RECOVERABILITY OF PROCESSES

Philip M. Merlin

Technical Report #44
February 1974

Department of Information and Computer Science
University of California, Irvine

ACKNOWLEDGEMENT

I would like to express my sincere thanks to Professor David J. Farber for his excellent support and guidance.

1. INTRODUCTION

During the last few years, computing systems have become more and more complex and, in many cases, are composed of several, almost independent, units working in a parallel mode. This characterization seems to be clear for distributed computer systems as well as for the centralized systems. Today, "one" computer is composed by several CPU's, memories, I/O processors, pieces of software, and each of these units are almost independent. Several computer networks, connecting tens of computers and serving a wide range of users, have been developed and operating. This movement through the complexity of the computing systems will, probably, continue in the future [FALK].

With the increased number of elements that compose computer systems, the probability of failure of at least one of these elements is relatively high. In this case, it is important to protect the remaining elements. That means it is necessary to organize the systems so that if parts of its elements are malfunctioning, the rest of the elements will continue to work correctly. In this case, the total performance of the system can be reduced by a failure, but one or a few elements may not cause the entire system to collapse. This philosophy is called the "best-effort" [FARB, METC].

To deal with the complexity of the systems, the concept of "process" was introduced [DENNIS,SALT,HORN,DENNING and many others]. To study the properties of processes, several models were proposed [PETRI, ESTR, KARP, GOST, CERF, POSTEL, LARSON, GOST1]. But, a great amount of obscurity covers part of this concept. Many properties of processes are not understood to date.

The concept of failure recoverable processes and failure immune processes have almost not been theoretically studied. Without a better understanding of these two concepts, it is unlikely to expect a good implementation of the "best-effort" philosophy in complex systems that will be designed in the near future.

This manuscript proposes an approach to the study of failure recoverable processes and for the analysis of the possible sequences of events that happens when a failure occurs. This approach is based on a model of processes behavior in which the concept of a failure is introduced. Section 2 describes the model and points out the necessary and sufficient conditions for the recoverability of a process if a given failure occurs. Section 2 also describes the case of multiple failures and an algorithm for the analysis of multifailures of the same kind is given. The algorithm can be extended for the general case of multiple failures. Section 3 discusses recoverability in a system of interacting processes.

2. THE MODEL

The proposed model is based on a variation of Petri nets.

2.1. PETRI NETS

Petri nets were developed by Carl Adam Petri and further elaborated by Anatol Holt [HOLT]. Petri nets model conditions represented by nodes and events represented by transition bars. The holding of a condition is represented by placing a token on that node. Directed arcs connect nodes to bars and bars to nodes. A transition bar (event) can fire (occur) if all the nodes (conditions) input to that transition bar (events) have tokens (hold). When a transition bar fires it removes one token from each input node and places one token on each output arc.

Figure 1 shows, as an example of Petri net use, the model of a simple protocol, P3, connecting between processes P1 and P2. Note that protocols are a special kind of processes [GOST1]. In this protocol, when P1 is ready to send a message (A holds a token) f1 fires. Then a message is sent (M holds a token) and the sender enter in the state of waiting for the acknowledge (W holds a token). When P2 is ready to receive the message (B holds a token) f2 will be fired. C holds a token (the message is received). In this position f4 can fire and the token of C pass to E and to K. A token on E is the notification to P2 that the message was received, and the token in K represents that the acknowledge is sent. Now f3 can fire removing the token from W and K and putting a token on D. The token on D represents the notification to P1 that the transfer was finished. P1 can remove the token from D, and trigger again the process P3 by putting a token on A. In the other side, P2 can remove the token from E and to start his part in the protocol by putting a token in B.

The state of the Petri net is defined by the set of nodes holding tokens. All the possible states in which a Petri net can stay and the possible transitions between them define a state machine called Token Machine (TM). The TM for the net of figure 1 is shown in figure 2, assuming initial condition AB.

The description above is correct only if all the components of the protocol P3 work properly. But, what happens if the message M is lost? What is the behavior of P1 and P2 under this failure? With the tools developed to date we can not answer this question in general. In the next section we will develop new tools for a better understanding of these questions.

