Nested Stack Automata

ALFRED V. AHO

Bell Telephone Laboratories, Inc., Murray Hill, New Jersey

ABSTRACT. A new type of automaton, called a nested stack automaton, is introduced. The nested stack automaton provides a mathematical model for a restricted form of embedded list structure. Special cases of nested stack automata include many of the important machine models studied in automata theory, such as pushdown automata and stack automata.

The fundamental properties of languages accepted by nested stack automata are investigated. It is shown that the class of languages accepted by one-way nondeterministic nested stack automata is the same as the class of indexed languages.

KEY WORDS AND PHRASES: automata, formal languages, indexed languages, pushdown automata, stack automata, acceptors, balloon automata, closure properties, decidability results, list storage

CR CATEGORIES: 5.21, 5.22

1. *Introduction*

A new kind of memory structure, called a nested stack, is introduced. A nested stack provides a mathematical model for a restricted form of embedded list structure. The computational power of this structure is investigated through the vehicle of an abstract machine, called a nested stack automaton, which consists of a finite-state machine with a nested stack as the main auxiliary memory.

Special cases of a nested stack automaton include the pushdown automaton and the recently introduced stack automaton [7, 8]. It is shown that nested stack automata have less computational capability than Turing machines and are, in fact, recursive. The family of nested stack automata is equivalent to one of the largest known closed classes of balloon automata which define recursive sets.

A language can be finitely specified as the set of finite length strings of symbols either generated by some generative device, as a grammar, or accepted by some recognitive device, as an abstract machine. It is shown that the class of indexed languages [1] is exactly that class of sets accepted by one-way nondeterministic nested stack automata. It is also shown that a characterization for one-way nondeterministic stack automata which can read the stack in only one direction is provided by a restricted class of indexed languages. Several of the fundamental properties of the classes of languages defined by nested stack automata are also presented.

Recently, another automaton model, called a list-storage acceptor (lsa), was introduced [9]. The lsa is an automaton model with the auxiliary memory organized as a linear list. The unrestricted lsa is capable of defining exactly all the recursively enumerable sets. A temporal restriction on the input operation of lsa is used to limit the definitional power of lsa to a subset of the recursive sets. A nested stack automaton, on the other hand, achieves recursiveness by means of a spatial restriction on the access of information from the memory structure.

2. *Pushdown-Oriented Memory Structures*

We consider here a number of memory organizations consisting of a single list of symbols from some finite alphabet. In the list, one symbol, called the *active symbol*, is distinguished. The active symbol can be read and, depending on the type of memory structure, the list can be altered in one of several ways during one memory move.

A pushdown list is a simple type of memory structure consisting of a single list of symbols in which the symbol on the top of the list is the active symbol. In one memory move the active symbol can be read and replaced by a finite length string of symbols. If the active symbol is replaced by ϵ, the string of 0 length, then the top symbol is erased from the list. In such a structure, information can be accessed only on a last-in first-out basis. Pushdown lists have wide application in the construction of compilers and other language processing devices.

However, in certain situations a simple pushdown list is not a sufficiently powerful memory structure, by itself, to permit the recognition and processing of all syntactic structures found in some programming languages, such as ALGOL.

A memory structure consisting of a single pushdown list can be made more powerful by the addition of an ability to access any symbol previously written on the list. This access can be done by means of a pointer which can travel up and down the pushdown list one symbol at a time. The active symbol in this structure is the symbol referenced by this pointer. In this structure two types of move are possible.

1. If the active symbol is the top symbol of the list, then the active symbol may be read and replaced by a finite length string of symbols in a single move.

2. The active symbol may be read and the pointer may be moved up one symbol, kept stationary, or moved down one symbol. However, the pointer cannot be moved up past the top symbol on the list nor below the bottom symbol on the list.

Such a memory structure is called a *stack*.

A *nested stack* is a yet more powerful memory structure obtained by permitting stacks to be embedded or nested within stacks to arbitrary depths. A nested stack is a single list of symbols with one pointer referencing the active symbol. Each stack within this list is delimited by a top-of-stack and bottom-of-stack marker. This structure can be altered in a single move in one of four different ways.

1. The active symbol can be read and a new stack with a finite number of stack symbols can be created and embedded between the active symbol and the symbol immediately below it. The top symbol in the new stack becomes the new active symbol. (This is the stack creation mode.)

2. If a stack becomes empty, denoted by an active symbol which is both the top symbol of a stack and a bottom-of-stack marker, then this empty stack can be removed from the list. The symbol immediately above that empty stack becomes the new active symbol. (This is the stack destruction mode.)

3. If the active symbol is the top symbol of a stack, then the active symbol may be read and replaced by a finite length string of symbols. (This is the pushdown mode.)

4. The active symbol may be read and the pointer may be moved up one symbol, kept stationary, or moved down one symbol. The pointer cannot be moved up past the top symbol of a stack but it can be moved down below the bottom of any stack except the outermost. (This is the stack reading mode.)

Notation. The following symbolic conventions are observed with regard to a nested stack structure. The symbol \$ is appended to the left of the top symbol of each stack. In each stack, the top symbol together with \$ is treated as a single symbol. The symbol # is used to denote the end of the list and the bottom of the outermost stack. The symbol ¢ is used to mark the bottom of each embedded stack. The entire list is written with its top symbol on the left.

Example. Let L_1, L_2, and L_3 be the three lists of symbols $a_1a_2L_2L_3a_3$, b, and c, respectively. These lists can be represented in a single list as the nested stack structure:

$$\$a_1\ a_2\ \$b\ ¢\ \$\hat{c}\ ¢\ a_3\ \#. \tag{1}$$

We have chosen c to be the active symbol, as indicated by the pointer "^". The following are examples of the four types of nested stack moves.

(i) A new stack containing the symbols d_1d_2 can be embedded below c as in (2).

$$\$a_1\ a_2\ \$b\ ¢\ \$c\ \$\hat{d}_1\ d_2\ ¢\ ¢\ a_3\ \# \tag{2}$$

(ii) In (1) the symbol c can be replaced by the empty string to obtain (3).

$$\$a_1\ a_2\ \$b\ ¢\ \$\hat{¢}\ a_3\ \# \tag{3}$$

(iii) In (3) the pointer can be moved down (to the right) one symbol to obtain (4).

$$\$a_1\ a_2\ \$b\ ¢\ \$¢\ \hat{a}_3\ \# \tag{4}$$

(iv) In (3) the empty stack can be removed to obtain (5).

$$\$a_1\ a_2\ \$b\ \hat{¢}\ a_3\ \# \tag{5}$$

Notice that the only way to move up (to the left) from the top symbol of an embedded stack is to first delete all symbols from that stack and then remove the empty stack.

3. *Definition of Nested Stack Automaton*

An automaton model can be constructed by attaching a finite-state control to some type of memory structure and a read-only input tape as in Figure 1. The memory structure determines the general classification of the machine. For example, if the memory structure is a pushdown list, then the resulting machine is a pushdown automaton, or if the memory is a stack structure, then the machine becomes a stack automaton. Here we are interested in an abstract machine in which the memory structure consists of a nested stack. Such a machine is called a *nested stack automaton.*

Four distinct types of machine can be associated with a particular memory structure. The type of machine is distinguished by

(1) the direction of motion of the input head on the input tape—if the input head never moves left on any move, the machine is said to be *one-way*; otherwise, the machine is *two-way*;

(2) whether the finite control ever has a choice for its next move—if a choice ever exists, then the machine is *nondeterministic*; if no choice ever arises, the machine is *deterministic.*

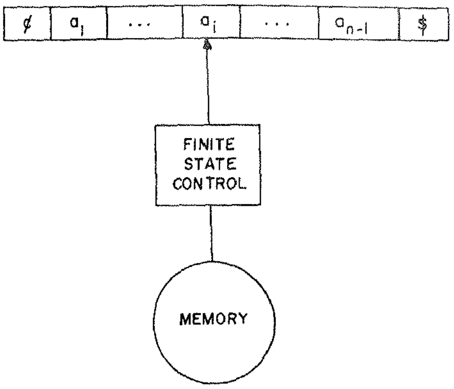

FIG. 1. An abstract machine

Thus four classes of machines may be associated with a particular memory structure, since a machine can be one-way deterministic (1D), one-way nondeterministic (1N), two-way deterministic (2D), or two-way nondeterministic (2N). The set of these four classes of machines is called the *family* of machines associated with that memory structure.

The computational power of a particular kind of memory structure can be measured in terms of the four classes of sets accepted by each of the four classes of abstract machines in the family of machines having that kind of memory structure for auxiliary storage.

Several of the well-known grammatically defined classes of languages also enjoy a representation in terms of the class of sets accepted by some class of abstract machines. For example, a context-free language can be specified as either the set generated by a context-free grammar or the set accepted by some 1N pushdown automaton. There are several advantages to such a dual representation. For example, some properties of a class of languages might be more discernible from a machine representation than a grammatical characterization. Moreover, a one-way machine can be considered a model of a single-pass recognizer for the language it defines. Since recognition speed is of concern in practical situations, the deterministic machines often define an important subset of the class of languages defined by the nondeterministic machines; such a subset may not be obvious from a grammatical definition.

We are now ready to present the definition of a nested stack automaton (nsa). Intuitively, a nested stack automaton consists of an input tape, a finite-state control, and a storage tape whose structure is that of a nested stack (see Figure 2). Formally, a nested stack automaton S is specified by the following quantities:

(1) Q, Σ, and Γ, which are finite nonempty sets of *states, inputs*, and *storage* symbols, respectively

(2) special symbols \$, ¢, and #, which we assume are not in either Σ or Γ

(i) \$ is used exclusively as both the right endmarker on the input tape and as a top-of-stack marker on the storage tape.

(ii) ¢ is the left endmarker on the input tape and a bottom-of-embedded-stack marker on the storage tape.

(iii) # is the endmarker for the storage tape.

