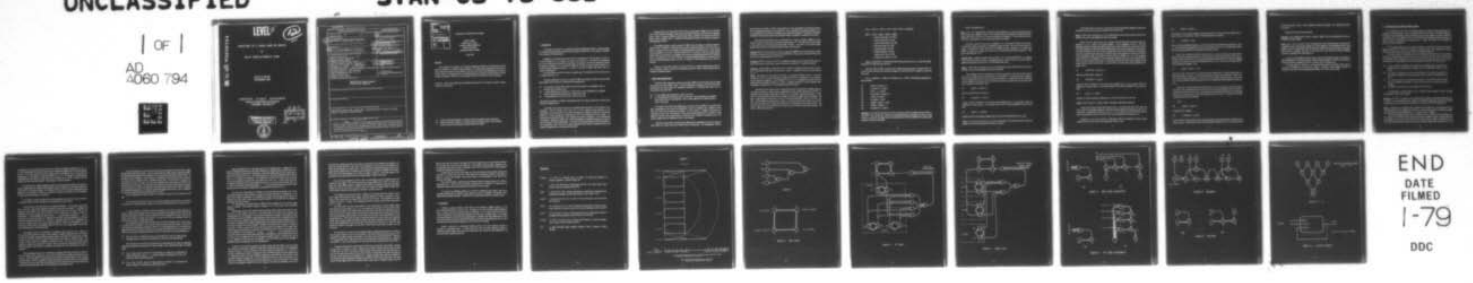
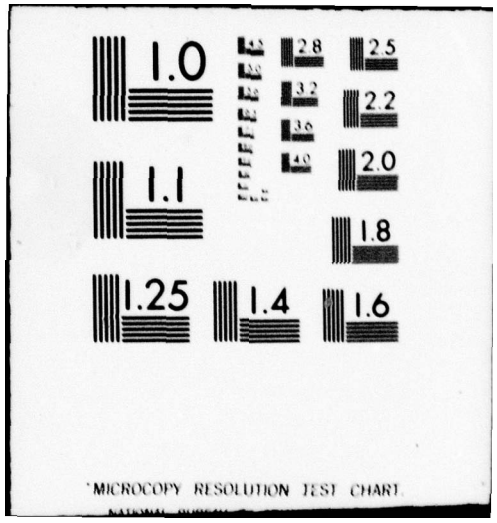
AD-A060 794 STANFORD UNIV CALIF DEPT OF COMPUTER SCIENCE F/G 12/1
VARIATIONS OF A PEBBLE GAME ON GRAPHS.(U)
SEP 78 J R GILBERT, R E TARJAN N00014-76-C-0688
UNCLASSIFIED STAN-CS-78-661 NL



END
DATE FILMED
1-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

LEVEL II

12

NW

VARIATIONS OF A PEBBLE GAME ON GRAPHS

by

John R. Gilbert and Robert E. Tarjan

STAN-CS-78-661
SEPTEMBER 1978

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: STAN-CS-78-661

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): VARIATIONS OF A PEBBLE GAME ON GRAPHS.

5. TYPE OF REPORT & PERIOD COVERED: technical rept.,

6. PERFORMING ORG. REPORT NUMBER: STAN-CS-78-661

7. AUTHOR(s): John R. Gilbert, Robert E. Tarjan

8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0688, NSF-MCS75-22870

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Stanford University
Computer Science Department
Stanford, Calif. 94305

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS:
Office of Naval Research
Department of the Navy
Arlington, Va. 22217

12. REPORT DATE: September 1978

13. NUMBER OF PAGES: 23 (25p.)

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
ONR Representative: Philip Surra
Durand Aeromautics Bldg., Rm. 165
Stanford University
Stanford, Calif. 94305

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

approved for public release
distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

black-white pebble game, pebble game, polynomial-space complete problems, register allocation, space bounds.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Abstract: We examine two variations of a one-person pebble game played on directed graphs, which has been studied as a model of register allocation. The black-white pebble game of Cook and Sethi is shown to require as many pebbles in the worst case as the normal pebble game, to within a constant factor. For another version of the pebble game, the problem of deciding whether a given number of pebbles is sufficient for a given graph is shown to be complete in polynomial space.

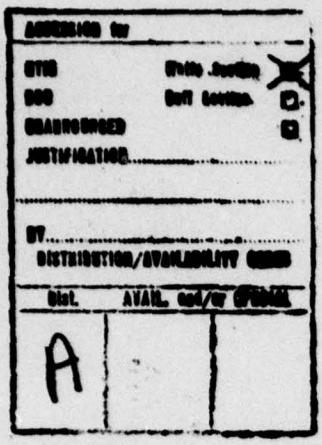
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

094 120

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LB



Variations of a Pebble Game on Graphs

John R. Gilbert*
Robert Endre Tarjan**
Computer Science Department
Stanford University
Stanford, California 94305

April 1978

Abstract.

We examine two variations of a one-person pebble game played on directed graphs, which has been studied as a model of register allocation. The black-white pebble game of Cook and Sethi is shown to require as many pebbles in the worst case as the normal pebble game, to within a constant factor. For another version of the pebble game, the problem of deciding whether a given number of pebbles is sufficient for a given graph is shown to be complete in polynomial space.

Keywords: black-white pebble game, pebble game, polynomial-space complete problems, register allocation, space bounds.

*: Research partially supported by a National Science Foundation graduate fellowship.
**: Research partially supported by National Science Foundation grant MCS-75-22870 and by Office of Naval Research contract N00014-76-C-0688.

1. Introduction.

A number of researchers have studied a one-person pebble game played on directed graphs as a model of storage allocation problems ([C], [CS], [HPV], [PH], [PT]). In this paper we consider two variations of the pebble game.

Suppose G is a directed graph with vertex set V and edge set E. We will write $G = (V, E)$. If (v, w) is an edge of G, v is a *predecessor* of w and w is a *successor* of v. The number of predecessors of v is its *in-degree* and the number of successors is its *out-degree*. We will be interested in graphs of bounded in-degree, so we will denote by $\mathcal{G}(n, d)$ the class of acyclic directed graphs on n vertices having maximum in-degree d. A *source* is a vertex of in-degree zero, and a *sink* is a vertex of out-degree 0.

We will consider time to be divided into integral steps. The notation $[a, b]$ will mean the set of integers i with $a \le i \le b$.

The *black pebble game* is played on a graph $G \in \mathcal{G}(n, 2)$ by placing a number of tokens called *black pebbles* on the vertices of G according to the following rules.

(a) At each time step, one black pebble may be either placed on an unpebbled vertex or removed from a pebbled vertex.

(b) A black pebble may be placed on a vertex only if all its predecessors are pebbled. Thus a black pebble may be placed on a source at any time.

(c) A black pebble may be removed from a vertex at any time.

The object of the game is to pebble a distinguished vertex of G, using no more than a certain fixed number of pebbles at once.

Intuitively, we can think of this game as modelling register allocation for the evaluation of an expression. We can consider each vertex of G to be an operator whose operands are its predecessors, so the sources of G are the atomic subexpressions. Pebbles are registers, and placing a pebble on a vertex corresponds to computing the value of the subexpression in the register. An operation can be performed only if all its operands are present in registers, *i.e.*, all its predecessors are pebbled. Removing a pebble from a vertex corresponds to freeing that register to be used to store the result of another computation. The vertex for the top-level operation in the expression is the distinguished vertex which we are trying to pebble, and the number of pebbles we use is the number of registers used in computing the expression.

Hopcroft, Paul, and Valiant [HPV] have shown that any graph in $\mathcal{G}(n, 2)$ can be pebbled with at most $O(n/\log n)$ pebbles. Paul, Tarjan, and Celoni [PTC] have proved that for infinitely many n there is a graph in $\mathcal{G}(n, 2)$ which requires $cn/\log n$ pebbles, so the bound in [HPV] is tight to within a constant factor. An algorithm which pebbles any graph with $O(n/\log n)$ pebbles is also presented in [PTC].

The *pebbling problem* is, given a graph $G \in \mathcal{G}(n, 2)$ and an integer k, to decide whether k pebbles suffice to pebble G. Sethi [S] has shown that the pebbling problem is NP-hard, that is, that any problem in NP can be reduced to the pebbling problem in polynomial time. It seems likely that this problem is not in NP. Sethi also shows that if we restrict the rules of the black pebble game so that no vertex can ever be pebbled more than once, the pebbling problem is NP-complete, that is, any problem in NP can be reduced to the restricted pebbling problem in polynomial time and the restricted pebbling problem is in NP. (A discussion of NP-completeness can be found in [AHU].)

Section 2 of this paper extends the result of [PTC] to a modified pebble game using two kinds of pebbles. Section 3 shows that the pebbling problem for another modified pebble game is complete in polynomial space.

