SYNCHRONIZATION PRIMITIVES

Richard J. Lipton

Computer Science Department
10 Hillhouse
Yale University
New Haven, Connecticut 06520

LIMITATIONS OF SYNCHRONIZATION PRIMITIVES*

Richard J. Lipton

Computer Science Department
Yale University
New Haven, Connecticut 06520

*A preliminary version of this paper appeared in the Sixth Annual ACM Symposium on Theory of Computing, April 1974.

ABSTRACT

A mathematical model of the process concept is presented. This model can represent a wide variety of synchronization primitives. The study of synchronization primitives is then reduced to the study of relations between systems of processes. For one relation--simulate--it is possible to show that there are differences between several synchronization primitives. The relation simulate can be justified, for weakening it leads to synchronization anomalies.

1. Introduction

One of the basic issues in computer science is: What is the relative power of computing devices? The particular question addressed here is: What is the relative power of parallel or asynchronous models of computation? This question has been largely neglected in the literature. While there has been much careful study of parallel models (i.e., Karp and Miller [6]), these studies have dealt exclusively with properties such as "determinacy" rather than the comparative power of parallel models. Peterson [13] has shown that many models are equivalent. But his notion of equivalence — a straightforward generalization of the notion of equivalence of grammars in automata theory — is unsatisfactory in that it is too weak. Patil [12] and Kosaraju [8] both claim to have "proved" that there are differences among certain parallel models. But they only state informally what a "synchronization problem" is; hence, their proofs cannot be considered rigorous.

In this paper the power of four parallel models is investigated. These models are: PV (Dijkstra [3]), PVchunk (Vantilborgh and van Lamsweerde [14]), PVmultiple (Patil [12]), up/down (Wodon [15]). Since these models are motivated by operating systems, they are often called "synchronization primitives" instead of parallel models. In order to compare these synchronization primitives it is necessary to define precisely what they are. The definition used is the one contained in Lipton [9]; it is a slight modification of Lipton [10]. A very large class of synchronization primitives can fit into this framework. Next it is necessary to define precisely what a "synchronization problem" is. The key idea here is that the fuzzy concept of synchronization problem is replaced by the precise concept of "simulate." Simulate is a relation between parallel systems of processes.

Thus, intuitively the system of processes Q simulates the system of processes P provided the "behavior" of P is faithfully represented in the "behavior" of Q. Mathematically Q simulates P provided there is a morphism from Q to P with certain required properties. Simulate is justified on both theoretical and practical grounds. It cannot be weakened without allowing pathological and inefficient solutions to exist. Simulate is used to compare synchronization primitives as follows: Say synchronization primitive C_1 → synchronization primitive C_2 provided there is some system of processes in C_1 that cannot be simulated by any system of processes from C_2. Intuitively $C_1 \rightarrow C_2$ means that C_1 can "solve" a synchronization problem that C_2 cannot. The principal results are displayed in figure 1. For example, up/down → PV means that up/down can solve a synchronization problem that PV cannot. The relations PV → PVchunk, PV → PVmultiple, and PV → up/down are all false; for PVchunk, PVmultiple, and up/down are generalizations of PV. The relation PVchunk → PVmultiple, however, is still an open question.

The differences displayed in figure 1 are all proved in a uniform way based on the invariance theorem. Roughly this theorem states that

> if Q simulates P, then all the "local behavior" of P is also the local behavior of Q.

Local behavior is a generalization of the concept of "conflict" as used in Petri Nets and Parallel Program Schemata (Holt [5] and Keller [7]). For instance, the synchronization primitive up/down contains the behavior:

> "process-1 can stop process-2 but process-2 cannot stop process-1,"

while the synchronization primitive PV does not contain this asymmetric behavior. It then follows by the invariance theorem that

$$up/down \rightarrow PV.$$

This is a non-trivial result, for simulate is not too strong a relation. In particular if Q simulates P, then Q can contain "bookkeeping steps" (i.e., steps that are not present in P). Therefore a single step in P can be split into many steps in Q. Moreover, Q and P can contain conditional branching and program variables; this is not allowed in Patil [12] or in Kosaraju [9].

2. Preliminaries

A system of processes is a collection of actions and a collection of states. Each state is assumed to be of the form

$$(L_1, \ldots, L_n, G)$$

where n is the number of processes in the system. The $L_1, \ldots, L_n$ each represent a program counter--one for each process. G represents both the program variables and semaphore variables. There is no loss in generality in assuming that G contains both program variables and semaphores, for no assumptions have yet been made about the structure of G. For example, G might be (x,y,S) where x and y are program variables and S is a semaphore. The program variables play exactly the same role they play in sequential processes. Since conditional branching is allowed the program variables can be used in a nontrivial way. In many other models of synchronization (i.e. Petri Nets [5]) program variables and conditional branching are not allowed. The semaphores act as "flags"; they allow different processes to "synchronize." In many graph oriented models (i.e. Petri Nets) semaphores are called "places." Actions are functions that map states to states. In graph oriented models (i.e. Petri Nets) actions are called "transitions." Alternatively, actions have been called "indivisible steps"; here the fuzzy concept of indivisible is replaced by the precise concept of function. For instance, suppose that states are of the form

$$(L_1, L_2, (x,y,S)) .$$

Then f is an action where f is defined by

$$f(L_1,\ L_2,\ (x,y,S)) = \begin{cases} (2,\ L_2,\ (y,y,S)) & ,\ L_1 = 1\ ; \\ (L_1,\ L_2,\ (x,y,S)) & ,\ \text{otherwise.} \end{cases}$$

This action corresponds intuitively to the assignment: $x \leftarrow y$. In defining f it was, however, necessary to state explicitly how the program counters are used. The action f can "change" state w , that is $f(w) \neq w$, if and only if the L_1 component of w is 1 . In a sense 1 is the "address" of action f . Whenever f changes state w , the L_1 component of $f(w)$ is equal to 2 . In a sense 2 is the "address" of the "next" action from the process of f .

It is unconventional to make the program counters explicit--usually they are implicit. The reasons for this are twofold. First, in parallel systems of processes there are, by definition, many program counters present. Without making them explicit it is difficult to be precise. Second, explicit program counters help avoid awkward phrases such as "suppose that f is the next action to be executed in the process after g executed with the value of the program variable (that g branched on) equal to 0 ." These phrases are replaced by L_i = the address of f .

Since actions are functions, it is convenient to have the following notation for functions. Let D be the cartesian product of the sets $D_1, \ldots, D_k$; also let " $(x_1, \ldots, x_k)$ " be a typical element of the set D .* Then the notation

<u>when</u> $B(x_1, \ldots, x_k)$ <u>do</u> $x_1 \leftarrow f_1(x_1, \ldots, x_k); \ldots; x_k \leftarrow f_k(x_1, \ldots, x_k)$

*Each x_i acts like a variable, in the usual sense.

where B is a predicate and each f_i is a function, denotes the function g, from D to D, defined by

$$g(x_1, \ldots, x_k) = \begin{cases} (f_1(x_1, \ldots, x_k), \ldots, f_k(x_1, \ldots, x_k)) & \text{if } B(x_1, \ldots, x_k) ; \\ (x_1, \ldots, x_k) & \text{otherwise.} \end{cases}$$

For example, let " (x_1, x_2, x_3) " be a typical element in the cartesian product $\{1,2,3\} \times \{1,2,3\} \times \{1,2,3\}$ and let

$$f = \underline{\text{when}}\ x_1 = 1\ \underline{\text{do}}\ x_1 \leftarrow 2\ ;\ x_2 \leftarrow x_2\ ;\ x_3 \leftarrow x_3 .$$

Then $f(1,2,3) = (2,2,3)$ and $f(2,2,3) = (2,2,3)$. Assignments of the form " $x \leftarrow x$ " will be deleted; therefore,

$$f = \underline{\text{when}}\ x_1 = 1\ \underline{\text{do}}\ x_1 \leftarrow 2 .$$

A "selector" notation will be used. If " $(x_1, \ldots, x_k)$ " is a typical element of the cartesian product $D_1 \times \ldots \times D_k$ and $(a_1, \ldots, a_k)$ is in $D_1 \times \ldots \times D_k$, then

$$x_i[(a_1, \ldots, a_k)]$$

denotes a_i. For example, let " (x_1, x_2) " be a typical element in the set $\{1,2\} \times \{1,2\}$. Then $x_1[(2,1)] = 2$ and $x_2[(2,1)] = 1$.

Definition. $\mathcal{P} = \langle A, D, w \rangle$ is a system of processes provided there are functions address and program-counter with domain A such that (let " $(L_1, \ldots, L_n, G)$ " be a typical element of D)

(1) w is in D. Each element of D is a state; w is the initial state.

(2) Each element of A is an action. An action f is a function from D to D of the form

$$\underline{\text{when}}\ L_k = \text{address}(f) \wedge p(G)\ \underline{\text{do}}\ L_k \leftarrow b(G)\ ;\ G \leftarrow t(G)$$

where k = program-counter (f) and for all states x, $b(G[x]) \neq$ address (f). $p(G)$ is called the auxiliary predicate of f.

(3) If address (f) = address (g) and program-counter (f) = program-counter (g), then $f = g$.

