INFORMATION AND CONTROL 20, 313–330 (1972)

Language Recognition by Marking Automata

R. W. RITCHIE

University of Washington, Seattle, Washington 98105

AND

F. N. SPRINGSTEEL*

University of Montana, Missoula, Montana 59801

An off-line, memory-restricted Turing machine model, the marking automaton (MA) is presented here as a device strictly intermediate between finite and linear bounded automata. Although much more restricted than the latter, MA are shown capable of recognizing, deterministically, various kinds of context-free (CF) languages and an important related class, as well as such non-CF languages as $\{xx\}$. It is not known whether all CF languages are recognizable by MA; however, among the familiar subclasses shown to consist of MA recognizable languages are the Dyck, standard, and bounded CF languages. More importantly, each member of the class of structured CF languages, consisting of all structural descriptions (Phrase-markers) of the sentences in a CF language, is shown to be MA recognizable. The closure of the MA recognizable languages under various set (e.g., boolean) operations is revealed in the proof that all bounded CF languages are MA recognizable.

I. INTRODUCTION

The recent interest in transformational grammars has given rise to the search for a simple device which can recognize their sets of Phrase-markers (P-markers—the domains of the transformations). For the case of context-free based grammars we present one model of such a device here, in various forms. It also has other interesting capabilities in language recognition. In many ways it compares favorably with push-down automata, and it naturally generalizes finite automata.

Marking automata (MA) are models of a device intermediate between

* Some of this author's work was completed at Bowdoin College, Brunswick, Maine, on a Faculty Research Fund Grant.

(two-way, deterministic, nonprinting) finite automata and linear bounded automata (lba). Their main advantage over the former is in being able to tag or mark without permanently altering a fixed number of their tapes' squares. We define recognition of a language by a marking automaton in terms of a restricted lba with end markers on its tape. Then we show in Theorem 1 the equivalence (in terms of language recognition) of this form of the model with another, defined herein. This second form of the model is that of a bounded counting automaton (BCA) which although easier to describe precisely, is surprisingly powerful. Other equivalent forms of the model are mentioned which secure its place in the tape complexity hierarchy of off-line Turing machines. However, the main results about context-free language recognition are proved with the aid of the BCA form only.

These results include the recognition by MA of λ-free Dyck languages (Theorem 2), Dyck languages (Corollary 2.1) and standard context-free languages (Corollary 2.2) as well as the principal result, Theorem 3 (and Corollary 3.1), that any structured context-free language (and, hence, any set of P-markers for a context-free based transformational grammar) can be recognized by a marking automaton. Lastly, a direct use of the first form of the MA model shows that any bounded context-free language is also recognizable, and in doing so uncovers certain properties of the class of MA recognizable languages (Theorem 4 and Corollary 4.1).

The question of whether or not all context-free languages are recognizable by MA is left open, and other work (Springsteel, 1967) suggests a solution only by abandoning the determinism of our model. However, no CF language has been proved not to be recognizable by deterministic MA.

The reader will note that each of Theorems 1, 2 and 3 has been given a "proof outline" as well as a formal proof. The outline is a short statement giving the idea of the proof, and this is all some will want to read. The proofs of the first two theorems are rather long—not surprisingly so in a new theory with as yet few fundamental principles. However, to claim reasonable rigor, proofs must be given, and in the cases of Theorems 1 and 2 are found in an appendix. The techniques used, while "classical," have not to our knowledge appeared in print.

II. Marking and Bounded Counting Automata

Marking automata were introduced in Kreider and Ritchie (1966) as the "smallest reasonable" generalization of (two-way, deterministic, nonprinting)

finite automata. The informal model of an MA is that of a deterministic two-way automaton with a pre-fixed, finite supply of reference marks which can be attached to or removed from the squares of its input tape (its only tape), but no other type of tape modification or extension is allowed. Clearly, such machines are limited by output to computing a very restricted class of functions. Their main indicated use is accepting or rejecting input strings, as tape "recognizers." This notion, however, has important consequences for the theory of formal languages (Aho and Ullman, 1968) as well as in automata theory. The ability to attach a mark to an arbitrary square of the tape provides an MA with the ability to act as a counter, bounded by the number of squares (or symbols) in the input string. This observation permits a formalization of the notion of a bounded counting automaton (BCA) which captures exactly, in language recognition power, the formal notion above of an MA.

To *prove* this equivalence it is necessary (at least) to have available a mathematical definition of a language being "recognizable by marking automata." (Note that this is not the same as defining the property of *being* an MA.) Fortunately, such a definition exists, in slightly different forms, in at least two sources, Kreider and Ritchie (1966) and Springsteel (1967). The definition in the former need only be modified to allow basic input alphabets other than {0, 1}. This yields a slight (i.e., only formal) variation on the definition in the latter and is presented here for completeness. Our definition is in terms of an lba with end markers [cf. Kuroda (1964)].