2. Black-white pebble games.

The *black-white pebble game* is played on a graph $G \in \mathcal{G}(n, 2)$ with two types of tokens called *black pebbles* and *white pebbles*. Black pebbles are manipulated according to the rules of the black pebble game, and white pebbles are manipulated according to the following rules, which are in a sense duals of the black rules.

(d) A white pebble may be placed on a vertex at any time.

(e) A white pebble may be removed from a vertex only if all its predecessors are pebbled (with either black or white pebbles). Thus a white pebble may be removed from a source at any time.

The object of the black-white pebble game is to begin with no pebbles on the graph, make legal manipulations which cause the distinguished vertex to be pebbled with a pebble of either color, and finish with no pebbles on the graph. A fixed number k of pebbles is assumed to be available, but their color is not specified. The number of black pebbles and the number of white pebbles in use may vary as long as there are never more than k pebbles on the graph at once. Of course, a pebble may not change color while it is on the graph.

Intuitively, we can think of the black-white pebble game as modelling the proof of a theorem. Each vertex is a lemma which can be deduced from its predecessors. The distinguished vertex is

the theorem to be proved. Placing a black pebble on a vertex corresponds to proving that lemma from its predecessors; placing a white pebble on a vertex corresponds to assuming that lemma to be true, intending later to justify the assumption by proving the predecessors. The number of pebbles available is the maximum number of intermediate results it is possible to "remember" at one time.

Cook and Sethi [CS] show that for infinitely many n there are graphs in $\mathcal{G}(n, 2)$ which require a number of black and white pebbles proportional to $n^{1/4}$. The main result of this section is that there are in fact graphs requiring $cn/\log n$ pebbles. The proof in this section closely parallels that in [PTC]. We will assume their Lemma 1 and Corollary 1.

Lemma 1 [PTC]. For any value of i there is a graph $C(i) \in \mathcal{G}(c2^i, 2)$ with 2^i sources and 2^i sinks such that: For any $j \in [1, 2^i]$, if S is any set of j sources and T is any set of j sinks, then there are at least j vertex-disjoint paths in $C(i)$ from S to T.

Corollary 1 [PTC]. For any $j \in [0, 2^i-1]$, if j pebbles are placed on any j vertices of $C(i)$, and T is any set of at least $j+1$ sinks, then at least 2^i-j sources are connected to T via pebble-free paths.

Lemma 2. In any graph, if a path from vertex s to vertex t is pebble-free both at times t_1 and t_2, and t is pebbled at some time in the interval $[t_1, t_2]$, then s is pebbled at some time in the interval $[t_1, t_2]$.

Proof. By induction on the length of the path. If the path has length 0 then $s = t$ and the statement is trivial. If the path has length at least one then there is a successor s' of s on the path. By the induction hypothesis s' must be pebbled in $[t_1, t_2]$. If s' is pebbled black then s must be pebbled when the pebble is placed on s', and if s' is pebbled white then s must be pebbled when the pebble is removed from s'. This completes the proof of Lemma 2.

This lemma will be used to show a $cn/\log n$ lower bound on the number of pebbles required to pebble a sequence of graphs which are essentially the same as those constructed in [PTC]. In particular we will define a graph $G(i)$ for each $i \geq 10$. Let $G(10) = C(10)$ from Lemma 1. Then $G(i+1) = (V(i+1), E(i+1))$ is built from two copies of $G(i)$ and two copies of $C(i)$ as follows. Let $G(i) = (V(i), E(i))$ have sources $S(i) = \{s(i, j) : j \in [1, 2^i]\}$ and sinks $T(i) = \{t(i, j) : j \in [1, 2^i]\}$. Let $C(i) = (VC(i), EC(i))$ have sources $SC(i) = \{sc(i, j) : j \in [1, 2^i]\}$ and sinks $TC(i) = \{tc(i, j) : j \in [1, 2^i]\}$. Let $G_1(i)$ and $G_2(i)$ be two copies of $G(i)$ and let $C_1(i)$ and $C_2(i)$ be two copies of $C(i)$. Let $S(i+1) = \{s(i+1, j) : j \in [1, 2^{i+1}]\}$, and $T(i+1) = \{t(i+1, j) : j \in [1, 2^{i+1}]\}$ be two new sets of vertices. Let $G(i+1) = (V(i+1), E(i+1))$, where

$$V(i+1) = S(i+1) \cup T(i+1) \cup V_1(i) \cup V_2(i) \cup VC_1(i) \cup VC_2(i), \text{ and}$$

$$\begin{aligned} E(i+1) = {} & E_1(i) \cup E_2(i) \cup EC_1(i) \cup EC_2(i) \\ & \cup \{(s(i+1, j), t(i+1, j)) : j \in [1, 2^{i+1}]\} \\ & \cup \{(s(i+1, j), sc_1(i, j)) : j \in [1, 2^i]\} \\ & \cup \{(s(i+1, j+2^i), sc_1(i, j)) : j \in [1, 2^i]\} \\ & \cup \{(tc_1(i, j), s_1(i, j)) : j \in [1, 2^i]\} \\ & \cup \{(t_1(i, j), s_2(i, j)) : j \in [1, 2^i]\} \\ & \cup \{(t_2(i, j), sc_2(i, j)) : j \in [1, 2^i]\} \\ & \cup \{(tc_2(i, j), t(i+1, j)) : j \in [1, 2^i]\} \\ & \cup \{(tc_2(i, j), t(i+1, j+2^i)) : j \in [1, 2^i]\} . \end{aligned}$$

Figure 1 shows $G(i+1)$. The "left half" of $S(i+1)$ will refer to $\{s(i+1, j) : j \in [1, 2^i]\}$, with similar definitions for "right half" and for $T(i+1)$.

Let $m(i) = |S(i)| = |T(i)| = 2^i$, and let $n(i) = |V(i)|$ be the total number of vertices of $G(i)$. It is easy to show that $G(i)$ has maximum in-degree two and that there is a constant c_0 such that $n(i) \le c_0 i 2^i$.

Let $c_1 = 49/1024$, $c_2 = 3/1024$, $c_3 = 110/1024$, and $c_4 = 1/1024$. The following inequalities are easily verified:

(1) $c_3 m(i)/4 \ge 2c_2 m(i+1) + 1$

(2) $(1-4c_2)m(i+1) \ge c_3 m(i+1)$

(3) $c_2 m(i) - 1 \ge c_4 m(i+1)$

(4) $\lceil c_1 m(i+1)/8 \rceil \ge 2c_2 m(i+1) + 1$

(5) $c_1 m(i)/2 \ge 2c_2 m(i) + 1$

(6) $(1-2c_2)m(i) \ge c_1 m(i)$

(7) $c_3 m(i)/2 - 2c_2 m(i) \ge c_1 m(i)$

(8) $c_3 m(i)/2 \ge 2c_2 m(i) + 1$

(9) $(1-2c_2)m(i+1) \ge c_3 m(i+1)$.

Lemma 3. If in the time interval $[0, t]$ at least $c_1 m(i)$ sinks of $G(i)$ are pebbled with any colors in any order, and at times 0 and t there are at most $c_2 m(i)$ pebbles on the graph, then there is a time interval $[t_1, t_2] \subseteq [0, t]$ during which at least $c_3 m(i)$ sources of $G(i)$ are pebbled and at least $c_4 m(i)$ pebbles are always on the graph.

Proof is by induction on i.

Basis. Let $i = 10$. Suppose $G(10) = G(10)$ has 49 sinks pebbled in $[0, t]$, and at times 0 and t there are no more than 3 pebbled vertices. Any 7 of these 49 sinks are connected, via paths which are pebble-free both at 0 and at t, to at least 1018 sources, by Corollary 1. Thus at least one of the sinks, say v, is connected to at least 146 of the sources via such paths.

Let t_0 be a time in $[0, t]$ at which v is pebbled. Let t_1-1 be the last time before t_0 at which v is connected to these 146 sources via a pebble-free path, and let t_2+1 be the first time after t_0 at which v is connected to these 146 sources via a pebble-free path. During $[t_1, t_2]$, at least one pebble is always on the graph, and at least $146 \geq 110$ sources of $G(10)$ must be pebbled. This proves the lemma for $i = 10$.

Inductive step. Suppose the lemma holds for some $i \geq 10$. To prove the lemma for $i+1$, suppose that at least $c_1 m(i+1)$ sinks of $G(i+1)$ are pebbled during $[0, t]$, and that there are at most $c_2 m(i+1)$ pebbles on the graph at times 0 and t. We will consider four cases.

