COMPUTATION MODULES AND PETRI NETS*

Kim P. Gostelow
Department of Information & Computer Science
University of California, Irvine

ABSTRACT

Petri-nets are used as a model of processes, and a property of a net called proper termination is defined and discussed. Proper termination is argued to be a useful property which a construct called a "module" should possess. This property assures reentrancy and freedom from deadlock in the net, and a theorem is given concerning the substitution or interchange of modules in a larger environment.

1. INTRODUCTION

My study of theoretical models stems from an interest in system design, whereby when it is shown that some useful properties of a model of, say, "process" behavior can be defined, then it may be advantageous to build these properties into systems from the beginning. That is, if good theories can be found, machines could be designed to fit these theories and hence be more predictable, buildable, efficient, and perhaps elegant.

In this paper, the focus is on moving towards a precise characterization of a module - a unit of hardware or software (or both) which is replaceable, usable as a building block, etc. The problem, of course, is to restrict the allowable actions of a module so as to prevent disagreeable behavior within a larger environment, yet permit enough freedom so as to ensure versatility in module use.

Flow of control is the object of the investigation, a Petri-net is the tool, and proper termination (PT) is the primary ingredient in the characterization of a module. Petri-nets can conveniently model a wide variety of computational tasks (including resource allocation, synchronization, and interprocess communication), and a net which is properly terminating possesses certain useful properties, such as being deadlock-free and reentrant. To demonstrate, theorems concerning "harmonious cooperation"[1-3] and module replacement are given.

2. THE MODEL

2.1 PETRI-NETS

Activity which may occur in the control of a process is described here by means of a Petri-net[4]. An example of a Petri-net is given in

*This research was supported in part by the U.S. Atomic Energy Commission contract no. AT(04-3)-34, PA214, and in part by the National Science Foundation under Grant GJ-1045 (Distributed Computer System Project).

Figure 1 and the net operates as follows: place a

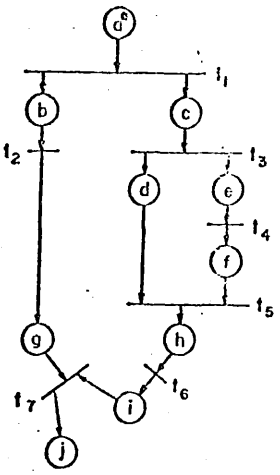

Figure 1
A Petri-Net N

(the circle named "a") contains a token (the dark spot in place a) and place a is the only place referenced as an input place by transition t_1 (there is a directed arc from place a inbranching to the bar named "t_1"); t_1 is enabled since every input place of t_1 holds at least one token, and t_1 may fire whenever desired. When t_1 fires (no other transition in Figure 1 can fire at this point) one token is removed from each input place (place a in this example) and one token is placed on each of the output places b and c (there is a directed arc outbranching from t_1 to each of places b and c). This is the end of one basic operational cycle in the interpretation of the Petri-net of Figure 1. At this point, transitions t_2 and t_3 are both enabled and may fire in an arbitrary and unspecified order. However, for simplicity, we require that only one transition be in the act of firing at any given instant in time. (Note that the asynchronous nature of the computation is thus represented by the absence of any sequencing constraints; for example, between t_2 and t_3.) Once one of the transitions t_2 or t_3 is chosen to fire after t_1, then operation is just as it was for t_1: input place tokens are diminished by one and output place tokens are increased by one. In Figure 1, one possible firing sequence or computation is given by the string of transition firings $<t_1,t_3,t_4,t_5,t_2,t_6,t_7>$, and the sequence of markings (a specification of the number of tokens on each place) generated by the above firing sequence is <a, bc, bde, bdf, bh, gh, gi, j>. The marking "a" in Figure 1 is the initial marking, and at any given point in the operation of a net, the current token configuration is called the current marking. Marking "j" is said to be reachable from marking "a" or from any other marking which may precede it (such as "bh", "bc", etc.). The above "bag of symbols" notation to denote a marking is sometimes more convenient than the formal specification of a marking as a vector, with each component in the vector corresponding to the count of tokens on a given place. Both notations are used in this paper.

