Correspondence

On the Modeling of Demand Paging Algorithms by Finite Automata

C. C. YANG

Abstract—Two finite automata are devised for modeling two classes of demand paging algorithms. The first one of one input and three outputs models the class of algorithms with a constant amount of allocated space. The second one of one input and six outputs models the class of algorithms with a variable amount of allocated space. Some evaluation techniques are developed following each model. The memory states of the first class algorithm with the Least Recently Used (LRU) replacement policy and the working set model of the second class are recursively defined by strings of the loaded pages. The adopted replacement policy and the state string updating procedure are imbedded in the recursive definition of memory states. Properties of some algorithms are developed to fit the finiteness assumption of a reference string.

Index Terms—Demand paging algorithm, finite automaton, Least Recently Used replacement policy, memory management, miss-

M.S. and Ph.D. degrees from the University of California, Berkeley, in 1962 and 1964, respectively.

Since 1964 he has been with the Department of Electrical Engineering, Princeton University, Princeton, N. J., and he is presently an Associate Professor. He is a consultant with the U. S. Army (Frankford Arsenal). His research interests include pattern recognition, image processing, approximation theory, and the application of mathematical techniques in the life sciences. He is the author of Biological Oscillators: Their Mathematical Analysis (New York: Academic, 1973).

Dr. Pavlidis is a member of the Association for Computing Machinery, the Pattern Recognition Society, and Sigma Xi.

Steven L. Horowitz was born in New York City, N. Y., on March 4, 1953. He received the B.S. degree in electrical engineering from Princeton University, Princeton, N. J., in 1974.

His present plans include pursuing research in picture processing while attending graduate school in Computer Science at Princeton University.

INTRODUCTION

In a virtual storage computer system [1], the memory allocation problem includes three aspects [2]–[3]: 1) page fetching or loading, 2) page placement and 3) page replacement. Placement is irrelevant in a paging environment [3]. Loading is on demand only in demand paging system. Therefore, replacement is all that is left. In addition, besides the page size and the number of allocated page frames, the replacement policy is one of the most important parameters [1] in designing a virtual storage computer system. Aho et al. [4] discovered that the minimum cost is always achieved by a demand paging algorithm under usual assumptions about memory system organization. Thus, a systematic study and a unified approach of demand paging algorithms should deserve considerable interest and attention.

Although analytic studies of the algorithms just mentioned have established some important formal models like the working set model [5], the stack algorithm [6], the independent reference model [4], the probabilistic model based on a finite-state first-order Markov chain [7] and the formal treatment of the anomalous behavior of the algorithm with the FIFO (First-In First-Out) replacement policy [8], the well established automata theory [9] did not fruitfully serve in this area until the appearance of Gelenbe’s recent article [10] in which he uses a stochastic automaton to formalize the class of the random partially preloaded algorithms.

This correspondence presents two finite automata that formalize two classes of demand paging algorithms: one involves constant allocated space like the algorithm with the LRU (Least-Recently Used) replacement policy and the other involves variable allocated
spaces such as the working set model. This note is condensed from the author's report [11] so that only important results without accompanying the detailed verifications are included here.

BACKGROUND

Let \( P \) be the set of pages of a program and \( K(P) \) the number of pages in \( P \). Let \( P^* \) denote the free monoid generated by \( P \). An element in \( P^* \) is called a reference string denoted by \( W_T \). The length of \( W_T \) is finite and is denoted by \( L_T \).

The space allocated in main memory for processing \( W_T \) is accounted by a number of page frames accepting pages of equal size. The set of loaded pages resided in the allocated space at time \( t \) constitutes the memory state or simply state denoted by \( S(t) \) at \( t \). Later on, subscripts \( n \) and \( T \) will be added to differentiate the cases of constant and variable allocated spaces. In our proposed automata models, a state is represented by a string of the loaded pages and the ordering of these pages is determined by the adopted replacement policy of a demand paging algorithm. At \( t = 0 \), the state \( S(0) \) is represented by the null string \( \Lambda \) in \( P^* \) and is referred to as the initial state. Let \( p_i \) be the current referenced page of \( W_T \) at time \( t \).

