

Deterministic Context-Sensitive Languages: Part I*

DANIEL A. WALTERS

RCA Laboratories, Princeton, New Jersey 08540

A context-sensitive grammar G is said to be $CS(k)$ iff a particular kind of table-driven parser, $\mathcal{T}_k(G)$, exists. Corresponding to each $\mathcal{T}_k(G)$, we define a class of parsers $\bar{\mathcal{T}}_k(G)$. $\mathcal{T}_k(G)$ is itself a $\bar{\mathcal{T}}_k(G)$. The main results are:

1. Any processor $\bar{\mathcal{T}}_k(G)$ for a $CS(k)$ grammar G accepts exactly the sentences of G .
2. The set of languages generable by $CS(k)$ grammars is exactly the set of languages accepted by deterministic linear-bounded automata (DLBA's).
3. (a) It is undecidable whether there exists any $k \geq 0$ such that an arbitrary CSG is $CS(k)$.
(b) For every fixed $k \geq 0$, there is no algorithm that will decide if G is $CS(k)$ and also construct $\bar{\mathcal{T}}_k(G)$ if it exists.
4. For any DLBA M , algorithms are given to (i) construct a $CS(k)$ grammar G_M that generates the language accepted by M , and (ii) construct a processor $\bar{\mathcal{T}}_1(G_M)$.
5. $CS(k)$ grammars are unambiguous.
6. The sentences of a $CS(k)$ grammar can be parsed in a time proportional to the length of their derivations.

1. INTRODUCTION

Efficient processing of programming languages has great practical importance and has consequently received much theoretical attention. A large part of this attention has been devoted to finding models of programming languages. These models have been of two complementary types: generative and recognitive. The most important generative model has been phrase-structure grammars, especially context-free grammars. The recognitive models have been various restricted Turing machines. These two approaches to modeling have been related by showing, for many different $\langle$ grammar, automaton $\rangle$

* A preliminary version of this paper was presented at the Tenth Annual Symposium on Switching and Automata Theory, Waterloo, Ontario, October 15-17, 1969.

pairs, that a language is generable by a particular kind of grammar if and only if it is recognizable by a particular kind of automaton.

The context-free grammars (CFG's) have received the most study. The class of languages generated by CFG's has been shown to be equal to the class of languages recognizable by non-deterministic pushdown automata. The class of languages recognizable by deterministic pushdown automata (DPDA's) has also been studied, since it provides an elegant model of many simple languages. This class of automata was related to grammars in (Knuth, 1965). He defined the class of $LR(k)$ grammars, for each $k \geq 0$, and showed that a language is generable by an $LR(k)$ context-free grammar for some k if and only if it is recognizable by a DPDA.

This work extends the methods of Knuth to context-sensitive grammars (CSG's). The class of context-sensitive grammars properly includes the class of context-free grammars, but has received much less attention, partly because of the lack of an efficient general scheme for processing these grammars. A class of automata that is equivalent to CSG's is the class of non-deterministic linear-bounded automata (NLBA's). The class of languages recognizable by deterministic linear-bounded automata (DLBA's) has not been related to grammars,¹ and this will be done in Part II. In Part I, we define a class of grammars—called $CS(k)$ grammars—that are equivalent to the DLBA's and present some of their properties.

The rest of this section defines the basic concepts and notation used later. Section 2 defines $CS(k)$ grammars and their processors, and establishes some of their properties. These two sections comprise Part I. Section 3 gives a non-standard formulation of a deterministic linear-bounded automaton (DLBA), and proves the equivalence of this formulation to the standard one. Section 4 shows the equivalence of $CS(k)$ grammars and DLBA's. Section 5 discusses the time bounds for $CS(k)$ processors, and Section 6 describes some problems left open in this work.

Definitions

A *context-sensitive grammar* (CSG) is a quadruple $\{V_T, V_N, S, \mathcal{P}\}$, where V_T is a finite set of *terminal symbols*, V_N is a finite set of *nonterminal symbols*, $V_T \cap V_N = \emptyset$ (the empty set), $V = V_T \cup V_N$ is called the *vocabulary*, $S \in V_N$ is called the *distinguished symbol*, and $\mathcal{P}$ is a finite set of *rules*. Let the members of $\mathcal{P}$ be numbered from 1 to π . The p -th rule is denoted

¹ It is not known whether the class of languages accepted by DLBA's is equal to the class accepted by NLBA's.

