INFORMATION AND CONTROL 7, 207–223 (1964)

Classes of Languages and Linear-Bounded Automata*

S.-Y. KURODA

Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts

0. Let V_T and V_N be disjoint finite sets, and put $V = V_T \cup V_N$. A *grammar* G over the *terminal vocabulary* V_T and the *nonterminal vocabulary* V_N is a semi-Thue system (cf. Davis (1958)) in which the axiom is an element S of V_N, called the *initial symbol* of G, and for each production $PgQ \rightarrow P\bar{g}Q$ there are strings φ, ψ, ω in V and a string χ in V_N, such that $g = \varphi\chi\psi$ and $\bar{g} = \varphi\omega\psi$. Here P and Q are variables over strings in V. However, we will denote this production simply by $g \rightarrow \bar{g}$, dropping variables, and call it a *rule* of G. In general, for any strings σ, τ in V, we write $\sigma \rightarrow \tau$ if (σ, τ) satisfies a production of G, or in other words, if there is a rule $g \rightarrow \bar{g}$ of G such that $\sigma = \pi g \rho$, $\tau = \pi \bar{g} \rho$ for some π and ρ. By $\sigma \Rightarrow \tau$ we mean that τ is a theorem of G under the premise σ in the sense of the theory of combinatorial system, that is to say, $\sigma = \tau$ or there exist a finite number of strings $\sigma_1, \sigma_2, \cdots, \sigma_n$ such that $\sigma \rightarrow \sigma_1, \sigma_1 \rightarrow \sigma_2, \cdots, \sigma_{n-1} \rightarrow \sigma_n, \sigma_n \rightarrow \tau$. A string x *in* V_T is a *sentence* of G, if x is a theorem of G, that is to say, if $S \Rightarrow x$. The set of sentences of G is the *language generated* by G, which will be denoted by $L(G)$. It is a subset of the free semigroup V_T^* [1] generated by V_T. Conversely, a subset of the free semigroup V_1^* generated by a finite set V_1 is a *language*, if there exists a finite set V_2 and a grammar G with the terminals V_1 and the nonterminals V_2 which generates L. G is then *a grammar of* L. According to Chomsky (1959) G is context-sensitive, if each rule is of the type

* This work was supported in part by the U. S. Army, the Air Force Office of Scientific Research, and the Office of Naval Research; in part by the National Science Foundation (Grant G-16526); and in part by the National Institutes of Health (Grant MH-04737-03).

[1] In general, for a finite set V, we mean by V^* the free semigroup generated by V. The identity element of a semigroup will be denoted by I.

$$\chi_1 A \chi_2 \rightarrow \chi_1 \omega \chi_2$$

where A is a nonterminal and $\omega \neq I$. A context-sensitive grammar is context-free, if for each rule, $\chi_1 = \chi_2 = I$. A context-free grammar is one-sided linear if, for each rule, $\omega = aB$, or for each rule $\omega = Ba$, where a is a terminal and B is a nonterminal or I. Languages that have context-sensitive, constext-free and one-sided linear grammars are context-sensitive, context-free and one-sided linear languages, respectively. By a set of strings we understand a subset of the free semigroup generated by some finite set. Then, the following theorems are well-known.

THEOREM 0.1. *A set of strings is a one-sided linear language if and only if it is a regular event, and hence, equivalently, it is accepted by a finite automaton* (Chomsky, 1959).

THEOREM 0.2. *A set of strings is a context-free language if and only if it is accepted by a nondeterministic pushdown storage automaton* (Chomsky, 1962; Schützenberger, 1963).

THEOREM 0.3. *A set of strings is a language in the above sense if and only if it is accepted by a Turing machine* (see, for example, Davis (1958)).

Recently, Landweber (1963) showed the following:

THEOREM 0.4. *A set of strings accepted by a linear-bounded automaton is a context-sensitive language.*

A linear-bounded automaton, as defined by Myhill (1960) and, following him, by Landweber, is a deterministic automaton specified by a quintuple $(\Sigma, V, s_0, \Phi, \mu)$, where the set Σ of states and the vocabulary V are finite sets, the initial state s_0 is an element of Σ, the set Φ of final states is a subset of Σ and, finally, the behavior function μ is a function from $V' \times \Sigma$ to $V' \times \Sigma \times \{-1, 0, 1\}$ where $V' = V \cup \{\#\}$, satisfying the condition: if $\mu(\#, s) = (a, s', k)$ then $a = \#$. $\#$ is a symbol outside V and called the boundary symbol.[2] The automaton functions as follows. It is given, as input, a tape blocked into squares containing a string $\#x\#$, where x is a string in V and $\#$ is the boundary symbol. Before operating on the input it is set in the initial state s_0. At the initial stage, then, it reads the left boundary in the state s_0. In general, if it is reading a letter a in a state s, and if $\mu(a, s) = (a', s', k)$, it prints a' in the scanned square, and moves k squares to the right (i.e., one square to the right, one square to the left, or no move at all, according as $k = 1, -1$ or 0, respectively) and enters in the state s'. Continuing the calcula-

