ONE-WAY STACK AUTOMATA - EXTENDED ABSTRACT*

Seymour Ginsburg

System Development Corporation, Santa Monica, California

Sheila A. Greibach

Harvard University

Michael A. Harrison

University of California at Berkeley

INTRODUCTION

In [3], the notion of a stack automaton, both deterministic and nondeterministic, was defined. This device embodies many features used in the recognition aspect of currently used compilers. It is less powerful than a Turing machine but more potent than a pushdown automaton. Specifically, a stack automaton allows reading, but not writing, in the interior of the stack. [This occurs, for example, when reading symbol tables.] It permits writing and erasing only on a last-in first-out basis. The stack automaton also permits reading the input many times (technically, a two-way read which corresponds to a multi-pass compiler). The present paper discusses the important case when the stack automaton reads the input tape from left to right only, i.e., a single-pass compiler. This device, a generalization of the pushdown automaton, is called a "one-way stack automaton." Of special interest are the sets of words recognized by one-way stack automata, hereafter called "languages."

Our motivation in studying one-way stack automata is two-fold. First, it is a natural specialization of the stack automata. And second, ALGOL can be recognized by this type of device. As is well-known, ALGOL cannot be entirely recognized by a pushdown automaton. Thus sets accepted by one-way stack automata may be better approximations to currently used programming languages than sets recognized by pushdown automata, i.e., context free languages.

There are five sections to the paper. Section one introduces one-way stack automata and languages. Section two considers various operations which preserve the family of languages. In section three, a representation theorem for recursively enumerable sets is proved. Using this theorem, a number of non-closure results are obtained. In section four, some closure properties of D-languages (languages accepted by deterministic one-way stack automata) are presented. Section five considers decision problems. It is shown elsewhere [5] that each one-way stack language is context sensitive, and each D-language is accepted by a deterministic linear bounded automaton.

In this abstract we shall give the pertinent definitions and main results. All technical lemmas and proofs are omitted. They may be found in the full version of this paper [4].

Section 1. Preliminaries

In this section, we define the basic objects with which we shall be concerned, namely the one-way stack automata and the sets recognized or accepted by them.

Roughly speaking, a one-way stack automaton consists of a "finite state control", an "input" sequence or tape, and a "stack". The device advances the input tape at most one symbol per move. The stack is a "last-in first-out" store, i.e., it may be written or erased from the right end in the conventional way. In addition, the interior part of the stack may be read but not rewritten.

A one-way stack automaton operates in the following manner. If the device is in a state, reading both an input symbol and a stack symbol, then it simultaneously

(i) goes to another state.

(ii) moves at most one symbol to the right of the input symbol just read.

(iii) does exactly one of two alternatives:

(a) It may move its stack pointer (= read-write head) one symbol to the left or to the right, or keep it stationary.

(b) If it is reading the rightmost symbol on the stack, then it may write a (possibly empty) finite sequence of symbols onto the stack, simultaneously erasing the symbol just read.

*The research reported in this abstract was sponsored in part by the Air Force Cambridge Research Laboratories, Office of Aerospace Research, under Contract AF 19(628)-5166, CRL - Algorithmic Languages Program.

The reader is referred to [3] for a further discussion of a stack automaton as well as the motivation for its definition.

We now formalize the above intuitive description.

Definition. A one-way stack automaton, abbreviated one-way sa, is a 9-tuple $A = (K,\Sigma,\phi,\$,\Gamma,\delta,q_o,Z_o,F)$ satisfying the following conditions:

(1) K is a finite nonempty set (of states).

(2) Σ is a finite nonempty set (of inputs).

(3) ϕ and $\$$ are two elements not in Σ (the left and right endmarkers for the input).

(4) Γ is a finite nonempty set (of stack symbols).

(5) Z_o is in Γ (the initial stack symbol).

(6) δ is a function[1] from $K \times (\Sigma \cup \{\phi,\$\}) \times \Gamma$ into the set of finite subsets of $\{0,1\} \times K \times \{-1,0,1\} \times \Gamma^*$ having[2] the following property: (*) If (d,q',e,w) is in $\delta(q,a,Z)$ and $w \neq Z$, then $e = 0$.

