MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 57

Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes

Suhas S. Patil

PATIL

February 1971

Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes

Suhas S. Patil

Project MAC, M.I.T., Cambridge, Massachusetts

ABSTRACT: This paper presents an example from a class of coordinations which cannot be expressed with the P and V primitives of Dijkstra without the help of conditional statements. Then the paper shows how the class of coordinations corresponding to Petri nets can be expressed in terms of processes with P and V primitives and conditional statements. One important property of these processes is that they involve a fixed amount of computation for any given coordination. At the end the paper suggests a generalization of the primitives.

Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes

Suhas S. Patil
Project MAC, M.I.T., Cambridge, Massachusetts

In multiprocessing, it is necessary to provide a means of interprocess communication so that a cluster of processes whose actions interact or could interfere may cooperate among themselves for smooth operation. Semaphores and the P and V primitives were introduced by Dijkstra [1] for just this purpose. These primitives have been very popular and have been used, sometimes in a modified form, in many multiprocessing systems. Yet there has not been much theoretical study to understand their nature, limitations, or capabilities. Such a study is the subject of this paper. In contrast to the lack of study of these primitives, much work has been done by Holt and others in understanding Petri nets [2,3], an abstract formalism for representing and studying asynchronous concurrent systems. In this paper we will relate the P and V primitives to Petri nets. The study will bring out the limitation and capabilities of these primitives and suggest a generalization of the P and V primitives of Dijkstra.

We will begin our study of the P and V primitives with the following example called "The Cigarette Smokers Problem." Before proceeding ahead it may be recalled that a sempahore is a variable whose value can only be non-negative. Instruction P[S] decrements the value of semaphore S by 1 and if the value of the semaphore is 0, the execution of the instruction is held up until it becomes 1 or larger. This provides a way to allow a process to wait until other processes catch up. Furthermore, if the value of the semaphore is 1 and several processes try to execute P[S] then only one of the processes is permitted to complete execution of P[S] and others are held waiting for the value of the semaphore to become greater than 0. The instruction V[S] just increments the value of the semaphore by 1.

Work reported herein was supported in part by Project MAC, an MIT research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-N00014-70-A-0362-0001.

The Cigarette Smokers Problem

Three smokers are sitting at a table. One of them has tobacco, another has cigarette papers, and the third one has matches -- each one has a different ingredient required to make and smoke a cigarette but he may not give any ingredient to another. On the table in front of them, two of the three ingredients will be placed, and the smoker who has the necessary third ingredient should pick up the ingredients from the table, make a cigarette and smoke it. Since a new set of ingredients will not be placed on the table until this action is completed, the other smokers who cannot make and smoke a cigarette with the ingredients on the table must not interfere with the fellow who can. Therefore, coordination is needed among the smokers. The actions of the smokers without the coordination are as follows.

X - the smoker with tobacco

α_x : pick up the paper
pick up the match
roll the cigarette
light the cigarette
smoke the cigarette
return to α_x

Y - the smoker with paper

α_y : pick up the tobacco
pick up the match
roll the cigarette
light the cigarette
smoke the cigarette
return to α_y

Z - the smoker with matches

α_z : pick up the tobacco
pick up the paper
roll the cigarette
light the cigarette
smoke the cigarette
return to α_z

To inform the smokers about the ingredients which are placed on the table, three semaphores a, b and c representing tobacco, paper and match respectively, are provided. On placing an ingredient on the table, the corresponding semaphore is incremented by performing a V[ ] operation. On the smokers' side semaphores X, Y and Z are used to signal that a cigarette has been smoked. The smoker who completes smoking a cigarette performs the operation V[ ] on the corresponding semaphore.

The smokers' problem is, then, to define some additional semaphores and processes, if necessary, and to introduce necessary P and V statements in these processes so as to attain the necessary cooperation among themselves required to ensure continued smoking of cigarettes without reaching a deadlock. There is, however, a restriction that the process which supplies the ingredients cannot be changed and that no conditional statements may be used. The first restriction is placed because the smokers are seeking cooperation among themselves and therefore should not change the supplier, and the second restriction is justified because P and V primitives were introduced to avoid having to coordinate processes by repeatedly testing a variable until it changes its value, and because the operation of making and smoking a cigarette has no conditional actions. It will be seen that the cigarette smokers' problem has no solution.