[2] This condition means that the boundary symbol $\#$ is not affected during computation.

tion in this way, if it runs off the right end of the given tape and at this time it finds itself in one of the final states of Φ, then by definition the string x is *accepted*, or otherwise, *rejected*, by the automaton. The set of all strings accepted is the *language accepted* by the automaton.[3]

Now, if we allow the behavior function to be multivalued, we have a nondeterministic automaton. We understand, henceforth, by *linear-bounded automaton* a possibly nondeterministic automaton thus obtained. Only when there is possibility of confusion, we use the phrase "nondeterministic linear-bounded automaton" as a synonym of "linear-bounded automaton". A linear-bounded automaton in Myhill's sense is referred to as a deterministic linear-bounded automaton. We then mean by a string *accepted* by a nondeterministic automaton M, a string for which there is a computation of M which, given the string as input, ends up off the right end of the tape in a final state. On the other hand, a string is said to be *rejected* by M if there is a computation of M which, given the string as input, never ends, or ends up off the left end of the tape, or, finally, ends up off the right end of the tape in a nonfinal state. Because of the nondeterminacy of M, a string can in general be both accepted and rejected by M. The set of all strings accepted by M is called the *languages accepted by* M. The set of all strings rejected by M is called the *language rejected by* M.

It is easy to see that Landweber's proof of Theorem 0.3 does not depend on determinacy of the automaton, and Theorem 0.4 remains valid, if we understand, under our convention, the phrase "linear-bounded automaton" as meaning "nondeterministic linear-bounded automaton".

[3] This definition of linear-bounded automaton is essentially the same as given by Myhill (1960), but looks somewhat different from it in that this makes clearer the role of the boundary marker. It is easily seen that the left boundary marker is dispensable. Indeed, we have only to duplicate the vocabulary and have, for each letter, so to speak, a marked copy. At the first scanning of the leftmost letter, the automaton is now supposed to replace it by the marked copy and afterwards marked letters are supposed to be replaced only by marked ones. Thus at the left end stands always a marked letter and the automaton can recognize the left end of the tape. On the other hand, the right boundary marker is not dispensable, but we have to refer to it only once. In fact at the beginning of computation, one could replace the rightmost letter, i.e., the letter preceding the boundary marker, by the marked copy of it. Then in the same way as at the left end, the automaton can now recognize the right end of the tape. Hence, we could rather conveniently make a convention once and for all, that a linear-bounded automaton can recognize both ends of the tape and that the behavior function is a function from $V \times \Sigma$, rather than from $V' \times \Sigma$ as defined above.

Then, the converse of Theorem 0.3, under this interpretation, will be proved in Section 1, and hence it will be established that a set of strings is a context-sensitive language if and only if it is accepted by a nondeterministic linear-bounded automaton (Theorem 1). This will be a supplement to the previous results (Theorems 0.1, 0.2, and 0.3) concerning the parallelism between the hierarchy of types of grammars and the hierarchy of types of automata, and, in a sense, completes the automata theoretical characterization of the types of grammars proposed by Chomsky (1959).

On the other hand, a language rejected by a linear-bounded automaton will also be shown to be a context-sensitive language. Indeed we can construct another linear-bounded automaton which accepts this language (Theorem 2). When, in particular, it is rejected by a deterministic automaton, a linear-bounded automaton which accepts it can also be deterministic, and since it is nothing but the complement of the language accepted by the automaton, it follows that the class of languages accepted by deterministic linear-bounded automata forms a Boolean algebra.

Finally, it will be shown that context-free languages are accepted by deterministic linear-bounded automata (Theorem 3).

1. As remarked in Chomsky (1959), if a language L has a grammar such that, for each of its rules $\varphi \rightarrow \psi$, the length of ψ is not smaller than that of φ, then there is a context-sensitive grammar G which generates L. Hence, in the sequel, we agree to call such a kind of grammar also context-sensitive. Notice further that we can impose on a context-sensitive grammar G, without affecting its generative capacity, a restriction that no rule involves terminal symbols except such rules as are of the form $A \rightarrow a$ where A and a are a nonterminal symbol and a terminal symbol, respectively.

DEFINITION 1. *A context-sensitive grammar is of order n if there appears no string of length greater than n in any rule of the grammar.*

We say that two grammars G and G' are *equivalent* if they generate the same languages: $L(G) = L(G')$.

