INFORMATION AND CONTROL **8**, 607–639 (1965)

On the Translation of Languages from Left to Right

DONALD E. KNUTH

Mathematics Department, California Institute of Technology, Pasadena, California

There has been much recent interest in languages whose grammar is sufficiently simple that an efficient left-to-right parsing algorithm can be mechanically produced from the grammar. In this paper, we define LR(k) grammars, which are perhaps the most general ones of this type, and they provide the basis for understanding all of the special tricks which have been used in the construction of parsing algorithms for languages with simple structure, e.g. algebraic languages. We give algorithms for deciding if a given grammar satisfies the LR(k) condition, for given k, and also give methods for generating recognizers for LR(k) grammars. It is shown that the problem of whether or not a grammar is LR(k) for *some* k is undecidable, and the paper concludes by establishing various connections between LR(k) grammars and deterministic languages. In particular, the LR(k) condition is a natural analogue, for grammars, of the deterministic condition, for languages.

I. INTRODUCTION AND DEFINITIONS

The word "language" will be used here to denote a set of character strings which has been variously called a *context free language*, a (*simple*) *phrase structure language*, a *constituent-structure language*, a *definable set*, a *BNF language*, a *Chomsky type 2* (*or type 4*) *language*, a *push-down automaton language*, etc. Such languages have aroused wide interest because they serve as approximate models for natural languages and computer programming languages, among others. In this paper we single out an important class of languages which will be called *translatable from left to right*; this means if we read the characters of a string from left to right, and look a given finite number of characters ahead, we are able to parse the given string without ever backing up to consider a previous decision. Such languages are particularly important in the case of computer programming, since this condition means a parsing algorithm can be mechanically constructed which requires an execution time at worst proportional to the length of the string being parsed. Special-purpose methods for translating computer languages (for example, the well-known precedence algorithm, see Floyd (1963)) are based on the fact that the languages being considered have a simple left-to-right structure. By considering all languages that are translatable from left to right, we are able to study all of these special techniques in their most general framework, and to find for a given language and grammar the "best possible" way to translate it from left to right. The study of such languages is also of possible interest to those who are investigating human parsing behavior, perhaps helping to explain the fact that certain English sentences are unintelligible to a listener.

Now we proceed to give precise definitions to the concepts discussed informally above. The well-known properties of *characters* and *strings* of characters will be assumed. We are given two disjoint sets of characters, the "*intermediates*" I and the "*terminals*" T; we will use upper case letters A, B, C, $\cdots$ to stand for intermediates, and lower case letters a, b, c, $\cdots$ to stand for terminals, and the letters X, Y, Z will be used to denote either intermediates or terminals. The letter S denotes the "principal intermediate character" which has special significance as explained below. Strings of characters will be denoted by lower case Greek letters α, β, γ, $\cdots$, and the *empty string* will be represented by ϵ. The notation α^n denotes n-fold concatenation of string α with itself; $\alpha^0 = \epsilon$, and $\alpha^n = \alpha\alpha^{n-1}$. A *production* is a relation $A \rightarrow \theta$ where A is in I and θ is a string on $I \cup T$; a *grammar* $\mathcal{G}$ is a set of productions. We write $\varphi \rightarrow \psi$ (with respect to $\mathcal{G}$, a grammar which is usually understood) if there exist strings α, θ, ω, A such that $\varphi = \alpha A \omega$, $\psi = \alpha\theta\omega$, and $A \rightarrow \theta$ is a production in $\mathcal{G}$. The transitive completion of this relation is of principal importance: $\alpha \Rightarrow \beta$ means there exist strings α_0, α_1, $\cdots$, α_n (with $n > 0$) for which $\alpha = \alpha_0 \rightarrow \alpha_1 \rightarrow \cdots \rightarrow \alpha_n = \beta$. Note that by this definition it is not necessarily true that $\alpha \Rightarrow \alpha$; we will write $\alpha \equiv> \beta$ to mean $\alpha = \beta$ or $\alpha \Rightarrow \beta$. A grammar is said to be *circular* if $\alpha \Rightarrow \alpha$ for some α. (Some of this notation is more complicated than we would need for the purposes of the present paper, but it is introduced in this way in order to be compatible with that used in related papers.) The *language defined by* $\mathcal{G}$ is