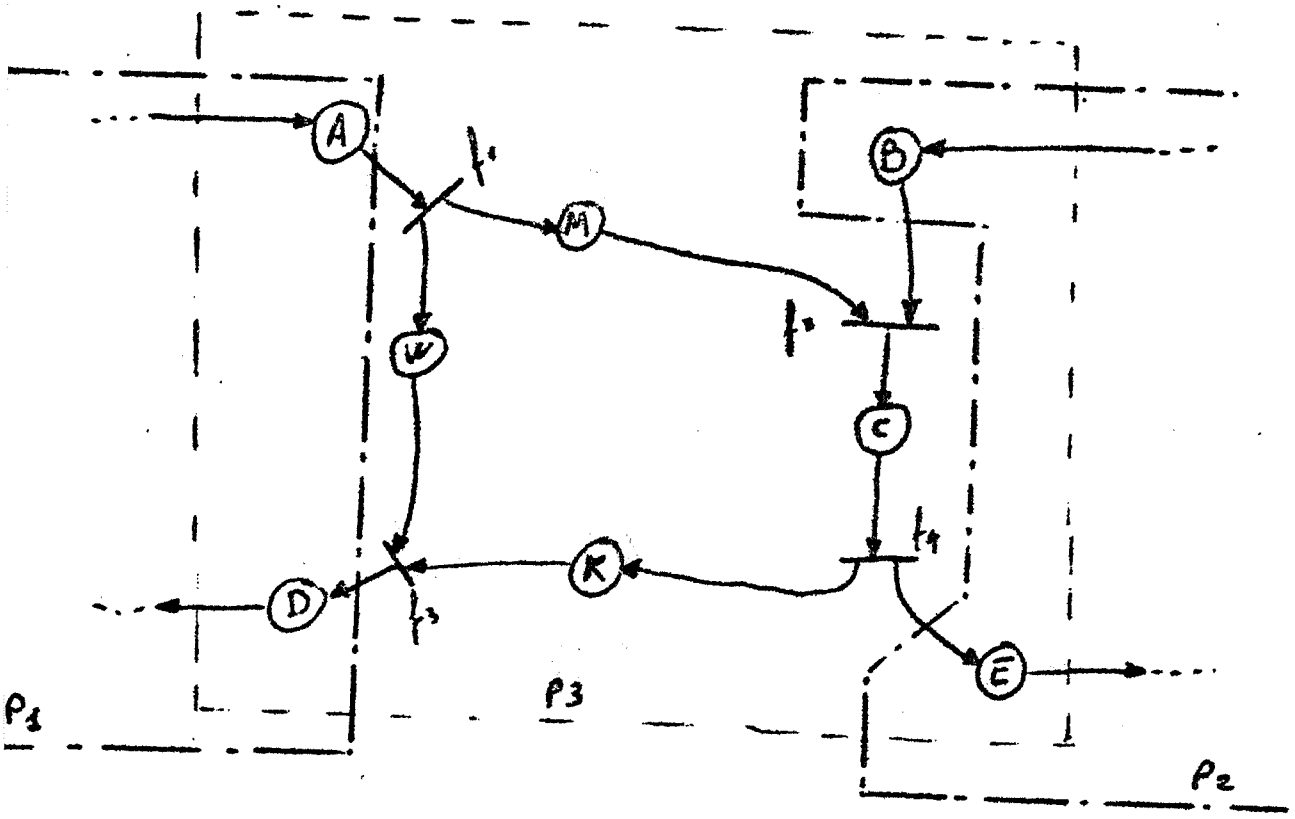

Figure 1: Petri-net representation of processes

2.2 THE REPRESENTATION OF POSSIBLE FAILURE

Suppose that a condition in a Petri net may fail. In this case, a token held by this condition may disappear. It is possible to represent this characteristic by adding a new branch to the TM. This branch will represent the possible flow of the execution when the "problematic" token disappears. For illustration, suppose that the message M in figure 1 can be lost. It means that when M holds a token, this token can disappear. This situation can be represented by adding an arc from state WMB (figure 2) to a new state WB. Since in state WB no bars can fire, WB is a final state. This new TM is shown in figure 3; thick lines represent the TM in the case that no failures occur, and thin lines describe the paths added since a failure. In the arc connecting these two parts, we write the name of the failing condition (M in the example). The same representation will be used in the following TM's figures in this work. This new machine, including the TM and the added paths for possible errors, will be called, in general, ERROR TOKEN MACHINE (ETM).

In the ETM, we call its states that exist also in the TM "LEGAL STATES". The other states are called "ILLEGAL STATES". Note that a process may be in an "illegal state" only if a failure has occured.

From Figure 3, we can conclude that the process represented in figure 1 is not recoverable from a failure in M, because under such failure the execution sequence arrive at a state (WB) where there is no way to return to normal execution (a legal state).

Note that in general, the ETM for a possible failure include all the possible paths in which the execution can flow if this failure occurs. It means, it is necessary to add to the TM new branches exiting from all the states that include the possible faulting condition.

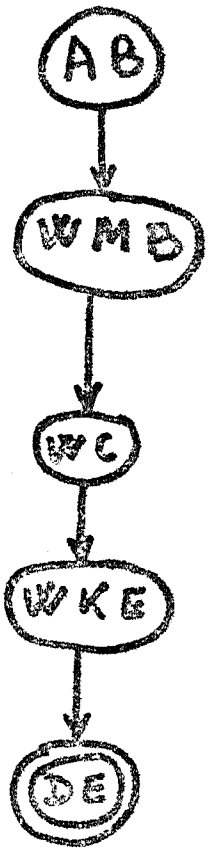

Figure 2: Token Machine for Petri-net of Figure 1

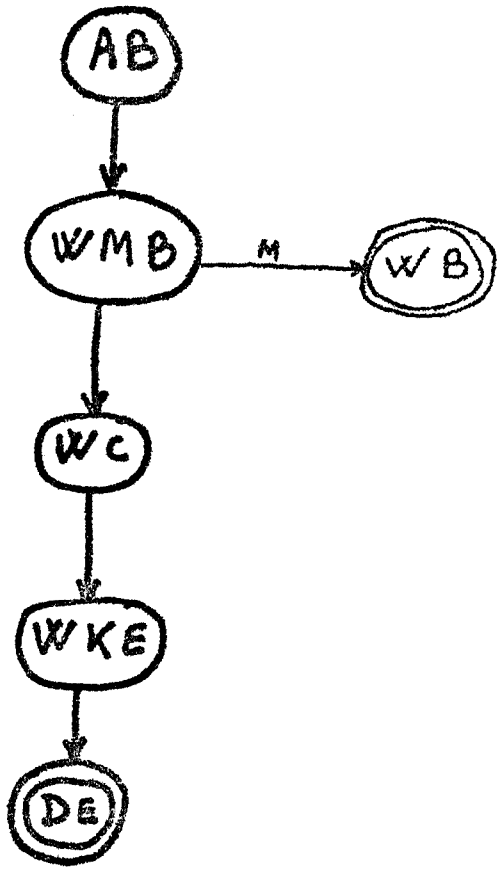

Figure 3: ETM for Figure 2 under a failure in M

At this point, we can state the conditions for recovery:

"A process P is recoverable from failure F if and only if in the ETM of P for failure F, all the directed paths through illegal states arrive to legal states."

It means, after a failure, the execution sequence must return to normal execution after a finite number of steps.

From the properties of directed graphs, we can derive an equivalent set of conditions:

"A process P is recoverable from failure F if and only if in the ETM of P for failure F:

1. The number of illegal states is finite.
2. There are no final illegal states.
3. There are no directed loops including only illegal states."

In the previous discussion we deal only with the case that a token may disappear. But in the same way, because of a fault, a node in a Petri net may generate a token. This situation may also be represented by adding new branches to the correspondent nodes in the TM. The approach is similar to the previous case. In the following sections, we deal only with the first kind of failures.

2.3 MULTIPLE FAILURES

In this section the previous approach is extended to the case that several failures can occur. This extention is introduced by the two following examples.

2.3.1 EXAMPLE

Figure 4 describes a variation of the protocol shown in figure 1. Assume (for simplicity) that arc T can be activated only after "a long time". This assumption is not mathematicaly represented in our model and we leave this point to further exploration in the future.