LEMMA 1. *For any order n context-sensitive grammar $G(n \geqq 3)$ there exists an order $n - 1$ context-sensitive grammar G' equivalent to G.*

PROOF: As remarked above, we can assume that no rule of G involves a terminal symbol unless it is of the form $A \rightarrow a$. Let $\varphi \rightarrow \psi$ be a rule of G. If the length of ψ is less than 3, let it be a rule of G'. Otherwise

we can write $\varphi = A\varphi'$, $\psi = BCD\psi'$, where A, B, C and D are nonterminal symbols. If $\varphi' = I$, introduce two new symbols A_1 and A_2, and let

$$A \to A_1A_2$$

$$A_1 \to BC$$

$$A_2 \to D\psi'$$

be rules of G'. If $\varphi' \neq I$, write $\varphi' = E\varphi''$, E being a nonterminal symbol, introduce two new symbols A' and E', and let

$$AE \to A'E'$$

$$A' \to B$$

$$E'\varphi'' \to CD\psi'$$

be rules of G'. G' thus obtained is of order $n - 1$ and equivalent to G.

By repeated use of Lemma 1, we get:

LEMMA 2. *A context-sensitive grammar is equivalent to an order* 2 *grammar.*

Notice that if, for each rule $AB \to CD$ of an order 2 context-sensitive grammar, we introduce a new symbol A' and replace the rule by three rules: $AB \to A'B$, $A'B \to A'D$ and $A'D \to CD$, we get a new grammar which is equivalent to the old one and context-sensitive as initially defined at the beginning of this paper. Thus we have actually proved the remark made at the beginning of this section.

DEFINITION 2. *A context-sensitive grammar is length-preserving if for any rule* $\varphi \to \psi$, (1) φ *is the initial symbol, or* (2) ψ *does not contain the initial symbol and the lengths of* φ *and* ψ *are equal.*

Hence, in particular, rules of an order 2 length-preserving grammar are of one of the following types:

$$A \to C$$

$$AB \to CD$$

$$S \to EF$$

where C and D are not the initial symbol.

DEFINITION 3. *A context-sensitive grammar is linear-bounded if it is of order* 2 *and length-preserving and if, S being the initial symbol,* $S \to EF$ *implies* $E = S$.

LEMMA 3. *For any order* 2 *context-sensitive grammar G, there exists a linear-bounded grammar* G' *equivalent to G.*

PROOF: Let V_T, V_N, and S be the set of terminals, the set of nonterminals, and the initial symbol of G, respectively, and put $V_N' = \{S', Q\} \cup V_N$, where Q and S' are new symbols. The rules of a grammar G' with the terminals V_T, the nonterminals V_N', and the initial symbol S' will be defined as follows:

$$\begin{aligned}
&S' \to S'Q, \\
&S' \to S, \\
&\left.\begin{aligned} QA &\to AQ, \\ AQ &\to QA, \end{aligned}\right\} && \text{for all } A \text{ in } V_T \cup V_N. \\
&A \to B, && \text{if } A \to B \text{ is a rule of } G, \\
&AB \to CD, && \text{if } AB \to CD \text{ is a rule of } G, \\
&AQ \to BC, && \text{if } A \to BC \text{ is a rule of } G.
\end{aligned}$$

G' thus defined is clearly linear-bounded. We shall see that it is equivalent to G. Define a map f from $(V_T \cup V_N')^*$ to $(V_T \cup V_N)^*$ by putting

$$\begin{aligned}
f(S') &= S \\
f(Q) &= I \\
f(A) &= A && \text{for } A \text{ in } V_T \cup V_N, \\
f(xy) &= f(x)f(y), && \text{for any strings } x \text{ and } y.
\end{aligned}$$

Then, if $x \to y$ in G', $f(x) \Rightarrow f(y)$ in G, and hence, in particular, if y is a string in V_T and $S' \Rightarrow y$ in G', it follows that $S \Rightarrow y$ in G. This shows that $L(G') \subset L(G)$. Conversely, if $x \to y$ in G, then $x \to y$ in G' or $xQ \Rightarrow y$ in G', and hence, in general, if $x \Rightarrow y$ in G, it follows that, for some $n \geqq 0$, $xQ^n \Rightarrow y$ in G'. In particular, if y is a string in V_T and $S \Rightarrow y$ in G, we have $SQ^n \Rightarrow y$ in G' for some n, and by the first two rules of G' given above, it follows that $S' \Rightarrow y$ in G'. This means that $L(G) \subset L(G')$.