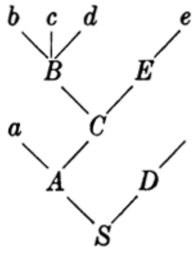
$$\{\alpha \mid S \Rightarrow \alpha \text{ and } \alpha \text{ is a string over } T\}, \tag{1}$$

namely, the set of all terminal strings derivable from S by using the productions of $\mathcal{G}$ as substitution rules. A *sentential form* is any string α for which $S \Rightarrow \alpha$.

For example, the grammar

$$S \to AD,\ A \to aC,\ B \to bcd,\ C \to BE,\ D \to \epsilon,\ E \to e \tag{2}$$

defines the language consisting of the single string "*abcde*". Any sentential form in a grammar may be given at least one representation as the leaves of a *derivation tree* or "parse diagram"; for example, there is but one derivation tree for the string *abcde* in the grammar (2), namely,



(3)

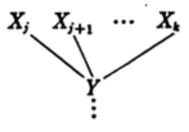
(The root of the derivation tree is S, and the branches correspond in an obvious manner to applications of productions.) A grammar is said to be *unambiguous* if every sentential form has a unique derivation tree. The grammar (2) is clearly unambiguous, even though there are several different *sequences* of derivations possible, e.g.

$$S \to AD \to aCD \to aBED \to abcdED \to abcdeD \to abcde \tag{4}$$

$$S \to AD \to A \to aC \to aBE \to aBe \to abcde \tag{5}$$

In order to avoid the unimportant difference between sequences of derivations corresponding to the same tree, we can stipulate a particular order, such as insisting that we always substitute for the leftmost intermediate (as done in (4)) or the rightmost one (as in (5)).

In practice, however, we must start with the terminal string *abcde* and try to reconstruct the derivation leading back to S, and that changes our outlook somewhat. Let us define the *handle* of a tree to be the leftmost set of adjacent leaves forming a complete branch; in (3) the handle is *bcd*. In other words, if $X_1, X_2, \cdots, X_t$ are the leaves of the tree (where each X_i is an intermediate or terminal character or ϵ), we look for the smallest k such that the tree has the form



for some j and Y. If we consider going from $abcde$ backwards to reach S, we can imagine starting with tree (3), and "pruning off" its handle; then prune off the handle ("e") of the resulting tree, and so on until only the root S is left. This process of pruning the handle at each step corresponds exactly to derivation (5) in reverse. The reader may easily verify, in fact, that "handle pruning" always produces, in reverse, the derivation obtained by replacing the *rightmost* intermediate character at each step, and this may be regarded as an alternative way to define the concept of handle. During the pruning process, all leaves to the right of the handle are terminals, if we begin with all terminal leaves.

We are interested in algorithms for parsing, and thus we want to be able to recognize the handle when only the leaves of the tree are given. Number the productions of the grammar 1, 2, $\cdots$, s in some arbitrary fashion. Suppose $\alpha = X_1 \cdots X_n \cdots X_t$ is a sentential form, and suppose there is a derivation tree in which the handle is $X_{r+1} \cdots X_n$, obtained by application of the pth production. ($0 \leqq r \leqq n \leqq t$, $1 \leqq p \leqq s$.) We will say (n, p) is *a handle* of α.

A grammar is said to be *translatable from left to right with bound* k (briefly, an "LR(k) grammar") under the following circumstances. Let $k \geqq 0$, and let "$\dashv$" be a new character not in $I \cup T$. A k-sentential form is a sentential form followed by k "$\dashv$" characters. Let $\alpha = X_1X_2 \cdots X_nX_{n+1} \cdots X_{n+k}Y_1 \cdots Y_u$ and $\beta = X_1X_2 \cdots X_nX_{n+1} \cdots X_{n+k}Z_1 \cdots Z_v$ be k-sentential forms in which $u \geqq 0$, $v \geqq 0$ and in which none of $X_{n+1}, \cdots, X_{n+k}, Y_1, \cdots, Y_u, Z_1, \cdots, Z_v$ is an intermediate character. If (n, p) is a handle of α and (m, q) is a handle of β, we require that $m = n$, $p = q$. In other words, a grammar is LR(k) if and only if any handle is always uniquely determined by the string to its left and the k terminal characters to its right.