DEFINITION 1. A language L on a finite alphabet Σ is said to be *recognizable by marking automata* (MA) iff there is a deterministic lba, **A**, with end marker (boundary symbol) #, and input–output alphabet $\Sigma \cup M$, where M is a finite, nonempty set of symbols (called marks or markers) such that $M \cap \Sigma$ is void and # is in M; **A** has among its finite set of states an initial state q_0 and a subset F of final states, q_0 not in F, which is partitioned into F_0 (the set of accepting states of **A**) and $F_1 = F - F_0$ (the set of rejecting states of **A**) such that for any sequence of steps of **A** starting with a configuration

$$\# \, q_0 x \, \#, \qquad \text{where } x \text{ is in } \Sigma^*,$$

the following conditions hold:

(1) there is a fixed upper bound, k (not dependent on x) on the number of occurrences of symbols from M which appear on the tape of **A** at any step;

(2) for any two symbols σ and τ in Σ, σ is never printed on a tape

square originally occupied by τ nor on an end square (originally occupied by #);

(3) if x is in L, then **A** halts in a state of F_0 and **A** is said to *accept* x; if x is in $\Sigma^* - L$, then **A** halts in a state of F_1 and is said to *reject* x. (**A** is said to halt if $\# q_0 x \#$ eventually yields a configuration in $\# \Sigma^* F \Sigma^* \#$.) In the case that Definition 1 is satisfied, we sometimes say that **A** *is* the MA which recognizes L, even though we have *not* defined MA, and we call L an MA language.

Note that the end marker # is not strictly necessary on the left since we could have **A**, started on the leftmost square of the input, move left on its first step and mark that end square. Similarly on the right, if we allow **A** to "sense" a blank square, then we could have it mark the first one found in that direction, doing away with a premarked # there. But it is more convenient to retain the end markers, as we will see below.

We are now ready to define formally the automata which we will most often use as recognizers of MA languages.

DEFINITION 2. A *bounded counting automaton* (BCA) is a seven-tuple $\mathbf{B} = \langle \Sigma, S, q_0, k, f, A, \# \rangle$ where

Σ is a finite, nonempty set (the alphabet of **B**),

S is a finite, nonempty set (the states of **B**),

$q_0 \in S$ is the initial state of **B**, k is a natural number, $A \subset S$ is the set of accepting states, and, where # is an end marker not in Σ, f is a partial function from $(\Sigma \cup \{\#\}) \times S \times \{0, +\}^k$ into $S \times \{1, 0, -1\}^k \times \{L, R, N\}$.

The set of instructions of **B**, given by f, consists of basic actions of the form

$$\langle \sigma_i, q_j, (b_1, \ldots, b_k) \rangle \xrightarrow{f} \langle q_l, (n_1, \ldots, n_k), m \rangle,$$

where $\sigma_i \in \Sigma \cup \{\#\}$; $q_j, q_l \in S$; $b_1, \ldots, b_k \in \{0, +\}$; $n_1, \ldots, n_k \in \{1, 0, -1\}$ and $m \in \{L, R, N\}$.

We think of **B** as having k counters $C_1, \ldots, C_k$, each of which can be tested for being zero or positive, and which can be increased or decreased by one as long as the respective counts (numerical contents) $| C_1 |, \ldots, | C_k |$ all stay within a lower bound of zero and an upper bound equal to the length of the current input string. Since the zero testing can be used to determine the count changes (and other results of a move, this being their only natural use), we have a k-tuple $(b_1, \ldots, b_k)$ of 0's and +'s as part of the arguments

of f and a k-tuple $(n_1, \ldots, n_k)$ of 1's, 0's and -1's as part of the values of f; $b_j = +$ iff $| C_j | > 0$.

B is given its input on a separate tape, marked at the ends by #'s, between which it can move in either direction, but can never write or erase. The movement on the tape is either one square right, one square left, or none at all according as m equals R, L or N.

At any time, t, during the computation of **B** the basic actions under f are determined by the symbol σ_i being scanned and the current state q_j as well as the settings of the counters; the outcome at time $t + 1$ includes an internal change to the new state q_l besides the k-tuple of counter changes, each $| C_j |$ to $| C_j | + n_j \leqslant$ input length, and the tape movement, m.

Recall that f was defined as a *partial* function. For those $(k + 2)$-tuples on which f is *not* defined, the BCA **B** will be said to *halt*. If the current state is a member of A, the set of accepting states, then the string ($\in \Sigma^*$) on the input tape will be said to be *accepted*. If not (i.e., if the state is in $S - A$), the input is said to be *rejected*.

Clearly **B** accepts or rejects every string in Σ^* if and only if it eventually halts for every input from Σ^*. BCA with this property are said to *recognize* the set of input strings (languages) which they accept. It is this class of automata we are most interested in, for it is equivalent (in language recognition power) to the class of all MA. That is, it includes enough BCA to recognize all (but only) MA languages, as we show in Theorem 1.

Remark. The following observation eliminates much unnecessary detail from the description of BCA and will be used in the proof of Theorem 1 and subsequently. By adding more counters to a given BCA, zero testing of the original counters can be extended to ordinal comparisons between the contents, even with a fixed integer added.

