ON THE COMPLEXITY OF DECIDING FAIR TERMINATION OF PROBABILISTIC CONCURRENT FINITE-STATE PROGRAMS

Louis E. Rosier and Hsu-Chun Yen

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-85-22 October 1985

On The Complexity of Deciding Fair Termination of Probabilistic Concurrent Finite-State Programs

Louis E. Rosier and Hsu-Chun Yen

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

Abstract

In this paper, we consider the fair termination problem for probabilistic concurrent finite-state programs. We analyse the complexity of deciding, given a system of probabilistic concurrent finite-state programs, whether the system will terminate (with probability 1) under five different fairness assumptions. It turns out that these problems are either complete for PTIME, or the second or third levels of the alternating logspace hierarchy, even when the number of programs is fixed. However, when a more succinct representation of the inputs is allowed, the problems become either EXPTIME or PSPACE complete. In particular, for those instances of size n with k concurrent programs, we show lower bounds of $\Omega(n^{(k-6)/96})$ deterministic time and (k-11)/4*log n 1-alternating ATM space ((k-17)/6*log n 2-alternating ATM space) (with a binary tape alphabet). Corresponding upper bounds of $O(n^{d*k})$ deterministic time and d*k*log n 1(2)-alternating ATM space (for some positive constant d) are also shown. It has been conjectured that there are problems solvable by nondeterministic TM's in k*log n space (and binary tape alphabet) that require $\Omega(n^k)$ deterministic time. Now these problems are in PTIME for every fixed value of k; but if we accept the conjecture then the order of the polynomial, in each case, grows linearly with the value of k. Hence, the same must be true of problems requiring k*log n 1(or 2)-alternating ATM space (with a binary tape alphabet).

1. Introduction

The use of randomization (or probabilities) seems to be a recent trend in the design of concurrent algorithms. In comparison to their non-probabilistic counterparts, probabilistic programs often enjoy the merit of being simplier and easier to design. They also often require less shared memory. (c.f. [13, 16, 17].) Perhaps more importantly, sometimes they can produce solutions, under certain constraints, that can not be achieved without using randomization. For example, in [13] it was shown that, there is no deterministic, deadlock-free, symmetric and truly distributed solution for the dining philosophers

problem. But, if randomization is allowed it becomes possible to design a relatively simple algorithm that meets these specifications. Unfortunately, probabilistic programs suffer the drawback that few verification techniques are well developed. Recently, however, the termination problem (i.e., the problem of deciding, given a system of concurrent programs, whether the system will with probability 1 terminate) has received some attention. In particular, people are interested in the fair termination problem; i.e., does the system terminate (with probability 1) providing the scheduling of programs is, in some sense, fair [4, 5, 12, 15, 21]. So far the research in this area seems to emphasize two directions. The first concentrates on designing inference rules that can serve as a mathematical foundation for proving fair termination. (c.f. [12, 15].) The second, on the other hand, concerns itself with algorithmically solving special cases - such as the case where the programs are finite-state. Consequently for such cases, the space and/or time complexity for determining fair termination becomes an issue. (See e.g. [4], [5] and more recently [21].) So far, however, no attempt has been made to compare the complexity results under different fairness constraints. Neither has any result appeared on studying the complexity of the problem in terms of the number of concurrent programs (i.e., the degree of concurrency).

Therefore, the main contribution of this paper is that, we give a comprehensive analysis of the fair termination problem for probabilistic concurrent finite-state programs under 5 types of fairness constraints. It turns out that the problems are either complete for PTIME or $A\Sigma_2^L$ or $A\Sigma_3^L$, even when the number of programs is a fixed constant. ($A\Sigma_2^L$ ($A\Sigma_3^L$) denotes the second (third) level of the alternating logspace hierarchy [2].) Here it is assumed that the inputs are essentially provided in the form of the global state graph - in a manner consistent with the definitions given in [4]. However, when a more succinct representation of the inputs is allowed (one in which each program is described only in terms of the memory to which it has access), the problems become either EXPTIME or PSPACE complete. In particular, for those instances of size n with k concurrent programs, we show lower bounds of $\Omega(n^{(k-6)/96})$ deterministic time and (k-11)/4*log n 1-alternating ATM space ((k-17)/6*log n 2-alternating ATM space) (with a binary tape alphabet)[2]. In the latter case, this means that the problems are hard for the class of problems solvable by k*log n space bounded ATM's which use a binary tape alphabet and make at most one (two) alternation(s) during the course of a computation. Upper bounds of $O(n^{d*k})$ deterministic time (for some positive contant d) are also shown. It has been conjectured that there are problems solvable by nondeterministic TM's in k*log n space (and binary tape alphabet) that require $\Omega(n^k)$ deterministic time. So while the problems we consider are in PTIME for every fixed value of k, the order of the polynomial (for sure in some cases - more than likely in the others) grows linearly with the value of k - and thus the problems are, in some sense, gradually intractable.

Problems requiring $\Omega(n^k)$ deterministic time were previously shown in [1]; while problems requiring k*log n nondeterministic TM space (with a binary tape alphabet) were considered in [9]. See also [7, 18]. The fact that the problems become more difficult, with our more concise representation of the instances,

is not really surprising. For example, [11] showed earlier that the lockout problem for similarly expressed instances of (non-probabilistic) concurrent systems was EXPTIME complete. The important contribution here is that we explain the role played by the degree of concurrency in determining the problem's complexity. We have also considered five different types of fairness and shown that the problem complexity depends on the type of fairness involved. Lastly, one should note that not many problems have appeared in the literature which are proven complete for various classes of alternating space or which are gradually intractable in the manner described here.

The remainder of this paper is organized as follows. In section 2, we define the model of probabilistic concurrent finite-state programs used in [4] as well as five types of fairness criteria (taken from [12, 15]). Based on this model, we show that the fair termination problem depends only on the topological structure of the program's transitions, and not on the particular values of the nonzero transition probabilities. This constitutes a generalization of a result shown in [4]. Therefore, we are able to use a graph-theoretical approach to determine whether programs will terminate with probability 1 under each of the fairness constraints. In section 3, we investigate upper and lower complexity bounds for the problems defined in section 2. In section 4, we consider the case when a more succinct representation of the inputs is allowed.

2. The model

Informally, a system of probabilistic concurrent programs consists of several programs running in parallel. In addition, probabilistic moves are allowed in each program; i.e., the next move to be executed in each program is chosen in a probabilistic manner among those executable moves. (One might imagine this as flipping a perhaps biased die to decide the next move when several moves are available.) On the top of the system, a *scheduler* is involved to determine which program to execute (or schedule) next. As we shall see later, our model is essentially that of [4] although our notations are somewhat different. (See [4] for a detailed comparison.) Based on this model, our goal is to determine, given a system of probabilistic concurrent programs, whether the system will terminate *properly* (i.e., with probability 1) under a given fairness criteria.

First, we define our model of probabilistic concurrent programs. Formally, a system of k probabilistic concurrent finite-state programs (denoted by $FPCP_k$, or FPCP if the parameter k is not important) is a 4-tuple (S,s_0,δ,X) where:

S is a finite set of states,

s_0 ($\in$ S) is the initial state,

X ($\subseteq$ S) is the set of termination states, and

$\delta=\{\delta_1,...,\delta_k\}$ is the set of transition functions which satisfies:

- $\forall\ 1\leq i\leq k$, δ_i: $S \rightarrow 2^{S\times[0,1]}$,
- (Let $P^i_{q,r}$ be the number t $(0\leq t\leq 1)$ such that $(r,t)\in\delta_i(q)$.) $\forall\ s\in S$, if $(s',t)\in\delta_i(s)$ for some $s'\in S$ and $t>0$, then $\Sigma_{s'\in S}P^i_{s,s'}=1$.

Let the classes of such systems be denoted by $\mathcal{FPCP}_k$ and $\mathcal{FPCP}$, respectively. We assume that X is a "sink", i.e., if a state in X is reached, with probability 1 the system will stay in X. Intuitively, $P^i_{q,r}$, with its value ranging between 0 and 1, stands for the probability of reaching state r from state q by executing an atomic step of program i. Note that, unlike the definition in [4], here a program i can be "undefined" in some state q; i.e., $\Sigma_{r\in S}P^i_{q,r}=0$. The notion of "enabledness" can, therefore, be introduced. Program i is said to be *enabled* in state q iff $P^i_{q,r}\neq 0$ for some state r in S; otherwise, we say it is *disabled*. Hence, our problems are somewhat more general than those in [4], where each program is considered to be enabled in every state. As we will see later, this kind of generalization allows us to distinguish several fairness notions that are identical for the problem instances considered in [4]. A *schedule* of P is a partial mapping:

$$\sigma: \cup_{n=0}^{\infty}(\{s_0\}\times S^n)\rightarrow\{1,2,...,k\}$$

Given a schedule σ, the corresponding *computation tree* (or *computation*) is a tree with s_0 as its root and for every n, $(s_0,s_1,...,s_n)$ is a path in the tree iff for every j, $1\leq j\leq n$, there exists a program i_j $(1\leq i_j\leq k)$ such that $\sigma(s_0,s_1,...,s_{j-1})=i_j$ and $P^{i_j}_{s_{j-1},s_j}>0$. The reader should note that, in each state the program to be scheduled next depends not only on the current state, but also on the sequence of states (history) that have been visited thus far. Furthermore, since σ is a single-valued function, this method of scheduling is deterministic. Note that given a schedule, the corresponding computation tree is uniquely decided, and vice versa. Therefore, in the remainder of this paper, the words "schedule" and "computation tree" will be used interchangeably. A *computation path* (or *path*) e over a schedule σ is a sequence

$$e: s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \rightarrow \ ... \ \rightarrow s_m$$

where s_0 is the initial state, for each j, $1\leq j\leq m$, s_j is in S, $1\leq i_j\leq k$, $s_{i+1}=\sigma(s_0,...,s_i)$ and $P^{i_j}_{s_{j-1},s_j}>0$. (We also consider infinite computation paths. In this case $m=\infty$.) We define $\sigma_e(j)=i_j$, i.e., the j-th program scheduled in e. Let Δ be the set of all infinite sequences of states of S. Let $\Delta(s_0,...,s_n)=\{w\in\Delta|w=s_0,s_1,...,s_n,...\}$, i.e., the subset of Δ such that each element has $s_0,...,s_n$ as its prefix. Given a schedule σ, the probability measure μ_σ is defined by

$$\mu_\sigma(\Delta(s_0,...,s_n))=P^{i_1}_{s_0,s_1}*P^{i_2}_{s_1,s_2}*...*P^{i_n}_{s_{n-1},s_n},$$

where $i_j=\sigma(s_0,...,s_{j-1})$, j=1,...,n. Consider an arbitrary set of infinite paths L in σ. If, for some h, $L=\cup_{1\leq i\leq h}\Delta(s_0,...,s_{d_i})\cup L'$, where $\Delta(s_0,...,s_{d_i})\cap\Delta(s_0,...,s_{d_j})=\phi$ for $i\neq j$ and L' does not contain $\Delta(s_0,...,s)$ for any $s_0,...,s$, we define $\mu_\sigma(L)=\Sigma_{1\leq i\leq h}\mu_\sigma(\Delta(s_0,...,s_{d_i}))$; otherwise, $\mu_\sigma(L)=0$. (See [10] for a formal

definition of the probability measure concerning Markov chains.)

We now consider the following types of fairness. They were first defined and studied in [12]. (The names in parentheses refer to those used in [12].) Given a system $FPCP_k$ and a path e: $s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \rightarrow \ldots \rightarrow s_m \rightarrow \ldots$, we say e is:

1. *1_fair* (*impartial*) iff for every program i there are infinitely many j's such that $\sigma_e(j)=i$. (I.e., every program is scheduled infinitely often in e.)

2. *2_fair* (*just*) iff for every program i if there exists a number t such that i is enabled for every state s_j ($j \geq t$), then there are infinitely many l's such that $\sigma_e(l)=i$. (I.e., every program that is enabled continuously is scheduled infinitely often in e.)

3. *3_fair* (*fair*) iff for every program i if there are infinitely many j's such that i is enabled in s_j, then there must be infinitely many l's such that $\sigma_e(l)=i$. (I.e., every program that is enabled infinitely often is scheduled infinitely often in e.)

The reader should realize, at this point, that every 1_fair path is 3_fair, and every 3_fair path is 2_fair. (Now [4] considers only systems where each program is enabled in every state. In such a context, the above three types of fairness are identical.) These types of fairness serve in many cases as an appropriate notion of what it means for a computation to be "fair". See e.g. [4, 12]. There are situations, however, for which they are not deemed powerful enough. (See [15].) As a result, the concept of *state-fairness* was introduced [15]. A path is said to be *state-fair* if it is the case that, whenever a program is enabled in some state that occurs infinitely often, then that program must be scheduled infinitely many times in that state. Compared to fairness types 1-3 defined previously, state fairness is defined on the *transition* level; and hence is a more severe constraint. (Note that fairness types 1-3 are defined on the *process* level.) An interesting question, therefore, arises. What happens if one imposes state fairness in conjunction with each of the earlier types of fairness. One can readily see that, a state fair path is also 2 and 3_fair. The combination of fairness types 1-3 and state fairness ends up defining two additional notions of fairness: a computation path e is

4. *4_fair* iff it is state-fair.

5. *5_fair* iff it is state-fair and 1_fair.

Given an FPCP P ($=(S,s_0,\delta,X)$) and a schedule σ, let L^t be the set of t_fair paths of σ. σ is said to be *t_fair* (t=1, 2, 3, 4 or 5) iff $\mu_\sigma(L^t)=1$. Our main concern is to determine whether there exists a fair schedule such that P does not terminate with probability 1. More precisely, let $f_X^{(n)}(\sigma)$ be the probability of either:

1. reaching a state in X for the first time using exactly n moves under schedule σ, or

2. entering a state s (in n steps) in which no program is scheduled next.

Let $f_X^*=\Sigma_{n=0}^{\infty} f_X^{(n)}(\sigma)$. We further define $h_{s,X}^t=\inf\{f_X^*(\sigma)$: σ is a t_fair schedule starting at s$\}$. ($h_{s,X}^t$ is

abbreviated as h_X^t if s is the initial state.) The *non-termination problem* (NTP) is to decide, given an FPCP, whether $h_X^t<1$ for t=1, 2, 3, 4 or 5. The *termination problem* (TP) is the complement of the NTP. The NTP has been studied to some degree in [4, 5, 21]. In [4], it is shown that for FPCP's the NTP under 1_fairness depends only on the topological structure of the transition functions, and not on the particular probabilistic transition assignments. In other words, [4] showed that it was possible to use a purely combinatorial approach to solve the NTP. (See also [21].) In the remainder of this section, we generalize the results in [4] to deal with fairness types 2-5.

Given an FPCP $P=(S,s_0,\delta,X)$, we define its corresponding *execution graph* (hereafter, *graph*) (denoted by G_P) to be a directed labelled graph (V,E) where V=S and an edge $(q,r)_i$ (i is the label of the edge) is in E iff $P^i_{q,r}\neq 0$ (i.e., there is a nonzero probability for program i to reach state r from q in one step providing program i is scheduled). The node s_0 is called the *initial node*. A subgraph G' of G_P is said to be a *reachable* subgraph iff some node in G' is reachable from s_0. Given a set of nodes V', the *restricted* graph over V' is a subgraph G'=(V',E') of G_P such that an edge $(q,r)_i$ of E is in E' iff both q and r are in V' and there is no r' in V-V' such that $(q,r')_i \in E$. A reachable strongly connected component G'=(V',E') of G_P (=(V,E)) is called a *blackhole* iff G' satisfies:

1. G' is the restricted graph over V',
2. $V'\cap X=\phi$, and
3. $E'\neq\phi$.

A program i is said to be *active* in a blackhole G'=(V',E') iff there are states q and r in V' such that $(q,r)_i$ is in E'. A blackhole G' (of G_P) is said to be of

1. *type 1* iff every program i is active in G'.
2. *type 2* iff, if i is a program such that $\forall s\in V'\ \exists s'\in V$ such that $(s,s')_i$ is in E, then program i must be active in G'.
3. *type 3* iff, whenever i is a program such that $(q,r)_i$ is in E for some q in V' and r in V-V', then program i must be active in G'.
4. *type 4* iff for every q in V', there is no r' in V-V' and no j, $1\leq j\leq k$, such that $(q,r')_j$ is in E.
5. *type 5* iff it is type 1 and type 4.

In what follows, we show that for any FPCP, the particular nonzero transition probabilities have no effect on whether the FPCP will (fairly) terminate or not with probability 1. Based on this fact, we are then able to give a combinatorial solution of the NTP for each type of fairness. In order to show this, the following lemmas are required. They are, basically, extensions of the results given in [4], where only the 1_fairness was considered.

Lemma 2.1: Given an FPCP $P=(S,s_0,\delta,X)$, $\min\{ h^f_{s,X} \mid s\in S \}$ is either 0 or 1, where f =1, 2, 3, 4 or 5.

Proof. Since the proof of the same theorem (also called Zero-One Law) for type 1 fairness, given in [4], does not depend on any particular type of fairness, this lemma then follows. □

A slight modification of Theorem 2.1 in [4] yields:

Lemma 2.2: Let $P=(S,s_0,\delta,X)$ be an FPCP. If there exists some t_type blackhole in G_P, then there is a t_fair schedule σ that satisfies the following conditions:

1. $f^*_X(\sigma)<1$,
2. with probability 1, every "active" program is scheduled infinitely often in every path.