This definition is more readily understandable if we take a particular value of k, say $k = 1$. Suppose we are constructing a derivation sequence such as (5) in reverse, and the current string (followed by the delimiter $\dashv$ for convenience) has the form $X_1 \cdots X_nX_{n+1}\alpha \dashv$, where the tail end "$X_{n+1}\alpha \dashv$" represents part of the string we have not yet examined; but all possible reductions have been made at the left of the string so that the right boundary of the handle must be at position X_r for $r \geqq n$. We want to know, by looking at the next character X_{n+1}, if there is in fact a handle whose right boundary is at position X_n; if so, we want this handle to correspond to a unique production, so we can reduce the string and repeat the process; if not, we know we can move to the right and read a new character of the string to be translated. This process will work if and only if the following condition ("LR(1)") always holds in the grammar: If $X_1X_2 \cdots X_nX_{n+1}\omega_1$ is a sentential form followed by "$\dashv$" for which all characters of $X_{n+1}\omega_1$ are terminals or "$\dashv$", and if this string has a handle (n, p) ending at position n, then all 1-sentential forms $X_1X_2 \cdots X_nX_{n+1}\omega$ with $X_{n+1}\omega$ as above must have the same handle (n, p). The definition has been phrased carefully to account for the possibility that the handle is the empty string, which if inserted between X_n and X_{n+1} is regarded as having right boundary n.

This definition of an LR(k) grammar coincides with the intuitive notion of translation from left to right looking k characters ahead. Assume at some stage of translation we have made all possible reductions to the left of X_n ; by looking at the next k characters $X_{n+1} \cdots X_{n+k}$, we want to know if a reduction on $X_{r+1} \cdots X_n$ is to be made, *regardless* of what follows X_{n+k}. In an LR(k) grammar we are able to decide without hesitation whether or not such a reduction should be made. If a reduction is called for, we perform it and repeat the process; if none should be made, we move one character to the right.

An LR(k) grammar is clearly unambiguous, since the definition implies every derivation tree must have the same handle, and by induction there is only one possible tree. It is interesting to point out furthermore that nearly every grammar which is known to be unambiguous is either an LR(k) grammar, or (dually) is a right-to-left translatable grammar, or is some grammar which is translated using "both ends toward the middle." Thus, *the* LR(k) *condition may be regarded as the most powerful general test for nonambiguity that is now available.*

When k is given, we will show in Section II that it is possible to decide if a grammar is LR(k) or not. The essential reason behind this that *the possible configurations of a tree below its handle may be represented by a regular (finite automaton) language.*

Several related ideas have appeared in previous literature. Lynch (1963) considered special cases of LR(1) grammars, which he showed are unambiguous. Paul (1962) gave a general method to construct left-to-right parsers for certain very simple LR(1) languages. Floyd (1964a) and Irons (1964) independently developed the notion of *bounded context* grammars, which have the property that one knows whether or not to reduce any sentential form $\alpha\theta\omega$ using the production $A \rightarrow \theta$ by examining only a finite number of characters immediately to the left and right of θ. Eickel (1964) later developed an algorithm which would construct a certain form of push-down parsing program from a bounded context grammar, and Earley (1964) independently developed a somewhat similar method which was applicable to a rather large number of LR(1) languages but had several important omissions. Floyd (1964a) also introduced the more general notion of a *bounded right context* grammar; in our terminology, this is an LR(k) grammar in which one knows whether or not $X_{r+1} \cdots X_n$ is the handle by examining only a given *finite* number of characters immediately to the left of X_{r+1}, as well as knowing $X_{n+1} \cdots X_{n+k}$. At that time it seemed plausible that a bounded right context grammar was the natural way to formalize the intuitive notion of a grammar by which one could translate from left to right without backing up or looking ahead by more than a given distance; but it was possible to show that Earley's construction provided a parsing method for some grammars which were *not* of bounded right context, although intuitively they should have been, and this led to the above definition of an LR(k) grammar (in which the *entire* string to the left of X_{r+1} is known).