LEMMA 4. *For any linear-bounded grammar G there exists a linear-bounded automaton which accepts the language $L(G)$ generated by G.*

PROOF: Let V_T, V_N, and S be the set of terminals, the set of nonterminals, and the initial symbol of G, respectively. We shall construct a linear-bounded automaton M with the set V as alphabet and the set of states Σ, as follows. We put $V = V_T \cup V_N$ and let Σ have states t_0, t_1, s_0, s_1, r_0, r_1, and s_A for each A such that there is a rule of the

type $AB \to CD$. t_0 will be the initial state and a unique final state. The instructions of M are as given below:

$$
\begin{aligned}
&(\#, t_0) \to (\#, t_1, 1), && \\
&(a, t_1) \to (a, t_1, 1) && \text{for all } a \text{ in } V_T, \\
&(\#, t_1) \to (\#, s_0, -1), && \\
&(A, s_0) \to (A, s_0, 1) && \text{for all } A \text{ in } V, \\
&(A, s_0) \to (A, s_0, -1) && \text{for all } A \text{ in } V, \\
&(B, s_0) \to (A, s_0, 0) && \text{for all rules } A \to B \text{ of } G, \\
&\left.\begin{aligned}(C, s_0) &\to (A, s_A, 1)\\ (D, s_A) &\to (B, s_0, 0)\end{aligned}\right\} && \text{for all rules } AB \to CD \text{ of } G, \\
&(S, s_0) \to (S, r_0, -1) && \\
&(\#, r_0) \to (\#, r_1, 1) && \\
&\left.\begin{aligned}(S, r_1) &\to (\#, s_1, 1)\\ (A, s_1) &\to (S, s_0, 0)\end{aligned}\right\} && \text{for all rules } S \to SA \text{ of } G, \\
&(\#, s_1) \to (\#, t_0, 1), && \\
&(X, x) \to (X, x, 0) && \text{for all other pairs } (X, x).
\end{aligned}
$$

It is easily verified that M, thus defined, accepts $L(G)$. Indeed, first notice that the states t_0 and t_1 are introduced to check whether an input is a string in the terminals placed between two #'s. If so, M goes to the states s. If not, the input is rejected. Next, assume that M scans in the state s_0 any one of the squares of a tape x and that, after a finite number of applications of instructions, M is scanning a tape y in the state s_0. Then we see that x is derivable from y in G. Hence, if, given a string in the terminals, M yields a tape $\#\# \cdots \#S\#$ and is scanning S in the state s_0, then the string is in $L(G)$. Now, without loss of generality, we can assume that there is actually a rule $S \to SA$ in G, and hence, correspondingly, we have an instruction $(S, r_1) \to (\#, s_1, 1)$, to the effect that, after the above configuration, M yields a tape $\#\# \cdots \#$, scanning # in s_1, and then scans off the right end in t_0, accepting the tape. Notice that this is the only situation in which M runs off the right end in the final state. Thus, M accepts at most strings in $L(G)$. Conversely, if x is in $L(G)$, M can simulate an S-derivation of x backwards. Hence $L(G)$ is just the language accepted by M.

It follows immediately from these lemmas that for any context-sensitive language there exists a linear-bounded automaton which generates it. Combining this with Theorem 0.4 we have:

THEOREM 1. *A set of strings is a context-sensitive language if, and only if, it is accepted by a linear-bounded automaton.*

Landweber (1963) showed that the intersection of two context-sensitive languages is also a context-sensitive language. The following corollary gives another proof to the same result.

COROLLARY. *The intersection L of two context-sensitive languages L_1 and L_2 is a context-sensitive language.*

PROOF: Let M_1 and M_2 be linear-bounded automata which generate L_1 and L_2, respectively. We can surely construct a linear-bounded automaton M which acts as follows. Given an input x, M stores x, say, on the lower half of the tape and calculates on the upper half, simulating M_1. If M_1 accepts x, M goes back to the left end and starts calculation on the lower half, simulating this time M_2. If M_2 also accepts x, M is made to accept x. In all other cases, M is not made to finish calculation successfully. Then, the language M accepts is the intersection L of L_1 and L_2. This proves that L is a context-sensitive language.

2. Let M be a linear-bounded automaton and Σ' a subset of the set Σ of states of M. We define $L(M, D, \Sigma')$, where D is either L or R standing for left or right, as the set of strings x such that, given x as an input, there is a computation of M in which M stops off the left end, if $D = L$, or off the right end, if $D = R$, in one of the states in Σ'. Further, $L(M, \omega)$ is the set of strings x such that given x as an input M possibly never stops. In general, these sets are not mutually disjoint. But if M is deterministic, any two of such sets are disjoint except possibly when they are specified by the same D and two subsets Σ_1' and Σ_2' not disjoint from each other.

THEOREM 2. *Let M be a linear-bounded automaton and let*

$$L = L(M, D, \Sigma') \qquad \text{or} \qquad L = L(M, \omega).$$

Then, there exists a linear-bounded automaton M' which accepts L. If, in particular, M is deterministic, so is M'.

PROOF: $L(M, R, \Sigma')$ is nothing but the language accepted by a linear-bounded automaton which differs from M only in that Σ' is designated as the set of final states. It is also immediately obvious that a trivial modi-

fication of M yields a linear-bounded automaton to accept $L(M, L, \Sigma')$. What is left is the case of $L = L(M, \omega)$.