Specifically, for the BCA **B**, with k counters, any inequality of the form

$$| C_j | \leqslant | C_l | + i, \tag{1}$$

where i is a sufficiently small nonnegative integer, can be checked in certain configurations to determine the next move because such an operation can be simulated by a parallel BCA **B′**, with $2k$ counters, using a sequence of basic actions. To do so, **B′** has a pair of counters corresponding to each counter of **B** to be checked, the second counter in a pair in **B′** being used in an auxiliary manner. To test the inequality (1) involving the j-th and l-th counters of **B**, above, **B′** initiates its simulation with the counts $| C_j |$ and $| C_l |$ in the first counters of its j-th and l-th pairs, respectively, and zero in the corresponding second counters. To the count $| C_l |$ in the first counter

of its l-th pair it adds the fixed integer i, one unit at a time. $\mathbf{B}'$ then transfers the counts from each of the first counters in these two pairs to their respective second counters, alternating between the two pairs, one unit per basic step. It notes which transfer is completed first, resets the counters and makes the thus determined move.

The operations of a BCA such as $\mathbf{B}'$ are made apparently so much more flexible than those of the basic model $\mathbf{B}$ by using two counters, "inefficiently," to hold one count. Since extended comparisons between their counters can be made by BCA, the role of the function f should be elaborated accordingly.

THEOREM 1. *A language L is recognized by an* MA *if and only if it is recognized by a* BCA.

Proof outline. Each half of the proof consists of a fairly straightforward simulation of one kind of automaton by the other. Naturally we take the simplest equivalent form of the automaton being simulated, but use the strongest developed powers of the simulator.

An MA simulates a BCA by using one mark to correspond to each counter, the mark recording that the number in the counter is n by being n squares to the right of the left end marker on the input tape. An extra mark is used as a place holder, giving a total of $k + 1$ marks on the tape to imitate k counters.

The converse follows similarly, using a result in Kreider and Ritchie (1966) that an MA need only contain one kind of marker. Thus the only important thing to record about the markers when using a BCA to simulate a suitably reduced MA is their positions. This requires one counter for each occurence of the marker on the tape, plus an extra counter to record the position of the scanned square, so that our BCA can imitate the "finding" of a mark by a successful comparison of this counter with one of the others, using the extended capability of BCA. End of Outline.

A number of other types of deterministic automata can similarly be shown to be equivalent to MA. For example, multihead (or multitape) nonprinting two-way finite automata, or, off-line Turing machines which cannot add to or write on their input tape but which have a finite number (k) of extra storage tapes, each of length $\leqslant \log n$, where n is the length of the input. [See Aho and Ullman (1968) or Hartmanis and Stearns (1965) for a discussion of tape complexity classes.] However, we shall use only the marking and counting options of this model of automata, and hence will deal only with MA and BCA here.

III. Context-Free Language Recognition by Marking Automata

In this section we investigate the question of which languages, especially context-free (CF) languages, are recognizable by MA. [For a definition of CF languages see any of Chomsky (1959), Chomsky and Shutzenberger (1963), Ginsburg (1966) or Ginsburg and Harrison (1967).] Since no specific CF languages have been demonstrated *not* recognizable by MA, one might conjecture that they all are, at least all deterministic CF languages. [See Ginsburg and Greibach (1966) for a definition of the latter.] We do not affirm this answer to the general question here, but show only that certain important subclasses of the CF languages are recognizable by MA, namely, the Dyck languages, the standard and the bounded CF languages, and the structured CF languages. We define two kinds of Dyck languages, one excluding λ, in parallel.

Definition 3. The (λ-free) *Dyck language* $D_n(D_n^+)$ on the alphabet $\{a_1, ..., a_n\}$ is the set of all (nonnull) words in the free semigroup generated by $A_n = \{a_1, a_1', ..., a_n, a_n'\}$ which reduce to the identity λ, the null word, under the sole relation $a_i a_i' = \lambda$, $1 \leqslant i \leqslant n$.

Dyck languages can be thought of as consisting of well-formed strings of matching labeled parentheses. We define λ-free Dyck languages here for convenience in proving MA recognition. However, since $\{\lambda\}$ is a regular event (Rabin and Scott, 1959) and the MA languages are closed under finite unions (Kreider and Ritchie, 1966) we can drop the qualifier "λ-free" in Theorem 2 and obtain Corollary 2.1.

Dyck languages are easily seen to be CF languages. [See Ginsburg (1966) for a proof.] Since the CF languages are closed under intersection with regular events (Bar-Hillel, Perles and Shamir, 1961), the next definition is forced to be that of a subclass of the CF languages.

Definition 4. A *standard* (CF) *language* is the intersection of a Dyck language and a regular event.

Theorem 2. *Any λ-free Dyck language, D_n^+, is recognizable by an* MA.

Convention. For the purposes of this proof (and the next), we will refer to a left parenthesis symbol, like a_i, (or a left bracket) as a "paren" and to a right parenthesis a_i' (or a right bracket) as a "thesis."

Proof outline. The idea of the proof is to use a BCA which not only can carry out the usual parenthesis checking algorithm—proceeding by

adding one to the count whenever a paren is encountered and subtracting one when a thesis is met but also can compare parens and theses of the same (nested) depth for proper matching of their labels. It does this by recording, simultaneously with parenthesis checking, the maximum depth d of any nest (the highest count obtained) and then checking on d subsequent passes that the labels of depth d are properly matched, that those at depth $d - 1$ are matched, and so on. For this purpose two counts are maintained and compared as necessary, one holding the depth of the paren being checked and the other that of the corresponding thesis. End of Outline.

