JOURNAL OF COMPUTER AND SYSTEM SCIENCES: 2, 1–12 (1968)

Deterministic Stack Automata and the Quotient Operator

J. E. HOPCROFT

Cornell University, Ithaca, New York

and

Bell Telephone Laboratories, Murray Hill, 07971

AND

J. D. ULLMAN

Bell Telephone Laboratories, Murray Hill, 07971

Received July 15, 1967

ABSTRACT

A stack automaton is a pushdown automaton with the added privilege of scanning the contents of its pushdown tape without erasing. In this paper, the deterministic stack automaton with a one-way input (dsa) is considered.

It is shown that if L is a language accepted by a dsa and R is a regular set, then $L/R = \{w \mid \text{for some } x \text{ in } R, wx \text{ is in } L\}$ is accepted by a dsa. As a corollary, end markers are not needed on the input of the dsa. It is also shown that if L is accepted by a dsa, then $\text{Max}(L) = \{w \mid w \text{ in } L \text{ and for no } x \text{ is } wx \text{ in } L\}$ is accepted by a dsa.

I. INTRODUCTION

The stack automaton has been studied as an abstract recognition device [1], [2], [4]. The general device is nondeterministic, has a two-way input with end markers, a finite control and a pushdown store called the stack. Unlike a pushdown automaton, however, the stack automaton can cause its head to leave the top of the pushdown store and scan the stack in a read-only mode. This paper is concerned with deterministic one-way stack automata.

The study of operations on families of languages has proven important in the theory of languages. Many of the closure properties of (deterministic) one-way pushdown automata have been established for (deterministic) one-way stack automata [2]. In particular, the family of languages accepted by one-way stack automata has been shown to be closed under union, intersection with a regular set, concatenation, Kleene closure, generalized sequential machine mappings and quotient with a regular set. (If L_1 and L_2 are languages, then the quotient of L_1 with L_2, denoted L_1/L_2, is the set

$\{w \mid \text{for some } x \text{ in } L_2, wx \text{ is in } L_1\}$.) The family of languages accepted by deterministic one-way stack automata has been shown to be closed under generalized sequential machine mappings, intersection with a regular set and complementation. It is known [3] that if L is accepted by a deterministic pushdown automaton and R is a regular set, then L/R is a context free language. The corresponding result for the family of languages for deterministic one-way stack automata was previously unknown and is established in this paper.

Part of the interest in this result stems from the fact that the previously established closure properties, with the exception of complementation, have been shown to hold in general for (deterministic) one-way automata [5]. However, there are families of languages defined by deterministic one-way automata not closed under quotient with a regular set and thus a proof must depend on the particular structure of the device under consideration. The technique of proof used is new to the literature and should have other applications.

Another reason for the interest in the quotient with a regular set is related to the removal of endmarkers. For one-way stack automata, it is known that end markers on the input are unnecessary for the nondeterministic case. For the deterministic case the left end marker is obviously unnecessary. Since the family of languages accepted by deterministic one-way stack automata is closed under quotient with a regular set, the right end marker is also unnecessary.

II. Definitions and Notation

In this section we define the one-way stack automaton and provide the notation necessary to establish the results of the paper.

Intuitively, a one-way stack automaton consists of an input terminal, a finite control, and a stack. It is assumed that an input symbol is supplied at the input terminal when requested by the stack automaton, whereupon it is immediately used to determine the next move. The stack is a conventional pushdown store with the added capability of scanning the stack interior in a read-only mode. A move of the device is determined by the state of the finite control, the stack symbol scanned and an input symbol, provided one is requested.

Formally a *one-way stack automaton*[1] (abbreviated sa) S is an 8-tuple $(K, T, I, \delta, \delta_b, q_0, Z_0, F)$ where:

[1] The definition of sa given here differs from that in Ref. [2]. The major differences are the absence of end markers on the input and the use of two functions governing the move of the sa; one function being applicable only when the stack head is at the top of stack and the other applicable only when the stack head is not at the top of stack. Due to the lack of end markers it is not obvious that the two models are equivalent in the sense that they accept the same family of languages. That the models are equivalent will follow from the main result of the paper.

(1) K is a finite set of *states* which contains q_0, the *start state.*

(2) T is a finite set of *stack symbols* which contains b, the *blank symbol* and Z_0, the *start symbol.*

(3) I is a finite set of *input* symbols.

(4) δ is a mapping from $K \times (I \cup \{\epsilon\}) \times (T - \{b\})$ to subsets of $K \times \{L, S, R\}$ and determines the next move of S when the stack head is not at the top of the stack. (The stack is always a finite string of $(T - \{b\})^*$ followed by blanks to the right. The leftmost blank is called the *top of stack.*) The ϵ in the second component of the argument represents the situation where the sa moves without requesting an input symbol. The significance of L, S and R in the range of δ is that the stack head moves left, remains stationary, or moves right, respectively.