As the length of an input tape is left fixed during computation by M, the number of all possible configurations of mechanism involved during the computation is bounded, and actually by $n^l lm$, where n, m, and l are the number of letters of M, the number of states of M, and the length of the input, respectively. Thus, if the computation does not end after $n^l lm$ steps, there is a configuration of the machine which recurs in the computation and hence the input has a computation which, having a cycle of configuration, never ends. Let M' be a machine which consists of M and a counter which functions in such a way that, given an input, it simulates M and at the same time it counts the number of steps of computation, and further that it will accept an input of length l if and only if the content of the counter exceeds $n^l lm$. Then, the language accepted by M' is just $L(M, \omega)$. As the counter has to count only up to $n^l lm$, it can be a linear-bounded automaton, and hence also is M'.

We are indebted for this theorem to Mr. L. Haines. We first proved the following corollary by constructing a context-sensitive grammar of $L(M, \omega)$, directly from the instructions of M by a generalization of the method used by Landweber (1963) in the proof of Theorem 0.2. In view of Theorem 1, it is equivalent to the first half of the theorem. It is weaker than the theorem in that it gives no information as to the determinacy of M'.

COROLLARY 1. *$L(M, D, \Sigma')$ and $L(M, \omega)$ are all context-sensitive languages.*

For the sake of interest of construction, we add our original proof of the corollary.

Construction of context-sensitive grammars for $L(M, D, \Sigma')$ and $L(M, \omega)$: $L(M, R, \Sigma')$ is nothing but the language accepted by a linear-bounded automaton which differs from M only in that Σ' is designated as the set of final states, and hence by Theorem 0.4 it is a context-sensitive language. It is immediately obvious that the same is true for $L(M, L, \Sigma')$ by a trivial modification of the theorem. What is left to us is, hence, to construct a context-sensitive grammar for $L(M, \omega)$.

As remarked in the proof of Theorem 2, if the computation is to continue infinitely, some of the configurations should recur. Thus we have a sequence of configurations $C_0, C_1, \cdots, C_h, \cdots, C_g$ such that, with input x, M starts computation at the initial configuration C_0, and

passes the C_i's successively as it computes, where all C_i's are mutually distinct for $h \leqq i < g$ while $C_h = C_g$. Conversely, if we have a sequence of configurations of the above kind a tape found in the initial configuration C_0 is in $L(M, \omega)$. Hence, just as Landweber (1963), in the proof of Theorem 0.4, traced a sequence of configurations in computation of a tape accepted by M backwards from the final configuration, we have only to trace sequences of the above kind backwards from a knot of a loop and express our trace in terms of derivations on grammar.

To represent formally configurations during computation, we could make use of the same device as Landweber (1963). Thus, when $\Sigma = \{s_1, \cdots, s_m\}$ and $V = \{a_1, \cdots, a_n\}$ are the set of states and the alphabet of M, respectively, we introduce new symbols b_{ip}, $1 \leqq i \leqq n$, $1 \leqq p \leqq m$, and represent by a string $a_{i_1}a_{i_2} \cdots a_{i_{k-1}}b_{i_kp}a_{i_{k+1}} \cdots a_{i_l}$ a configuration with a tape $a_{i_1}a_{i_2} \cdots a_{i_{k-1}}a_{i_k}a_{i_{k+1}} \cdots a_{i_l}$ and a state s_p scanning a_{i_k}. However, this apparatus alone is not sufficient for us, since we should make sure that we have made a loop during our derivation, and for that purpose we should be equipped with some device or another to memorize the configuration C_g until we reach C_h. To meet this, we further introduce doubly indexed a_{ij}, $1 \leqq i, j \leqq n$, and b_{ijpq}, $1 \leqq i, j \leqq n$, $1 \leqq p, q \leqq m$, which, so to speak, stand, at the same time, for a letter a_i appearing on tape at C_g and a letter a_j on tape at C_k, $h \leqq k \leqq g$, on the one hand, and on the other, for b_{ip} in C_g and b_{jq} in C_k, respectively. One more thing that should be memorized during the derivations from C_g through C_h is the particular square on tape which has been scanned at C_g. Hence, we further introduce symbols a'_{ij}, b'_{ijpq}, each of which is to mean that it is a result of sequential replacements beginning from a particular letter scanned at C_g. Finally, two occurrences of one and the same configuration $C_h = C_g$ are required to be distinguished, and it suffices to introduce additional symbols b''_{iipp}, $1 \leqq i \leqq n$, $1 \leqq p \leqq m$, to denote the scanned symbol a_i and the state s_p scanning a_i at C_g.