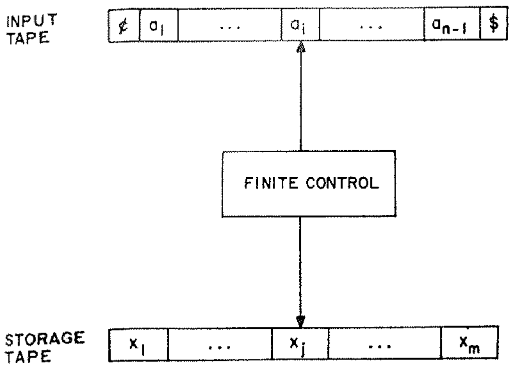

FIG. 2. Nested stack automaton

(3) δ, a (finite-control) mapping from,[1]
  (i) in the pushdown mode,
    $Q \times \Sigma' \times \$\Gamma$ into finite subsets[2] of $Q \times D \times \$\Gamma^*$
  (ii) in the stack reading mode,
    (a) $Q \times \Sigma' \times \Gamma'$ into subsets of $Q \times D \times D$
    (b) $Q \times \Sigma' \times \$\Gamma'$ into subsets of $Q \times D \times \{1\}$
    (c) $Q \times \Sigma' \times \{\#\}$ into subsets of $Q \times D \times \{-1\}$
  (iii) in the stack creation mode,
    $Q \times \Sigma' \times (\Gamma' \cup \$\Gamma')$ into finite subsets of $Q \times D \times \$\Gamma^*¢$
  (iv) in the stack destruction mode,
    $Q \times \Sigma' \times \{\$¢\}$ into subsets of $Q \times D$
(4) q_0, a designated symbol in Q, representing the *initial state*
(5) Z_0, a designated symbol in Γ, representing the *initial storage symbol*
(6) F, a subset of Q, representing *final states*

The nested stack automaton S is denoted by $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F, \$, ¢, \#)$ or $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ whenever the symbols $, ¢, and # are understood.

An instantaneous description of S can be indicated by means of a quadruple of the form $(q, a_0a_1 \cdots a_n, i, X_1 \cdots \hat{X}_j \cdots X_m)$, with $n, m \geq 1$, called a *configuration* of S. Here the following conditions hold.

1. q is in Q and represents the state of the finite control.

2. $a_0 \cdots a_i \cdots a_n$ is the content of the input tape. If S is two-way, $a_0 = ¢$. If S is one-way, $a_0 = \epsilon$. (The left endmarker is obviously of no use on a one-way machine.) Each a_i, $0 < i < n$, is in Σ and $a_n = \$$. The string of symbols $a_1 \cdots a_{n-1}$ is called the input string.

3. The third component, i, is the position of the input head on the input tape, indicating that S is currently scanning the ith symbol on the input tape. If S is two-way, $0 \leq i \leq n$. If S is one-way, $1 \leq i \leq n$.

4. $X_1 \cdots X_j \cdots X_m$ is the string of symbols on the storage tape with X_i in $\Gamma' \cup \$\Gamma'$, $1 < i < m$. X_j is the active symbol (symbol under the storage tape head). If $m > 1$, then $X_1 = \$Z$ with Z in Γ and $X_m = \#$. If $m = 1$, then $X_m = \$\#$.

Intuitively, S makes a move in the following fashion. Depending on the values of
  (i) q, the current state of the finite control,

[1] Here $\Sigma' = \Sigma \cup \{¢, \$\}$, $\Gamma' = \Gamma \cup \{¢\}$, and $D = \{-1, 0, 1\}$.
[2] Γ^* is the set of all finite length strings of symbols from Γ (including ϵ, the string of length 0).

(ii) a, the current input symbol, and
(iii) X, the current active memory symbol.

S consults the mapping δ encoded as the finite control and chooses one value in the set $\delta(q, a, X)$. This value then determines

(i) the new state of the finite control (the first component of $\delta(q, a, X)$),
(ii) the direction in which the input head is to move (the second component of $\delta(q, a, X)$), and
(iii) the type of memory move (determined by the form of the third component of $\delta(q, a, X)$). (The four possible types of memory moves are similar to those outlined for a nested stack structure.)

If for every argument, $\delta(q, a, X)$ contains at most one element, then S is deterministic. Otherwise, S is nondeterministic. If the second component of $\delta(q, a, X)$ is always 0 or +1, then S is one-way. Otherwise, S is two-way. Thus S may be a 1D nsa, 1N nsa, 2D nsa, or 2N nsa. Wherever the name "nested stack automaton" is used, it is used generically as a name for any one of the four types of machine in the family of nested stack automata.

Formally, a relation $\vdash_S$ (or $\vdash$ whenever S is clear) is defined over the set of configurations to model a move by S. We write:

(i) $(q, a_0 \cdots a_n, i, \alpha\$\hat{A}Y\beta) \vdash (p, a_0 \cdots a_n, i + d, \alpha\$\hat{Z}_1 \cdots Z_kY\beta)$ if it is true that $\delta(q, a_i, \$A)$ contains[3] $(p, d, \$Z_1 \cdots Z_k)$, $k \geq 0$.

(ii) $(q, a_0 \cdots a_n, i, X_1 \cdots \hat{X}_j \cdots X_m) \vdash (p, a_0 \cdots a_n, i + d, X_1 \cdots \hat{X}_{j+d'} \cdots X_m)$ if $\delta(q, a_i, X_j)$ contains (p, d, d'). Note that if X_j is of the form $\$Y$, Y in Γ', then $d' = 1$, or if $X_j = \#$, then $d' = -1$.

(iii) $(q, a_0 \cdots a_n, i, X_1 \cdots \hat{X}_j \cdots X_m) \vdash (p, a_0 \cdots a_n, i + d, X_1 \cdots X_j \$\hat{A}_1 \cdots A_k ¢X_{j+1} \cdots X_m)$, $j < m$, if $\delta(q, a_i, X_j)$ contains $(p, d, \$A_1 \cdots A_k¢)$.

(iv) $(q, a_0 \cdots a_n, i, X_1 \cdots X_{j-1}\$\hat{¢}\ X_{j+1} \cdots X_m) \vdash (p, a_0 \cdots a_n, i + d, X_1 \cdots \hat{X}_{j-1} X_{j+1} \cdots X_m)$ if $\delta(q, a_i, \$¢)$ contains (p, d).

We must further specify that in (i)-(iv) above, $0 \leq i + d \leq n$ if S is two-way and $1 \leq i + d \leq n$ if S is one-way. (Intuitively, the input head cannot move left from the left endmarker nor right from the right endmarker.)

We extend the relation $\vdash$ in the conventional manner.

1. $Q \vdash^0 Q$ where Q is any configuration of S.
2. $Q_1 \vdash^{k+1} Q_3$ if $Q_1 \vdash^k Q_2$ and $Q_2 \vdash Q_3$, where Q_1, Q_2, Q_3 are each configurations of S, and $k \geq 0$.
3. $Q_1 \vdash^* Q_2$ if $Q_1 \vdash^i Q_2$ for some $i \geq 0$.

$\vdash^*$ is the reflexive and transitive closure of $\vdash$ over the set of configurations and is used to represent a sequence of moves. A sequence of moves by S will often be called a *scan*.

An input string $w = a_1 \cdots a_{n-1}$ in Σ^* can be accepted by S in one of two modes:

1. If S can go from the initial configuration $(q_0, a_0a_1 \cdots a_n, 1, \$\hat{Z}_0\#)$ by some sequence of moves into a configuration in which the input head is at the right endmarker and the finite control is in a final state, then w is *accepted by final state* by S.

2. If by some sequence of moves S can go from its initial configuration to a configuration of the form $(q, ¢w\$, n, \$\hat{\#})$ for any q in Q, then w is *accepted by empty stack* by S.

[3] If $k = 0$, then a configuration of the form $(q, ¢w\$, i, \alpha\$\hat{Z}_1 \cdots Z_kY\beta)$, $Y \neq \epsilon$, is the configuration $(q, ¢w\$, i, \alpha\$\hat{Y}\beta)$. This convention is used throughout.

The set of words accepted by final state by S and the set of words accepted by empty stack by S are denoted $T(S)$ and $N(S)$, respectively.

LEMMA 3.1. *Given a 1D (1N, 2D, or 2N) nested stack automaton S, a 1D (1N, 2D, or 2N, respectively) nsa S' can be effectively constructed from S such that $N(S') = T(S)$, and conversely.*

PROOF. S' initially writes a special symbol on the bottom of the stack tape just above #. S' then simulates the behavior of S, and if S' reaches a configuration in which S is in a final state of S and if the input head is over the right endmarker, S' goes into a special state which causes S' to empty the stack tape and halt.

The formal details are left to the reader. The proof of the converse is similar and is omitted here.

Lemma 3.1 states that for a class of nsa of a given type, acceptance by final state and empty stack both define the same class of sets. Thus henceforth, without any loss of generality, we can consider that a given nested stack automaton S accepts on empty stack. The set $N(S)$ will be called the nested stack automaton language defined by S.

We should also point out that the left endmarker on the input tape is not needed by either a 2N or 2D nsa. This can be readily seen by noting that given a two-way (2N or 2D) nsa S, an equivalent two-way (2N or 2D, respectively) nsa S' with no left endmarker on the input tape can be effectively constructed from S such that $N(S') = N(S)$. (The definition of $N(S')$ for S' should be obvious.) S' operates by first copying a reference string equal in length to the input string on the input tape on the bottom of its storage tape just above the storage tape marker #. S' then proceeds to simulate the behavior of S. S' has a special state to indicate that S is reading the left endmarker. Also, before S moves its input head to the left, S' first creates a vacuous stack to mark the current position of the storage tape head on the storage tape. S' then consults the reference string on the bottom of the storage tape to make sure its input head will not move off the left end of the input tape. Then S' returns its storage tape head to the vacuous stack, which is erased. Finally, S' simulates the move made by S. If S' enters a configuration indicating that S has erased all symbols from its storage tape, S' then enters another special state, which causes S' to erase the reference string from its storage tape. The details of such a construction are left to the reader.

Using a slight variant of the argument in [17], it can also be shown that the right endmarker on the input tape does not enlarge the class of sets accepted by 1D and 1N nsa accepting by final state. This remark, however, does not hold for 1D nsa accepting by empty stack, since any language accepted by empty stack by a 1D nsa without a right endmarker must have the prefix property.[4]

4. *Related Machines*

A nested stack automaton is a very powerful device, and special cases of nsa include many of the important types of abstract machines studied in automata theory.

A pushdown automaton (pda) is a nsa $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ in which δ is a mapping having only the pushdown mode form. One-way nondeterministic pushdown automata are particularly important in the theory of context-free languages [4, 5].

[4] A language L has the prefix property if xy in L, $y \neq \epsilon$, implies that x is not in L.