As \( p_i \) is referenced, the state \( S(t-1) \) transits to its next state \( S(t) \). The string of \( S(t) \) is obtained by updating that of \( S(t-1) \). Thus whenever a state-transition occurs, an updating operation always follows unless \( p_i \) does not affect the ordering of all loaded pages in \( S(t-1) \).

As \( p_i \) is referenced, if \( p_i \) is missing from main memory, we say that a page-fault or page-exception happens. When such a page-fault occurs, a loading or paging-in operation is initiated to move \( p_i \) from secondary storage to an allocated page frame if such a frame is available. Otherwise, a paging-out activity that removes a selected loaded page to secondary storage is needed in order to leave room in the allocated space for loading \( p_i \) into the freed page frame.

AUTOMATA MODEL A

Let \( n \) be the number of page frames allocated for processing \( W_T \) at all times, \( 0 \leq t \leq L_T \). A demand paging algorithm with constant allocated space can be conveniently formalized by the finite automaton \( A = (R, P, Q_0, h_o, S_0(0)) \) satisfying the following conditions.

Condition 1: \( R \) is the set of states \( S_0(t), 0 \leq t \leq L_T \).

Condition 2: \( P \) is the input alphabet.

Condition 3: \( Q_o = \{ q_0, q_1 \} \) is the output alphabet.

Condition 4: \( Q_o \times P \rightarrow R \) is the next state function and \( h_o : R 	imes P \rightarrow Q_o \) is the output function.

State transition and output generation are determined by \( S_0(t) = q_o(S_0(t-1), p_i) \) and \( q_1 = h_o(S_0(t-1), p_i) \) such that

a) if \( p_i \) is in \( S_0(t-1) \), then \( p_i \) is also in \( S_0(t) \) and \( q_1 = q_o \);

b) if \( p_i \) is not in \( S_0(t-1) \) and \( q_o(S_0(t-1)) \) \(< q_o(S_0(t-1)) \) \(< q_o(S_0(t-1)) \) \(< q_o(S_0(t-1)) \), then \( p_i \) is loaded into \( S_0(t) \) and \( q_1 = q_o \);

c) if \( p_i \) is not in \( S_0(t-1) \) and \( q_o(S_0(t-1)) \) \(< q_o(S_0(t-1)) \), then \( p_i \) replaces some loaded page and \( q_1 = q_o \).

Condition 5: \( S_0(0) = A \) is the initial state.

In Condition 4c), the loaded page to be selected for replacement is determined by the page-replacement rule and the tied-page breaking rule if tied pages exist. The significances of the outputs \( q_o \) through \( q_1 \) can be seen from Condition 4; namely, \( q_o \) reflects a single updating operation; \( q_1 \), the activities of loading followed by updating; and \( q_o \), the operations of paging-out followed by loading and then updating.

The algorithms with the simple replacement policies like LRU, FIFO, FILO (First-In Last-Out), RAND (RANDom), LFU (Least Frequently Used), etc., can be all formalized by this model.

For the purposes of evaluation and investigation, we need the following definitions:

The page-fault rate \( s(n) \) for processing \( W_T \) is defined to be the number of referenced pages not missing from the allocated space per reference. By the significance of an output \( q_o \), we have

\[ s(n) = \frac{\# q_o}{L_T} \quad \text{(1a)} \]

where \( \# q_o \) indicates the number of all \( q_o \) outputs. On the contrary, the page-fault rate \( f(n) \) for processing \( W_T \) is defined as the number of page-faults per reference. Since a page-fault is reflected by the generation of either one \( q_1 \) or one \( q_o \) output, this rate becomes

\[ f(n) = \frac{\# q_1 + \# q_o}{L_T} . \quad \text{(2a)} \]

The missing-page rate \( m(n) \) for processing \( W_T \) that measures the number of referenced pages missing from allocated space per reference is identical to \( f(n) \).

