Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 13, No. 1, August 1976
Printed in Belgium

Storage Requirements for Deterministic Polynomial Time Recognizable Languages

STEPHEN COOK

University of Toronto, Toronto, Ontario, Canada

AND

RAVI SETHI

*The Pennsylvania State University, University Park, Pennsylvania 16802**

Received February 14, 1975; revised November 18, 1975

An intriguing question is whether $(\log n)^2$ space is enough to recognize the class $\mathcal{P}$ of languages recognizable in deterministic polynomial time. This question has earlier been narrowed down to the storage required to recognize a particular language called *SP*. *SP* is clearly in $\mathcal{P}$ and it has been shown that if SP has tape complexity $(\log n)^k$, then every member of $\mathcal{P}$ has tape complexity $(\log n)^k$. This paper presents further evidence in support of the conjecture that *SP* cannot be recognized using storage $(\log n)^k$ for any k. We have no techniques at present for proving such a statement for Turing machines in general; we prove the result for a suitably restricted device.

1. INTRODUCTION

A striking example of practical tradeoffs between storage space and execution time is provided by the IBM 1401 Fortran compiler. Faced with a limited memory the compiler consists of 63 distinct sequential phases for analyzing the source program [3].

On a more formal level are questions about the relationship between languages accepted by time-bounded and space-bounded Turing machines, as in [1]. There is an interesting relation between the time and storage required to recognize context-free languages. The recognition algorithm in [9] requires time no more than $O(n^3)$, but requires at least linear storage, whereas the algorithm in [5] requires recognition space no more than $O((\log n)^2)$ and requires more than polynomial time. An intriguing question is whether $(\log n)^2$ space is enough to recognize all languages recognizable in deterministic polynomial time, i.e., the class of languages that has come to be called $\mathcal{P}$. Such speculation is based on attempts to extend the methods in [5] to recognize languages in $\mathcal{P}$.

* Present address: Bell Laboratories, Murray Hill, N.J. 07974.

The above question has been narrowed down in [2] to the storage required to recognize a particular language called SP. SP is clearly in $\mathscr{P}$, and it has been shown that if SP can be recognized using $(\log n)^k$ storage, for any k, then every language in $\mathscr{P}$ can be recognized using $(\log n)^k$ storage. Other languages like SP have been considered in [4].

It was conjectured in [2] that SP *cannot* be recognized using $(\log n)^k$ storage even if the device used is nondeterministic. (Context-free languages are recognized deterministically in [5].) In support of the conjecture a simple marking machine M_0 was considered. M_0 required $O(n^{1/2})$ markers to accept the set of strings SP. The machine M_0 can be viewed as playing a game similar to the ones described in [6] to carry out a computation on program schema, and in [7, 8] to determine a register allocation for a straight-line program.

This paper presents further evidence in support of the conjecture that SP cannot be recognized using storage $(\log n)^k$ for any k. In Section 2 we consider a game on directed acyclic graphs (dags) and show that at least $O(n^{1/4})$ markers are needed to play the game on some n node dags. The $O(n^{1/4})$ bound is used in Section 3 to show that a fairly general machine to recognize SP also requires $O(n^{1/4})$ storage.

2. A Game on Dags

We are given a directed acyclic graph (*dag*) (N, E), where N is a set of nodes and E is a set of edges. Dags will be depicted as in Fig. 1 with the edges assumed to be pointing downward. Nodes with no edges entering them are called *roots* and nodes with no edges leaving them are called *leaves* or *terminals*.

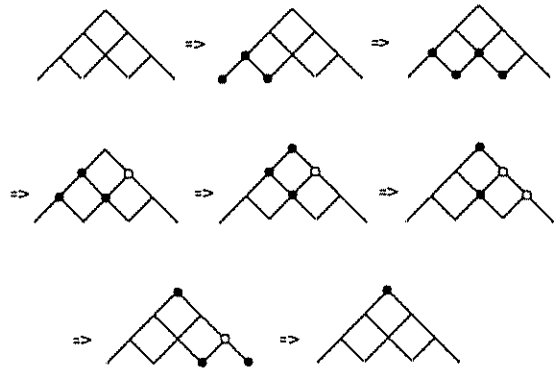

FIG. 1. A computation using four markers. For a black marker to be placed on a node there must be markers on all its direct descendants. For a white marker to be picked up from a node there must be markers on all its direct descendants.

The game we will describe is an extension of those in [2, 6, 7, 8]. In [7, 8] for example, there is an unlimited supply of *black markers*. Initially, there are no markers on the dag. In one move, a marker can be placed on a leaf. If there are markers on all direct descendants of a node x, then a marker can be placed on x. The object of the game is to place a marker on one of the roots of the dag.