Moreover, 2D pda are capable of recognizing many syntactic features found in modern programming languages which are not recognizable by a 1N pda [2].

A stack automaton (sa) is a nsa $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ in which δ is a mapping having only the pushdown mode and the stack reading mode forms. Stack automata were introduced by Ginsburg, Greibach, and Harrison to model certain computations occurring in the process of compilation [7].

The family of sa defines several interesting classes of languages. A 1N sa is capable of recognizing some syntactic features in algorithmic programming languages which are not recognizable by a 1N pda [8]. It has been shown that if L is a 1N sa language, then L is recognizable by a deterministic, linear-bounded automaton [16]. A 2D sa can recognize a language which is not context-sensitive [19] and in fact can simulate a nondeterministic $n \log n$ tape-bounded Turing machine [18].

An interesting subclass of sa are the nonerasing stack automata [7]. A nonerasing stack automaton (nesa) is a sa $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ in which δ is a mapping having the stack reading mode form as well as the form

$$\delta: \quad Q \times \Sigma' \times \$\Gamma \rightarrow \text{finite subsets}^5 \text{ of } Q \times D \times \$\Gamma^+$$

(a nesa is never permitted to erase a symbol written on the stack). A 2D nesa can simulate a linear-bounded automaton [7], and thus the class of languages accepted by 2D nesa includes all context-sensitive languages. In fact, it has been shown that the class of languages recognized by 2D nesa is equivalent to that recognized by deterministic $n \log n$ tape-bounded Turing machines [15]. Moreover, the class of languages recognized by 2N nesa is equivalent to that recognized by nondeterministic n^2 tape-bounded Turing machines [15].

Greibach has shown that 1N sa languages are not closed under substitution. We will see that the 1N nsa languages are equivalent to the indexed languages which are closed under substitution. This then implies that a 1N nsa is strictly more powerful than a 1N sa [12]. At present it is not known whether a 2N nsa is more powerful than a 2N sa.

A nsa is restricted in that it is not able to access information on its storage tape above any embedded stack without first erasing that embedded stack. Notice that if this restriction is removed, the nsa becomes equivalent to a Turing machine in recognitive capability.

Closure properties and decidability results for languages defined by nsa resemble those for languages defined by pda and sa of the same type. Recently, an approach toward unifying results of this nature through the use of closed classes of automaton models, called balloon automata, has been proposed by Hopcroft and Ullman [13, 14].

Essentially, a balloon automaton is an abstract machine in which the memory structure is represented by Z, the set of positive integers. The extraction of a finite amount of information from the memory structure on each move by a balloon automaton is represented by a recursive function from Z to some finite subset of Z. Changing the content of the memory is represented by a partial recursive function from Z and the set of states of the finite control to Z.

In this theory, a closed class of balloon automata, rather than the balloon automaton per se, is the more significant concept. Loosely speaking, a set C of balloon automata forms a closed class if

[5] $\Gamma^+ = \Gamma^* - \{\epsilon\}$.

(1) C contains those balloon automata which manipulate the balloon in only trivial ways (i.e. those automata which always extract or store the same information on each move);
(2) two automata M_1 and M_2 are in C implies that machine M_3 is also in C provided that
   (i) M_3 can distinguish two memory configurations only if either M_1 or M_2 can also distinguish between these memory configurations, and
   (ii) M_3 cannot alter its memory in any manner different from the way in which M_1 or M_2 can change its memory.

Rigorous definitions of "balloon automaton" and "closed class of balloon automata" are somewhat complicated [13] and are not given here. While our definition of a nested stack automaton is not in terms of the balloon automata formulation, it should be clear that all that is required for this is an arithmetization of the nested stack structure and a recoding of the mapping δ. It should then be clear that each of the four classes of nested stack automata is equivalent to a closed class of balloon automata. These details are left for the interested reader. Rather than reformulating proofs for theorems about languages defined by closed classes of balloon automata, we use these theorems directly to obtain closure properties and decidability results for the class of languages defined by each class of nested stack automata.

Balloon automata were introduced primarily to model those classes of automata which are abstract machines with restrictions on the type of local memory manipulations. For example, each of the four classes of machines in the family of pushdown automata, stack automata, nonerasing stack automata, and Turing machines can be formulated as a closed class of balloon automata.

However, a family of automata which is defined as a class of abstract machines in which the number of moves made by the machine or the number of available memory cells is functionally related to the length of the input string cannot be directly formulated as a closed class of balloon automata. In fact, the class of linear-bounded automata is not equivalent to any closed class of one-way nondeterministic balloon automata,[6] and the class of on-line deterministic 2^n tape-bounded Turing machines is not equivalent to any closed class of one-way deterministic balloon automata.[7]

5. *1N NSA and Indexed Languages*

In [1] a new type of grammar, called an indexed grammar, was introduced. The language generated by an indexed grammar is called an indexed language. In this section it is shown that the class of sets accepted by 1N nsa is exactly the same as the class of indexed languages.

An *indexed grammar* G is a 5-tuple (N, T, F, P, S) in which:
(1) N is a finite nonempty set of *nonterminal* symbols
(2) T is a finite set of *terminal* symbols
(3) F is a finite set each element of which is a finite set of ordered pairs of the form (A, ω) where A is in N and ω is in $(N \cup T)^*$. An element f in F is

[6] A set accepted by a lba is recursive but the emptiness problem is not solvable for ba. The emptiness problem is solvable for all closed classes of one-way balloon automata which define recursive sets [12].
[7] The class of languages defined by 2^n tape-bounded Turing machines is not closed under inverse homomorphism.

called an index. An ordered pair (A, ω) in f is written $A \to \omega$ and is called an index production.

(4) P is a finite set of *productions* of the form $A \to \alpha$ where A is in N and α is in $(NF^* \cup T)^*$.

(5) S is a distinguished symbol in N.

A relation $\xrightarrow[G]{}$ (or $\to$ whenever G is understood) is defined over $(NF^* \cup T)^*$ as follows. We say α *directly generates* β in G, written $\alpha \xrightarrow[G]{} \beta$, if either:[8]

(1) $\alpha = \gamma A\zeta\delta$, $A \to X_1\eta_1X_2\eta_2 \cdots X_k\eta_k$ is a production in P where $\eta_i = \epsilon$ if X_i is in T, and $\beta = \gamma X_1\theta_1X_2\theta_2 \ldots X_k\theta_k\delta$ where, for $1 \le i \le k$,

$$\theta_i = \begin{cases} \eta_i\zeta & \text{if } X_i \text{ is in } N, \\ \epsilon & \text{if } X_i \text{ is in } T, \end{cases}$$

or

(2) $\alpha = \gamma Af\zeta\delta$, f contains the index production $A \to X_1 \cdots X_k$, and $\beta = \gamma X_1\theta_1 \cdots X_k\theta_k\delta$ where, for $1 \le i \le k$,

$$\theta_i = \begin{cases} \zeta & \text{if } X_i \text{ is in } N, \\ \epsilon & \text{if } X_i \text{ is in } T. \end{cases}$$

In both (1) and (2) above, the nonterminal A is said to be expanded. The relation $\to$ is extended to $\xrightarrow{*}$ in the following manner:

(1) $\alpha \xrightarrow{0} \alpha$.

(2) If $\alpha \xrightarrow{k-1} \beta$ and $\beta \to \gamma$, then $\alpha \xrightarrow{k} \gamma$, for $k \ge 1$.

(3) $\alpha \xrightarrow{*} \beta$ if and only if $\alpha \xrightarrow{i} \beta$ for some $i \ge 0$.

The relation $\xrightarrow{*}$ is the reflexive and transitive closure of $\to$ over $(NF^* \cup T)^*$.

A sequence of strings $\alpha_0, \alpha_1, \cdots, \alpha_n$ such that $\alpha_{i-1} \to \alpha_i$, $1 \le i \le n$, is called a *derivation* of α_n from α_0 (in G). If in each string α_i, $0 \le i < n$, the leftmost nonterminal is expanded to directly derive α_{i+1}, then this derivation is a *leftmost* derivation.

The *language* generated by the indexed grammar $G = (N, T, F, P, S)$ is denoted $L(G)$ and is the set $\{w \text{ in } T^* \mid S \xrightarrow[G]{*} w\}$.

In [1] it was shown that given any indexed grammar $G = (N, T, F, P, S)$ an equivalent indexed grammar $G' = (N', T, F', P', S)$ could be effectively constructed from G such that

(1) $L(G') = L(G)$;

(2) each index production in each index in F' is of the form $A \to B$ where A and B are in N'; and

(3) each production in P' is of one of the forms $A \to BC$, $A \to a$, or $A \to Bf$ where A is in N', B and C are in $N' - \{S\}$, a is in T, and f is in F'. If ϵ is in $L(G)$, then $S \to \epsilon$ is in P'.

A grammar of this form is called a *normal form* indexed grammar.

THEOREM 5.1. *Given a normal form indexed grammar $G = (N, T, F, P, S)$, a 1N nsa M can be effectively constructed from G such that*[9] $N(M) = L(G)$.

[8] For an indexed grammar $G = (N, T, F, P, S)$, to avoid unduly long quantifiers we make the following symbolic conventions:
(1) A, B, C are in N. (2) a, b, c are in T. (3) f is in F. (4) X is in $N \cup T$.
(5) ω is in $(N \cup T)^*$. (6) ζ, η, θ are in F^*. (7) $\alpha, \beta, \gamma, \delta$ are in $(NF^* \cup T)^*$.

[9] The normal form is used merely to make the proof notationally simpler.

Proof. M is constructed to be capable of nondeterministically simulating a leftmost derivation of any word w in $L(G)$. As each terminal symbol is generated, it is compared with the current input symbol. M accepts the input string if and only if there exists a derivation of the input string from S in G.

M considers each nonterminal symbol appearing on the top of a stack to be indexed by all indices appearing between that nonterminal and the end of the storage tape. An expansion of a nonterminal using an index production from an index is simulated by creating a new stack just below that index. In simulating a derivation in this fashion, M does not need to duplicate strings of indices on its storage tape.

We can assume that T does not contain the symbols \$, ¢, and #. Let $Q = \{q\} \cup \{q_A \mid A \text{ is in } N\}$ and $\Gamma = N \cup T \cup F$. Let $M = (Q, T, \Gamma, \delta, q, S, \varphi)$ where δ is constructed in the following manner for all a in T.