2.1.1 Notation - syntax

Definition: A Petri-net is a triple N = ($PLACES_N$, $TRANSITIONS_N$, $qzero_N$) where

- $PLACES_N$ = a finite indexing set of elements called (names of) places
- $TRANSITIONS_N$ = a finite indexing set of elements called (names of) transitions referencing places in $PLACES_N$ as input places and as output places
- $qzero_N$ = initial marking of N, given as a vector with one component of the vector assigned to count the number of tokens on one place.

Since $PLACES_N$ in net N is an indexing set, if the current marking is q and we want to know the number of tokens on place p, just write q_p. Also, it is necessary to define some subsets of $PLACES_N$: $p \in$ INPUT $PLACES_N$ iff p is an input place of some transition $t \in TRANSITIONS_N$; $p \in$ OUTPUT $PLACES_N$ iff p is an output place of some transition $t \in TRANSITIONS_N$; $p \in$ ENTRY $PLACES_N$ iff $p \notin$ OUTPUT $PLACES_N$; $p \in$ EXIT $PLACES_N$ iff $p \notin$ INPUT $PLACES_N$. Let "t references p in N" be a predicate which is true if p is an input place or an output place of transition t in net N. Finally there is a simple syntax to a net which must be stated: every transition must have at least one input place, and refences (directed arcs) are made only by transitions

to places - never between two places or between two transitions.

2.1.2 Notation - semantics

To describe the dynamics of a net, we need only one definition. Let the reachability set $RS_N(q)$ be the set of all markings which can be reached from current marking q on net N.

2.2 PROCESS DESCRIPTION

This section comes, in part, from reference[5] in a slightly different formulation. The purpose is to capture the notion of a process, or rather, of a process description.

2.2.1 Definition of process and subprocess descriptions

Given a net N (Figure 1), some subsets of the places and transitions in N may be partitioned or recognized as significant in their own right for various reasons; for example, Figure 2 recognizes

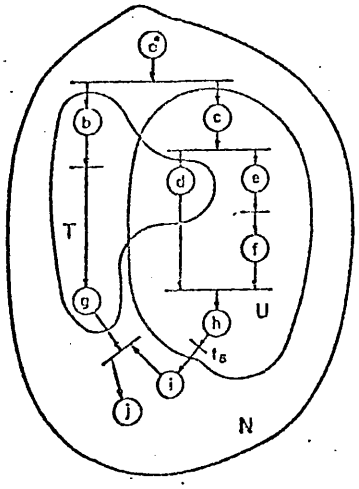

Figure 2

N and T are process descriptions, while U is not a process description

three such collections (N, T, and U) of places and transitions. However, it is not necessarily true that an arbitrary collection of subsets of places and transitions will itself be a Petri-net. For example, T in Figure 2 is a Petri-net whereas U is not a Petri-net (since $t_6 \in TRANSITIONS_U$ and t_6 references i in N, but $i \notin PLACES_U$).

Definition: Let $N = (PLACES_N, TRANSITIONS_N, qzero_N)$ be a Petri-net and let

$PLACES_P \subseteq PLACES_N$

$TRANSITIONS_P \subseteq TRANSITIONS_N$

$(qzero_P)_r = (qzero_N)_r$, for all $r \in PLACES_P$.

Then $P = (PLACES_P, TRANSITIONS_P, qzero_P)$ is a process description if P is a Petri-net (i.e., if $t \in TRANSITIONS_P$ & t references r in $N \Rightarrow r \in PLACES_P$).

That is, if P includes some subset of transitions of N and if every place in N referenced by those transitions is also in the PLACES component of P, then P is a process description. This implies that activity caused by the firing of transitions in P is confined to P. If, however, the action of one process does affect another process, then those processes must share some component of their description. For example, there may be a non-null intersection of the PLACES component of two different process descriptions, indicating at least one place common to both processes. Again in Figure 2, note that N is itself a process description, and T is completely contained within N. In such cases we sometimes say that T is a subprocess description of process description N and write $T \subseteq N$.