COROLLARY 2.1. *The Dyck language D_n, for any n, is recognizable by* MA.

COROLLARY 2.2. *Each standard* CF *language is recognizable by* MA.

Proof. Regular events are MA recognizable, and the MA languages are closed under intersection (Kreider and Ritchie, 1966).

Corollary 2.2 indirectly raises the question of whether the MA languages are closed under (free semigroup) homomorphisms, since it is well known that every CF language is the homomorphic image of a standard CF language. [See, e.g., Chomsky and Shutzenberger (1963).] The second author showed in Springsteel (1967) that the homomorphic images of the MA languages include all recursively enumerable (or, type 0) languages, so this is not a feasible way of showing MA recognition of all CF languages.

We can show, however, that an important class of languages—closely related to the CF class—are all MA recognizable. These are the structured CF languages, each of which is essentially the set of all structural descriptions of the strings in some ordinary CF language. As interest in (CF-based) transformational grammars increases, attention is focusing on these languages, since it is the structural descriptions (sometimes called "Phrase-markers") rather than the weakly generated strings themselves to which the transformations apply. Recall that a structural description of a string in a CF language is the result of inserting labeled brackets (parentheses) around the symbols in the string to display the "derivational history" of the string. (Sometimes "trees" are used instead, but not here.) For convenience, this class is defined here as in Peters and Ritchie (to appear).

DEFINITION 5. The set of *structual descriptions* generated by a CF grammar G is the language generated by its associated grammar G', as below. If $G = (N, \Sigma, R, S)$ is a CF grammar and we let P be the set of all parens $[_A$, and T the set of all theses $]_A$, labeled by the nonterminals A

in N, then $G' = (N, \Sigma \cup P \cup T, R', S)$, where $A \to [_A w]_A$ is a rule in R' iff $A \to w$ is one in R.

[The bracketed grammars of Ginsburg and Harrison (1967) are essentially the special G' obtained from those CF grammars G in which no nonterminal appears as the left-hand side of two or more rules or, somewhat more precisely, modifications of such G' which have a finite number of sentence symbols $S_1, \ldots, S_k$.]

We call $L(G')$ a *structured* (structurally described) CF language and, because G' is naturally associated with the CF grammar G, it has also been called the language "strongly generated by G."

Clearly, this gives a natural way of associating with any CF language L a structured language which acts as a set of Phrase-markers (structural descriptions) for L. Each string x in the structured language is a Phrase-marker, in the linguistic sense, for the sentence y obtained by deleting all labeled parens and theses which occur in x.

THEOREM 3. *Each structured* CF *language is recognizable by* MA.

Proof outline. The proof of this theorem can be based on the previous one, the procedures of which can be used to recognize the Dyck language part of (i.e., check the correctness of the labeled bracketing in) a purported structural description, ignoring terminal symbols. Now these procedures can be modified in a natural way also to check the correctness of application of the CF production rules, because at most a fixed, finite number of terminal symbols or brackets of depth $l + 1$ can occur between matched brackets of depth l.

Proof. Given the structured language $L(G')$ associated with the CF grammar $G = (N, \Sigma, R, S)$, let W be the maximum length of any word w which occurs as the right-hand side of a rule in R, and let n be the number of parens in P, i.e., the number of nonterminals in N. Using Theorem 2, let $\mathbf{B}_n$ be the BCA which recognizes the λ-free Dyck language D_n^+ on the alphabet P, now, with $P \cup T$ replacing A_n, i.e., with $[_A$ and $]_A$ for all A in N replacing a_i and a_i', $1 \leqslant i \leqslant n$, respectively. Modify $\mathbf{B}_n$ to extend it to the alphabet $\Sigma \cup P \cup T$, but to ignore symbols in Σ during its parenthesis checking. To the two stages of operation (parity checking and label checking) of this automaton, corresponding to those of $\mathbf{B}_n$ in Theorem 2, we add a third stage of rule checking, i.e., at each depth l, we check the correctness of application of the rules in R for the purposes of deriving certain parts of the current input string. This requires an examination of

at most $2W$ brackets and terminal symbols of depth $l+1$ interspersed between two matched brackets at depth l, as we now show.

Suppose that a paren $[_A$ is scanned of depth l. This implies that our BCA is to check whether some one of the finitely many rules of the form $A \to [_A w]_A$ has been correctly applied to obtain the sequence of brackets and terminals of depth $l+1$ between this paren and its matching thesis. Since at most one rule could produce the sequence scanned on the tape, this modification is also easily made, using the fact that all such sequences are bounded in length by $2W$, which can be seen as follows.