Since an automation discussed in this note is actually a sequential machine [9] that transforms \( W_T \) considered as an input string into an output string \( W_o \) of equal length. This length-preserving property assures that

\[ s(n) + f(n) = 1. \quad \text{(3)} \]

Furthermore, a loading string \( W_o \) can be easily obtained from \( W_T \) by deleting those pages in \( W_T \) with corresponding outputs being \( q_o \) in \( W_o \). Thus, \( f(n) \) can be alternatively represented as

\[ f(n) = \frac{L_o}{L_T} \quad \text{(2b)} \]

where \( L_o \) is the length of \( W_o \).

To show that \( s(n) \) can have another form, suppose that a page \( p_i \) in \( P \) referenced at both times \( t - x_i \) and \( t \), then \( q_o \) is said to be recurrent at \( t \) (but not recurrent at \( t - x_i \)) with the interreference interval \( x_i \). Let \( L(t,n) \) be the longest substring of \( W_T \) such that the number of distinct referenced pages over the interval \([ t - L(t,n), t ]\) equals \( L(S_0(t)) \). Let \( N(n) \) be the number of all recurrent pages \( p_i \) (not necessarily distinct) at distinct reference times \( t \) such that \( x_i \leq L(t,n) \) for all \( t \) and \( f \). As shown in [11], \( N(n) = \# q_o \). Hence, we have

\[ s(n) = \frac{N(n)}{L_T} \quad \text{(1b)} \]

Now we establish the least upper bound of \( s(n) \). First, the following statements are equivalent: 1) \( n \geq K(P) \); 2) \( \# q_o = 0 \); and 3) \( \# q_1 = K(P) \). Verification of the equivalence is shown in [11]. Second, the least upper bound is found to be

\[ s_{max} = 1 - \frac{K(P)}{L_T} \quad \text{(4a)} \]

which holds if and only if each of the \( K(P) \) pages of a program is loaded once and only once, i.e., \( \# q_1 = 0 \). Equations (3) and (4a) imply that the greatest lower bound of \( f(n) \) is

\[ f_{min} = \frac{K(P)}{L_T} \quad \text{(5a)} \]

Third, if \( K(P) \) is relatively much smaller than \( L_T \), then we have the limiting values

\[ s_{max} = 1, \quad f_{min} = 0, \quad \text{if} \quad K(P) \ll L_T \quad \text{(4b)} \]

AUTOMATA MODEL B

Let \( n(t) \) be the number of page frames allocated for processing \( W_T \) at time \( t \). Then the value of \( n(t) \) at each time \( t \) should be determined by some means that can reflect the memory demand at \( t \), i.e.,

\[ n(t) = L(g(S_T(t))) \quad \text{(6)} \]

where the subscript \( T \) is the window size [12] associated with the interval \([ t - T + 1, t ]\). Comparing the values of \( n(t) \) and \( n(t-1) \), there exist three possibilities: 1) \( n(t) = n(t-1) \); 2) \( n(t) < n(t-1) \); and 3) \( n(t) > n(t-1) \). For accommodating these possibilities, we need to have more operations besides updating, loading and paging-out. A tagging operation means that a loaded page just leaves a state without actually paging-out. A reutilizing operation means that a tagged page that is referenced returns to a state. Now a demand paging algorithm with variable allocated spaces can be formalized by \( B = (R', P, Q_0, h_o, S_T(0)) \) satisfying the following conditions.

Condition 1: \( R' \) is the set of states \( S_T(t), 0 \leq t \leq L_T \).

Condition 2: \( P \) is the input alphabet.
The significance of the new outputs, $q_i$, through $q_t$ can be easily seen from Condition 4. The page-success rate $s(T)$, the page-fault rate $f(T)$ and the missing-page rate $m(T)$ can be represented as

$$ s(T) = \frac{#q_0 + #q_1}{L_T} \quad (7a) $$

$$ f(T) = \frac{#q_1 + #q_1'}{L_T} \quad (8a) $$

$$ m(T) = \frac{#q_1 + #q_1' + #q_4 + #q_0}{L_T} \quad (9a) $$

From (8a) and (9a), $m(T)$ is obviously an upper bound of $f(T)$. By the length-preserving property of a sequential machine, we have