(7) q_o is in K (the start state).

(8) $F \subseteq K$ (the set of final states).

The formalism[3] (d,q',e,Z) is in $\delta(q,a,Z)$ means the following: Suppose the one-way sa A is in state q, reading a on the input tape and Z on the stack. Then A may[4]

(i) go to state q';

(ii) move right on the input tape if $d = 1$ and remain stationary if $d = 0$;

and (iii) move left on the stack if $e = -1$, right if $e = 1$, and stay stationary if $e = 0$.

The formalism $(d,q',0,w)$ is in $\delta(q,a,Z)$ means that if A is in configuration (q,a,Z) of $K \times (\Sigma \cup \{\phi,\$\}) \times \Gamma$; then A may

(i) move on the input as specified by d;

(ii) go to state q';

and (iii) write w in place of Z.
[The "write" command will be restricted shortly so that it is applicable only when A scans the rightmost stack symbol.]

The symbolism introduced so far allows only a single move of the device. The formalism is now expanded to allow us to discuss sequences of basic operations of the device.

Definition. An instantaneous description (abbreviated ID) of a one-way sa is any element of $K \times (\Sigma \cup \{\phi,\$\})^* \times (\Gamma \cup \{\uparrow\})^*$, where[5] $\uparrow$ is a symbol not in Γ.

The ID $(q,a_i \ldots a_k, Z_1 \ldots Z_j \uparrow Z_{j+1} \ldots Z_\ell)$ denotes the fact that A is in state q, reading input a_i, with $Z_1 \ldots Z_\ell$ on the stack and A scanning Z_j. $\uparrow$ is referred to as the "stack pointer".

Notation. Given a one-way sa $A = (K,\Sigma,\phi,\$,\Gamma,\delta,q_o,Z_o,F)$, let $\vdash$ be the relation between ID's defined as follows: Let $i,k,\ell \geq 1$; $a_1,\ldots,a_{k_*}$ in $\Sigma \cup \{\phi,\$\}$; $a_{k+1} = \epsilon$; and $Z_1,\ldots,Z_\ell$ in Γ; y in Γ^*; and Z in Γ.

(1) If (d,q',e,Z_j) is in $\delta(q,a_i,Z_j)$ where $1 \leq i \leq k$ and $1 \leq j \leq \ell$ satisfy the following conditions: (i) $e \geq 0$ if $j = 1$, and (ii) $e \leq 0$ if $j = \ell$; then

$(q,a_i \ldots a_k, Z_1 \ldots Z_j \uparrow \ldots Z_\ell) \vdash (q',a_{i+d} \ldots a_k, Z_1 \ldots Z_{j+e} \uparrow \ldots Z_\ell)$.[6]

---

[1] For sets of words X and Y, the (complex) product of X and Y, written XY, is the set $\{xy/x \text{ in } X, y \text{ in } Y\}$, where xy is the concatenation of x and y. Let $X^0 = \{\epsilon\}$, where ϵ is the empty word. For $i \geq 0$, let $X^{i+1} = X^i X$ and $X^* = \bigcup_{i=0}^{\infty} X^i$. Thus Σ^* is the free semigroup with identity generated by Σ.

[2] Because of a difference in point of view between [3] and the present paper, a one-way sa now means a nondeterministic device. In [3], a one-way sa was taken as a deterministic device.

[3] To avoid long strings of quantifiers, unless stated otherwide q and q' are in K, a is in $\Sigma \cup \{\phi,\$\}$, Z is in Γ, d is in $\{0,1\}$, and e is in $\{-1,0,1\}$.

[4] A is nondeterministic and thus may have other choices.

[5] As in [3], for technical convenience many elements are called instantaneous descriptions even though they do not correspond to actual configurations of a one-way sa, e.g., any ID with two occurrences of $\uparrow$.

[6] $a_{k+1}a_k$ is to be interpreted as ϵ.

(2) If $(d,q',0,w)$ is in $\delta(q,a_i,Z_j)$ for $1 \le i \le k$, then
$(q,a_i \ldots a_k, yZ\uparrow) \vdash (q', a_{i+d} \ldots a_k, yw\uparrow)$.