Case 1. There exists an interval $[t_1, t_2] \subseteq [0, t]$ during which at least $c_3 m(i)/4$ sources of $G_1(i)$ are pebbled and at least $c_2 m(i)$ pebbles are always on the graph.

The subgraph of $G(i+1)$ consisting of all vertices and edges on paths from the left half of $S(i+1)$ to the sources of $G_1(i)$ satisfies Lemma 1 and Corollary 1. So does the similar subgraph from the right half of $S(i+1)$. Let t_0 be the last time before t_1 at which there are not more than $c_2 m(i+1)$ pebbles on the graph, and let t_3 be the first time after t_2 at which there are not more than $c_2 m(i+1)$ pebbles on the graph. Since

$$c_3 m(i)/4 \geq 2c_2 m(i+1) + 1, \tag{1}$$

there are at least $2\,(m(i) - 2c_2 m(i+1)) =$

$$(1-4c_2)m(i+1) \geq c_3 m(i+1) \tag{2}$$

sources of $G(i+1)$ connected to the $c_3 m(i)/4$ sources pebbled from t_1 to t_2 by paths which are pebble-free at both t_0 and t_3. At least these sources of $G(i+1)$ must be pebbled in $[t_0, t_3]$, and at least

$$c_2 m(i) - 1 \geq c_4 m(i+1) \tag{3}$$

pebbles must be on the graph throughout $[t_0, t_3]$. Thus the lower bound holds in this case.

Case 2. There exists an interval $[t_1, t_2] \subseteq [0, t]$ during which at least $c_3 m(i)/4$ sources of $G_2(i)$ are pebbled and at least $c_2 m(i)$ pebbles are always on the graph.

The lemma holds in this case by a proof like that above, considering subgraphs whose only intersections with $G_1(i)$ are the direct connections from $S_1(i)$ to $T_1(i)$.

Case 3. There exists an interval $[t_1, t_2] \subseteq [0, t]$ during which at least $c_1m(i+1)/4$ sinks of $G(i+1)$ are pebbled and at least $c_2m(i)$ pebbles are always on the graph.

Either the left or right half of $T(i+1)$ contains at least $\lceil c_1m(i+1)/8 \rceil$ of the sinks which are pebbled in $[t_1, t_2]$. Again we will apply Corollary 1 to two subgraphs of $G(i+1)$. The first subgraph contains all vertices and edges on paths from the left half of $S(i+1)$ to the sinks $S_1(i)$ of $G_1(i)$ (including all of $C_1(i)$); the direct connections from $S_1(i)$ to $T_1(i)$; the edges from $T_1(i)$ to $S_2(i)$; the direct connections from $S_2(i)$ to $T_2(i)$; the edges from $T_2(i)$ to $SC_2(i)$; all vertices and edges on $m(i)$ vertex-disjoint paths from $SC_2(i)$ to $TC_2(i)$ in $C_2(i)$; and the edges from $TC_2(i)$ to the half of $T(i+1)$ containing at least $\lceil c_1m(i+1)/8 \rceil$ of the sinks which are pebbled in $[t_1, t_2]$. This subgraph satisfies Lemma 1 and Corollary 1, as does the similar subgraph which starts from the right half of $S(i+1)$. Let t_0 be the last time before t_1 at which there are not more than $c_2m(i+1)$ pebbles on the graph and let t_3 be the first time after t_2 at which there are not more than $c_2m(i+1)$ pebbles on the graph. Since

$$\lceil c_1m(i+1)/8 \rceil \geq 2c_2m(i+1) + 1, \tag{4}$$

there are at least $2(m(i) - 2c_2m(i+1)) =$

$$(1-4c_2)m(i+1) \geq c_3m(i+1) \tag{2}$$

sources of $G(i+1)$ connected to the $c_1m(i+1)$ sinks pebbled from t_1 to t_2 by paths which are pebble-free at both t_0 and t_3. At least these sources of $G(i+1)$ must be pebbled in $[t_0, t_3]$, and at least

$$c_2m(i) - 1 \geq c_4m(i+1) \tag{3}$$

pebbles must be on the graph throughout $[t_0, t_3]$. Thus the lemma holds in this case.

Case 4. None of cases (1) - (3) hold. Figure 1 may help in following this argument.

Since case (3) does not hold, there must be a time $t_1 \in [0, t]$ such that fewer than $c_1m(i+1)/4$ sinks of $G(i+1)$ are pebbled during $[0, t_1]$ and the number of pebbles on the graph at time t_1 is at most $c_2m(i)$. Similarly there must be a time $t_{10} \in [0, t]$ such that fewer than $c_1m(i+1)/4$ sinks of $G(i+1)$ are pebbled during $[t_{10}, t]$ and there are at most $c_2m(i)$ pebbles on the graph at t_{10}.

During $[t_1, t_{10}]$ at least $c_1m(i+1)/2 = c_1m(i)$ sinks of $G(i+1)$ are pebbled, of which at least $c_1m(i)/2$ must be in either the left or the right half of $T(i+1)$. Since

(5) $c_1 m(i)/2 \geq 2c_2 m(i) + 1,$

the number of sinks of $G_2(i)$ connected to these sinks of $G(i+1)$ via paths which are pebble-free at both t_1 and t_{10} is at least $m(i) - 2c_2 m(i)$. Thus at least these $m(i) - 2c_2 m(i) =$

(6) $(1-2c_2)m(i) \geq c_1 m(i)$

sinks of $G_2(i)$ are pebbled during $[t_1, t_{10}]$, with no more than $c_2 m(i)$ pebbles on the graph at t_1 and t_{10}. By the induction hypothesis there is a time interval $[t_2, t_9] \subseteq [t_1, t_{10}]$ during which $c_3 m(i)$ sources of $G_2(i)$ are pebbled and $c_4 m(i)$ pebbles are always on $G_2(i)$.

Since case (2) does not hold, there must be a time $t_3 \in [t_2, t_9]$ such that fewer than $c_3 m(i)/4$ sources of $G_2(i)$ are pebbled during $[t_2, t_3]$ and there are at most $c_2 m(i)$ pebbles on $G(i+1)$ at t_3. Similarly there must be a time $t_8 \in [t_2, t_9]$ such that fewer than $c_3 m(i)/4$ sources of $G_2(i)$ are pebbled during $[t_8, t_9]$ and there are at most $c_2 m(i)$ pebbles on $G(i+1)$ at t_8. Thus in $[t_3, t_8]$ at least $c_3 m(i)/2$ sources of $G_2(i)$ are pebbled. Since

(7) $c_3 m(i)/2 - 2c_2 m(i) \geq c_1 m(i),$

there are at least $c_1 m(i)$ sinks of $G_1(i)$ connected to these $c_3 m(i)$ sources of $G_2(i)$ by paths which are pebble-free both at t_3 and at t_8. During $[t_3, t_8]$ these sinks of $G_1(i)$ must be pebbled. There are at most $c_2 m(i)$ pebbles on the graph at t_3 and t_8, so by the induction hypothesis there is an interval $[t_4, t_7] \subseteq [t_3, t_8]$ during which $c_3 m(i)$ sources of $G_1(i)$ are pebbled and $c_4 m(i)$ pebbles are always on $G_1(i)$.

Since case 1 does not hold, there must be a time $t_5 \in [t_4, t_7]$ such that fewer than $c_3 m(i)/4$ sources of $G_1(i)$ are pebbled during $[t_4, t_5]$ and there are at most $c_2 m(i)$ pebbles on $G(i+1)$ at time t_5. Similarly there must be a time $t_6 \in [t_4, t_7]$ such that fewer than $c_3 m(i)/4$ sources of $G_1(i)$ are pebbled during $[t_6, t_7]$ and there are at most $c_2 m(i)$ pebbles on $G(i+1)$ at time t_6. During $[t_5, t_6]$ at least $c_3 m(i)/2$ sources of $G_1(i)$ are pebbled.

Since

(8) $c_3 m(i)/2 \geq 2c_2 m(i) + 1,$

at least $2(m(i) - 2c_2 m(i)) =$

(9) $(1-2c_2)m(i+1) \geq c_3 m(i+1)$

sources of $G(i+1)$ are connected to these $c_3 m(i)/2$ sources of $G_1(i)$ by paths which are pebble-free at both t_5 and t_6. These sources of $G(i+1)$ must be pebbled during $[t_5, t_6] \subseteq [t_4, t_7] \subseteq [t_2, t_9] \subseteq [0, t]$

while at least $c_4m(i) + c_4m(i) = c_4m(i+1)$ pebbles are always on the graph. This completes the proof of Lemma 3.

Finally we can prove the main theorem.

Theorem 1. For infinitely many n, there is a graph $G \in \mathcal{G}(n, 2)$ such that pebbling some vertex in G requires $c_5n/\log n$ pebbles.