$$ s(T) + m(T) = 1. \quad (10) $$

Due to the additional tagging and reutilizing operations introduced in the automata model $B$, each of the $K(P)$ pages of a program is loaded once and only once, the number of (8a) always assumes the minimum value $K(P)$ and accordingly $f(T)$ is always minimized and has the value $f_{min}$ as shown in (8a), i.e.,

$$ f(T) = f_{min}. \quad (8b) $$

Since $m(T)$ is an upper bound of $f(T)$, $m(T)$ is lower bounded by $f_{min}$, i.e.,

$$ m(T) \geq f_{min}. \quad (9b) $$

It is easy to show that $s(T)$ is upper bounded by $s_{max}$, i.e.,

$$ s(T) \leq s_{max}. \quad (7b) $$

Let $N_i(T)$ be the number of all recurrent pages (not necessarily distinct) $p_i$ in $W_t$ with $1 \leq i \leq T$ for all $p_i$ referenced at distinct reference times $t_i$, $0 < t_i \leq L_T$ where $z$ denotes an interreference interval. Then the page-success rate $s(T)$ can be alternatively represented as

$$ s(T) = \frac{N_i(T)}{L_T}. \quad (7c) $$

FINE FORMALIZATION OF SOME DEMAND PAGING ALGORITHMS

Both models proposed previously seem too general to reflect the adopted replacement policy and the state-string updating procedure. This problem can be easily solved by making some refinements. First, we establish the basis of a recursive definition for determining the strings of all states, i.e.,

$$ g(\Lambda, p_t) = p_t \quad (10) $$

where $p_t$ is the first referenced page at $t = 1$ or the head of $W_t$ and the initial state $S(0)$ and its state $S(1)$ are denoted by the strings $\Lambda$ and $p_0$, respectively. At $t = 1$, the output is always $q_0$, i.e.,

$$ h(\Lambda, p_1) = q_0. \quad (11) $$

The map $g$ in (10) stands for either $g_e$ or $g_b$ and similar is the map $h$ in (11). Second, let $S(t - 1) = w(p_t)$ for $(w, p_t)$ in $P^* \times P$ for $t > 1$. Now, we are ready to perform fine formalizations. The first-class algorithms with the LRU replacement policy and the working set model of the second class are formalized in the following. The algorithms with other replacement policies can be similarly developed as shown in [11].

The LRU strategy replaces the loaded page that is the least recently referenced one by $p_t$. Each ordered-pair on the right-hand side of (12) is the string of $S_n(t)$ and the output generated at time $t$.

$$ (g_e(w(p_t), p_t), h_e(w(p_t), p_t)) $$

$$ = (\begin{cases} 
(p_e(w_1, p_1), q_1), & \text{if } p_1 = p_t; \\
(p_e(w_0, p_0), q_0), & \text{if } p_1 \text{ is in } w_1; \\
p_e(w_1, p_1), & \text{if } p_1 \text{ is not in } w_1 \text{ and } l_p(w_1) < n - 1; \\
p_e(w_0, p_0), & \text{if } p_1 \text{ is not in } w_1 \text{ and } l_p(w_0) = n - 1. 
\end{cases}) \quad (12a) $$

In (12b), $w_t - p_t$ means that $p_t$ is deleted from $w_t$. The working set model [5] is based on the principle of locality and provides an adaptive estimator for memory demand. At each time $t$, the substring $L(t, T)$ of $W_t$ with length $T$ over the interval $[T - T + 1, T]$ maps into the state $S(t)$ such that (6) is satisfied. The set of distinct pages in $L(t, T)$ is the working set $W(t, T)$ with window size $T$ at time $t$. Thus the following terms are obviously equivalent: 1) $n(t)$; 2) $l_p(S(t))$ and 3) the working set size $w(t, T)$ which is the cardinality of $W(t, T)$. The page-replacement rule and the state-string updating procedure in this model are also governed by the LRU strategy. State transition and output generation can be determined by (10), (11) and the following.