To give a precise statement of the problem, we will present below the processes involved. The set of processes in the enclosed area when taken together, represent the agent who puts the ingredients on the table. It should be recalled that semaphores a, b, and c represent tobacco, paper and match respectively, and the action of putting these ingredients on the table is represented by V[ ] operation on the corresponding semaphores. Among the processes representing the agent, such actions are performed by the processes R_a, R_b and R_c which represent the three possible actions of the agent. Initially semaphore s is 1 and the processes R_a, R_b and R_c compete to perform P[s], and the process which succeeeds puts the ingredients on the table by performing V[ ] on two of the semaphores. Processes β_x, β_y and β_z detect the completion of smoking of a cigarette; β_x performs V[s] when smoker X

indicates completion of smoking the cigarette by performing V[X]. The semaphore s serves to signal that it is time for the agent to place a new set of ingredients on the table.

R_a	R_b	R_c	the agent
r_a: P[s]	r_b: P[s]	r_c: P[s]	
V[b]	V[a]	V[a]	
V[c]	V[c]	V[b]	initially s = 1 and a,b,c,X,Y and Z = 0
goto r_a	goto r_b	goto r_c	
β_x: P[X]	β_y: P[Y]	β_z: P[Z]	
V[s]	V[s]	V[s]	
goto β_x	goto β_y	goto β_z	

X	Y	Z	
α_x: P[b]	α_y: P[a]	α_z: P[a]	
P[c]	P[c]	P[b]	the smokers
▬	▬	▬	
V[X]	V[Y]	V[Z]	
goto α_x	goto α_y	goto α_z	

▬ The blank stands for some operation performed by the process (in the case of the smoker this operation is that of smoking the cigarette).

On the smokers' side an attempt is made to represent their actions by means of the processes X, Y and Z. These processes, however, do not correctly represent their activity because of the possibility of a deadlock. For example, in one case, tobacco and paper are placed on the table, and process Y may perform P[a] before process Z, the process which was supposed to perform P[a], and thereby create a deadlock in which no process is able to proceed.

The smokers' problem is to define additional semaphores and processes and to introduce appropriate P and V statements so as to make them deadlock free. No alteration may, however, be made to the processes defining the agent.

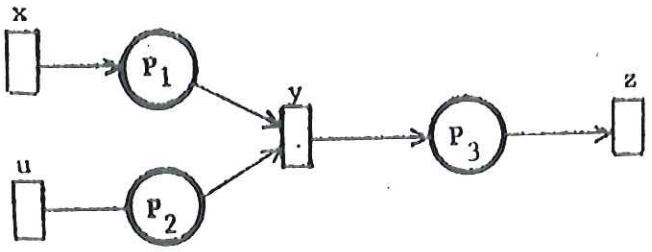
In order to obtain the necessary machinery to show that the smokers' problem has no solution, we will establish a correspondence between programs using semaphores and Petri nets. A brief introduction to Petri nets is given below, but a more complete treatment of Petri nets can be found in the work of Holt [2,3].

Petri Nets

Petri nets are directed graphs with two types of nodes called places and transitions. Directed arcs connect places to transitions and transitions to places. An arc may not directly connect a place to a place or a transition to a transition. The places can hold markers. A transition is said to be enabled (ready to fire) when all input places of the transition have markers. When a transition fires, it removes a marker from each input place and places a marker in each of the output places. A conflict between transitions arises over a place which is shared as a common input place, when both transitions are ready to fire and the place has just one marker. In this situation only one of the transitions fires and the other is disabled. Simulation of the net proceeds as transitions fire changing the marking distribution of the net. An example of a Petri net is shown in Figure 1b.