1. For each production in P of the form $A \to BC$, $A \to b$ with b in T, and $A \to Bf$, $\delta(q, a, \$A)$ contains $(q, 0, \$BC)$, $(q, 0, \$b)$, and $(q, 0, \$Bf)$, respectively.
2. $\delta(q, a, \$a)$ contains $(q, 1, \$)$.
3. For all A in N, $\delta(q, a, \$A)$ contains $(q_A, 0, \$)$.
4. For all A in N and Y in $N \cup \{¢\}$,
   (i) $\delta(q_A, a, \$Y)$ contains $(q_A, 0, 1)$, and
   (ii) $\delta(q_A, a, Y)$ contains $(q_A, 0, 1)$.
5. For all A in N and f in F such that f contains an index production of the form $A \to B$,
   (i) $\delta(q_A, a, \$f)$ contains $(q, 0, \$B¢)$, and
   (ii) $\delta(q_A, a, f)$ contains $(q, 0, \$B¢)$.
6. $\delta(q, a, \$¢)$ contains $(q, 0)$.
7. For all Y in $N \cup F \cup \{¢\}$, $\delta(q, a, Y)$ contains $(q, 0, -1)$.
8. $\delta(q, a, \$f)$ contains $(q, 0, \$)$.
9. If and only if $S \to \epsilon$ is in P does $\delta(q, \$, \$S)$ contain $(q, 0, \$)$.

Rule 1 simulates a derivation by expanding a nonterminal by means of a production. Rule 2 compares, symbol by symbol, the terminal string being derived on the working tape with that on the input tape. Rules 3–5 are used to expand a nonterminal A by using an index production of the form $A \to B$ from the first index below A on the storage tape. Notice that if the first index below A does not contain an index production of the form $A \to B$, then A cannot be expanded by means of an index production. Rules 6 and 7 are used to return the storage tape head to the symbol at the top of a stack. Rule 8 erases indices appearing at the top of a stack. Rule 9 is used to take care of the case where ϵ is in $L(G)$.

The following two lemmas describe in detail the manner in which M simulates a derivation in G.

Lemma 5.1. *If*

$$Af_1 \cdots f_k \xrightarrow{n} a_1 \cdots a_m \tag{6}$$

is a valid leftmost derivation in G, with $k \geq 0$, $m \geq 1$, and A in N, then (7) is a valid scan by M, for $n \geq 1$, $Z\beta_1\beta_2 \cdots \beta_{k+1}$ in $(N \cup \{¢\})^*$, α in $(N \cup F \cup \{\$, ¢\})^*$.

$$\begin{aligned}(q, a_1 \cdots a_m\$, 1, \alpha\$\hat{A}Z\beta_1f_1\beta_2f_2 \cdots \beta_kf_k\beta_{k+1}\#) \\ \vdash^* (q, a_1 \cdots a_m\$, m+1, \alpha\$\hat{Z}\beta_1f_1\beta_2f_2 \cdots \beta_kf_k\beta_{k+1}\#).\end{aligned} \tag{7}$$

Proof. The proof proceeds by induction on n. For convenience, we let $\gamma = \beta_1f_1 \cdots \beta_kf_k\beta_{k+1}\#$ and $\zeta = f_1 \cdots f_k$.

If $n = 1$, then (6) is of the form $A\zeta \to a$ where a is in T. From the construction of δ, scan (7) is then of the form:

$$\begin{aligned}(q,\ a\$,\ 1,\ \alpha\$\hat{A}Z\gamma) &\vdash (q,\ a\$,\ 1,\ \alpha\$\hat{a}Z\gamma)\\ &\vdash (q,\ a\$,\ 2,\ \alpha\$\hat{Z}\gamma)\end{aligned}$$

Suppose Lemma 5.1 is true for all $n < n'$ with $n' > 1$. Consider the derivation $A\zeta \xrightarrow{n'} a_1 \cdots a_m$ where $m \geq 1$ and A is in N. Such a derivation can begin by expanding A in one of three ways:

(i) $A\zeta \to B\zeta C\zeta \xrightarrow{n_1} a_1 \cdots a_i\, C\zeta \xrightarrow{n_2} a_1 \cdots a_i a_{i+1} \cdots a_m$
where n_1 and n_2 are both less than n'.

(ii) $A\zeta \to Bf\zeta \xrightarrow{n_1} a_1 \cdots a_m$
where $n_1 < n'$.

(iii) $Af_1 \cdots f_k \to Bf_2 \cdots f_k \xrightarrow{n_1} a_1 \cdots a_m$
where f_1 contains the index production $A \to B$ and $n_1 < n'$.

From the construction of δ and the inductive hypothesis, the following are valid scans of the form (7) by M.

$$\begin{aligned}
\text{(i)}\quad (q, w\$, 1, \alpha\$\hat{A}Z\gamma) &\vdash (q,\ \$,\ 1,\ \alpha\$\hat{B}CZ\gamma)\\
&\vdash^* (q, w\$, i+1, \alpha\$\hat{C}Z\gamma)\\
&\vdash^* (q, w\$, m+1, \alpha\$\hat{Z}\gamma)\\
\text{(ii)}\quad (q, w\$, 1, \alpha\$\hat{A}Z\gamma) &\vdash (q, w\$, 1, \alpha\$\hat{B}fZ\gamma)\\
&\vdash^* (q, w\$, m+1, \alpha\$\hat{f}Z\gamma)\\
&\vdash (q, w\$, m+1, \alpha\$\hat{Z}\gamma)\\
\text{(iii)}\quad (q, w\$, 1, \alpha\$\hat{A}Z\gamma) &\vdash (q_A,\ w\$,\ 1,\ \alpha\$\hat{Z}Y_1 \cdots Y_j f_1\gamma')\\
&\qquad\qquad (Y_1 \cdots Y_j = \beta_1)\\
&\vdash (q_A, w\$, 1, \alpha\$Z\hat{Y}_1 \cdots Y_j f_1\gamma')\\
&\vdash^* (q_A, w\$, 1, \alpha\$ZY_1 \cdots Y_j\hat{f}_1\gamma')\\
&\vdash (q, w\$, 1, \alpha\$ZY_1 \cdots Y_j f_1\$\hat{B}\not{c}\gamma')\\
&\vdash^* (q, w\$, m+1, \alpha\$ZY_1 \cdots Y_j f_1\$\hat{\not{c}}\gamma')\\
&\vdash (q, w\$, m+1, \alpha\$ZY_1 \cdots Y_j\hat{f}_1\gamma')\\
&\vdash (q, w\$, m+1, \alpha\$ZY_1 \cdots \hat{Y}_j f_1\gamma')\\
&\vdash^* (q, w\$, m+1, \alpha\$\hat{Z}\gamma)
\end{aligned}$$

Here $w = a_1 \cdots a_m$, and $\gamma' = \beta_2 f_2 \cdots \beta_k f_k \beta_{k+1}\#$.

Lemma 5.2 is the converse of Lemma 5.1.

LEMMA 5.2. *If (8) is a valid scan by M, for any* $Z\beta_1\beta_2 \cdots \beta_{k+1}$ *in* $(N \cup \{\not{c}\})^*$, α *in* $(N \cup F \cup \{\$, \not{c}\})^*$, *and* $m \geq 1$,

$$\begin{aligned}(q,\ a_1 \cdots a_m\$,\ 1,\ \alpha\$\hat{A}Z\beta_1 f_1 \cdots \beta_k f_k \beta_{k+1}\#)&\\ \vdash^n (q,\ a_1 \cdots a_m\$,\ m+1,\ \alpha\$\hat{Z}\beta_1 f_1 \cdots \beta_k f_k\beta_{k+1}\#)&\qquad (8)\end{aligned}$$

then (9) is a valid derivation in G, for all $n \geq 1$.

$$Af_1 \cdots f_k \xrightarrow{*} a_1 \cdots a_m . \tag{9}$$

PROOF. The details of a proof by induction on n are very similar to those in the proof of the previous lemma and are left for the reader.

From Lemmas 5.1 and 5.2 we have $S \xrightarrow{*} w$, $w = a_1 \cdots a_{n-1}$, $n \geq 2$, if and only if $(q, w\$, 1, \$\hat{S}\#) \vdash^* (q, w\$, n, \$\hat{\#})$. If and only if ϵ is in $L(G)$, (from construction 9) we have $(q, \$, 1, \$\hat{S}\#) \vdash (q, \$, 1, \$\hat{\#})$. Thus $L(G) = N(M)$ and the proof of Theorem 5.1 is complete.

THEOREM 5.2. *Given a 1N nsa $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varphi)$, an indexed grammar $G = (N, \Sigma, F, P, S)$ can be effectively constructed from M such that $L(G) = N(M)$.*

PROOF. We can assume without loss of generality that M writes at most two symbols on a stack on any one move in either the pushdown mode or the stack creation mode. The method of proof is to simulate the behavior of M accepting a word w in $N(M)$ by means of a derivation of w from S in G.

The behavior of M at the top of a stack is represented by a nonterminal in N of the form (p, Y, q) where p and q are in Q and Y is in $\Gamma \cup \{¢\}$. The nonterminal (p, Y, q) is capable of generating all input strings in Σ^* which cause M to go from state p to state q erasing the symbol Y from the top of a stack.

The behavior of M in the stack reading and stack creation modes is simulated by means of index productions within the indices. A nonterminal of the form $(p, Y, q)'$ derives all strings in Σ^* which cause M, starting in state p and scanning the storage symbol Y, to go down into the storage tape and then return in state q to this symbol Y for the first time. A nonterminal of the form $(p, Y, q)''$ derives strings in Σ^* which cause M, starting in state p and scanning the symbol Y, to go down into the storage tape and return to Y in state q without ever reading any symbols above Y in the stack tape but perhaps reading Y several times before finally returning to Y in state q.

Let $N = \{(p, X, q), (p, X, q)', (p, X, q)'', (p, \$X, q)', (p, \$X, q)'' \mid p$ and q are in Q, X is in $\Gamma \cup \{¢\}\} \cup \{(p, A, q)_{B¢}, (p, A, q)_{¢} \mid p$ and q are in Q, A and B are in $\Gamma\}$.

$$F = \{f_U, f_{\$X} \mid U \text{ is in } \Gamma \cup \{¢, \#\}, X \text{ is in } \Gamma \cup \{¢\}\}.$$

The set of productions P is constructed as follows.[10] For convenience, we denote by the symbol d_a, where a is in $\Sigma \cup \{\$\}$, the symbol a if $d = 1$ and ϵ if $d = 0$.