$Y_{p1}Y_{p2}\dots Y_{pm_p} \rightarrow X_{p1}X_{p2}\dots X_{pn_p}$, where $1 \leq m_p \leq n_p$, $Y_{pi}, X_{pi} \in V$ and² $Y_{p1}\dots Y_{pm_p} \notin V_T^+$. If $m_p \geq 2$, rule p is a *context-sensitive rule*; if $m_p = 1$, rule p is a *context-free rule*. $Y_{p1}\dots Y_{pm_p}$ is the *subject* of rule p , and $X_{p1}\dots X_{pn_p}$ is its *RHS* (right-hand side).

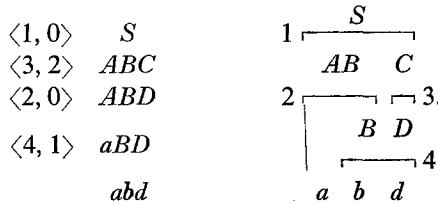
A *derivation* in a CSG is a sequence $\alpha_1, \alpha_2, \dots, \alpha_{m+1}$, $m \geq 0$, of strings such that for each i , $1 \leq i \leq m$, there are strings $\beta_i, \gamma_i, \delta_i, \zeta_i$ such that $\alpha_i = \beta_i\gamma_i\delta_i$, $\alpha_{i+1} = \beta_i\zeta_i\delta_i$, and $\gamma_i \rightarrow \zeta_i \in \mathcal{P}$. Associated with each of $\alpha_1, \dots, \alpha_m$ there must be a pair $\langle p, r \rangle$ denoting that $\gamma_i \rightarrow \zeta_i$ is the p -th rule of $\mathcal{P}$ and the first symbol of γ_i is the r -th symbol of α_i . Each α_i is a *line* of the derivation, and the process of applying a rule to one line to produce the next line is a *step* of the derivation. The sequence $\alpha_1, \dots, \alpha_{m+1}$ is said to be a *derivation of α_{m+1} from α_1* , and α_{m+1} is said to be *derivable from α_1* .

A string $\omega \in V_T^*$ is a *sentence* if it is derivable from S . The set of all sentences generable by a CSG forms the (context-sensitive) *language* defined by G , and is written $L(G)$.

A *context-free grammar* (CFG) is a CSG all of whose rules are context-free or are of the form $A \rightarrow \lambda$ for $A \in V_N$. Such a rule is called a λ -rule. If a CFG contains no λ -rules, it is clearly also a CSG.

We will define a *parse* as a two-dimensional description of a set of derivations. A parse of β as α is a bracketed diagram showing how β is derived from α . Such a parse is obtained from any derivation of β from α by writing down β , then bracketing the RHS of the last rule used in the derivation, writing the subject of the rule above the bracket, and associating the rule number with the bracket. The string resulting from replacing the bracketed symbols by those above the bracket is the penultimate line of the derivation. If this bracketing is continued until all the steps in the derivation have been utilized, the result is a *parse of β as α* . The set of derivations that would yield the same parse under this construction is the set described by the parse.

For example, in the grammar³ (1) $S \rightarrow ABC$, (2) $AB \rightarrow aB$, (3) $C \rightarrow D$, (4) $BD \rightarrow bd$, a derivation and the corresponding parse are



² V_T^+ is the set of all nonnull strings over V_T . The null string is denoted by λ , and $V_T^* = V_T^+ \cup \{\lambda\}$. V^k denotes all members of V^* of length k . For a single symbol x , $x^k = \{x\}^k$.

³ Upper- and lower-case letters are nonterminal and terminal symbols, respectively.

Corresponding to every parse is a unique *rightmost derivation*, which can be constructed from the parse as follows: Write down S . Find the rightmost bracket that has no brackets above it, and delete it. The next line of the derivation is determined by $\langle p, r \rangle$, where p is the rule number labeling the deleted bracket and $r + 1$ is the position of the leftmost bracketed symbol. Repeat this process until all brackets have been deleted. Alternatively, the rightmost derivation is the one that allows the parse to be constructed as in the preceding paragraph without ever drawing a bracket wholly to the left of the previous one drawn. The derivation in the example above is the rightmost one. If rule 2 had been used before rule 3, it would not have been rightmost.