Petri net representation of processes

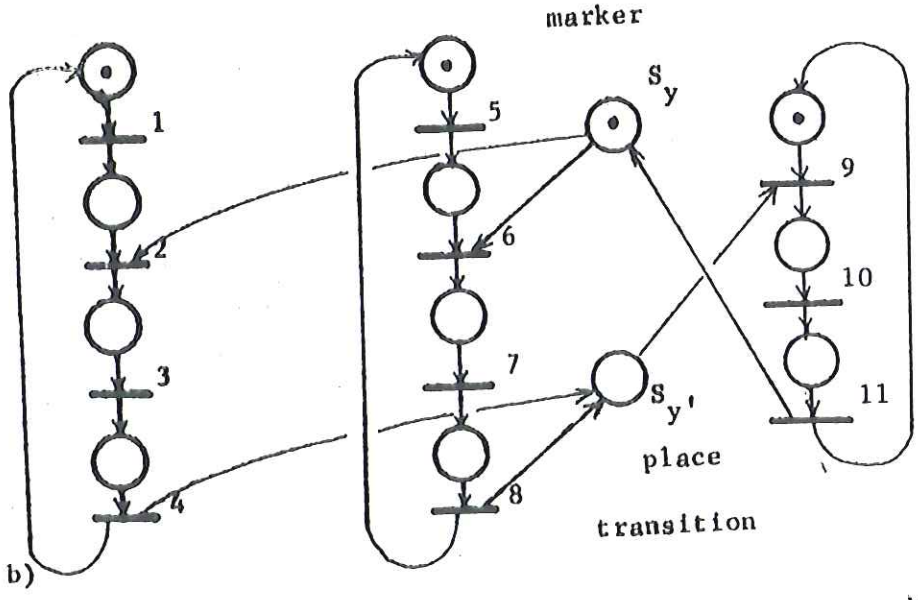
The flow of control in processes and coordination (harmonious cooperation) among the processes can be expressed clearly in Petri nets. The use of Petri nets for representation of processes simplifies analytic treatment of problems especially when the processes do not use conditional statements as in the case of the smokers' problem. In dealing with such processes, each instruction is represented by a transition (Figure 1). The transitions corresponding to instructions which are adjacent have a place in common; the place is an output place of the transition corresponding to the instruction which precedes and is an input place of the transition corresponding to the instruction which follows. The transitions corresponding to instructions other than the primitives P and V for operation on semaphores have just one input place and one output place. Just as the instructions form a sequence in the process, the transitions form a chain, and a marker at a place in the chain denotes the place where control



	P_1		P_2		P_3
1	x ← x + x	5	u ← u * u	9	P[S_y,]
2	P[S_y]	6	P[S_y]	10	z ← z + y
3	y ← x	7	y ← u	11	V[S_y]
4	V[S_y,]	8	V[S_y,]		goto 9
	goto 1		goto 5		

initially $S_y = 1$, $S_{y'} = 0$

a)



b)

Figure 1 Flow of Control in Processes and the corresponding Petri nets

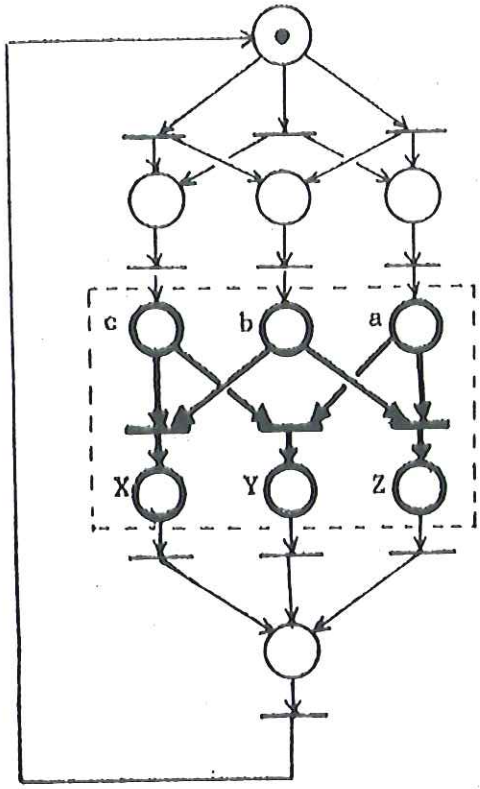
is. The movement of the marker through firing of transitions in the net corresponds to flow of control in the process.

In addition to the places mentioned above, there is a place for each semaphore. A transition representing the instruction P[S] has an additional arc from the place corresponding to semaphore S and from the transition corresponding to the instruction V[S], there is an additional arc directed to the place corresponding to semaphore S. The effect of the execution of instruction P[S] and V[S] is now clearly seen; P[S] decrements the number of markers at S by one and V[S] increments the number of markers at S by 1. The movement of the marker representing flow of control is then stopped at a transition representing P[S] until there is a marker at the place corresponding to S i.e. until the value of semaphore S is greater than 0 (Figure 1b).