Proof. For $n = n(i)$, $i \geq 10$, let $G = G(i)$. Since pebbling all sinks of $G(i)$ beginning and ending with no pebbled vertices requires $c_4m(i)$ pebbles, there must be some sink whose pebbling requires $c_4m(i)$ pebbles, or else we could pebble all the sinks one after another, with the graph empty at some point after each sink is pebbled. Since $m(i) = 2^i$ and $G(i)$ has $n(i) \leq c_0i2^i$ vertices, the number of pebbles required is $c_5n(i)/\log n(i)$ for some constant c_5.

Since any black pebbling strategy is also a black-white pebbling strategy, the $O(n/\log n)$-pebble algorithm in [PTC] also works for the black-white pebble game. Thus the lower bound on worst-case number of pebbles in Theorem 1 is tight to within a constant factor.

The only place that Theorem 1 uses any information about the conditions under which a vertex can be pebbled is in the proof of Lemma 2. Therefore at least $c_5n/\log n$ pebbles are required in the worst case of any pebble game on $\mathcal{G}(n, 2)$ for which Lemma 2 holds.

3. A polynomial-space complete pebbling problem.

In this section we show that the pebbling problem for another modified pebble game is complete in polynomial space. That is, any problem which can be solved in polynomial space can be reduced to this pebbling problem in logarithmic space, and the problem can itself be solved in polynomial space. Any problem which can be solved nondeterministically in polynomial space can also be solved deterministically in polynomial space. Again, see [AHU] for a discussion of polynomial-space completeness.

This modified pebble game will use only black pebbles. We will find it convenient to allow a pebble to move from a vertex to its successor in a single time step. This "sliding rule" does not affect our results, since we shall see that it always saves exactly one pebble.

The major change in the pebble game is the introduction of cyclic graphs and "or vertices." Let G be a directed graph in which every vertex has in-degree at most two. Let every vertex of G be designated either an and vertex or an or vertex, and let t be a particular vertex. We are given some number k of black pebbles which can be manipulated according to the following rules.

(a) At each time step, one pebble may be placed on an unpebbled vertex, removed from a pebbled vertex, or moved from a pebbled vertex to an unpebbled successor of that vertex.

(b) If a vertex is an and vertex, a pebble may be placed on it or moved to it only if all its predecessors are pebbled. Thus an and vertex which is a source can be pebbled at any time.

(c) If a vertex is an or vertex, a pebble may be placed on it or moved to it only if at least one of its predecessors is pebbled. Thus an or vertex which is a source can never be pebbled.

(d) A pebble may be removed from any pebbled vertex at any time.

The problem is to decide whether or not it is possible to place a pebble on vertex t by legal manipulations using at most k pebbles.

Lemma 4. For all $k > 0$, a graph G can be pebbled using k pebbles in the modified pebble game if and only if it can be pebbled using $k+1$ pebbles in the modified pebble game without ever moving a pebble from a vertex to one of its successors, that is, without ever using the "sliding rule."

Proof. Suppose G can be pebbled using k pebbles with the sliding rule. We replace each use of the sliding rule to move a pebble from a vertex x to a vertex y with two steps, first placing a pebble on y and then removing the pebble from x. This uses at most $k+1$ pebbles.

Conversely, suppose a scheme exists for pebbling G with $k+1$ pebbles without the sliding rule, for some $k > 0$. We transform this scheme into one using k pebbles as follows. Suppose there are $k+1$ pebbles on G at time t_0. Then the move at t_0 must be to place a pebble on some vertex y. If this is not the final move, the move at t_0+1 must be to remove a pebble from some vertex x. If x is

a predecessor of y we replace these two moves with a single move sliding the pebble from x to y. If x is not a predecessor of y we reverse the moves, first removing the pebble from x and then placing it on y. If t_0 is the final move we slide a pebble from any predecessor of y to y (if y has no predecessors we just pebble it, using one pebble, since $k > 0$). We apply this transformation simultaneously to all instants at which there are $k+1$ pebbles on G, and get a scheme with the sliding rule which uses only k pebbles. This completes the proof of Lemma 4.

The proof that the modified pebbling problem is polynomial-space complete will proceed by using a pebble graph to simulate boolean registers, gates, and signal lines, essentially identical to those used in real-world hardware. The devices can then be used to build a polynomial-space bounded Turing machine, a PDP-10, or whatever is desired; the simulation of a PS-bounded Turing machine will be given in some detail.

In the figures to follow, an and vertex will be represented by a circle-dot ⊙, an or vertex by a circle-plus ⊕, and a vertex whose type is not mentioned by an empty circle ○.

In the first step of the construction a few simplifying assumptions will be made, to be proved later. The first is that certain subgraphs will always contain exactly one pebble. A prologue and epilogue will be added to the basic construction to ensure that this is the case. Also we will generalize the notion of and and or vertices to allow vertices with an arbitrary number of predecessors. If the predecessors of vertex v are $w_1, \ldots, w_n$, and boolean variables $x_1, \ldots, x_n$ are such that x_i is true if and only if w_i is pebbled, we will allow the rule for pebbling v to be any monotone boolean function of the x_i's. Using  to represent "and" and to represent "or", we can for example have the five-vertex "graph" in Figure 2. There v can be pebbled if w_1 and either w_2 or both w_3 and w_4 are pebbled. Note that and do not represent vertices, but only building blocks for "generalized edges."

The boolean circuits we will simulate are composed of *gates* and *logic lines*. A gate is a device with some inputs and an output, which computes a particular boolean function of its inputs. A logic line is simply a path connecting one gate's output to some inputs of other gates (or of itself). An initial value, *true* or *false*, is given for each logic line and hence for each gate input. The boolean circuits function by repeatedly executing two steps. First every gate simultaneously computes its output as a function of its inputs. Then every logic line simultaneously picks up the value of the gate to whose output it is connected, and applies that value to each input to which it is connected.

The functioning of the circuit is controlled by a subgraph called the *clock*, shown in Figure 3. This is a cycle of four vertices a, b, c, and d, and will always have a pebble on exactly one of its vertices. This clock pebble will move around the cycle once for each iteration of the two steps described above. While a is pebbled the gates will compute their outputs, and for b to be pebbled all outputs will have to have been correctly computed. The transfer of values along the logic lines will be similarly controlled by c and d. One such iteration will be called a *tick*.

A gate will consist of a two-vertex cycle for each input and output, and some "control" edges between these cycles and the clock. Each two-vertex cycle will always contain exactly one pebble. In one position this pebble will be interpreted as a true value, and in the other as a false value. An *and* gate is shown in Figure 4. Clock vertex a is a predecessor of both output vertices t_c and f_c, so the output value can change when a is pebbled and only then. All the vertices in the gate are predecessors of vertex b in such a way that b can be pebbled only if the output of the gate is the correct function of its inputs. The reason for the complexity of the network connecting b to the gate vertices is that more vertices will eventually have to be added to each cycle in order to eliminate our simplifying assumptions. The network of generalized edges will ensure that, when b is pebbled, no cycle has its pebble anywhere else besides the t or f vertex.

A gate computing any boolean function of any number of inputs can be constructed in this way.

The structure involving clock vertices c and d which implements the logic lines is very similar to the above. An example is given in Figure 5 of an output a being applied to two inputs b and c.

It is now necessary to modify the above constructions to use only two-input and and or vertices. Notice that at present all the vertices are partitioned into subgraphs, each of which has been assumed always to contain exactly one pebble: One such subgraph is the clock cycle and the others are the input or output cycles of gates. The clock cycle is the only subgraph which has vertices which are not two-input and vertices.

The modifications which follow will add vertices to the clock subgraph, but we will continue to assume that each subgraph in the partition always contains exactly one pebble. We will call these subgraphs single pebble subgraphs, or SPSG's. In the following figures, as in the earlier ones, edges within a SPSG are drawn as wide lines and those between SPSG's are drawn as narrow lines. With this convention the following statement is true, and will be preserved by the modifications.

(*) Each or vertex in a SPSG has exactly two wide edges and no narrow edges entering. Each and vertex has exactly one wide edge and at most one narrow edge entering.

We will work our way down to a graph with only simple and and or vertices by repeatedly making the following two substitutions to replace complex narrow "generalized edges" by simpler edges.

(I) For a vertex v with a narrow "and" edge entering as in Figure 6a, we substitute the graph in Figure 6b. Here $x_1, \ldots, x_i$ may be vertices or the symbols  and  with more narrow edges entering them.

(II) For a vertex v with a narrow "or" edge entering as in Figure 7a, we substitute the graph in Figure 7b. Recall that ⊛ vertices are or vertices.