(5) δ_b is a mapping from $K \times (I \cup \{\epsilon\}) \times (T - \{b\})$ to subsets of $K \times [\{S, L, E\} \cup (T - \{b, Z_0\})]$ and determines the move of S when the stack head is scanning the blank at the top of stack. The move depends not only on the present state and an input symbol, if one is requested, but also on the rightmost nonblank stack symbol. The symbol S in the range of δ_b signifies that the stack head does not move, L signifies that it moves left without erasing the rightmost nonblank, and E signifies a move left followed by the printing of a blank (erasing) over the rightmost nonblank. A nonblank stack symbol in the range of δ_b (we assume S, E and L are not stack symbols) signifies that the symbol is printed over the blank at the top of stack, the stack head then moving one cell right to the new top of stack.

(6) $F \subseteq K$ is the set of *final states.*

Initially, q_0 is the state of the sa, and all stack cells contain the blank symbol, b, except one, which contains Z_0. The stack head initially scans the cell immediately to the right of the one holding Z_0 (i.e., the top of stack). Note that Z_0 is never printed by the sa, so Z_0 marks the bottom (left end) of the stack. We can assume that the stack head never moves left from Z_0. That is, $\delta(q, a, Z_0)$ does not contain (p, L) for q and p in K, and a in $I \cup \{\epsilon\}$.

A *configuration* of S[2] is a combination of state, nonblank portion of the stack and position of the stack head. A configuration of S is denoted (q, y, i), where q is in K, y is in $(T - \{b\})^*$ and i is an integer denoting the number of cells from the top of stack to the cell scanned by the stack head (i.e., $i = 0$ if the head is at the top of the stack, $i = 1$ if at the rightmost nonblank, etc.).

We define the relation $\vdash_S$ between configurations of S as follows:

[2] Unless otherwise stated, S always denotes the dsa $(K, T, I, \delta, \delta_b, q_0, Z_0, F)$. To avoid long strings of quantifiers we adopt the following conventions for certain symbols (either with or without subscripts or primes). Unless otherwise stated, q is in K, a is in $I \cup \{\epsilon\}$, w and x are in I^*, Z is in T, y is in $(T - \{b\})^*$, and D is in $\{L, S, R\}$ or $\{S, L, E\} \cup (T - \{b, Z_0\})$.

(1) For $i > 0$, if (q_1, D) is in $\delta(q, a, Z)$, then

$$a : (q, y_1Zy_2, i) \vdash_S (q_1, y_1Zy_2, i + m) \quad \text{where} \quad i = |y_2| + 1$$

and $m = +1$, 0 or -1 respectively as $D = L$, S or R.

(2) If (q_1, L) is in $\delta_b(q, a, Z)$, then

$$a : (q, yZ, 0) \vdash_S (q_1, yZ, 1).$$

(3) If (q_1, S) is in $\delta_b(q, a, Z)$, then

$$a : (q, yZ, 0) \vdash_S (q_1, yZ, 0).$$

(4) If (q_1, E) is in $\delta_b(q, a, Z)$, then

$$a : (q, yZ, 0) \vdash_S (q_1, y, 0).$$

(5) If (q_1, Z_1) is in $\delta_b(q, a, Z)$, Z_1 in T, then

$$a : (q, yZ, 0) \vdash_S (q_1, yZZ_1, 0).$$

If for $1 \leqslant j \leqslant n$, $a_j : (q_j, y_j, i_j) \vdash_S (q_{j+1}, y_{j+1}, i_{j+1})$, then

$$a_1a_2...a_n : (q_1, y_1, i_1) \vdash_S^* (q_{n+1}, y_{n+1}, i_{n+1}).$$

We define an sa, to be a *deterministic one-way stack automaton* (dsa) if, intuitively, there is at most one move possible from any configuration. Formally, we say S is a dsa, if, for each q in K, Z in T, and a in I,

(1) $\delta(q, a, Z)$, $\delta(q, \epsilon, Z)$, $\delta_b(q, a, Z)$ and $\delta_b(q, \epsilon, Z)$ each contain at most one element,

(2) $\delta(q, a, Z)$ nonempty implies $\delta(q, \epsilon, Z)$ empty, and

(3) $\delta_b(q, a, Z)$ nonempty implies $\delta_b(q, \epsilon, Z)$ empty.

An sa accepts a language by final state. The language accepted by an sa S, denoted $T(S)$, is the set $\{w \mid w : (q_0, Z_0, 0) \vdash_S^* (q, y, i)$ for some q in F, y in T^* and integer $i\}$ and is called an *sa language*. A language accepted by a dsa is called a *dsa language*. [Note that a string w may be written as $a_1a_2...a_n$ in several different ways depending on which a_i are ϵ. Furthermore, w may take the configuration $(q_0, Z_0, 0)$ to several different configurations even for a dsa depending on the number of moves on ϵ input at the end of w. If any one of these configurations contains a final state, then w is in $T(S)$].