With this informal explanation in mind, we shall now define a grammar G to generate $L(M, \omega)$. The sets V_T and V_N of the terminals and of the nonterminals of G are given by:

$$V_T = V$$

$$V_N = \{S, A, a_{ij}, a'_{ij}, b_{ip}, b''_{iipp}, b_{ijpq}, b'_{ijpq}\}.$$

G will have the following rules:

(0) $S \to b''_{iipp}A,\ Ab''_{iipp},\ Ab''_{iipp}A,\ b''_{iipp}$

$A \to a_{ii}A,\ a_{ii}.$

(1) If $(a_k, s_q) \to (a_i, s_p, -1)$ is an instruction of M,

$b_{jp}a_i \to a_j b_{kq}$

$b_{i_1 jrp}a_{i_2 i} \to a_{i_1 j}b_{i_2 krq}$

$b'_{i_1 jrp}a_{i_2 i} \to a'_{i_1 j}b_{i_2 krq}$

$b_{i_1 jrp}a'_{i_2 i} \to a_{i_1 j}b'_{i_2 krq}$

$b''_{i_1 i_1 pp}a_{ii} \to a'_{i_1 i_1}b_{ikpq}$

for $1 \leqq i_1, i_2, j \leqq n$, $1 \leqq r \leqq m$.

(2) If $(a_k, s_q) \to (a_i, s_p, 1)$ is an instruction of M,

$a_i b_{jp} \to b_{kq}a_j$

$a_{i_1 i}b_{i_2 jrp} \to b_{i_1 krq}a_{i_2 j}$

$a'_{i_1 i}b_{i_2 jrp} \to b'_{i_1 krq}a_{i_2 j}$

$a_{i_1 i}b'_{i_2 jrp} \to b_{i_1 krq}a'_{i_2 j}$

$a_{ii}b''_{i_1 i_1 pp} \to b_{ikpq}a'_{i_1 i_1}$

for $1 \leqq i_1, i_2, j \leqq n$, $1 \leqq r \leqq m$.

(3) $(a_k, s_q) \to (a_i, s_p, 0)$ is an instruction of M,

$b_{ip} \to b_{kq}$

$b_{i_1 irp} \to b_{i_1 krq}$

$b'_{i_1 irp} \to b'_{i_1 krq}$

$b''_{iipp} \to b'_{ikpq}$

for $1 \leqq i_1 \leqq n$, $1 \leqq r \leqq m$.

(4) $b'_{iipp} \to b_{ip}$

$a_{ii}b_{jp} \to a_i b_{jp}$

$b_{jp}a_{ii} \to b_{jp}a_i$

$a_{ii}a_j \to a_i a_j$

$a_j a_{ii} \to a_j a_i$,

for $1 \leqq i \leqq n$, $1 \leqq p \leqq m$.

(5) $\#b_{i1} \rightarrow \#a_i$

for $1 \leqq i \leqq n$, assuming that s_1 is the initial state of M.

We shall sketch how this grammar generates $L(M, \omega)$. Let us understand by the phrase "singly (doubly) indexed a-symbols," the symbols in G represented by a's with single (double) index(es) and by the phrase "singly (doubly) indexed b-symbols," the symbols represented by b's with two (four) indexes. A string is singly (doubly) indexed if it consists solely of singly (doubly) indexed symbols and if it contains a unique b-symbol (and a unique primed symbol). For a singly indexed string x, let us mean by $T(x)$ a string in singly indexed a's obtained from x by replacing its unique b-symbol by an a-symbol with the same index as the first index of the b-symbol. For a doubly indexed string x, let us mean by $T_1(x)(T_2(x))$ a string in singly indexed a's whose ith member has the same index as the first (second) index of the ith member of x. For a singly (doubly) indexed string x whose kth member is a b-symbol, let us mean by $C(x)(C_1(x), C_2(x))$ a configuration of M consisting of a tape $T(x)(T_1(x), T_2(x))$ with the kth square scanned by M in the state s_q, where q is the second (third, fourth) index of the b-symbol of x.

Now, by the rules in (0), a doubly indexed string x is generated whose first and second (third and fourth) indexes are the same in each symbol and whose b-symbol is doubly primed:

$$x = a_{i_1 i_1} a_{i_2 i_2} \cdots a_{i_{k-1} i_{k-1}} b''_{i_k i_k p p} a_{i_{k+1} i_{k+1}} \cdots a_{i_l i_l} .$$

One of the final rules of (1), (2), or (3) is applicable to such a string x if and only if there is a configuration of M from which the configuration $C_1(x) = C_2(x)$ is derived directly by a single application of an instruction, and then the rule yields a doubly indexed string y such that one of its symbols is singly primed and $T_1(y) = T_1(x)$. In general, one of the rules in (1), (2), or (3) is applicable to a doubly indexed string y if and only if there exists a configuration of M from which the configuration $C_2(y)$ is directly derivable, and then the rule yields another doubly indexed string y' such that $T_1(y) = T_1(y')$. Rules in (4) are successively applicable to a doubly indexed string y and yield a singly indexed string z, if and only if it happens that y is diagonal (i.e., for each member, the first and the second indexes coincide and so do also the third and the fourth for a b-symbol), and that the b-symbol of y is singly primed, or, in other words, if and only if $C_1(y) = C_2(y)$. One of the rules in

(1), (2), or (3) is applicable to a singly indexed string z, if and only if there is a configuration of M from which the configuration $C(z)$ is directly derivable, and then the rule yields another singly indexed string. Finally, the rule (5) is applicable to a singly indexed string z to yield a terminal string w, if and only if it happens that the b-symbol of z comes at the left end of z and its second index is 1, or, in other words, if and only if $C(z)$ is an initial configuration of M. Hence we see that z is derivable from S in G, if and only if z is in $L(M, \omega)$.