These substitutions give an equivalent graph in the following sense: Suppose that in a certain configuration before one of the above modifications, a pebble could be moved from u to v. Then this is also the case after the modification. Conversely, suppose that after the modification a pebble can be moved from u to v. For modification (II) it is clear that the pebble could be moved from u to v without the modification. For modification (I), notice that no gate input or output vertex can be pebbled except when the clock pebble is on vertex a or c. Thus no gate vertex can be pebbled between the time the clock pebble moves from u to v_1 and the time it moves from v_i to v. Thus the conditions represented by edges $x_1, \ldots, x_i$ must all have been satisfied when the clock pebble was on u, so without the modification the pebble could be moved from u to v.

We now have a construction for a boolean circuit which computes according to the rule of alternately letting every gate compute its output and letting every output be propagated to the inputs with which it is connected. The construction uses only two-input and and or vertices, but assumes that every SPSG always contains exactly one pebble. We will now add an epilogue to guarantee that this is the case.

It is a consequence of (*) above that if a SPSG ever has no pebbles on it, it can never again contain a pebble. (The problem of getting a pebble on it in the first place will be discussed presently.)

The boolean circuit we will construct will have one gate output which is the "answer" in the sense that if that output is ever true, we will accept, that is, we will be able to pebble the distinguished sink t of the graph. If we can force the presence of a pebble on each SPSG at some time between the time the "answer" output becomes true and the time vertex t is pebbled, there will have to have been a pebble on each SPSG during the whole computation of the boolean circuit. If in addition we choose k, the number of pebbles, equal to the number of SPSG's, then no SPSG can ever have contained more than one pebble. Therefore the structure shown in Figure 8 is added to the graph. Vertices t_0 and f_0 are the true and false vertices of the output of the "answer" gate. When this gate first computes a true output the clock pebble will be on vertex b, so the pebble on t_0 (or the pebble on b) can be moved to e. Vertices $t_1, \ldots, t_m$ and $f_1, \ldots, f_m$ are respectively the true and false vertices of all the other inputs and outputs. The pebble on e can be moved all the way to t if one of (t_i, f_i) is pebbled for all i, and conversely no pebble can reach t unless for all i one of (t_i, f_i) is pebbled at some time after t_0 is pebbled. Thus every SPSG has to have contained a pebble between the computation of t_0 and the pebbling of t.

Finally we will address the problem of getting a pebble onto each SPSG in the first place. We would like to be able to specify an initial value for each logic line and let the computation proceed from there. Thus we would like to be able to pebble a specified one of (t_i, f_i) for each i, and also to pebble clock vertex a. Suppose that we have k SPSG's, namely one clock subgraph and $k-1$ inputs and outputs, so we have k pebbles. Notice that in each SPSG the vertex which we wish to pebble initially is an and vertex. If in the i'th SPSG this vertex is v_i, then the situation is as in Figure 9a, recalling (*), except that the narrow edge from x_i to v_i may not be present. We replace this with the subgraph in Figure 9b. Here p is a single vertex which is a predecessor of all the v_i's.

We have now violated (*) because an or vertex in each SPSG has an entering narrow edge, *i.e.*, an edge from outside the SPSG. This means that the pebble can be removed from the SPSG at any time and later replaced at its initial vertex. To keep this from fouling everything up we add to p a graph which requires that all k pebbles be used to pebble p, so removing a pebble from a SPSG will require the whole pebbling to be started over again to replace it. The added graph is the pyramid S_k which Cook [C] has proved to require k pebbles. Figure 10 shows S_4.

The only thing remaining to check is that the initialization must be complete before the computation can begin. This is the case because the clock pebble cannot move to b until every input and output SPSG contains a pebble; no input SPSG can change its value until the clock pebble has moved all the way to c; and we don't care whether the output SPSG's change their values because moving the clock pebble to b will ensure that they are correct.

This completes the main construction. Suppose we are trying to simulate a boolean circuit with n gates, each having at most m inputs. The total number of vertices in all gates is at most $3n(m+1)$ since each input and output SPSG finally has three vertices. The epilogue has at most another $3(n-1)(m+1) + 1$ vertices. The prologue has exactly $(n+1)(n+2)/2$ vertices since the pyramid is $k = n+1$ wide.

The clock SPSG had four vertices initially. In the modifications to remove complex edges, each time one of the symbols  or  was eliminated the number of vertices added to the clock SPSG was linear in the number of lines into the symbol. Each gate was originally connected to clock vertex b by a network which may have had size exponential in the number of gate inputs. Thus the contribution from all gates is at most $c(m)n$ vertices, where $c(m)$ is exponential in m but independent of n. The size of the network connecting a logic line to clock vertex d was originally linear in the number of inputs connected to the line. Each input is connected to only one logic line, so the total contribution of clock vertices from all the logic lines is linear in mn. Thus the total number of vertices in the graph is at most $n^2/2 + c'(m)n$, and the total number of edges is at most twice that.

An informal way to show that the pebbling problem is polynomial-space hard is to observe that the gate in Figure 11 acts as a one-bit memory. With these registers and ordinary *and*, *or*, and *not* gates, any modern computer with storage n can be built with at most $O(n^2)$ devices. Therefore a Turing machine with a tape bound $p(n)$ can be simulated with $O(p(n)^2)$ devices, or a graph of size $O(p(n)^4)$. A more formal simulation of a $p(n)$-space bounded Turing machine is given below.

Suppose machine M has a space bound of $p(n)$ and has s states and t tape symbols. Suppose an input word w of length n is given. Define an array of $2p(n)+n$ squares, each of which can hold a value from 1 to $(s+1)t$. The value in a square encodes the tape symbol in that cell and either the fact that the head does not point to that cell, or the state of the machine if the head does point to that cell. Each square can be represented by $r = \lceil \log_2((s+1)t) \rceil$ logic lines, which are initialized to the machine's initial configuration reading w with $p(n)$ cells of blank tape on each side. The value of one of these logic lines after the machine makes a single move is a boolean function of the $3r$

inputs which give the value of this square and the two adjacent ones, so a gate which computes this function can be built for each logic line. If the output from each gate is applied to the corresponding logic line, we have a simulation which will run exactly like the Turing machine, making one transition at each clock tick.

Another gate can be built which takes the r lines for one tape cell and decides whether the machine is in a final state at that cell. These values can be combined by $2p(n)+n-1$ two-input *or* gates in a binary tree, yielding a final answer which is true momentarily if (and only if) the machine ever accepts.

The number of gates is $O(p(n))$ and the maximum number of inputs per gate is $3r$, which is independent of n. Thus the size of the graph to simulate M on w is $O(p(n)^2)$. The graph can be constructed using scratch space proportional to the log of its size, or to log n.

To see that the pebbling problem can be solved in polynomial space, simply note that a supposed pebbling can be checked in space equal to the size of the graph, so the problem takes linear space non-deterministically. Therefore it can be done deterministically in quadratic space.

4. Conclusion.

Theorem 1 gives an $n/\log n$ lower bound on the number of pebbles required in the black-white pebble game in the worst case. Therefore in the worst case black-white pebbling is only a constant factor more efficient than black pebbling. It is not known whether there exist classes of graphs for which black-white pebbling saves more than a constant factor. For example, the pyramid graph S_k (with $(k^2+k)/2$ vertices) requires $k+1$ pebbles in the black pebble game. Cook and Sethi [CS] prove that S_k requires $\Omega(k^{1/2})$ pebbles in the black-white pebble game, but the most efficient black-white pebbling strategy known for S_k uses $\lfloor k/2 \rfloor + 2$ pebbles.

Section 3 shows that a considerably modified version of the black pebbling problem is complete in polynomial space. It seems quite likely that the black pebbling problem is polynomial-space complete even without the introduction of or vertices and cyclic graphs, but we have so far been unable to prove this.

References.

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[C] S. Cook, "An observation on time-storage trade off," *Proc. Fifth Annual ACM Symp. on Theory of Computing* (1973), 29-33.

[CS] S. Cook and R. Sethi, "Storage requirements for deterministic polynomial time recognizable languages," *Journal Comp. and Sys. Sci.* 13 (1976), 111-123.

[HPV] J. E. Hopcroft, W. Paul, and L. Valiant, "On time versus space," *Journal ACM* 24 (1977), 332-337.