In the Petri nets that represent flow of control in cooperating processes where the processes do not use conditional statements, no more than one input place of a transition is shared as input place with another transition. This is because only places corresponding to semaphores are shared as input places by transitions and the P[ ] instruction operates only on a single semaphore. Petri nets in which no more than one input place of a transition is shared as input place with other transitions are called simple Petri nets.

The desired coordination among the smokers can be expressed by the Petri net shown in Figure 2. Note that this Petri net is not a simple Petri net. A solution to the smokers' problem, however, if it exists, will provide a simple Petri net equivalent to this net, but we will show that there does not exist any simple Petri net equivalent to this net and thereby rule out any solution to the smokers' problem.

The part of the net drawn with bold lines in Figure 2 is an instance of the two out of three net. It is a decoding net whose inputs are a pair of markers and whose output tells which of the three possibilities that pair represents. This is the smallest of the class of decoding nets called r out of n nets. The 2 out of 3 problem states that the 2 out of 3 net cannot be transformed into an equivalent simple net. The author would like to acknowledge that the 2 out of 3 net was brought to the attention of the author by Jack B. Dennis, who also suggested how one might go about proving that there



A Petri net representing the desired coordination. The rest of the net corresponds to the supplier of the ingredients.

The 2 out of 3 net is drawn with bold lines.

Figure 2 A Petri net representing the desired coordination

is no simple Petri net equivalent to it. The proof given below is along those lines.

Theorem. There is no simple Petri net equivalent to the 2 out of 3 net.

Proof: We will assume that there is a simple Petri net equivalent to the 2 out of 3 net and construct a sequence of inputs to the net which leads to malfunction. Let a, b, and c be the input places and X, Y and Z be the output places of the net. Let I_{ab}, I_{ac} and I_{bc} represent the three different inputs to the net. In the case of I_{ab}, markers are placed in the places a and b, and as a result the net should place a marker in the output place Z, the outputs corresponding to I_{ac} and I_{bc} are Y and X.

In what follows, the initial value of j is 0, and the S_0 is the null sequence. S_j is a specific sequence from the set of sequences $(I_{ac} + I_{bc})^*$ where * is the Kleene star.

1. Consider P^j, the set of places into which a marker cannot be put into when the sequences of inputs to the net are taken from the set of sequences $S_j(I_{ac} + I_{bc})^*$. We know that P^j is a non-empty set because the output place Z (which corresponds to the input I_{ab}) is in P^j.

2. Let the semi-degree of a place in P^j for a particular sequence from the set of sequences $S_j(I_{ac} + I_{bc})^*I_{ab}$ be the length of the shortest sequence of transition firings which puts a marker in that place starting from the application of input I_{ab}. Furthermore, let the degree of a place in P^j be defined to be its smallest semi-degree over the set of sequences in $S_j(I_{ac} + I_{bc})^*I_{ab}$. Now, pick a place from the set P^j with the smallest degree and call it p_j; if there is more than one such place, any one of them will do. Since p_j is of the smallest degree, at least one transition which feeds into p_j has no input places from P^j. We will try to fill the input places of such a transition. As the net is a simple net, at most one input place of the transition may be a common input place with another transition. The task of filling such a place, if any, will be taken up only after the other places are filled because the other places, not being shared, will keep the markers once they are filled until the transition is fired.

3. Let p refer to the input place (of the transition) which we are trying to fill. Since p is not in P^j, there is a sequence in $S_j(I_{ac} + I_{bc})^*$ which puts a marker in p. Let this sequence be called S'_j. Now in attempting to fill the next input place we may have difficulty because there may be no sequence in $S'_j(I_{ac} + I_{bc})^*$ which can put a marker in the place. If this is the case let S_{j+1} be equal to S'_j and repeat the above procedure from step 1. Otherwise continue to fill the rest of the input places of that transition and let the sequence of inputs which achieves this be denoted by S.

4. At each step above, P^{j+1} will be larger than P^j because it will include all members of P^j and at least the place which we could not fill. Since at each step the set P^{j+1} is larger than P^j the above process must terminate because the net has a finite number of places. When the process terminates, we get a sequence S from the set of sequences $S_k(I_{ac} + I_{bc})^*$, a place p_k which is a member of P^k and a transition t which feeds into p_k and whose input places are filled. Since all input places of the transition are filled, it can be fired causing a marker to be placed in the place p_k. Thus we have a contradiction because by definition there is no sequence in $S_k(I_{ac} + I_{bc})^*$ which can cause a marker to be placed in any place in P^k. This completes the proof.