III. Preliminary Results

In this section we establish some preliminary results which will simplify the proof of the main theorem.

First we show that acceptance by final state can be changed to acceptance by pair of state and stack symbol scanned without affecting the family of languages accepted.

LEMMA 1. *Let S be a* dsa *and let $B_1 = \{(q_1, Z_1), (q_2, Z_2), \ldots, (q_m, Z_m)\}$ and $B_2 = \{(p_1, X_1), (p_2, X_2), \ldots, (p_r, X_r)\}$ be sets of pairs of state and stack symbol. Let L be the set of all w in I^* such that either*

$$\text{(i) } w : (q_0, Z_0, 0) \vdash_S^* (q, yZ, 0)$$

for some y in $(T - \{b\})^*$ and (q, Z) in B_1, or (ii)

$$w : (q_0, Z_0, 0) \vdash_S^* (q, y_1Zy_2, j),$$

where $j = |y_2| + 1$ and (q, Z) is in B_2. Then L is a dsa *language.*

Proof. We construct a dsa S' from S such that S' accepts L. S' has an unprimed, a primed and a doubly primed state for each state of S. S' simulates a move of S going from an unprimed state to a doubly primed state. If the stack head is at the top of stack and the combination of state and top stack symbol is in B_1, S' on ϵ input goes to a primed version of the new state, accepts the input, and then enters an unprimed version. If the combination of state and top stack symbol is not in B_1, S' on ϵ input goes directly to the unprimed version. Likewise, if the stack head is not at the top of stack, S' goes from the doubly primed state to the unprimed version either through the primed version or directly depending on whether or not the pair of state and stack symbol scanned is in B_2.

Formally let $S' = (K', T, I, \delta', \delta_b', q_0, Z_0, F')$ where $K' = \{q, q', q'' \mid q \text{ in } K\}$, $F = \{q' \mid q \text{ in } K\}$, and δ' and δ_b' are defined as follows.

(1) If $\delta(q, a, Z) = (q_1, D)$,[3] D in $\{L, S, R\}$, then $\delta'(q, a, Z) = (q_1'', D)$.

(2) If (q, Z) is in B_2, then $\delta'(q'', \epsilon, Z) = (q', S)$ and $\delta'(q', \epsilon, Z) = (q, S)$.

(3) If (q, Z) is not in B_2, then $\delta'(q'', \epsilon, Z) = (q, S)$.

(4) If $\delta_b(q, a, Z) = (q_1, D)$, D in $\{S, L, E\} \cup T - \{b, Z_0\}$, then $\delta_b'(q, a, Z) = (q_1'', D)$.

(5) If (q, Z) is in B_1, then $\delta_b'(q'', \epsilon, Z) = (q', S)$ and $\delta_b'(q', \epsilon, Z) = (q, S)$.

(6) If (q, Z) is not in B_1, then $\delta_b'(q'', \epsilon, Z) = (q, S)$.

Clearly $T(S') = L$.

The language accepted by a dsa S by empty stack, denoted $N(S)$, is the set $\{w \mid w : (q_0, Z_0, 0) \vdash_S^* (q, \epsilon, 0) \text{ for some } q \text{ in } K\}$.

LEMMA 2. *Let $L \subseteq I^*$ be a language accepted by some* dsa *by final state. Let* **\$** *be a symbol not in I. Then there exists a* dsa *which accepts $L\$ = \{w\$ \mid w \text{ in } L\}$ by empty stack.*

[3] Strictly speaking $\delta(q, a, Z) = \{(q_1, D)\}$. For simplicity of notation we delete the brackets.

Proof. Let S be a dsa accepting L by final state. Without loss of generality we assume that S never erases Z_0 since we can replace $\delta_b(q, a, Z_0) = (q_1, E)$ by $\delta_b(q, a, Z_0) = (q_2, S)$, where q_2 is a new state from which no move is possible, and q_2 is in F only if q_1 is in F. Furthermore, we may assume that S never makes an infinity of moves on ϵ input [2].[4]

For each q in K, let q' be a new symbol and K' be the set of all such q'. We construct $S' = (K \cup K' \cup \{\bar{q}\}, T, I \cup \{\$\}, \delta', \delta_b, q_0, Z_0, \phi)$ to accept $L\$$ by empty stack as follows. S' behaves as S until S enters a final state. If S enters a final state and no move on ϵ input is possible, then on input $\$$, S' enters the special state $\bar{q}$. On an input in I, S' behaves as S. If S enters a final state, and moves on ϵ input are possible, S' makes the moves on ϵ input but uses primed states. When no further move on ϵ input is possible, S' on input $\$$ enters $\bar{q}$ and on an input in I returns to an unprimed state, behaving as S. The state $\bar{q}$ causes S' to raise its stack head to the top of the stack and then erase the stack. Formally

(1) For q in $K - F$, $\delta'(q, a, Z) = \delta(q, a, Z)$.