Lemma 2.3: Let $P=(S,s_0,\delta,X)$ be an FPCP. If there exists a t_fair (t=1, 2, 3, 4 or 5) schedule σ such that $f^*_X(\sigma)<1$, there must exist a t_type blackhole $G'=(V',E')$ in G_P such that the set of states occurring infinitely often in σ exactly coincides with V'.

Proof. Assume that σ is a t_fair schedule. According to Lemma 2.1, $f^*_X(\sigma)<1$ implies that there exists a state s such that, $h^t_{s,X}=0$ and s occurs in σ with non-zero probability. Consider the subtree of σ starting at s. Note that each path of the subtree is infinite; otherwise, $h^t_{s,X}>0$. Let s_1 be a state that occurs infinitely often in this subtree. Let V' and E' be the sets of states and edges reachable from s_1, respectively. It is reasonably easy to see that $G'=(V',E')$ is a blackhole, otherwise, we have $h^t_{s,X}>0$, which is a contradiction. Furthermore, we can also claim that G' is a t_type blackhole. If this is not the case, let $s_0,\ldots,s_1$ be the finite path that leads to the blackhole G' in σ. We immediately have $\mu_\sigma(s_0,\ldots,s_1)>0$. Let L denote the set of all t_fair paths in σ. Clearly, $\mu_\sigma(L)<1$, which implies that σ is not t_fair - a contradiction. The lemma then follows. □

Using the above lemmas, we can immediately derive necessary and sufficient conditions for an FPCP to (fairly) terminate with probability 1. We have:

Theorem 2.1: Given an FPCP P, let G_P denote its graph representation. P will <u>not</u> terminate with probability 1 under t_fairness (t=1, 2, 3, 4 or 5) iff, there exists a type t blackhole in G_P.

3. The complexity measured as a function of the size of the state space

In this section, we derive the complexity of the NTP (TP) for FPCP's under fairness types 1-5. We assume that an instance $P=(S,s_0,\delta,X)$, as described in the previous section, is encoded in the usual manner for labelled state graphs. Note that this representation amounts to expressing a system of concurrent programs in terms of its global state space. One should note that there are instances where this encoding scheme is quite verbose. In the next section, we explore a more succinct representation (one in which each

program is described only in terms of the memory to which it has access) and see how the complexity is altered.

Before proceeding, we first define some complexity classes that will be used throughout the rest of this paper. An *alternating Turing machine* (ATM) is a Turing machine with four kinds of states: namely, existential, universal, accepting and rejecting states. A universal state leads to acceptance iff all successors lead to acceptance; while an existential state leads to acceptance iff there exists a successor that leads to acceptance. Detailed definitions can be found in [2]. The classes of languages accepted by time (space) bounded ATM's were also defined in [2]. In particular, $A\Sigma_k^L$ refers to the set of languages accepted by O(log n) space-bounded ATM's in which the initial state is existential and the ATM is restricted to make at most k-1 alternations during the course of a computation. We refer to such ATM's as O(log n) space bounded (k-1)-alternating ATM's. $A\Pi_k^L$ (=co-$A\Sigma_k^L$) is the class of languages accepted by O(log n) space bounded (k-1)-alternating ATM's whose initial states are universal. Unless otherwise specified the reader may assume that all ATM's discussed in this paper have initial states which are existential. Note that $A\Sigma_1^L$=NLOGSPACE (or NL, for short) and that the entire hierarchy is contained in PTIME, i.e., $\cup_{k=1}^{\infty} A\Sigma_k^L \subseteq$ PTIME [2]. Other properties of this hierarchy as well as some complete problems for $A\Sigma_2^L$ (and $A\Sigma_3^L$) can be found in [19, 20].

We summarize the results of this section in Table 2.1, where PTIME-C, $A\Sigma_2^L$-C and $A\Sigma_3^L$-C stand for PTIME-complete and complete for the class of O(log n) space bounded 1-alternating and 2-alternating ATM's, respectively. Since the TP is the complement of the NTP, the complexity results for TP can be displayed simply by replacing $A\Sigma_2^L$-C with $A\Pi_2^L$-C and $A\Sigma_3^L$-C with $A\Pi_3^L$-C. We give the complexity of the problem for the case when the degree of concurrency is a fixed constant and when it is a problem parameter. (Actually it is an internal parameter since it is defined implicitly within P.)

	$\cup_k$ $\mathcal{FPCP}_k$	$\mathcal{FPCP}_k$, $k \geq 2$	$\mathcal{FPCP}_1$
type 1	PTIME-C	PTIME-C	$A\Sigma_2^L$-C
type 2	PTIME-C	PTIME-C	$A\Sigma_2^L$-C
type 3	PTIME-C	PTIME-C	$A\Sigma_2^L$-C
type 4	$A\Sigma_2^L$-C	$A\Sigma_2^L$-C	$A\Sigma_2^L$-C
type 5	$A\Sigma_3^L$-C	$A\Sigma_3^L$-C	$A\Sigma_2^L$-C

Table 3.1: The complexity of the fair NTP.

In [4], an algorithm was presented to check the termination (with probability 1) of FPCP's under 1_fairness. Using Theorem 2.1, one can easily extend their procedure for other types of fairness. Hence, we have:

Lemma 3.1: The fair NTP for $\mathcal{FPCP}$ is solvable in PTIME for fairness types 1-5.

Proof. The following algorithm is a slight modification of the one given in [4]. Let P be an instance of

the problem with k processes. Let n denote the size of P (i.e., the number of bits in the representation). Note that $k \leq n$.

Algorithm:

1. Construct $G_P=(V,E)$.
2. Let $Q=X$. Delete from G_P all nodes in Q and delete all edges $e=(q,r)_i$ such that
   a. $q \in V-Q$ and $r \in Q$, or
   b. $q,r \in V-Q$ and for some s in Q, $(q,s)_i \in E$.

   Let G' be the resulting graph.
3. Partition G' into strongly connected components. Let B be one component. Check to see if B is a t_type blackhole. If so, the algorithm terminates with the conclusion that P will not terminate with probability 1 under some type t fair schedule; otherwise, let $Q=Q \cup \{u|u$ is a node in B$\}$ and repeat steps 2 and 3.
4. If no such t_type blackhole exists, the algorithm concludes that P will always terminate with probability 1.

To determine the complexity of the above algorithm, first note that at most n iterations are needed. (Since the resulting subgraph G' in step 2 is proper.) As far as each step is concerned, steps 1 and 2 require time polynomial in n. Step 3 requires time polynomial in n for type 2-4 blackholes and time polynomial in n and k for type 1 and 5 blackholes. (Note that k appears when checking type 1 and 5 blackholes because, by definition, every program must be scheduled for fairness types 1 and 5.) The lemma then follows. □

Theorem 3.1: The NTP for $\mathcal{FPCP}$, under fairness types 1-3, is PTIME-complete. The result also holds for $\mathcal{FPCP}_k$, when k is a fixed constant greater than 1.

Proof. The fact that the problem is in PTIME follows directly from Lemma 3.1. In what follows, we show that it is also PTIME-hard. To show this, we reduce the path system problem, which is well-known to be PTIME-complete [8], to the NTP for FPCP's.

Recall that a path system is a 4-tuple $PS=(X,R,S,T)$ where X is a finite set of nodes, S ($\subseteq X$) is a set of starting nodes, T ($\subseteq X$) is the set of terminal nodes, and R ($\subseteq X \times X \times X$) is the set of rules. A node x in X is *admissible* iff either $x \in T$ or $\exists y,z \in X$ such that $(x,y,z) \in R$ and both y and z are admissible. PS is said to have a solution iff there is an admissible node in S.

Now given a path system $PS=(X,R,S,T)$, we show how to construct an $FPCP_2$ $P=(S',s_0,\delta,X')$ such that, PS has no solution iff $h^t_{X'}<1$ for t =1, 2 or 3. P will have the property that both processes are enabled in each state. Since for such instances 1, 2 and 3_fairness are the same, we only consider 1_fairness in the remainder of the proof. Let $R=\{r_1,...,r_d\}$, $S=\{s_1,s_2,...,s_m\}$ and $T=\{t_1,t_2,...,t_n\}$. For

convenience, we first construct the graph G_p=(V,E) as follows:

$V=X\cup\{s_0\}\cup\{x^1,...,x^d|\ \forall\ x\in X\text{-}T\}$,

E= the collection of edges added by 1-5 below:

(1) add $(s_0,s_1)_i$ for i=1 and 2,

(2) $\forall j,\ 1\le j\le m$, add $(s_j,s_{(j+1\ \underline{\text{mod}}\ m)})_i$ for i=1 and 2,

(3) $\forall j,\ 1\le j\le n$, add $(t_j,t_{(j+1\ \underline{\text{mod}}\ n)})_i$ for i=1 and 2,

(4) $\forall\ x\in X\text{-}T\text{-}S$, add $(x,s_0)_i$ for i=1 and 2,

(5) $\forall\ x\in X\text{-}T$, $\forall\ 1\le j\le d\text{-}1$, add $(x,x^1)_i$, (x^j,x^{j+1}) and (x^d,x) for i=1 and 2,

(6) $\forall j$ if $r_j=(x,y,z)$ is in R, add $(x^j,y)_1$ and $(x^j,z)_2$.

See Figure 3.1. The FPCP P is constructed from G_p by assigning probabilities uniformly to each choice of move, i.e., for each node, every outgoing transition concerning the same process i will be assigned the same probability. Furthermore, let X' (the set of termination states) equal T. We claim that PS has no solution iff P has a type 1 blackhole containing the node s_0. To show this, first assume that PS has no solution. Let A and B be the sets of nonadmissible and admissible nodes, respectively. Let $A'=\{s_0\}\cup\{a,a^1,...,a^d|a\in A\}$. Let $B'=\{b,b^1,...,b^d|b\in B\}$. According to the construction, it is reasonably easy to see that each node in A' connects to at most one node in B'. Furthermore, the subgraph containing A' is a strongly connected component. Since we chose X'=T, such a strongly connected component is obviously a type 1 blackhole. The only if part is proved.

Now, consider the case where $h^t_{X'}<1$. By Theorem 2.1 there must exist a type 1 blackhole. By construction then such a blackhole must contain every node in S. In other words, every node in S must occur infinitely often in every 1_fair schedule. If some node, say s, in S is admissible, there must exist a rule (s,x,y) such that both x and y are admissible. Furthermore, either x or y must occur infinitely often in the 1_fair schedule. Inductively some node in T must be in the 1_fair schedule, which obviously, is a contradiction. The if part is therefore proved.

Consequently, there exists a type 1 fair schedule σ iff $h^t_{X'}=0$. The theorem then follows. □

Note that two programs are required in the PTIME hardness proof of Theorem 3.1. If we restrict ourselves to the case where only one program is involved, a somewhat lower complexity result can be obtained. Before showing this, we need the following easy lemma:

Lemma 3.2: For an FPCP $P=(S,s_0,\delta,X)$ of <u>one</u> program, $h^t_X=1$ (t=1, 2, 3, 4 or 5) iff there is no blackhole in G_p.

Proof. The lemma is trivial since, in such cases, all blackholes are of type 1, 2, 3, 4 and 5. □

With this lemma, we can immediately derive:

Theorem 3.2: The fair NTP for $\mathcal{FPCP}_1$, under fairness types 1-5, is $A\Sigma_2^L$-complete.

Proof. First, we derive the upper bound. According to Lemma 3.2, an $FPCP_1$ P will not terminate properly iff there is a blackhole in G_P. We show how to construct an O(log n) space bounded ATM M that accepts a given $FPCP_1$ P iff G_P has this property. We now sketch the operation of M on an instance $P=(S,s_0,\delta,X)$ in $\mathcal{FPCP}_1$. Let $|P|=n$. Note that G_P is obtainable from P in deterministic logspace.

A computation of M has two phases - first the existential phase and then the universal phase. In the first phase, M (nondeterministically) traverses a path in G_P (of length at most n) and (nondeterministically) chooses a state, say q, of G_P at which to stop and enter the second phase. In the second phase M traverses all paths (of length n) in G_P that emanate from q. If during one of these traversals M encounters a state of X or a deadend state it immediately rejects; otherwise it accepts. One can easily see that, M accepts G_P iff there exists a reachable node q such that q can not reach a deadend state or a state in X; i.e., it can reach a blackhole. Furthermore, during the course of any computation the work space needed is logarithmic in n and the number of alternations required is one. The details of M's construction will be left to the reader.

Now, we show the lower bound. Let M be an O(log n) space bounded 1-alternating ATM. Given an input string x, we show how to construct an $FPCP_1$ P in such a way that, M accepts x iff P will not properly terminate. Let |x| denote the length of x. A configuration of M is a 3-tuple (p,i,s), where p is the current state, i is the input head position, and s is the contents of the work tape (including the head position). Since M uses only O(log |x|) space, the number of distinct configurations is polynomial in |x|. A configuration (p,i,s) is called a universal (existential, accepting, rejecting) configuration iff p is a universal (existential, accepting, rejecting) state. Let V be the set of all configurations of M on x. For a configuration c, let $\#_c$ be the number of outgoing transitions of c. Now $P=(S,s_0,\delta,X)$ is constructed as follows.

1. $S = V \cup (V \times V)$,

2. $s_0 = c_0$, where c_0 is the initial configuration,

3. δ:

   $(q,1/\#_p) \in \delta(p)$, where p is an existential configuration and p can reach q in one step,

   $([r,r],1) \in \delta(r)$ for every universal configuration r,

   $\forall\, r \in V\ ([q,r],1/\#_p) \in \delta([p,r])$, where p is a universal configuration and p can reach q in one step,

   $\forall\, r \in V\ ([r,r],1) \in \delta([v,r])$ if v is an accepting configuration,

4. $X = \{[w,r] \mid [w,r] \in S$ and w is a rejecting configuration$\}$.

See Figure 3.2. Clearly, M accepts x iff there is a blackhole in $G_{\mathcal{P}}$. Furthermore, the above construction can easily be carried out in deterministic logspace. This completes the proof. □

Now, for solving the fair NTP, for fairness type 4, it suffices to find a type 4 blackhole, which is essentially the same as determining the fair NTP for $\mathcal{FPCP}_1$. Hence, the same proof as was used in Theorem 3.2 can be applied here. We immediately have:

Theorem 3.3: The fair NTP for $\mathcal{FPCP}$ and $\mathcal{FPCP}_k$, under 4_fairness, is $A\Sigma_2^L$-complete.

Lastly, we consider the fair NTP for FPCP's under 5_fairness. From Theorem 2.1, we have that an FPCP $\mathcal{P}$ does not terminate properly under 5_fairness iff there exists a type 4 blackhole in which every program is scheduled. In the 4_fairness case the algorithm only had to find a type 4 blackhole; but in this case it must also check that every program is scheduled in the blackhole. As we will see in the following theorem, this extra checking step "increases" the problem complexity.

Theorem 3.4: The fair NTP for $\mathcal{FPCP}$ and $\mathcal{FPCP}_k$, $k \geq 2$, under 5_fairness, is $A\Sigma_3^L$-complete.

Proof. First we show the upper bound. To prove this, we construct an O(log n) space bounded 2-alternating ATM M that accepts a given FPCP $\mathcal{P}$ iff $\mathcal{P}$ has the properties mentioned above. We now sketch the operation of M on an instance $\mathcal{P}=(S,s_0,\delta,X)$ with k concurrent processes. Let $|\mathcal{P}|=n$ and note that $k \geq n$. Recall that $G_{\mathcal{P}}$ is obtainable from $\mathcal{P}$ in deterministic logspace.

Now a computation of M consists of four phases, A, B, C and D which run in sequence. The states entered in phases A, B and D (C) will be existential (universal). In phase A, M (nondeterministically) traverses a path in $G_{\mathcal{P}}$ (of length at most n) and (nondeterministically) chooses a state, say q, of $G_{\mathcal{P}}$ at which to stop and enter phase B. In phase B, M (nondeterministically) traverses a path in $G_{\mathcal{P}}$ (of length at most k*n) that begins at q and which passes through at least one move of each of $\mathcal{P}$'s k processes before ending in the same state q. (If M traverses k*n steps in $G_{\mathcal{P}}$ which fail to meet these requirements, then M immediately rejects the input.) In phase C, M generates a configuration for each state reachable in $G_{\mathcal{P}}$ from q. The depth of the tree created in this phase need be no greater than n. In phase D, M, from each of the configurations (states) generated in phase C, (nondeterministically) traverses a path in $G_{\mathcal{P}}$ (of length at most n). If during this traversal M encounters a state in X or a deadend state it immediately rejects; otherwise it accepts. Clearly the computation described only requires O(log n) space and can be carried out by a 2-alternating ATM. Furthermore, such a machine can easily be constructed from $\mathcal{P}$ in deterministic logspace. The fact that M accepts $\mathcal{P}$ iff it has the correct properties follows from the observation that the state q, found in phase A, is in a type 5 blackhole iff phase B can succeed and each state of $G_{\mathcal{P}}$ found in phase C cannot reach a deadend state or a state in X.