$$ (g_b(w(p_t), p_t), h_b(w(p_t), p_t)) $$

$$ = \begin{cases} 
(p_b(w_1, p_1), q_1), & \text{if } p_1 = p_t; \\
p_b(w_1 - p_1, p_t), & \text{if } p_1 \text{ is in } w_1; \\
p_b(w_1, p_1), & \text{if } p_1 \text{ is not in } w_1 \text{ and } l_p(w_1) < n - 1; \\
p_b(w_0, p_0), & \text{if } p_1 \text{ is not in } w_1 \text{ and } l_p(w_0) = n - 1. 
\end{cases} \quad (12b) $$

PROPERTIES OF SOME DEMAND PAGING ALGORITHMS

With respect to the first-class algorithm with the LRU replacement policy, the string of each state is a LRU stack as defined by Matuson et al. [6]. The state-string or stack updating procedure is imbedded in the recursive definition for the string of a state. The page-success rate $s(n)$ which is equivalent to the success function [6] is directly determined by (1a) without utilizing the stack distances [6]. The important inclusion property [6] between a pair of states with different allocated spaces at the same time $t$ can be easily verified [11] by means of the automata model $A$ and so is the fact that $s(n)$ is nondecreasing with $n$, i.e.,

$$ s_{max} \geq s(n + 1) \geq s(n) \geq s(1) \geq 0. \quad (14) $$

Denning et al. [12] discovered nine properties of the working set model relying on some arguments based on a probabilistic approach under some assumptions. Eight of them in their updated versions to fit any reference string of finite length can be easily developed and verified without utilizing the concept of probability [11]. Let
TABLE I
Illustrating the Properties of the Working Set Model

<table>
<thead>
<tr>
<th>( T )</th>
<th>( t )</th>
<th>( W_T )</th>
<th>( T-1 ) ( f(T) )</th>
<th>( s(T) )</th>
<th>( m(T) )</th>
<th>( \sum m(z) )</th>
<th>( w(T) )</th>
<th>( N_r(T) )</th>
<th>pages ( p_i ) recurrent at ( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( S_T(T) )</td>
<td>( a \ b \ c \ b \ a \ d \ c \ a \ a )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>( W_L )</td>
<td>( q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 )</td>
<td>0.40</td>
<td>0.30</td>
<td>0.70</td>
<td>1.80</td>
<td>1.70</td>
<td>(p₁,p₁)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( \Delta(T,T) )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>( S_T(T) )</td>
<td>( a \ b \ c \ b \ a \ d \ c \ a \ a )</td>
<td>0.40</td>
<td>0.50</td>
<td>0.50</td>
<td>3.10</td>
<td>2.30</td>
<td>(p₁,p₁)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( W_L )</td>
<td>( q_1 \ q_2 \ q_3 \ q_4 \ q_5 \ q_6 \ q_7 \ q_8 \ q_9 )</td>
<td>0.40</td>
<td>0.50</td>
<td>0.40</td>
<td>3.60</td>
<td>2.90</td>
<td>(p₁,p₁)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>( \Delta(T,T) )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| \( w(T) \) | \begin{align*} w(T) &= \frac{1}{L_t} \sum_{i=1}^{L_t} w(t_i, T) \end{align*} |

The updated properties entail the following.

Property 1:

\[ 1 = w(1) \leq w(T) \leq w(T+1) \leq \min(T+1, K(P)) \]

where \( \min \) is an operation that simply chooses the minimum value between \( T+1 \) and \( K(P) \).