A few comments on the definition of a system of processes are appropriate now. Restriction (3) of the definition of a system of processes does not allow the following actions to be in a system of processes:

$$\underline{\text{when}}\ L_1 = 1 \wedge S > 0\ \underline{\text{do}}\ L_1 \leftarrow 2$$

$$\underline{\text{when}}\ L_1 = 1 \wedge S = 0\ \underline{\text{do}}\ L_1 \leftarrow 3\ .$$

This restriction only allows the program-counter L_1 to contain the address of at most one action. However, the action

$$f = \underline{\text{when}}\ L_1 = 1\ \underline{\text{do}}\ L_1 \leftarrow \underline{\text{if}}\ S > 0\ \underline{\text{then}}\ 2\ \underline{\text{else}}\ 3$$

is allowed in a system of processes. Action f demonstrates a second feature: actions can conditionally branch. As stated earlier many other studies of synchronization do not allow branching. Some like Petri Nets do not allow branching since there are no program variables present. Others like Brinch

(Cert [1]) do allow only a very weak kind of branching. The branching is weak since the decisions are based on the values of uninterpreted predicates. Here since actions are functions arbitrary branching can be performed.

Definition. An active timing for the system of processes $\mathcal{P}$ is a finite sequence of actions $f_1 \ldots f_n$ such that

$$f_1(w) \neq w$$
$$f_2 \circ f_1(w) \neq f_1(w)$$
$$\vdots$$
$$f_n \circ \ldots \circ f_1(w) \neq f_{n-1} \circ \ldots \circ f_1(w)$$

where " $\circ$ " is function composition and w is the initial state of $\mathcal{P}$. The value $_{\mathcal{P}}(f_1 \ldots f_n)$ is equal to $f_n \circ \ldots \circ f_1(w)$.

Active timings and their corresponding values are fundamental to this model. Active timings correspond to what is often called a "computation." The value of an active timing $f_1 \ldots f_n$ is the state that results after the actions $f_1 \ldots f_n$ are performed. Thus, value maps active timings to states; for example, value$(fg) = g(f(w))$. Not any sequence of actions $f_1 \ldots f_n$ is an active timing: only those sequences such that each action f_i can change the state that results after $f_1 \ldots f_{i-1}$ are performed. An advantage of considering timings as sequences is that fuzzy concepts such as "time required for an instruction" are avoided.

Definition. Suppose that $\mathcal{P} = \langle A, D, v \rangle$ is a system of processes, and " $(L_1, \ldots, L_n, G)$ " is a typical element of D.

(1) For actions f and g, $\underline{\text{process}}_{\rho}(f,g)$ if and only if program-counter (f) = program-counter (g). Process_{ρ} is an equivalence relation on the actions of ρ; let ρ_i be the i^{th} equivalence class, i.e. $\rho_i = \{f \mid \text{program-counter } (f) = i\}$.

(2) An action f is in the $\underline{\text{ready-set}}_{\rho}(\alpha)$ where α is an active timing if and only if $f(\text{value } (\alpha)) \neq \text{value } (\alpha)$.

(3) An action f is in the $\underline{\text{pointer-set}}_{\rho}(\alpha)$ where α is an active timing if and only if $L_k[\text{value } (\alpha)] = \text{address } (f)$ where k = program-counter (f).

The notions process, ready-set, and pointer-set are basic to this model. First, consider the relation process. Two actions are intuitively in the same process if and only if process (f,g) if and only if f and g use the same program-counter. There is a one-to-one correspondence between processes and program-counters. In some models (i.e. Petri Nets) the concept of program-counter and process is missing. In effect there are two kinds of "control" in systems of processes: program-counters and semaphores. Only the semaphores are present in Petri Nets. The ability to have both program counters and semaphores is an attractive feature of systems of processes. Moreover, since conditional branching is allowed in systems of processes, the control afforded by the program-counters is nontrivial. Second, consider the function ready-set. Ready-set (α) can be defined in a number of alternative ways--some precise and some informal. Thus, f is in ready-set (α) if and only if

(1) f can "execute" after the active timing α is performed;

(2) αf is an active timing;

(3) f can "fire" after α is performed (in Petri Net language);

(4) the "transition" from state value (α) to f(value (α)) is defined (in Parallel Schema language);

(5) L_k[value (α)] = address (f) and $p(G)$ is true where L_k is the program-counter that f uses and $p(G)$ is the auxiliary predicate of f;

(6) f is in pointer-set (α) and $p(G)$ is true where $p(G)$ is again the auxiliary predicate of f.

The last two alternatives are dependent on the restriction:

$$\text{for all states } x\text{, } b(G[x]) \neq \text{address } (f)$$

in the definition of a system of processes. Third, consider the function pointer-set. Pointer-set (α) is intuitively the set of actions f such that the program-counter of f "points to" f (i.e. it contains the address of f). Since many models, as noted before, do not have program-counters, pointer-set has no immediate counterpart in other models.

Theorem 1. Suppose that $\mathcal{P}$ is a system of processes. Then

(I) For any timing α, ready-set (α) $\subseteq$ pointer-set (α).

(II) For any timing α, pointer-set (α) $\cap$ $\mathcal{P}_i$ contains at most one action.

(III) For any timings α and β, $\mathcal{P}_i \cap$ pointer-set (α) = $\mathcal{P}_i \cap$ pointer-set ($\alpha\beta$) provided no action from β is in $\mathcal{P}_i$.

Proof. Let " $(L_1, \ldots, L_n, G)$ " be a typical element of D where $\mathcal{P} = \langle A, D, w \rangle$.

(I) Suppose that f is in ready-set (α). Then by the definition of a system of processes,

$$f = \underline{\text{when}}\ L_i = \text{address}\ (f) \wedge p(G)\ \underline{\text{do}}\ L_i \leftarrow b(G)\ ;\ G \leftarrow t(G)$$

where i = program-counter (f). Also by the definition of ready-set, $f(\text{value}\ (\alpha)) \neq \text{value}\ (\alpha)$. Thus by the definition of when do notation, $L_i[\text{value}\ (\alpha)] = \text{address}\ (f)$; hence, f is in pointer-set (α).

(II) Suppose that f and g are both in pointer-set $(\alpha) \cap \mathcal{P}_i$. Since f and g are actions, it can be assumed that

$$f = \underline{\text{when}}\ L_i = \text{address}\ (f) \wedge p(G)\ \underline{\text{do}}\ L_i \leftarrow b(G)\ ;\ G \leftarrow t(G)$$
$$g = \underline{\text{when}}\ L_i = \text{address}\ (g) \wedge p'(G)\ \underline{\text{do}}\ L_i \leftarrow b'(G)\ ;\ G \leftarrow t'(G)\ .$$

Since f and g are both in pointer-set (α), $L_i[\text{value}\ (\alpha)] = \text{address}\ (f)$ and $L_i[\text{value}\ (\alpha)] = \text{address}\ (g)$. Therefore, by condition (3) of the definition of a system of process, $f = g$.

(III) It will be proved that if $i \neq j$, f is in $\mathcal{P}_i$, and g is in $\mathcal{P}_j$, then f is in pointer-set (α) if and only if f is in pointer-set (αg). Clearly, assertion (III) then follows by induction on the length of β. Suppose that f is in $\mathcal{P}_i$, g is in $\mathcal{P}_j$, and $i \neq$. The definition of a system of processes implies that $L_i[\text{value}\ (\alpha)] = L_i[\text{value}\ (\alpha g)]$. Now f is in pointer-set (αg) if and only if $L_i[\text{value}\ (\alpha g)] = \text{address}\ (f)$ and f is in pointer-set (α) if and only if $L_i[\text{value}\ (\alpha)] = \text{address}\ (f)$. Therefore, f is in pointer-set (α) if and only if f is in pointer-set (αg). □

The three basic properties of Theorem 1 will be referenced by their respective Roman numerals. Property I means that in order for an action f to execute its program counter must point to the address of f. Property II means that actions have unique address, i.e. a program counter can point to at most one action. Properties I and II immediately imply that each process ρ_j is sequential, i.e. at most one action from ρ_i is ever in ready-set (α). Property III states that program counters are "local" in the sense that only process ρ_i can change its program counter L_i.

Definition. Suppose that $\rho = \langle A, D, w \rangle$ is a system of processes, and let " $L_1, \ldots, L_n, (G, S_1, \ldots, S_m)$ " be typical element of D where $D = D_1 \times \ldots \times D_n \times (E \times \mathbb{Z}^m)$ and $\mathbb{Z}$ is the set of integers. We will now define four classes of systems of processes:

A. ρ is a PV system of processes provided each action of ρ is in one of the following forms:

(1) when $L_i = a \wedge S_j > 0$ do $L_i \leftarrow a'$; $S_j \leftarrow S_j - 1$

(2) when $L_i = a$ do $L_i \leftarrow a'$; $S_j \leftarrow S_j + 1$

(3) when $L_i = a$ do $L_i \leftarrow b(G)$; $G \leftarrow t(G)$.

An action of the form (1) is called a $P(S_j)$; an action of the form (2) is called a $V(S_j)$; an action of the form (3) is called a nonsynchronizer. The definition of a PV system of processes is essentially due to Dijkstra [3]

B. ρ is an up/down system of processes provided each action of ρ is in one of the following forms:

(1) when $L_i = a \wedge \sum_{j \in F} S_j \geq 0$ do $L_i \leftarrow a'$; $S_k - 1$

(2) when $L_i = a \wedge \sum_{j \in F} S_j \geq 0$ do $L_i \leftarrow a'$; $S_k + 1$

(3) when $L_i = a$ do $L_i \leftarrow b(G)$; $G \leftarrow t(G)$.

An action of the form (1) is called a $\{S_j \mid j \in F\}$: down (S_k) ; an action of the form (2) is called a $\{S_j \mid j \in F\}$: up (S_k) ; an action of the form (3) is called a nonsynchronizer. The definition of an up/down system of processes is essentially due to Wodon [15].