It is natural to ask if we can in fact always parse the strings corresponding to an LR(k) grammar by going from left to right. Since there are an infinite number of strings $X_1 \cdots X_{n+k}$ which must be used to make a parsing decision, we might need infinite wisdom to be able to make this decision correctly; the definition of LR(k) merely says a correct decision *exists* for each of these infinitely many strings. But it will be shown in Section II that only a finite number of essential possibilities really exist.

Now we will present a few examples to illustrate these notions. Consider the following two grammars:

$$S \to aAc,\ A \to bAb,\ A \to b. \tag{6}$$

$$S \to aAc,\ A \to Abb,\ A \to b. \tag{7}$$

Both of these are unambiguous and they define the same language, $\{ab^{2n+1}c\}$. Grammar (6) is not LR(k) for any k, since given the partial string ab^m there is no information by which we can replace any b by A; parsing must wait until the "c" has been read. On the other hand grammar (7) is LR(0), in fact it is a bounded context language; the sentential forms are $\{aAb^{2n}c\}$ and $\{ab^{2n+1}c\}$, and to parse we must reduce a substring ab to aA, a substring Abb to A, and a substring aAc to S. This example shows that LR(k) is definitely a property of the *grammar*, not of the *language* alone. The distinction between grammar and language is extremely important when semantics is being considered as well as syntax.

The grammar

$$S \rightarrow aAd,\ S \rightarrow bAB,\ A \rightarrow cA,\ A \rightarrow c,\ B \rightarrow d \tag{8}$$

has the sentential forms $\{ac^nAd\} \cup \{ac^{n+1}d\} \cup \{bc^nAB\} \cup \{bc^nAd\} \cup \{bc^{n+1}B\} \cup \{bc^{n+1}d\}$. In the string $bc^{n+1}d$, d must be replaced by B, while in the string $ac^{n+1}d$, this replacement must not be made; so the decision depends on an unbounded number of characters to the left of d, and the grammar is not of bounded context (nor is it translatable from right to left). On the other hand this grammar is clearly LR(1) and in fact it is of bounded right context since the handle is immediately known by considering the character to its right and two characters to its left; when the character d is considered the sentential form will have been reduced to either aAd or bAd.

The grammar

$$S \rightarrow aA,\ S \rightarrow bB,\ A \rightarrow cA,\ A \rightarrow d,\ B \rightarrow cB,\ B \rightarrow d \tag{9}$$

is not of bounded right context, since the handle in both ac^nd and bc^nd is "d"; yet this grammar is certainly LR(0). A more interesting example is

$$S \rightarrow aAc,\ S \rightarrow b,\ A \rightarrow aSc,\ A \rightarrow b. \tag{10}$$

Here the terminal strings are $\{a^nbc^n\}$, and the b must be reduced to S or A according as n is even or odd. This is another LR(0) grammar which fails to be of bounded right context.

In Section III we will give further examples and will discuss the relevance of these concepts to the grammar for ALGOL 60. Section IV contains a proof that the existence of k, such that a given grammar is LR(k), is recursively undecidable.

Ginsburg and Greibach (1965) have defined the notion of a *deterministic language*; we show in Section V that these are precisely the languages for which *there exists* an LR(k) grammar, and thereby we obtain a number of interesting consequences.

II. ANALYSIS OF LR(k) GRAMMARS

Given a grammar $\mathcal{G}$ and an integer $k \geqq 0$, we will now give two ways to test whether $\mathcal{G}$ is LR(k) or not. We may assume as usual that $\mathcal{G}$ does not contain useless productions, i.e., for any A in I there are terminal strings α, β, γ such that $S \Rrightarrow \alpha A\gamma \Rightarrow \alpha\beta\gamma$.