2.2.2 Operations on process descriptions

In the following, assume that P and Q are process descriptions and that $TRANSITIONS_P \cap TRANSITIONS_Q = \phi$. Figure 3a shows a pair of process descriptions P and Q which have identical names for two places, while Figure 3b shows process description S which is the result of combining P and Q in a particular way:

Definition: $S = P \cup Q$ is the union process description of process descriptions P and Q if

$TRANSITIONS_S = TRANSITIONS_P \cup TRANSITIONS_Q$

$PLACES_S = PLACES_P \cup PLACES_Q$

$(qzero_S)_r = (qzero_P)_r + (qzero_Q)_r$ for all $r \in PLACES_S$, where $(qzero_P)_r \triangleq 0$ if $r \notin PLACES_P$, and similarly for Q.

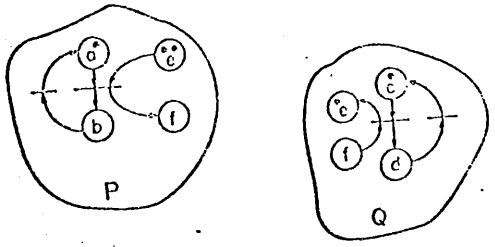

Figure 3a
Two process descriptions P and Q

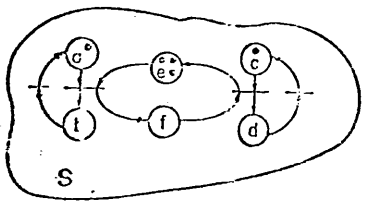

Figure 3b
S = P ∪ Q is the
union process description of P and Q

That is, S is composed of the transitions of P and of Q, the places of P and of Q, and any tokens on a place p in P or in Q will appear in S. Note that if $p \in PLACES_P \cap PLACES_Q$, the p appears only once in S, p is a place shared by P and Q, and p holds tokens equal in number to the sum of tokens on p from P and Q. (Recall that shared transitions are not considered here.) Given that P and Q are process descriptions, it is easy to show that P ∪ Q is indeed a process description.

In Figure 4a, S and $P \subseteq S$ are process descriptions. Removal of P from S leaves Q in Figure 4b called the <u>difference</u> of S and P.

<u>Definition</u>: Q = S - P is the <u>difference process description</u> of process descriptions S and P if

$TRANSITIONS_Q = TRANSITIONS_S - TRANSITIONS_P$

$PLACES_Q = (PLACES_S - PLACES_P)$
$\cup \{p \in PLACES_S \mid t \in TRANSITIONS_Q$
$\wedge$ t <u>references</u> p <u>in</u> S$\}$

$(qzero_Q)_r = (qzero_S)_r - (qzero_P)_r \geq 0$ for all $r \in PLACES_Q$.

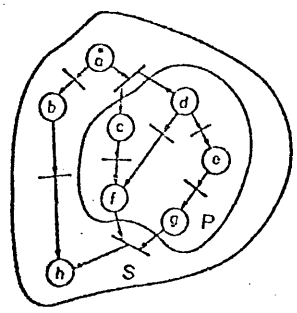

Figure 4a
S and $P \subseteq S$ are process descriptions

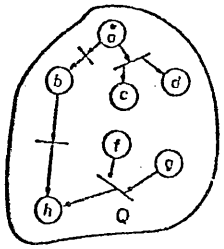

Figure 4b
Q=S-P is the difference process description

Also, the set of places common to two process descriptions P-Q and Q is called <u>ATTACHMENT PLACES</u>. In Figure 3, P and Q have e and f in common and these are the ATTACHMENT PLACES of P to Q; places c, d, f, and g in Figure 4 are the ATTACHMENT PLACES of Q=S-P to P.

3. <u>PROPER TERMINATION</u>

3.1 WHAT IT IS AND WHY

Proper termination (introduced in [6] and generalized here) is a property which a process

description may or may not possess. Informally speaking, the process description of Figure 1 is properly terminating because there is a bound on the number of tokens which it will hold at any point, and because it is always possible to reach the "end" of the net (represented by place j) where the "end" in no way feeds any tokens to other portions of the net. Such a process description, as with all properly terminating process descriptions, is "well-behaved" or "structured" in the sense of an asynchronous system. For example, Figure 5 is a process description which is not properly terminating because the token count on place b is unbounded.