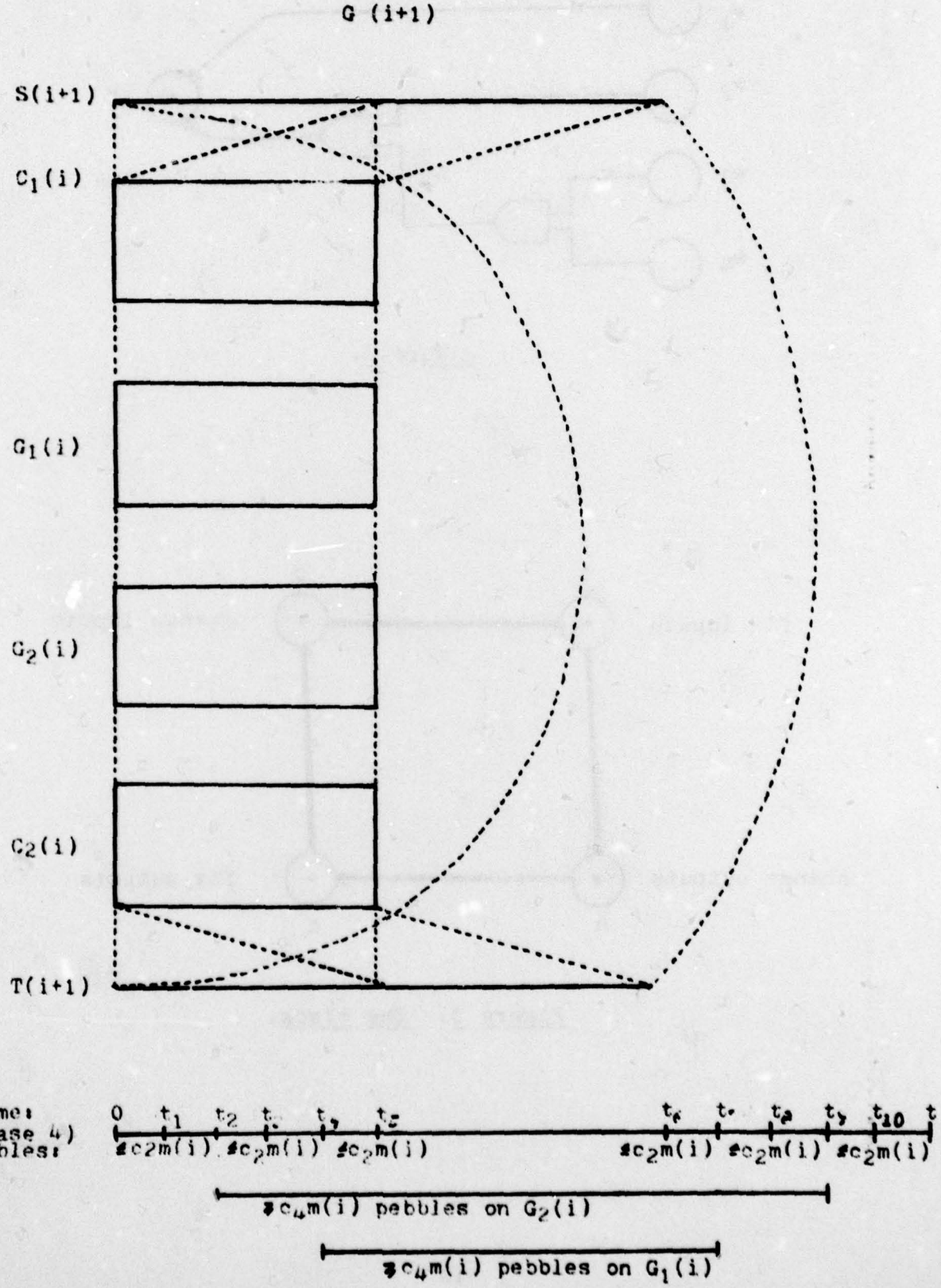
[PH] M. S. Paterson and C. E. Hewitt, "Comparative Schematology," *Record of Project MAC Conference on Concurrent Systems and Parallel Computation* (1970), 119-128.

[PT] W. Paul and R. E. Tarjan, "Time-space trade-offs in a pebble game," Stanford Computer Science Department report STAN-CS-77-619 (1977).

[PTC] W. Paul, R. E. Tarjan, and J. R. Celoni, "Space bounds for a game on graphs," *Math. Systems Theory* 10 (1976/77), 239-251.

[S] R. Sethi, "Complete register allocation problems," *SIAM J. Comput.* 4 (1975), 226-248.

Figure 1.


G (i+1)
S(i+1)
$C_1(i)$
$G_1(i)$
$G_2(i)$
$C_2(i)$
T(i+1)
time:
(case 4)
total pebbles:
0
t_1
t_2
t_{10}
t
$\leq c_2 m(i)$
$\leq c_2 m(i)$
$\leq c_2 m(i)$
$\leq c_2 m(i)$
$\leq c_2 m(i)$
$\leq c_2 m(i)$
$\leq c_4 m(i)$ pebbles on $G_2(i)$
$\leq c_4 m(i)$ pebbles on $G_1(i)$

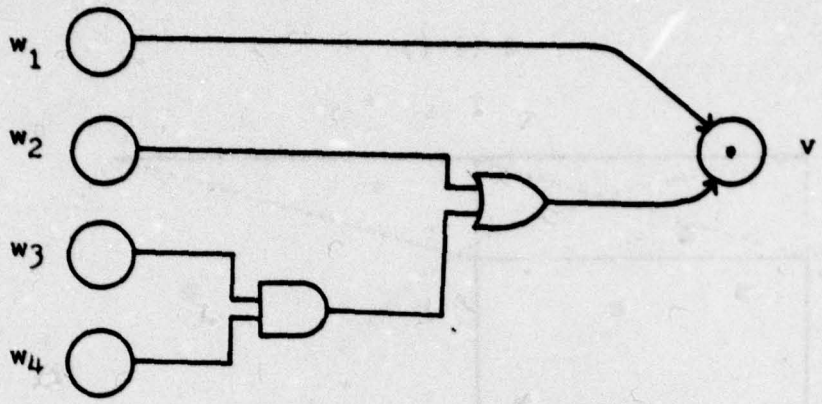


Figure 2.

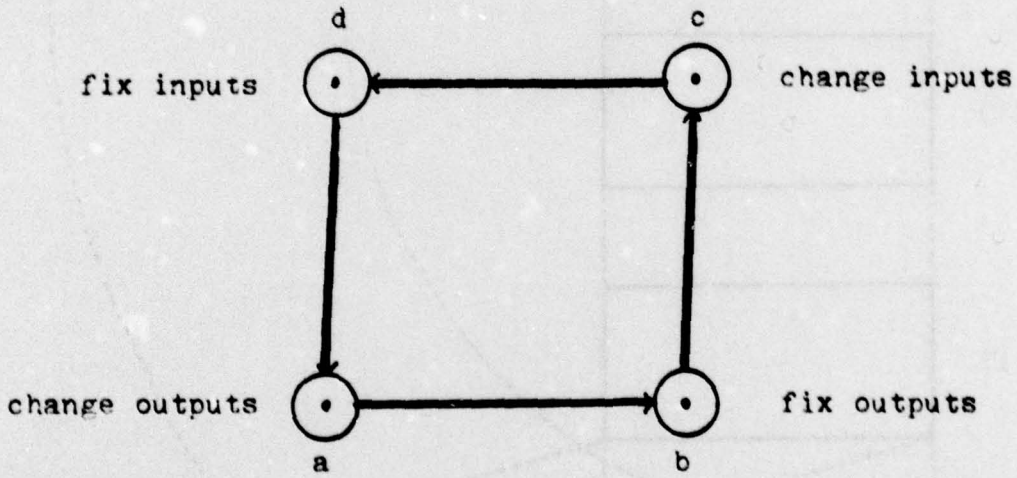


Figure 3. The clock.