The fact that the smokers' problem cannot be solved with semaphores and the primitives P and V is revealing, but it should be remembered that in that problem we disallowed the use of conditional statements. If conditional statements were permitted, the coordination among the smokers could be expressed as shown in Figure 3. The operation of the processes in this figure is explained below with the help of Figure 4.

In the net of Figure 4, transitions r_1, r_2 and r_3 produce two markers for each marker placed in places a, b and c respectively. Thus, if markers are placed in b and c, places S_b^x, S_b^z, S_c^x and S_c^y get markers and transition x fires placing a marker in place X. This is a correct operation except for the unused markers left behind in S_b^z and S_c^y. These markers will cause unwanted firings of transitions y and z when a marker from the next input is placed in place a. The net will operate correctly if we can find a way of removing the unused markers so that they do not interfere with the next input. We could do this if a transition could be made to undergo a dummy firing in which it

```
r3: P[c]          r2: P[b]          r1: P[a]
    V[S_c^x]          V[S_b^x]          V[S_a^y]
    V[S_c^y]          V[S_b^z]          V[S_a^z]
    goto r3           goto r2           goto r1

αx: P[S_b^x]      αy ...            αz ...
    P[S_c^x]
    P[t_x]
    if x > 0 then  x ← x-1
                   V[t_x]
                   goto αx
    else
    V[t_x]
    P[t_y]
    P[t_z]
    y ← y+1
    z ← z+1
    V[t_y]
    V[t_z]
    V[S_a^y]
    V[S_a^z]
    P[X]
    goto αx
```

initially x, y and z are 0 and t_x, t_y and t_z are 1

Note: instruction such as $P[t_x]$ locks variable x so as to avoid simultaneous access to variable x.

Figure 3 A solution to the smokers' problem with a conditional statement

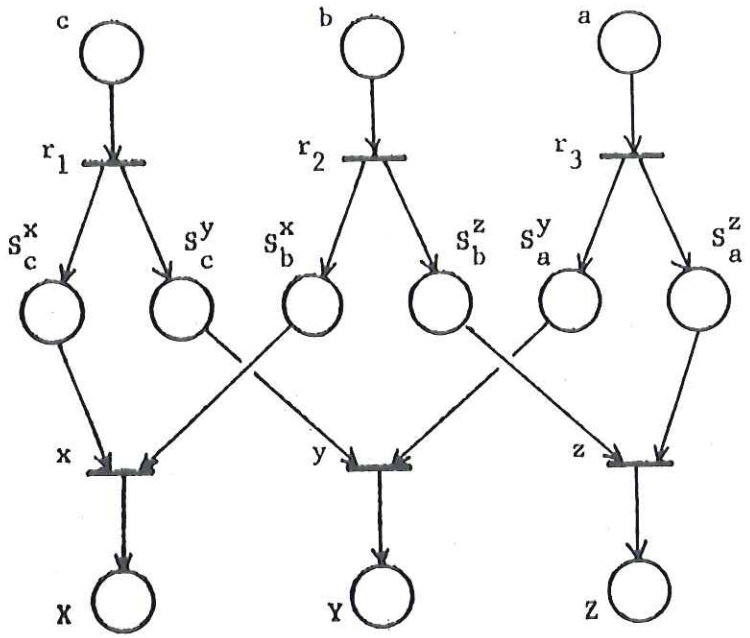
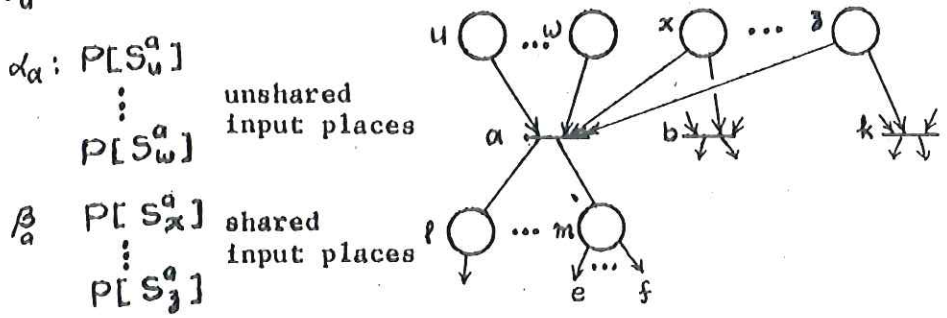


Figure 4 A Petri net for explanation of Figure 3

could remove markers from the input places and not place any markers in the output places. This is exactly what is done in the processes of Figure 3 with the help of conditional statements and variables x, y and z. If the transition x is fired when the variable x is greater than 0, the transition just removes markers from its input places but does not put any markers in its output places. In order to remove the left over markers mentioned earlier, say the marker in place S_c^y, we will set y to 1, fill the other input place of the transition y, namely the place S_a^y, and cause the transition to undergo a dummy firing.