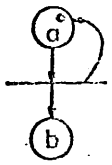

Figure 5

Place b is unbounded, and this process description is not properly terminating

Such behavior is not considered proper. S in Figure 4a is not properly terminating since the marking "ff" is reachable from $qzero_S$, but marking "ff" cannot reach the "end" of the net (place h). Lastly, consider Figure 4a with new transitions t_1, t_2 and place i as shown in Figure 6. This process description (which is not properly terminating for the same reasons as Figure 4a) demonstrates that "isolated regions" are the reason that the end of the net cannot be reached, where the isolated region of Figure 6 is the set of markings {ff, if, ii} which alternate among one another. Once an isolated region has been entered, it is not possible to leave it. Note that a terminal marking, such as ff in Figure 4a, is actually a special case of an isolated region.

Definition: $I \subseteq RS_p(qzero_p)$ is an isolated region (sometimes called a knot) of a process description P if $\forall q' \in I \Rightarrow RS_p(q') = I$ and I is noncempty.

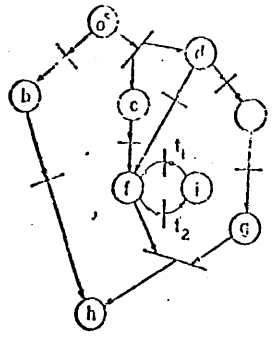

Figure 6

Places f and i contributed to form an isolated region with the given initial marking

Thus I = {ff, if, ii} is an isolated region of Figure 6 since every marking in I can reach every other marking in I. Again, once a marking gets into I, it cannot get out.

Definition: Process description P is properly terminating (PT) if

(1) $|\ RS_p(qzero_p)\ |$ is finite,

and (2) If I is any isolated region in $RS_p(qzero_p)$, then for any $q \in I$: $q_p \neq 0 \Rightarrow p \in$ EXIT PLACES$_p$.

That is, the only isolated regions allowed are singleton isolated regions (e.g., I = {j} in Figure 2) which are therefore terminal markings, and furthermore, any place with a token in such a marking must not be an input place to any transition in that process description.

Theorem: PT is decidable for any process description.

Proof - Keller[7] has extended Karp & Miller's[8] reachability tree work to include Petri-nets, and has shown that finiteness of the reachability set is decidable, and thus, condition (1) of PT. Given that the reachability set is finite, it is possible to enumerate the isolated regions and inspect the markings for condition (2) of PT. □

Also, let $PT_p(q) \subseteq RS_p(q)$ be the set of isolated regions in $RS_p(q)$ if P is a PT process description,

and let $PT_P(q) = \phi$ if P is not PT. Thus, $PT_P(q)$ is a set of sets, where the latter sets are each an isolated region in $RS_P(q)$ when P is PT with initial marking q, but $PT_P(q)$ is empty if P is not PT. Note that a process description P may be PT with initial marking q, but not PT with initial marking $q' \neq q$.

3.2 RESOURCE ALLOCATION, DEADLOCK, AND MODULARITY

Complex forms of resource allocation are easily modelled by Petri-nets. Figure 7 shows an

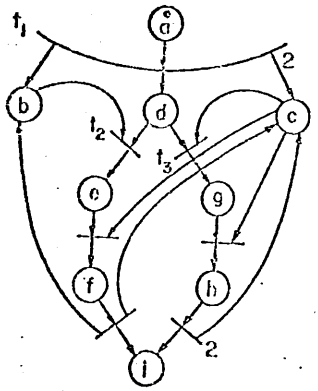

Figure 7
Resource allocation with Petri-nets

example where places b and c each represent a resource to be allocated and returned in one of two possible ways depending upon a decision made by transition t_2 or t_3. (The small numeral 2 in the figure next to a directed arc means two tokens are implied by actions directed along that arc - thus, two tokens are placed on place c by firing t_1). In general, allocation dependencies and conditions far beyond any present computer system's capabilities can be represented. And with this capability comes the problem of deadlock, or rather, how to ensure freedom from deadlock and obtain "harmonious cooperation" in the words of Habermann and Dijkstra[1-3].