Now, we show the lower bound. Here, we reduce an arbitrary problem in $A\Sigma_3^L$ to the NTP. Let M be an O(log n) space bounded 2-alternating ATM. Given an input string x, we show how to construct an $FPCP_2$ P with the property that, M accepts x iff P will not properly terminate under type 5 fairness. Let |x| (=n) denote the length of x. A configuration of M is a 3-tuple (p,i,s), where p is the current state, i is the input head position, and s is the content of the work tape (including the head position). Let c_0 be the initial configuration. Since M uses only O(log |x|) space, the number of distinct configurations is therefore polynomial in |x|. A configuration (p,i,s) is called a universal (existential, accepting, rejecting) configuration iff p is a universal (existential, accepting, rejecting) state. Let V (E, U) be the set of (existential, universal) configurations of M on x. Let $\#_c$ denote the number of outgoing transitions of a configuration c. Now $P=(S,s_0,\delta,X)$, with two programs A and B, is constructed as follows.

1. $S = V \cup (V\times V) \cup (V\times V\times V) \cup \{[\theta,r]|r\in U\}$,

2. $s_0 = c_0$,

3. δ: For each configuration p, q$\in$V, v$\in$U and t$\in$E, we have:

   a. δ_A:

      $(q,1/\#_p)\in\delta_A(p)$ if p can reach q in one step,

      $([r,r],1)\in\delta_A(r)$,

      $([q,r],1/\#_p)\in\delta_A([p,r])$ if p can reach q in one step,

      $([t,t,r],1)\in\delta_A([t,r])$,

      $([q,t,r],1/\#_p)\in\delta_A([p,t,r])$ if p can reach q in one step,

      $([t,t,r],1)\in\delta_A([v,t,r])$ if v is a rejecting configuration,

      $([\theta,r],1)\in\delta_A([v,t,r])$ if v is an accepting configuration,

      $([r,r],1)\in\delta_A([\theta,r])$.

   b. δ_B: $([\theta,r],1)\in\delta_B([\theta,r])$, if r$\in$U.

4. $X = \phi$.

See Figure 3.3. We can now argue that M accepts x iff there exists a type 5 blackhole in G_P. The crucial point is that in order for B to be scheduled infinitely often (which is required in a type 5 blackhole), some state of the form $[\theta,r]$ must occur infinitely often. In other words, each time A reaches a state of the form [s,t,r], A must be able to visit at a later time some state [v,t,r], where v is an accepting configuration. (Otherwise, A will get stuck without being able to visit some state $[\theta,r]$ ever again.) Hence, M accepts x iff there exists a type 5 blackhole. Since the above construction can easily be carried out in deterministic logspace, this completes the proof. □

4. The complexity when a more succinct representation of the input is assumed

In this section, we investigate the complexity of the fair NTP (TP) for $\mathcal{FPCP}$'s over a different representational scheme. For this scheme, the NTP for fairness types 1-3 is shown to require $\Omega(n^{(k-6)/96})$ deterministic time; while for fairness type 4(5) k*log n 1(2)-alternating ATM space is required. Corresponding upper bounds are also derived. The results are illustrated in the following table.

	Lower bound	Upper bound
type 1	$\Omega(n^{(k-6)/96})$det. time	$O(n^{d*k})$ det. time
type 2	$\Omega(n^{(k-6)/96})$det. time	$O(n^{d*k})$ det. time
type 3	$\Omega(n^{(k-6)/96})$det. time	$O(n^{d*k})$ det. time
type 4	(k-11)/4*log n 1-alt. space	O(k*log n) 1-alt. space
type 5	(k-17)/6*log n 2-alt. space	O(k*log n) 2-alt. space

Table 4.1: The complexity of the fair NTP for $\mathcal{FPCP}_k$.
Here, d denotes a positive constant.

We now describe the representation of FPCP's assumed in this section. First some additional notation is in order. Let V be a finite set of memory bit locations. Let $\mathcal{A}(V)$ be the set of configuations or states that V can be in. (Note, of course, that $|\mathcal{A}(V)|=2^{|V|}$.) Now an $FPCP_k$ P is a 4-tuple (S,s_0,δ,X) where:

- $S=\{S_{ij}|S_{ij}$, $1\leq i,j\leq k$, is the finite set of memory locations shared between process i and process j (each memory location denotes a single bit). S_{ii}, $1\leq i\leq k$, denotes the local memory of process i; i.e., the memory which is not shared.$\}$,
- $s_0 \in \mathcal{A}(\cup_{1\leq i,j\leq k} S_{ij})$ is the initial system state,
- $X=(X_1, \ldots, X_k)$, where each X_i, a subset of $\mathcal{A}(\cup_{1\leq j\leq k} S_{ij})$, is the set of termination states for each process i, and
- $\delta=\{\delta_1,\ldots,\delta_k\}$ is the set of transition functions satisfying: $\forall i$, $1\leq i\leq k$, δ_i: $\mathcal{A}(\cup_{1\leq j\leq k} S_{ij}) \rightarrow 2^{\mathcal{A}(\cup_{1\leq j\leq k} S_{ij})\times[0,1]}$.

Again, we assume that X is a sink; i.e., if the system reaches a state in which each process i, $1\leq i\leq k$, is in a state in X_i then the system will remain in X.

Let $S_i = \cup_{j=1}^{k} S_{ij}$, $1\leq i\leq k$. Then in this representation a state of process i is a configuration of S_i, which is exactly the memory to which process i has access. The size of P is polynomial in $\Sigma_{i=1}^{k}|\mathcal{A}(S_i)|$, the sum of the number of configurations attainable in the memory accessible to each process. The size of P represented in the manner described in section 2 depends on $|\mathcal{A}(\cup_{i=1}^{k} S_i)|$, the number of global memory configurations which can be exponential in the former measure. Let $|P|$ denote the size of P as described in this section. Then the size of P as described in the previous section is bounded by a polynomial in $|P|^k$. Thus, according to Theorem 3.1, the NTP for fairness types 1-3 is solvable in $O(n^{d*k})$ deterministic time

(for some positive constant d). Also, according to Theorems 3.3 and 3.4, the NTP under 4(5)_fairness is solvable in O(k*log n) space by an ATM which makes at most one (two) alternations; i.e., in $A\Sigma_2^{O(k*\log n)}$. Since it was shown in [2] that for any fixed r, $A\Sigma_r^{k*\log n} \subseteq DSPACE((k*\log n)^2)$, we have:

Theorem 4.1:

1. The NTP under fairness types 1-3 is in EXPTIME.

2. The NTP under fairness types 4-5 is in PSPACE.

Let $C(n,k)$ to be the restricted class of FPCP's with k programs such that each program has at most n states. It is precisely this subclass for which we derive our lower bounds. Note that for P in $C(n,k)$, $|P|$ is bounded by a polynomial in n and k. For $n \geq 2$, the size of $P \in C(n,k)$ as measured in the previous section is $O(n^{c*k})$ for some positive constant c. Hence, for some positive constant d, the NTP for $C(n,k)$ under 1-3 (4, or 5) fairness is solvable in $O(n^{d*k})$ deterministic time (O(k*log n) 1 or 2-alternating ATM space). Now, we derive the lower bounds. Before doing this, the following definitions are required.

Let $\Sigma=\{0,1\}$. A mapping g: $\Sigma^* \rightarrow \Sigma^*$ is said to be *computable* in S(n) space (T(n) time) iff there exists a deterministic Turing machine M such that, given an input $x \in \Sigma^*$, M will output g(x) using at most S(|x|) space (T(|x|) time). Now, consider two problems L and L' over Σ^*. L is said to be $(S(n),Q(n))_{space}$ $((T(n),Q(n))_{time}$-reducible to L' iff, there exists a mapping g computable in S(n) space (T(n) time) such that:

1. $x \in L$ iff $g(x) \in L'$, and

2. $\forall x \in \Sigma^*$, $|g(x)| \leq Q(|x|)$.

Let C be a class of problems over Σ. L is said to be *C-hard* with respect to $(S,Q)_{space}$-reducibility ($(T,Q)_{time}$-reducibility) if for every L' in C, there exists a constant c such that, L' is $(S(n),c*Q(n))_{space}$ $((T(n),c*Q(n))_{time})$ reducible to L. (See [9].) The following results were given in [9] (or, can be obtained by a straightforward generalization of those given in [9]). See also [7].

Lemma 4.1: If a function f: $\Sigma^* \rightarrow \Sigma^*$ is computable by an S(n) space bounded TM M with tape symbols $\{0,1,\#\}$ such that at any time the work contains at most k #'s, then f is computable in S(n) + (k+2)*log S(n) space by a TM M' with tape symbols $\{0,1\}$.

Lemma 4.2: Let L ($\subseteq \Sigma^*$) be C-hard with respect to $(S,Q)_{space}$-reducibility.

(1) If L is solvable in T(n) deterministic time, then for any problem L' in C, there are constants c_1 and c_2 such that L' is solvable in $T(c_1*Q(n))+c_2*n*S(n)*2^{S(n)}$ deterministic time.

(2) If L is solvable in S'(n) 1(2)-alternating ATM space, then for any problem L' in C, there are constants c_1 and c_2 such that L' is solvable in S"(n) + $c_2$log S"(n) 1(2)-alternating ATM space, where $S"(n)=S'(c_1*Q(n))+2*\log(c_1*Q(n))+S(n)$.

In the remainder of this section we will be describing systems of probabilistic concurrent programs. For ease of expression, we describe each program via a flowchart. The building blocks of our flowcharts are

illustrated in Figures 4.1(a)-4.1(c). Here a circle represents a state of the flowchart. Two states, called the *initial* and *terminal* states, are distinguished from the rest. (In what follows, the reader should not confuse the state of the flowchart with the state of the program which it describes.) Initially, a *token* is placed on the initial flowchart state. The semantics of these flowchart programs is as follows. Each time the program is scheduled the token will move along the flowchart, according to the following rules, until the next state (i.e., circle) is reached. (The move is considered to be atomic.)

1. (Assignment block)
   See Figure 4.1(a). Assume that state s contains the token. Once the program is scheduled, with probability p_i ($1 \leq i \leq t$ and $0 \leq p_i \leq 1$) the sequence of instructions $l_1,...,l_m$ in A_i will be executed. The token then moves to s'_i.

2. (Waiting block)
   See Figure 4.1(b). Here P represents a predicate. Assume that s contains the token. Now, when the program is scheduled and P is "true" ("false"), the token moves to s' (s). Note that as long as P is "false" we have busy waiting.

3. (Conditional block)
   See Figure 4.1(c), where each dashed line is labelled with a predicate P_i ($1 \leq i \leq t$). The semantics of this block is that, if one and only one of the predicates, say P_i, is true, then A_i is executed before the token is moved to s'_i; otherwise, the token is moved back to s directly.

In what follows, we derive a lower bound for the fair NTP under fairness types 1-5.

Theorem 4.2: The NTP for $C(n,k)$ under fairness types 1-5 is $A\Sigma_2^{(k-1)*\log n}$-hard with respect to $((2+\epsilon)\log n, n^4\log^9 n)_{space}$-reducibility.

Proof. Let M be an arbitrary k*log n space bounded 1-alternating ATM. Assume that the set of tape symbols of M is {0,1}. Consider an arbitrary input string x. (Let n=|x|.) In what follows, we show how to construct a $P=(S,s_0,\delta,X)$ in $C(O(n^4\log^9 n),k+1)$ to simulate the computation of M on x in such a way that, M accepts x iff P will not terminate with probability 1 under fairness types 1-5.

The structure of P is shown in Figure 4.2, where circles and boxes are used to represent programs and shared variables, respectively. Before describing each entity of P in detail, we briefly address the general idea of the simulaton. The simulation is, in some sense, similar to the one used in proving the lower bound of Theorem 3.2. (Note, however, that a log n space bounded 1-alternating ATM was considered there.) Recall there that the proof involved constructing a system P'' consisting of a single program (say P) to simulate M on x. P's progress was divided into two phases - the existential phase and the universal phase. First P simulated, in a nondeterministic fashion, the existential moves of M on x until a universal configuration was reached. If the configuration of M, at this point, was c, P stored the configuration c in its local memory and entered the second phase. In the universal phase, P simulated universal moves of M (at each step P followed the computation of M by choosing just one of the possible successor

configurations - each successor configuration had equal probability of being chosen) until either a rejecting or an accepting configuration was reached. In the former case, P entered a terminal state and the simulation stopped; however, in the latter case the simulation began anew from the stored configuration c. By definition, M accepted x iff there existed some reachable configuration c such that from c, no rejecting configuration could be reached. Hence, in $G_{P''}$ (the graph of P'') the configuration c was in a type 1-5 blackhole. We then concluded that M accepted x iff P'' would not terminate with probability 1 under type 1-5 fairness. See Theorem 3.2. The reader should realize that the ability to restart the simulation at the stored configuration was crucial. Now, the same simulation for an O(k*log n) space bounded ATM would be more complicated. In general, the program would require $O(n^k)$ states, which will not work for our purposes here since we require the FPCP to be in $\mathcal{C}(O(n^c),O(k))$, for some fixed constant c.

In order to overcome this difficulty, the system P constructed here consists of k+1 programs, $\mathcal{A}$, L_i ($1 \leq i \leq k$), where each program is of size $O(n^4 * \log^9 n)$. First, M's k*log n bit work tape is partitioned into k blocks of length log n each. Program $\mathcal{A}$ will simulate the computation described above; while L_i, $1 \leq i \leq k$, acts as a storage vehicle for the i-th block of the work tape of M (and the i-th block of the work tape for the stored configuration). Now, $\mathcal{A}$ will not store the contents of M's work tape but will store the positions of both tape heads (and the positions of the tape heads for the stored configuration). In order to simulate a move, $\mathcal{A}$ must retrieve and update (from the appropriate L_i) the current bit being scanned on M's work tape. This is carried out by sending (and later receiving) "messages" through the chain $L_1,\ldots,L_k$. The program $\mathcal{A}$ ($L_1,\ldots,L_k$, respectively) communicates with L_1 ($L_2,\ldots,L_k$, $\mathcal{A}$, respectively) by using the shared memory denoted by W_0 ($W_1,\ldots,W_k$, respectively) as a communication media. Each message sent through the chain has (3 + log k + log log n + 1) bits and consists of the following information:

a_field (3 bits):

= 0: no-op mode,

= 1: retrieving mode,

= 2: updating mode,

= 3: phase_change mode,

= 4: reset mode,

= 5: terminating mode.

b_field (log k bits): block number.

c_field (log log n bits): displacement within a block.

d_field (1 bit): data.

Let W_i^h(h=a, b, c, d, ab, ac,..., etc.) denote the value of the h_field(s) of W_i, $1 \leq i \leq k$. (Let W_i be short for W_i^{abcd}.) The value of W_i^a, for values 1, 2, 3, 4 or 5 indicate whether $\mathcal{A}$ is (1) retrieving a bit of the work tape, (2) updating a bit of the work tape, (3) storing the work tape contents (of the stored configuration), (4) resetting the work tape contents, or (5) causing the system to terminate.

In what follows, we describe the programs L_i, $1 \leq i \leq k$, and $\mathcal{A}$ via flowcharts. The flowcharts are not actually part of the description of P but they readily illustrate the structure of the system and the ease with which the actual description of P can be produced by a transducer. We first describe the memory to which programs L_i, $1 \leq i \leq k$, and $\mathcal{A}$ have access. The memory, in each case, is referenced (in the flowcharts) by variable names in order to make the description of the flowcharts easier to understand. Let |M| denote the number of states in M.

Program L_i, $1 \leq i \leq k$

local variables:

Y (log n bits): used to store the i-th block of the current work tape.

Y' (log n bits): used to store the i-th block of the work tape when the simulation switches from its existential phase to its universal phase.

shared variables: W_{i-1} and W_i

Program $\mathcal{A}$

local variables:

ST (log |M| bits): used to represent the current state.

ST' (log |M| bits): used to store the state at the phase change.

TP (log k + log log n bits): used to represent the current work tape head position.

TP' (log k + log log n bits): used to store the work tape head position at the phase change.

IP (log n bits): used to indicate the input head position of the input tape.

IP' (log n bits): used to store the input head position at the phase change.

shared variables: W_0 and W_k.

Now, the program L_i, $1 \leq i \leq k$, is described by the flowchart in Figure 4.3. To construct $\mathcal{A}$, let U, E, A and R denote the set of universal, existential, accepting and rejecting states of M, respectively. A transition of M, l= $[(p,a,b) \rightarrow (q_1,b_1,d_1,d'_1),...,(q_t,b_t,d_t,d'_t)]$, implies that if M is in state p and a (b, respectively) is the current input (work tape, respectively) symbol then M can enter state q_1 (q_2,...,q_t,

respectively), change the tape symbol to b_1 (b_2,...,b_t, respectively), move its input and work head in directions d_1 (d_2,...,d_t, respectively) and d'_1 (d'_2,...,d'_t, respectively), respectively. Let $\#(l)$=t. Let π be a set of M's transitions where $l \in \pi$. (Note, that any two distinct transitions begin with a unique triple.) Figure 4.4 shows how the transition l can be simulated. Note that the predicate on each dashed line matches the current state, input symbol and tape symbol with the corresponding transition. The predicates on the other dashed lines correspond to other transitions. Let B_π denote such a flowchart segment where all the transitions in π are represented. Now, we define $\pi_u=\{u_1,...,u_f\}$ ($\pi_e=\{e_1,...,e_g\}$) to be the set of transitions defined on universal (existential) states of M, respectively. The program $\mathcal{A}$ is shown in Figure 4.5.