C. $\mathcal{P}$ is a PV chunk system of processes provided each action of $\mathcal{P}$ is in one of the following forms:

(1) when $L_i = a \wedge S_j \geq b$ do $L_i \leftarrow a'$; $S_j \leftarrow S_j - b$

(2) when $L_i = a$ do $L_i \leftarrow a'$; $S_j \leftarrow S_j + b$

(3) when $L_i = a$ do $L_i \leftarrow b(G)$; $G \leftarrow t(G)$ where $b > 0$.

An action of the form (1) is called a P(S_j with amount b) ; an action of the form (2) is called a V(S_j with amount b) ; an action of the form (3) is called a nonsynchronizer. The definition of a PV chunk system of processes is essentially due to Vantilborgh and van Lamsweerde [14].

D. $\mathcal{P}$ is a PV multiple system of processes provided each action of $\mathcal{P}$ is in one of the following forms:

(1) when $L_i = a \wedge S_{b_1} > 0 \wedge \ldots \wedge S_{b_k} > 0$ do $L_i \leftarrow a'$; $S_{b_1} \leftarrow S_{b_1} - 1$; ... ; $S_{b_k} \leftarrow S_{b_k} - 1$

(2) when $L_i = a$ do $L_i \leftarrow a'$; $S_{b_1} \leftarrow S_{b_1} + 1$; ... ; $S_{b_k} \leftarrow S_{b_k} + 1$

(3) when $L_i = a$ do $L_i \leftarrow b(G)$; $G \leftarrow t(G)$.

An action of the form (1) is called a $P(\{S_{b_1}, \ldots, S_{b_k}\})$; an action of the form (2) is called a $V(\{S_{b_1}, \ldots, S_{b_k}\})$; an action of the form (3) is called a nonsynchronizer. The definition of a PV multiple system of processes is essentially due to Patil [12].

The four classes of processes PV, PV chunk, PV multiple, and up/down are now compared. The actions of these systems of processes are divided into nonsynchronizers and synchronizers. Nonsynchronizers cannot use the semaphore variables (i.e. $S_1, \ldots, S_m$); synchronizers, on the other hand, cannot use the program variables. These four classes differ in what auxiliary predicates are allowed. In a PV system of processes only identically true or predicates of the form $S_j > 0$ are allowed as auxiliary predicates. PV chunk generalizes the predicates of the form $S_j > 0$ to predicates of the form $S_j \geq b$ where $b > 0$, while PV multiple generalizes the predicates of the form $S_j > 0$ to conjunctions of these predicates. Up/down, in contrast, allows very different auxiliary predicates. It allows predicates of the form: determine if the sum of a set of semaphores is nonnegative. PV, PV chunk, and PV multiple are similar in that they directly link the auxiliary predicate and the assignments made on the semaphores. For example, in PV multiple the auxiliary test $S_1 > 0 \wedge S_3 > 0$ implies that the assignments are $S_1 \leftarrow S_1 - 1$; $S_3 \leftarrow S_3 - 1$. Conversely, in up/down there is no connection at all between the auxiliary predicates and the assignments made.

Theorem 2. Suppose that $\mathcal{P}$ is a PV (respectively PV chunk, PV multiple, up/down) system of processes. Then if f is a nonsynchronizer and α is an active timing, then

(1) f is in ready-set (α) if and only if f is in pointer-set (α). Moreover, if f is a V(S_i) (respectively V(S_i with amount b), V({S_{b_1}, ..., S_{b_k}})), then (1) is true.

Proof. Suppose that f is a nonsynchronizer and α is an active timing. By property 1, if f is in ready-set (α), then f is in pointer-set (α); thus, suppose that f is in pointer-set (α). Since f is a nonsynchronizer

$$f = \underline{\text{when}}\ L_i = \text{address}(f)\ \underline{\text{do}}\ L_i \leftarrow b(G)\ ;\ G \leftarrow t(G)$$

where for all x, b(G[x]) ≠ address (f). Since f is in pointer-set (α), L_i[value (α)] = address (f). Thus, by the definition of the when do notation, f(value (α)) ≠ value (α); hence, f is in ready-set (α). The rest of the theorem is proved in a similar manner. □

Note, Theorem 2 critically uses the restriction that a nonsynchronizer be an action of the form:

$$\underline{\text{when}}\ L_i = a\ \underline{\text{do}}\ L_i \leftarrow b(G)\ ;\ G \leftarrow t(G)$$

where for all x in D, b(G[x]) ≠ a. Without this restriction a nonsynchronizer f might be in pointer-set (α) and not in ready-set (α).

An example of a system of processes is now presented. This example has a dual purpose. First, it should aid in the understanding of the basic concepts. Second, it is related to the first reader-writer problem of Courtois, Heymans, Parnas [2]. Consider the system of processes, RW1, whose actions are ($1 \le i \le 2$):

writer

(1) when $L = 1 \wedge a = 0 \wedge b = 0$ do $L \leftarrow 2$; $b \leftarrow b+1$

(2) when $L = 2$ do $L \leftarrow 3$

(3) when $L = 3$ do $L \leftarrow 1$; $b \leftarrow b-1$

reader-i

(1) when $L_i = 1 \wedge b = 0$ do $L_i \leftarrow 2$; $a \leftarrow a+1$

(2) when $L_i = 2$ do $L_i \leftarrow 3$

(3) when $L_i = 3$ do $L_i \leftarrow 1$; $a \leftarrow a-1$

The states of RW1 are of the form

$$(L, L_1, L_2, (a,b))$$

where L, L_1, L_2 are program counters and a and b are semaphores. The initial state is $(1, 1, 1, (0,0))$. RW1 has three processes:

$$\text{writer} = \{1,2,3\}$$

$$\text{reader-1} = \{(1,1), (2,1), (3,1)\}$$

$$\text{reader-2} = \{(1,2), (2,2), (3,2)\} .$$

The active timings of RW1 include, for instance,

$$123 \text{ and } (1,1)(1,2)(2,2) .$$

The value $(123) = (1, 1, 1, (0,0))$, while value $((1,1)(1,2)(2,2)) = (1, 2, 3, (2,0))$.

The relationship between RW1 and the first reader-writer problem is now explored. In the first reader-writer problem there are two kinds of processes: "readers" and "writers." In RW1 the case of two readers and

one writer is considered. These processes are to "share a resource" with the added constraint that a writer must have "exclusive access to the resource." Thus, readers can "simultaneously access the resource." In RW1 the actions 1, (1,1), (1,2) act as "gates" to the shared resource: the writer must execute 1 to enter the resource, while reader-i must execute (1,i) to enter the resource. The semaphores a and b are used to control who can enter. Semaphore a is equal to the number of readers using the resource, while semaphore b is equal to the number of writers using the resource. Thus, a writer can enter provided $a = 0$ and $b = 0$; a reader can enter provided $b = 0$. In RW1 the actions 2, (2,1), (2,2) correspond to the computing done in accessing the resource. These actions are essentially non-operations; this is done purely to simplify the example. In RW1 the actions 3, (3,1), (3,2) act as "exits from the resource." Thus action 3 sets b (the number of writers using the resource) to $b-1$. In summary the system of processes RW1 captures the essential behavior of the readers and writers in the first reader-writer problem.

3. Simulate

The comparison of synchronization primitives is reduced to the study of relations between the basic objects: systems of processes. In particular the relation simulate--Lipton [9]--is studied in detail. This relation is first defined. Next simulate is carefully motivated. The main argument in favor of simulate is: simulate is "efficient" in a number of natural ways, while weaker relations are often very "inefficient." The statement that PV is "weaker than" PV chunk becomes the precise statement:

> there is a PV chunk system of processes that cannot be simulated by any PV system of processes.

In this way synchronization primitives are compared.

Definition. Suppose that Q and $\mathcal{P}$ are systems of processes. r is a realization from Q to $\mathcal{P}$ provided

$$r : \{f \mid f \text{ action in } Q\} \to \{g \mid g \text{ action in } \mathcal{P}\} \cup \{\Lambda\}$$

where Λ is the empty sequence. r is extended to sequences of actions in the obvious way, i.e.,

$$r(f_1 \ldots f_n) = r(f_1) \ldots r(f_n)$$

where f_i $(1 \leq i \leq n)$ is an action in Q. If $r(f) \neq \Lambda$ for an action in Q, then say f is observable; otherwise, say f is unobservable.

Suppose that r is a realization from Q to $\mathcal{P}$. Then an action in Q is unobservable provided it does a "bookkeeping operation" that is not "present" in $\mathcal{P}$; an action is observable provided it does an "operation" that is "present" in $\mathcal{P}$.

Definition. Q simulates $\mathcal{P}$ with respect to the realization r provided r is a realization from Q to $\mathcal{P}$ such that

(1) $r\{\alpha | \alpha$ active in $Q\} = \{\beta | \beta$ active in $\mathcal{P}\}$;

(2) if ready-set$_Q(\alpha) \cap Q_i$ is empty, then ready-set$_{\mathcal{P}}(r(\alpha)) \cap r(Q_i)$ is empty;

(3) there is a $c > 0$ such that for all active timings α in Q, $1 +$ length $(r(\alpha)) \geq c$ length (α) ;

(4) for any observable actions f and g,

(a) if process$_Q(f,g)$, then process$_{\mathcal{P}}(r(f), r(g))$;

(b) if $r(f) = r(g)$, then process$_Q(f,g)$.