The relation $\vdash$ completely describes the atomic acts of a one-way sa. Condition (1) allows one-way motion on the input tape and two-way reading of the stack. Restrictions (i) and (ii) prevent A from going off either end of the stack. The convention $a_{k+1} = \epsilon$ allows the automaton to leave the right end of the input tape. Condition (2) permits the device to write on the right end of the stack, where writing ϵ is actually erasing.

Note that A "blocks" if it tries to write in the interior of the stack.

The notation for describing a sequence of movements of A is now given.

Definition. For each x,y in $(\Sigma \cup \{\phi,\$\})^*$, and w_1, w_2, w_1', w_2' in Γ^*; let

$$(q,xy,w_1\uparrow w_2) \vdash^* (q',y,w_1'\uparrow w_2')$$

if there exist $k \ge 0$, f_i, g_i, and h_i for $0 \le i \le k$ such that $f_0 = q$, $g_0 = xy$, $h_0 = w_1\uparrow w_2$, $f_k = q'$, $g_k = y$, $h_k = w_1'\uparrow w_2'$, and $(f_i,g_i,h_i) \vdash (f_{i+1}, g_{i+1}, h_{i+1})$ for $0 \le i < k$.

The final states in a one-way sa are used to "accept" a set of words by the following procedure.

Definition. A word x in Σ^* is accepted by a one-way sa A if

$$(q_0,\phi x\$,Z_0\uparrow) \vdash^* (q,\epsilon,w_1\uparrow w_2)$$

for some q in F and some w_1, w_2 in Γ^*. The set of words accepted by A, denoted by T(A), is called a one-way sa language (abbreviated language).

We now specialize the model to be "deterministic" in nature, i.e., for each (q,a,Z) in $K \times (\Sigma \cup \{\phi,\$\}) \times \Gamma$, there is to be one and only one "next move" which is possible.

Definition. A deterministic one-way sa is a one-way sa $A = (K,\Sigma,\phi,\$,\Gamma,\delta,q_0,Z_0,F)$ with the following properties:

(α) $\#(\delta(q,a,Z)) = 1$[7] for each (q,a,Z) in $K \times (\Sigma \cup \{\phi,\$\}) \times \Gamma$.

(β) If[8] $\delta(q,a,Z_0) = (d,q',e,y)$ then $y = Z_0 w$ for some w in Γ^*.

Condition (β) prevents the stack from being emptied. (For the leftmost stack symbol is always Z_0).

Definition. A language L is called a deterministic language, abbreviated D-language, if there is a deterministic one-way sa A such that T(A) = L.

An important specialization of a one-way sa is now noted. By restricting the stack to function as a pushdown store, i.e., to be read only at the right end, we obtain a pushdown automaton. Specifically, a pushdown automaton (pda) is a one-way sa with the restriction that (d,q',e,w) in $\delta(q,a,Z)$ implies $e = 0$.[9]

To obtain the family of deterministic pushdown automata, we need only start from a one-way deterministic sa. Specifically, a deterministic pushdown automaton is a one-way sa in which $\delta(q,a,Z) = (d,q',e,w)$ implies $e = 0$.[9]

In this paper, we shall be concerned with one-way sa and languages. We were led to these concepts by specializing to the one-way case the stack automata and languages accepted by stack automata as discussed in [3]. A practical reason for studying one-way sa is that ALGOL can be recognized by these devices. [To see this observe that ALGOL would be accepted by a pushdown automaton if there were no declaration statements. As was noted in [3], a stack automaton can search symbol tables and thus recognize declaration statements. A one-way sa can search symbol tables as follows. It guesses that it is reading an identifier on the input tape, it has its stack pointer move left into the stack, and then it guesses that it has found the proper entry in the proper symbol table.] As is well-known, ALGOL cannot be entirely recognized by a pushdown automaton. Thus sets accepted by one-way sa may be better approximations to currently used programming languages than sets recognized by pushdown automata.

Section 2. Closure Properties of Languages

There are a number of operations which have proved to be important in the theory of context free languages.[10] In this section, many of these operations are shown to preserve languages.

---

[7] For any set E, #(E) is the number of elements in E.