Let $P=(S,s_0,\delta,X)$ be as described above where each X_i (and $X_{\mathcal{A}}$) = {the unique program state where the flowchart token is on Θ and all variables are set to 0}, $1 \leq i \leq k$, and s_0 is the system state where all variables are set to zero except for ST (which is set to the initial state of M) and each respective flowchart token resides in its respective initial state. Now, consider the size of P. For each L_i, $1 \leq i \leq k$, it is easy to see that S_i (the local and shared memory of program L_i) contains some constant plus 2*(log n)+2*(log log n) bits. Hence, the number of possible transitions of L_i is $O(n^4*\log^4 n)$. $S_{\mathcal{A}}$ contains some constant plus 2*(log n)+4*(log log n) bits. Thus, the number of possible transitions of $\mathcal{A}$ is $O(n^4*\log^8 n)$. Since each transition can be represented in O(log n) bits, the length of P is $O(n^4*\log^9 n)$.

Now, we describe the Turing machine transducer T that when given x constructs P. Clearly the only hard part is the construction of δ. To construct δ_i (for L_i), $1 \leq i \leq k$, the work tape of T is of the form #i#j#, where i, of length 2*(log n + log log n) bits, is to keep track of the variables, and j, of length log log n bits, is a pointer for scanning the work tape. (Three more #'s are used to separate variables in the i part.) Similarly, to construct $\delta_{\mathcal{A}}$ for $\mathcal{A}$, T's work tape requires no more than 2*(log n)+4*(log log n) bits with 7 #'s. Therefore, by Lemma 4.1, the construction can be carried out in deterministic space $(2+\epsilon)$log n for any $\epsilon > 0$.

Now the system P behaves as described earlier. $\mathcal{A}$, using the L_i's to store the contents of the work tape, moves through successive configurations of M on x in much the same fashion as was described in Theorem 3.2. $\mathcal{A}$ terminates whenever it comes across a rejecting configuration. If $\mathcal{A}$ terminates it subsequently causes the termination of each L_i, $1 \leq i \leq k$. $\mathcal{A}$ will process forever (with nonzero probability) if it finds a universal configuration, following an existential configuration, for which each emanating path results in an accepting state. Now the reader should be able to ascertain the following two facts concerning the operation of $\mathcal{A}$:

(1) Whenever $\mathcal{A}$ is at a conditional block in its flowchart one and only one predicate will be true.

(2) Whenever $\mathcal{A}$ is at a waiting block either the predicate is true or $W_k^a=0$.

For a program P, we call a transition a no-op transition if it doesn't change the configuration of P's

accessible memory (this implicitly includes, of course, the position of the token in P's flowchart). Now L_i ($\mathcal{A}$), $1 \leq i \leq k$, can make no-op transitions iff W^a_{i-1} (W^a_k) is zero. By construction all but exactly one of the W^a_i's, $1 \leq i \leq k$, will be zero at any given time. Hence, at any given time, exactly one of the programs will be able to make a transition that is not a no-op. Now as long as $\mathcal{A}$ moves through configurations each L_i, $1 \leq i \leq k$, must subsequently be scheduled for a transition that is not a no-op. Also note that the L_i's are essentially deterministic. Since each L_i, $1 \leq i \leq k$, must eventually execute a transition, $\mathcal{A}$ must continue the simulation or terminate. As was the case in Theorem 3.2, the probabilities force the entire subtree of the stored universal configuration to be explored. Hence either $\mathcal{A}$ terminates (with probability 1) or the system enters a type 1-5 blackhole (with nonzero probability).

Thus, we have shown that for any $k>0$ and $\epsilon>0$, and any L in $A\Sigma_2^{k*\log n}$, there is a constant c such that L is $((2+\epsilon)\log n, c*n^4\log^9 n)_{space}$-reducible to the fair NTP for $C(n,k)$ under fairness types 1-5 □

Corollary 4.1: The NTP for $C(n,k)$ under fairness types 1-5 requires $(k-11)/4*\log n$ 1-alternating ATM space, for $k>11$.

Proof. Assume that the problem can be solved in $(k-11-\epsilon)/4*\log n$ 1-alternating ATM space for some $k>11$ and $\epsilon>0$. From Theorem 4.2 we know that the NTP is $A\Sigma_2^{(k-1)*\log n}$-hard with respect to $((2+\epsilon_1)\log n, n^4\log^9 n)_{space}$-reducibility. Thus, according to Lemma 4.2 we have that any language in $A\Sigma_2^{(k-1)*\log n}$ can be solved in $S'(n)+c_2*\log S'(n)$ 1-alternating ATM space, where $S'(n) = ((k-11-\epsilon)/4)\log(c_1*n^4*\log^9 n) + 2*\log(c_1*n^4*\log^9 n) + (2+\epsilon_1)\log n$, for some c_1 and c_2. This amount is in $(k-1-\epsilon_3)\log n$ 1-alternating ATM space for some $\epsilon_3>0$. Hence, we have a contradiction. □

Corollary 4.2: The NTP for $C(n,k)$ under fairness types 1-5 requires PSPACE if k is a problem parameter.

Define $ATIME_2(n^k)$ to be the set of languages accepted by $O(n^k)$ time bounded ATM's that use at most one alternation during the course of a computation. We have that $DTIME(n^k) \subseteq NTIME(n^k) \subseteq ATIME_2(n^k)$ and $DSPACE(k*\log n) \subseteq NSPACE(k*\log n) \subseteq A\Sigma_2^{k*\log n}$. In [9], the following conjecture was used:

Conjecture: For any $\epsilon>0$, $NSPACE(k*\log n) \not\subseteq DTIME(n^{k-\epsilon})$.

Consequently, $A\Sigma_2^{k*\log n} \not\subseteq DTIME(n^{k-\epsilon})$, for any $\epsilon>0$. Hence, we have:

Corollary 4.3: Under the above conjecture, the NTP for $C(n,k)$ under fairness types 1-5 requires $O(n^{(k-1)/4})$ deterministic time, for $k>13$.

Proof. Assume that the NTP for $C(n,k)$ can be solved in $n^{(k-1)/4-\epsilon}$ deterministic time, for some $\epsilon>0$. According to Lemma 4.2, we have that any language in $A\Sigma_2^{(k-1)*\log n}$ can be solved in $(c_1*n^4*\log^9 n)^{(k-1)/4-\epsilon} + c_2*n*((2+\epsilon_1)*\log n)*2^{(2+\epsilon_1)*\log n}$ deterministic time, for some c_1 and c_2. It can be easily seen that, for $k>13$ the above term is in $O(n^{k-1-\epsilon+\epsilon_2})$, for any ϵ_2. Choose $\epsilon_2<\epsilon$, we have that

$A\Sigma_2^{(k-1)*\log n}$ can be solved in $O(n^{k-1-\epsilon_3})$, for some $\epsilon_3>0$, which, obviously, contradicts the conjecture. □

As far as type 5 fairness is concerned, every program has to be scheduled infinitely often (this is not required for type 4 fairness). Therefore, a bigger lower bound is expected for fairness types 1-3, 5. As a matter of fact, we are able to show that:

Theorem 4.3: The NTP for $C(n,k)$ under fairness type 1-3, 5 is $A\Sigma_3^{(k-2)*\log n}$-hard with respect to $((3+\epsilon)\log n, n^6\log^{11}n)_{space}$-reducibility.

Proof. Let M be an arbitrary k*log n space bounded 2-alternating ATM. Assume that the set of tape symbols of M is {0,1}. Consider an arbitrary input string x. (Let n=|x|.) In what follows, we show how to construct a $P=(S,s_0,\delta,X)$ in $C(O(n^6\log^{11}n),k+1)$ to simulate the computation of M on x in such a way that, M accepts x iff P will not terminate with probability 1 under fairness types 1-3, 5.

The basic idea of the simulation is very similar to the one in the previous theorem, where M's k*log n bit work tape was partitioned into k blocks. The essential difference is that the simulation follows the one in the proof of Theorem 3.4 instead of Theorem 3.2. Hence, here we only describe the crucial points that differ from those of Theorem 4.2. Figure 4.6 presents the overall structure of P. Programs A and B here play the roles of A and B in Theorem 3.4; except that the L_i's store the contents of M's work tape as was the case in Theorem 4.2. Programs A, L_i, $1\leq i\leq k$, play essentially the same roles they did in Theorem 4.2; except that the L_i's must store three (instead of two) log n bit pieces of M's work tape, since the simulation described in Theorem 3.4 has two (instead of one) stored configurations at which the simulation can be restarted. Now the memory shared by A and B is only the single bit d. Now B's only transition is to change d from 1 to 0; however B will only be enabled (by A setting d to 1) when A comes across an accepting configuration of M. Each of the other programs will always be enabled. Hence, blackholes of type 1-3, 5 are essentially the same. The remaining details of the simulation are left to the reader.

Let P be (S,s_0,δ,X) be as described above where each X_i (and $X_{\not\theta}$) = {the unique program state where the flowchart token is on Θ and all variables are set to 0}, $1\leq i\leq k$, and s_0 is the system state where all variables are set to zero except for ST (which is set to the initial state of M) and each respective flowchart token resides in its respective initial state. Now, consider the size of P. For each L_i, $1\leq i\leq k$, it is easy to see that S_i (the local and shared memory of program L_i) contains some constant plus 3*(log n)+2*(log log n) bits. Hence, the number of possible transitions of L_i is $O(n^6*\log^4 n)$. S_A contains some constant plus 3*(log n)+5*(log log n) bits. Thus, the number of possible transitions of A is $O(n^6*\log^{10}n)$. The size of B is a constant. Since each transition can be represented in O(log n) bits, the length of P is $O(n^6*\log^{11}n)$.

Now, we describe the Turing machine transducer T that when given x constructs P. Again the only hard part is to construct δ. To construct δ_i (for L_i), $1\leq i\leq k$, the work tape of T is of the form #i#j#,

where i, of length 3*log n + 2*(log log n) bits, is to keep track of the variables, and j, of length log log n bits, is a pointer for scanning the work tape. (Four more #'s are used to separate variables in the i part.) Similarly, to construct δ_A for A, T's work tape requires no more than 3*(log n)+5*(log log n) bits with 9 #'s. Finally, δ_B can be constructed in constant space. Therefore, by Lemma 4.1, the construction can be carried out in deterministic space $(3+\epsilon)\log n$ for any $\epsilon>0$.

□

Corollary 4.4: The NTP for $C(n,k)$ under fairness types 1-3, 5 requires (k-17)/6*log n 2-alternating ATM space, for $k>17$.

Proof. Assume that the problem can be solved in $(k-17-\epsilon)/6*\log n$ 2-alternating ATM space for some $k>17$ and $\epsilon>0$. From Theorem 4.3 we know that the NTP is $A\Sigma_3^{(k-3)*\log n}$-hard with respect to $((3+\epsilon_1)\log n, n^6\log^{11}n)_{space}$-reducibility. Thus, according to Lemma 4.2 we have that any language in $A\Sigma_3^{(k-2)*\log n}$ can be solved in $S'(n)+c_2*\log S'(n)$ 2-alternating ATM space, where $S'(n) =((k-17-\epsilon)/6)\log(c_1*n^6*\log^{11}n) + 2*\log(c_1*n^6*\log^{11}n) + (3+\epsilon_1)\log n$, for some c_1 and c_2. This amount is in $(k-2-\epsilon_3)\log n$ 2-alternating ATM space for some $\epsilon_3>0$. Hence, we have a contradiction. □

In what follows, we show that, given an FPCP in $C(n,k)$, the NTP (TP) under fairness types 1-3 requires time $\Omega(n^{(k-6)/96})$. To show this, we reduce the two-person pebble game problem (with k pebbles), which is known to require $\Omega(n^{O(k)})$ deterministic time [1], to the fair NTP.

A *two-person pebble game* G is a 4-tuple (N,R,S,T), where

N is a finite set of nodes,

R ($\subseteq N\times N\times N$) is the set of rules,

S ($\subseteq N$) is the set of initial nodes,

T ($\in N$) is the *terminal node.*

G is said to be an (n,k)-pebble game iff |N|=n and |S|=k. Initially, pebbles are placed on initial nodes, i.e., all nodes in S. The playing (pebble-moving) rule is that, whenever $(x,y,z)\in R$ and nodes x and y, but not z, contain pebbles, a pebble can be moved from x to z. Two players, say A and B, take turns moving the pebbles. Each player can make at most one move during his turn. A winning position (or win) for a player is when he either moves a pebble to T, or he forces his opponent to be unable to move. At such a time the game is over. The *pebble game problem* is to, given a pebble game, determine whether the first player has a winning strategy, i.e., whether the first player can always manage to win regardless of his opponent's moves. As one can easily see, the pebble game problem possesses the characteristics similar to that of the computation of an ATM. More precisely, the moves of the first player, when trying to obtain

a winning position (regardless of how his opponent moves), correspond to the existential branches in an ATM; while the moves of the second player, when trying to prevent the first player from winning, correspond to the universal branches in an ATM. Because of this alternating behavior, the pebble game problem requires exponential execution time. In fact, the following result concerning the complexity of pebble game problem was shown in [1]:

The (n,k)-pebble game problem requires $\Omega(n^{(k-1)/4-\epsilon})$ deterministic time for any $\epsilon>0$ and $k>5$ on multitape Turing machines.

Using this result, we are able to show:

Theorem 4.4: The (n,k)-pebble game problem is $(O(n^{12}*\log^2 n), n^{12}\log n)_{time}$-reducible to NTP (TP) for $FPCP_{2*k+4}$ for fairness types 1-3.

Proof. Let G=(N,R,S,T) be an (n,k)-pebble game. Without loss of generality, we assume that the first player cannot win in a single move. We show how to construct an FPCP $P=(S,s_0,\delta,X)$ in $C(O(n^{12}\log n),2*k+4)$ to simulate G in such a way that, P will not terminate with probability 1 for fairness types 1-3 iff G has a winning strategy for the first player. Without loss of generality, we assume that each node of G is labelled with a unique number i, $1\leq i\leq n$. In this way, a node can be referenced using a (log n)-bit array. Similarly, each pebble is assigned with a number ranging from 1 to k. For brevity, let d_i denote the node that contains the i-th pebble. Let d_i^0 indicate the initial position of the i-th ($1\leq i\leq k$) pebble before the game starts.

The overall structure of the FPCP we constructed is depicted in Figure 4.7, where circles and boxes denote programs and shared variables, respectively. Before describing each entity in detail, we first present the general idea of how the simulation works. The idea will be for the game to be played over and over again as long as A keeps winning. If B wins the system will terminate (providing the schedule is fair). Basically, programs A and B are used to simulate the two players in the pebble game. A plays for the first player. Each program (A and B) contains in its finite-state control the description of G. However, in order to keep the size of the program small, the dynamic information (i.e., the current location of each pebble) is not kept in A (or B). Instead, k programs $(L_1,...,L_k)$ are used to store this information. Each L_i, $1\leq i\leq k$, contains log n bits of local memory to remember the current position of the i-th pebble. The simulation then proceeds as follows.

For A to move it must play a rule $(x,y,z)\in R$, such that nodes x and y have pebbles but z does not. It uses $G_1,...,G_k$ to (nondeterministically) choose the pebbles which are on nodes x and y and then $L_1,...,L_k$ to supply the actual node values. It uses C_1 and C_2 to choose a rule $(x,y,z)\in R$ with the correct first two coordinates and $L_1,...,L_k$ to verify that z currently does not contain a pebble. (The particular nondeterministic choices will be forced via the schedule.) If A finds that the chosen rule cannot be played

the process of selecting a rule begins anew; otherwise A plays the rule. If as a result A "wins", A uses $L_1,...,L_k$ to restore the original position of each pebble so that the game can begin anew; otherwise A uses $L_1,...,L_k$ to update the position of the moved pebble and then activates B. (A now remains dormant until B reactivates it.) B chooses what move to play in a probabilistic fashion (thus ensuring in a fair schedule that all such choices are played infinitely often). It obtains and updates the information stored in $L_1,...,L_k$ in much the same fashion as A does. If it chooses a rule it cannot play, it returns to its initial state and selects a new rule. If playing a rule places a pebble on T (i.e., B wins), B terminates and in turn causes all other programs to terminate (providing the schedule is fair); otherwise B merely updates the information in $L_1,...,L_k$ and reactivates A. Until A again activates B the only move available to B is that of termination. Now at each instant in which A is active all programs will be enabled (although perhaps only for no-ops). A at times may be disabled but as long as the schedule is fair it will eventually become enabled. Thus, for such a system fairness types 1-3 are equivalent.

Before describing the simulation in detail, we first look at the type of messages contained in each W_i. Each W_i, $1 \leq i \leq k$, is of length 3+log k+log n bits which contains the following types of information.

a_field (3 bits):

= 0: no-op mode,

= 1: retrieving mode,

= 2: updating mode,

= 3: reset mode,

= 4: matching mode,

= 5: matching_fail,

= 6: terminating mode,

b_field (log k bits): used to represent an index of a pebble,

c_field (log n bits): used to denote the position of a pebble.

In the following, we first describe the variables used in each program:

Program L_i, $1 \leq i \leq k$

local variables: Y (log n bits) - used to store the position of the i-th pebble.

shared variables: W_{i-1} and W_i.

Program G_i, $1 \leq i \leq k$

local variables: none.

shared variables: W (2 bits) and W_0.

Program C_1 (C_2)

local variables: none.

shared variables: U (2 bits) and V (1 bit).