The definition of simulate is now examined. Suppose that Q simulates $\mathcal{P}$ with respect to realization r. Condition (1) has two parts. First, if α is an active timing in Q, then $r(\alpha)$ is an active timing in $\mathcal{P}$. Informally, whenever Q can "make a change," then $\mathcal{P}$ can also "make a corresponding change." This part of condition (1) is essentially the notion "safe" as used in Dijkstra [3]. Second, if β is an active timing in $\mathcal{P}$, then $r(\alpha) = \beta$ for some active timing α in Q. Informally, for any "sequence of changes" in $\mathcal{P}$, there is a "corresponding sequence of changes" in Q. Condition (1) is called onto safe. Condition (2) can be rephrased as follows. Say a set of actions B is stopped at α provided ready-set $(\alpha) \cap B$ is empty. Then this condition is equivalent to

if Q_i is stopped at α, then $r(Q_i)$ is stopped at $r(\alpha)$.

Of course Q_i may be stopped at α, while Q_i is not stopped at α. This condition has been motivated by the requirement stated in Dijkstra [3]:

"that stopping a process in its 'remainder of cycle' has no effect upon the others." This condition is called deadlock free on processes. Condition (3) informally means that an active timing in Q can "only spend a bounded portion of its time doing bookkeeping." Note this condition cannot be expressed by

$$\text{length } (r(\alpha)) \geq c \text{ length } (\alpha) .$$

In order to see this just consider the case where $r(\alpha) = \Lambda$ and yet $\alpha \neq \Lambda$. This condition is called busy wait free; it is essentially the concept used in Dijkstra [3]. Condition (4) can be rephrased as follows. Let $\bar{\mathcal{P}}_i$ be the set $\{r(f) | f$ is observable and f is in $Q_i\}$. Then condition (4a) is: for each i, there is a j such that $\bar{\mathcal{P}}_i \subseteq \bar{\mathcal{P}}_j$. Condition (4b) is: for each distinct i and j, $\bar{\mathcal{P}}_i$ and $\bar{\mathcal{P}}_j$ are disjoint. Informally this condition means that "the process structure of Q is finer than the process structure of $\mathcal{P}$." Condition (4) is called faithful; it is implicit in the synchronization literature.

In summary one justification of simulate is that each of the four parts of its definition are found in the synchronization literature. This is especially the case for onto safe and busy wait free.

Another justification for simulate is that the deletion of any one of the four conditions from the definition of simulate results in a pathological relation. First, consider deleting the restriction that r be onto safe. Let Q be a system of processes whose active timings are exactly

$$\Lambda, f, fg, fgf, fgfg, \ldots \quad \text{(alternate } f \text{ and } g\text{)}$$

where process (f,g). Also let $r(f) = r(g) = h$ where h is any actio

in some system of process $\mathcal{P}$. It is easy to see that r is deadlock free on processes, busy wait free, and faithful. Clearly r is not in general onto safe. Since Q and $\mathcal{P}$ have no intuitive connection, deleting onto safe leads to a pathological relation. Second, consider deleting the restriction that r be busy wait free. Suppose that r is an onto safe deadlock free faithful realization from Q to $\mathcal{P}$. Add the following unobservable actions to Q:

$$f = \underline{\text{when}}\ L = 1\ \underline{\text{do}}\ L \leftarrow 2$$

$$g = \underline{\text{when}}\ L = 2\ \underline{\text{do}}\ L \leftarrow 1$$

where L is a new program counter. Then, for example,

$$fgfg \ldots . fg \quad (fg \text{ repeated})$$

is an active timing, and hence Q can spend unbounded amounts of time doing nothing. Deleting busy wait free therefore leads to an inefficient relation. Third, consider deleting the restriction that r be faithful. Suppose that Λ, f, g, fg, gf are the active timings of the system of processes $\mathcal{P}$. Consider the PV system of processes Q whose actions are:

process-1

(1) $\underline{\text{when}}\ L_1 = 1\ \underline{\text{do}}\ L_1 \leftarrow x$

(2) $\underline{\text{when}}\ L_1 = 2\ \underline{\text{do}}\ L_1 \leftarrow 3$

(3) $\underline{\text{when}}\ L_1 = 3\ \underline{\text{do}}\ L_1 \leftarrow 6$

(4) $\underline{\text{when}}\ L_1 = 4\ \underline{\text{do}}\ L_1 \leftarrow 5$

(5) $\underline{\text{when}}\ L_1 = 5\ \underline{\text{do}}\ L_1 \leftarrow 6$

process 2

(6) when $L_2 = 1$ do $L_2 := 2$; $x := 4$

The states of Q are (L_1, L_2, x) ; the initial state is $(1,1,2)$. The active timings of Q are all the prefixes of the sequences

$$6145 , \quad 1623 , \quad 1263 , \quad 1236 .$$

Let r be the realization from Q to $\mathcal{P}$ defined by

$$r(1) = \Lambda, \quad r(2) = f , \quad r(3) = g , \quad r(4) = g , \quad r(5) = f , \quad r(6) = \Lambda$$

The actions 1 and 6 are unobservable. It is easy to verify that r is onto safe, busy wait free, and deadlock free on processes. Actions 2 and 5 correspond to f , while actions 3 and 4 correspond to g . This is inefficient, for each action of $\mathcal{P}$ occurs twice in Q . In addition, it seems intuitively unpleasing that action 1 can "bind the entire future of Q ." Once action 1 executes either

(i) L_1 is set to 2 or (ii) L_1 is set to 4 .

In case (i), fg executes; in case (ii), gf executes. Deleting faithful therefore leads to a pathological relation.

Last consider deleting the restriction that r be deadlock free on processes. Of the four restrictions that make up the definition of simulation this is perhaps the most controversial. Indeed this restriction is perhaps the key reason that seemingly paradoxical results are obtained here. For example, the fact that PV cannot "solve certain synchronization problems" (i.e. not all systems of processes can be simulated by PV systems of semaphores)

is counterintuitive to some. However, deadlock free on processes is a natural restriction, for without it pathological behavior can occur. In order to see this consider a system of processes $\mathcal{P}$ whose actions are:

SUPERVISOR

(1) when $L_1 = 1$ do $L_1 \leftarrow 2$; $S \leftarrow S+1$

USER

(2) when $L_2 = 1 \wedge S > 0$ do $L_2 \leftarrow 2$

(3) when $L_2 = 2$ do $L_2 \leftarrow 1$

OTHER

(4) when $L_3 = 1$ do $L_3 \leftarrow 2$

(5) when $L_3 = 2$ do $L_3 \leftarrow 1$.

The states of $\mathcal{P}$ are of the form (L_1, L_2, L_3, S) ; the initial state is $(1,1,1,1)$. Intuitively the behavior of $\mathcal{P}$ is as follows: process USER can "Loop" until the process SUPERVISOR executes, once this occurs the USER is stopped from starting a new "cycle." The process OTHER can, on the other hand, execute whether or not SUPERVISOR has executed. Now consider the PV system of processes Q whose actions are:

SUPERVISOR

(1) when $L_1 = 1 \wedge T > 0$ do $L_1 \leftarrow 2$; $T \leftarrow T-1$

USER

(2) when $L_2 = 1 \wedge T > 0$ do $L_2 \leftarrow 2$; $T \leftarrow T-1$

(2') when $L_2 = 2$ do $L_2 \leftarrow 3$; $T \leftarrow T+1$

(3) when $L_2 = 3$ do $L_2 \leftarrow 1$

OTHER

(4) when $L_3 = 1$ do $L_3 \leftarrow 2$

(5) when $L_3 = 2$ do $L_3 \leftarrow 1$.

The states of Q are of the form (L_1, L_2, L_3, T) ; the initial state is $(1,1,1,1)$. The behavior of Q seems to be intuitively equivalent to the behavior of $\mathcal{P}$: the process SUPERVISOR stops the process USER, while the process OTHER is unaffected. Let r be the realization defined by

$$r(1) = 1 \ , \quad r(2) = 2 \ , \quad r(2') = \Lambda \ , \quad r(3) = 3 \ , \quad r(4) = 4 \ , \quad r(5) = 5 \ .$$

Then r is onto safe, busy wait free, and faithful; however, r is not deadlock free on processes. r fails to be deadlock free on processes since action 1 is not in $\text{ready-set}_Q(2)$ while action $r(1)$ is in $\text{ready-set}_{\mathcal{P}}(r(2))$. Intuitively r is not deadlock free on processes because in Q the SUPERVISOR is sometimes stopped when it should not be stopped. In order to understand the importance of the failure of r to be deadlock free, Q and $\mathcal{P}$ must be examined with respect to their "response times." Let $N_{\text{SUPERVISOR}}$, N_{USER} , N_{OTHER} be the maximal times that the processes SUPERVISOR, USER, OTHER ever wait for a chance to execute under some given scheduling. In $\mathcal{P}$ the process SUPERVISOR waits at most $N_{\text{SUPERVISOR}}$. On the other hand, in Q the process SUPERVISOR can wait as long N_{USER} (just stop the USER after action 2 has executed; the SUPERVISOR cannot execute again until the USER does). Since N_{USER} could easily be much larger than $N_{\text{SUPERVISOR}}$ the response time of SUPERVISOR is seriously degraded. In real time systems this degrading of response time could even lead to the failure of a system. This anomaly of response times follows directly from that failure of r to

be deadlock free on processes. Thus deleting the restriction deadlock free on processes does indeed result in a pathological relation.