1. If $\delta(p, a, \$A)$ contains (i) $(q, d, \$BC)$, (ii) $(q, d, \$B)$, or (iii) $(q, d, \$)$, then P contains, respectively, the productions
   (i) $(p, A, r) \to d_a(q, B, s)f_C(s, C, r)$ for all r and s in Q,
   (ii) $(p, A, r) \to d_a(q, B, r)$ for all r in Q, and
   (iii) $(p, A, q) \to d_a$.
2. If $\delta(p, a, X)$ contains $(q, d, 0)$, then P contains $(p, X, q)' \to d_a$.
3. If (i) $\delta(p, a, X)$, or (ii) $\delta(p, a, \$Y)$ contains $(r, d, +1)$ and $\delta(s, b, Z)$ contains $(q, d', -1)$, then
   (i) f_X contains the index production

$$(p, X, q)' \to d_a(r, Z, s)'' \, d_b',$$

[10] To avoid unduly long quantifiers, unless otherwise stated, the following symbolic conventions will be observed in this proof: (1) p, q, r, s are in Q; (2) a, b, c are in Σ; (3) A, B, C are in Γ; (4) X, Y, Z are in $\Gamma' = \Gamma \cup \{¢\}$; and (5) U is in $\Gamma' \cup \{\#\}$.

or

(ii) $f_{\$Y}$ contains the index production

$$(p, \$Y, q)' \to d_a(r, Z, s)''\, d_b',$$

respectively. Also, the index f_Z contains the index production $(s, Z, s)' \to \epsilon$.

4. P contains

(i) $(p, U, r)'' \to (p, U, q)'(q, U, r)''$

and

(ii) $(p, U, r)'' \to (p, U, r)'$

for all p, q, r in Q, and U in $\Gamma \cup \{¢, \#\}$.

5. If $\delta(p, a, X)$ or $\delta(p, a, \$X)$ contains (i) $(q, d, \$AB¢)$, (ii) $(q, d, \$A¢)$, or (iii) $(q, d, \$¢)$, then the index (a) f_X, or (b) $f_{\$X}$ contains, respectively, the following index productions for all r_1, r_2, and s in Q:

(a) (i) $(p, X, s)' \to d_a(q, A, r_1)_{B¢}(r_1, B, r_2)_{¢}(r_2, ¢, s)$,
(ii) $(p, X, s)' \to d_a(q, A, r_1)_{¢}(r_1, ¢, s)$,
(iii) $(p, X, s)' \to d_a(q, ¢, s)$.
(b) (i) $(p, \$X, s)' \to d_a(q, A, r_1)_{B¢}(r_1, B, r_2)_{¢}(r_2, ¢, s)$,
(ii) $(p, \$X, s)' \to d_a(q, A, r_1)_{¢}(r_1, ¢, s)$,
(iii) $(p, \$X, s)' \to d_a(q, A, s)$.

6. P contains the productions

$$(q, A, r)_{B¢} \to (q, A, r) f_B f_¢\,,$$

$$(q, A, r)_¢ \to (q, A, r) f_¢\,,$$

for all q and r in Q, A and B in Γ.

7. P contains the productions

$$(p, Y, q) \to (p, \$Y, r)' f_{\$Y}(r, Y, q)$$

for all p, q, r in Q, Y in Γ'.

8. If $\delta(p, a, \$¢)$ contains (q, d), then P contains $(p, ¢, q) \to d_a$.

9. Finally, P contains the starting productions $S \to (q_0, Z_0, p) f_\#$ for all p in Q.

Productions in 1 simulate the behavior of M at the top of a stack. Productions in 3 and 4 and index productions in 3 simulate M in the stack reading mode. Productions in 6 and index productions in 5 model the stack creation mode. In 7 the productions represent a transition from the pushdown mode into either the stack reading mode or stack creation mode. The productions in 8 represent the effect of destroying a stack.

Lemmas 5.3 and 5.4 complete the proof of Theorem 5.2. Lemma 5.3 describes the manner in which a derivation in G simulates the behavior of M.

LEMMA 5.3. *If (10), (11), and (12) below are valid scans by M for any value of $n \geq 1$, then (13), (14), and (15) below are all valid derivations in G. We assume here that $w_1 = a_1 \cdots a_{n_1-1}$, $w_2 = b_1 \cdots b_{n_2-1}$, and $w_3 = c_1 \cdots c_{n_3-1}$.*

$$(p_1, w_1\$, 1, \alpha_1\$\hat{A}X_1 \cdots X_i\#) \vdash^n (q_1, w_1\$, m_1 + 1, \alpha_1\$\hat{X}_1 \cdots X_i\#),$$
$$1 \leq (m_1 + 1) \leq n_1, \quad (10)$$

without reading any symbols in α_1.

$$(p_2\,,\,w_2\$,\,1,\,\alpha_2\$\hat{X}Y_1\cdots Y_j\#)\vdash^n(q_2\,,\,w_2\$,\,m_2+1,\,\alpha_2\$\hat{X}Y_1\cdots Y_j\#),\quad 1\le(m_2+1)\le n_2\,, \tag{11}$$

with the first move not in the pushdown mode and without reading any part of α_2 or returning to $\$X$ except on the last move.

$$(p_3\,,\,w_3\$,\,1,\,\alpha_3\hat{Z}_1\cdots Z_k\#)\vdash^n(q_3\,,\,w_3\$,\,m_3+1,\,\alpha_3\hat{Z}_1\cdots Z_k),\quad 1\le(m_3+1)\le n_3\,, \tag{12}$$

without reading any part of α_3 or returning to Z_1 except on the last move.

$$(p_1\,,A,\,q_1)f_{X_1}\cdots f_{X_i}f_{\#}\xrightarrow{*}a_1\cdots a_{m_1}\,.^{11} \tag{13}$$

$$(p_2\,,\,\$X,\,q_2)'f_{\$X}f_{Y_1}\cdots f_{Y_j}f_{\#}\xrightarrow{*}b_1\cdots b_{m_2}\,. \tag{14}$$

$$(p_3\,,Z_1\,,\,q_3)'f_{Z_1}\cdots f_{Z_k}f_{\#}\xrightarrow{*}c_1\cdots c_{m_3}\,. \tag{15}$$

PROOF. The proof proceeds by induction on n. For notational convenience assume the following.

$$\beta_1=X_1\cdots X_i\# \qquad \zeta_1=f_{X_1}\cdots f_{X_i}f_{\#}$$
$$\beta_2=Y_1\cdots Y_j\# \qquad \zeta_2=f_{Y_1}\cdots f_{Y_j}f_{\#}$$
$$\zeta_3=f_{Z_1}\cdots f_{Z_k}f_{\#}$$

If $n=1$, from the definition of a nsa there can be no scan of the form (11). Corresponding to scans (10) and (12) with $n=1$ we have the following derivations in G:

$$(p_1\,,A,\,q_1)\zeta_1\to a$$

(from construction 1 (iii)) where $a=a_1$ if $m_1=1$ and $a=\epsilon$ otherwise,

$$(p_3\,,Z_1\,,\,q_3)'\zeta_3\to c$$

(from construction 2) where $c=c_1$ if $m_3=1$ and $c=\epsilon$, otherwise.

Now suppose that Lemma 5.3 is true for all values of $n<n'$ where $n'>1$. Consider scans of the form (10), (11), and (12) in which $n=n'$. Scan (10) can have one of the following two forms:

1. (a) First move by M is in the pushdown mode with M replacing A by two (or one) symbols on the top of the stack. (The situation where A is replaced by one symbol will not be shown here.)

$$\begin{aligned}(p_1\,,\,w_1\$,\,1,\,\alpha_1\$\hat{A}\beta_1)&\vdash(r_1\,,\,w_1\$,\,1+d',\,\alpha_1\$\hat{B}C\beta_1)\\&\vdash^{n_1}(r_2\,,\,w_1\$,\,m'+1,\,\alpha_1\$\hat{C}\beta_1)\\&\vdash^{n_2}(q_1\,,\,w_1\$,\,m_1+1,\,\alpha_1\$\hat{X}\cdots X_i\#)\end{aligned}$$

where both n_1 and n_2 are less than n' and

$$(m_1+1)\ge(m'+1)\ge(1+d')\ge1\,.$$

[11] We define $a_i\cdots a_m=\epsilon$ if $m<i$.

2. (a) First move by M is in either the stack reading mode or the stack creation mode. M then causes its stack head to return to $\$A$.

$$(p_1, w_1\$, 1, \alpha_1\$\hat{A}\beta_1) \vdash^{n_1} (r_1, w_1\$, j_1 + 1, \alpha_1\$\hat{A}\beta_1)$$

(without reading α_1 or returning to $\$A$ except on the last move)

$$\vdash^{n_2} (q_1, w_1\$, j_2 + 1, \alpha_1\$\hat{X}_1X_2 \cdots X_i\#)$$

where n_1 and n_2 are both less than n', $n_1 > 0$, and $(j_2 + 1) \geq (j_1 + 1) \geq 1$.

Scan (11) can have one of the following two forms:

3. (a) On the first move M moves its stack head into the stack in the stack reading mode.

$$\begin{aligned}(p_2, w_2\$, 1, \alpha_2\$\hat{X}\beta_2) &\vdash (r_1, w_2\$, 1 + d', \alpha_2\$X\hat{Y}_1\beta_2')\\ &\vdash^{n_1} (s_1, w_2\$, j_1 + 1, \alpha_2\$X\hat{Y}_1\beta_2')\\ &\vdash^{n_2} (s_2, w_2\$, j_2 + 1, \alpha_2\$X\hat{Y}_1\beta_2')\\ &\quad\vdots\\ &\vdash^{n_m} (s_m, w_2\$, j_m + 1, \alpha_2\$X\hat{Y}_1\beta_2')\\ &\vdash (q_2, w_2\$, j_m + 1 + d'', \alpha_2\$\hat{X}\beta_2)\end{aligned}$$

where

$$\beta_2' = Y_2 \cdots Y_j\#, \; 0 < n_i < n' \quad \text{for} \quad 1 \leq i \leq m,$$

and $(j_m + 1 + d'') \geq (j_m + 1) \geq (j_{m-1} + 1) \geq \cdots \geq (j_1 + 1) \geq (1 + d') \geq 1$.