(2) For q in F, if $\delta(q, \epsilon, Z)$ is empty, then $\delta'(q, \$, Z) = (\bar{q}, S)$ and for $a \neq \epsilon$, $\delta'(q, a, Z) = \delta(q, a, Z)$.

(3) For q in F, if $\delta(q, \epsilon, Z) = (q_1, D)$, then $\delta'(q, \epsilon, Z) = (q_1', D)$ and for q in K, if $\delta(q, \epsilon, Z) = (q_1, D)$, then $\delta'(q', \epsilon, Z) = (q_1', D)$.

(4) For q in K, if $\delta(q, \epsilon, Z)$ is empty, then $\delta'(q', \$, Z) = (\bar{q}, S)$ and for $a \neq \epsilon$, $\delta'(q', a, Z) = \delta(q, a, Z)$.

(5) Statements similar to (1) to (4) above hold for δ_b.

(6) $\delta(\bar{q}, \epsilon, Z) = (\bar{q}, R)$ and $\delta_b(\bar{q}, \epsilon, Z) = (\bar{q}, E)$.

Clearly S' is deterministic and accepts $L\$$ by empty stack.

The main theorem of the paper states that the family of languages accepted by dsa is closed under quotient with a regular set. The gist of the proof is to show that a dsa can be modified so that it will have "sufficient information" to answer the question: "Does there exist an input sequence in a regular set such that if the sequence is applied to the dsa in its present configuration, the dsa will end up in a configuration in which it accepts the input?" Part of the needed information is stored on the stack and part in the finite control.

We now describe the construction by which the dsa is modified so as to store the needed additional information on the stack. To this end we introduce a finite automaton which later will be fixed to be a finite automaton accepting the regular set with respect to which the quotient is taken.

[4] Although the model in Ref. [2] is different, the same technique applies.

Let S be a dsa and let $A = (K_A, I, \delta_A, p_0, F_A)$[5] be a finite automaton. There exists a dsa S' which operates as S except that whenever S prints a stack symbol, S' prints both the stack symbol and a mapping which answers the following two questions:

(1) For q_1 and q_2 in K and p_1 and p_2 in K_A, does there exist an input string w which will take S from state q_1 to q_2 and take A from state p_1 to p_2 while S either keeps its stack head steady or moves the stack head lower on the stack and back?

(2) For q_1 in K and p_1 in K_A does there exist an input string w which causes S in state q_1 to erase its stack (possibly first increasing the length) and causes A in state p_1 to enter a final state?

To this end let α be a mapping from $K \times K_A \times K \times K_A$ to $\{0,1\} \times \{0,1\}$ and let C be the set of all such mappings. Let y be a string of nonblank stack symbols. The mapping α is said to *describe* y if when $\alpha(q_1, p_1, q_2, p_2) = (i_1, i_2)$, then

(1) $i_1 = 1$ if and only if there exists w in I^* such that $\delta_A(p_1, w) = p_2$ and $w : (q_1, y, 1) \vdash_S^* (q_2, y, 1)$ by a sequence of moves in which the stack head never reaches the top of stack (i.e., S never enters a configuration $(q', y, 0)$ for any q' in K).

(2) $i_2 = 1$ if and only if there exists w in I^* such that $\delta_A(p_1, w)$ is in F_A and $w : (q_1, y, 0) \vdash_S^* (q_2, \epsilon, 0)$.

It is important to note that if the same α describes both y and y', then for Z in T, the same α' describes yZ and $y'Z$. That is, if a stack symbol is added to the stack, the mapping describing the new stack depends only on the new stack symbol and the mapping describing the old stack.[6]

LEMMA 3. *Let S be a* dsa *and A be a finite automaton. There exists a* dsa

$$S' = (K, T \times C, I, \delta', \delta_b', q_0, [Z_0, \alpha_0], \emptyset)^{7} \textit{ such that}$$

$$x : (q_0, [Z_0, \alpha_0], 0) \vdash_{S'}^* (q, [Z_0, \alpha_0][Z_1, \alpha_1]\ldots[Z_n, \alpha_n], i)$$

if and only if

$$x : (q_0, Z_0, 0) \vdash_S^* (q, Z_0Z_1\ldots Z_n, i),$$

and for $0 \leqslant i \leqslant n$, *$\alpha_i$ is the mapping that describes $Z_0Z_1\ldots Z_i$.*