A further reason why simulate is a reasonable relation is that it is weaker than "isomorphism." Consider the system of processes F1 whose actions are:

producer-1

(1) when $L_1 = 1$ do $L_1 \leftarrow 2$; $a \leftarrow a+1$

(2) when $L_1 = 2$ do $L_1 \leftarrow 1$

producer-2

(3) when $L_2 = 1$ do $L_2 \leftarrow 2$; $b \leftarrow b+1$

(4) when $L_2 = 2$ do $L_2 \leftarrow 1$

consumer

(5) when $L_3 = 1 \wedge a > 0 \wedge b > 0$ do $L_3 \leftarrow 2$; $a \leftarrow a-1$; $b \leftarrow b-1$

(6) when $L_3 = 1$ do $L_3 \leftarrow 1$.

The states of F1 are of the form

$$(L_1, L_2, L_3, (a,b)) ,$$

the initial state is $(1, 1, 1, (0,0))$. Intuitively, F1 represents the behavior where "two processes put records into two individual buffers while another process consumes them." The key restriction is that the consumer can "proceed only when both buffers are nonempty" (i.e. $a > 0$ and $b > 0$). Now consider the PV system of processes, F2, whose actions are:

producer-1

(1) when $L_1 = 1$ do $L_1 \leftarrow 2$; $a \leftarrow a+1$

(2) when $L_1 = 2$ do $L_1 \leftarrow 1$

producer-2

(3) when $L_2 = 1$ do $L_2 \leftarrow 2$; $b \leftarrow b+1$

(4) when $L_2 = 2$ do $L_2 \leftarrow 1$

consumer

(5) when $L_3 = 1 \wedge a > 0$ do $L_3 \leftarrow 2$; $a \leftarrow a-1$

(5') when $L_3 = 2 \wedge b > 0$ do $L_3 \leftarrow 3$; $b \leftarrow b-1$

(6) when $L_3 = 3$. do $L_3 \leftarrow 1$.

The states are of the form

$$(L_1, L_2, L_3, (a,b)) ;$$

the initial state is $(1, 1, 1, (0,0))$. Let r be the realization from P2 to P1 defined by

$r(1) = 1$, $r(2) = 2$, $r(3) = 3$, $r(4) = 4$, $r(5) = \Lambda$, $r(5') = 5$, $r(6) =$

Note 5' is an unobservable action. It is easy to see that P2 simulates P1 with respect to realization r . This example shows that a PV system of processes can simulate a non-PV system of processes. Thus intuitively equivalent but nonidentical systems of processes can be related by simulation.

For future reference it is now noted that PV chunk, PV multiple up/down are "generalizations" of PV. Clearly, if $\mathcal{P}$ is a PV system of processes, then $\mathcal{P}$ is a PV chunk system of processes and $\mathcal{P}$ is a PV multiple

system of processes; however, P need not be an up/down system of processes. For example, the PV system of processes whose one action is

$$\underline{\text{when}}\ L_1 = 1 \wedge S > 0\ \underline{\text{do}}\ L_1 \leftarrow 2\ ;\ S \leftarrow S-1$$

is not an up/down system of processes. However, each PV system of processes can be simulated by an up/down system of processes (Wodon [9]). For example, the above action is replaced by

$$\underline{\text{when}}\ L_1 = 1 \wedge T \geq 0\ \underline{\text{do}}\ L_1 \leftarrow 2\ ;\ T \leftarrow T-1$$

where the initial value of T is less than the initial value of S. In summary, PV chunk, PV multiple, and up/down are <u>generalizations</u> of PV in the following sense: every PV system of processes can be simulated by a PV chunk (respectively PV multiple, up/down) system of processes.

4. Invariance of local behavior of a System of Processes

We will now state and then prove a necessary condition for Q to simulate $\mathcal{P}$. In order to state this condition, we will first formalize the concept of the "local behavior" of a system of processes. We will then state the invariance theorem:

> if Q simulates $\mathcal{P}$, then the local behavior of $\mathcal{P}$ is the local behavior of Q.

The invariance theorem will be used later to prove the results stated in the introduction.

Definition. Suppose that Σ is a finite set. Then the set Π is a Σ-slice provided

(1) each element of Π is a finite sequence of distinct elements from Σ;

(2) each element of Σ is in Π;

(3) if $\alpha\beta$ is in Π, then α is in Π.

A slice is a tree--coded as a set of words--that describes the purely parallel or local behavior of a system of processes. Previous workers (Karp and Miller [6], Keller [7]) have divided local behavior into two classes those with "conflicts" and those without conflicts. Slices allow a finer distinction. For example, the slice $\{\Lambda, a, b\}$ contains a conflict, while the slice $\{\Lambda, a, b, ab, ba\}$ does not.

Definition. The system of processes $\mathcal{P} = \langle A, D, w \rangle$ defines the Σ-slice provided there is a one-to-one correspondence from A to Σ such that $d\{\alpha | \alpha \text{ active in } \mathcal{P}\} = \Pi$. Note, we extend d to timings of $\mathcal{P}$ by $d(\alpha_1 \ldots \alpha_k)$ to be $d(\alpha_1) \ldots d(\alpha_k)$.

The following two observations on the relation defines are often useful. Suppose that $\mathcal{P}$ defines Π. The first observation is: distinct actions of $\mathcal{P}$ lie in different processes. In order to see this suppose that f and g are distinct actions in $\mathcal{P}$. By the definition of slice part (2), since $\mathcal{P}$ defines Π, f and g are both in ready-set (Λ). By properties I and II, it follows that not process (f,g). The second observation is: each action of $\mathcal{P}$ is in ready-set (Λ). This has already been established in the argument for the first observation.

Consider the up/down system of processes M whose actions are:

(1) when $L_1 = 1 \wedge S \geq 0$ do $L_1 \leftarrow 2$; $T \leftarrow T+1$

(2) when $L_2 = 1$ do $L_2 \leftarrow 2$; $S \leftarrow S-1$.

The states of M are of the form $(L_1, L_2, (S,T))$; the initial state is $(1, 1, (0,0))$. The set of active timings of M is $\{\Lambda, 1, 2, 12\}$: 21 is not active because $S[\text{value } (2)] = -1$. Thus, M defines the $\{x,y\}$-slice $\{\Lambda, x, y, xy\}$. Informally, this slice represents the local behavior where y "stops" x and x does not stop y. In M action 2 stops action 1 and action 1 does not stop action 2.

Definition. The system of processes $\langle A, D, w\rangle$ implicitly defines the Σ-slice Π provided there is a subset A' of A and an active timing α in $\langle A, D, w\rangle$ such that $\langle A', D, \text{value } (\alpha)\rangle$ defines Π.

Consider the system of processes RW1 represented in Figure 2. Let RW1 = $\langle A, D, w\rangle$, and let $\mathcal{P} = \langle\{1, (1,1), (1,2)\}, D, \text{value } (\Lambda)\rangle$. Then the active timing of $\mathcal{P}$ are

$$\{\Lambda,\ 1,\ (1,1),\ (1,2),\ (1,1)\ 1,2),\ (1,2)(1,1)\}\ .$$

Thus, RWI implicitly defines the slice $\{\Lambda,x,y,z,xy,yz\}$.

Clearly, different slices represent different kinds of local behavior. For example, the slice $\{\Lambda,x,y,z,xy,yz\}$ represents the behavior where x and y each stop z , z stops x and y , but x and y do not stop each other.

Before we state and prove the invariance theorem we will prove the following theorem; it is used in the proof of the invariance theorem.

Theorem 3. Suppose that Q simulates $\mathcal{P}$ with respect to the realization r . Also suppose that β is an active timing in $\mathcal{P}$. Then there is an active timing α in Q such that r is a one-to-one correspondence from ready-set$_Q(\alpha)$ to ready-set$_{\mathcal{P}}(\beta)$ and $r(\alpha) = \beta$.

The central idea of this proof is as follows. First, use onto safe to construct an active timing α' such that $r(\alpha') = \beta$. Second, use bucy wait free to extend α' to an active timing α such that $r(\alpha) = \beta$ and α has the property: if αh is an active timing, then h is an unobservable action. Essentially α is constructed by executing after α' as many unobservable actions as possible. Third, use deadlock free on process r and faithful to argue that r is a one-to-one correspondence from ready-set$_Q(\alpha)$ to ready-set$_{\mathcal{P}}(\beta)$.

Proof. Let ready-set$_{\mathcal{P}}(\beta) = \{g^1, \ldots, g^m\}$. Since r is onto safe, for each $1 \leq i \leq m$, there is an active timing $\lambda^i f^i$ in Q such that $r(\lambda^i) = \beta$ and $r(f^i) = g^i$. By property I, $\{g^1, \ldots, g^m\} \subseteq$ pointer-set$_{\mathcal{P}}(\beta)$; hence by property II, if $i \neq j$, then not process (g^j, g^i) . Since r is

faithful realization, if $i \neq j$, then not process$_Q(f^i, f^j)$. We can therefore assume, by renaming if necessary, that

(1) for $1 \leq i \leq m$, f^i is in Q_i.

The definition of busy wait free implies there is a constant > 0, say c, such that for α, an active timing in Q, $1 + \text{length}(r(\alpha)) \geq c \cdot \text{length}(\alpha)$. We can, therefore, select an active timing α such that $r(\alpha) = \beta$ and α has maximal length. The key property of α is: if αh is an active timing, then h must be an observable action. We now assert that

(2) ready-set$(\alpha) \cap Q_i$ is not empty $(1 \leq i \leq m)$.