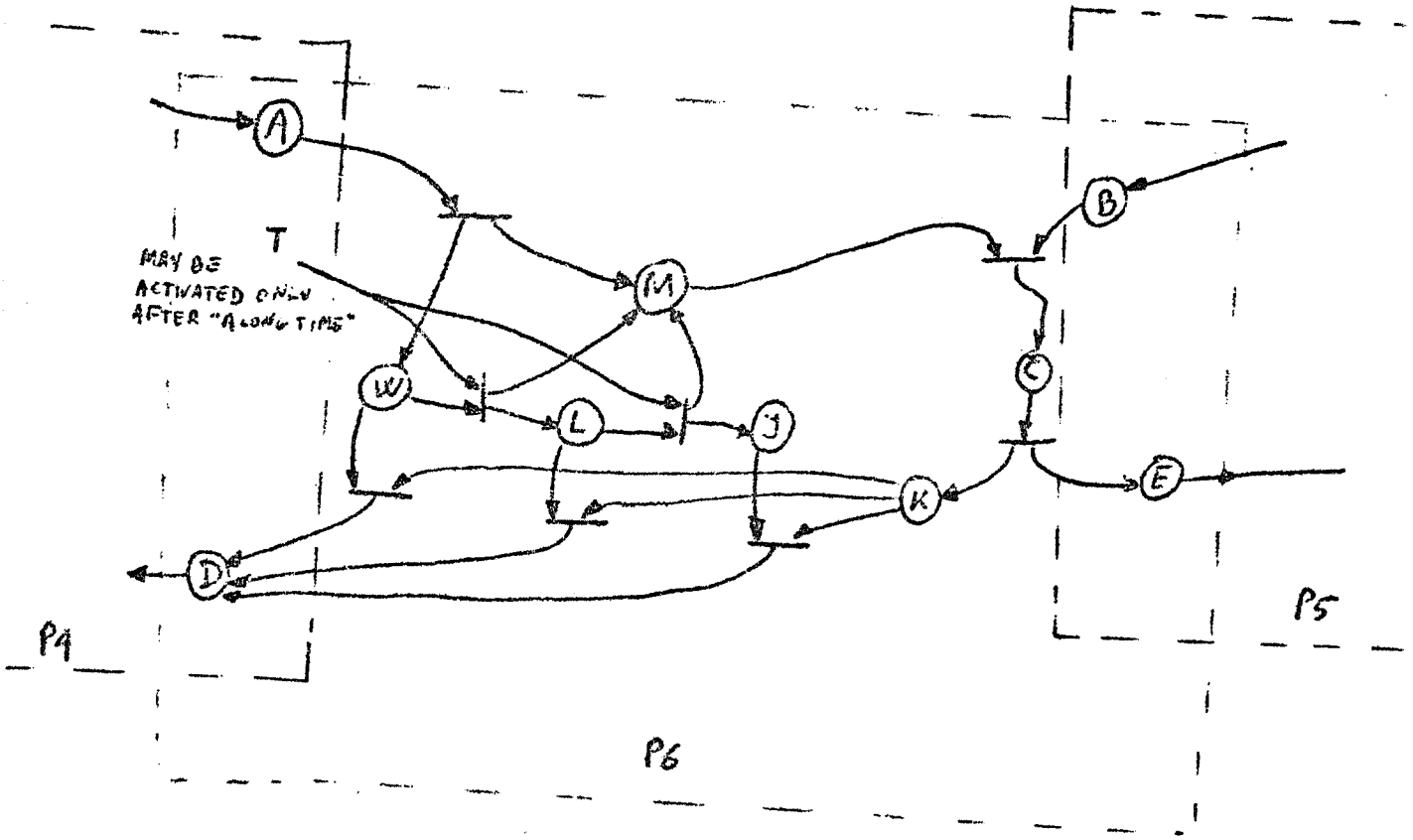

Figure 4: Petri Net of example 2.3.1

Figure 5 shows the ETM for this protocol (assuming initial condition AB) for the case that M may fail. The ETM of figure 5 shows that in spite of the failure in M the execution sequence (in the case of failure) return to a legal state (DE). It means, the protocol described in figure 4 is recoverable from one failure in M.

Suppose now, that a second failure may occur in M. In this case, a new branch will be added, but now to the node LMB (figure 5). This procedure can be applied again for a given number of possible failures. Note that this procedure is suitable not only for the case of multiple failures in one condition, it can be applied in general when several conditions may fail.

Figure 6 shows the ETM for the process of figure 4 in the case that three failures in M may occur. In this ETM we can see that the process is recoverable from one or two failures but not for the third. Paths 1 and 2 return to a legal state, but not path 3.

2.3.2 EXAMPLE

Figure 7 describes another variation of the Petri net shown in figure 1. In this case, there is a protocol process (P9) connecting between the processes P7 and P8. Note the appearance of the ill defined function T that is described in figure 4. Figure 8 shows the ETM (assuming initial condition AB) for the protocol P9 in which a failure in M may occur. In this case, the new branch for the failure returns to a legal state (WMB → WB → AB); it means, this failure is recoverable.

Suppose now, that two faults may occur in M. In this case, the sequence in the ETM of figure 8 will be:

$$AB \rightarrow WMB \rightarrow WB \rightarrow AB \rightarrow WMB \rightarrow WB \rightarrow AB \qquad (1)$$