[5] A finite automaton A is a 5-tuple $(K_A, I, \delta_A, p_0, F_A)$ where K_A is a finite set (of *states*), I is a finite set (of *inputs*), δ_A is a mapping from $K_A \times I$ to K_A, p_0 is in K_A (the *start* state), and $F_A \subseteq K_A$ (the *final* states). δ_A is extended to $K_A \times I^*$ as follows. For each p in K_A, a in I and x in I^*, $\delta_A(p, \epsilon) = p$ and $\delta_A(p, xa) = \delta_A(\delta_A(p, x), a)$. The set accepted by A is $\{x \mid \delta_A(p_0, x) \text{ in } F_A\}$. A set is *regular* if and only if it is accepted by some finite automaton. A will always denote the finite automaton $(K_A, I, \delta_A, p_0, F_A)$.

[6] Although we shall not prove it here, there is an effective procedure for constructing the new mapping from the stack symbol and old mapping.

[7] Unless other wise stated, S' will always denote the dsa $(K, T \times C, I, \delta', \delta_b', q_0, [Z_0, \alpha_0], \emptyset)$.

Proof. The functions δ' and δ_b' are defined in the following manner.

(1) $\delta'(q, a, [Z, \alpha]) = \delta(q, a, Z)$. (If the stack head of S' is in the stack, then S' disregards the mappings and moves exactly as S.)

(2) If $\delta_b(q, a, Z) = (q_1, D)$, D in $\{S, L, E\}$, then $\delta_b'(q, a, [Z, \alpha]) = (q_1, D)$. (If with stack head at the top of the stack, S does not move its stack head or moves it left, then S' does likewise. If S erases a stack symbol, then S' erases the corresponding stack symbol and mapping.)

(3) If $\delta_b(q, a, Z_1) = (q_1, Z_2)$, Z_2 in T, then $\delta_b'(q, a, [Z_1, \alpha_1]) = (q_1, [Z_2, \alpha_2])$ where α_2 is the mapping which describes yZ_2, y being any string in T^* described by α_1.[8]

Clearly S' satisfies the conditions of the lemma.

Next we show that the dsa S' of Lemma 3 can be further modified so that when it moves its stack head into the stack, it can carry information in its finite control to answer the question: For q in K and p in K_A does there exist a w in I^* which causes S to erase its stack and for which $\delta_A(p, w)$ is in F_A?

To this end let m be a mapping from $K \times K_A$ to $\{0,1\}$ and let M be the set of all such maps. Let $y = [Z_0, \alpha_0][Z_1, \alpha_1] \ldots [Z_i, \alpha_i] \ldots [Z_n, \alpha_n]$ be a string in $(T \times C)^*$ with the property that for $0 \leqslant j \leqslant n$, α_j describes $Z_0 Z_1 \ldots Z_j$. The mapping m is said to be *associated* with $[Z_i, \alpha_i]$ in the string y if $m(q, p) = 1$ if and only if there exists w in I^* such that $\delta_A(p, w)$ is in F_A and $w : (q, Z_0 Z_1 \ldots Z_i \ldots Z_n, n - i + 1) \vdash_S^* (q_1, \epsilon, 0)$ for some q_1 in K.

We now define a function Δ of $M \times T \times C$ into M which relates the mapping associated with $[Z_{i-1}, \alpha_{i-1}]$ to the symbol $[Z_{i-1}, \alpha_{i-1}]$ and the mapping associated with $[Z_i, \alpha_i]$. Formally $\Delta(m_1, [Z, \alpha]) = m_2$ where $m_2(q, p) = 1$ if and only if there exists a in $I \cup \{\epsilon\}$, q_1 and q_2 in K, p_1 and p_2 in K_A such that (i) the first component of $\alpha(q, p, q_1, p_1)$ is 1, (ii) $\delta(q_1, a, Z) = (q_2, R)$ and $\delta_A(p_1, a) = p_2$, and (iii) $m_1(q_2, p_2) = 1$. [For the dsa with stack head at position $n - i \neq 0$ to erase the stack, the stack head must first get to the top of the stack. To do so the stack head must, possibly first wiggling below position $n - i$ (first component of $\alpha = 1$), pass through position $n - (i + 1)$ ($\delta(q_1, a, R) = (q_2, R)$) and starting from this position ultimately get to the top and erase the stack ($m_1 = 1$)].

Δ is extended to $M \times (T \times C)^*$ as follows. For each m in M, X in $T \times C$ and y in $(T \times C)^*$, $\Delta(m, \epsilon) = m$ and $\Delta(m, yX) = \Delta(\Delta(m, y), X)$. The inverse function is defined by $\Delta^{-1}(m, y) = \{m' \mid \Delta(m', y) = m\}$.