A sentence is *ambiguous* if there are two rightmost derivations of it. A grammar is *ambiguous* if it has any ambiguous sentences; otherwise, it is *unambiguous*.

We describe derivations by writing $\alpha \Rightarrow \beta$ if β is derived from α (in zero or more steps), $\alpha \rightarrow \beta$ if β is derived from α in exactly one step, and $\alpha \xrightarrow{p} \beta$ if this one step used rule p . For any strings $\alpha_1, \alpha_2, \alpha_3, \beta$ if we write $\alpha_1 \underline{\alpha_2} \alpha_3 \Rightarrow \beta$ or $\alpha_1 \underline{\alpha_2} \alpha_3 \rightarrow \beta$, we mean that no symbols in α_1 or α_3 were replaced in the derivation. That is, we mean that β is of the form $\alpha_1 \gamma \alpha_3$ and $\alpha_2 \Rightarrow \gamma$ or $\alpha_2 \rightarrow \gamma$. The strings α_1 and α_3 , and their substrings, are said to be *carried down* from the line $\alpha_1 \alpha_2 \alpha_3$ to the line β .

2. CS(k) GRAMMARS

Knuth (1965) has defined the LR(k) grammars and given an algorithm that constructs a processor for a grammar if the grammar is LR(k) (for a fixed k). The definition of the algorithm and the processors it constructs has three parts: the description of an automaton to process strings, the explication of the notion of “state” for the automaton, and a construction method for the parsing table that controls the automaton.

We will proceed similarly for CS(k) grammars. We will first describe a class of processors for context-sensitive grammars, and then give an iterative specification of a parsing table $\mathcal{P}_k(G)$ that controls a processor of this class for a CS(k) grammar G . This specification is a generalization of Knuth’s construction algorithm, and reduces to it if G is context-free. This iterative specification will provide our basic definition.

First, we define a grammar $G' = (V_N', V_T', S', \mathcal{P}')$ corresponding to a context-sensitive grammar⁴ $G = (V_N, V_T, S, \mathcal{P})$, as follows:

$$V_N' = V_N \cup \{S'\}, V_T' = V_T \cup \{\#\}, S', \# \notin V, \mathcal{P}' = \mathcal{P} \cup \{S' \rightarrow S\#^k\}.$$

⁴ If G is context-free, $\mathcal{P}$ may contain Λ -rules.

Let the rules of $\mathcal{P}$ be numbered from 1 through π in both $\mathcal{P}$ and $\mathcal{P}'$, and add the rule $S' \rightarrow S\#^k$ to $\mathcal{P}'$ as rule 0. A $CS(k)$ or $LR(k)$ processor examines input strings from $V_T^*\#^k$ to determine if they belong to $L(G')$; clearly, $\omega \in L(G)$ if and only if $\omega\#^k \in L(G')$.

The $LR(k)$ processor is an automaton with a two-track stack (pushdown list) for storage, a one-way input tape, and a finite control mechanism directed by the parsing table $\mathcal{T}_k(G)$. The storage configuration of the automaton will be represented by

$$\begin{array}{c} \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \\ \bullet x_1 \dots x_n \end{array} \Bigg| \beta,$$

where the contents of the stack is depicted to the left of the vertical line, and the unread portion of the input is to the right. The $\mathcal{S}_i$ are called *states*, the x_i are members of V , $\mathcal{S}_0$ is the unique *start state*, $\mathcal{S}_n$ and x_n ($n \geq 0$) are the top entry on the stack, and $\beta \in V_T^*\#^k$. The automaton starts in the configuration

$\begin{array}{c} \mathcal{S}_0 \\ \bullet \end{array} \Bigg| \omega\#^k$, where $\omega \in V_T^*$ is the input string to be recognized. If the automaton reaches the configuration, $\begin{array}{c} \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \\ \bullet x_1 \dots x_n \end{array} \Bigg| \omega_2\#^k$, where $\omega = \omega_1\omega_2$, then

(i) the $LR(k)$ processor has “reduced” ω_1 to $x_1\dots x_n$ —i.e., it has reconstructed a rightmost derivation of ω_1 from $x_1\dots x_n$,⁵ and (ii) if ω_3 is the k -length prefix of $\omega_2\#^k$, then there exists some ω_4 such that $S' \Rightarrow \omega_1\omega_3\omega_4$. In other words, if the automaton reaches the configuration shown, the input string can be a sentence only if $x_1\dots x_n\omega_2$ is a line of its rightmost derivation.⁶