Program A

local variables:

I and J (log k bits each) - used to store the indices of two pebbles,

d_I, d_J and d_Z (log n bits each) - used to store the positions of pebbles,

CT (log n bits) - a counter.

shared variables: U, V, W, E, W_0, and W_k.

Program B

local variables:

I (log k bits) - used to store the index of a pebble,

d_I, d_J and d_Z (log n bits each) - used to store the positions of pebbles,

shared variables: E, W_0 and W_k.

The detailed description of each program is shown in Figures 4.8-4.12. At this point the intent reader should study the flowcharts throughly. Note that the finite state control of A (and B) contains a table recording the static structure of the pebble game. Basically this table is assumed to be organized in such a way that, given two positions x and y, the set of nodes z for which $(x,y,z)\in R$, can be obtained in a sequential manner. We use $(x,y)_i$ to denote the i-th such z. Furthermore, the number of such z's (at most n) (denoted by #(x,y)) is also kept in the table.

Now, we are able to argue that G has a win for the first player iff P will not terminate with probability 1 under fairness types 1-3. To see this, first note that the only terminal state in B's flowchart is Θ. Moreover, B is able to enter Θ iff either

1. A is active, or

2. B moves a pebble to T.

Now whenever B terminates it forces every other program to follow suit (in a fair schedule). Furthermore, if at some point the first player cannot move then A will never again activate B (or restart the game). Since the only subsequent move available to B causes termination the system must terminate

if the schedule is fair. The *if* part is, therefore, trivial. To prove the *only if* part, we have to show that if the first player can manage to win no matter how the second player reacts to his moves, then we are able to find a fair schedule σ such that P will not terminate with probability 1 under σ. The strategy for constructing σ is the following. The first player's choices are determined (in some intelligent way) by selecting the "correct" G_i's ($1 \leq i \leq k$) and C_j's (j=1 or 2) each time A is activated to proceed. Note that letting the schedule perform this selection will ensure that the first player can always "manage to win", providing the pebble game problem has a solution. Now the game when A "wins" is replayed over and over again. In each successive iteration the schedule is employed to force A to play within the strategy. On the other hand, in order to ensure that "no matter how" the second player responds the first player always wins, probabilities are used in program B each time when several moves are available. In this way, in order to form a fair schedule B has to, in some sense, "try" every possible move eventually. The *only if* part is therefore proved.

It is worth mentioning here that the above schedule is not state-fair. This is because, only one G_i will be scheduled to proceed at a time even though all G_i's are enabled; and the correct schedule may not require all G_i's to be scheduled. In this case, the schedule will not be state-fair.

Let $P=(S,s_0,\delta,X)$ be as described above. Now the single termination state for each program has the flowchart token on Θ and all variables set to zero. s_0 is the system state in which the value of all variables is zero except that Y in L_i is set to d_i^0 and each respective flowchart token resides in its respective initial state. Now consider the size of P. For each L_i, $1 \leq i \leq k$, it is easy to see that S_i (the local and shared memory of program L_i) contains some constant plus 3*(log n) bits. Hence, the number of possible transitions of L_i is $O(n^6)$. The size of G_i, $1 \leq i \leq k$, is $O(n^2)$. S_A contains some constant plus 6*(log n) bits. Thus, the size of A is $O(n^{12})$. Similarly, the size of B is $O(n^{10})$. Finally the sizes of C_1 and C_2 are constants. Since each transition can be represented in O(log n) bits, the length of P is $O(n^{12}*\log n)$. Furthermore, the construction can be carried out in $O(n^{12}*\log^2 n)$ deterministic time. The theorem then follows. □

Corollary 4.5: The NTP for FPCP in $C(n,k)$, under fairness types 1-3, requires $\Omega(n^{(k-6)/96 - \epsilon})$ deterministic time, for any $\epsilon>0$ and $k>1254$, on multitape Turing machines.

Proof. First, note that for $k>1254$, there exists an $\epsilon_1>0$ such that $\lim_{n\to\infty} (n^{12}*\log^2 n)/(n^{(k-6)/96 - \epsilon_1}) = 0$. Suppose that the NTP can be solved in $n^{(k-6)/96 - \epsilon_2}$, for some $\epsilon_2>0$. Since the (n,k)-pebble game problem is $(O(n^{12}*\log^2 n), O(n^{12}\log n))_{time}$ reducible to the NTP for $FPCP_{2*k+4}$, choose $\epsilon=\min\{\epsilon_1,\epsilon_2\}$. We immediately have that the pebble game problem can be solved in $O(n^{(k-1)/4 - \epsilon})$ deterministic time, which is a contradiction. □

Corollary 4.6: The NTP (TP) for FPCP in $C(n,k)$, under fairness types 1-3, requires exponential time when the degree of concurrency, k, is a problem parameter.

References

[1] Adachi, A. and Iwata, S., Some combinatorial game problems require $\Omega(n^k)$ time, *JACM*, Vol. 31, No. 2, April 1984, pp. 361-376.

[2] Chandra, A., Kozen, D. and Stockmeyer, L., Alternation, *JACM*, Vol. 28, No. 1, January 1981, pp. 114-133.

[3] Garey, M. and Johnson, D., "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H.Freeman and Company, San Francisco, 1979.

[4] Hart, S., Sharir, M. and Pnueli, A., Termination of probabilistic concurrent programs, *ACM Trans. on Programming Languages and Systems*, Vol. 5, No. 3, July 1983, pp. 356-380.

[5] Hart, S., Sharir, M. and Pnueli, A., Verification of probabilistic systems, *SIAM J. of Computing*, 13, 1984, pp. 292-314.

[6] Hopcroft, J. and Ullman, J., "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley, Reading, Mass., 1979.

[7] Iwata, S. and Kasai, T., Problem requiring k*log n deterministic space, *Proc. 15th Southeastern Conf. on Combinatorics, Graph Theory and Computing*, Baton Rouge, 1984.

[8] Jones, N. and Laaser, W., Complete problems for deterministic polynomial time, *Theoretical Computer Science*, 3, 1977, pp. 105-117.

[9] Kasai, T. and Iwata, S., Gradually intractable problems and nondeterministic log-space lower bounds, to appear in *Mathematical Systems Theory.*

[10] Kemeny, J., Snell, J. and Knapp, A., "Denumerable Markov Chains", D. van Nostrad Company, 1966.

[11] Ladner, R., The complexity of problems in systems of communicating sequential processes, *J. of Computer and System Sciences*, 21, 1980, pp. 179-194.

[12] Lehmann, D., Pnueli, A and Stavi, J., Impartiality, justice and fairness: The ethics of concurrent termination, *Automata, Languages and Programming*, LNCS 115, 1981, pp. 264-277.

[13] Lehmann, D. and Rabin, M., On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem, *Proc. of the 10th ACM Symp. on Principles of Programming Languages*, 1981, pp. 133-138.

[14] Lichtenstein, O. and Pnueli, A., Checking that finite state concurrent programs satisfy their linear specification, *Proc. of the 12th Annual ACM Symp. on Principles of Programming Languages*, 1985, pp. 97-107.

[15] Pnueli, A., On the extremely fair treatment of probabilistic algorithms, *Proc. of the 15th Annual ACM Symp. on Theory of Computing*, 1983, pp. 278-290.

[16] Rabin, M., N-process synchronization by $4*\log_2$N-valued shared variable, *Proc. of the 21st Annual Symp. on Foundations of Computer Science*, 1980, pp. 407-410.

[17] Rabin, M., The choice coordination problem, *Acta Informatica*, 17, 1982, pp. 121-134.

[18] Rosier, L. and Yen, H., A multiparameter analysis of the boundedness problem for vector addition systems, *Fundamentals of Computation Theory*, LNCS 199, 1985, pp. 361-370.

[19] Rosier, L. and Yen, H., Logspace hierarchies, polynomial time and the complexity of fairness problems concerning ω-machines, Univ. of Texas at Austin, Dept. of Computer Sciences, Tech. Report No. 85-08, May 1985.

[20] Ruzzo, W., Simon, J. and Tompa, M., Space-bounded hierarchies and probabilistic computations, *J. of Computer and System Sciences*, 28, 1984, pp. 216-230.

[21] Vardi, M., Automatic verification of probabilistic concurrent finite-state programs, *Proc. of the 26th Annual Symposium on Foundations of Computer Science*, 1985.

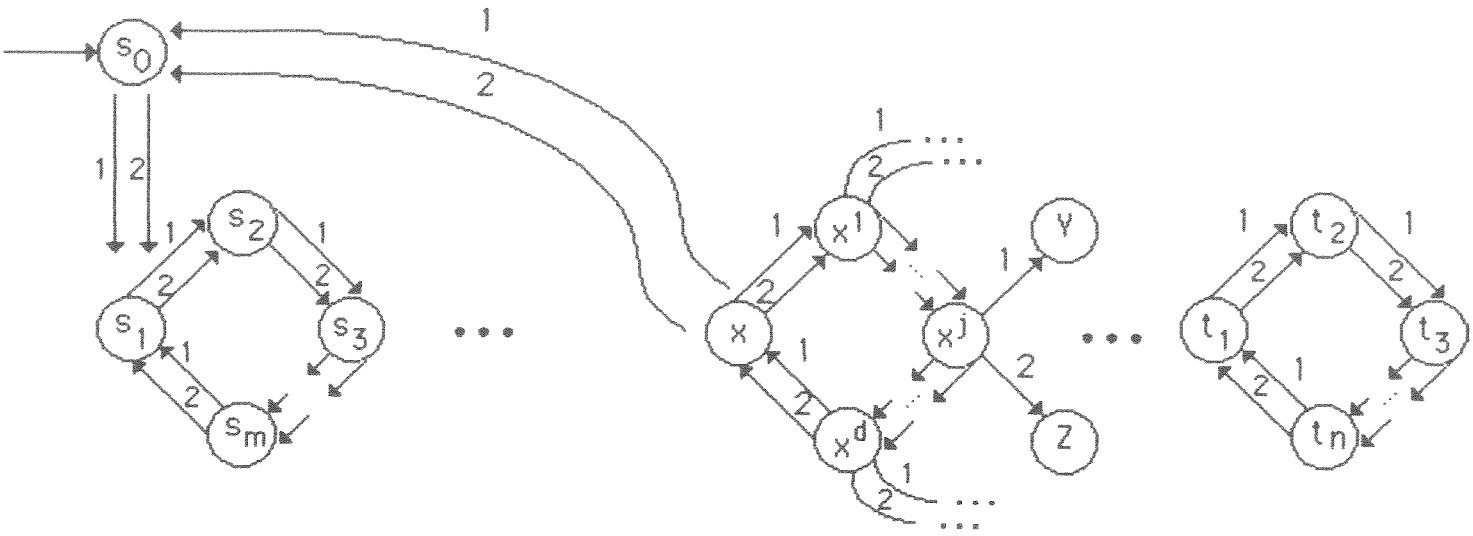

Figure 3.1 The graph G_p constructed in Theorem 3.1.

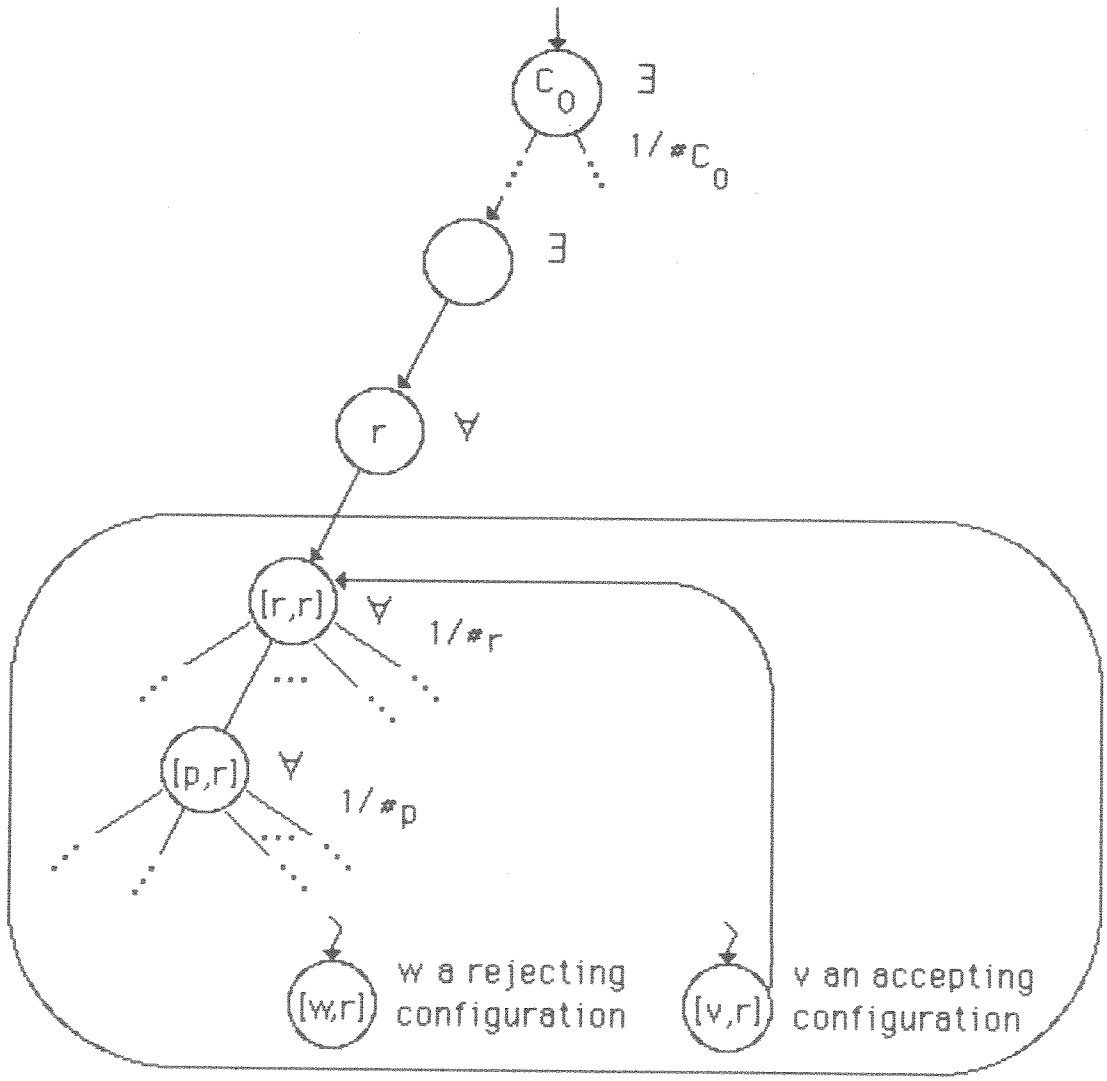

Figure 3.2 The single program system P constructed in Theorem 3.2.

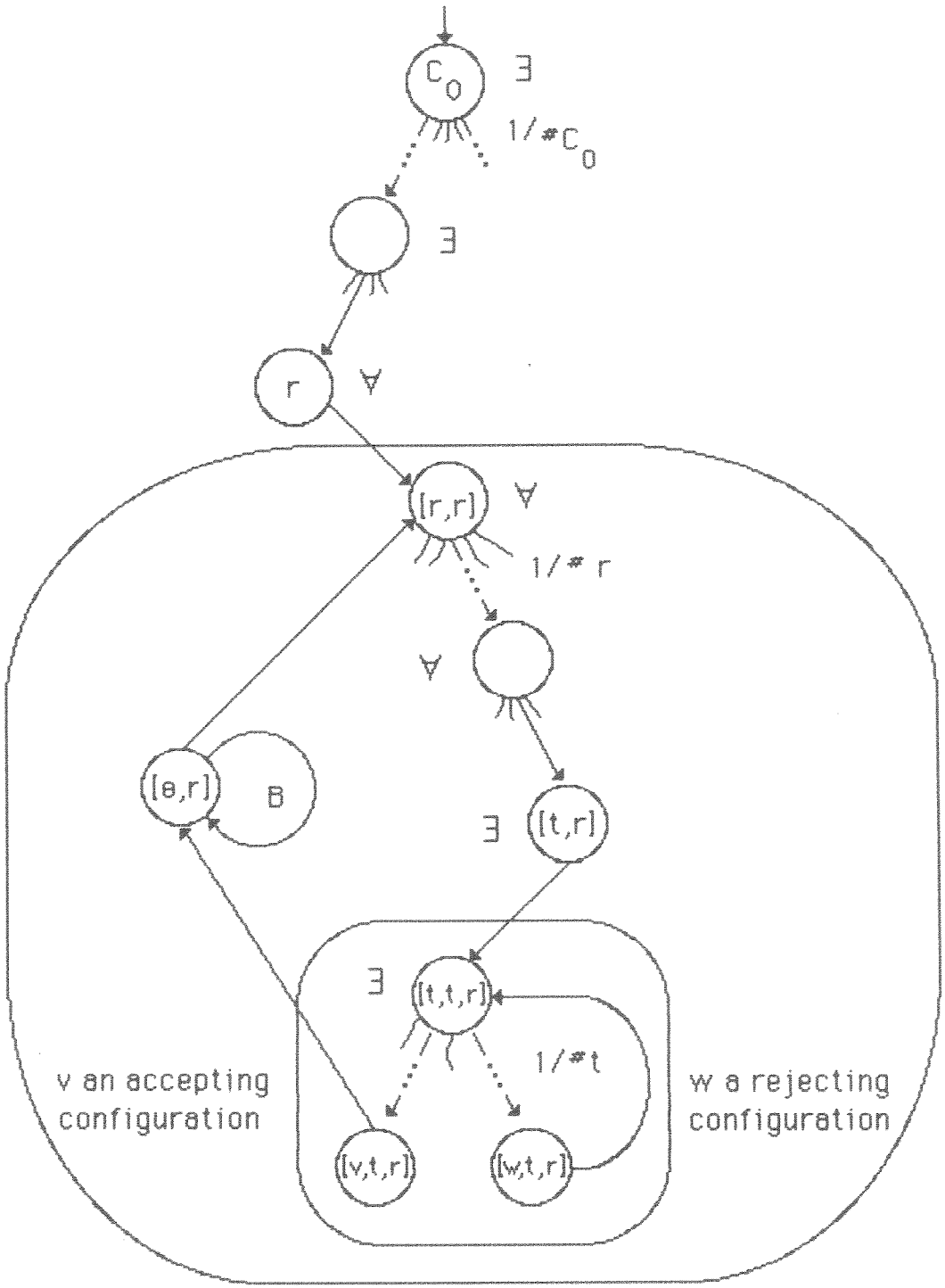

Figure 3.3 The two program system P constructed in Theorem 3.4. (All unlabelled edges correspond to transitions of A; the edges labelled B correspond to transitions of B.)