In the next lemma a dsa S'' is constructed from S'. S'' simulates S'. Whenever the stack head of S' enters the stack, the stack head of S'' enters the stack. However, S'' also computes in its finite control, the mapping m associated with the stack symbol

[8] If there is no y in T^* described by α_1, then $\delta_b'(q, a, [Z_1, \alpha_1])$ can be left undefined since $[Z_1, \alpha_1]$ will never appear on the stack.

scanned. Whenever the stack head moves lower on the stack, the mapping Δ determines uniquely the new mapping m to be stored in the finite control. The crux of proof is to compute the new m whenever the stack head moves higher on the stack. The difficulty is that Δ^{-1} is not necessarily unique. In the situation where Δ^{-1} is multivalued, S'' must determine the appropriate m in Δ^{-1} without losing the position on the stack. The details of this construction are given in the next lemma.

LEMMA 4. *Let S be a* dsa. *Let S' be the* dsa *constructed from S according to Lemma* 3. *There exists a* dsa $S'' = (K'', T \times C, I, \delta'', \delta''_b, q_0, [Z_0, \alpha_0], \emptyset)$ *such that:*

(1) $x : (q_0, [Z_0, \alpha_0], 0) \vdash^*_{S''} (q, [Z_0, \alpha_0] \ldots [Z_n, \alpha_n], 0)$ *if and only if* $x : (q_0, Z_0, 0) \vdash^*_S (q, Z_0 \ldots Z_n, 0)$,

(2) $x : (q_0, [Z_0, \alpha_0], 0) \vdash^*_{S''} ([q, m_i], [Z_0, \alpha_0] \ldots [Z_n, \alpha_n], n - i + 1), i > 0$, *if and only if* $x : (q_0, Z_0, 0) \vdash^*_S (q, Z_0 \ldots Z_n, n - i + 1)$ *and m_i is the mapping associated with $[Z_i, \alpha_i]$ in the string $[Z_0, \alpha_0] \ldots [Z_n, \alpha_n]$.*

Proof. S'' is to behave exactly as S', except when S' moves into the stack. Thus

(1) If $\delta'_b(q, a, [Z, \alpha]) = (q_1, S), (q_1, E)$ or $(q_1, [Z_1, \alpha_1])$, then

$$\delta''_b(q, a, [Z, \alpha]) = \delta'_b(q, a, [Z, \alpha]).$$

When S' moves into the stack, S'' must compute the appropriate table m. Thus

(2) If $\delta'_b(q, a, [Z, \alpha]) = (q_1, L)$, then $\delta''_b(q, a, [Z, \alpha]) = ([q_1, m_0], L)$ were m_0 is the mapping associated with the top nonblank symbol of the stack. (N.B.: m_0 depends only on $[Z, \alpha]$).

(3) For each q in K let $\bar{q}$ be a new symbol and let $\bar{K}$ be the set of all such symbols.

(a) If $\delta'(q, a, [Z, \alpha]) = (q_1, L)$, then for each m in M,

$$\delta''([q, m], a, [Z, \alpha]) = ([\bar{q}_1, m], L).$$

(b) For each $\bar{q}_1$ in $\bar{K}$, m in M, and $[Z, \alpha]$ in $T \times C$,

$$\delta''([\bar{q}_1, m], \epsilon, [Z, \alpha]) = ([q_1, \Delta(m, [Z, \alpha])], S).$$

(Whenever S' moves left, S'' moves left and then computes the new table.)

(4) If $\delta'(q, a, [Z, \alpha]) = (q_1, S)$, then for each m in M,

$$\delta''([q, m], a, [Z, \alpha]) = ([q_1, m], S).$$

The only case remaining is the situation where S in state q moves right and enters state q_1. Since $\Delta^{-1}(m, [Z, \alpha])$ may not be unique, S'' must somehow determine the element of $\Delta^{-1}(m, [Z, \alpha])$ associated with the symbol to the right of $[Z, \alpha]$ on the stack. If $\Delta^{-1}(m, [Z, \alpha])$ is unique and $\delta'(q, a, [Z, \alpha]) = (q_1, R)$, then

$\delta''([q, m], a, [Z, \alpha]) = ([q_1, \Delta^{-1}(m, [Z, \alpha])], R)$. If $\Delta^{-1}(m, [Z, \alpha]) = \{m_1, m_2, \ldots, m_r\}$,

$r \geqslant 2$, then S'' moves in the following manner. Assume the stack of S is $Y_nY_{n-1} \dots Y_1$[9] where $Y_n = Z_0$. Furthermore, assume S is scanning Y_i when it moves right one symbol. S'' moves right one symbol, computes $\Delta^{-1}(m, Y_i) = \{m_1, m_2, \dots, m_r\}$ and then continues to move right (up the stack), computing for each j, $1 \leqslant j \leqslant r$, $\Delta^{-1}(m_j, Y_{i-1})$, $\Delta^{-1}(m_j, Y_{i-2}Y_{i-1})$, $\Delta^{-1}(m_j, Y_{i-3}Y_{i-2}Y_{i-1})$, etc. until one of two conditions is reached.

