simulation* as found in *Lecture Notes in Computer Science*, Volume 84, "Net Theory and Applications", Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979.

4. Nutt, G. J., *Evaluation Nets for Computer System Performance Analysis*, Fall Joint Computer Conference, 1972.

5. Peterson, J. L., *Petri Nets*, Computing Surveys, Volume 9, Number 3, September 1977.

6. Torn, A. A., *Simulation Graphs: A General Tool for Modeling Simulation Designs*, Simulation, Volume 37, Number 6, December 1981.