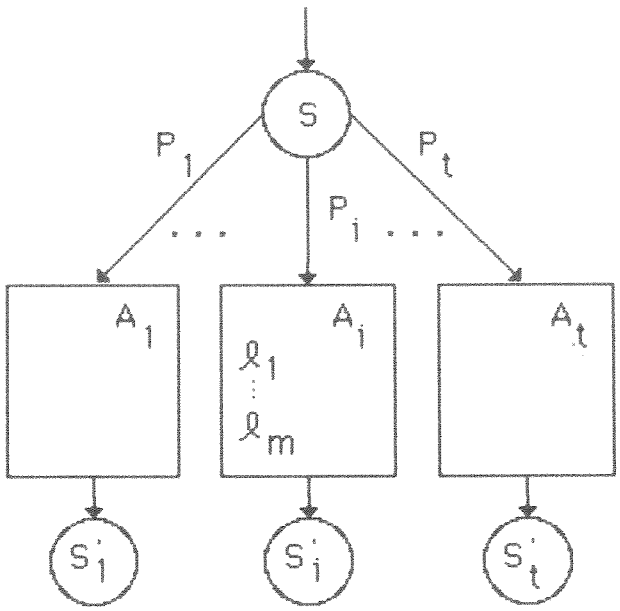

Figure 4.1 (a) An assignment block.

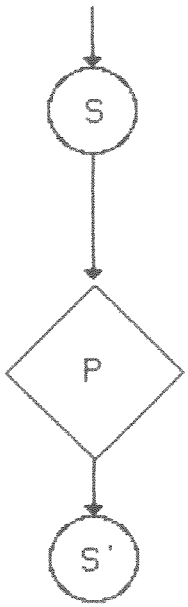

Figure 4.1 (b)
A waiting block.

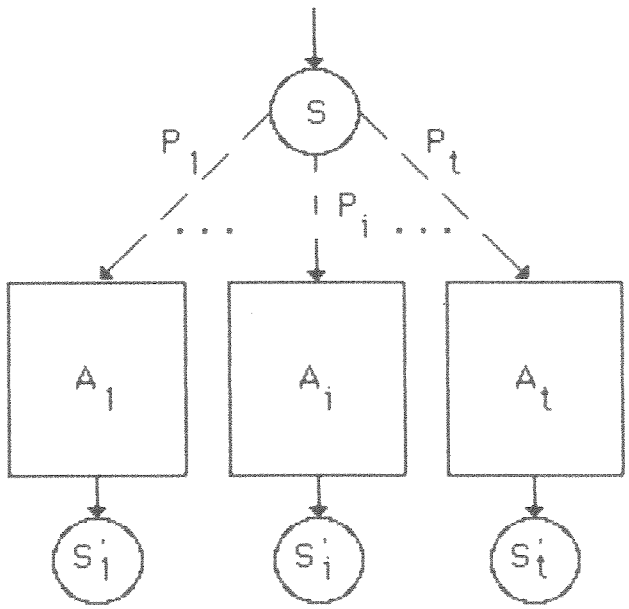

Figure 4.1 (c)
A conditional block.

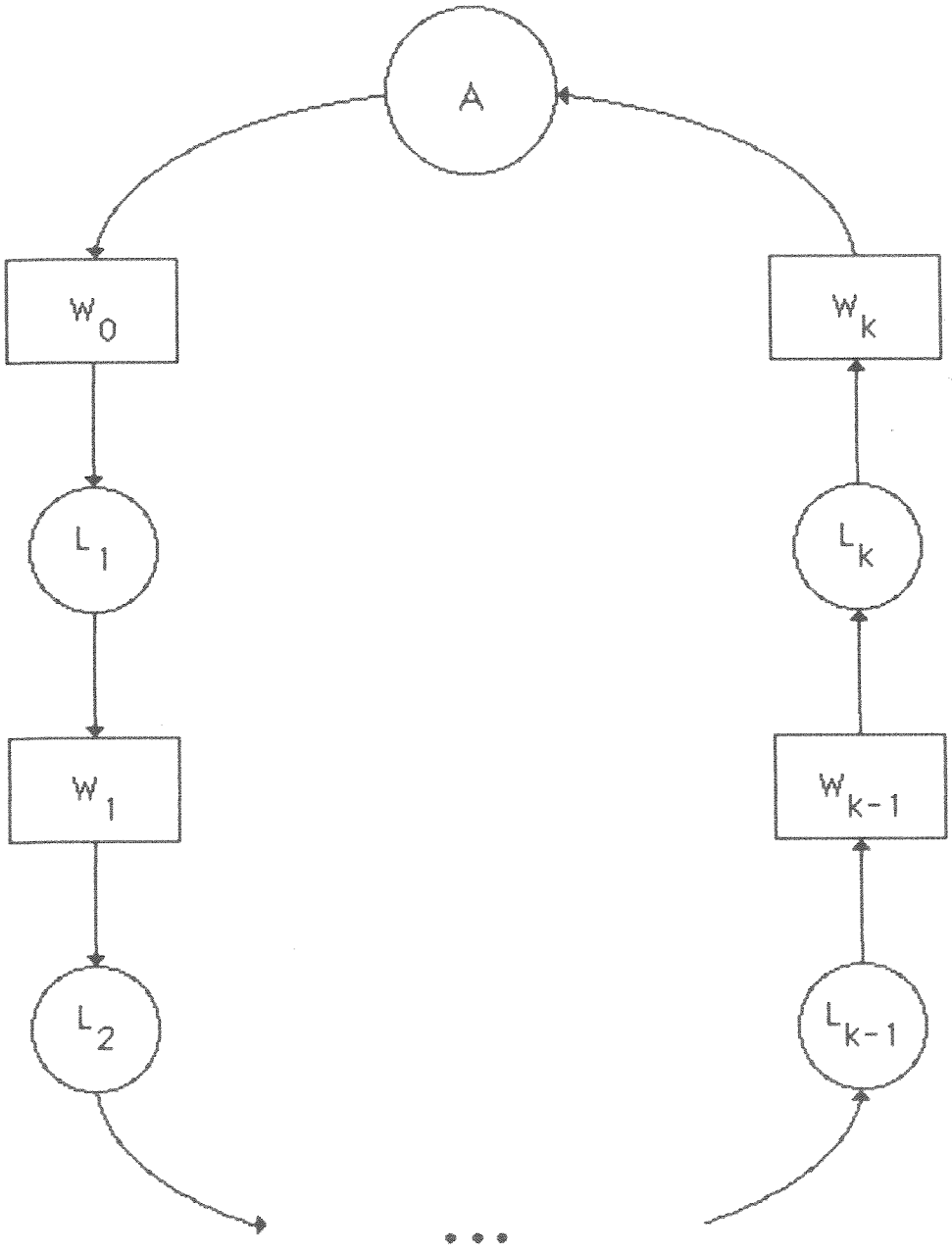

Figure 4.2 The system P constructed in Theorem 4.2.

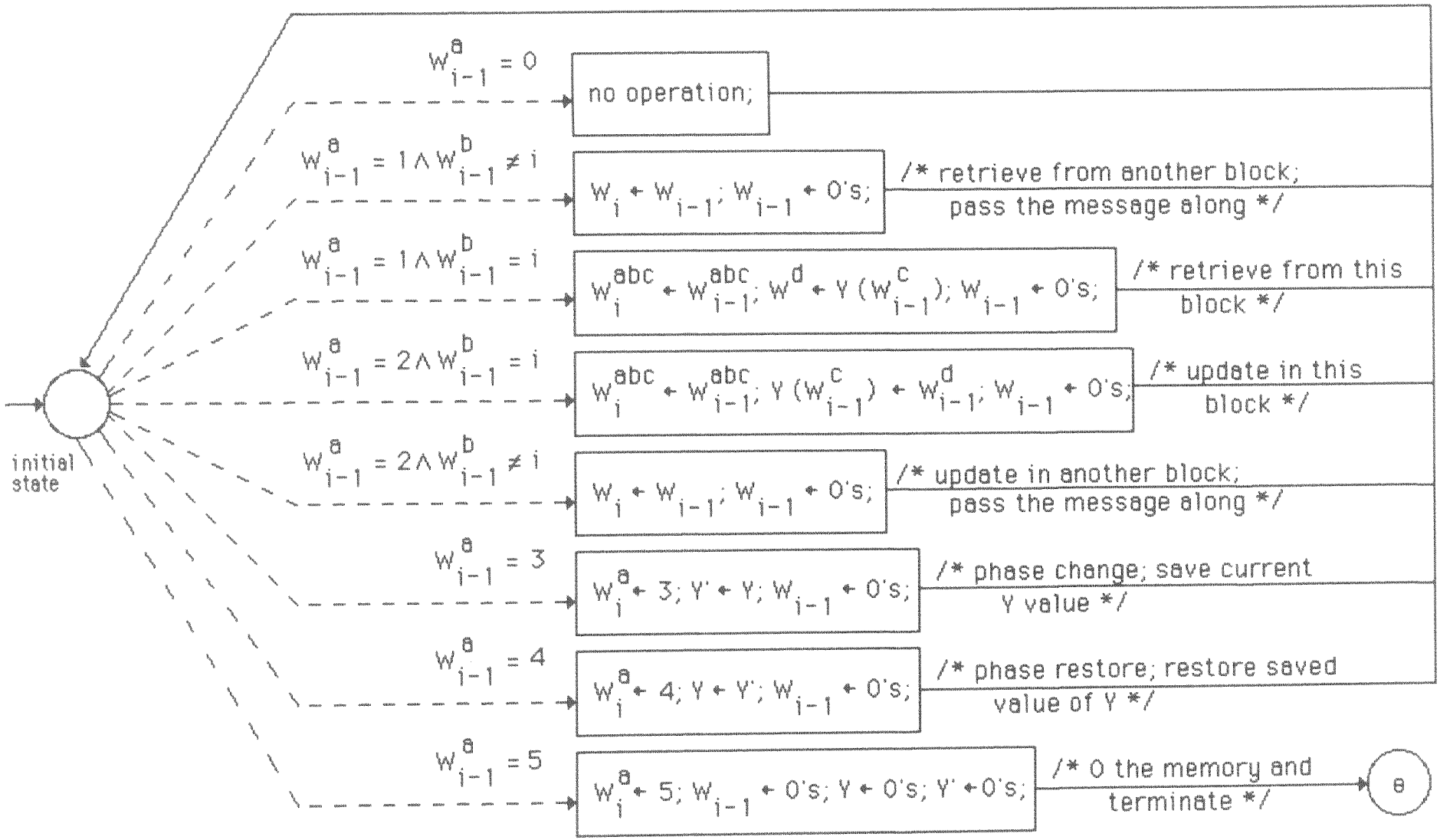

Figure 4.3 The flowchart for program L_i. (Y(j) denotes the j^{th} bit of Y.)

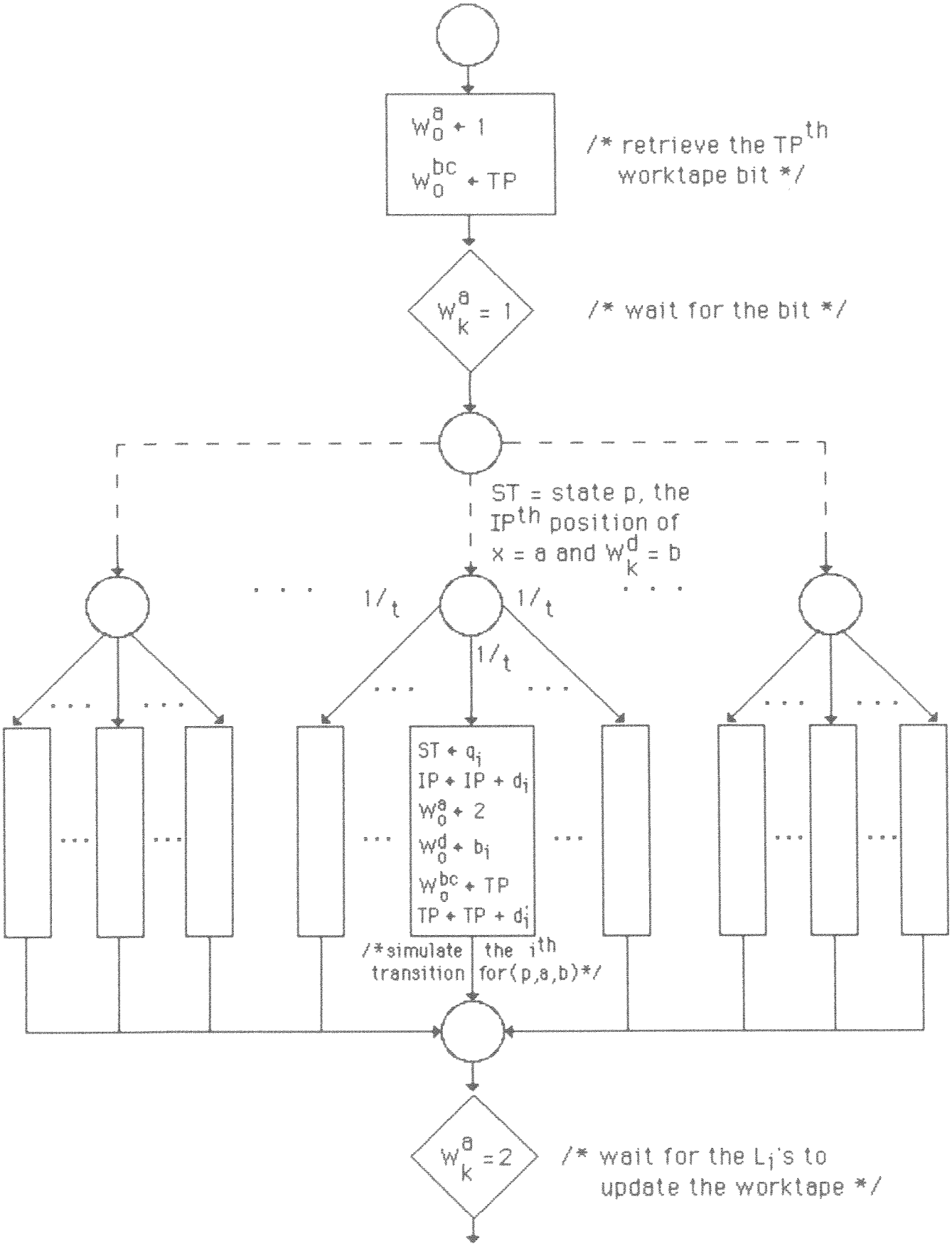

Figure 4.4 A flowchart segment to simulate the transitions of $\pi = \{[\ldots],\ldots,[(p,a,b) \rightarrow (q_1,b_1,d_1,d_1'),\ldots,(q_t,b_t,d_t,d_t')],\ldots,[\ldots]\}$

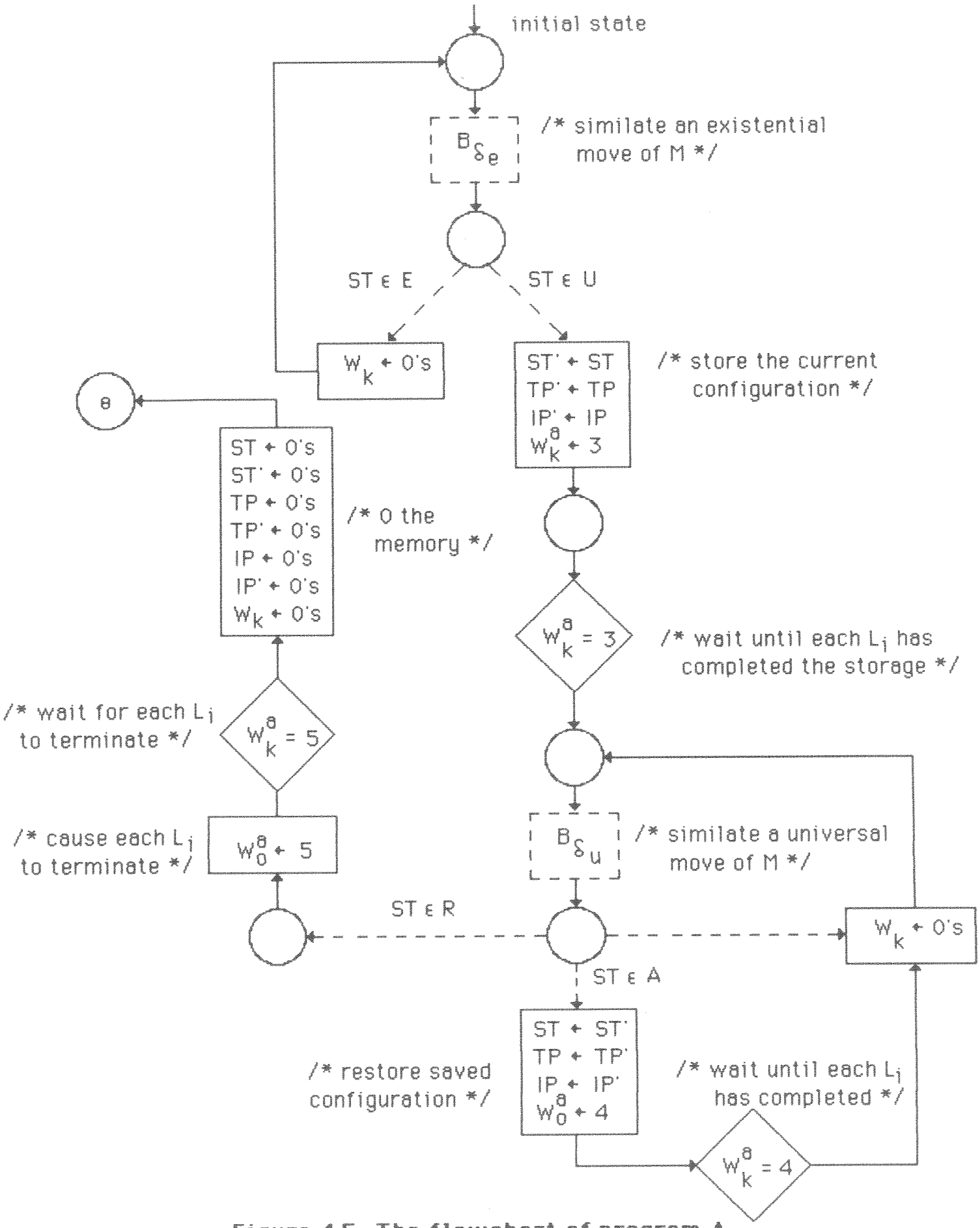

Figure 4.5 The flowchart of program A.