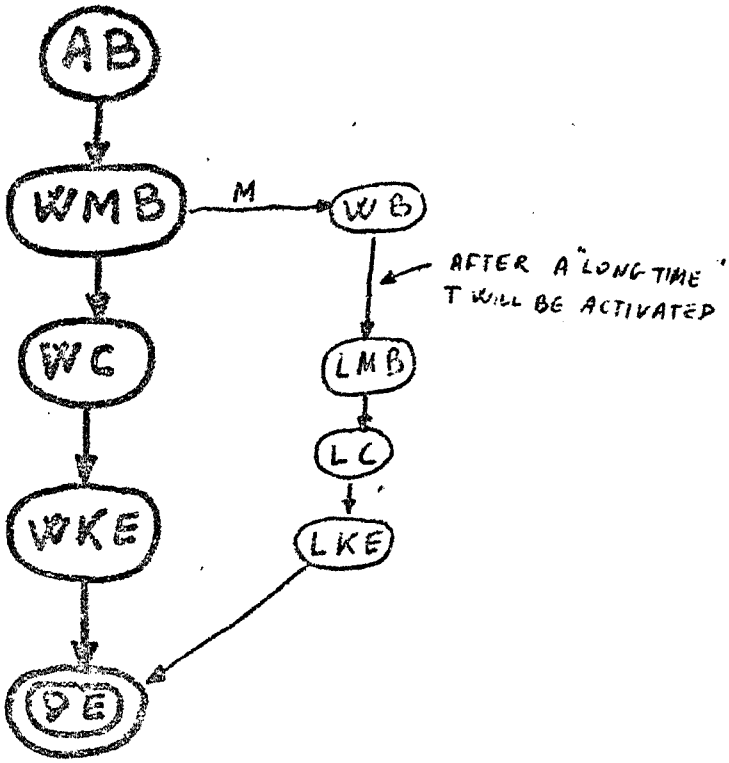

Figure 5: EMT of example 2.3.1

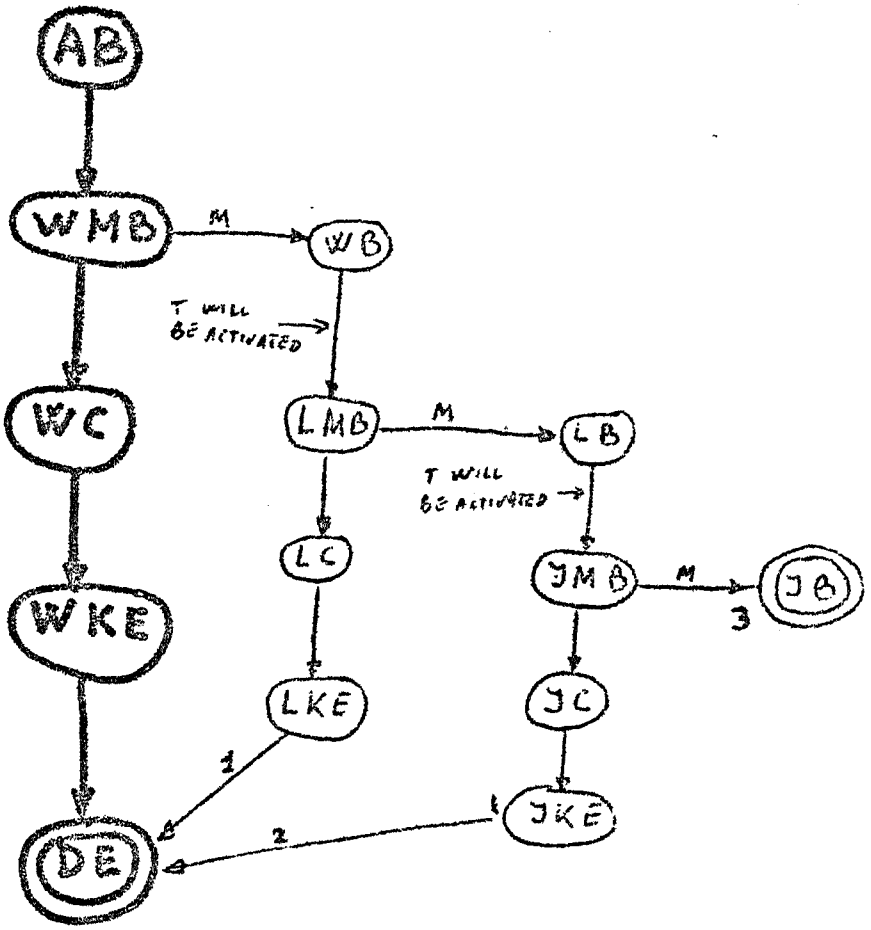

Figure 6: ETM of example 2.3.1

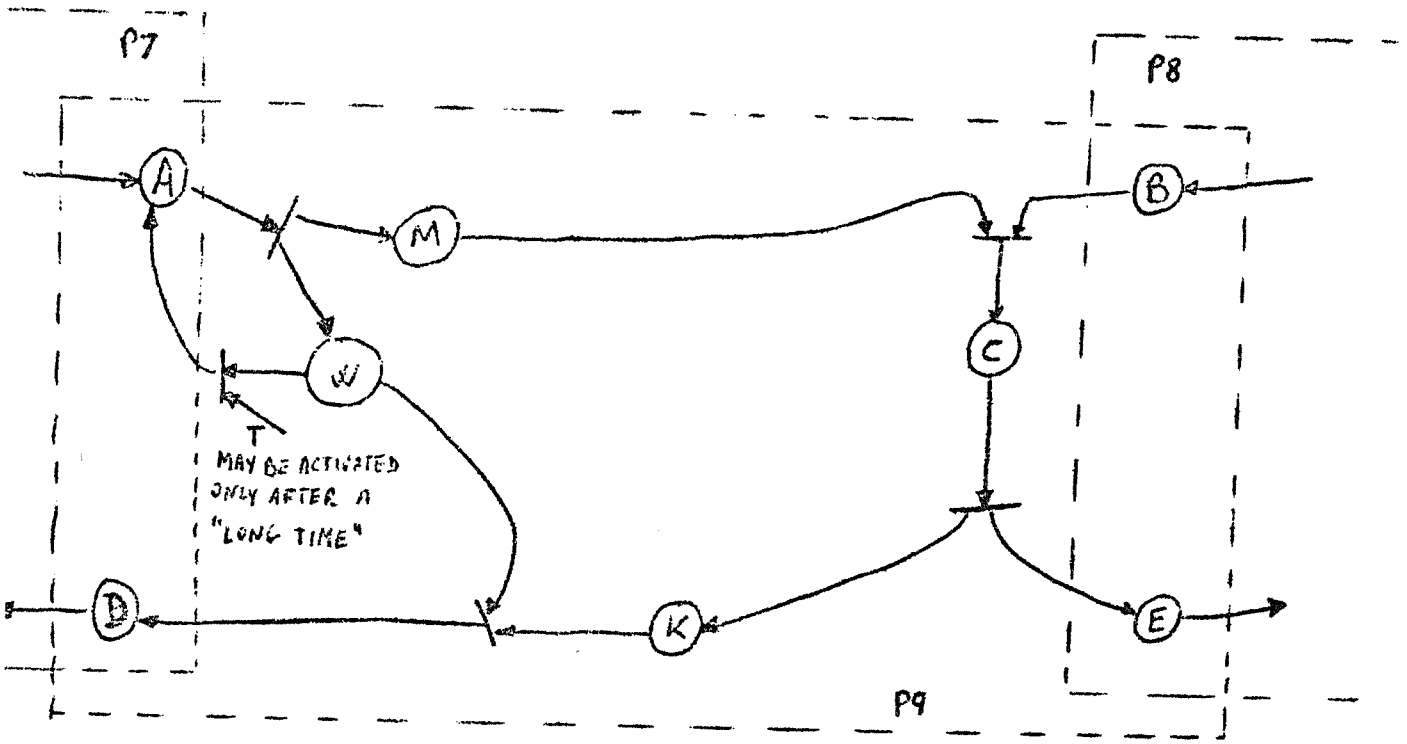

Figure 7: Petri-net of example 2.3.2

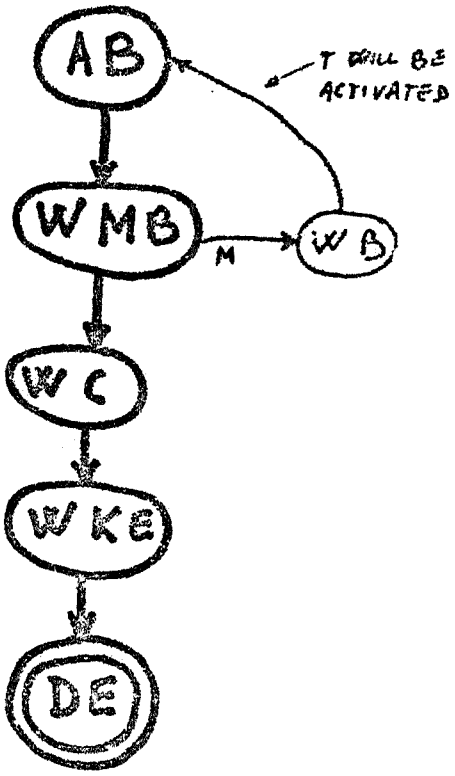

Figure 8: ETM of example 2.3.2

It means, for one or two failures, the machine executes the same sequence around the loop:

$$WB \rightarrow AB \rightarrow WMB \qquad (2)$$

The result is similar for more than two errors in M. Loop (2) will be executed as many times as a failure in M may occur. But always, the execution sequence will return to a legal state. It means, this protocol is recoverable from any number of failures in M.

2.3.3 TEST FOR RECOVERY UNDER MULTIPLE FAILURES OF THE SAME KIND

Define the set C1 of conditions such that:

1. The number of illegal states is finite,
2. There are no final illegal states,
3. There are no directed loops including only illegal states.

Algorithm: (suppose P is a process and F a possible failure.)

(1) ETM1 ← TM; name the nodes of ETM1 "new nodes".

(2) I ← 0 .

(3) add to the "new nodes" of ETM1 branches for the case that F fail; name this new graph ETM; name only, the added nodes "new nodes".

(4) if ETM=ETM1 (end; P is recoverable from any number of failures F).

(5) if C1 (I ← I+1; ETM1 ← ETM; go to (3)).