4. (a) On the first move M creates a new stack under $\$X$ with two (one or zero) symbols on this stack.

$$\begin{aligned}(p_2, w_2\$, 1, \alpha_2\$\hat{X}\beta_2) &\vdash (r_1, w_2\$, 1 + d', \alpha_2\$X\$\hat{A}B\text{¢}\beta_2)\\ &\vdash^{n_1} (r_2, w_2\$, j_1 + 1, \alpha_2\$X\$\hat{B}\text{¢}\beta_2)\\ &\vdash^{n_2} (s_1, w_2\$, j_2 + 1, \alpha_2\$X\$\hat{\text{¢}}\beta_2)\\ &\vdash^{l_1} (s_2, w_2\$, k_1 + 1, \alpha_2\$X\$\hat{\text{¢}}\beta_2)\\ &\vdash^{l_2} (s_3, w_2\$, k_2 + 1, \alpha_2\$X\$\hat{\text{¢}}\beta_2)\\ &\quad\vdots\\ &\vdash^{l_m} (s_{m+1}, w_2\$, k_m + 1, \alpha_2\$X\$\hat{\text{¢}}\beta_2)\\ &\vdash (q_2, w_2\$, k_m + 1 + d'', \alpha_2\$\hat{X}\beta_2)\end{aligned}$$

where n_1, n_2, and l_i, $1 \leq i \leq m$, are each less than n' and greater than zero, and $(k_m + 1 + d'') \geq (k_m + 1) \geq (k_{m-1} + 1) \geq \cdots \geq (k_1 + 1) \geq (j_2 + 1) \geq (j_1 + 1) \geq (1 + d') \geq 1$.

Scan (12) can take on only the same form as scan (11). (These forms are not repeated here.)

Corresponding to forms 1(a) and 2(a) of scan (10), from the construction of G and the inductive hypothesis we have the leftmost derivations 1(b) and 2(b), respectively, in G.

$$\begin{aligned} 1.\ \text{(b)}\ (p_1, A, q_1)\zeta_1 &\rightarrow d_a'(r_1, B, r_2)f_C\zeta_1(r_2, C, q_1)\zeta_1 \\ &\xrightarrow{*} d_a'x_1(r_2, C, q_1)\zeta_1 \\ &\xrightarrow{*} d_a'x_1x_2 \end{aligned}$$

where $a = a_1$, $x_1 = a_{1+d'} \cdots a_{m'}$ and $x_2 = a_{m'+1} \cdots a_{m_1}$.

$$\begin{aligned} 2.\ \text{(b)}\ (p_1, A, q_1)\zeta_1 &\rightarrow (p_1, \$A, r_1)'f_{\$A}\zeta_1(r_1, A, q_1)\zeta_1 \\ &\xrightarrow{*} x_1(r_1, A, q_1)\zeta_1 \\ &\xrightarrow{*} x_1x_2 \end{aligned}$$

where $x_1 = a_1 \cdots a_{j1}$ and $x_2 = a_{j_1+1} \cdots a_{j_2}$.

The following derivations correspond to forms 3(a) and 4(a) of scan (11).

$$\begin{aligned} 3.\ \text{(b)}\ (p_2, \$X, q_2)'f_{\$X}\zeta_2 &\rightarrow a(r_1, Y_1, s_m)''\zeta_2 b \\ &\rightarrow a(r_1, Y_1, s_1)'\zeta_2(s_1, Y_1, s_m)''\zeta_2 b \\ &\xrightarrow{*} ax_1(s_1, Y_1, s_m)''\zeta_2 b \\ &\vdots \\ &\xrightarrow{*} ax_1 \cdots x_{m-2}(s_{m-1}, Y_1, s_m)''\zeta b \\ &\rightarrow ax_1 \cdots x_{m-2}(s_{m-1}, Y_1, s_m)'\zeta b \\ &\rightarrow ax_1 \cdots x_{m-2}x_{m-1}b \end{aligned}$$

with

$$a = d'_{b_1}, \qquad b = d''_{b_{i_m+1}}, \qquad x_1 = b_{1+d'} \cdots b_{j_1}, \qquad x_i = b_{j_i+1} \cdots b_{j_{i+1}}$$

for $1 < i < m$.

$$\begin{aligned} 4.\ \text{(b)}\ (p_2, \$X, q_2)'f_{\$X}\zeta_2 &\rightarrow a(r_1, A, r_2)_{B\text{¢}}\zeta_2(r_2, B, s_1)_{\text{¢}}\zeta_2(s_1, \text{¢}, q_2)\zeta_2 \\ &\rightarrow a(r_1, A, r_2)f_B f_{\text{¢}}\zeta_2(r_2, B, s_1)_{\text{¢}}\zeta_2(s_1, \text{¢}, q_2)\zeta_2 \\ &\xrightarrow{*} ax_1(r_2, B, s_1)_{\text{¢}}\zeta_2(s_1, \text{¢}, q_2)\zeta_2 \\ &\rightarrow ax_1(r_2, B, s_1)f_{\text{¢}}\zeta_2(s_1, \text{¢}, q_2)\zeta_2 \\ &\xrightarrow{*} ax_1x_2(s_1, \text{¢}, q_2)\zeta_2 \\ &\rightarrow ax_1x_2(s_1, \$\text{¢}, s_2)'f_{\$\text{¢}}\zeta_2(s_2, \text{¢}, q_2)\zeta_2 \\ &\vdots \\ &\xrightarrow{*} ax_1x_2y_1 \cdots y_m(s_{m+1}, \text{¢}, q_2)\zeta_2 \\ &\rightarrow ax_1x_2y_1 \cdots y_m b \end{aligned}$$

where

$$a = d'_{b_1}, \qquad b = d''_{b_{k_m+1}}, \qquad x_1 = b_{1+d'} \cdots b_{j_1},$$

$$x_2 = b_{j_1+1} \cdots b_{j_2}, \qquad y_1 = b_{j_2+1} \cdots b_{k_1}, \qquad y_i = b_{k_{i-1}+1} \cdots b_{k_i}$$

for $1 < i \leq m$.

The derivations corresponding to scan (12) are similar to those for scan (11) and so are not shown here.

The next lemma is the converse of Lemma 5.3.

LEMMA 5.4. *If (16), (17), and (18) are derivations in G, then (19), (20), and (21) are valid scans by M, for all values of n.*

$$(p_1, A, q_1)f_{X_1} \cdots f_{X_i} f_{\#} \xrightarrow{n} a_1 \cdots a_{m_1}. \tag{16}$$

$$(p_2, \$X, q_2)' f_{\$X} f_{Y_1} \cdots f_{Y_j} f_{\#} \xrightarrow{n} b_1 \cdots b_{m_2}. \tag{17}$$

$$(p_3, Z_1, q_3)' f_{Z_1} \cdots f_{Z_k} f_{\#} \xrightarrow{n} c_1 \cdots c_{m_3}. \tag{18}$$

$$(p_1, a_1 \cdots a_{m_1}\$, 1, \alpha_1\$\hat{A}X_1 \cdots X_i\#) \vdash^* (q_1, a_1 \cdots a_{m_1}\$, m_1 + 1, \alpha_1\$\hat{X}_1 \cdots X_i\#), \tag{19}$$

without reading any part of α_1.

$$(p_2, b_1 \cdots b_{m_2}\$, 1, \alpha_2\$\hat{X}Y_1 \cdots Y_j\#) \vdash^* (q_2, b_1 \cdots b_{m_2}\$, m_2 + 1, \alpha_2\$\hat{X}Y_1 \cdots Y_j\#), \tag{20}$$

without reading any part of α_2 or returning to $\$X$ except on the last move.

$$(p_3, c_1 \cdots c_{m_3}\$, 1, \alpha_3\hat{Z}_1 \cdots Z_k\#) \vdash^* (q_3, c_1 \cdots c_{m_3}\$, m_3 + 1, \alpha_3\hat{Z}_1 \cdots Z_k\#), \tag{21}$$

without reading any part of α_3 or returning to Z_1 except on the last move.

PROOF. The proof is again by induction on n. However, the details are similar to those for the proof of Lemma 5.3 and so are left to the reader.

From Lemmas 5.3 and 5.4 we have that $S \xrightarrow{*} (q_0, Z_0, p) \xrightarrow{*} a_1 \cdots a_{n-1}$ if and only if

$$(q_0, a_1 \cdots a_{n-1}\$, 1, \$\hat{Z}_0\#) \vdash^* (p, a_1 \cdots a_{n-1}\$, n, \$\hat{\#})$$

for some p in Q. Thus $L(G) = N(M)$.

Summarizing the results of Theorems 5.1 and 5.2, we have the following characterization for indexed languages.

THEOREM 5.3. *L is an indexed language if and only if L is accepted by some 1N nested stack automaton.*

6. *Properties of NSA Languages*

Each of the four classes of nsa languages has closure properties characteristic of a closed class of the same type of balloon automata. These properties can be found in [13] and are not restated here.

An *abstract family of languages* (AFL) is a class of languages closed under the operations of union, concatenation, $+$, ϵ-free homomorphism, inverse homomorphism, and intersection with regular sets [6]. A *full AFL* is an AFL closed under arbitrary homomorphism.

Using the properties of indexed languages, closed classes of balloon automata, and AFLs, a number of additional fundamental properties of nsa languages can be derived. In [1] it is shown that the class of indexed languages is a full AFL,[12] is

[12] Any class of languages accepted by a closed class of 1N balloon automata possesses all properties of a full AFL [11]. In [6] it is shown that a class of languages is a full AFL if and only if the class is countable and accepted by a class of accepting devices, called an *abstract family of acceptors* (AFA). An AFA in many aspects is similar to a closed class of 1N balloon automata. It should be clear to the reader familiar with AFA that the class of 1N nsa can also be formulated as an example of an AFA.

closed under reversal and full substitution, and is not closed under intersection or complement. Theorem 5.3 implies that the class of 1N nsa languages also has these properties.

We quote a theorem from [10].

THEOREM 6.1. *If the class of languages accepted by a closed class of 2N balloon automata is closed under length-preserving homomorphism, then that class of languages is an AFL.*

THEOREM 6.2. *The class of languages accepted by 2N nsa is closed under length-preserving homomoprhism.*

PROOF. Only an intuitive argument is outlined. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varphi)$ be a 2N nsa and let h be a length-preserving homomorphism from Σ into Σ' such that $h(a) = a'$ for each a in Σ. A 2N nsa M' can be constructed in the following manner from M such that $N(M') = h(N(M))$.