Habermann[9], Holt[3], and others have developed systems to avoid or detect deadlock - that locking condition due to unfortunate scheduling of requests for resources which causes a system to come to a grinding halt. However, these previous studies have embedded their systems in comparatively strict environments. For example, no alternative requests, conditionals, or variable paths were allowed, and single initial maximum resource use had to be stated with a promise never to exceed that maximum. These assumptions allowed a fairly straight-forward and simple definition of "deadlock" itself. However with the power of Petri-nets to express much more sophisticated resource systems, the definition of just what constitutes "deadlock" becomes a list of all the ways in which control fails to behave "properly". It was apparent that there was no purpose to this approach, largely because the problem was not really confined simply to "resources" but to "flow of control" in general. This is especially clear when one realizes that in a Petri-net representing process/resource interaction, there is no distinction between resources and program control conditions; they are identical. Since "harmonious cooperation" or "proper flow" was the original goal, this should be the case regardless of the presence or not of "resource allocation" specifications.

Proper termination fills the need. Thus, rather than define some rather complex condition called "deadlock" and show that it does not occur, the approach here is to show that proper termination guarantees harmonious cooperation - the original goal. Harmonious cooperation of a finite number of process descriptions $P_1, P_2, \ldots P_n$, as defined by Habermann and Dijkstra[1,2], means that the system $P_1 \cup P_2 \cup \ldots \cup P_n$ will complete if for each P_i:

(1) only a finite number of firings of transitions in P_i are necessary for P_i to complete,

and (2) once a transition in P_i becomes enabled, either it becomes disabled due to the firing of other transitions, or it fires within a finite period of time (the finite delay property[8]).

Lemma: Given a finite system of process descriptions P_1, ..., P_n and the above conditions, if the system $P_1 \cup P_2 \cup \ldots \cup P_n$ is PT then there is harmonious cooperation.

Proof - Let P be the union process description of P_1, ..., P_n. Now, if all enabled transitions eventually fire, system behavior must imply either

(a) entering a loop and never selecting the exit (infinite loop),

(b) entering an infinite path of distinct markings in the reachability set,

or (c) entering a finite isolated region.

But (a) is not possible by conditions (1) and (2) of harmonious cooperation, and PT implies that (b) cannot occur, so (c) must be the case. But, PT implies that the only isolated regions which are possible are those which are also terminal markings. These terminal markings are composed only of EXIT PLACES, so no transition can be partially enabled yet wait forever to be fired (i.e., hung-up). Thus, the system completes. □

The converse is not true only because, within the language of Petri-nets, there is no way to distinguish a control condition from a resource. Thus, some terminal marking could simply represent a control hang-up as opposed to a deadlock of resources, in which case strict adherence to the definition of "harmonious cooperation" would allow such a marking. But since this fact (that the hang-up is due to control and not due to resources) cannot be represented with Petri-nets, there is no way to distinguish it from the case of hang-up due to resource allocations. However, the question also arises, should such a distinction be made? I think not since hang-up due to resources or hang-up due to control is still hang-up, but rather than change the definition of "harmonious cooperation" the lemma is allowed to remain as above.

The condition of PT also allows detection of such cases as processes requesting more resources than are available, since it appears in a Petri-net simply as a control hang-up condition. PT unifies the problem. I will remark here that some work on rapid decision of the PT condition has been investigated[6] and a significant extension of that work is to be reported on shortly.

4. MODULARITY

Modularity becomes important because it is nice to build upon the work of others (utilize their modules), and because our only really useful method of solving large problems is to break the problem into several smaller ones, solve each of these, and then combine the solutions.

The approach taken here is subprocess replacement. For example, Figure 8a gives a PT system P containing subprocess description $Q \subseteq P$, and the point is to replace Q with R (Figure 8b), obtain

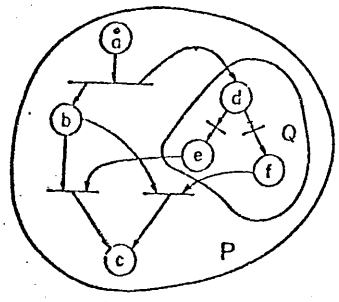

Figure 8a

Process descriptions P and $Q \subseteq P$

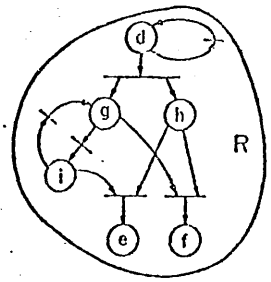

Figure 8b

Process description R

$S = (P-Q) \cup R$ (Figure 8c), and determine under what conditions we can be sure that S will behave just like P behaves for any process descriptions P, $Q \subseteq P$, and R.