(6) end; P is recoverable from up to I failures in F

This algorithm has several interesting properties:

1. The trivial case is when after the first execution of (3) there is a directed loop including legal and illegal nodes, and F is not included in the illegals. In this case, in the second execution of (3) will be no change in ETM, so that (4) holds and the algorithm terminates. Figure 7 and 8 (section 2.3.2) shows an example of this simple case.
2. The algorithm may not terminate.

3. "Recovery from any number of failures F" includes the case that after the occurrence of certain numbers of failures F the control continues executing a sequence of legal states that do not include the condition F. In this case the failure F cannot occur more. So that, "recoverable from any number of failures F" means, in general, that faults in F can not inhibit the process to return to normal execution.

2.4 DISCUSSION

In the previous sections, a method has been developed for checking if a process is recoverable from a given possible failure. The fact that all the examples that introduce the method are related to protocols does not restrict its generality. All the methods presented and all the characteristics described in this manuscript apply to processes in general.

In section 2.3 the method has been extended for the case that several failures of the same kind can occur. In section 2.3.3 an algorithm is presented. This algorithm tests the behavior of a process under any number of occurrences of a given failure, especially from the point of view of recoverability. This approach can be generalized to study the case of any number of occurrences of several distrinct failures.

The presented method allows one not only to know if a process is recoverable under failures, but also to know what are the possible sequences of events under those failures. As shown in the following section, this knowledge is important for the study of the interaction between processes in the case of failures.

In the next section, the behavior of processes and failures is analyzed in a more complex environment of processes relations. The tools developed in this section will be used.

3. FAILURES IN INTERACTING PROCESSES

In real systems, usually the relations between the processes is much more complex than the presented in the previous section. In general, processes are interpreted (or executed) by processors (real or virtual). These processors are also processes interpreted by processors of a lower level; and so on. It means, processes are organized in an hierarchical structure of layers. In the examples of section 2, only one level of this structure is represented. Figure 7, for example, describes in general the mechanism that process P9 uses to transfer the message M from process P7 to process P8. The message is represented as a closed unit (a token in M) and no reference is made about its content. The handling of this content (the encoded data) is a higher level in the structure of the processes. The lower level process (figure 7) executes the higher level process, it transfers the encoded data from P7 to P8.

In general, there exists interactions between processes at different levels, as well as processes at the same level. In this structure of layers and interactions, what are the effects of a failure in one (or more) process to the behavior of other processes? A partial answer to this question is given in this section.

A simplified version of the approach proposed by Gostelow-van Weert [GOST1] is used as a basis for the modeling of the processes' layers. In this approach, processes are encoded in its processors (by means of nodes and tokens) and the processor is in charge of the execution of the encoded process.

The proposed method for the study of recoverability is introduced by the two following examples.

3.1 EXAMPLE

Suppose that a condition of a process is encoded in its processor as a pair of nodes. Figure 9(a) shows the condition I when it is true (a token is in I)

and figure 9(b) shows I when it is false (a token is in $\overline{I}$).

In figure 10, four interconnected processes are described. Processes P12 and P13 represent a protocol connecting between P10 and P11. This system is a variation of the system shown in figure 7. In figure 10, P13 is the encoding of certain levels of details of the content of the message. S is the encoding of a condition or property of the message (say checksum, parity or other). S is transmitted by P12 to S1 under control of R and C (R and C represent the node C of figure 7). Note the difference between M that represents the message and S or S1 that represents a condition of the content of the message.

In this particular example, S has to be always true (for example, the checksum has to be always true). If S1 is true G will receive a token and the acknowledge is sent (K). If S1 is false (because of a failure in the transmission), $\overline{G}$ receives a token and the system will wait for retransmission. Note that S has to be set to "true" (by P10) before the message is sent.

Figure 11 describes the ETM of this system (assuming initial conditions ABS) when a failure may occur in M (a token is lost) or in the condition S1 ($\overline{S1}$ receives a token instead of S1). The ETM shows that the system is recoverable from any number of failures in M or in S1. If one of these failures occur, the system returns to the initial state and a retransmission will take place.

3.2 EXAMPLE

In this section, the previous example is generalized. Figure 12 shows an extension of the system described in figure 10 and figure 15 shows a schematic diagram of the processes and its sharing areas. In this example, the transmitted data is represented by a vector of conditions ($\underline{I}$ in the sender and $\underline{J}$ in the receiver). Each element of the vector may represent a data item (a bit) or a condition that the message has to fit (for example true checksum). The "condition checker" in figure 12 represnts a Petri net; its function is to test the conditions $\underline{J}$ and to transfer the result (by a token in G or $\overline{G}$) to P12.

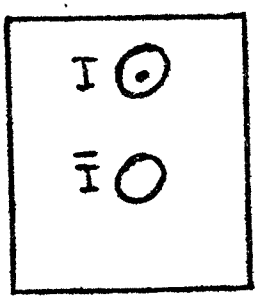

Figure 9(a): encoding of binary variable I="1"

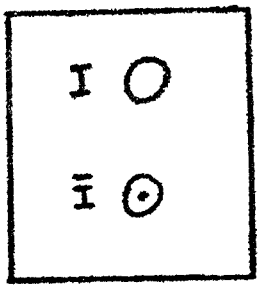

Figure 9(b): encoding of the binary variable I="0"