Suppose M' has as input the string $y = a_1' \cdots a_{n-1}'$, each a_i' in Σ'. The first sequence of moves M' executes consists of nondeterministically writing just above the endmarker $\#$ on the storage tape a string of symbols $bx = ba_1 \cdots a_{n-1}$ such that $h(x) = y$. Here b is a new symbol used as a special marker.

M' then proceeds to simulate the behavior of M with x as input. M' uses the position of its input head on $a_1' \cdots a_{n-1}'$ merely as a counter to determine the input symbol being read by M. To simulate a move of M, M' first creates a vacuous stack to mark the location of its storage tape head on the storage tape. M' then consults the string $a_1 \cdots a_{n-1}$ written on the bottom of its storage tape to determine the input symbol being read by M. M' uses the position of its input head to locate this symbol by starting with its storage tape head at b and its input head at a_i' (assuming M is currently reading the input symbol a_i). M' then simultaneously moves its input head to the left one symbol and its storage tape head down (to the right) one symbol until M' is reading ¢ on the input. At this point the storage tape head is at a_i. M' then stores a_i in its finite-state control. Next, M' moves its input head to a_i' by going through this procedure in reverse.

Finally, M' locates the current position in the storage tape by moving its storage tape head to the vacuous stack previously created and then destroying this stack. Then M' simulates a move by M.

M' is capable of executing all sequences of moves that M can make. Finally, M' is constructed to accept the input y if and only if M accepts x. Thus $N(M') = h(N(M))$.

The details of the construction are left for the reader.

The next theorem follows immediately.

THEOREM 6.3. *The class of languages accepted by 2N nsa is an AFL.*

We suspect that Theorem 6.3 may also hold for 2D nsa languages.

For each family of languages $\mathcal{L}$, there are three decidability questions of considerable interest. Let L and L' be any two languages in $\mathcal{L}$. We may ask the following questions:

(1) Is L empty?

(2) Given w, an arbitrary string of symbols over a finite alphabet, is w in L?

(3) Is L the same as L'?

These three questions are commonly known as the emptiness, membership, and equivalence problems, respectively, for a family of languages. The representation of the family of languages can in some cases influence the solvability or unsolva-

TABLE I. DECIDABILITY RESULTS FOR NSA LANGUAGES

	Class of NSA Languages			
Question	1D	1N	2D	2D
Emptiness	S	S	U	U
Membership	S	S	S	S
Equivalence	?	U	U	U

S = Solvable; U = Unsolvable; ? = Unknown.

bility of a particular question. Here we consider the four classes of nsa languages to be represented as the family of sets accepted by the set of machines in each class.

In [1] it is shown that the emptiness problem is solvable for the class of indexed languages. Using results from [14] this implies solvability of the emptiness problem for 1D and 1N nsa languages and solvability of the membership problem for all four classes of nsa languages.

This also means that the class of 2N nsa languages is recursive, and hence that a 2N nsa is of less computational power than a Turing machine. Furthermore, it then follows that 2N nsa languages are not closed under arbitrary homomorphisms or gsm[13] mappings and thus are not full AFLs.

The emptiness problem is unsolvable for 2D and 2N nsa languages since this problem is unsolvable for 2D and 2N pda languages [11].

Since the equivalence problem is unsolvable for context-free languages [3], the equivalence problem must perforce be unsolvable for 1N and 2N nsa languages. Moreover, unsolvability of emptiness for 2D nsa languages also implies unsolvability of equivalence for this class. We do not know whether equivalence is solvable for 1D nsa languages (or, for that matter, 1D sa and 1D pda languages).

In Table I the decidability results are summarized for each of the four classes of nsa languages.

7. *Reading Pushdown Automata*

There are several interesting modifications that can be made to the behavior of an nsa. One such modification is to permit reading of the storage tape in only one direction. In this section we consider a sa such that whenever the sa is in the stack reading mode the stack head can never move upward. To return to the pushdown mode of operation at the top of the stack, this sa can in one reset move cause its stack head to return to the top of the stack. Such a device is called a reading pushdown automaton (rpa). Clearly, an rpa can be 1D, 1N, 2D, or 2N, and again it should be clear that each of these four classes of devices can be formulated as a closed class of balloon automata. We show that the class of 1N rpa languages does have an exact characterization in terms of a subclass of restricted indexed grammars, called *R*-grammars.

Formally, an rpa R is a 10-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F, \$, \cent, \#)$ (or a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ when $\$$, $\cent$, and # are understood) in which all symbols have the same meaning as for an nsa except now

(1) $\cent$ is only the left endmarker on the input tape and is not a storage tape symbol;

[13] Generalized sequential machine.

(2) δ is a mapping from

(i) in the pushdown mode,

$$Q \times \Sigma' \times \$\Gamma \quad \text{into finite subsets of} \quad Q \times D \times \$\Gamma^*,$$

(ii) in the stack reading mode,

$$\text{(a)} \ Q \times \Sigma \times \$\Gamma \quad \text{into subsets of} \quad Q \times D \times \{1\}$$

and

$$\text{(b)} \ Q \times \Sigma \times \Gamma \quad \text{into subsets of} \quad Q \times D \times \{0, 1\},$$

(iii) in the reset mode,

$$Q \times \Sigma' \times (\Gamma \cup \{\#\}) \quad \text{into subsets of} \quad Q \times D.$$

Here $D = \{-1, 0, 1\}$ and $\Sigma' = \Sigma \cup \{¢, \$\}$.

A configuration of R is as for an nsa and the definition of the relation $\vdash$ over the set of configurations is as for an nsa for the pushdown and stack reading modes. In the reset mode we write $(q, a_0 \cdots a_n, i, \$A\alpha\hat{B}\beta) \vdash (p, a_0 \cdots a_n, i + d, \$\hat{A}\alpha B\beta)$ if $\delta(q, a_i, B)$ contains (p, d). All other definitions associated with nsa apply in the obvious manner to rpa.

Although a 1N rpa does not appear to be as powerful as a 1N sa, a 1N rpa does have the same symbol table searching power as a 1N sa. For example, 1N rpa can syntactically check that in a programming language with declarations all identifiers appearing in a program have been previously declared.

Given an rpa R of some type we can effectively construct an rpa R' of the same type such that $N(R') = T(R)$ (or $T(R') = N(R)$). Thus acceptance by final state or empty stack does not change the class of languages accepted by rpa of a given type. As with nsa, acceptance is by empty stack unless otherwise stated.

We now show that a set L is accepted by a 1N rpa if and only if L is generated by a special type of indexed grammar called an R-grammar.

A *restricted indexed grammar* G is a 6-tuple (N, I, T, F, P, S) where

(1) N, I, T are finite disjoint sets of nonterminal, intermediate, and terminal symbols, respectively;

(2) F is a finite set of indices of the form $[B_1 \to \beta_1, B_2 \to \beta_2, \cdots, B_k \to \beta_k]$ where B_i is in I and β_i is in $(I \cup T)^*$, $1 \leq i \leq m$;

(3) P is a set of productions of two forms:

(i) $A \to \alpha$ where A is in N and α is in $(NF^* \cup IF^* \cup T)^*$, or

(ii) $B \to \beta$ where B is in I and β is in $(I \cup T)^*$ (a production of this form is called an *intermediate* production);

(4) S is a distinguished symbol in N.

The definitions of a derivation in a restricted indexed grammar $G = (N, I, T, F, P, S)$ and of the language generated by G are the same as for the indexed grammar $(N \cup I, T, F, P, S)$. Notice, however, that in a restricted indexed grammar an intermediate symbol cannot generate any new indices.

An *R-grammar* G is a restricted indexed grammar (N, I, T, F, P, S) in which each intermediate production and each index production is of one of the forms $A \to aB$ or $A \to a$ where A and B are in I and a is in $T \cup \{\epsilon\}$.

A reduced form R-grammar (N, I, T, F, P, S) is one in which each production (not an intermediate production) is of one of the forms $A \to BC$, $A \to Bf$, or $A \to a$ where A is in N, B and C are in $N \cup I$, f is in F, and a is in $T \cup \{\epsilon\}$.

It should be clear that given an arbitrary R-grammar G, an equivalent reduced form R-grammar G' can be effectively constructed from G such that $L(G') = L(G)$.

THEOREM 7.1. *Given a reduced form R-grammar $G = (N, I, T, F, P, S)$ an rpa $R = (Q, T, \Gamma, \delta, q, S, \varphi)$ can be effectively constructed from G such that $N(R) = L(G)$.*

PROOF. The proof is similar to that for Theorem 5.1, except that no new stacks can be created and the finite control of the rpa is used to simulate that part of a derivation involving only intermediate productions.

We let $Q = \{q\} \cup \{q_A \mid A \text{ is in } I\}$ and $\Gamma = N \cup I \cup F$. δ is constructed in the following manner.

1. For all a in $T \cup \{\$\}$, if $A \to BC$, $A \to Bf$, or $A \to \epsilon$ is in P, where A is in N, then $\delta(q, a, \$A)$ contains $(q, 0, \$BC)$, $(q, 0, \$Bf)$, or $(q, 0, \$)$, respectively.

2. For all b in T, if $A \to b$ is in P, then $\delta(q, b, \$A)$ contains $(q, 1, \$)$.

3. For all a in $T \cup \{\$\}$ and A in I, $\delta(q, a, \$A)$ contains $(q_A, 0, \$)$.

4. For all a in $T \cup \{\$\}$ and Y in $N \cup I$, $\delta(q_A, a, \$Y)$ and $\delta(q_A, a, Y)$ contain $(q_A, 0, 1)$.

5. If the index f contains the index production $A \to bB$ or $A \to b$ where b is in T, then
   (i) $\delta(q_A, b, f)$ contains $(q_B, 1, 1)$ or $(q, 1)$, respectively.
   (ii) $\delta(q_A, b, \$f)$ contains $(q_B, 1, \$)$ or $(q, 1, \$)$, respectively.

6. If the index f contains the index production $A \to B$ or $A \to \epsilon$, then, for all a in $T \cup \{\$\}$,
   (i) $\delta(q_A, a, f)$ contains $(q_B, 0, 1)$ or $(q, 0)$, respectively.
   (ii) $\delta(q_A, a, \$f)$ contains $(q_B, 0, \$)$ or $(q, 0, \$)$, respectively.