Property 2:

\[ w(T+1) - w(T) = m(T) - \frac{1}{L_t} w(L_t, T) \]

Property 3:

\[ 0 < f_{\min} \leq m(T+1) \leq m(T) \leq m(0) = 1 \]

Property 4:

\[ m(T) = 1 - s(T) \]

Property 5:

\[ m(T+1) - m(T) = -s'(T+1) \]

where

\[ s'(T+1) = s(T+1) - s(T) \]

Property 6:

\[ w(T) = \sum_{z \in L_t} \frac{r_w}{r_{L_t}} m(z) - \frac{1}{L_t} \sum_{L_t} w(L_t, z) \]

Property 7:

\[ w(T+1) + w(T-1) \leq 2w(T) - \frac{1}{L_t} [w(L_t, T) - w(L_t, T-1)] \]

Property 9:

\[ \lim_{r \to L_t} m(T) = f_{\min} \]

Some of these properties are demonstrated in Table I. Property 8 is shown below.

\[ w(T) \] be the average working set size for processing \( W_T \) of a finite length \( L_t \). Then

\[ w(T) = \frac{1}{L_t} \sum_{i=1}^{L_t} w(t_i, T) \]
Distinguishing Sets for Optimal State Identification in Checking Experiments

RAYMOND T. BOUTE

Abstract—A new concept, called distinguishing set or D-set is presented. Its use yields a considerable reduction in the length of checking sequences. Arbitrarily chosen examples have indicated a reduction of 30–50 percent. It is shown that the sequences of a distinguishing set actually constitute the optimum, i.e., minimum length, for state identification through input–output observations only.

Other features, such as telescoping1 and the use of a transition check status table, account for further reduction in length. Also, since a distinguishing set may exist in cases where a distinguishing sequence does not, it can often replace the (usually long) locating sequences.

Finally, the principle can be extended to the design of characterizing sequences, and thus yield shorter locating sequences.

Index Terms—Checking experiments, distinguishing sequences, distinguishing sets, fault detection, machine identification, sequential machines, state identification, transition check status table, transition verification.

I. INTRODUCTION

The concept of experiments on sequential machines [1] has led to methods for checking classes of machines against faults [2], [4], [5]. For strongly connected and reduced machines, checking experiments consist, in principle, of three parts: initialization [1]–[3], state identification [2], and transition verification [2]. Since the third part involves a large number of state identifications also, these last two parts do not have to be realized separately. Combining them will usually shorten the resulting checking sequence.

Over the years, many methods have been developed to improve the efficiency of the state identification procedure; to name only a few, resolving sequences, 1/O sequences [4], [5], and variable-length distinguishing sequences [6].

We introduce the concept of distinguishing set (D-set) with a similar purpose (Section II). In fact, we will show how this concept arises in a natural fashion when establishing the minimal requirements that are imposed on the state identification sequences of any checking experiment by the condition that the states must be distinguished by knowing only the responses to those sequences. For machines that have a D-set, this approach presents a solution to the shortest checking sequence problem in the sense described by Moore [1] and Henning [2,], that is, based on the state identification and transition verification principle. Since the design of D-sets turns out to be closely related to the design of an adaptive distinguishing experiment, the sequences can be optimized in a similar fashion [3].

The possibility of constructing shorter checking experiments in case there exists an adaptive distinguishing experiment for the machine has been anticipated by an example in [6], but no design procedure or generalization was given. Further, it was not realized that an optimal set of state identifying sequences could be constructed for those machines.

In Section III we illustrate the practical implementation of these ideas in the design of checking experiments. The telescoping feature, which is illustrated below, and the use of a transition check status table provide a method for making shortcuts in a systematic, rather than the usual ad hoc fashion.

II. DISTINGUISHING SETS

A. Notation

Let M be a sequential machine [7]. We denote by I, O, Q the input, output, and state sets, respectively. Further, $\delta(q,i)$ denotes the next state corresponding to present state $q$ and input $i$, and $\lambda(q,\overline{i})$ the sequence of output symbols when the machine is started in state $q$ and the input sequence $\overline{i}$ is applied. The bar $\overline{()}$ will be used whenever dealing with sequences of symbols. By $I^*$ we denote the set of finite input sequences together with the empty sequence $\lambda$. These symbols, without subscripts, refer to the good machine.

1 By telescoping we mean the following: If two sequences have to be applied during an experiment, and the final part (suffix) of one sequence is the same as the initial part (prefix) of the other sequence, then we can replace the experiment in such a way that this common part has to be applied only once. This can be done, provided that such an arrangement is compatible with the conditions (if any) on the state of the machine before the application of the second sequence.