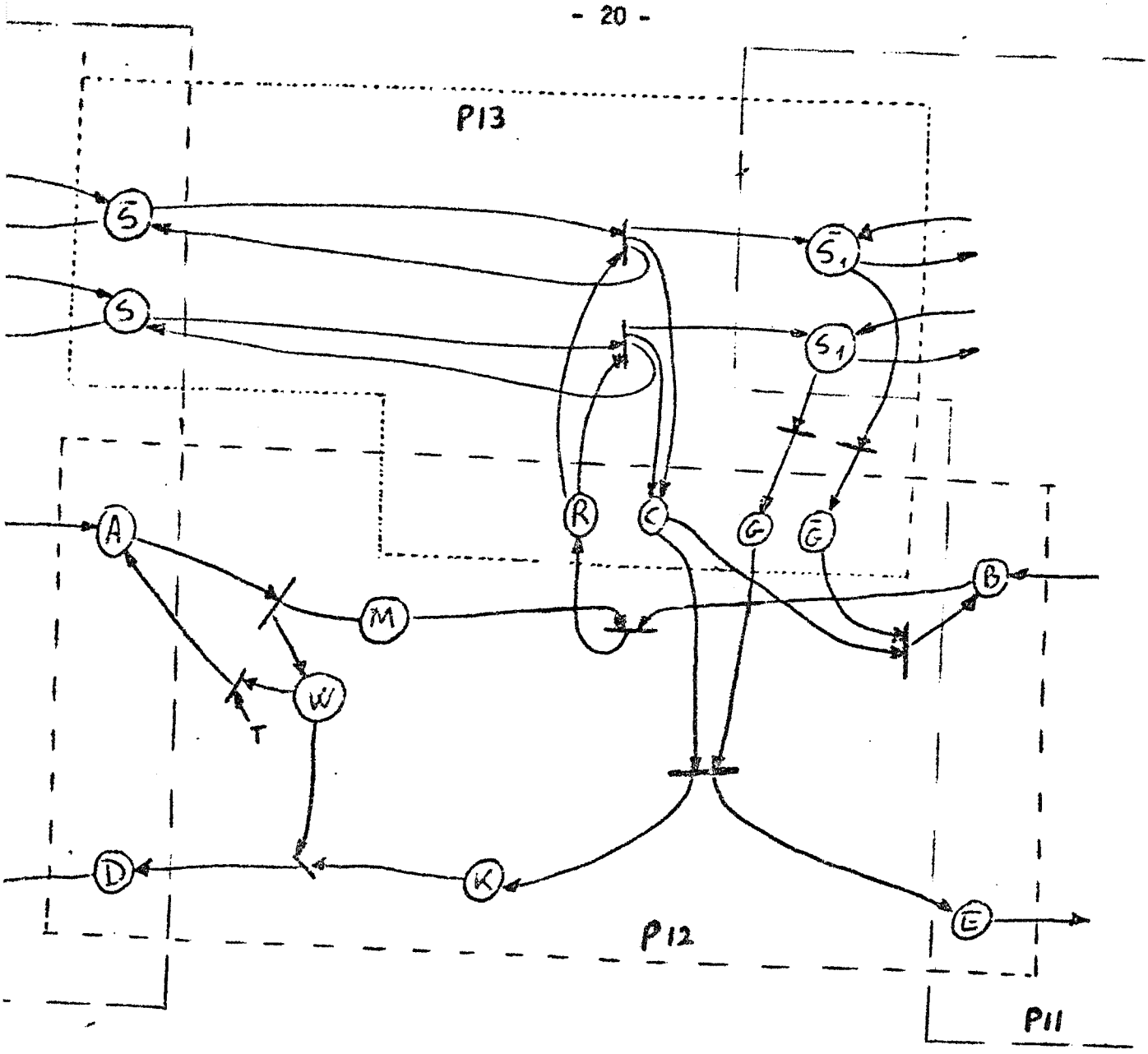

Figure 10. Petri-net of example 3.1

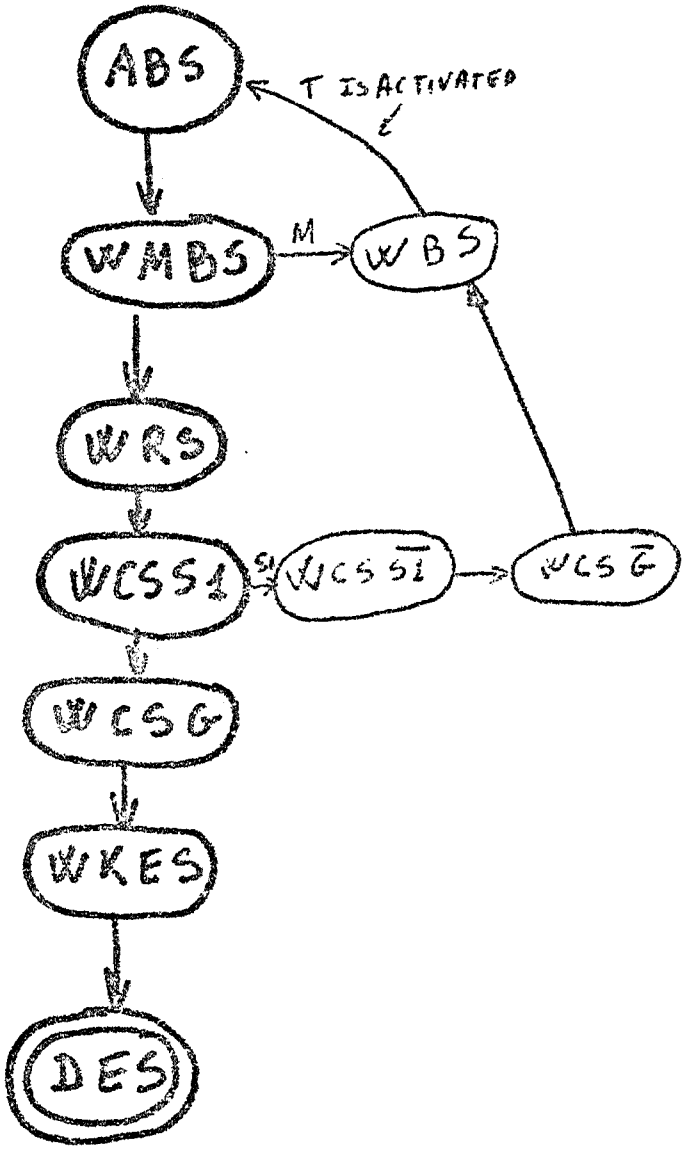

Figure 11: ETM of example 3.1

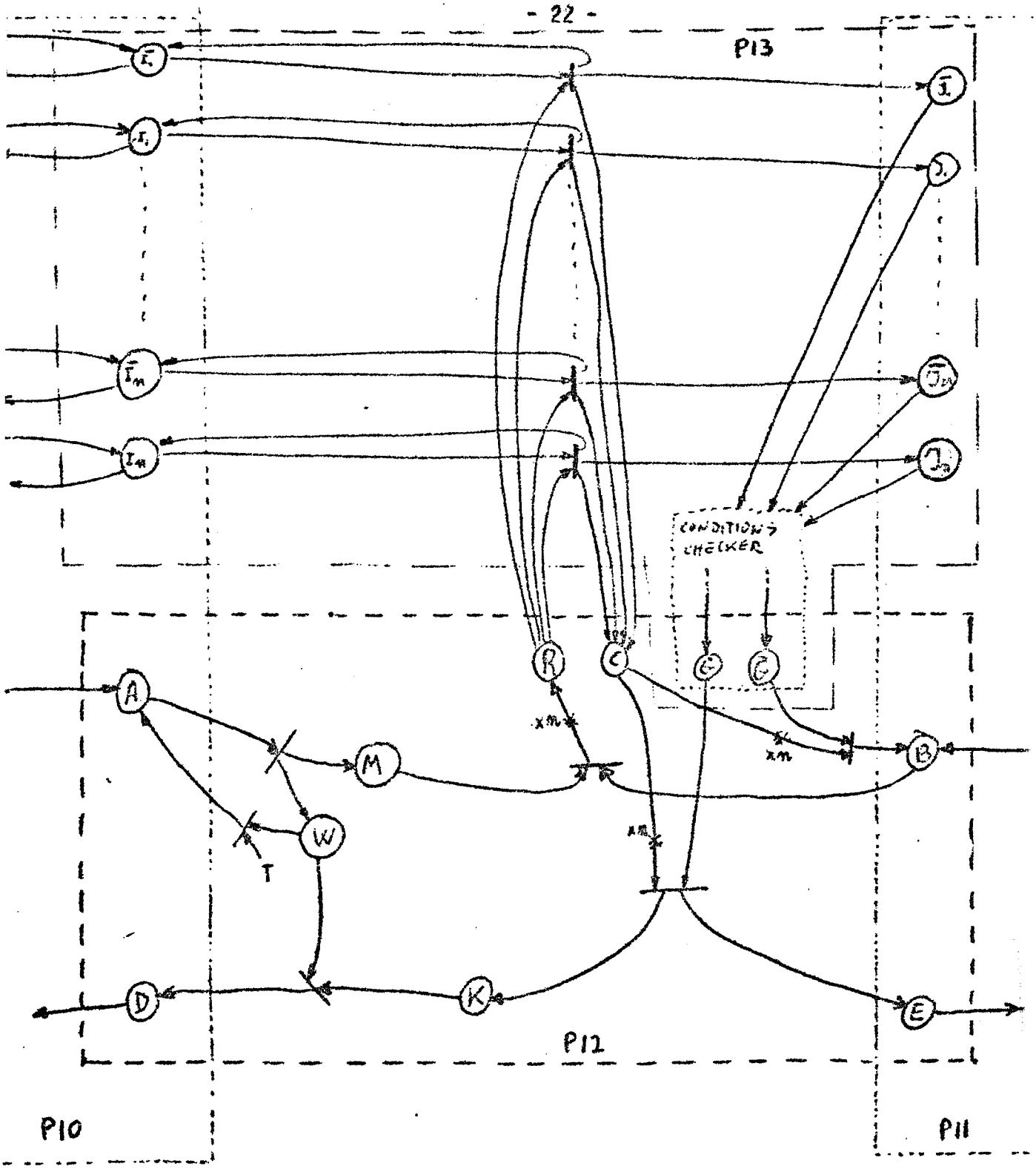

Figure 12: Petri-net of example 3.2

$\times n$ means n parallel arcs

he "condition checker" can evaluate any binary function of the variables J. his function may be unchangeable (if the "condition checker" is a low level rocess) or variable (if the "condition checker" is encoded in a low level rocessor and this processor is enabled to change the encoding). As an example f unchangeable function, the condition checker can be designed to test the arity of the vector J. A variable function is when another process decides hat test the condition checker will execute and changes the encoding appropriately.

Figure 13 shows a symbolic representation of the ETM of the system for the ase when an error may occur in M or in the received conditions J. This is symbolic representation because the complete ETM has different branches for ach different value of the vector I. It means, the ETM of figure 13 is a unction of I. In the same way, there has to be different branches for different rrors in J; but the effects of all the errors are functionally similar so that e choose to represent the failures of J in general, as a vector J*.

Note that the transition from state $WR^n\underline{I}$ to $WC^n\underline{IJ}$ is not effected in one tep. Furthermore, there are several paths in which this transition can take lace; the different paths are dependent in the order that the