In Figure 3, processes r_1, r_2 and r_3 perform the task of transitions r_1, r_2 and r_3 respectively and processes X, Y and Z perform the task of transitions x, y and z with the provision for dummy operation. The reason why the variables x, y and z are incremented by 1 instead of being set to 1 is that this way it is not necessary to wait for the left over markers to be removed before the next input is received.

The fact that a solution to smokers' coordination can be had with the help of conditional statements suggests that conditional statements greatly enhance the capabilities of P and V primitives. Below we shall examine just what type of coordinations can be had if we permit the use of conditional statements. We will first examine the class of coordinations corresponding to the conflict free Petri nets, and later study the class of coordinations corresponding to the Petri nets which may have conflicts. In both cases we will find that with the use of conditional statements, the P and V primitives are adequate for expressing the coordinations. In considering the use of conditional statements the first thing that comes to the mind is the repeated testing of a variable, say x and y, to get a construction such as "wait until x and y both become 1 and then proceed with some given action." Such constructions are wasteful of computational resources since the variables may be tested an arbitrary number of times. As a result, programs using such constructions may not have any upper bound on the amount of computation required to perform the necessary coordination. Therefore, we wish to avoid such constructions, and it is interesting to note that we can always get by without them as will be seen below.



P_a

α_a : $P[S^a_u]$
⋮ unshared input places
$P[S^a_\omega]$

β_a $P[S^a_x]$ shared input places
⋮
$P[S^a_y]$

$P[t_a]$ lock a

if $a > 0$ then $a \leftarrow a - 1$
$V[t_a]$ unlock a
goto β_a go back to β_a because it is a dummy operation.

else
$V[t_a]$
$P[t_b]$
⋮ lock variables b,...,k
$P[t_k]$
$b \leftarrow b + 1$ increment b,...,k so that the corresponding
⋮ transitions will undergo dummy transitions
$k \leftarrow k + 1$
$V[t_b]$
⋮ unlock b,...,k
$V[t_k]$
$V[S^b_\alpha]$ fill the other shared input
⋮ places of b,...,k
$V[S^k_\gamma]$
$V[O^e_\ell]$
$V[O^e_m]$ fill the output places
⋮
$V[O^f_m]$
goto α_a

Figure 5 Processes for conflict free Petri nets

Coordination Corresponding to Conflict Free Petri Nets

A conflict free net is one in which two transitions which share an input place are never enabled (ready to fire) at the same time. Thus the net is spared from having to make a choice among transitions. Furthermore, an enabled transition may proceed to fire without the fear of being disabled.

In programs corresponding to such nets, we have one process for each transition. The generic arrangement of instruction in such a process is shown in Figure 5. In the program, the semaphore S_u^a corresponds to the unshared place u which in an input place of a and S_x^a corresponds to the share place x which is an input place of transition a. Below we list the function of the other variables and semaphores used in the processes.

$a,\ldots,k$	variables used with transitions a,...,k respectively for dummy firings.
$t_a,\ldots,t_k$	semaphores used as locks to avoid simultaneous references to corresponding variables a,...,k.
$S_\alpha^b,\ldots,S_\gamma^k$	semaphores representing the other shared input places of transitions b,...,k (i.e. the places which are not shared with transition a).
$O_m^e,\ldots,O_m^f$	semaphores representing the output place m of the transition which is a common input place of the transitions e,..,f.

In understanding the operation of the processes, it will help to know that for each shared input place x, there are semaphores $S_x^a,\ldots,s_x^k$, one for each transition which feeds from it, and in the operation of the program when a transition fills this place, all of these semaphores are incremented so that the presence of the marker may be broadcast to all transitions which share that place. Later on when a transition fires picking up the marker, all of these semaphores are decremented by causing other transitions to do through dummy firings. The dummy firing, as in Figure 3, are performed to remove the unwanted markers. In order to do this correctly, the transition increments appropriate variables by 1 and increments the semaphores corresponding to the other input shared places of these transitions.