[8] We write $\delta(q,a,Z) = (d,q',e,y)$ instead of $\delta(q,a,Z) = \{(d,q',e,y)\}$.

[9] The comparison is not quite obvious since a one-way sa has endmarkers while pushdown automata customarily do not have endmarkers. Furthermore, there is a slight distinction between "ϵ-moves" in pushdown automata and moves in one-way sa in which d = 0. It is not difficult to prove that the families of languages accepted by these two-types of devices are identical.

[10] Context free languages may be characterized as those sets accepted by pushdown automata.

The first major result on operations concerns the intersection of a language and a regular set.[11]

Theorem 2.1. If L is a language and R is a regular set, then $L \cap R$ is a language.

Corollary 1. L-R is a language for each language L and each regular set R.

By modifying the proof of Theorem 2.1 slightly, we obtain

Corollary 2. If L is a D-language and R is a regular set, then $L \cap R$ and L-R are D-languages.

We now consider the basic operations of union, product, and *.

Theorem 2.2. The family of languages is closed under union, product, and *.

Theorem 2.2 can also be obtained as a corollary of the following result.

Theorem 2.3. Let $L \subseteq \Sigma^*$ be a context free language. For each a in Σ, let Σ_a be a finite set and $L_a \subseteq \Sigma_a^*$ be a language. Then

$\{w_1 \cdots w_n / n \geq 1,\ a_{i(1)} \cdots a_{i(n)}$ in Σ, w_j in $L_{a_{i(j)}}$, $a_{i(1)} \cdots a_{i(n)}$ in $L\}$

is a language.

We now introduce a transformation device which preserves languages.

Definition. A sequential transducer is a 5-tuple $S = (K,\Sigma,\Delta,H,s_o)$ where

(i) K, Σ, and Δ are finite nonempty sets (of states, inputs, and outputs resp).

(ii) s_o is in K (the start state).

(iii) H is a finite subset of $K \times \Sigma^* \times \Delta^* \times K$.

The sequential transducer transforms words as follows:

Definition. Let $S = (K,\Sigma,\Delta,H,s_o)$ be a sequential transducer. For each u in Σ^*, let S(u) be the set of words v with the property that there exist words $u_1,\ldots,u_k$ in Σ^*, $v_1,\ldots,v_k$ in Δ^*, and $s_1,\ldots,s_k$ in K such that $u = u_1 \cdots u_k$, $v = v_1 \cdots v_k$, and $(s_i,u_{i+1},v_{i+1},s_{i+1})$ is in H for each $0 \leq i < k$. For each $U \subseteq \Sigma^*$, let $S(U) = \bigcup_{u \text{ in } U} S(u)$.

Theorem 2.4. S(L) is a language for each sequential transducer S and each language L.

A number of important special cases follow from the theorem on sequential transducers.

Definition. A generalized sequential machine (abbreviated gsm) is a 6-tuple $S = (K,\Sigma,\Delta,\delta,\lambda,q_o)$, where

(i) K, Σ, and Δ are finite nonempty sets (of states, inputs, and outputs respectively).

(ii) δ is a function from $K \times \Sigma$ into K (next state function).

(iii) λ is a function from $K \times \Sigma$ into Δ^* (output function).

(iv) q_o is in K (start state).

δ and λ are extended to $K \times \Sigma^*$ by letting $\delta(q,\varepsilon) = q$, $\lambda(q,\varepsilon) = \varepsilon$, $\delta(q,xa) = \delta[\delta(q,x),a]$, and $\lambda(q,xa) = \lambda(q,x)\lambda(\delta(q,x),a)$ for each (q,x,a) in $K \times \Sigma^* \times \Sigma$.

Theorem 2.5. If $S = (K,\Sigma,\Delta,\delta,\lambda,q_o)$ is a gsm and L is a language, then $S(L) = \{\lambda(q_o,x)/x \text{ in } L\}$ is a language.

Languages are also preserved under inverse gsms.

Theorem 2.6. If $S = (K,\Sigma,\Delta,\delta,\lambda,q_o)$ is a gsm and L is a language, then

$S^{-1}(L) = \{x \text{ in } \Sigma^*/\lambda(q_o,x) \text{ in } L\}$ is a language.

