MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

REMARKS ON CONCURRENT PROGRAMMING PROBLEMS

Elód Knuth
Computer and Automation Institute of Hungarian Academy of Sciences

I. SYMMETRY CONCEPT IN CONCURRENT ALLOCATION

1. The cyclic allocation problem

We introduce this problem as an extensive generalization of the well-known mutual exclusion problem, see e.g. Dijkstra [1].

Let S be a (finite) set of exclusively accessible resources and P be a (finite) set of cyclic processes. Each of the processes consists of two sections called free section and allocation section. Entering the allocation section needs engaging a definite subset $S_i \subset S$ prescibed by the process $P_i \in P$.

As it is usual, we do not suppose anything of the speed of the processes, so the sequence of events is completely unpredictable.

The problem is to construct suitable algorithms for the processes which satisfy the allocation demands. The difficulty is the same as in the mutual exclusion problem, that is, the execution of any algorithm is time consuming with unpredictable execution time therefore there may well occur conflicts among the demads arisen.

The requirements of the solution we use are comprised in the following points A and B. We remark that these are not the only possible requirements of pratical interest.

A.) SATURATION

A process necessarily must not be suspended if its demands are satisfiable. (This heuristic definition will be exactly reintroduced in part II.)

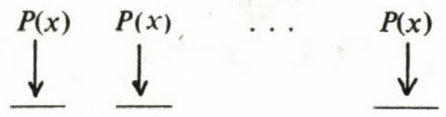
B.) SYMMETRY

Suppose that once resources are released and a set P' of processes of satisfiable demands comes to existance. Now symmetry means that any of the elements of P' has the same possibility to seize its resources (and continue its progress).

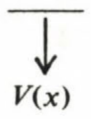
2. On the symmetry concept

As an axiom we suppose the symmetry of the usual semaphore operations. This is illustrated in the following figure.

Suspended processes:



Alarming process:



Now we suppose that $V(x)$ causes an alarm of one and only one waiting process to continue its progress and we have no information about which will really do. All the suspended processes have the same chance.

Remark: It is no matter that the semaphore primitives really have this property or not. However, the requirement "a solution must be symmetric if P and V are" is a proper logical formulation of the fact that we are not allowed to cause any static priority in our algoritms. (To construct priority systems, first we must be able to build symmetric, priority less systems).

3. Parallel semaphore operations

Parallel (multiple) semaphore operations have introduced by several authors, e.g. Patil [2]. The reason of introducing them is the deadlock situation caused by the use of sequential P operations. The effects of the operations are the following:

The execution of the operation:

$$P(x_1, x_2, \ldots x_k)$$

causes suspension if not all the semaphore values seized are positive, otherwise all semaphore values are decreased by one and the executing process continues its progress.

The execution of the operation

$$V(x_1, x_2, \ldots x_k)$$

increases all the semaphore values by one and performs the necessary alarms for the waiting processes.

It is obvious that if these operations may really be put into practice, the cyclic allocation will be no problem, for assigning a semaphore to each resource we could program our processes simply by :

```
LOOP:

        free section;
        P(· , · , . . . , ·);
        allocation section;
        V(· , · , . . . , ·);
```

4. Definition of parallel operations

Giving an exact definition is far from trivial, and there are several possibilities. Now we quote the version of Patil.

If not all the semaphore values $x_1, \dots, x_k$ are positive the execution of the operation

$$P(x_1, \dots, x_k)$$

suspends the executing process placing it into the waiting list corresponding to the first semaphore having nonpositive value. Any later reactivation of the process causes reexecution of the operation which may well result suspension again in a queue corresponding to another semaphore.
The operation

$$V(x_1, \dots, x_k)$$

must alarm the waiting lists corresponding to semaphores of values having become positive by the execution. However, it is not sufficient to remove one process from the waiting list. All the waiting processes must be alarmed as we have no information of which of them can continue.