tokens from I rrive to J. This characteristic is represented in figure 13 by a star marking his transition as a "module". In figure 13 there are other modular ransitions, for example, the transition from $WC^n\underline{IJ}$ to $WC^nG\underline{I}$ that represents he execution of the "condition checker".

The modularity introduced with the division of the system into its rocesses, as well as the use of modular transitions in the ETM, and the symbolic epresentation of variables (I,J and J*) of the higher level simplify the nderstanding of the system behavior.

Figure 13 shows that the system is recoverable from failures in M and in J. nder these failures, the system returns to its initial condition and starts its peration again.

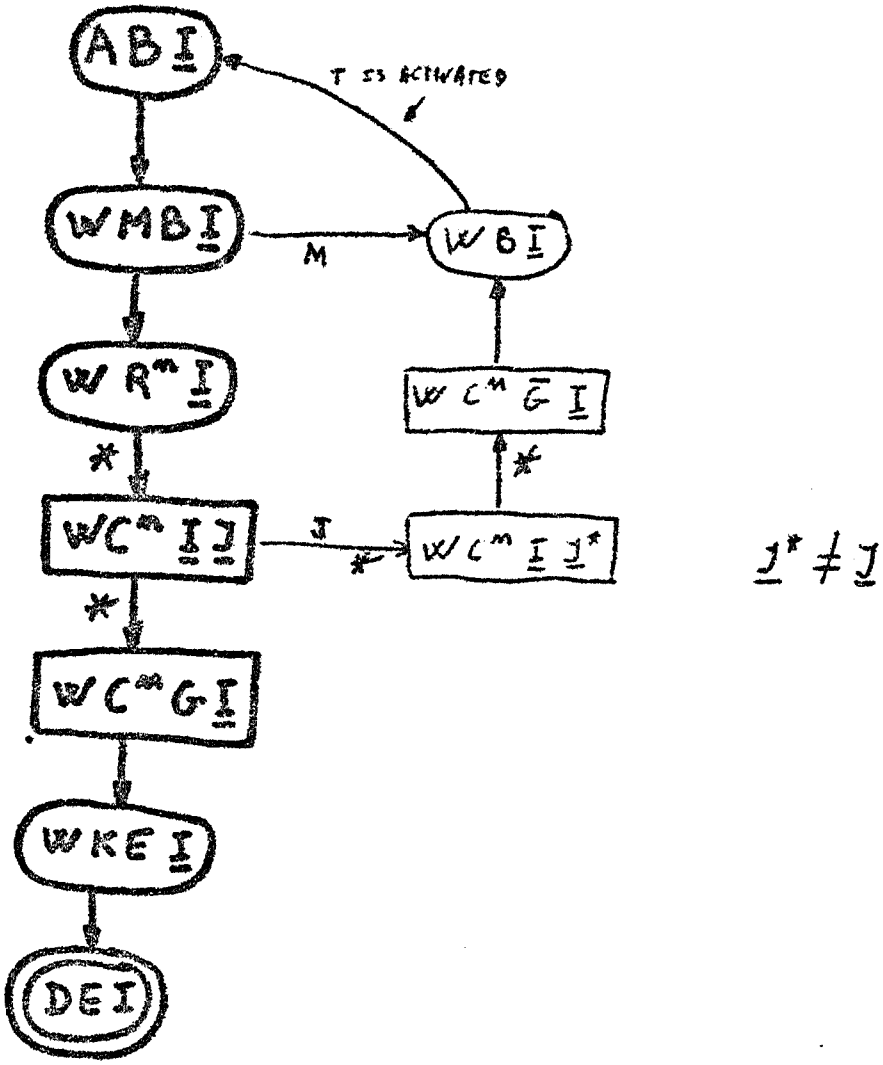

Figure 13: ETM of example 3.2

Figure 14 shows the ETM for process P13 for the case that a failure in J may occur. Process P13 is not recoverable from these failures.

In this example, there exists a very interesting interaction between the processes. A failure in a not recoverable process (P13) is not spread to the other processes in the system. The failure is detected by the condition checker and the control process P12 is notified. P12 is in charge of the recovery of the system in case of failure. The result is a recoverable system as shown in the ETM of figure 13.

This example shows a way of analyzing a system of processes in case of possible failures, and points out that in a well structured system, the system can defend itself against failures in part of its processes.