Suppose placing a marker on a node is viewed as "proving" a node. Then the rule governing the placement of a marker on a nonleaf x can be thought of as "x can be proved once all direct descendants of x have been proved." Suppose that in addition to the black markers mentioned above, we also had an unlimited supply of *white markers*. Then if y, a direct descendant of x, had not yet been proved we could place a white marker on y to "remember" that at a later date, we had to prove y. The rule for placing a black marker on x would then become "if there is a black marker or a white marker on each direct descendant of a node x, then a black marker can be placed on x."

The progress of the game will be described by listing nodes that have markers on them. A *configuration* of a dag (N, E) is a pair (W, B), where $W, B \subseteq N$ and $W \cap B = \phi$.

A *move* of the game is specified by relating configurations. $(W, B) \Rightarrow_k (W', B')$ read "(W, B) *directly derives* (W', B') *using* k *markers*" if and only if

(i) $\forall w \in W$ w is a terminal or $w \in W'$ or all direct descendants of w are in $W \cup B$,

(ii) $\forall b \in B'$ b is a terminal or $b \in B$ or all direct descendants of b are in $W' \cup B'$,

and

(iii) $|W| + |B| \leqslant k$, $|W'| + |B'| \leqslant k$.

A computation is a sequence of configurations: (W, B) *derives* or *computes* (W', B') *using* k *markers*, written $(W, B) \overset{*}{\Rightarrow}_k (W', B')$ if and only if there exists a sequence of configurations

$$(W, B) = (W_0, B_0), (W_1, B_1), \ldots, (W_n, B_n) = (W', B')$$

such that $\forall i$, $1 \leqslant i \leqslant n$, $(W_{i-1}, B_{i-1}) \Rightarrow_k (W_i, B_i)$.

The computation *begins* with (W, B),
ends with (W', B'),
uses k markers,
has *length* n.

When not interested in the number of markers used we will drop k in writing $\Rightarrow_k$ or $\overset{*}{\Rightarrow}_k$.

Figure 1 gives an example of a computation that ends with a black marker on the root of the dag and uses four markers.

Blacks go up and whites go down, but the rules governing them are duals. The next lemma is an immediate consequence of the definition of $\Rightarrow$ and states that if blacks and whites are interchanged a computation can be made to run backward.

LEMMA 1. $(W, B) \overset{*}{\Rightarrow}_k (W', B')$ *if and only if* $(B', W') \overset{*}{\Rightarrow}_k (B, W)$.

Proof. From symmetry, we need only show that if

$$(W, B) \overset{*}{\Rightarrow}_k (W', B') \qquad \text{then} \qquad (B', W') \overset{*}{\Rightarrow}_k (B, W).$$

From the definition of *move* it follows that if $(W, B) \Rightarrow_k (W', B')$, then $(B', W') \Rightarrow_k (B, W)$. A simple induction establishes the result. ∎

Figure 1 contains a very special kind of dag, and it is for just such a dag that the computations we will consider use a lot of markers. A *pyramid* S_m is a dag with $m \cdot (m+1)/2$ nodes. The nodes are divided into m levels with i nodes at level i, i.e., levels go top down. Node j at level i is a direct ancestor of nodes j and $j+1$ at level $i+1$, for $1 \leqslant i < m$. The node at level 1 is the only root of the pyramid and is called r.

The computations on the pyramid that we will be interested in start with no markers on the pyramid, end with a black marker on the root, and use k markers. Such computations will be referred to as *α-computations*. The dual problem is to start with a white marker on the root and end with no markers on the pyramid.

$$\alpha\colon (\phi, \phi) \overset{*}{\Rightarrow}_k (\phi, \{r\}),$$

$$\bar{\alpha}\colon (\{r\}, \phi) \overset{*}{\Rightarrow}_k (\phi, \phi).$$

We define $\alpha(k)$ and $\bar{\alpha}(k)$ to be the largest value of m such that there is an α-computation or $\bar{\alpha}$-computation, respectively, of a pyramid S_m using k markers. From Lemma 1 it follows that $\alpha(k) = \bar{\alpha}(k)$.

In the sequel we will need to relate a node x in a dag to nodes that may be some distance from x, but "cover" it. A node x is *covered* by a set Y, denoted $x < Y$, if all paths from x to a terminal node pass through an element of Y. A set X is *covered* by a set Y if $\forall x \in X$, $x < Y$. (See Fig. 2.)