No doubt, we could hardly introduce these complicated operations as "primitives". This was correctly pointed out by Parnas [3]. It is noted in the same paper that the parallel operation can be programmed in a "straightforward" way using P and V and conditionals. Let us take a closer look at this question:

5. Implementation problems of parallel operations

Let us denote $P_1 \leftrightarrow P_2$ if the subsets of resources needed by the two processes are not disjoint. Consider the equivalence classes generated by the transitive closure of the relation $\leftrightarrow$, and assign mutual exclusion semaphores to each class. Let m be one of them.

Let $x[\ ]$ be the semephore array representing the resources (initially 0) and $y[\ ]$ be a corresponding integer array having initial values 1.

Replace the parallel P operation by the following sceme:

```
entry: $P(m)$;
      for $i := 1$ step 1 until $k$ do
         if $y[i] < 1$ then
             begin
                  $V(m)$;
                  $P(x[i])$;
                  goto entry
             end;
      for $i := 1$ step 1 until $k$ do
         $y[i] := y[i] - 1$;
      $V(m)$;
```

If we can prove this version works correctly the following problem will remain unsolved: How many $V[\ ]$ operations must be performed by our implemetation of parallel V operation. Solving this question the program becomes a little more complicated, but the solution is in fact possible.

However, now I am not concerned to give a complete implementation or discussion of its correctness. The only thing I call attention to is that we are not fully agreed with Parnas on the "straightforward" possibility of implementation. Of course, there are several other possibilities of the implementation but we have not heard about a solution simpler than this, and we would be glad to hear of it if any exists.

Be that as it may, the main inconvenience of these implementations is caused by static priorities which are unavoidable whenever the resources are consistently sequentially tested and released.

6. Ensuring the symmetry

There are famous particular cyclic allocation problems such as the so-called "Dining Philosophers Problem" solved by Dijkstra [4] using only single semaphore operations, and the so-called "Cigarette Smokers Problem" having been considered unsolvable for Patil tried to prove its impossibility while Hebermann [5] and Parnas [3] found a crafty solution for it.

These solutions are built on particular tricks, however, we found an extreme generalization of Hebermann's technique which led to a general existence theorem:

Arbitrary cyclic allocation problem can be programmed symmetrically using only the simple P, V primitives (and even without using conditionals).

This result will be published in the next issue of the Hungarian journal Acta Cybernetica (1976).

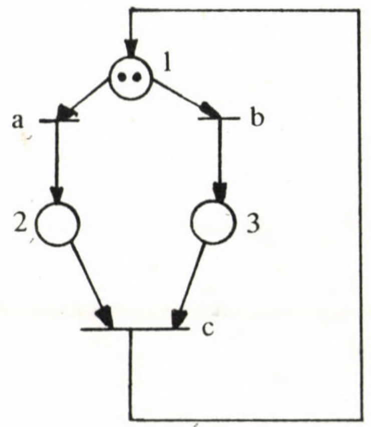
II. A PROBLEM IN THE THEORY OF PETRI NETS

1. Petri Nets

Now I give a very brief introduction to Petri Nets; one can find detailed investigations in the works Holt-Commoner [6] and Petri [7].

A Petri Net is defined to be a labelled directed graph with two node types called places and transitions such that every edge connects a place to a transition or a transition to a place.

A marking M is a function from place labels into nonnegative integers. For example:



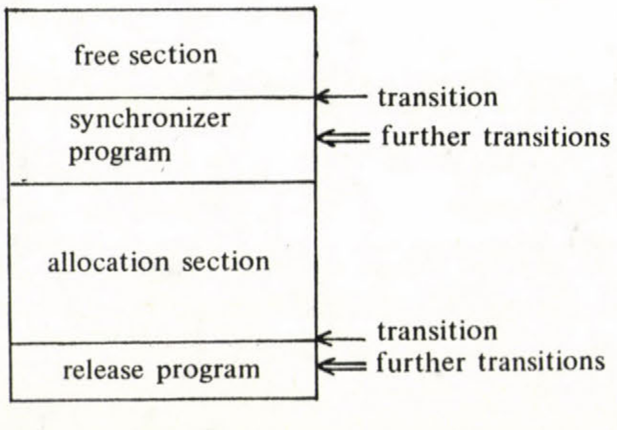
$M = [2,0,0]$

The marking of a net can be changed by firing a transition. A transition is firable if its input places are all positively marked. The firing decreases the number of markers at all the input places by one, and increases the outout markers by one.