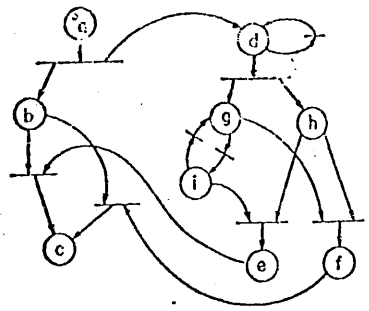

Figure 8c

Process description S

In general, to achieve this equivalence in behavior between P and S, Q and R (considered as completely isolated systems) must have identical input-output terminal characteristics where the terminals are the ATTACHMENT PLACES. This is done by requiring Q and R, each in a stand-alone environment, to be identically PT. Furthermore, Q and R must behave identically when they are placed into the environment P-Q. This is equivalent to saying that the subsystems Q and R must behave identically "under load". This latter condition is assured by local structural conditions called "proper substitution", and the fact that P and Q are PT.

In the following, let P, Q $\subseteq$ P, and R be process descriptions, and assume

$\text{TRANSITIONS}_{P-Q} \cap \text{TRANSITIONS}_{R} = \phi$.

Definition: R is properly substituted for Q in P to form S = (P-Q) $\cup$ R if

(1) $\text{qzero}_{Q} = 0$, and $\text{qzero}_{R} = 0$,

(2) $p \in \text{INPUT PLACES}_{Q} \Rightarrow p \notin \text{INPUT PLACES}_{P-Q}$,
$p \in \text{INPUT PLACES}_{R} \Rightarrow p \notin \text{INPUT PLACES}_{P-Q}$,

and (3) the following relation holds for the ATTACHMENT PLACES:
$\text{ATTACHMENT PLACES} = \text{PLACES}_{Q} \cap \text{PLACES}_{R} = \text{PLACES}_{P-Q} \cap \text{PLACES}_{Q} = \text{PLACES}_{P-Q} \cap \text{PLACES}_{R}$.

Theorem: Let R be properly substituted for Q in P to give S = (P-Q) $\cup$ R, and let q be any marking with tokens only on places in ATTACHMENT PLACES. If ($\forall$q) (Q and R are PT with initial marking q, and furthermore, $PT_{Q}(q) = PT_{R}(q)$), then $PT_{P}(\text{qzero}) = PT_{S}(\text{qzero})$ where qzero is restricted to tokens only on places in PLACES_{P-Q}.

Proof - (schema) - By the PT hypothesis, Q and R have identical PT behavior at the ATTACHMENT PLACES (external terminals) when in an isolated environment. Hence, to ensure their identical behavior in the environment P-Q, we need only respect that isolation.

(A) Q and R can receive and give tokens in P-Q only through the ATTACHMENT PLACES, either as initial tokens or due to firings in P-Q, by condition (1) of proper substitution and by the restriction of qzero to PLACES_{P-Q} in the theorem statement. Please see Figure 9.

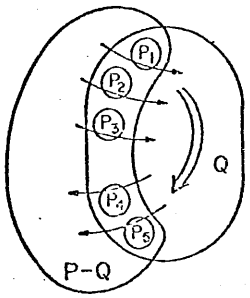

Figure 9

Q(R) receive and give tokens only via the p_i in ATTACHMENT PLACES

1) With respect to Q: ATTACHMENT PLACES = $\text{PLACES}_{P-Q} \cap \text{PLACES}_{Q}$ are the only elements in common between P-Q (condition (3) of proper substitution). Furthermore, each place p $\in$ ATTACHMENT PLACES is used uni-directionally by condition (2) of proper substitution. That is, P-Q and Q are behaviorally isolated except at the ATTACHMENT PLACES. Thus, the behavior of Q in P-Q is completely characterized by the $PT_{Q}(q)$ hypothesis for all possible inputs from P-Q.