In showing that $\alpha(k) < 2k^2$, it will be convenient to consider computations that start with a certain number of black markers or end with white markers on the

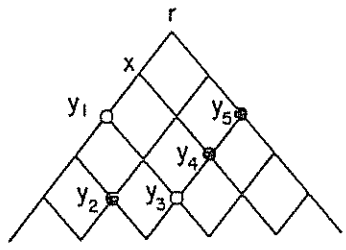

FIG. 2. A configuration of a dag. Node x is covered by the set y_1, y_2, y_3, y_4. Node r is covered by $\{y_1, y_2, \ldots, y_5\}$.

pyramid: both gifts. We are thus considering computations of the type $(\phi, B) \overset{*}{\vdash}_k (\phi, \{r\})$ or $(\phi, \phi) \overset{*}{\vdash}_k (W', \{r\})$. There is one point, however, that has not yet been mentioned. It is easy to see that if $r < B$ or if $r < W'$, then the number of levels in the pyramid under question does not come into the picture. We will therefore insist that $r < B$ and $r < W'$ be false with B and W' as above.

Let S_m be a pyramid with root r. Let $(W, B) \overset{*}{\vdash}_k (W', B')$ be a computation. A path from r to a terminal is an *open path* at (W', B') for the computation, if there is no node from $B \cup W'$ on the path.

We can now define δ *and* $\bar{\delta}$ *computations*, which relate configurations as below and have the added restriction that the computation end with a configuration that has an open path.

$$\delta\colon (\phi, B) \overset{*}{\vdash}_k (W', \{r\}),$$

$$\bar{\delta}\colon (\{r\}, B) \overset{*}{\vdash}_k (W', \phi).$$

Note that δ and $\bar{\delta}$ are dual computations. δ-computations end with a black marker on the root, while $\bar{\delta}$-computations start with a white marker on the root. Once again $\delta(k)$ and $\bar{\delta}(k)$ give the maximum number of levels m such that a $\delta(k)$ or $\bar{\delta}(k)$ computation, respectively, exists for S_m. From Lemma 1, $\delta(k) = \bar{\delta}(k)$.

THEOREM 1. *Let r be the root of a pyramid S_m. Let (W, B) be a configuration of S_m with $r \in W$. If there exists a configuration (W', B') such that $(W, B) \overset{*}{\vdash}_k (W', B')$, and there is an open path at (W', B'), then m, the number of levels in S_m is such that $m < 2k^2$. Alternatively $\alpha(k) \leqslant \delta(k) = \bar{\delta}(k) < 2k^2$.*

COROLLARY. *If there is some computation for S_m with k markers such that the root r is marked at some point in the computation but initially and finally there are no nodes marked, then $m < 2(k+1)^2$.*

Proof of Corollary. If the root is marked with a black marker, then the computation can be turned into an α computation

$$\alpha\colon (\phi, \phi) \overset{*}{\vdash}_{k+1} (\phi, \{r\})$$

with $k+1$ markers simply by never removing the black marker from r once it is placed there. Similarly, if r is marked with a white marker, then the computation can be turned into an $\bar{\alpha}$ computation

$$(\{r\}, \phi) \overset{*}{\vdash}_{k+1} (\phi, \phi)$$

with $k+1$ markers by placing the white marker on r initially. In either case, the theorem states that $m < 2(k+1)^2$.

Proof of Theorem 1. When $k = 1$ or 2, the result is easily seen to be true, so consider $k > 2$.

Since $(W, B) \overset{*}{\Rightarrow}_k (W', B')$, there exist configurations $(W, B) = (W_0, B_0), \cdots (W_n, B_n) = (W', B')$ such that for all i, $1 \leqslant i \leqslant n$, $(W_{i-1}, B_{i-1}) \Rightarrow_k (W_i, B_i)$. For all i, $0 \leqslant i \leqslant n$, let C_i represent (W_i, B_i). For all nodes x, let $S_{m,x}$ represent the part of the pyramid below node x.

We will construct a new computation in which, if x is any node such that in C_i all markers lie on nodes in $S_{m,x}$, then the same property holds for $C_{i+1}, C_{i+2}, \cdots$.

For some i, $0 \leqslant i < n$, let C_i be the first configuration in which all markers are on $S_{m,x}$, for some x, but in C_{i+1}, all markers are not on $S_{m,x}$. Construct new configurations $D_i, D_{i+1}, \cdots, D_n$, by deleting from $C_i, C_{i+1}, \cdots, C_n$ those nodes that are not in $S_{m,x}$. By definition $C_i = D_i$. Clearly for all j, $i < j \leqslant n$, since $C_{j-1} \Rightarrow_k C_j$ it follows that $D_{j-1} \Rightarrow_k D_j$. And if $D_n = (W'', B'')$, there is an open path at (W'', B''). Moreover, $r \in W$. Thus we need consider only those computations in which, if all markers are on $S_{m,x}$ in come configuration C_i for some x and i, then for all $j > i$, all markers in C_j are on $S_{m,x}$.

When the computation starts at (W, B), there is no open path since $r \in W$. So consider the first configuration C_i at which there is an open path P. From the definition of open paths, there is no node from $B \cup W_i$ on path P. In particular there is no node from B on path P. Since P is not open in C_{i-1}, there must be a white marker on some node of P in C_{i-1}. In the open path P, let x be the node nearest the root (x may even be the root), on which there was a white marker in C_{i-1}.