Suppose conversely that (2) is false, and let ready-set$_P(\alpha) \cap Q_i$ be empty. Since r is deadlock free on processes and $r(\alpha) = \beta$, ready-set$_P(\beta) \cap r(Q_i)$ is empty. Now $r(f^i)$ is in $r(Q_i)$ and $r(f^i) = g^i$ thus, $r(f^i)$ is in ready-set$_P(\alpha) \cap r(Q_i)$. This is a contradiction, and hence (2) is true. Let h^i $(1 \leq i \leq m)$ be an action in ready-set$_Q(\alpha) \cap Q_i$. The key property of α implies that for $1 \leq i \leq m$, h^i is an observable action. Since r is onto safe and αh^i is active for $1 \leq i \leq m$, $r\{h^1, \ldots, h^m\} \subseteq \{g^1, \ldots, g^m\}$. Moreover, since r is faithful, r is a one-to-one map from $\{h^1, \ldots, h^m\}$ to $\{g^1, \ldots, g^m\}$; hence, $r\{h^1, \ldots, h^m\} = \{g^1, \ldots, g^m\}$. In order to complete the proof, suppose that αh is an active timing in Q and h is not in $\{h^1, \ldots, h^m\}$. Again since r is onto safe, $r(h) = g^i$, for some i. Since r is faithful, h is Q_i where $1 \leq i \leq m$. However, we can conclude by properties I and II that $h = h^i$. This is a contradiction, and hence ready-set$_Q(\alpha)$ $= \{h^1, \ldots, h^m\}$. □

Theorem 4. [Invariance Theorem] If Q simulates $\mathcal{P}$ and $\mathcal{P}$ implicitly defines the Σ-slice Π, then Q implicitly defines the Σ-slice Π.

Before proceeding with the proof of the invariance theorem it may be helpful to sketch the proof of the following special case:

> If Q simulates $\mathcal{P}$ and $\mathcal{P}$ defines $\{\Lambda, a, b, ab\}$ then Q implicitly defines $\{\Lambda, a, b, ab\}$.

Suppose that Q simulates $\mathcal{P}$ with respect to realization r. Since $\mathcal{P}$ defines $\{\Lambda, a, b, ab\}$ the active timings of $\mathcal{P}$ are exactly $\{\Lambda, f, g, fg\}$, for some actions f and g. By Theorem 3, there is an active timing β in Q and actions F and G such that βF and βG are active, $r(F) = f$, and $r(G) = g$. Since r is onto safe, βGF is not an active timing, otherwise, $r(\beta GF) = gf$ would be an active timing. In order to complete the proof it must be shown that βFG is an active timing. Once this is proved it will follow that Q implicitly defines $\{\Lambda, a, b, ab\}$. The only way βFG could fail to be an active timing is if G is not in ready-set (βF). Therefore suppose that βFG is not an active timing. Let G be in Q_i. Since G was in ready-set (β) and F and G must be from different processes, it follows that ready-set $(\beta F) \cap Q_i$ is empty. Since r is deadlock free on processes, ready-set $(f) \cap r(Q_i)$ is empty; this is a contradiction for g is in this intersection (G in Q_i implies g in $r(Q_i)$).

Proof of the Invariance Theorem. Let Q simulate $\mathcal{P}$ with respect to the realization r, let $\mathcal{P} = \langle A, D, w \rangle$, and let $Q = \langle B, D', x \rangle$. Since $\mathcal{P}$ implicitly defines Π, there exists an A_0, a subset of A, and an active timing α such that $\langle A_0, D, \text{value}(\alpha) \rangle$ defines Π. Also let

be the one-to-one correspondence between A_0 and Π. Suppose that g is in A_0. Then by the definition of a slice, g is in ready-set$_{\rho}(\alpha)$. By Theorem 3, there exists an active timing β in Q and a subset B_0 of ready-set$_Q(\beta)$ such that $r(\beta) = \alpha$ and r is a one-to-one correspondence from B_0 to A_0. We now assert that $\langle B_0, D', \text{value } (\beta)\rangle$ defines Π where the one-to-one correspondence from B_0 to Π is the composition of d and r.

Suppose first that δ is active $\langle B_0, D', \text{value } (\beta)\rangle$. Then $\beta\delta$ is active in Q. Since r is onto safe, $r(\beta\delta)$ is active; hence $r(\delta)$ is active in $\langle A_0, D, \text{value } (\alpha)\rangle$. Thus $d(r(\delta))$ is in Π.

On the other hand, suppose that δ is a timing in $\langle B_0, D', \text{value } (\beta)\rangle$ and $d(r(\delta))$ is in Π. Since $\langle A_0, D, \text{value } (\alpha)\rangle$ defines Π, $r(\delta)$ is active in $\langle A_0, D, \text{value } (\alpha)\rangle$; and hence $\alpha r(\delta)$ is active in We now assert that

(1) if $i \neq j$, then not process$_Q(\delta_i, \delta_j)$.

For suppose that process$_Q(\delta_i, \delta_j)$. Since δ_i and δ_j are both in ready-set$_Q(\beta)$, $\delta_i = \delta_j$ by properties I and II. Since $d(r(\delta))$ is in Π, $i = j$ by the definition of slice; hence, (1) is true. We now assert that $\beta\delta$ is active. Suppose conversely that $\delta = \lambda f u$, $\beta\lambda$ is active, and $\beta\lambda f$ is not active. Let f be in Q_i. We next assert that

(2) ready-set$_Q(\beta\lambda) \cap Q_i$ is nonempty.

For suppose that ready-set$_Q(\beta\lambda) \cap Q_i$ is empty. Then since r is deadlock free on processes, ready-set$_{\rho}(r(\beta\lambda)) \cap r(Q_i)$ is empty. Now $r(f)$ is in $r(Q_j)$. In addition, $r(\beta\lambda f)$ is active, for $r(\beta\lambda f u) = \alpha r(\delta)$ is active. Since f is an observable action, $r(f)$ is in ready-set$_{\rho}(r(\beta\lambda))$;

...since, we have a contradiction. Therefore, (2) is true; hence, we can suppose that h is in $\text{ready-set}_Q(\beta\lambda) \cap Q_1$.

Since f is in h_0, f is in $\text{ready-set}_Q(\beta)$; hence by properties I and II, f is in $\text{pointer-set}_Q(\beta)$. By (1) and property III, f is in $\text{pointer-set}_Q(\beta\lambda)$. Since both f and h are in Q_1, it follows from properties I and II that $f = h$. This is a contradiction, and hence $\beta\delta$ is active in Q. Therefore, δ is active in $\langle \mathcal{B}_0, D', \text{value } (\beta)\rangle$.

We have therefore shown that $\langle \mathcal{B}_0, D', \text{value } (\beta)\rangle$ defines Π; thus, Q implicitly defines Π. □

Definition. The class C of systems of processes defines the Σ-slice Π provided for some $\mathcal{P}$ in C, $\mathcal{P}$ defines Π.

Definition. Suppose that C and C' are classes of systems of processes. Then $C \rightarrow C'$ provided there is a $\mathcal{P}$ in C such that for no Q in C' does Q simulate $\mathcal{P}$.

The relation $\rightarrow$ is the relation that is used to compare synchronization primitives. Intuitively $C \rightarrow C'$ means that there is some "synchronization problem" that C can "solve" but C' cannot.

Definition. A class C of systems of processes is closed provided if $\langle A, \ldots \rangle$ is in C and A_0 is a subset of A and α is an active timing, then $\langle A_0, D, \text{value } (\alpha)\rangle$ is in C.

Corollary 5. Suppose that C and C' are classes of systems of processes. Also suppose that C' is closed. Then if C defines Π and C' does not define Π, then $C \rightarrow C'$.

Proof. Suppose that C defines Π and C' does not define Π. Let $\mathcal{P}$ be in C such that $\mathcal{P}$ defines Π. Assume that $C \rightarrow C'$ is false; let Q be in C' such that Q simulates $\mathcal{P}$. Since $\mathcal{P}$ implicitly defines Π, by the invariance theorem, Q implicitly defines Π. Since C' is closed, there is a Q' in C' such that Q' defines Π. This is a contradiction; hence, $C \not\rightarrow C'$. □

Corollary 5 is the key method we use to compare classes of systems of processes.

Theorem 6. The class of PV (respectively PV chunk, PV multiple, up/down) systems of processes is closed.

Proof. Immediate from the definitions of these systems of processes. □

5. Analysis of Slices

In this section the invariance theorem is used to show that there are differences between the classes of systems of processes: PV, PV chunk, PV multiple, and up/down. In particular, the results stated in the introduction are proved. It is possible to say much more about what systems of processes can define what slices--see Lipton [10] and Lipton, Snyder, Zalcstein [11].

Our first goal is to show that up/down $\not\rightarrow$ PV, up/down $\not\rightarrow$ PV chunk, and up/down $\not\rightarrow$ PV multiple. In order to prove this we will consider the slice $\Pi_2 = \{\Lambda, a, b, ab\}$. This slice is connected with the second reader-writer problem of Courtois, Heymans, Parnas [2]. Consider the system of processes RW2 whose actions are:

writer

(1) when $L = 1$ do $L \leftarrow 2$; $s \leftarrow s+1$

(2) when $L = 2 \wedge a = 0 \wedge b = 0$ do $L \leftarrow 3$; $b \leftarrow b+1$

(3) when $L = 3$ do $L \leftarrow 4$

(4) when $L = 4$ do $L \leftarrow 5$; $b \leftarrow b-1$

(5) when $L = 5$ do $L \leftarrow 1$; $s \leftarrow s-1$

reader-i $(1 \leq i \leq 2)$

$(1,i)$ when $L_i = 1 \wedge s = 0 \wedge b = 0$ do $L_i \leftarrow 2$; $a \leftarrow a+1$

$(2,i)$ when $L_i = 2$ do $L_i \leftarrow 3$

$(3,i)$ when $L_i = 3$ do $L_i \leftarrow 1$; $a \leftarrow a-1$

The states of RW2 are of the form

$$(i, L_1, L_2, (a,b,s)) ;$$

the initial state is $(1, 1, 1, (0,0,0))$. RW2 is identical to RW1 except for the two new actions 1 and 5. These actions use the semaphore s as follows. Intuitively s is equal to "the number of writers that are waiting for a chance to use the shared resource." Readers must now not only check that there are no writers using the resource $(b \geq 0)$ but also they must check that no writer is waiting $(s \geq 0)$. Informally, RW2 is the system of processes that represents the behavior of the readers and writers in the second reader-writer problem. Moreover, RW2 implicitly defines Π_1; hence, RW2 contains the behavior where a "writer can stop a reader but a reader cannot stop a writer."