The language rejected by M is the union of $L(M, \omega)$, $L(M, L, \Sigma)$, and $L(M, R, \Sigma')$, where Σ' is the complement of Σ. In general it is not disjoint from the language $L(M)$ accepted by M. But if M is deterministic it is the complement of $L(M)$ in the free semigroup V^* over the alphabet V of M. It follows from Theorem 2:

COROLLARY 2. *The language rejected by a linear-bounded automaton is a context-sensitive language.*

Let L be a subset of V_1^*, V_1 being a finite set. When there exists a linear-bounded automaton M which accepts L, it may not be the case that the alphabet V of M is equal to V_1, but it may rather be larger than V_1. Insofar as we are interested in whether strings in V_1 belong to L, we could conveniently make a convention that only strings in V_1 are fed as inputs into M. If M' is another linear-bounded automaton with the same alphabet as M, which possibly accepts more strings in V than M but accepts just the same strings in V_1 as M, then under the above convention, it turns out that M and M' are two devices which have the same behavior. Hence, introducing the notion of "input alphabet," we could conveniently modify the notion of acceptance by a linear-bounded automaton as follows. Let L be a subset of V_1^*. L is *accepted* by M if a string *in* V_1 is accepted by M just when it is in L. When necessary, this kind of acceptance might be referred to in terms of "relative to an input alphabet V_1," or "with respect to V_1." We can also define the notion of rejection in a relative sense. Anyway, however, if a set of strings is accepted or rejected relatively by some linear-bounded automaton, there is another one that accepts or rejects it absolutely, and if the former is deterministic, the latter can also be deterministic.

In particular, let L be a set of strings in a finite set V_1 and assume it is accepted by a deterministic linear-bounded automaton M. Then the complement L' of L in the usual sense, i.e., the complement in V_1^*, is the set rejected by M relatively to V_1. We have thus the following:

COROLLARY 3. *The complement of a language accepted by a deterministic linear-bounded automaton is accepted by a deterministic linear-bounded automaton, and* a fortiori *is a context-sensitive language.*

Since we can construct a deterministic linear-bounded automaton to accept the intersection of the languages accepted by deterministic linear-bounded automata (cf. the proof of Corollary to Theorem 1), we have:

COROLLARY 4. *The class* $\mathfrak{L}_1'$ *of languages accepted by deterministic linear-bounded automata forms a Boolean algebra.*

According to Myhill (1960), a set L is strongly representable by a deterministic linear-bounded automaton M, if, in our terminology, M accepts L and $L(M, L, \Sigma) = L(M, \omega) = \phi$. From the proof of Theorem 2, it is seen that, if a set L is ever accepted by a deterministic linear-bounded automaton, there is another one which strongly represents L. Thus, the class $\mathfrak{L}_1'$ is actually the same as the class of languages strongly representable. By the main theorem of Myhill (1960), this class contains the class of rudimentary attributes.

It is well-known that the class of regular events is closed under complementation but that neither the class of recursively enumerable sets nor the class of context-free languages is closed under complementation. It is, however, not known whether the class $\mathfrak{L}_1$ of context-sensitive languages is closed under complementation. Furthermore, it is known that nondeterministic finite automata and nondeterministic Turing machines are not essentially more powerful than (deterministic) finite automata and Turing machines, respectively, although nondeterministic pushdown storage automata are essentially more powerful than deterministic pushdown storage automata. However, we do not know if nondeterministic linear-bounded automata are essentially more powerful than deterministic linear-bounded automata. Corollary 4 indicates a relation between these two open questions: If nondeterministic linear-bounded automata are not essentially more powerful than deterministic linear-bounded automata, the class $\mathfrak{L}_1$ of context-sensitive languages is closed under complementation, and hence is a Boolean algebra.

3. Finally in this section we consider a relation between context-free languages and linear-bounded automata.

THEOREM 3. *A context-free language is accepted by a deterministic linear-bounded automaton.*

PROOF: Let L be a context-free language. It is known that there

exists a normal grammar G which generates L. (A grammar is normal if all rules are of the type $A \to BC$ or $A \to a$, where A, B, and C are nonterminals and a is a terminal.) It suffices to construct a deterministic linear-bounded automaton M which accepts L relative to the terminal vocabulary V_T of G. M will be so constructed that it checks whether strings in V_T are in L or not by means of "analysis by synthesis."