Consider path P from the root down to node x. As in [2] we will look at paths that "diverge" off P. From each node z of P there is a unique path that agrees with P from the root down to node z and then diverges from P by continuing in a straight line to the terminals. An example of a path diverging off P is given in Fig. 3 for S_5.

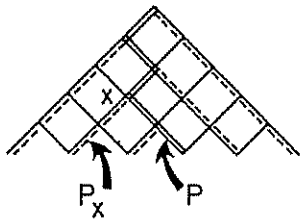

FIG. 3. Path P in S_5 is indicated by a double line. At each nonterminal node along path P, paths indicated by dotted lines diverge off P. For example, P_x diverges off P at node x.

Since all paths diverging off P were not open in C_{i-1}, on each such path there is a node either in B or in W_{i-1}. Since at most k markers are used in the computation, node x may be no more than $2k$ levels below the root.

Let us now turn to the conditions under which a white marker at x can be picked up. The first case is that x is a terminal. But then the level of x is m, given that the pyramid is S_m. Since x may not be more than $2k$ levels below the root it follows that $m \leqslant 2k$

which is less than $2k^2$ for $k > 1$. The remaining possibility is that both direct descendants of x have markers in C_{i-1}. Let y be the direct descendant of x that is on the open path P. Since y is a direct descendant of x, y may be no more than $2k + 1$ levels below the root.

Consider the part of the computation $C_0, C_1, \cdots, C_{i-1}$ that is restricted to $S_{m,y}$. Since x is not in $S_{m,y}$, in each configuration before C_i there must have been a marker not on $S_{m,y}$, so at most $k - 1$ markers may have been used on $S_{m,y}$.

The marker on node y in C_{i-1} may have been either black or white. If the marker was white then for path P to be open in C_i the marker at y can also be picked up in C_{i-1}. From the definition of $\Rightarrow$, both direct descendants of y must have markers on them in C_{i-1}. Consider the configuration C'_{i-1} formed from C_{i-1} by changing the marker at y to a black marker. From the definition of $\Rightarrow$, since there are markers on both direct descendants of y in C'_{i-1}, y is allowed to have a black marker on it. Therefore $C_{i-2} \Rightarrow C'_{i-1}$.

Since path P is open at C_i the inductive hypothesis can be applied to the computation restricted to $S_{m,y}$ and ending with a black on y, the root of $S_{m,y}$. Thus the node y is at most $2(k-1)^2$ levels from the terminals. Therefore

$$m \leqslant 2(k-1)^2 + 2k + 1 < 2k^2 \qquad \text{for} \quad k \geqslant 2. \quad \blacksquare$$

Phrased differently, Theorem 1 shows that there exists a dag with $O(m^2)$ nodes that requires $O(m^{1/2})$ markers, or $O(n)$ nodes and $O(n^{1/4})$ markers. While Theorem 1 is adequate for our purposes since it shows a lower bound that is significantly greater than $(\log n)^k$, it would be aesthetically satisfying to determine $\alpha(k)$ and $\delta(k)$. We conjecture that both functions are $O(k)$.

3. Path Machines

In [2] a certain set SP of strings was defined which has the property $SP \in \mathscr{P}$, and if SP has tape complexity $(\log n)^k$, then every member of $\mathscr{P}$ has tape complexity $(\log n)^k$. It was conjectured there that the latter is false for every k, so in fact SP requires more storage than $(\log n)^k$. We have no techniques at present for proving such a statement for Turing machines in general, and it is the purpose of this section to prove the theorem for a suitably restricted type of machine.

The set SP defined in [2] consists of all strings which code "solvable path systems." The definition below is the same as that in [2], except a restriction making the relation R acyclic has been added.

Definition. A *path system* is a quadruple $\mathscr{S} = \langle N, R, S, T \rangle$, where N is a finite set (of nodes), R is a three-place relation on N (the incidence relation), $S \subseteq N$ (S is the set of *source* nodes), and $T \subseteq N$ (T is the set of *terminal* nodes).

The *admissible* nodes of $\mathcal{S}$ are the least set A such that $T \subseteq A$ and such that if y, $z \in A$ and $R(x, y, z)$ holds, then $x \in A$. We say $\mathcal{S}$ is *solvable* iff at least one admissible node is a source node (i.e., a member of S).

The relation R must be acyclic in the following sense. There is no cycle of nodes $x_1, \cdots, x_k$ such that $x_1 = x_k$, and for all i, $1 \leqslant i < k$, there is a node y such that either $R(x_i, x_{i+1}, y)$ holds or $R(x_i, y, x_{i+1})$ holds. The acyclic restriction on R is necessary to ensure the soundness of the white–black strategy for path systems given below.