(1) S'' reaches the top of stack.

(2) S'' reaches a symbol Y_l for which $\Delta^{-1}(m_j, Y_lY_{l+1} \dots Y_{i-1})$ is nonempty for only one value of j.

In case (1), clearly that m_j for which $\Delta^{-1}(m_j, Y_1Y_2 \dots Y_{i-1})$ contains m_0 is the desired mapping. In case (2), the m_j such that $\Delta^{-1}(m_j, Y_lY_{l+1} \dots Y_{i-1})$ is nonempty is the desired mapping. Now, having determined the correct mapping, S'' must return its stack head to the symbol Y_{i-1}. The process of finding Y_{i-1} is similar for both cases and only case 2 will be handled. Let k be such that only $\Delta^{-1}(m_k, Y_lY_{l+1} \dots Y_{i-1})$ is nonempty. S'' selects m' from $\Delta^{-1}(m_k, Y_{l+1}Y_{l+2} \dots Y_{i-1})$ and m'' from $\Delta^{-1}(m_{k'}, Y_{l+1}Y_{l+2} \dots Y_{i-1})$ where $m_{k'}$ is any mapping in $\{m_1, m_2, \dots, m_r\}$ other than m_k with $\Delta^{-1}(m_{k'}, Y_{l+1}Y_{l+2} \dots Y_{i-1})$ nonempty. Clearly some such $m_{k'}$ exists since S'' stops moving right at the first symbol for which condition 2 above is satisfied. Finally S'' moves left computing $\Delta(m', Y_{l+1}), \Delta(m', Y_{l+1}Y_{l+2}), \dots$ and $\Delta(m'', Y_{l+1})$, $\Delta(m'', Y_{l+1}Y_{l+2}), \dots$ until a symbol Y_h is reached such that

$$\Delta(m', Y_{l-1}Y_{l+2} \dots Y_h) = \Delta(m'', Y_{l+1}Y_{l+2} \dots Y_h).$$

At this point S'' moves its stack head one symbol right and enters state $[q_1, m_k]$ where q_1 is the state that S entered when it moved right. The claim is made that $h = i$ and thus the stack head of S'' is properly positioned to simulate the next move of S. To see this, note that m' and m'' in $\Delta^{-1}(m, Y_{l+1}Y_{l+2} \dots Y_i)$ implies $\Delta(m', Y_{l+1}Y_{l-2} \dots Y_i) = \Delta(m'', Y_{l-1}Y_{l-2} \dots Y_i) = m$. Thus $h \leqslant i$. Furthermore m' in $\Delta^{-1}(m_k, Y_{l+1}Y_{l+2} \dots Y_{i-1})$ and m'' in $\Delta^{-1}(m_{k'}, Y_{l+1}Y_{l+2} \dots Y_{i-1})$, $m_k \neq m_{k'}$, implies $\Delta(m', Y_{l+1}Y_{l+2} \dots Y_{i-1}) = m_k$ and $\Delta(m'', Y_{l+1}Y_{l+2} \dots Y_{i-1}) = m_{k'}$. Thus $\Delta(m', Y_{l+1}Y_{l+2} \dots Y_g) \neq \Delta(m'', Y_{l+1}Y_{l-2} \dots Y_g)$ for $g < i$. Therefore $h = i$. (Intuitively, as the stack head moves back down the stack Y_h is the first symbol reached for which

$$\Delta(m', Y_{l+1}Y_{l+2} \dots Y_h) = \Delta(m'', Y_{l+1}Y_{l+2} \dots Y_h)$$

and since this is certainly true by the time the symbol Y_i is reached, h must be less than or equal i. On the other hand, once a symbol Y_h is reached such that the two

[9] Note that we have renumbered the symbols on the stack so that the numbers increase from top to bottom. The reason for this is to facilitate understanding of the construction by making the subscript of the stack symbol correspond to the position on the stack.

quantities are equal, they remain equal for all symbols lower on the stack. But since they are not equal at Y_{i-1}, h must be greater than $i - 1$ and hence $h = i$).

IV. Main Results

Using Lemma 4, it is easily shown that the class of languages accepted by dsa by final state is closed under quotient with a regular set.

THEOREM 1. *If $L \subseteq I^*$ is accepted by a* dsa *by final state and $R \subseteq I^*$ is a regular set, then L/R is accepted by a* dsa.