Theorem 7. Up/down defines Π_1.

Proof. Consider the up/down system of processes whose actions are:

(1) when $L_1 = 1 \wedge s \geq 0$ do $L_1 \leftarrow 2$; $T \leftarrow T+1$

(2) when $L_2 = 1$ do $L_2 \leftarrow 2$; $S \leftarrow S-1$.

The states of this system of processes are of the form $(L_1, L_2, (S,T))$; the initial state is $(1, 1, (0,0))$. Clearly the active timings of this system of processes are $\{\Lambda, 1, 2, 12\}$; hence, up/down defines Π_1. □

Theorem 8. The class of PV (respectively PV chunk, PV multiple) systems of processes does not define Π_1.

Proof. We will prove the theorem in the PV case; the other cases follow in a similar manner. Suppose that $\mathcal{P} = \langle\{S, C\}, D, \nu\rangle$ is a PV system of processes that defines Π_1, let $\alpha = (L_1, L_2, \ldots, L_n, (C, S_1, \ldots, S_m))$ be a typical element in D. Since $\mathcal{P}$ defines Π_1, we can assume that

the active timings of $\mathcal{P}$ are $\{\Lambda, f, g, gf\}$. By the definition of slice, not process (f,g). Therefore, by property III, f is in pointer-set (g) and g is in pointer-set (f). Since g is not in ready-set (f), Theorem 7 implies that g is $F(S_j)$ for some j. We can therefore assume that

$$g = \underline{\text{when}}\ L_j = a \wedge S_1 > 0\ \underline{\text{do}}\ L_1 \leftarrow a'\ ;\ S_1 \leftarrow S_1 - 1$$

without loss of generality. Since g is in ready-set (Λ) and g is not in ready-set (f), $S_1[w] > 0$ and $S_1[f(w)] \leq 0$. Therefore, f is $F(S_1)$; and we can assume that

$$f = \underline{\text{when}}\ L_2 = b \wedge S_1 > 0\ \underline{\text{do}}\ L_1 \leftarrow b'\ ;\ S_1 \leftarrow S_1 - 1$$

Then $S_1[f(w)] = 0$; hence, $S[g(w)] = 0$. This implies that f is not in ready-set (g), which is a contradiction. □

Theorem 9. Up/down → PV, up/down → PV chunk, up/down → PV multiple.

Proof. Use theorems 7 and 8 and corollary 5. □

Our second goal is to prove that PV multiple → PV chunk. In order to prove this we will consider the slice $\mathcal{L}_2 = \{\Lambda, a, b, c, d, ad, da, bc, cb\}$. We can represent this slice by Figure 2. Clearly, adjacent nodes can exclude each other while nonadjacent ones cannot. This slice is related to the "Five Dining Philosophers Problem" of Dijkstra [4]. Consider the system of processes PV whose actions are:

philosopher-i $(0 \leq i \leq 3)$

(1,i) when $L_i = 1 \wedge S_{i-1} > 0 \wedge S_{i+1} > 0$ do $L_i \leftarrow 2$; $S_{i-1} \leftarrow S_{i-1} - 1$; $S_{i+1} \leftarrow S_{i+1} - 1$

(2,i) when $L_i = 2$ do $L_i \leftarrow 3$

(3,i) when $L_i = 3$ do $L_i \leftarrow 1$; $S_{i-1} \leftarrow S_{i-1} + 1$; $S_{i+1} \leftarrow S_{i+1} + 1$

By convention $S_{-1} = S_3$ and $S_4 = S_0$. The states of DP are of the form

$$(L_1, L_2, L_3, L_4, (S_1, S_2, S_3, S_4)) ;$$

the initial state is $(1, 1, 1, 1, (1,1,1,1))$. Informally, DP is the system of processes that represents the behavior of the philosophers (for the case of four philosophers). In addition, DP implicitly defines Π_2.

Theorem 10. PV multiple defines Π_2.

Proof. Consider the PV multiple system of processes DP. The active timings of this system of processes are $\{\Lambda,1,2,3,4,14,41,23,32\}$. Therefore, PV multiple defines Π_2. □

Theorem 11. The class of PV chunk systems of processes does not define Π_2.

Proof. Suppose that $\mathcal{P} = \langle\{e,f,g,h\}, D, v\rangle$ defines Π_2, and let "$(L_1, \ldots, L_n, (G, S_1, \ldots, S_n))$" be a typical element of E. Since $\mathcal{P}$ defines Π_2, we can assume that the active timings of $\mathcal{P}$ are $\{\Lambda,e,f,g,h,eh,he,fg,gf\}$. By the same type of reasoning used in Theorem 8, we can assume that

$$e = \underline{\text{when}}\ L_1 = a_1 \wedge S_{i_1} \geq b_1\ \underline{\text{do}}\ L_1 \leftarrow a_1' ;\ S_{i_1} \leftarrow S_{i_1} - b_1$$

$$f = \underline{\text{when}}\ L_2 = a_2 \wedge S_{i_2} \geq b_2\ \underline{\text{do}}\ L_2 \leftarrow a_2' ;\ S_{i_2} \leftarrow S_{i_2} - b_2$$

g = when $L_3 = a_3 \wedge S_{i_3} \geq b_3$ do $L_3 \leftarrow a_3'$; $S_{i_3} \leftarrow S_{i_3} - b_3$

h = when $L_4 = a_4 \wedge S_{i_4} \geq b_4$ do $L_4 \leftarrow a_4'$; $S_{i_4} \leftarrow S_{i_4} - b_4$

where $b_1 > 0$, $b_2 > 0$, $b_3 > 0$, $b_4 > 0$. Since ef is not active, $S_{i_2}[e(w)] < b_2$. Now since f is active, $S_{i_2}[w] \geq b_2$; hence, $i_1 \neq i_2$. By the reasoning, $i_1 = i_2 = i_3 = i_4 = i$. Since eh is active, $S_i[w] \geq b_1 + b_4$. Since fh is not active, $S_i[w] < b_2 + b_4$; therefore, $b_2 > b_1$. Again since fg is active, $S_i[w] \geq b_2 + b_3$. Since eg is not active, $S_i[w] < b_1 + b_3$; therefore, $b_1 > b_2$. This is a contradiction, and thus $\mathcal{P}$ does not define Π_2 . □

Theorem 12. PV multiple → PV chunk.

Proof. Use Theorems 10 and 11 and Corollary 5. □

Our final goal is to prove that PV chunk → PV, PV chunk → up/down, PV multiple → PV, PV multiple → up/down. In order to prove this we will consider the slice $\Pi_3 = \{\Lambda, a, b, c, d, ab, ba, ac, ca, bc, cb\}$.

This slice is related to the "bounded first reader-writer problem" (Lipton [11]). Consider the system of processes BRWI whose actions are:

writer

(1) when $L = 1 \wedge n = 0$ do $L \leftarrow ?$; $b \leftarrow b+1$

(2) when $L = 2$ do $L \leftarrow 3$

(3) when $L = 3$ do $L \leftarrow 1$; $b \leftarrow b-1$

reader-i $(1 \leq i \leq 3)$

(1,i) when $L_i = 1 \wedge b = 0 \wedge a \leq 2$ do $L_i \leftarrow 2$; $a \leftarrow a+1$

(2,i) when $L_i = 2$ do $L_i \leftarrow 3$

(3,i) when $L_i = 3$ do $L_i \leftarrow 2$; $a \leftarrow a-1$.

The states of BRW1 are of the form

$$(L, L_1, L_2, L_3, (a,b)) ;$$

the initial state is $(1, 1, 1, 1, (0,0))$. BRW1 differs from RW1 only in the requirement that action $(1,i)$ must check that there are at most 2 readers using the resource $(a \leq 2)$. Informally, BRW1 represents the behavior of the first reader-writer problem (Courtois, Heymans, Parnas [2]) with the additional requirement that "at most 2 of the 3 readers can be reading at once." Note, BRW1 implicitly defines Π_3 .

Theorem 13. The class of PV multiple (PV chunk) systems of processes defines Π_3 .

Proof. Since BRW1 implicitly defines Π_3 and BRW1 is a PV chunk system of processes, PV chunk defines Π_3 . Now consider the PV multiple system of processes whose actions are:

(1) when $L_1 = 1 \wedge S_1 > 0 \wedge S_4 > 0$ do $L_1 \leftarrow 2$; $S_1 \leftarrow S_1-1$; $S_4 \leftarrow S_4-1$

(2) when $L_2 = 1 \wedge S_2 > 0 \wedge S_4 > 0$ do $L_2 \leftarrow 2$; $S_2 \leftarrow S_2-1$; $S_4 \leftarrow S_4-1$

(3) when $L_3 = 1 \wedge S_3 > 0 \wedge S_4 > 0$ do $L_3 \leftarrow 2$; $S_3 \leftarrow S_3-1$; $S_4 \leftarrow S_4-1$

(4) when $L_4 = 1 \wedge S_1 > 0 \wedge S_2 > 0 \wedge S_3 > 0$ do $L_4 \leftarrow 2$; $S_1 \leftarrow S_1-1$; $S_2 \leftarrow S_2-1$; $S_3 \leftarrow S_3-1$.