The first method of testing is to construct another grammar $\mathfrak{F}$ which reflects all possible configurations of a handle and k characters to its right. The intermediate symbols of $\mathfrak{F}$ will be $[A; \alpha]$, where α is a k-letter string on $T \cup \{ \dashv \}$; and also $[p]$, where p is the number of production in $\mathcal{G}$. The terminal symbols of $\mathfrak{F}$ will be $I \cup T \cup \{ \dashv \}$.

For convenience we define $H_k(\sigma)$ to be the set of all k-letter strings β over $T \cup \{ \dashv \}$ such that $\sigma \Rrightarrow \beta\gamma$ with respect to $\mathcal{G}$ for some γ; this is the set of all possible initial strings of length k derivable from σ.

Let the pth production of $\mathcal{G}$ be

$$A_p \rightarrow X_{p1} \cdots X_{pn_p}, \ 1 \leqq p \leqq s, \ n_p \geqq 0. \tag{11}$$

We construct all productions of the following form:

$$[A_p ; \alpha] \rightarrow X_{p1} \cdots X_{p(j-1)}[X_{pj} ; \beta] \tag{12}$$

where $1 \leqq j \leqq n_p$, X_{pj} is intermediate, and α, β are k-letter strings over $T \cup \{ \dashv \}$ with β in $H_k(X_{p(j+1)} \cdots X_{pn_p}\alpha)$. Add also the productions

$$[A_p ; \alpha] \rightarrow X_{p1} \cdots X_{pn_p}\alpha[p] \tag{13}$$

It is now easy to see that with respect to $\mathfrak{F}$,

$$[S; \dashv^k] \Rightarrow X_1 \cdots X_n X_{n+1} \cdots X_{n+k}[p] \tag{14}$$

if and only if there exists a k-sentential form $X_1 \cdots X_n X_{n+1} \cdots X_{n+k} Y_1 \cdots Y_u$ with handle (n, p) and with $X_{n+1} \cdots Y_u$ not intermediates. Therefore by definition, $\mathcal{G}$ will be LR(k) if and only if $\mathfrak{F}$ satisfies the following property:

$$[S; \dashv^k] \Rightarrow \theta[p] \text{ and } [S; \dashv^k] \Rightarrow \theta\varphi[q] \text{ implies } \varphi = \epsilon \text{ and } p = q. \tag{15}$$

But $\mathfrak{F}$ is a *regular* grammar, and well-known methods exist for testing condition (15) in regular grammars. (Basically one first transforms $\mathfrak{F}$ so that all of its productions have the form $Q_i \rightarrow aQ_j$, and then if $Q_0 = [S; \dashv^k]$, one can systematically prepare a list of all pairs (i, j) such that there exists a string α for which $Q_0 \Rightarrow \alpha Q_i$ and $Q_0 \Rightarrow \alpha Q_j$.)

When $k = 2$, the grammar $\mathfrak{F}$ corresponding to (2) is

$$
\begin{array}{ll}
[S;\ \dashv\dashv] \rightarrow [A;\ \dashv\dashv] & [C;\ \dashv\dashv] \rightarrow [B; e\dashv] \\
[S;\ \dashv\dashv] \rightarrow A[D;\ \dashv\dashv] & [C;\ \dashv\dashv] \rightarrow B[E;\ \dashv\dashv] \\
[S;\ \dashv\dashv] \rightarrow AD\dashv\dashv[1] & [C;\ \dashv\dashv] \rightarrow BE\dashv\dashv[4] \\
[A;\ \dashv\dashv] \rightarrow a[C;\ \dashv\dashv] & [B;\ e\dashv] \rightarrow bcde\dashv[3] \\
[A;\ \dashv\dashv] \rightarrow aC\dashv\dashv[2] & [E;\ \dashv\dashv] \rightarrow e\dashv\dashv[6] \\
[D;\ \dashv\dashv] \rightarrow \dashv\dashv[5] &
\end{array}
\qquad (16)
$$

It is, of course, unnecessary to list productions which cannot be reached from $[S;\ \dashv\dashv]$. Condition (15) is immediate; one may see an intimate connection between (16) and the tree (3).