From Theorems 2.5 and 2.6 there immediately follows

Corollary. If h is a homomorphism[12] and L is a language, then both h(L) and $h^{-1}(L)$ are languages.

---

[11] A finite state automaton, abbreviated fsa, is a 5-tuple $A = (K,\Sigma,\delta,p_1,F)$, where K and Σ are finite nonempty sets (of states and inputs respectively), δ is a function from $K \times \Sigma$ into K, p_1 is in K, and $F \subseteq K$. The function δ is extended to $K \times \Sigma^*$ by defining $\delta(q,\varepsilon) = q$ and $\delta(q,xa) = \delta[\delta(q,x),a]$ for each (q,x,a) in $K \times \Sigma^* \times \Sigma$. A set $R \subseteq \Sigma^*$ is said to be regular if there exists an fsa A such that $R = T(A)$, where $T(A) = \{x/\delta(p_1,x) \text{ in } F\}$.

[12] A homomorphism is a mapping h from Σ^* into Δ^* such that $h(\varepsilon) = \varepsilon$ and $h(a_1 \cdots a_k) = h(a_1) \cdots h(a_k)$, each a_i in Σ.

Another interesting operation which preserves languages is "quotient by a regular set."

Theorem 2.7. Let L be a language and R a regular set. Then
$L/R = \{w/wy \text{ in } L \text{ for some } y \text{ in } R\}$
and $L\backslash R = \{w/yw \text{ in } L \text{ for some } y \text{ in } R\}$
are languages.

Corollary. If L is a language, then so are
$\text{Init}(L) = \{u/uv \text{ in } L \text{ for some } v \text{ in } \Sigma^*\}$,
$\text{Fin}(L) = \{v/uv \text{ in } L \text{ for some } u \text{ in } \Sigma^*\}$,
and $\text{Sub}(L) = \{v/uvw \text{ in } L \text{ for some } u,v \text{ in } \Sigma^*\}$.

The family of languages is closed under reversal.[13]

Theorem 2.8. If L is a language, then so is L^R.

Section 3. Operations Which Do Not Preserve Languages

In this section we exhibit certain operations which do not preserve languages. This is done with the help of a representation theorem for recursively enumerable sets.

Definition. A phrase structure grammar is a 4-tuple $G = (V,\Sigma,P,\sigma)$, where (i) V is a finite nonempty set, (ii) $\Sigma \subseteq V$, (iii) P is a finite set of ordered pairs (u,v), written $u \to v$, with u in $(V-\Sigma)^*-\{\epsilon\}$ and v in V^*, and (iv) σ is in $V-\Sigma$.

A phrase structure grammar selects a certain set of words as follows.

Notation. Let $G = (V,\Sigma,P,\sigma)$ be a phrase structure grammar. For w,y in V^*, write $w \Rightarrow y$ if there exist z_1,z_2,u,v such that $w = z_1uz_2$, $y = z_1vz_2$, and $u \to v$ is in P. For w,y in V^*, write $w \overset{*}{\Rightarrow} y$ if there exist $z_0,\ldots,z_r$ such that $z_0 = w$, $z_r = y$, and $z_i \Rightarrow z_{i+1}$ for $0 \le i < r$. Let L(G), called the set "generated" by G, be the set $\{x \text{ in } \Sigma^*/\sigma \overset{*}{\Rightarrow} x\}$.

The following proposition is well-known [1]: "For $L \subseteq \Sigma^*$, L = L(G) for some phrase structure grammar G if and only if L is recursively enumerable.[14]

We now have a representation theorem for recursively enumerable sets.

Theorem 3.1. For each recursively enumerable set $E \subseteq \Sigma^*$, there exist deterministic context free languages[15] L_1,L_2, and a homomorphism h such that $E = h(L_1 \cap L_2)$.

Using the representation theorem, several non-closure results are now obtained. In the next section, other non-closure results are obtained from the representation theorem.

Theorem 3.2. (a) The family of languages does not contain the intersection of some pairs of deterministic context free languages and does not contain the complement of some context free languages.

(b) The family of languages is not closed under intersection or complementation.