Suppose that in the correct rule w has the form $\tau_0 n_1 \tau_1 n_2 \tau_2 \cdots \tau_{m-1} n_m \tau_m$, where $\tau_i \in \Sigma^*$ (some τ_i may be λ) and $n_j \in N$; m is a natural number. Then $m \leqslant W$ and the string of symbols our BCA is to check for between $[_A$ and the next $]_A$ of depth l, is

$$\tau_0[_{n_1}\text{----}]_{n_1}\tau_1[_{n_2}\text{----}]_{n_2}\tau_2 \cdots \tau_{m-1}[_{n_m}\text{----}]_{n_m}\tau_m ,$$

where the dashes replace all symbols of depth greater than $l+1$, which have already been checked (by a count-down procedure) and so can be ignored. The total number of terminal symbols and brackets to be checked at this stage is, then, less than or equal to $W + m \leqslant 2W$, since the length of $\tau_0\tau_1 \cdots \tau_m$, as part of $w \in (\Sigma \cup N)^*$, is at most $W - m$. Using finite control, the BCA $\mathbf{B}_n$ of Theorem 2 can easily be modified to carry out the checks above. Then Theorem 1 proves that $L(G')$ is recognizable by MA. Q.E.D.

COROLLARY 3.1. *The set of deep Phrase-markers of a* (CF-*based*) *transformational grammar is recognizable by* MA.

Each of the CF languages shown to be recognizable by MA above is, in fact, a deterministic CF language. (See the appropriate references for proof.) There is another class of CF languages of practical importance, which includes inherently ambiguous (and thus nondeterministic) CF languages; yet it is properly contained in the class of MA languages. This is the class of bounded CF languages, defined below, which are shown to be MA recognizable by an entirely different approach.

DEFINITION 6. A language $L \subset \Sigma^*$ is said to be *bounded* if there are words $w_1, \ldots, w_n \in \Sigma^*$ such that every string in L is of the form $w_1^{i_1} \cdots w_n^{i_n}$, $i_j \geqslant 0$.

An example of an inherently ambiguous, bounded, context-free language is

$$\{a^i b^j c^j : i, j \geqslant 1\} \cup \{a^i b^i c^j : i, j \geqslant 1\}$$

[cf. Chomsky and Schutzenberger (1963)]. Another example of a nondeterministic CF language which is *not* bounded is

$$\{\sigma_1\sigma_2 \cdots \sigma_n\sigma_n \cdots \sigma_2\sigma_1 : \sigma_i \text{ any symbol in } \Sigma, \text{ all } n \geqslant 1\}.$$

Both of these languages, and similar constructions, including the non-CF language $\{a^nb^nc^n : n \geqslant 1\}$, are clearly MA recognizable.

THEOREM 4. *Any bounded* CF *language is an* MA *language, but not conversely.*

Proof. The converse is disproved by the second example above. For the positive assertion of the theorem, we use the characterization of the bounded CF languages as the smallest class containing the finite sets and closed under the operations of set unions, language (concatenation) products, and the passage from a set L to

$$(x, y) * L = \bigcup \{x^nLy^n : n \geqslant 0\}, \qquad \text{where} \quad x, y \in \Sigma^* - \{\lambda\}.$$

[This is Theorem 5.4.1 in Ginsburg (1966).]

Clearly, the MA languages contain the finite sets and are closed under finite unions and products, by arguments similar to those in Kreider and Ritchie (1966). We need only show that $(x, y) * L$ is an MA language if L is.

Suppose that a language L is recognized by an MA, **A**. We modify **A** to obtain an MA, **A'**, which will recognize $L' = (x, y) * L$ as follows. **A'** has five phases of operation:

(1) Initially place two special markers, m_1 and m_2, not used by **A**, on the left and right end squares of the tape (originally occupied by #), and go to phase 2.

(2) Imitate **A** within the boundaries of the two m markers, accepting the whole input if the so-bounded string is accepted by **A**. If **A** rejects this bounded string, go to phase 3.

(3) Scan through the x on the left end of the m-bounded string, if x occurs there, and transfer m_1 to the rightmost symbol of this x. If x does not so occur in the presently bounded string, reject the input. If the scan and transfer can be performed, go to phase 4.

(4) Scan backwards through the y on the right end of the m-bounded string, if y occurs there, and transfer m_2 to the leftmost symbol of this y. If y does not so occur in the presently bounded string, reject the input. If the backward scan and transfer can be performed, go to phase 5.

(5) Loop through phases 2, 3, 4 a finite number of times, halting the process (and rejecting the input) whenever m_2 is supposed to be printed to the left of, or superimposed on, m_1.

Note that if the input is not of the form $x^m w y^n$, $(m, n \geqslant 0)$ then it is rejected the first time through phase 3 or phase 4. If it is of this form, but not for $m = n$ and some $w \in L$, then it gets rejected on a later loop through phases 2–3–4, e.g., if it is $x^n y^n$ but $\lambda \notin L$. Finally, if x^m and y^n are of such a form (essentially, overlapping) that we obtain the situation described in phase 5, then the procedure should reject such an input; it has proved not to be of the desired form in L', or it would have been accepted earlier, in phase 2.

Hence $\mathbf{A}'$, as outlined above, accepts only strings in L', and it clearly accepts all of these. Therefore the theorem is proved by induction in the class of bounded CF languages. Q.E.D.

The theorems and examples in this paper do not give a complete idea of just what languages, other than some CF, can be recognized by MA. For example, the context-sensitive language $\{xx : x \in \Sigma^*\}$ is not bounded, not CF but is easily seen to be MA recognizable.