Our second method for testing the LR(k) condition is related to the first but it is perhaps more natural and at the same time it gives a method for parsing the grammar $\mathcal{G}$ if it is indeed LR(k). The parsing method is complicated by the appearance of ϵ in the grammar, when it becomes necessary to be very careful deciding when to insert an intermediate symbol A corresponding to the production $A \rightarrow \epsilon$. To treat this condition properly we will define $H_k'(\sigma)$ to be the same as $H_k(\sigma)$ except omitting all derivations that contain a step of the form

$$A\omega \rightarrow \omega,$$

i.e., when an intermediate as the *initial character* is replaced by ϵ. This means we are avoiding derivation trees whose handle is an empty string at the extreme left. For example, in the grammar

$$S \rightarrow BC\dashv\dashv\dashv,\ B \rightarrow Ce,\ B \rightarrow \epsilon,\ C \rightarrow D,\ C \rightarrow Dc,\ D \rightarrow \epsilon,\ D \rightarrow d$$

we would have

$$H_3(S) = \{\dashv\dashv\dashv, c\dashv\dashv, ce\dashv, cec, ced, d\dashv\dashv, dce,$$
$$de\dashv, dec, ded, e\dashv\dashv, ec\dashv, ed\dashv, edc\}$$

$$H_3'(S) = \{dce, de\dashv, dec, ded\}.$$

As before we assume the productions of $\mathcal{G}$ are written in the form (11). We will also change $\mathcal{G}$ by introducing a new intermediate S_0 and adding a "zeroth" production

$$S_0 \rightarrow S\ \dashv^k \qquad (16)$$

and regarding S_0 as the principal intermediate. The sentential forms are now identical to the k-sentential forms as defined above, and this is a decided convenience.

Our construction is based on the notion of a "state," which will be denoted by $[p, j; \alpha]$; here p is the number of a production, $0 \leqq j \leqq n_p$, and α is a k-letter string of terminals. Intuitively, we will be in state $[p, j; \alpha]$ if the partial parse so far has the form $\beta X_{p1} \cdots X_{pj}$, and if $\mathcal{G}$ contains a sentential form $\beta A_p \alpha \cdots$; that is, we have found j of the characters needed to complete the pth production, and α is a string which may legitimately follow the entire production if it is completed.

At any time during translation we will be in a *set* $\mathcal{S}$ of states. There are of course only a finite number of possible sets of states, although it is an enormous number. Hopefully there will not be many sets of states which can actually arise during translation. For each of these possible sets of states we will give a rule which explains what parsing step to perform and what new set of states to enter.

During the translation process we maintain a *stack*, denoted by

$$\mathcal{S}_0 X_1 \mathcal{S}_1 X_2 \mathcal{S}_2 \cdots X_n \mathcal{S}_n \mid Y_1 \cdots Y_k \omega. \tag{17}$$

The portion to the left of the vertical line consists alternately of state sets and characters; this represents the portion of a string which has already been translated (with the possible exception of the handle) and the state sets $\mathcal{S}_i$ we were in just after considering $X_1 \cdots X_i$. To the right of the vertical line appear the k terminal characters $Y_1 \cdots Y_k$ which may be used to govern the translation decision, followed by a string ω which has not yet been examined.

Initially we are in the state set $\mathcal{S}_0$ consisting of the single state $[0, 0; \dashv^k]$, the stack to the left of the vertical line in (17) contains only $\mathcal{S}_0$, and the string to be parsed (followed by $\dashv^k$) appears at the right. Inductively at a given stage of translation, assume the stack contents are given by (17) and that we are in state set $\mathcal{S} = \mathcal{S}_n$.

Step 1. Compute the "closure" $\mathcal{S}'$ of $\mathcal{S}$, which is defined recursively as the smallest set satisfying the following equation:

$$\begin{aligned} \mathcal{S}' = \mathcal{S} \cup \{[q, 0; \beta] \mid \text{there exists } [p, j; \alpha] \text{ in } \mathcal{S}', j < n_p, \\ X_{p(j+1)} = A_q, \text{ and } \beta \text{ in } H_k(X_{p(j+2)} \cdots X_{pn_p}\alpha)\}. \end{aligned} \tag{18}$$

(We thus have added to $\mathcal{S}$ all productions we might begin to work on, in addition to those we are already working on.)