In Theorem 4.1, we shall see that the complement of a D-language is a D-language. By (a) of Theorem 3.2, languages are not closed under complementation. Thus we have

Corollary. There exists a context free language, thus a language, which is not a D-language.

Section 4. Closure Properties of D-languages

We now treat some basic closure properties of D-languages.

The first result we have is

Theorem 4.1. The complement of a D-language is a D-language.

Corollary. For each D-language L and regular set R, $R \cup L$ and R-L are D-languages.

Another important operation which preserves D-languages is the inverse gsm mapping.

Theorem 4.2. For each D-language L and each gsm S,
$S^{-1}(L) = \{w|S(w) \text{ in } L\}$
is a D-language.

The next two results show that the operations of eliminating fixed initial or terminal subwords preserve D-languages.

---

[13] Let E be an abstract set. Let $\epsilon^R = \epsilon$ and $(a_1\ldots a_t)^R = a_t\ldots a_1$, each a_i in E. For $U \subseteq E^*$, let $U^R = \{w^R/w \text{ in } U\}$. U^R is called the reversal of U.

[14] See [2] for a definition and a discussion of recursively enumerable sets.

[15] A set L is said to be a deterministic context free language if $L = T(A)$ for some deterministic pda.

Theorem 4.3. Let $w = a_1 \ldots a_n$, $n \geq 1$ be a word in $\Sigma^* - \{\epsilon\}$. If L is a D-language, then $B_w(L) = \{x/wx \text{ in } L\}$ is a D-language.

Theorem 4.4. Let $w = a_1 \ldots a_n$, $n \geq 1$, be a word in $\Sigma^* - \{\epsilon\}$. If L is a D-language, then $\{x/xw \text{ in } L\}$ is also a D-language.

A number of operations which do not preserve D-languages are now presented.

Theorem 4.5. The family of D-languages is not closed under (i) union, (ii) product, (iii) *, and (iv) homomorphism.[16]

Section 5. Decision Problems

We now consider the decidability of various questions.

Turning to solvability results, we have

Theorem 5.1. It is recursively solvable to determine whether T(A) is empty for an arbitrary one-way sa A.

The next solvability result concerns D-languages.

Theorem 5.2. It is recursively solvable to determine for an arbitrary D-language L and a regular set R whether L = R.

Turning to unsolvability results, we have

Theorem 5.3. It is recursively unsolvable to determine whether an arbitrary language is (a) context free, (b) regular, (c) a D-language.

We close with the following open problem:

(*) Is it recursively solvable to determine for an arbitrary one-way sa A whether T(A) is finite?

[16] (iv) implies non-closure under gsm mappings.

Given an arbitrary language $L \subseteq \Sigma^*$, let h be the homomorphism defined by $h(a) = \bar{a}$ for each a in Σ. Then L is finite if and only if h(L) is finite. Let $M(L) = \text{Init}(h(L))$. By the corollary to Theorem 2.7, M(L) is a language. Clearly L is finite if and only if M(L) is finite. [M(L) has the following interesting properties: (i) It is regular. (ii) It is finite if and only if it does not coincide with a^*, i.e., if and only if $a^* - M(L) = \emptyset$.] Thus (*) can be reduced to the following problem:

(**) Is it recursively solvable to determine whether T(A) is finite for an arbitrary one-way sa A, over a one-letter alphabet, with the property that $\text{Init}(T(A)) = T(A)$?

References

[1] Chomsky, N. "On Certain Formal Properties of Grammars," Information and Control, Vol. 2, 1959, pp. 137-167.

[2] Davis, M. "Computability and Unsolvability," McGraw-Hill Book Company, New York, 1958.

[3] Ginsburg, S., Greibach, S., and Harrison, M. A. "Stack Automata and Compiling," to appear in the Journal of the Association for Computing Machinery.

[4] Ginsburg, S., Greibach, S., and Harrison, M. A. "One-way Stack Automata," System Development Corporation Report No. TM-738/025/00, 22 April 1966, 58 pp.

[5] Ginsburg, S., Greibach, S., and Harrison, M. A. "On the Relation between One-way Stack Languages and Context Sensitive Languages," System Development Corporation report (in preparation).