The following corollary to Theorem 4 sums up the relevant information about the closure of the MA languages under operations studied here. The closure under complementation (within Σ^*) follows by determinism.

COROLLARY 4.1. *The class of languages recognizable by* MA (*the* MA *languages*) *is closed under the following operations*:

(a) *set union and complementation, hence, all boolean operations*;

(b) *concatenation* (*set product*);

(c) $(x, y) * L$.

APPENDIX

Extended proof of Theorem 1. We give slightly more details for the first part of the proof, because it is more often used here and because the imitation of a BCA by an MA is not quite as direct as the outline of the proof indicates: we must deal with the possibility of two or more counters having the same count, while only one symbol per square is allowed an MA.

Suppose that a BCA, $\mathbf{B} = \langle \Sigma, S, q_0, k, f, A, \# \rangle$, recognizes a language $L \subset \Sigma^*$. Then $k > 0$, and we can assume that all k counters of $\mathbf{B}$ start each computation with their counts all set at zero; any other fixed settings

can be generated. We simulate **B** with an MA, **A**, (actually with an lba) informally described as follows: **A** places at most $k + 1$ of its m-markers—$m(s)$, $m(s, \{i_1\})$, $m(s, \{i_1, i_2\})$,..., $m(s, \{i_1, ..., i_k\})$, where $s \in \Sigma \cup \{\#\}$, $1 \leqslant i_j \leqslant k$, and $i_j \neq i_l$ if $j \neq l$—on the tape at any step of a computation begun on an input of the form $\# x \#$, $x \in \Sigma^*$. For each m-marker $m(s, \{i_1, ..., i_k\})$, s will be called its *alphabetic* index and the i_j its *numeric* indices (there being none if $k = 0$). Thus the alphabet of **A** consists of 2^k m-markers for each symbol $s \in \Sigma \cup \{\#\}$ in addition to $\Sigma \cup \{\#\}$ itself. Furthermore, each of the integers from 1 to k, inclusive, appears as a numeric index of exactly one m-marker occurring on the tape at each step of a computation, since these indices indicate which of the k counters of **B** (in some fixed numbering) have the same count as the square on which the m-marker occurs. This convention takes care of the "multiple-count" possibility mentioned above.

To conform with our simplifying assumption about the start conditions of **B**, we require that **A** begin each computation by changing the left end marker $\#$ to $m(\#, \{1, 2, ..., k\})$. We assume that the moves of **B** are given in an atomic form: a zero test of exactly one of its k counters followed by the addition of one or zero to (or subtraction of one from) this counter, while changing its state and moving left or right or not at all on its tape. Each aforementioned basic action of a BCA is only a combination (by parallel execution) of such atomic moves. The entire simulation can then be seen from the following typical analysis for atomic moves.

Suppose a configuration in **B** results either in adding one to the j-th counter if it was zero or subtracting one if it was positive. (Adding zero is a "passive" action, imitated by doing nothing.) Then **A** starting at the same scan square, would imitate this move by, first, finding its unique m-marker with a numeric index j and, according as its alphabetic index is (1) $\#$ or (2) $\sigma \in \Sigma$, acting in one of the following patterns:

(1) Suppose the m-marker found is $m(\#, \{i_1, ..., i_l\})$ with $i_h = j$, $1 \leqslant h \leqslant l$. Then **A** changes it to $m(\#, \{i_1, ..., i_{h-1}, i_{h+1}, ..., i_l\})$ if $l > 1$ or to simply $\#$ if $l = 1$, and moves one square right. If **A** is now scanning an m-marker $m(\tau, \{j_1, ..., j_p\})$, it changes it to $m(\tau, \{j_1, ..., j_p, j\})$. If **A** is scanning an input symbol $\tau \in \Sigma$ instead, it changes it to $m(\tau, \{j\})$.

(2) If the m-marker with index j was found to be $m(\sigma, \{i_1, ..., i_l\})$, then **A** changes it as in (1), leaving σ if $l = 1$, moves *left* one square and acts much as above. The only difference is that **A** might now be scanning an end square, but, with $\#$ replacing τ, the action is the same as in pattern (1). Obviously, the alphabetic index of any m-marker on the tape is the

symbol of $\Sigma \cup \{\#\}$ which originally occupied the square now occupied by the marker.

After the adjustment of its marks as indicated above, **A** returns to the square scanned at the origin of this move, which it has kept track of by leaving a place-holder mark (if no other m-marker was already there) of the form $m(s)$, and restores the original symbol, s in this case and then moves on the tape as **B** moves and changes to the state to which **B** switches, to complete the imitation. Other typical atomic moves are imitated analogously.

Thus **A** uses predictably more states than **B** and it uses many more markers than **B** has counters, but it places at most $k + 2$ of them, counting the $m(s)$, on the tape at any one time. This completes our informal description of **A**; a formal description can now be provided by the reader, if desired.