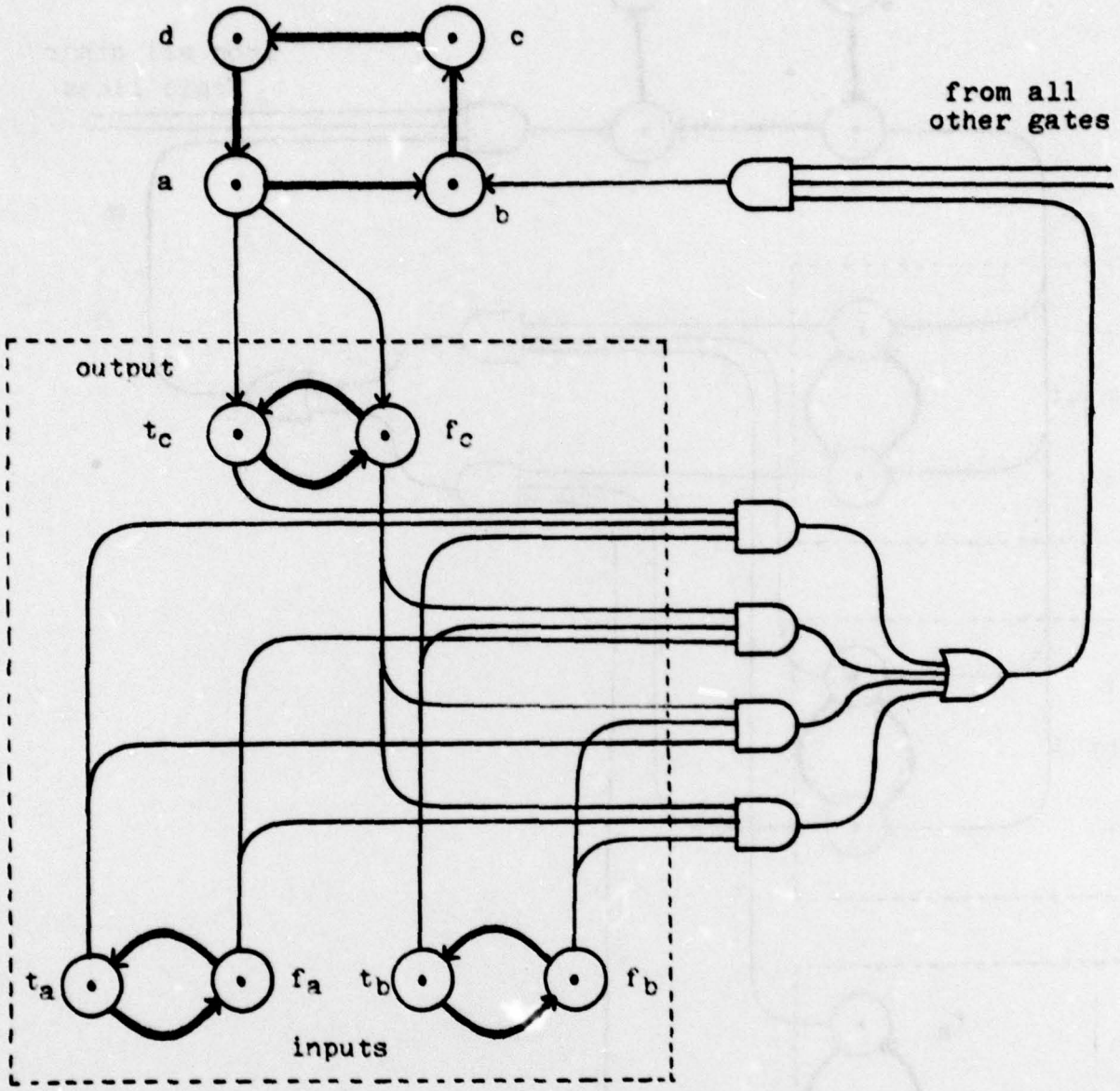


Figure 4. and gate.

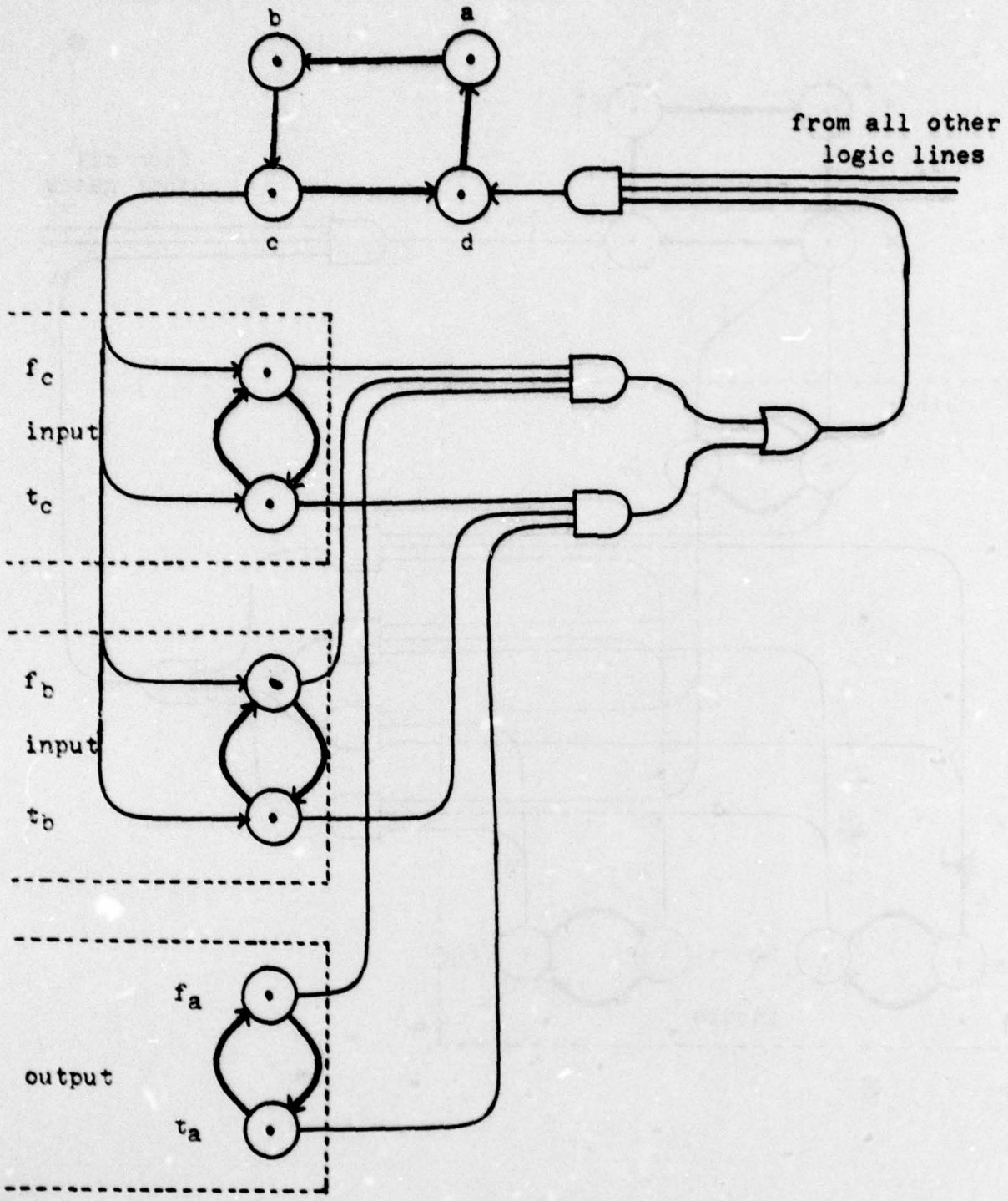
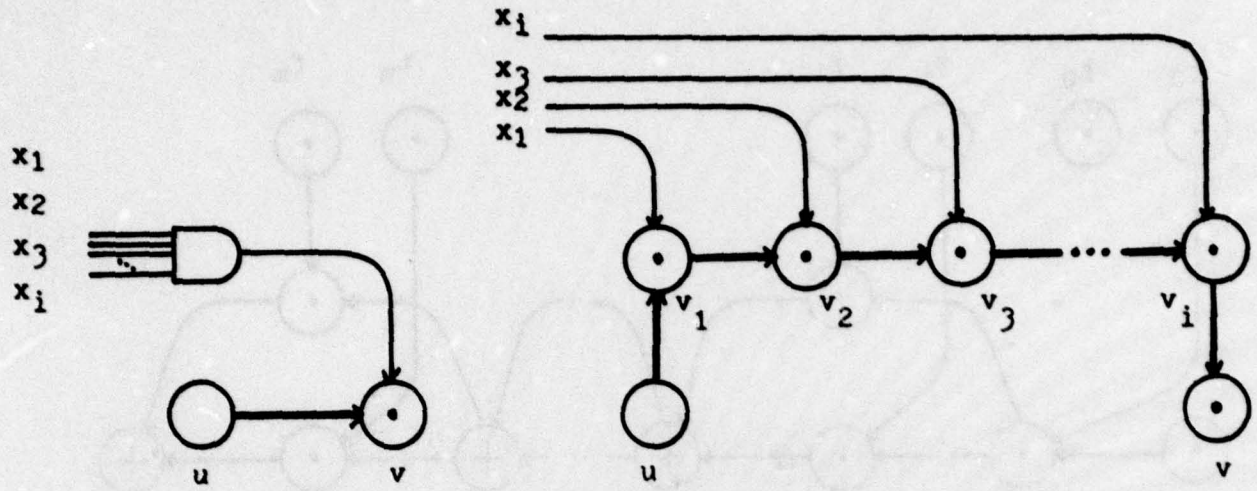
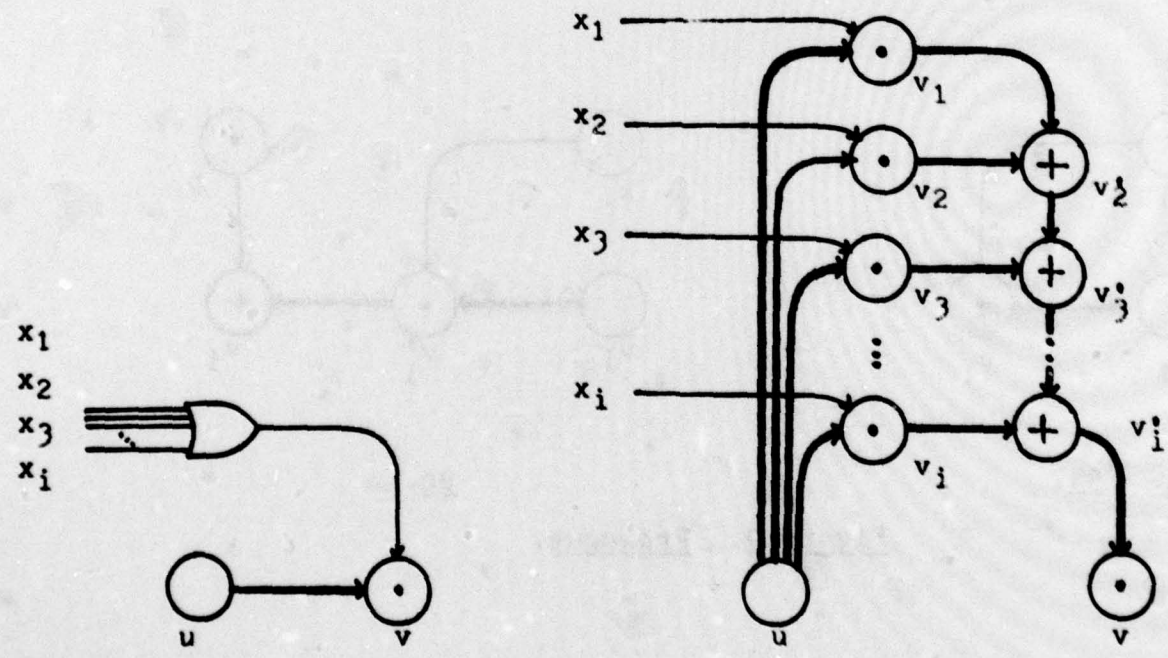