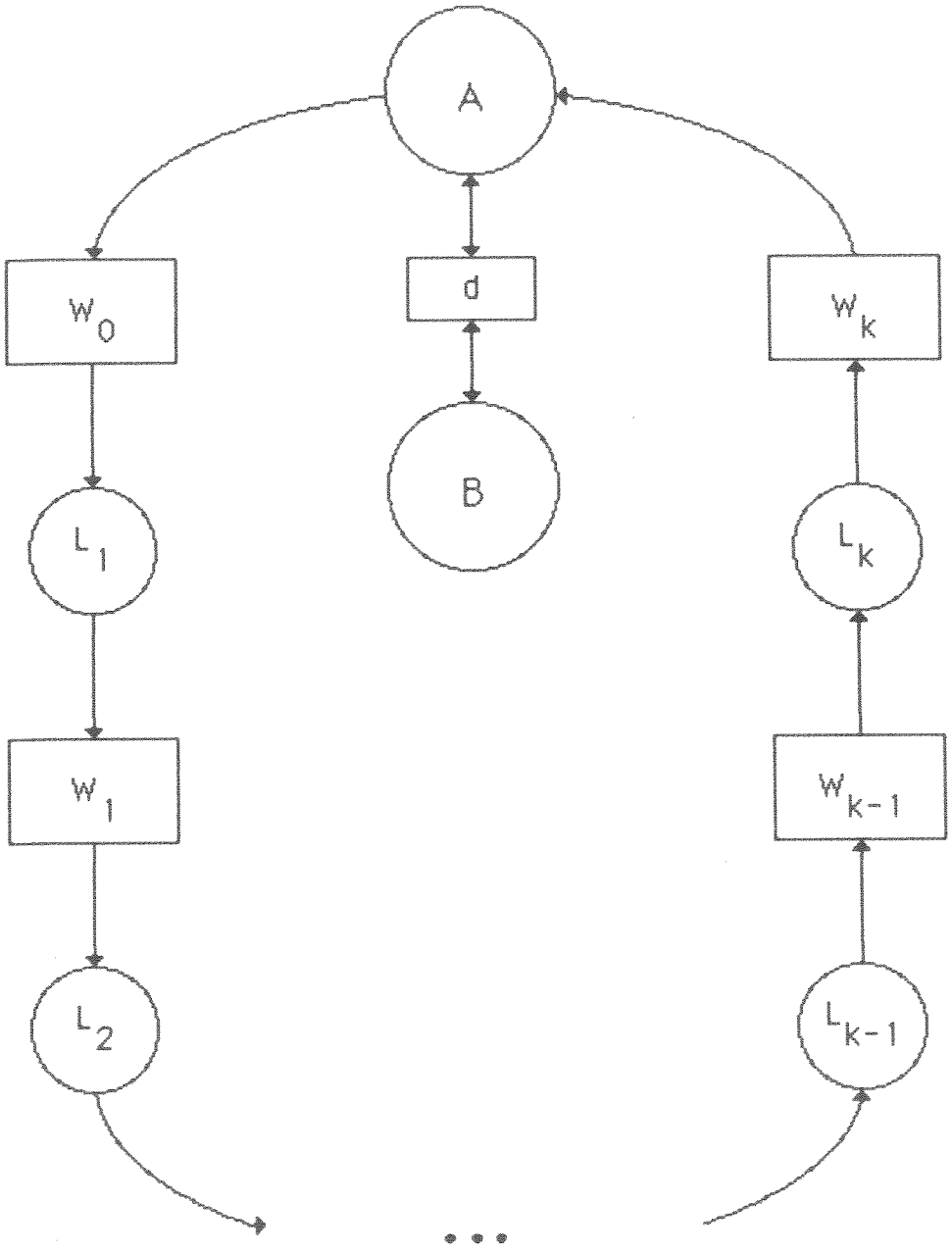

Figure 4.6 The system P constructed in Theorem 4.3.

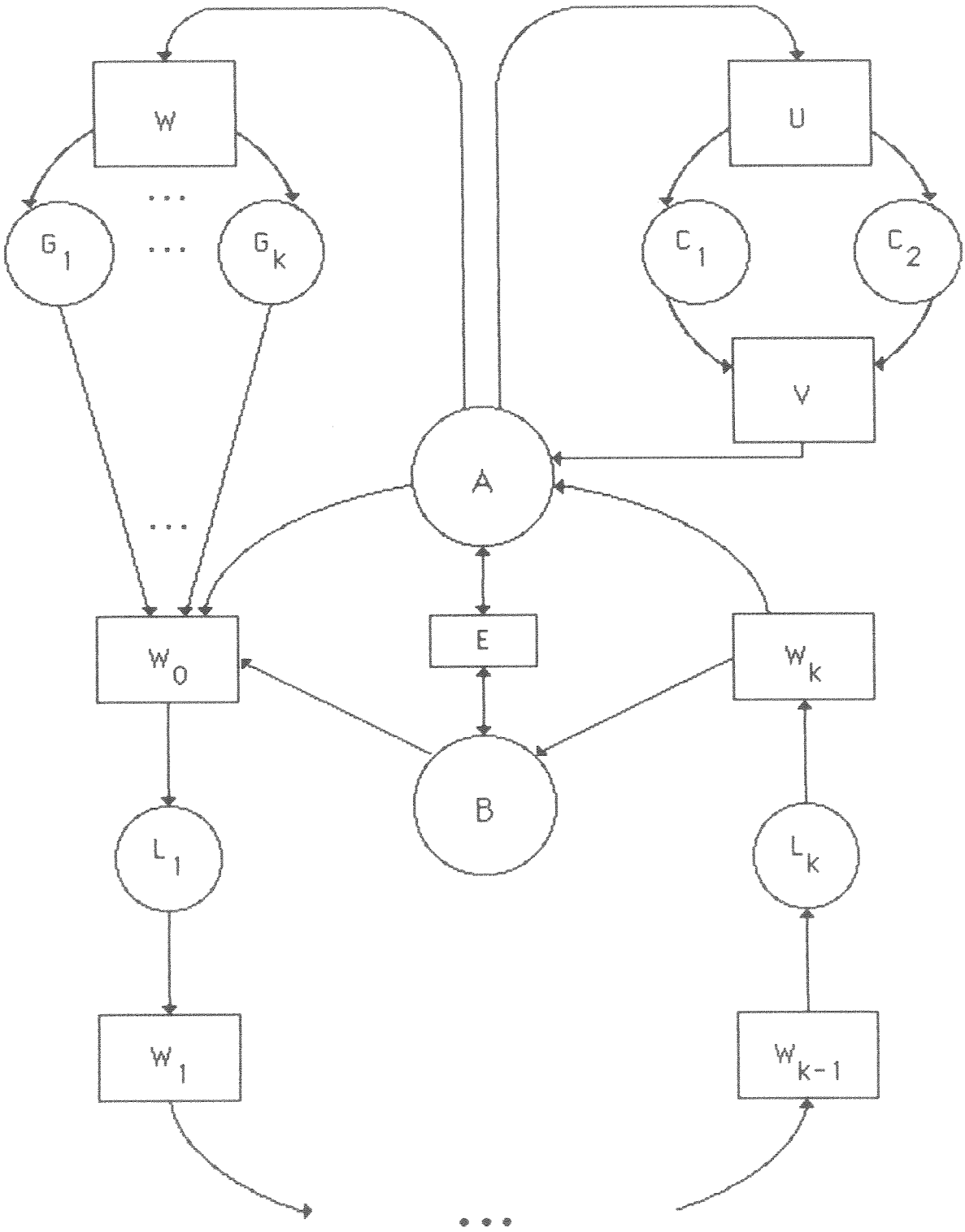

Figure 4.7 The system P constructed in Theorem 4.4.

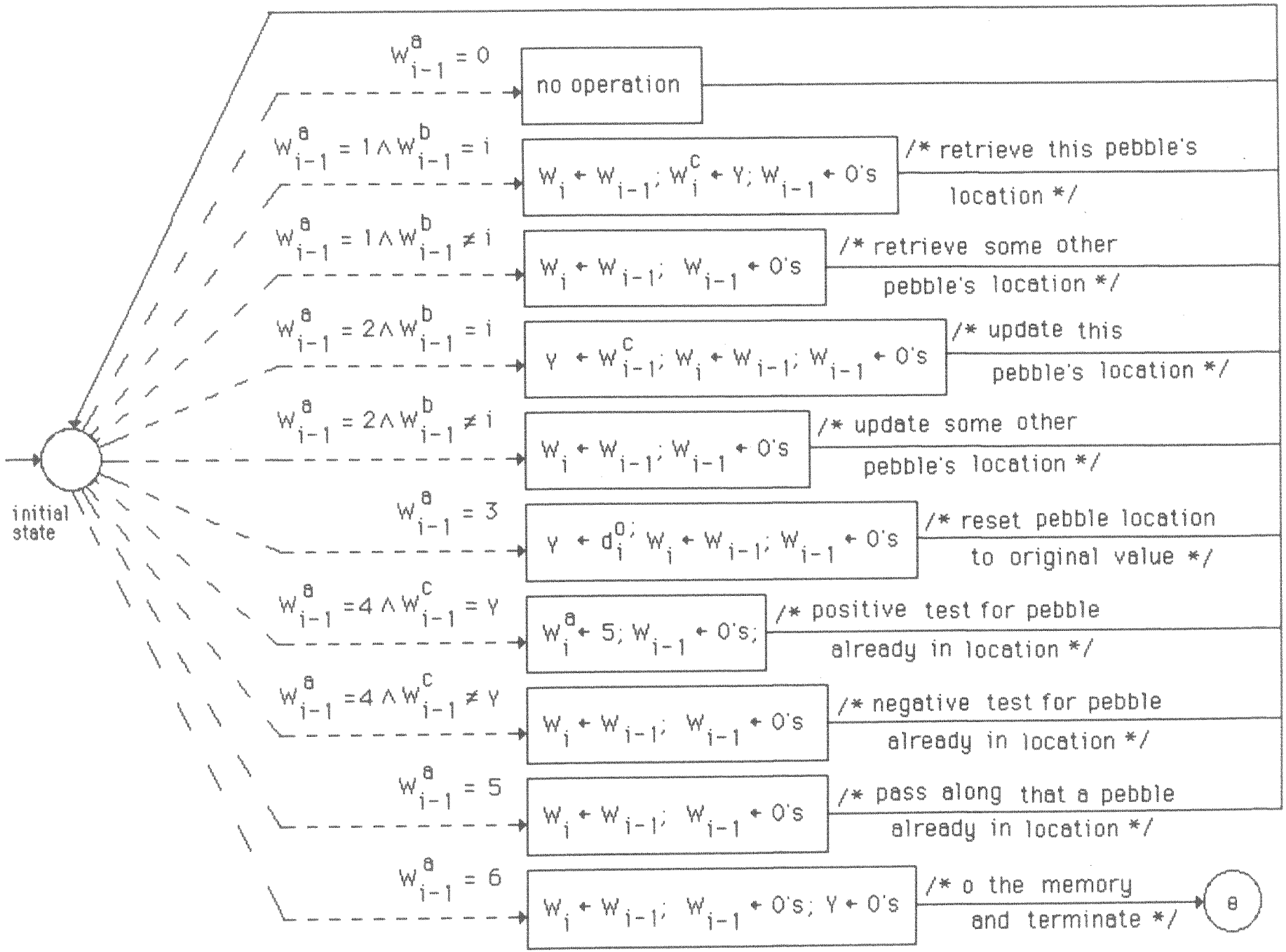

Figure 4.8 The flowchart for program L_i.

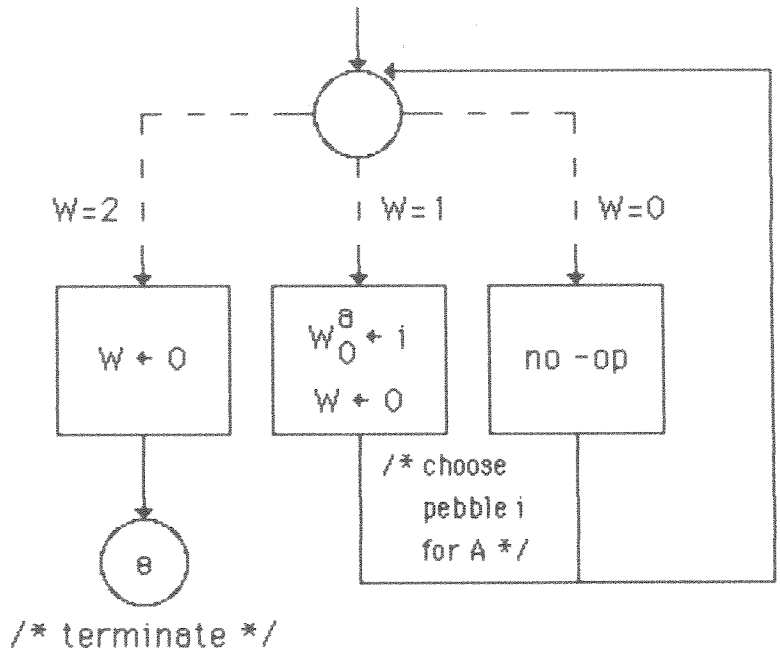

Figure 4.9 The flowchart for program G_i.