If a dag $D = (N, E)$ has either exactly zero or two edges leading out of each node, then D gives rise in a natural manner to a path system $\mathcal{S}(D) = \langle N, R, S, T \rangle$ as follows. N(nodes of D) $= N$ (nodes of $\mathcal{S}(D)$), $R(x, y, z)$ holds iff there are edges leading from x to y and z, S is the set of roots of D, and T is the set of leaves of D. All nodes on such a system are admissible, and the path system is solvable. If, however, the dag D is disconnected, and we let T be the leaves on one component and S the roots of another component, then only those nodes which lie in the same component as T will be admissible, and the path system will not be solvable.

Not all path systems arise naturally from dags, however, because we might have $R(x, y, z)$ and $R(x, y, u)$ true, but $R(x, u, z)$ false. Thus the relation R contains more information than can be simply represented with directed arcs.

Notice that the white–black game described for dags in the previous section can easily be adapted to form a nondeterministic strategy for establishing that an arbitrary path system is solvable. Thus we have the following definitions for a path system $\mathcal{S}$.

A *white–black configuration for* $\mathcal{S}$ (or simply *configuration*) is a pair (W, B) of sets of nodes of $\mathcal{S}$. We say $(W, B) \rightarrow_k (W', B')$ if

(i) $\forall w \in W$, $w \in T$ or $w \in W'$ or $R(w, y, z)$ holds for some y, $z \in W \cup B$;

(ii) $\forall b \in B'$, $b \in T$ or $b \in B$ or $R(b, y, z)$ holds for some y, $z \in W' \cup B'$; and

(iii) $|W| + |B| \leqslant k$, $|W'| + |B'| \leqslant k$.

The relation $\overset{*}{\rightarrow}_k$ is the reflexive, transitive closure of $\rightarrow_k$, just as for dags. It is not hard to see that a path system $\mathcal{S}$ is solvable iff $(\phi, \phi) \overset{*}{\rightarrow}_k (\phi, \{r\})$ for some $r \in S$ and k. By duality, $\mathcal{S}$ is solvable iff $(\{r\}, \phi) \overset{*}{\rightarrow}_k (\phi, \phi)$ for some $r \in S$ and some k. Further, it is clear that the lower bounds for k in terms of m derived there for S_m also apply to the path systems associated with S_m. That is, it follows from Theorem 1 that if $(\{r\}, \phi) \overset{*}{\rightarrow}_k (\phi, \phi)$ for a path system $\mathcal{S}(S_m)$, then $k > (\frac{1}{3}m)^{1/2}$.

The lower bound on k, the number of markers, represents a lower bound on storage for any Turing machine using the white–black method for determining whether a given path system is solvable. More precisely, any Turing machine using the white–black method to accept the set SP would require storage at least $cn^{1/4}$ for some $c > 0$.

Our purpose now is to define a class of machines which is general enough that any method similar to the white–black method for determining solvability of path systems can be realized on a machine in the class, but special enough so that a lower bound on

storage can still be proved. It seems that the difficulty in proving lower bounds for arbitrary Turing machines is that the latter have the capacity to do tricks from number theory. So we want to avoid any ability to count, or systematically search through all the nodes of a given input path system. Our new kind of machine will be a nondeterministic device with a limited number of markers which can be placed on arbitrary nodes of the input. The strategy used and the criteria for accepting the input will be established by an arbitrary finite state control. Thus most of our path machines will not work, in the sense that they will accept unsolvable path systems. But we will use Theorem 1 to prove a lower bound on the number of markers required for any such machine which does work. There are machines in the class which use the white-black method. And presumably if someone invents a similar strategy using white, black, and red markers, some machine in the class will be able to use that too.

DEFINITION. A *path machine* $\langle Q, M, \delta, q_0, q_a \rangle$ consists of a finite set Q of *states*, a finite set $M = \{m_1, \cdots, m_k\}$ of markers, a (nondeterministic) transition function δ which maps $\mathscr{D}$ into subsets of $Q \times M$, where $\mathscr{D}$ is the set of *displays* defined below, and initial and accepting states $q_0, q_a \in Q$.