Figure 5. Logic line.



6a. 6b.

Figure 6. "and" edge replacement.



7a. 7b.

Figure 7. "or" edge replacement.

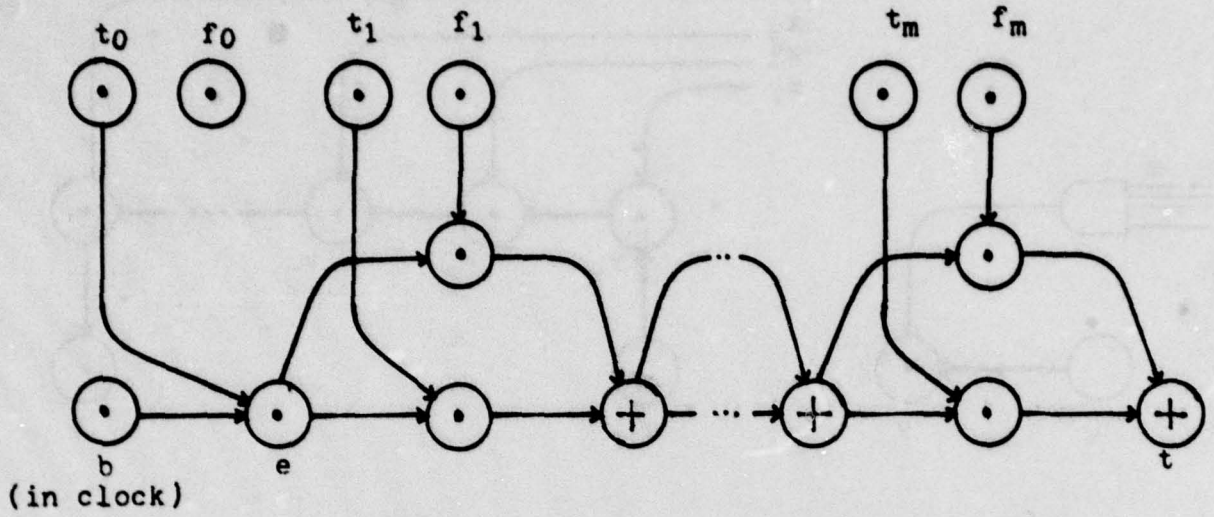
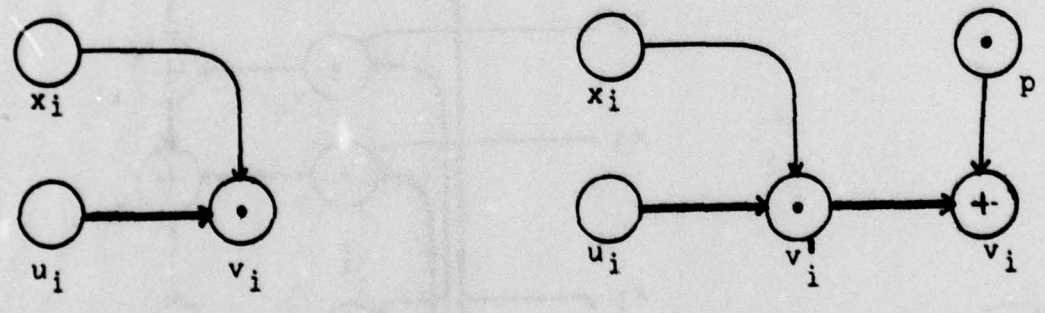


Figure 8. Epilogue.



9a. 9b.

Figure 9. Prologue.

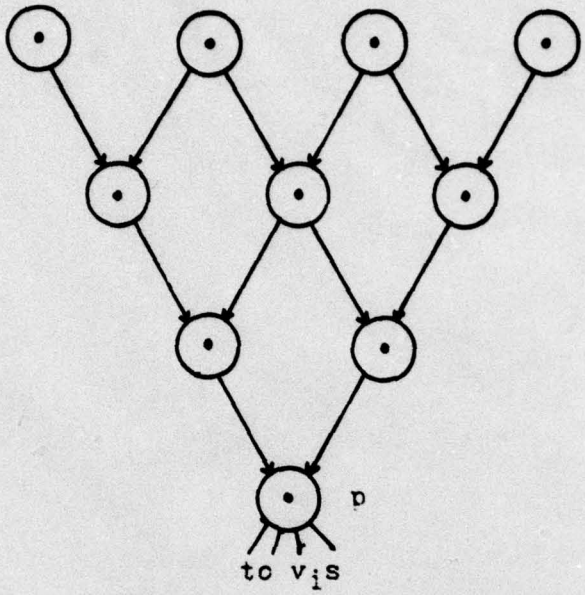


Figure 10. S_4.

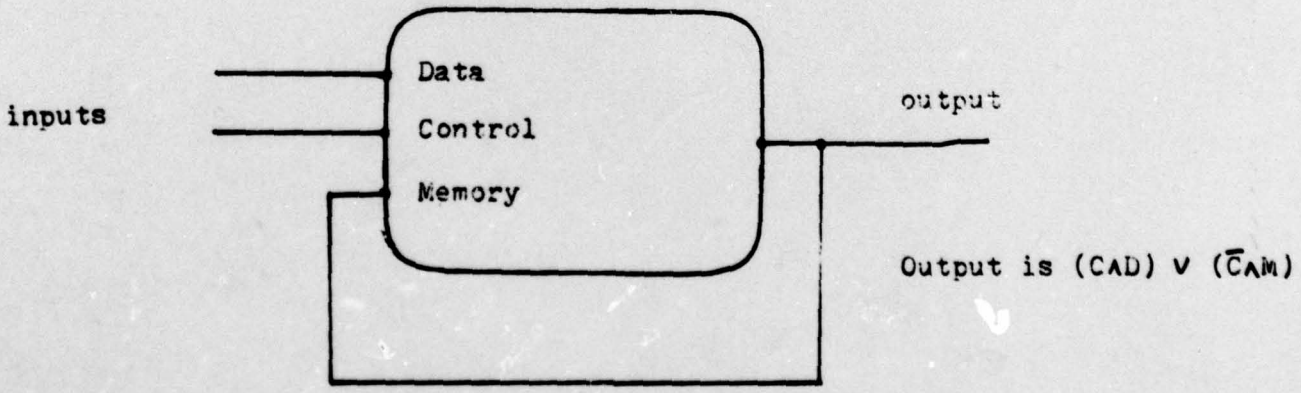


Figure 11. A one-bit memory.