7. If $A \to bB$ or $A \to b$ are intermediate productions in P with A and B in I and b in T, then
   (i) $\delta(q_A, b, Y)$ contains $(q_B, 1, 0)$ or $(q, 1)$, respectively, for all Y in Γ.
   (ii) $\delta(q_A, b, \$Y)$ contains $(q_B, 1, \$Y)$ or $\delta(q, 1, \$Y)$, respectively, for all Y in Γ.

8. If $A \to B$ or $A \to \epsilon$ are intermediate productions in P, then for all a in $T \cup \{\$\}$
   (i) $\delta(q_A, a, Y)$ contains $(q_B, 0, 0)$ or $(q, 0)$, respectively, for all Y in Γ.
   (ii) $\delta(q_A, a, \$Y)$ contains $(q_B, 0, \$Y)$ or $(q, 0, \$Y)$, respectively, for all Y in Γ.

9. For all a in $T \cup \{\$\}$ and f in F, $\delta(q, a, \$f)$ contains $(q, 0, \$)$.

As a result of the construction of R we have $Af_1 \cdots f_k \overset{*}{\to} w$ if and only if

$$(q, w\$, 1, \$\hat{A}Z\beta_1 f_1 \cdots \beta_k f_k \beta_{k+1}\#) \vdash^* (q, w\$, n, \$\hat{Z}\beta_1 f_1 \cdots \beta_k f_k \beta_{k+1}\#)$$

where A is in $N \cup I$, $f_1 \cdots f_k$ is in F^*, $w = a_1 \cdots a_{n-1}$, $n \geq 1$, and $Z\beta_1\beta_2 \cdots \beta_{k+1}$ is in $(N \cup I)^*$.

The proof of this statement is similar to the proof of Theorem 5.1 and the details are left to the reader.

We have $S \overset{*}{\to} a_1 \cdots a_{n-1}$, $n \geq 1$, if and only if $(q, a_1 \cdots a_{n-1}\$, 1, \$\hat{S}\#) \vdash^* (q, a_1 \cdots a_{n-1}\$, n, \$\hat{\#})$. Thus $N(R) = L(G)$.

THEOREM 7.2. *Given an rpa $R = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varphi)$, an R-grammar $G = (N, I, \Sigma, F, P, S)$ can be effectively constructed from R such that $L(G) = N(R)$.*

PROOF. As one might expect, the proof is similar to that for Theorem 5.2. It is assumed that R writes at most two symbols on the stack tape on any one move in the pushdown mode.

Let

$$N = \{(p, A, q) \mid p,\ q \text{ are in } Q,\ A \text{ is in } \Gamma\},$$
$$I = \{(p, Y, q)', (p, \$Y, q)' \mid p,\ q \text{ are in } Q,\ Y \text{ is in } \Gamma \cup \{\#\}\}.$$

P is constructed in the following manner. Again, $d_a = a$ if $d = 1$ and $d_a = \epsilon$ if $d = 0$.

1. If $\delta(p, a, \$A)$ contains $(q, d, \$BC)$, $(q, d, \$B)$ or $(q, d, \$)$, then P contains, respectively,

(i) $(p, A, r) \rightarrow d_a(q, B, s)f_C(s, C, r)$ for all r and s in Q,
(ii) $(p, A, r) \rightarrow d_a(q, B, r)$ for all r in Q,
(iii) $(p, A, q) \rightarrow d_a$.

2. P contains the productions

$$(p, A, q) \rightarrow (p, \$A, r)'f_{\$A}(r, A, q)$$

for all p, q, r in Q, and A in Γ.

3. If $\delta(p, a, \$A)$ contains $(q, d, 1)$, then $f_{\$A}$ contains the index productions $(p, \$A, r)' \rightarrow d_a(q, Y, r)'$ for all r in Q, Y in $\Gamma \cup \{\#\}$.

4. If $\delta(p, a, Y)$ contains $(q, d, 0)$ or $(q, d, 1)$, then f_Y contains $(p, Y, r)' \rightarrow d_a(q, Y, r)'$ or $(p, Y, r)' \rightarrow d_a(q, Z, r)'$, respectively, for all r in Q, Z in $\Gamma \cup \{\#\}$.

5. If $\delta(p, a, Y)$ contains (q, d), then f_Y contains $(p, Y, q)' \rightarrow d_a$.

6. Finally, P contains the starting productions $S \rightarrow (q_0, Z_0, p)f_\#$ for all p in Q.

A proof by induction similar to that for Lemmas 5.3 and 5.4 can be used to show that $(p, A, q)f_{B_1} \cdots f_{B_m}f_\# \xrightarrow{*} w$ if and only if

$$(p, w\$, 1, \$\hat{A}B_1 \cdots B_m\#) \vdash^* (q, w\$, n, \$\hat{B}_1 \cdots B_m\#)$$

where $w = a_1 \cdots a_{n-1}$. Again, the details are left to the reader.

We have $S \rightarrow (q_0, Z_0, p) \xrightarrow{*} w$ if and only if $(q_0, w\$, 1, \$\hat{Z}_0\#) \vdash^* (p, w\$, n, \$\hat{\#})$ where $w = a_1 \cdots a_{n-1}$. Thus $L(G) = N(R)$.

We now have the following theorem.

THEOREM 7.3. *The class of languages generated by R-grammars is equivalent to the class of languages recognized by 1N rpa.*

We observe that the 1N sa languages are a subset of the languages generated by restricted indexed grammars.

THEOREM 7.4. *Given a 1N sa* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \varphi)$, *a restricted indexed grammar* $G = (N, I, \Sigma, F, P, S)$ *can be effectively constructed from M such that* $L(G) = N(M)$.

PROOF. The proof of Theorem 5.2, without the index productions arising from constructions 5 and 6 of that proof, is applicable here. Notice that if we let

$$N = \{(p, A, q) \mid p, q \text{ are in } Q,\ A \text{ is in } \Gamma\} \cup \{S\},$$
$$I = \{(p, Z, q)', (p, \$Z, q)', (p, Z, q)'' \mid p, q \text{ are in } Q,\ Z \text{ is in } \Gamma \cup \{\#\}\}$$

in this proof, then all intermediate productions and index productions involve symbols only in $(I \cup \Sigma)$. Hence the grammar resulting from the construction in Theorem 5.2 is a restricted indexed grammar.

Theorem 7.4 states that the class of 1N sa languages is contained within the class of languages generated by restricted indexed grammars. We conjecture that this containment is proper in that the language $L = \{a^n b^{n^2} a^n \mid n \geq 1\}$ can be generated by the following restricted indexed grammar G_1.

$$G_1 = (\{S, A\}, \{B, C\}, \{a, b\}, P, S, \{f, g\})$$

where P contains the productions $S \to aAfa$, $A \to aAga$, and $A \to B$ with $f = [B \to b,\ C \to b]$ and $g = [B \to bBCC,\ C \to bC]$. We conjecture that L cannot be accepted by any 1N nsa.

An exact characterization of 1N nsa languages in terms of two-type bracketed grammars has been recently proposed by Schkolnick [20].

REFERENCES

1. Aho, A. V. Indexed grammars—an extension of context-free grammars. *J. ACM 15*, 4 (Oct. 1968), 647–671.
2. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. Time and tape complexity of pushdown automaton languages. *Inform. Contr. 13*, 3 (Sept. 1968), 186–206.
3. Bar-Hillel, Y., Perles, M., and Shamir, E. On formal properties of simple phrase structure grammars. *Z. Phonetik Sprachwissen. Kommunikationsforsch. 14* (1961), 143–172.
4. Chomsky, N. Context-free grammars and pushdown storage. Quart. Prog. Rep. 65, MIT Res. Lab. Electron., MIT, Cambridge, Mass., 1962.
5. Ginsburg, S. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, New York, 1966.
6. Ginsburg, S., and Greibach, S. A. Abstract families of languages. IEEE Conf. Record of Eighth Ann. Symp. on Switching and Automata Theory, 1967, pp. 128–139.
7. Ginsburg, S., Greibach, S. A., and Harrison, M. A. Stack automata and compiling. *J. ACM 14*, 1 (Jan. 1967), 172–201.
8. Ginsburg, S., Greibach, S. A., and Harrison, M. A. One-way stack automata. *J. ACM 14*, 2 (Apr. 1967), 389–418.
9. Ginsburg, S., and Harrison, M. A. One-way nondeterministic real-time list-storage languages. *J. ACM 15*, 3 (July 1968), 428–446.
10. Ginsburg, S., and Hopcroft, J. E. Two-way balloon automata and AFL. Tech. Rep. No. 738/042/00, System Development Corp., Santa Monica, Calif.
11. Gray, J. N., Harrison, M. A., and Ibarra, O. Two-way pushdown automata. *Inform. Contr. 11*, 1/2 (July–Aug. 1967), 30–70.
12. Greibach, S. Checking automata and one-way stack language. IEEE Conf. Record of Ninth Ann. Symp. on Switching and Automata Theory, 1968, pp. 287–291.
13. Hopcroft, J. E., and Ullman, J. D. An approach to a unified theory of automata. *Bell Syst. Tech. J 46* (1967), 1743–1829.
14. Hopcroft, J. E., and Ullman, J. D. Decidable and undecidable questions about automata. *J. ACM 15*, 2 (Apr. 1968), 317–324.
15. Hopcroft, J. E., and Ullman, J. D. Nonerasing stack automata. *J. Compt. Syst. Sci. 1* (1967), 166–186.
16. Hopcroft, J. E., and Ullman, J. D. Sets accepted by one-way stack automata are context sensitive. *Inform. Contr. 13*, 2 (Aug. 1968), 114–133.
17. Hopcroft, J. E., and Ullman, J. D. Deterministic stack automata and the quotient operator. *J. Comput. Syst. Sci. 2* (1968), 1–12.
18. Hopcroft, J. E., and Ullman, J. D. *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, Mass., 1969.
19. Knuth, D. E., and Bigelow, R. H. Programming languages for automata. *J. ACM 14*, 4 (Oct. 1967), 615–635.
20. Schkolnick, M. Two-type bracketed grammars. IEEE Conf. Record of Ninth Ann. Symp. on Switching and Automata Theory, 1968, pp. 315–326.

RECEIVED MARCH, 1968; REVISED DECEMBER, 1968