An instantaneous description (ID) of a path machine $Z = \langle Q, M, \delta, q_0, q_a \rangle$ with an input path system $\mathscr{S} = \langle N, R, S, T \rangle$ is a pair $\langle q, \psi \rangle$, where $q \in Q$ and $\psi: M \to N \cup \{n_0\}$. Thus if m_i is a marker then $\psi(m_i)$ is the node currently marked by m_i, and this will be a node of the path system $\mathscr{S}$. The symbol n_0 represents a "dummy node" not in N. The *display* $D(I)$ of an ID, $I = \langle q, \psi \rangle$ consists of a quintuple $\langle q, E_I, R_I, S_I, T_I \rangle$, where E_I is the equivalence relation on the markers M defined by $m_i E_I m_j$ iff $\psi(m_i) = \psi(m_j)$, $R_I = \psi^{-1}(R)$ (i.e., R_I is a three-place relation on M defined by $R_I(m_i, m_j, m_k)$ iff $R(\psi(m_i), \psi(m_j), \psi(m_k))$), $S_I = \psi^{-1}(S)$, and $T_I = \psi^{-1}(T)$. Thus the display of an instantaneous description tells the path machine everything there is to know about the currently marked nodes of the input path system. That is, it tells which pairs of markers are on the same node, which triples of marked nodes satisfy the incidence relation R, which marked nodes are source nodes, and which marked nodes are terminal nodes.

Suppose the path machine Z currently has an instantaneous description $I = \langle q, \psi \rangle$ (Z is in state q with markers on the nodes given by ψ). The possible next moves depend only on the display $D(I)$, and are determined by the transition function δ. A move consists simply of placing the ith marker (for some particular i) on an arbitrary new node, and assuming a new state. Precisely, we write $I \to I'$, where $I' = \langle q', \psi' \rangle$, if and only if for some marker m_i, $1 \leqslant i \leqslant k$, $(q', m_i) \in \delta(D(I))$, and $\psi'(m_j) = \psi(m_j)$ for all $j \neq i$.

A *computation* of Z with input $\mathscr{S}$ is a sequence $I_0, I_1, \cdots, I_t$ of ID's such that $I_i \to I_{i+1}$ for each $i \geqslant 0$, and $I_0 = \langle q_0, \psi_0 \rangle$, where q_0 is the initial state and $\psi_0(m_j) = n_0$ for $1 \leqslant j \leqslant k$ (recall n_0 is the "dummy node"). The computation is *accepting* iff the final state is the accepting state q_a. We say Z *accepts* $\mathscr{S}$ iff there is some accepting computation.

We would like to study only those path machines which accept precisely the solvable path systems, but no one path machine can do this because it only has a fixed number k of markers. Thus we say a path machine Z is *sound* iff Z accepts only solvable path systems. We will be interested in finding the largest n so some sound path machine with k markers accepts all solvable path systems with n or fewer nodes.

The first thing to observe is that our notion of path machine is general enough to code the white–black method. Thus a sound path machine with k markers can accept every pyramid path system $\mathcal{S}(S_m)$ for each $m \leqslant k - 1$. The idea is that the machine would have enough states to keep track of which markers were currently white and which were black, and the state transition function would not classify a marker as black unless it were placed immediately "above" two marked nodes (or on a node in T) and it would not allow removing any white marker unless there were two marked nodes immediately below it (or it was on a node in T). Finally, the machine would accept the input if and only if at some point in the computation a source node was marked, and at the end of the computation no white markers remained (except possibly on the dummy node n_0).

We now show how to apply Theorem 1 to derive a lower bound on the number of markers (and hence the storage) required by sound path machines to accept solvable path systems with a given number of nodes.

THEOREM 2. *No sound path machine with k markers accepts any pyramid $\mathcal{S}(S_m)$ if $m \geqslant 2(k+1)^2$.*

COROLLARY. *If a sound path machine with k markers accepts all solvable path systems with n or fewer nodes, then $k > \frac{1}{2}(n)^{1/4} - 1$.*

Proof of Theorem 2. Suppose a path machine Z with k markers accepts $\mathcal{S}(S_m)$, and suppose $m \geqslant 2(k+1)^2$. We construct a second path system S'' which looks locally like $\mathcal{S}(S_m)$, but which is not solvable.

Let $\mathcal{S} = \langle N, R, S, T\rangle$ and $\mathcal{S}' = \langle N', R', S', T'\rangle$ be two disjoint isomorphic copies of the path system $\mathcal{S}(S_m)$, and let $\gamma: N \to N'$ be an isomorphism between $\mathcal{S}$ and $\mathcal{S}'$. We construct another path system $\mathcal{S}'' = \langle N'', R'', S'', T''\rangle$ as follows. $N'' = N \cup N'$, $R''(x, y, z)$ holds iff $R'(\gamma'(x), \gamma'(y), \gamma'(z))$ holds, except $R''(x, y, z)$ is always false if $x \in N'$ and both $y, z \in N$, where γ' is the extension of γ to $N \cup N'$ by setting $\gamma'(x) = x$ for $x \in N'$. (In other words, if $x, y, z \in N$, $x', y', z' \in N'$, $\gamma(x) = x'$, $\gamma(y) = y'$, $\gamma(z) = z'$, and $R(x, y, z)$ and $R'(x', y', z')$ hold, then $R''(x, y, z)$, $R''(x, y', z)$, $R''(x, y', z')$, $R''(x, y, z')$, $R''(x', y, z')$, $R''(x', y', z)$, $R''(x', y', z')$ all hold, but $R''(x', y, z)$ is false.) We set $S'' = S'$, and $T'' = T$. Then the admissible nodes of $\mathcal{S}''$ comprise precisely the set N, and so $\mathcal{S}''$ is not solvable.