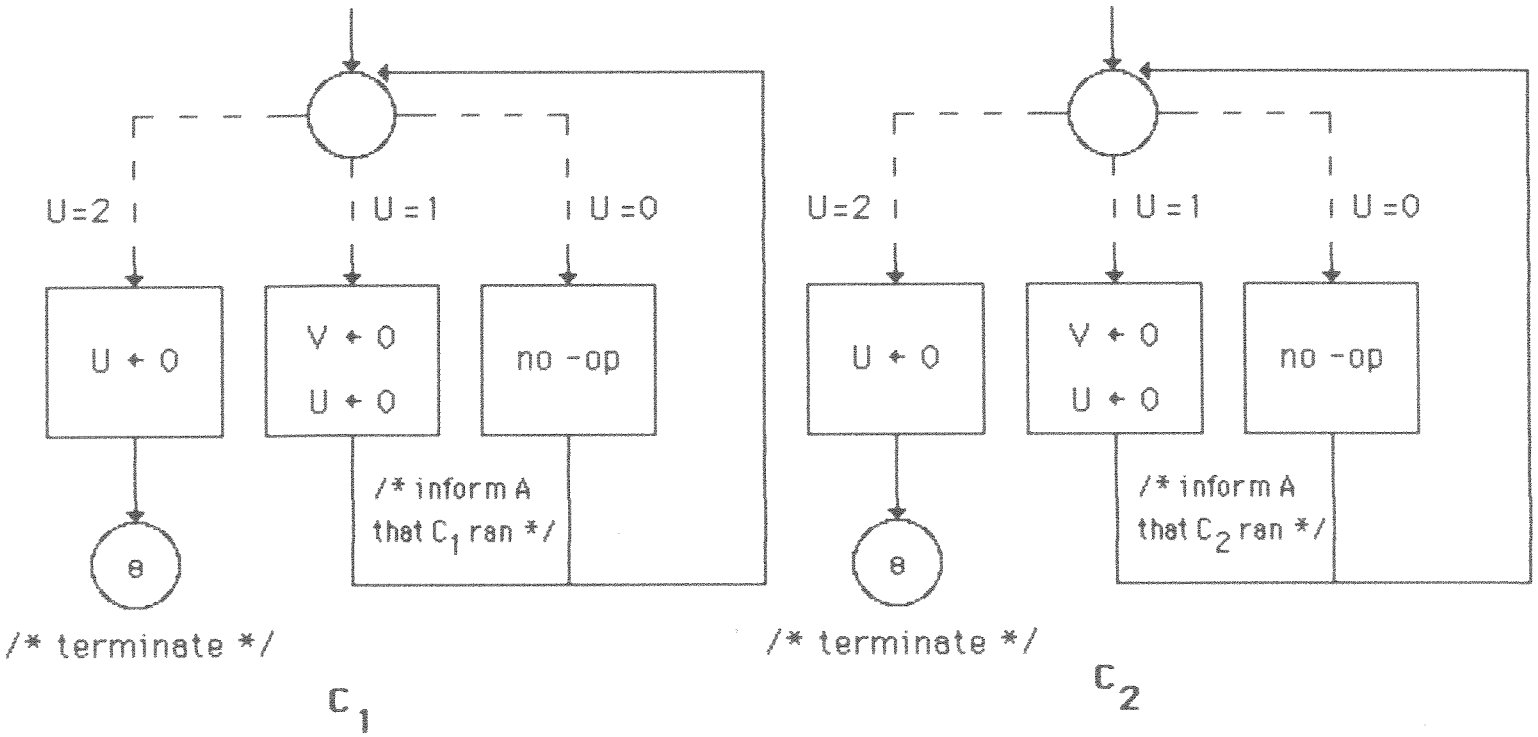

Figure 4.10 The flowcharts for programs C_1 and C_2.

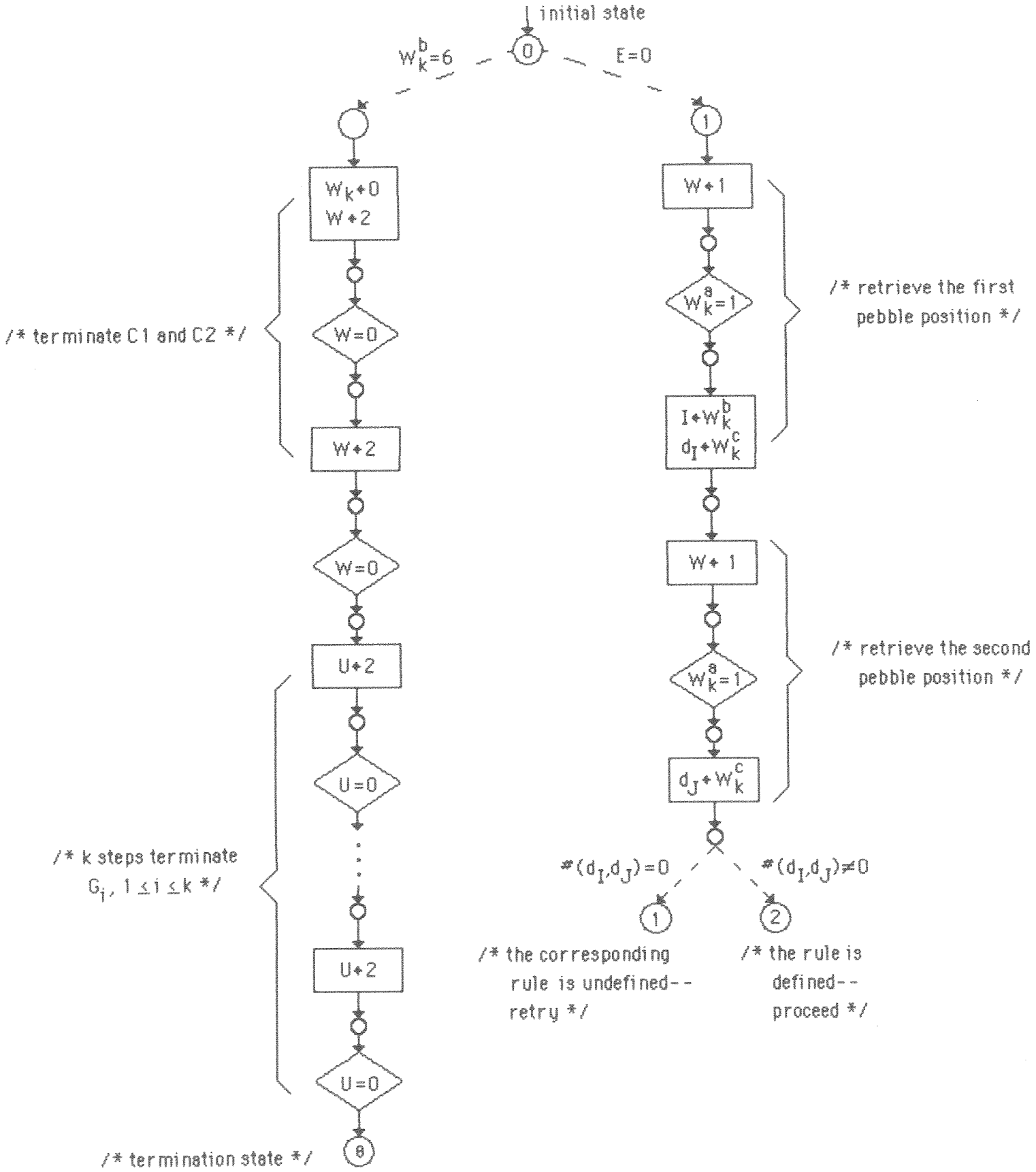

Figure 4.11 The flowchart of A.

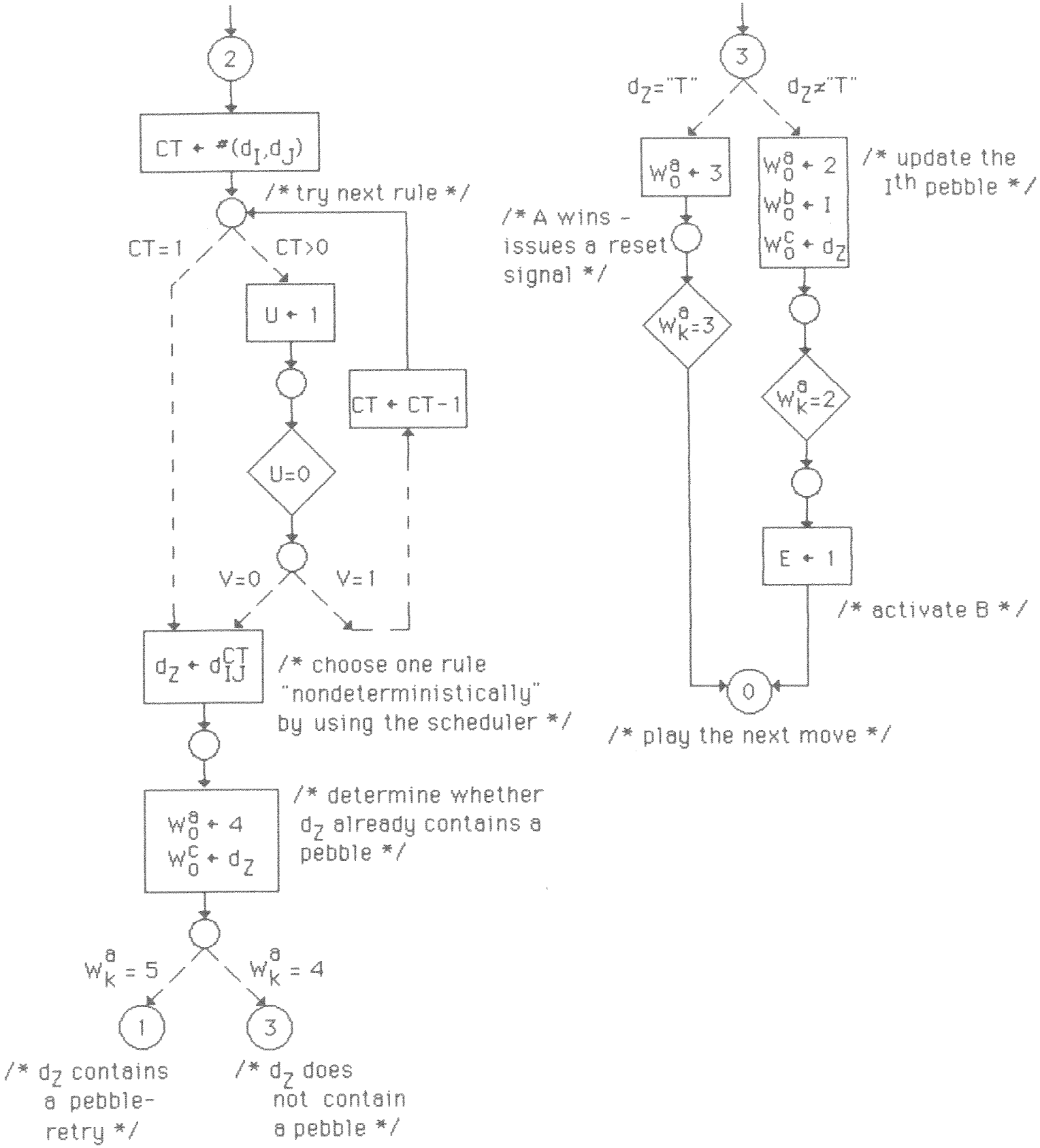

Figure 4.11 (Continued)

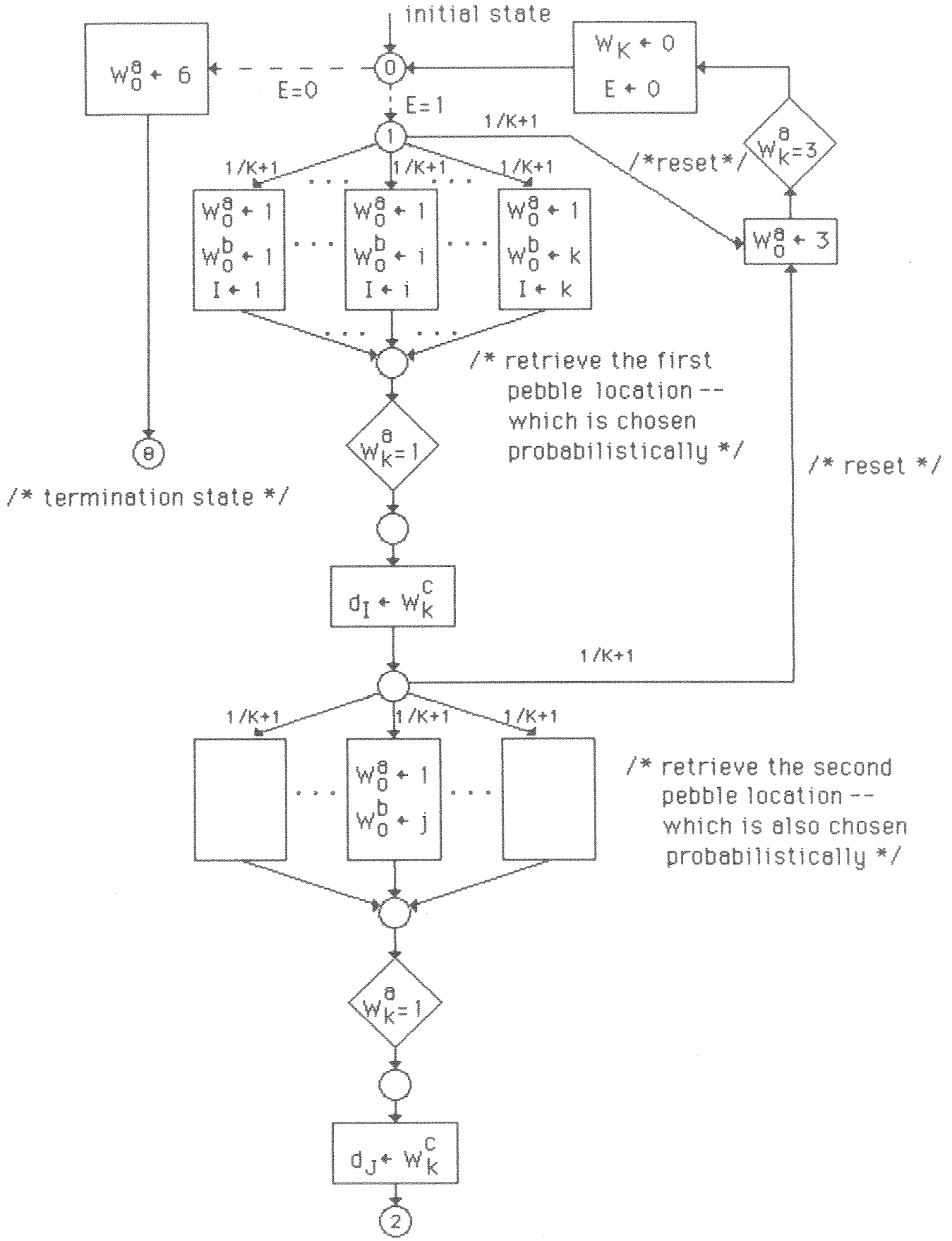

Figure 4.12 The flowchart of program B.

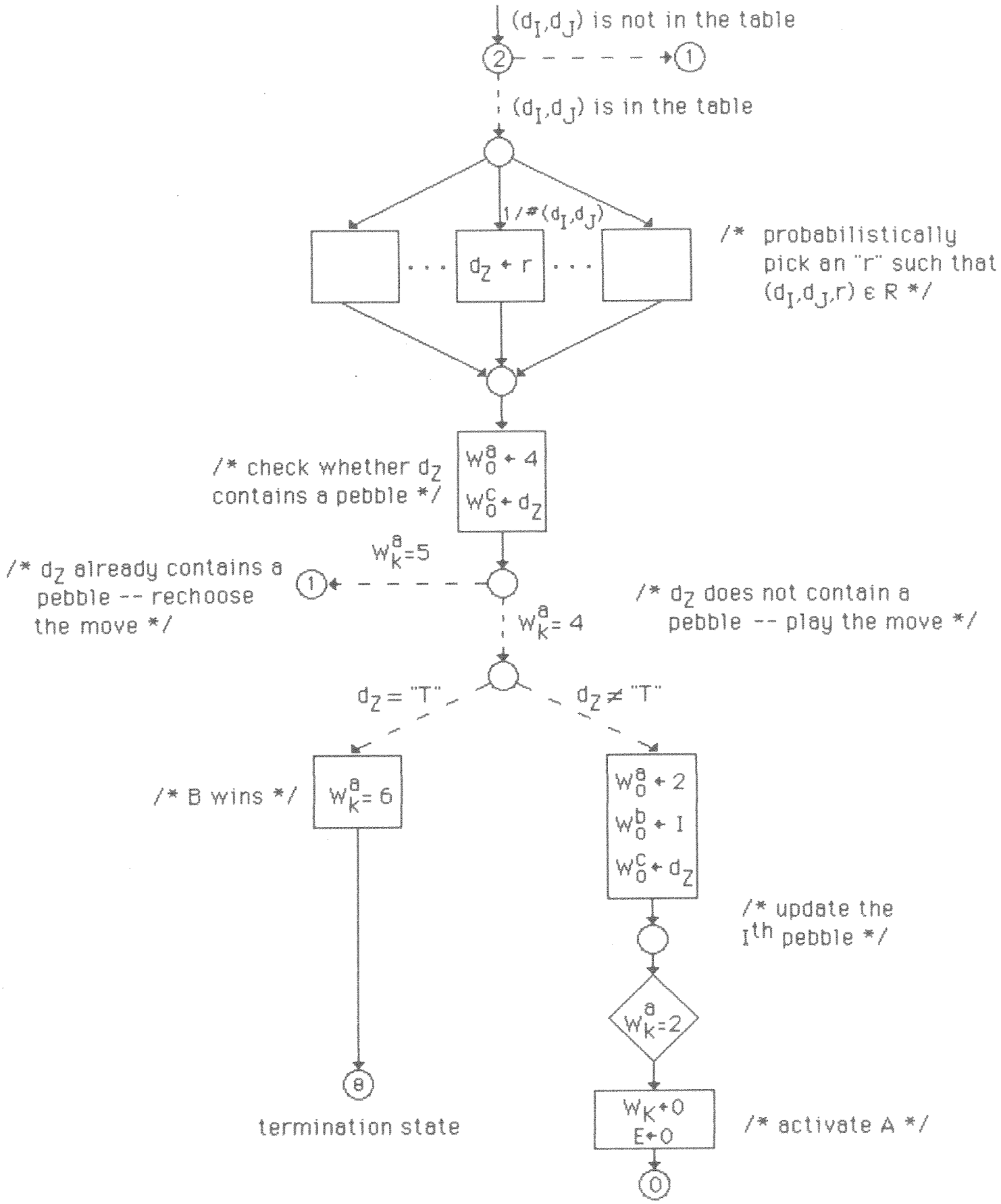

Figure 4.12 (Continued)