Notice that, as G is normal, an S-derivation, in G, of a string x in L which is of length n, consists of $2n - 1$ successive direct steps, and further that there exists one S-derivation $D(x)$ in which each intermediate term of the derivation is obtained from the preceding term by replacement of the first nonterminal, say, from the left. Let $D(x)$ be $[\varphi_1, \varphi_2, \cdots, \varphi_{2n}]$, $\varphi_1 = S$, $\varphi_{2n} = x$, $\varphi_i \to \varphi_{i+1}$, $1 \leqq i < 2n$, and let R_i be a rule of G which derives φ_{i+1} from φ_i. Then $D(x)$ is fully characterized by a series of rules $R_1, R_2, \cdots, R_{2n-1}$, and hence Γ being the set of rules in G, by an element of Γ^{2n-1}, the $2n - 1$ Cartesian power of Γ. Introducing a definite order in Γ, we can further regard Γ^{2n-1} as the set of numbers 0 through $N^{2n-1} - 1$ coded in the N-ary system, N being the number of rules of G. The ith significant digit of the number refers to a rule to be applied to the ith term of the possible derivation referred to by the number.

By a suitable extension of the alphabet, we can assume that each square of a tape has four portions, or equivalently, that, combining two portions of squares into one tape, we equip M with three tapes, which we refer to as the top, middle, and bottom tapes, the bottom one being of length two times that of the others. Before the beginning of a calculation, an input is fed on the top tape. It will remain intact until the end of the calculation. On the bottom tape, an N-ary number will be stored. In general, each stage of calculation refers to a particular digit of the N-ary number currently stored on the bottom tape. Call this digit the digit of the stage. Now, before the beginning of a calculation, 0 will be stored on the bottom tape and the initial symbol S of G will be written at the left end of the middle tape. In general, each stage of a calculation will be as follows.

Let m be the N-ary number currently stored on the bottom tape when M enters this stage of calculation, and let the digit of the stage be the ith digit of m. The digit of the beginning stage is defined as the leftmost digit, i.e., $i = 1$. (Hence it is 0, since $m = 0$.) In general, m and i are determined from what has happened in the preceding stage in a way specified in the following.

(I) If the rule R referred to by the digit of the stage is applicable to the leftmost nonterminal A which M finds on the middle tape, M will replace the content of the middle tape by the string that will be derived by the application of the rule R to the content of the middle tape, and

(I.1) will enter into the next stage with i changed to $i + 1$, if $i \neq 2n - 1$;

(I.2) If $i = 2n - 1$, M will check whether the string on the middle tape coincides with the string stored on the top tape, and

(I.2.1) if it is the case, M will stop, accepting the input;

(I.2.2) otherwise,

(I.2.21) if $m \neq N^{2n-1} - 1$, M will add 1 to m, replace the present content of the middle tape by the single letter S at the left end, and enter into the next stage with m changed to $m + 1$ and i changed to 1, while

(I.2.22) if $m = N^{2n-1} - 1$, M will reject the input.

(II) If the rule R is not applicable to the leftmost nonterminal A, then the same as (I.2.2).

(III) If M finds no nonterminals on the middle tape, it will check whether the content of the middle tape coincides with the string stored on the top tape, and

(III.1) if it is the case, M will accept the input;

(III.2) otherwise, the same as (I.2.2).

This is an informal description of a deterministic linear-bounded automaton which accepts L. A formal realization of this informal description will be omitted.

From the theorem and Corollary 3 of Theorem 2, we have:

COROLLARY 1. *The Boolean algebra $\mathfrak{B}_2$ generated by the class $\mathfrak{L}_2$ of context-free languages is contained in the class $\mathfrak{L}_1'$ of languages accepted by deterministic linear-bounded automata.*

Finally, we add one more remark.

COROLLARY 2. *Nondeterministic pushdown storage automata can be simulated by deterministic linear-bounded automata.*

This is an immediate consequence of the theorem and Theorem 0.2.

ACKNOWLEDGMENT

I would like to express my gratitude to Professor Noam Chomsky of the Massachusetts Institute of Technology for his valuable suggestions and encouragement. My thanks are also due to Mr. S. Isard, who pointed out a mistake in my original draft.

RECEIVED: July 24, 1963

References

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

CHOMSKY, N. (1962), Context-free grammars and pushdown storage. *Quart. Progr. Rept. No. 65*, Research Laboratory of Electronics, M.I.T., pp. 187–194.

DAVIS, M. D. (1958), "Computability and Unsolvability." McGraw-Hill, New York.

LANDWEBER, P. S. (1963), Three theorems on phrase structure grammars of type 1. *Inform. Control* **6,** 131-137.

MYHILL, J. (1960), Linear bounded automata, WADD Tech. Note No. 60–165, Wright-Patterson Air Force Base, Ohio.

SCHÜTZENBERGER, M. P. (1963), Context-free languages and pushdown automata. *Inform. Control* **6,** 246-264.