Now let $C = (I_0, I_1, \ldots, I_t)$ be an accepting computation of Z with input $\mathcal{S}$. We will construct from C a computation $C'' = (I_0'', \ldots, I_t'')$ of Z with input $\mathcal{S}''$, such that

$D(I_i) = D(I''_i)$, $0 \leqslant i \leqslant t$. Since the two computations have identical displays, C'' will also be accepting, and since $\mathcal{S}''$ is not solvable it follows that Z is not sound. The idea is that each time a marker is placed on a node x during the computation C, we place the marker on either x or $\gamma(x)$ for the corresponding step in the computation C'', according to a certain rule. We will give a recursive definition of a set F of ordered pairs of numbers, and then if the step $I_{i-1} \rightarrow I_i$ of the computation C involves placing a marker m_j on a node x, the corresponding step $I''_{i-1} \rightarrow I''_i$ of C'' will involve placing m_j on x if $(i, j) \in F$, and placing m_j on $\gamma(x)$ otherwise. F is defined in such a way that the markers placed on N by this rule will provide a white–black computation for $\mathcal{S}$. The idea is that if C places a marker on x, then C'' will place the marker on $\gamma(x)$, unless C is forced to place it on x in order for the displays of C and C'' to be identical.

Basis. $(i, j) \in F$ if step $I_{i-1} \rightarrow I_i$ involves placing m_j on a member of T.

Recursion. If $(i_1, j_1) \in F$ and $(i_2, j_2) \in F$, and if the step $I_{i_l-1} \rightarrow I_{i_l}$ involves placing marker m_{j_l} on node x_l, $l = 1, 2, 3$, and if there exists a time i at which all three nodes x_1, x_2, x_3 had markers simultaneously in C, and for each l, x_l was marked continuously between step i and step i_l (or between step i_l and step i), and $R(x_3, x_1, x_2)$ holds, then $(i_3, j_3) \in F$.

The above description completely specifies the sequence of moves for C''. To complete the definition of C'', we need only say that the state sequence of C'' is identical to that of C. Thus if $C = (I_0, \ldots, I_t) = (\langle q_0, \psi_0 \rangle, \ldots, \langle q_t, \psi_t \rangle)$, then $C'' = (\langle q_0, \psi''_0 \rangle, \ldots, \langle q_t, \psi''_t \rangle)$, where the ψ''_i are the marker positions defined by the moves described above.

It is clear (by induction on i) that the marker positions ψ_i at step i, $0 \leqslant i \leqslant t$, of the computation of C are the same, up to a possible application of γ, as the marker positions ψ''_i at step i of C''. In other words, $\gamma \circ \psi_i = \gamma \circ \psi''_i$ (where we define $\gamma(n_0) = n_0$). Lemma 1 below makes a further claim.

LEMMA 1. *For each $x \in N$, x and $\gamma(x)$ are never marked simultaneously during C''. In other words, we cannot have $\psi''_i(m_j) = x$ and $\psi''_i(m_k) = \gamma(x)$ for $x \in N$, $0 \leqslant i \leqslant t$, and any distinct markers m_j, m_k.*

Proof. If step l requires placing m_j on x (so (l, j) is in F) then no marker m_k could be on $\gamma(x)$, since the criterion used in the definition of F to place m_j on x would also place m_k on x. Similarly the definition of F would never place a marker on $\gamma(x)$ if x is currently marked.

It remains to show that C'' is a valid computation.

LEMMA 2. *C'' is a valid computation for the path machine Z with input $\mathcal{S}''$, and the display sequences of C and C'' are identical.*

Proof. It is sufficient to show the two display sequences are identical, since by assumption C is a valid computation for Z. Recall that the display $D(I)$ of an ID

$I = \langle q, \psi \rangle$ is a quintuple $\langle q, E_I, R_I, S_I, T_I \rangle$. We know the state components q of the corresponding displays for C and C'' are the same by definition. The equivalence relation E_I is the same for both computations by Lemma 1 and the remark preceding it. To see that the relation R_I is the same for both computations, note that by definition of R'', the only way R_I can differ for the two computations is for some three markers to occur on nodes x, y, and z, respectively, at some time i in the computation C, where $R(x, y, z)$ holds, but for the same markers to occur on nodes $\gamma(x)$, y, and z at the corresponding time i in C'' (note that $R''(\gamma(x), y, z)$ fails to hold). But this cannot happen, because this time i will satisfy the condition for i in the recursion step of the definition of F, so the marker for $\gamma(x)$ should have been placed on x instead of $\gamma(x)$.