The states of this system of processes are of the form

$$(L_1, L_2, L_3, L_4, (S_1, S_2, S_3, S_4)) ;$$

the initial state is $(1, 1, 1, 1, (1,1,1,2))$. The active timings of this system of processes are $\{\Lambda, 1, 2, 3, 4, 13, 31, 23, 32, 12, 21\}$; hence, the class of PV multiple systems of processes defines Π_3 . □

Theorem 14. The class of up/down systems of processes does not define Π_3 .

Proof. Suppose that $\mathcal{P} = \langle\{f_1, f_2, f_3, f_4\}, D, w\rangle$ is an up/down system of processes that defines Π_3 . Let " $(L_1, \ldots, L_n, (C, S_1, \ldots, S_n))$ " be a typical element in D . Since $\mathcal{P}$ defines Π_3 , we can assume that the active timings of $\mathcal{P}$ are

(1) $\{\Lambda, f_1, f_2, f_3, f_4, f_1f_2, f_2f_1, f_1f_3, f_3f_1, f_2f_3, f_3f_2\}$.

Suppose that f is an action in $\mathcal{P}$; select an action g in $\mathcal{P}$ with $f \neq g$. By (1), there is a timing α such that αf is active, αg is active, and αgf is not active. By properties I and II, not process $(1,1)$; by property III, f is in pointer-set (αg) . Since f is not in ready-set (αg) , f is a synchronizer by Theorem 2. Therefore, each action in $\mathcal{P}$ is a synchronizer. We now assert that

(2) for all actions g in $\mathcal{P}$, for some F and S_i , g is an $F : \text{down } (S_i)$.

Conversely assume that f is an $F : \text{up } (S_j)$; select an action g such that $f \neq g$. By (1) there is a timing α such that αf is active, αg is active, and αfg is not active. Suppose that g is either an

H : down (S_i) or an H : up (S_i) . As before g is in pointer-set (αf) . Since g is not in ready-set (αf) ,

$$\sum_{S_k \in H} S_k[\text{value } (\alpha f)] < 0 .$$

Since f is a F : up (S_j) , for all S_k , $S_k[\text{value } (\alpha f)] \geq S_k[\text{value } (\alpha)]$. Thus,

$$\sum_{S_k \in H} S_k[\text{value } (\alpha)] < 0$$

which is a contradiction, for g is in ready-set (α) . Therefore, (2) is true. Let f_i $(1 \leq i \leq 4)$ be an F_i : down (S_{B_i}) . We now assert that

(3) for $i \neq j$, B_i is in F_j .

Conversely assume that for $i \neq j$, B_j is not in F_j . Again by (1), there is a timing α such that αf_i is active, αf_j is active, and $\alpha f_i f_j$ is not active. As before, f_j is in pointer-set (αf_i) . Since f_j is not in ready-set (αf_i) ,

$$\sum_{S_k \in F_j} S_k[\text{value } (\alpha f_i)] < 0 .$$

Also since f_j is in ready-set (α) ,

$$\sum_{S_k \in F_j} S_k[\text{value } (\alpha)] \geq 0 .$$

This is a contradiction: since for each S_k in F_j , $S_k[\text{value } (\alpha f_i) =$

$S_k[\text{value}(Qf_1)]$. Therefore, (3) is true. By (1), f_1f_2 is active, f_4 is active, and f_4f_2 is not active. By (2) and (3),

$$\sum_{S_k \epsilon F_2} S_k[\text{value}(f_1)] = \sum_{S_k \epsilon F_2} S_k[\text{value}(\wedge)] - 1$$

and

$$\sum_{S_k \epsilon F_2} S_k[\text{value}(f_4)] = \sum_{S_k \epsilon F_2} S_k[\text{value}(\wedge)] - 1 .$$

Since f_1f_2 is active,

$$\sum_{S_k \epsilon F_2} S_k[\text{value}(f_1)] > 0 .$$

Thus,

$$\sum_{S_k \epsilon F_2} S_k[\text{value}(f_4)] > 0 .$$

Therefore, f_4f_2 is active; and this is a contradiction. □

Corollary 15. The class of PV systems of processes does not define Π_3 .

Proof. Suppose that $\mathcal{P}$ is a PV system of processes such that $\mathcal{P}$ defines Π_3 . Then since up/down is a generalization of PV, there is an up/down system of processes Q such that Q simulates $\mathcal{P}$. By the invariance theorem, Q implicitly defines Π_3 ; hence, up/down defines Π_3 . This is a contradiction. □

Theorem 16. PV chunk → PV, PV chunk → up/down, PV multiple → PV, PV multiple → up/down.

Proof. Use Theorems 13 and 14 and Corollaries 5 and 15. □

The results stated in the introduction are now proved. In particular, Theorems 9, 12, and 16 prove the results presented in Figure 1. Note, whether or not PV chunk ⇒ PV multiple is an open question. Also observe that not PV ⇒ PV chunk, not PV ⇒ PV multiple, and not PV ⇒ up/down. This follows since PV chunk, PV multiple, and up/down are generalizations of PV.

6. Conclusion

We have shown that there are differences among the classes of systems of processes: PV, PVchunk, PVmultiple, and up/down. These differences are based on the relation simulate. Since weakening simulate leads to inefficient and pathological solutions to synchronization problems, it follows that the differences are of practical as well as theoretical importance. These differences are proved in two steps: First, it is shown that there are local differences among the four classes. More exactly, it is shown that one class can define slices that another cannot. Second, the invariance theorem is used to conclude that there are "global differences" between these classes of systems of processes. More exactly, it is shown that one class cannot be simulated by another class.

A basic area of future research is the extension of this research to other classes of systems of processes. The paradigm is: Select a class of systems of processes and try to find slices that distinguish it from other classes. Interesting classes include: Petri Nets, Vector Addition Systems (Keller [7]), and further generalizations of PV. Some of these classes are studied in Lipton, Snyder, Zalcstein [11].

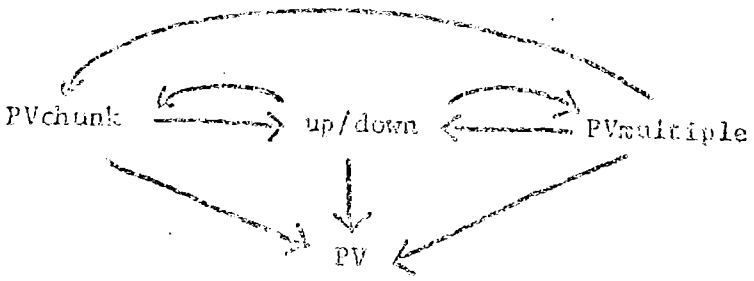

Figure 1. Results.

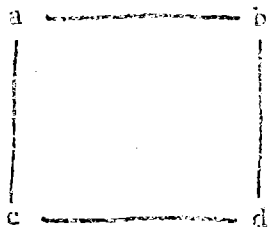

Figure 2. Adjacent nodes stop each other.

Acknowledgments

The author wishes to thank L. Snyder, R. Tuttle, and Y. Z[illegible]estein for their helpful comments.

REFERENCES

[1] V. G. Cerf. Multiprocessors, Semaphores, and a Graph Model of Computation, Ph.D Thesis, UCLA, 1972.

[2] P. J. Courtois, F. Heymans, D. L. Parnas. Concurrent Control with "Readers" and "Writers." CACM 14(10) : 667-668.

[3] E. W. Dijkstra. Cooperating Sequential Processes, Programming Languages, edited by F. Genuys. 43-112.

[4] E. W. Dijkstra. Hierarchical Orderings of Sequential Processes. Acta Informatica 1(2) : 115-138.

[5] A. W. Holt and F. Commoner. Events and Conditions, Applied Data Research, New York, 1970.

[6] R. M. Karp and R. E. Miller. Parallel Program Schema. JCSS 3(2) : 147-195.

[7] R. M. Keller. On Maximally Parallel Schemata. In Proceedings of Fifth Annual Symposium on Switching and Automata Theory, 1970.

[8] S. R. Kosaraju. Limitations of Dijkstra's Semaphore Primitives and Petri Nets, Technical Report 25, The Johns Hopkins University, 1975.

[9] R. J. Lipton. Limitations of Synchronization Primitives with Conditional Branching and Global Variables. Proceedings of Sixth Annual ACM Symposium on Theory of Computing, April 1974.

[10] R. J. Lipton. On Synchronization Primitive Systems. Yale Computer Science Research Report No. 22, 1973.

[11] R. J. Lipton, L. Snyder, and Y. Zalcstein. A Comparative Study of Models of Parallel Computation. To appear in the Proceedings of the 15th Annual Symposium on Switching and Automata Theory, 1974.

[12] S. S. Patil. Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes. Project MAC Computation Structures Group Memo 57.

[13] J. L. Peterson. Modelling of Parallel Systems. Ph.D Thesis, Stanford University, 1973.

[14] H. Vantilborgh and A. van Lamsweerde. On an Extension of Dijkstra's Semaphore Primitives. Information Processing Letters 1 : 181-186.