It should now also be clear that **A** accepts (or, rejects) exactly those strings $x \in \Sigma^*$ which **B** accepts (or, rejects) if we give **A** the same final (accepting, rejecting) states as **B**, i.e., $F = S$, $F_0 = A$ and $F_1 = S - A$. **A** always halts in a (final) state in F, since **B** always halts. If a certain configuration of the counters of **B** is required for acceptance (e.g., that only the first and last counters be set at zero), we can have **A** check its marks for the corresponding configuration—that $m(\#, \{1, k\})$ be on the left end square of the tape in this example. Therefore the language L is recognized by an MA.

For the converse we need Lemma 2 of Kreider and Ritchie (1966), or more precisely its formulation for alphabets more general than $\{0, 1\}$ but for the less general unary relations (languages) rather than for n-ary relations. In this form it is part of Theorem 3.2 of Springsteel (1967) and, in the present context, implies that M can be taken to be the singleton $\{\#\}$.

Having reduced the number of markers to one, we now construct a BCA which can imitate such an MA. Since the (possibly) k marks on the tape are indistinguishable, only their positions are relevant and these can be recorded by k counters, each of which is set at zero unless its corresponding mark is on the input (Σ^*) portion of the tape. But the BCA must be able to tell which mark, if any, is being scanned by the MA so that it can act accordingly. This is the purpose of the extra counter, C_0, which records the number of squares the scanned square is to the right of the left end marker. C_0 begins the computation, of course, set at one, since the MA starts in the configuration $\# q_0 x_1 \cdots x_n \#$, and goes up one or down one as the reading head moves right or left. Note that $| C_0 |$ need never exceed the length n of the input; for if the right end marker $\#$ is to be scanned,

we can hold $| C_0 |$ at n until the reading head moves left, again on the input portion.

Now we use the extended test capability of our BCA. It "knows" if it "sees" a mark by comparing $| C_0 |$ with each of $| C_1 |, \ldots, | C_k |$; if $| C_0 | = | C_j |$ for some j, then our BCA is scanning a square on which the imitated MA left a mark. Otherwise, it is not. This decision (plus the recognition of the symbol on the scan square) is all it needs to determine its next move. If the next action of the MA includes printing a new mark on the scan square, then the BCA takes the first counter set at zero among $C_1, \ldots, C_k$ and increases its count (by a transfer) to $| C_0 |$, the position of the scanned marker. All other details are routine, including the correspondence between final actions of the two machine models. For example, since "normal" MA [those guaranteed by Theorem 3.2 of Springsteel (1967)] always halt on the left end of the tape after restoring the input, we have that our simulating BCA will always halt with $| C_0 | = 1$ and $| C_j | = 0$, $1 \leqslant j \leqslant k$.

In any event, the BCA so described recognizes the same language, L, in Σ^* as the given MA. This completes the proof. Q.E.D.

Extended proof of Theorem 2. The BCA with two (pairs of) counters which recognizes $D_n{}^+$ will be called $\mathbf{B}_n$. The states of $\mathbf{B}_n$ are in the set $S = \{q_0, s_0, s, s', s_1, \ldots, s_n\}$. Most moves of $\mathbf{B}_n$ will be given in the form

$$\langle \sigma, X, k, l \rangle \rightarrow \langle Y, k', l' \rangle,$$

where $\sigma \in A_n$, X and Y are states (in S), and k (resp., l) is the count in the first (resp., second) counter pair [hereafter *not* referred to as a pair in light of the remarks following Definition 2] and k' (resp., l') the count resulting from the execution of the move, after $\mathbf{B}_n$ has switched to state Y from state X. Such moves (with $\rightarrow$) will here indicate a shift one square right on the tape. Leftward moves are similarly described (with $\leftarrow$) by

$$\langle Y, k', l' \rangle \leftarrow \langle \sigma, X, k, l \rangle.$$

Special moves will be given for the situations $\langle \#, X, k, l \rangle$.

First we want $\mathbf{B}_n$ to make one left-to-right pass across the tape using the regular parenthesis checking routine to see that the numerical parity of parens and theses is correct ignoring labels and simultaneously to record the maximum depth, d, of any nest of parentheses; d being the largest number accumulated in parity checking.

The first counter of $\mathbf{B}_n$ is always used to record the depth of the symbol currently being scanned while—on the first pass—the second counter is

used to register d, the maximum count of the first. Hence subtraction is not done here to the second count, and addition is performed here only when it would otherwise be exceeded by the first. The appropriate moves for the first complete pass by $\mathbf{B}_n$ are as follows where $1 \leqslant i \leqslant n$:

$$\begin{aligned}
&\langle a_i , q_0 , 0, 0\rangle \to \langle s, 1, 1\rangle;\\
&\langle a_i , s, k, l\rangle \to \langle s, k+1, l\rangle, \qquad \text{if} \quad k < l;\\
&\langle a_i , s, l, l\rangle \to \langle s, l+1, l+1\rangle;\\
&\langle a_i', s, k, l\rangle \to \langle s, k-1, l\rangle, \qquad \text{for} \quad k > 0;
\end{aligned}$$

also

$$\begin{aligned}
&\langle s_0 , 0, l\rangle \leftarrow \langle \#, s, 0, l\rangle, \qquad \text{if} \quad l > 0;\\
&\langle s_0 , 0, l\rangle \leftarrow \langle \sigma, s_0 , 0, l\rangle, \qquad \text{any } \sigma \in A_n ;
\end{aligned}$$

and

$$\langle \#, s_0 , 0, l\rangle \to \langle s', 0, l\rangle.$$

This sequence of moves returns $\mathbf{B}_n$ to the left end of the input tape in state s', with $l = d$, ready to begin label checking. Note that the first counter must return to zero when the *right* end marker is scanned (indicating a successful parity check) for $\mathbf{B}_n$ to continue, one reason the #'s were retained.