We will represent the storage configuration of the $CS(k)$ processor in the same way, but with a slightly different meaning. Instead of a stack and a one-way input tape, the $CS(k)$ processor will have two pushdown lists, L and R. The L pushdown list has two tracks, and is completely analogous to Knuth’s stack.⁷ The R pushdown list has one track. The storage configuration of the $CS(k)$ processor is represented as $\begin{array}{c} \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \\ \bullet x_1 \dots x_n \end{array} \Bigg| \beta$, where the contents of the L pushdown list is depicted to the left of the vertical line and has the same interpretation as before, and the contents of the R pushdown list is depicted to the right. The top symbol on R is the first symbol of β , and β is a member of $V^*\#^k$.

As for the $LR(k)$ case, the $CS(k)$ processor starts in the configuration,

⁵ This is also, of course, a rightmost derivation of $\omega = \omega_1\omega_2$ from $x_1 \dots x_n\omega_2$.

⁶ $LR(k)$ grammars are unambiguous.

⁷ We will use the word *stack* to refer to the L pushdown list when we are discussing $LR(k)$ and $CS(k)$ processors simultaneously.

$\mathcal{S}_0 \mid \omega \#^k$, where $\omega \in V_T^*$ is the input string. If the automaton reaches the configuration $\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \mid \beta \#^k$, where $\beta \in V^*$, then it has reconstructed a rightmost derivation of ω from $x_1 \dots x_n \beta$. That is, ω can be a sentence only if $x_1 \dots x_n \beta$ is a line of its rightmost derivation.

In both the $\text{LR}(k)$ and $\text{CS}(k)$ processors, a *state* $\mathcal{S}$ is a set of *partial states*⁸ of the form $[p, j, \alpha]$, where p is a rule number, $0 \leq p \leq \pi$; j is an integer $1 \leq j \leq n_p$; and $\alpha \in V^i \#^{k-i}$ with $0 \leq i \leq k$ for $\text{CS}(k)$ and $\alpha \in V_T^i \#^{k-i}$ for $\text{LR}(k)$. (If $p = 0$, j may also be 0; then $\alpha = \Lambda$ for all k .)

Suppose that the processor ($\text{LR}(k)$ or $\text{CS}(k)$) started with the input string $\omega \#^k$ and reached the configuration $\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \mid \beta \#^k$. The automaton will be said to be *in state* $\mathcal{S}_n$.

The significance of a partial state can be explained as follows. Suppose that there exists some $\omega \in V_T^*$ such that the $\text{LR}(k)$ processor reaches the configuration, $\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \mid \beta \zeta$, after starting with $\omega \#^k$, where $\beta \zeta \in V_T^* \#^k$ and $|\beta| = k$. The following three statements are equivalent:

- (1) The partial state $[p, j, \alpha]$ is a member of $\mathcal{S}_n$.
- (2) The processor has just found the first j symbols of the p -th rule, and α can follow the p -th rule if it is completed; that is, $x_1 \dots x_n = \gamma X_{p1} \dots X_{pj}$ for some γ , and $S' \Rightarrow \gamma Y_{p1} \alpha \delta$ for some δ .
- (3) There exist $\gamma, \delta \in V_T^* \#^*$ such that there is a rightmost derivation of the form,

$$\begin{aligned} S' &\Rightarrow x_1 \dots x_{n-j} Y_{p1} \alpha \gamma \\ &\xrightarrow{p} x_1 \dots x_{n-j} x_{n-j+1} \dots x_n \underline{X_{p,j+1} \dots X_{pn_p} \alpha \gamma} = x_1 \dots x_{n-j} X_{p1} \dots X_{pn_p} \alpha \gamma \\ &\Rightarrow x_1 \dots x_n \beta \delta. \end{aligned}$$

Note that since this is a rightmost derivation in a CFG, $\alpha \gamma \in V_T^* \#^k$; hence, $\alpha \gamma$ is not involved in the derivation of $x_{n-j+1} \dots x_n \beta \delta$ from $X_{p1} \dots X_{pn_p} \alpha \gamma$.

For the $\text{CS}(k)$ case, we will see later that a generalization of this last statement will characterize the partial states that belong to $\mathcal{S}_n$.