2) With respect to R: R's behavior in the environment P-Q is identical to its behavior in isolation, for the same reasons as for Q above.

3) With respect to P-Q: P-Q receives and gives tokens to Q and R only via ATTACHMENT PLACES, and Q and R have no initial tokens by condition (1)

of proper substitution. By conditions (2) and (3) of proper substitution, P-Q is behaviorally isolated from Q and R.

(B) Due to the behavioral isolation demonstrated in part A of P-Q to Q (R), and the PT hypothesis of Q(R), tokens input to Q (R) at the ATTACHMENT PLACES from P-Q must output from Q (R) at ATTACHMENT PLACES. But the PT equivalence of Q and R requires that P-Q see no distinction in reachability from input to Q and output from Q, to that of R.

(C) Now for the PT conditions of P and S:

Condition (1) - The identical PT behavior of Q and R in P-Q assures that $|RS_P(qzero)|$ is finite $\Leftrightarrow$ $|RS_S(qzero)|$ is finite.

Condition (2) - Now, I_P is an isolated region in $RS_P(qzero)$ iff

Case 1) There is no traversal through Q (R), and thus $I_P = I_S$ in S.

Case 2) There is traversal through Q $\Leftrightarrow$ there is traversal through R $\Leftrightarrow$ I_S exists in S for the same reasons of reachability that I_P exists in P.

In either case, I_S is in S. Now, the identical PT behavior of Q and R implies that EXIT PLACES$_Q$ = EXIT PLACES$_R$, and thus EXIT PLACES$_P$ = EXIT PLACES$_S$. Finally, $q \in I_P$ and $q_p \neq 0$ only if $p \in$ EXIT PLACES$_P$ $\Leftrightarrow$ $q' \in I_S$ and $q'_p \neq 0$ only if $p \in$ EXIT PLACES$_S$, since the presence of Q or R has no effect on such terminal markings (Q and R never retain any tokens internally due to their PT behavior). □

5. CONCLUSIONS

Petri-nets were used here as a model of asynchronous system operation, and a condition of control flow called proper termination was described. This property constrains activity to a point where such systems are free of deadlock and can be used as replaceable subsystems, as shown in the paper. However, proper termination also allows a wide latitude in behavior, and I feel that the appropriate line between module versatility and restrictive behavior lies near that of proper termination.

6. ACKNOWLEDGEMENTS

V. Cerf, G. Estrin, and S. Volansky contributed to early work in proper termination concerning its definition and decision algorithms, and T. van Weert contributed to the formulation of process description. Also, thanks to Wil Plouffe for his reading of the paper, and to Peg Gray for her typing.

7. REFERENCES

[1] Habermann, A.N. On the Harmonious Cooperation of Abstract Machines, Ph.D. Thesis, Mathematics Dept., Technische Hogeschool Eindhoven, Eindhoven, The Netherlands, 1967.

[2] Dijkstra, E. "Cooperating Sequential Processes" in Programming Languages, F. Genuys, ed., Academic Press, New York, 1968.

[3] Holt, R.C. On Deadlock in Computer Systems, Technical Report CSRG-6, University of Toronto, April, 1971.

[4] Baer, J.-L. "A Survey of Some Theoretical Aspects of Multiprocessing", Computing Surveys, 5,1 pp. 31-80, 1973.

[5] Gostelow, K.P. and T.J. van Weert Processes and Networks, Report from Stichting Academisch Rekencentrum Amsterdam/Rekencentrum Rijksuniversiteit Groningen, The Netherlands.

[6] Gostelow, K.P. and V.G. Cerf, S. Volansky, G. Estrin "Proper Termination of Flow of Control in Programs Involving Concurrent Processes", Proceedings ACM National Conference, August, 1972.

[7] Keller, R.M. Vector Replacement Systems: A Formalism for Modeling Asynchronous Systems, TR 117, Dept. of EECS, Princeton University, January, 1974.

[8] Karp, R.M. and R.E. Miller "Parallel Program Schemata", J. of Computer & System Sciences, 3, 2, pp. 147-195, May, 1969.

[9] Habermann, A.N. "Prevention of System Deadlocks" CACM, 12, 7, pp. 373-377, July, 1969.