Proof. Let \$ be a symbol not in I, let $R' = R\$$ and let $A = (K_A, I \cup \{\$\}, \delta_A, q_0, F_A)$ be a finite automaton accepting R'. By Lemma 2, construct a dsa S accepting $L\$$ by empty stack. By Lemmas 3 and 4 construct S'' from S and R'. Let $B_1 = \{(q, [Z, \alpha]) | \exists$ q_1 in K, p_1 in F_A for which the second component of $\alpha(q, p_0, q_1, p_1)$ is 1$\}$ and let $B_2 = \{([q, m], [Z, \alpha]) \; m(q, p_0) = 1\}$. Now the set

$$\{x \mid x : (q_0, [Z_0, \alpha_0], 0) \vdash^*_{S''} (q, y[Z, \alpha], 0), (q, [Z, \alpha]) \text{ in } B_1 \text{ or}$$

$x : (q_0, [Z_0, \alpha_0], 0) \vdash^*_{S''} ([q, m], y_1[Z, \alpha] y_2, j), j = |y_2| + 1$ and $([q, m], [Z, \alpha])$ in $B_2\}$ is L/R. Thus by Lemma 1, L/R is accepted by a dsa by final state.

Other authors [2] have considered dsa with end markers.[10] A language $L \subseteq I^*$ is accepted by a dsa in their formulation, if ¢L\$, ¢ and \$ not in I, is accepted by a dsa in our formulation. Clearly if ¢L\$ is accepted in our formulation, then so is L\$. Thus by Theorem 1, L is accepted. Thus the family of languages accepted by dsa is the same with or without end markers.

COROLLARY. *If ¢L\$, $L \subseteq I^*$, ¢ and \$ not in I, is accepted by a* dsa, *then L is accepted by a* dsa.

Proof. If ¢L\$ is accepted by a dsa $S = (K, T, I \cup \{¢, \$\}, \delta, \delta_b, q_0, Z_0, F)$, then clearly L\$ is accepted by the dsa $\hat{S} = (K \cup \hat{K}, T, I \cup \{¢\}, \hat{\delta}, \hat{\delta}_b, \hat{q}_0, Z_0, F)$ where $\hat{K} = \{\hat{q} \mid q \text{ in } K\}$, $\hat{\delta}(q, a, Z) = \delta(q, a, Z)$ for a in $I \cup \{\$, \epsilon\}$, $\hat{\delta}(\hat{q}, \epsilon, Z) = (\hat{q}_1, D)$ if $\delta(q, \epsilon, Z) = (q_1, D)$, $\hat{\delta}(\hat{q}, \epsilon, Z) = \delta(q, ¢, Z)$, and $\hat{\delta}_b$ is defined in a manner similar to δ. The corollary follows from Theorem 1 by letting $R = \{\$\}$.

Remarks. An operation closely related to quotient with a regular set is the operation Max defined by $\text{Max}(L) = \{x \mid x \text{ is in } L$ and for no y in $I^* - \{\epsilon\}$ is xy in $L\}$. Let F be

[10] There are other differences in the models but it is easily shown that these other differences do not affect the family of languages accepted.

the set of final states of a dsa accepting L. By letting $R = I^* - \{\epsilon\}$ and modifying the construction of Theorem 1 so that $B_1 = \{(q, [Z, \alpha]) \mid q$ in F and for no q' and p' is the second component of $\alpha(q, p_0, q', p') = 1\}$ and $B_2 = \{([q, m], [Z, \alpha]) \mid q$ in F and $m(q, p_0) = 0\}$ we obtain a dsa accepting $\mathrm{Max}(L)$.

Since $\Sigma^* - L = \mathrm{Max}(\Sigma^*\$ \cup L\$\Sigma^*)/\$$, the family of languages accepted by a nondeterministic sa is not closed under Max since it is not closed under complement. This provides an alternate proof that the family of languages accepted by dsa is properly contained in the family of languages accepted by an sa.

An sa is said to be *nonerasing* if for each q in K, a in $I \cup \{\epsilon\}$ and Z in $T - \{b\}$, $\delta_b(q, a, Z)$ is contained in $K \times [\{S, L\} \cup (T - \{b, Z_0\})]$. The techniques used in this paper can be used to show that the family of languages accepted by nonerasing dsa is closed under quotient with a regular set. Lemma 2 is no longer valid, and is replaced by a lemma stating that $L\$$ can be accepted by a dsa which enters an accepting configuration when the stack head is at the top of stack. Appropriate changes must also be made in Lemmas 3 and 4.

References

1. S. Ginsburg, S. Greibach, and M. Harrison. *J. ACM* **14**, 172–201 (1967).
2. S. Ginsburg, S. Greibach, and M. Harrison. *J. ACM* **14**, 389–418 (1967).
3. S. Ginsburg and S. Greibach. *Info. Control* **9**, 620–648 (1966).
4. J. Hopcroft and J. Ullman. *J. Computers Syst. Sci.* **1**, 166–186 (1967).
5. J. Hopcroft and J. Ullman. *BSTJ* **46**, 1793–1829 (1967).