P_a

α_a: $P[S^a_u]$
⋮
$P[S^a_w]$
β_a: $P[S^a_x]$
⋮
$P[S^a_y]$
$P[C_i]$
⋮
$P[C_j]$
$P[t_a]$
if $a > 0$ then $a \leftarrow a - 1$
$V[t_a]$
$V[C_i]$
⋮
$V[C_j]$
goto β_a

else
$V[t_a]$
$P[t_b]$
⋮
$P[t_k]$
$b \leftarrow b + 1$
⋮
$k \leftarrow k + 1$
$V[t_b]$
⋮
$V[t_k]$
$V[S^b_d]$
⋮
$V[S^g_y]$
$V[C_i]$
⋮
$V[C_j]$
$V[O_p]$
$V[O^e_m]$
⋮
$V[O^f_m]$
goto α_a

These instructions resolve conflicts among transitions.

the semaphores C's are ordered by their subscripts, and in the processes they must occur in that order

Figure 6 Processes for Petri nets which may have conflicts

Coordination Corresponding to Petri Nets With Conflict

Petri nets in general allow conflicts which represent a situation in which two enabled transitions share an input place. In this case an arbitrary choice of transitions must be made. The primitive P is capable of making such a choice; as in the case when different processes perform P[S] and the value of S is 1. The generic form of each process for implementing coordination expressed in a Petri net is shown in Figure 6 . The technique employed here is similar to that in the previous case of conflict free Petri nets (Figure 5) with the only addition of semaphores $C_1,\ldots,C_j$ to resolve the conflicts. Semaphores $C_1, \ldots, C_j$ correspond to conflict clusters [4] where a cluster is a set of mutually conflicting transitions. In the process operations are performed only on those clusters to which the transition belongs.

It is interesting to note that the solutions presented above avoid arbitrary amounts of repeated testing of variables and thus there is a bound on the number of instruction executions required for any given coordination. Moreover, the method by which this is achieved is a general one and could be used in programming practice.

Generalized Primitives

Even though the P and V primitive when taken together with the conditional statements, are powerful enough to express the general class of coordinations specifiable as Petri nets, they have a serious deficiency in that the processes becomes unnecessarily complicated. This costs both in terms of the loss of transparency of the programs and in terms of the overhead the extra instructions imply. Such indirect implementation of coordination would not have been necessary if we had a generalized primitive such as $P[S_1,\ldots,S_k]$ which waits for all semaphores to become non-zero and then simultaneously operates on all of the semaphores. This instruction cannot be broken down into a sequence of $P[S_1],\ldots,P[S_k]$ instructions. The sequence of instructions $P[S_1],\ldots,P[S_k]$ does not represent the same operation as the generalized instruction $P[S_1,\ldots,S_k]$ because when not all semaphores are non-zero the generalized primitive does not act on any semaphore but in the case of instruction $P[S_1],\ldots,P[S_k]$ the process decrements the semaphores until it encounters one which is zero. When semaphores are treated as resources, this has the effect of hoarding resources.

Structures for implementing Petri nets have been studied by the author in his earlier work [4]. Such structures could be used to implement $P[S_1,\ldots,S_k]$ instruction where k is some large but fixed number. When the number of semaphores on which this operation is to be performed is larger than this number, the compiler could always break the instruction into smaller ones with the aid of conditional statements by the method already presented.

BIBLIOGRAPHY

1. Dijkstra, E. W. Co-operating sequential processes. Programming Languages, F. Genuys, Ed., Academic Press, New York 1968. [First published as Report EWD 123, Department of Mathematics, Technological University, Eindhoven, The Netherlands, 1965.]

2. Holt, A. W., et. al., Final Report of the Information System Theory Project. Technical Report RADC-TR-68-305, Rome Air Development Center, Griffiss Air Force Base, New York, 1968.

3. Holt, A. W. and F. Commoner, Events and Conditions. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, Association for Computing Machinery, June 1970, pp 3-52.

4. Patil, S. S. Coordination of Asynchronous Events. Report MAC-TR-72, Project MAC, M.I.T., Cambridge, Massachusetts, June 1970.