Moves needed for the second stage of operation, the d consecutive left-to-right-to-left passes across the tape, include the following rightward moves for $1 \leqslant i, j \leqslant n$:

$$\begin{aligned}
&\langle a_i , s', k, l\rangle \to \langle s', k+1, l\rangle \qquad \text{if} \quad k < l-1,\\
&\langle a_i , s', l-1, l\rangle \to \langle s_i , l, l\rangle,\\
&\langle a_j , s_i , k, l\rangle \to \langle s_i , k+1, l\rangle \qquad \text{for} \quad k \geqslant l,\\
&\langle a_j', s_i , k, l\rangle \to \langle s_i , k-1, l\rangle \qquad \text{if} \quad k > l,\\
&\langle a_i', s_i , l, l\rangle \to \langle s', l-1, l\rangle,
\end{aligned}$$

and

$$\langle a_i', s', k, l\rangle \to \langle s', k-1, l\rangle \qquad \text{for any } k > 0.$$

This stage operates similarly to the first in the way that the first counter is used, but note that the second counter in the situation $\langle \#, s, 0, l\rangle$, at the right end marker, should now be decreased by one, the (possibly several) symbols of depth l having just been checked for matching. Thus we give it the leftward move

$$\langle s_0 , 0, l-1\rangle \leftarrow \langle \#, s', 0, l\rangle, \qquad \text{if} \quad l > 1,$$

which—together with earlier given moves for state s_0—permits it to count down to 1 in d complete passes.

The explanation for the second stage moves is now easily supplied. Note that $\mathbf{B}_n$ stays in state s' (on rightward passes) unless it reaches a situation $\langle a_i, s', l-1, l\rangle$, to which there must later correspond a checking situation $\langle a_i', s_i, l, l\rangle$. Otherwise, $\mathbf{B}_n$ halts in a rejecting state since the only accepting situation is $\langle \#, s', 0, 1\rangle$, i.e., $A = \{s'\}$, and $\mathbf{B}_n$ halts in state s' only in this situation. The failure of a paren, a_i, to match with a thesis, a_j', when $i \neq j$, causes $\mathbf{B}_n$ to halt in state $s_i \in S - A$. Other rejecting situations, in the first stage, include $\langle \#, q_0, 0, 0\rangle$ and $\langle a_i', q_0, 0, 0\rangle$, which force q_0 not to be an accepting state. Hence, $\mathbf{B}_n$ does not accept λ, but it does accept all (and only) nonnull words in D_n; only these words with properly matched labels on equally deeply nested parens and theses will "survive" both stages of operation. Since the BCA $\mathbf{B}_n$ does recognize D_n^+, this λ-free Dyck language is recognizable by MA, by Theorem 1. Q.E.D.

RECEIVED: May 19, 1971

REFERENCES

AHO, A. V. AND ULLMAN, J. D. (1968), The theory of languages, *Math. Systems Theory* **2**, 97–125.

BAR-HILLEL, Y., PERLES, M., AND SHAMIR, E. (1961), On formal properties of simple phrase structure grammars, *Z. Phonetik Sprachwiss Kommunikat.* **14**, 143–172.

CHOMSKY, N. (1959), On certain formal properties of grammars, *Information and Control* **2**, 137–167.

CHOMSKY, N. AND SCHUTZENBERGER, M. P. (1963), The algebraic theory of context-free languages, *in* "Computer Programming and Formal Systems," (P. Braffort and D. Hirschberg, Eds.), pp. 118–161, North Holland, Amsterdam.

GINSBURG, S. (1966), "The Mathematical Theory of Context-Free Languages," McGraw-Hill, New York.

GINSBURG, S. AND GREIBACH, S. (1966), Deterministic context-free languages, *Information and Control* **9**, 620–648.

GINSBURG, S. AND HARRISON, M. A. (1967), Bracketed context-free languages, *J. Comput. System Sci.* **1**, 1–23.

HARTMANIS, J. AND STEARNS, R. E. (1965), On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117**, 285–306.

KREIDER, D. AND RITCHIE, R. W. (1966), A basis theorem for a class of two-way automata, *Z. Math. Logik Grundlagen Math.* **12**, 243–255.

KURODA, S. Y. (1964), Classes of languages and linear-bounded automata, *Information and Control* **7**, 207–223.

PETERS, S. AND RITCHIE, R. W. (to appear), On the generative power of transformational grammars, *Information Sciences.*

RABIN, M. O. AND SCOTT, D. (1959), Finite automata and their decision problems, *IBM J. Res. Develop.* **3**, 114–125.

SPRINGSTEEL, F. N. (1967), "Context-free languages and marking automata," Doctoral dissertation, University of Washington, Seattle.