Petri Nets can be used to answer a number of interesting questions in connection with concurrent pıograms for example the deadlock problem.

2. Verification of cyclic allocation systems

Suppose we have a solution and we are intending to verify its correctness. Then each of our cyclic processes has the following form:



Suppose we have made a Petri Net representing our solution. Then we can use the usual methods to prove that the system is free of deadlocks, there may not be conflicting processes in their allocation section at the same time, and other properties.

However, what about the saturation condition?

It is obviously not desirable that a process must really not be suspended if its demands are satisfiable for releasing resourcces and recognizing the free resources surely needs firing a positive number of transitions.

So, we should weaken our condition:

"If the allocation demand is satisfiable then the process must not wait for long."
To describe this exactly we introduce the

3. Classification of transitions

Let T be the set of transitions of a Petri Net. Select a subset $S \subset T$ called significant transitions.

We define the net (regarding an initial marking and a set of significant transitions) to be regular if the net is free of deadlock, but if we prohibit the significant transitions from firing at any time then the system will necessarily get into a deadlock within firing a finite number of transitions.

The states (markings) reached are called main states. A main state is characterized by the property:

If a transition is firable then it is a significant transition.

Reintrodicing the SATURATION condition:

A solution of the cyclic allocation problem satisfies the condition if the Petri Net representing the system with two main transitions: leaving the free and the allocation section; is

a.) regular

b.) if the system is in a main state and the resources used by a process are all free then the transition at which the process rearched the local deadlock may not be after the free section and before the allocation section at the same time.

4. Problem

We propose to study the regularity of the nets with given significant transitions, and produce the possible main states.

A system which is not regular might be considered one having the generalized form of the "infinite after-you blocking" introduced by Dijksta [1].

Further, we remark that it is no matter that all the solutions of the cyclic allocations can be described by Petri Nets or not. Obviously not. We aimed to propose the regularity problem and call attention at the transitions of Petri Nets being able to have different roles (even in the net structure).

Naturally, this problem can be proposed not only for Petri Nets but for more general computational structures too.

References

[1] E.W. Dijkstra, Cooperating Sequential Processes. Programming Languages (1968).

[2] S.S. Patil, Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes. Proj. MAC, Computational Structures Group Memo 57. (1971).

[3] D.L. Parnas On a solution to the Cigarette Smoker's Problem (without conditional statements). CACM 181-184, (1975).

[4] E.W. Dijkstra , Hierarchical Ordering of Sequential Processes. Operating Systems Technincs (1972).

[5] A.N. Habermann, On a solution and a generalization of the cigarette smoker's problem. Techn. Rep. Carnegie-Mellon University, (1972)

[6] F. Commoner, S. Holt, Marked directed graphs. J.Comp. and System Sci. 5.5. 511-523, (1971).

[7] C.A. Petri, Concepts of net theory. Proceedings of the Symp. on Mathematical Foundations of Computer Science. High Tatras, 137-146. (1973).

Összefoglaló

Konkurens programok konstruálásáról

Knuth Előd

A dolgozat bemutatja az u.n. ciklikus allokáció feladatát, ismerteti a megoldás nehézségeit, és bevezeti a Petri-hálók regularitásának fogalmát, mely fontos szerepet játszik az allokációs rendszerek verefikálásában.

Р Е З Ю М Е

О КОНСТРУКЦИИ КОНКУРРЕНТНЫХ ПРОГРАММ

Е. Кунтх

Работа показывает задачу о так называемом "циклическом распределении ресурсов", трудности решения, и вводит понятия регулярности сетей Петри, которое играет важную роль в верификации систем распределения объектов.