The relation T_I is the same for both computations by the basis step in the definition of F.

To see that S_I is the same for both computations is more difficult. In fact, we will show that whenever a marker is placed on the source node r of $\mathcal{S}$ during the computation C, the corresponding move in C'' places a marker on $\gamma(r)$. This will suffice, since $S'' = S'$. More exactly, we will show that there is a white–black computation C_{WB} for $\mathcal{S}$ in the sense of the previous section which uses at most k markers, and such that for every marker placed on a node of N during C'', some marker is placed on the same node during C_{WB}. Furthermore, C_{WB} starts and ends with no markers on N. Since $m \geqslant 2(k+1)^2$, it follows from the corollary to Theorem 1 that the source node r is never marked during C_{WB}, and hence r is never marked during C''.

The computation C_{WB} is defined from C'' as follows. We will maintain an induction hypothesis stating that every marked node of N during C'' is marked with exactly one marker at the corresponding steps in C_{WB}. Initially there are no markers on N for C_{WB}. In general, suppose a given step of C'' involves moving marker m_j from node x to node y in $N \cup N' \cup \{n_0\}$. This results in two corresponding moves σ_1 and σ_2 in C_{WB} (and possibly more), either or both of which may be vacuous. If $x \in N' \cup \{n_0\}$, or if x has more than one marker on it before the move of C'', then σ_1 is vacuous. If $x \in N$ and x has a single marker on it before the move in C'', then (by the induction hypothesis) x will have a single marker on it for C_{WB}, and σ_1 consists of removing that marker. We argue below that that marker is black. If $y \in N' \cup \{n_0\}$ or if y has two or more markers after the move of C'' is executed, then σ_2 is vacuous. Otherwise σ_2 consists of placing a marker on y. This marker is black if y is terminal or if both immediate descendants of y are already marked in C_{WB}, and white otherwise. Now if placing this marker on y allows some white marker on some square z to turn black (because now both direct descendants of z are marked), then the following two additional moves are made in C_{WB}: the white marker on z is removed, and it is replaced with a black marker. These two moves are repeated if there is more than one such z.

Finally, after all moves of C'' have been completed, C_{WB} is terminated by removing all remaining markers (which will be black by the argument below).

It remains to argue that no white marker is ever removed during C_{WB}, except by

the "two additional moves" construction above which turns a white marker into a black. This is because no marker is ever placed on a node x during C_{WB} unless a marker is placed on x during the corresponding step of C''. By definition of the set F, this will not happen unless either x is terminal or there is a time i at which x and its two immediate descendants are marked in the computation C. By Lemma 1, x and its two immediate descendants in N are also marked at time i in the computation C'', and hence they are marked at the corresponding time in C_{WB}. At this time (or before) the "two additional moves" construction in the definition of C_{WB} will cause the white marker on x to be replaced by a black marker, if it is not already black. Hence all white markers turn black before being removed in C_{WB}, so C_{WB} is a valid white–black computation. This completes the proof of Lemma 2 and Theorem 2.

ACKNOWLEDGMENT

The authors are grateful to John Bruno for helpful discussions.

REFERENCES

1. R. V. BOOK, Comparing complexity classes, *J. Comput. System Sci.* 9 (1974), 213–229.
2. S. A. COOK, An observation on time-storage tradeoff, *J. Comput. System Sci.* 9 (1974), 308–316.
3. L. H. HAINES, Serial compilation and the 1401 Fortran compiler, *IBM Systems J.* 4 (1965), 73–80.
4. N. D. JONES AND W. T. LAASER, Complete problems for determinitic polynomial time, *in* "Proceedings of the Sixth Annual ACM Symposium on Theory of Computing" (May 1974), pp. 40–46.
5. P. M. LEWIS, R. E. STEARNS, AND J. HARTMANIS, Memory bounds for the recognition of context free and context sensitive languages, *in* "IEEE Conference Record on Switching Circuit Theory and Logical Design," (1965), pp. 191–202.
6. M. S. PATERSON AND C. E. HEWITT, Comparative schematology, *in* "Record of Project MAC Conference on Concurrent Systems and Parallel Computation" (June 1970), pp. 119–128, ACM, New Jersey (Dec. 1970).
7. R. SETHI, Complete register allocation problems, *SIAM J. Computing* 4 (1975), 226–248.
8. S. A. WALKER AND H. R. STRONG, Characterization of flowchartable recursions, *J. Comput. System Sci.* 7 (1973), 404–447.
9. D. H. YOUNGER, Recognition and parsing of context free languages in time $O(n^3)$, *Inform. Contr.* 10 (1967), 189–208.

Printed by the St Catherine Press Ltd., Tempelhof 37, Bruges, Belgium.