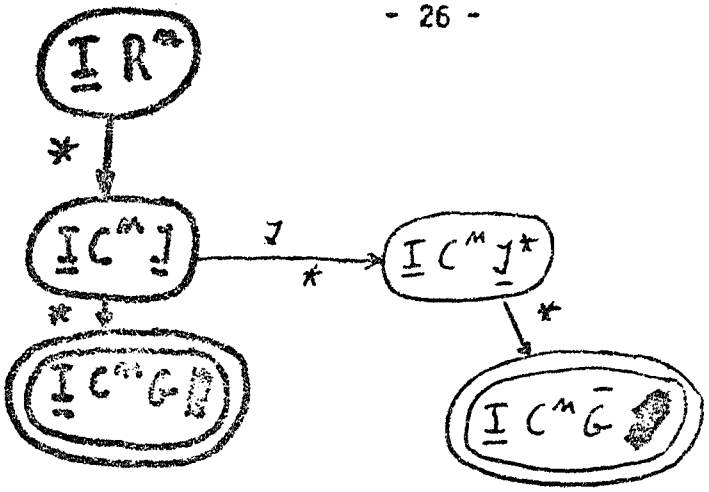

Figure 14: ETM of process P13

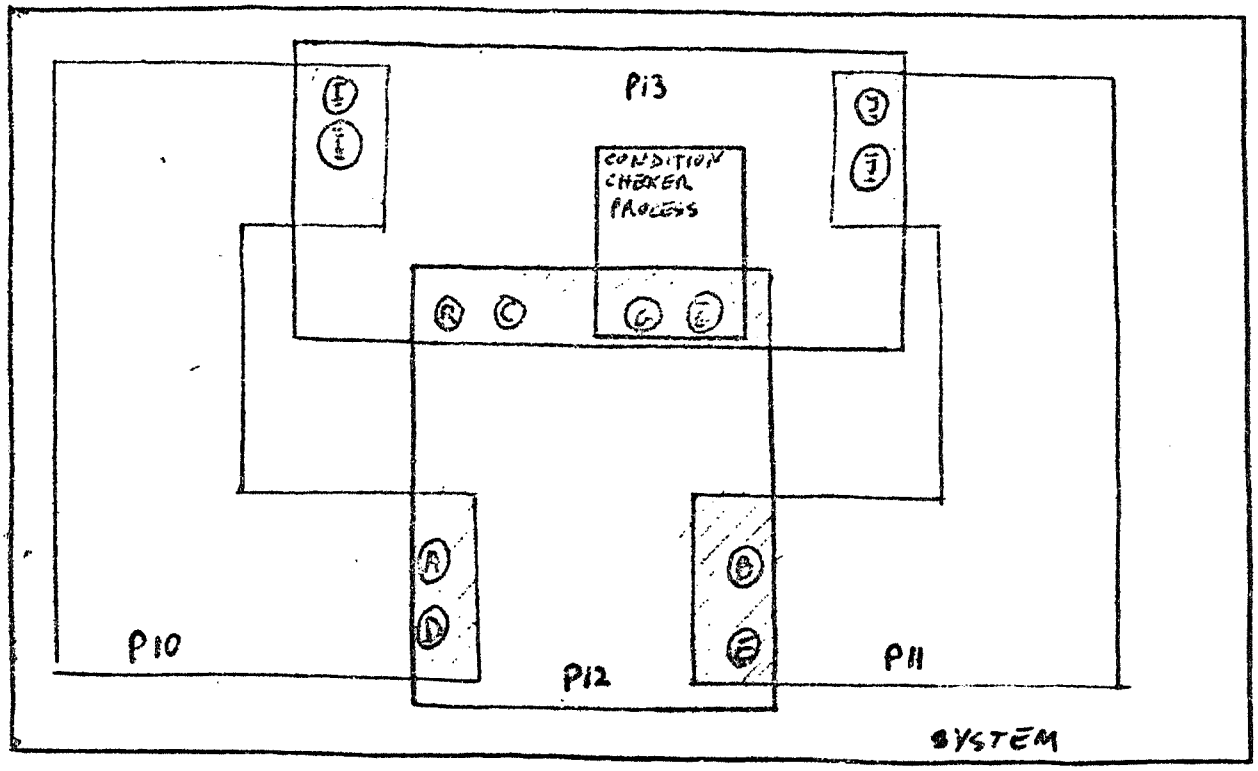

Figure 15: Diagram of processes in example 3.2

CERF — Cerf, V.G. Multiprocessors, Semaphores, and a Graph Model of Computation, Ph.D. Dissertation, ENG-7223, Computer Science Department, University of California, Los Angeles, April 1972 (UCLA-10P14-110).

DENNING — Denning, P.G. "Third Generation Computer Systems. Computer Surveys, Vol 3, No. 4, December 1971.

DENNIS — Dennis, J.B. and E.C. Van Horn. "Programming Semantic of Multiprogrammed Computations". Comm. ACM 9, 3 (March 1966), 143-155.

ESTR — Estrin, G. and R. Turn. "Automatic Assignment of Computations in a Variable Structure Computer System", IEEE Transactions on Computers, EC-12:756-773, December 1963

FALK — Falk, Howard. "Data Communications", IEEE Spectrum, January 1974, Pages 36-39.

FARB — Rowe, L.A., Hopwood, M.D. and Farber, D. J. "Software Methods for Achieving Fail-Soft Behavior in the Distributed Computing System", Record 1973 IEEE Symposium on Computer Software Reliability.

GOST — Gostelow, K.P. Flow of Control, Resource Allocation, and the Proper Termination of Programs, Ph.D. Dissertation, ENG-7179, Computer Science Department, University of California. Los Angeles, December 1971 (UCLA-10P14-106).

GOSTI — Gostelow, K.P. and T.J. van Weert. "Processes and Networks". Stichting Academic of Rekencentuin Amsterdam - Rekencentriun Rijkjuniversiteit Groningen, (The Netherlands), January 14, 1974.

HOLT — Hold,, A.W., H. Saint, R.M. Shapiro, and S. Warshall. "Final Report for the Information System Theory Project", Rome Air Development Center, Applied Data Researdh, Inc., New York, contract AF 30(602) -4211, 1968

HORN — Van Horn, E.C. "Computer Design for Asynchronously reproducible Multiprocessing". MIT Project MAC, Report MAC-TR-39, 1966.

KARP — Karp, R.M. and R.E. Miller. "Parallel Program Schemata", Journal of Computer and System Sciences, 3(4): 145-195, May 1969.

LARSON — Larson, Kenneth Clair. Computation Graphs, Department of Informatic and Computer Science, University of California, Irvine, Irvine, California, 1974

METC — Metcalfe, Robert. "Packet Communication"; MAC-TR-114; MIT, December 1973

PETR — Petri, C.A. Communication with Automata, Thesis, Dermstadt Institute of Technology, Bonn, Germany, 1962.

OSTEL — Postel, Jonathan Bruce. A Graph Model Analysis of Computer Communications Protocols, Computer Science Department, UCLA Los Angeles, California, 1974