The *parsing table* $\mathcal{T}_k(G)$ that directs the $\text{LR}(k)$ or $\text{CS}(k)$ processor is composed of a finite number of rows, each associated with a state of the processor. We will describe the format and meaning of $\mathcal{T}_k(G)$ for a $\text{CS}(k)$

⁸ Knuth uses *state* and *state set*, respectively, for our *partial state* and *state*.

processor; the $LR(k)$ processor is just a special case, with $m_p = 1$ for all p . A typical row of $\mathcal{T}_k(G)$, corresponding to a state $\mathcal{S}$, has the form,

State Name	Lookahead	Action	Stack Symbols	Goto States
$\mathcal{S}$	α_1	Shift	x_1	$\mathcal{S}_1$
	α_2	Shift		$\vdots$
	α_3	Reduce p	x_m	$\mathcal{S}_m$
	$\vdots$			
	α_ℓ	Reduce q		

$\mathcal{S}$ and $\mathcal{S}_1, \dots, \mathcal{S}_m$ ($m \geq 0$) denote states; $\alpha_1, \dots, \alpha_\ell$ ($\ell \geq 1$) $\in V^i \#^{k-i}$ for $0 \leq i \leq k$ are all different; $x_1, \dots, x_m \in V$ are all different. For a state $\mathcal{S}$, $L(\mathcal{S})$ denotes the set of strings in the *lookahead* column, and for each $\alpha \in L(\mathcal{S})$, $A(\mathcal{S}, \alpha)$ is the action associated with α . This action will be one of *shift*, *reduce p* ($1 \leq p \leq \pi$), or *accept*. The set of *stack symbols* will be denoted $S(\mathcal{S})$, and for each $x \in S(\mathcal{S})$, its associated *goto state* is $G(\mathcal{S}, x)$.

In order to explain how $\mathcal{T}_k(G)$ controls the $CS(k)$ processor, suppose the

processor has reached the configuration, $\begin{matrix} \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \\ \bullet x_1 \dots x_n \end{matrix} \mid x\eta\beta$, where $n \geq 0$,

$x\eta\beta \in V^* \#^k$, $|x\eta| = k$, and $x\eta$ is called the *lookahead string*. The row of $\mathcal{T}_k(G)$ named by $\mathcal{S}_n$ determines the next move of the processor, as follows: First, $x\eta$ is compared to the strings α_i in the lookahead column. If no α_i is equal to $x\eta$, the processor *rejects* the input string. Otherwise, the action associated with $x\eta$, $A(\mathcal{S}_n, x\eta)$, is performed as follows:

(i) $A(\mathcal{S}_n, x\eta) = \text{accept}$. We will see later that the configuration must

be $\begin{matrix} \mathcal{S}_0 \mathcal{S}_F \\ \bullet S \end{matrix} \mid \#^k$, where $\mathcal{S}_F$ is a unique accepting state. The processor stops, and

accepts the original input string (recognizes it as a member of $L(G)$).

(ii) $A(\mathcal{S}_n, x\eta) = \text{shift}$. The symbol x is popped off the R pushdown list and pushed onto the L pushdown list (on the lower track). The symbol x will

appear in the STACK column of $\mathcal{S}_n$; the state associated with it, $G(\mathcal{S}_n, x)$, is put on the top track of L . The processor is then in the configuration, $\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n G(\mathcal{S}_n, x) \mid \eta\beta$, and processing continues using the row of $\mathcal{T}_k(G)$ associated with $G(\mathcal{S}_n, x)$.

(iii) $A(\mathcal{S}_n, x) = \text{reduce } p$, for $1 \leq p \leq \pi$. The topmost n_p entries on the lower track of L are equal to $X_{p1} \dots X_{pn_p}$. L is popped n_p times, and the subject of rule p is pushed onto the R stack. The processor is then in the configuration, $\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_{n-n_p} \mid Y_{p1} \dots Y_{pm_p} x \eta \beta$. Processing continues using the row of $\mathcal{T}_k(G)$ associated with $\mathcal{S}_{n-n_p}$, and using the first k symbols of $Y_{p1} \dots Y_{pm_p} x \eta$ as the lookahead string. (The action required by $\mathcal{S}_{n-n_p}$ for this lookahead string will always be *shift*.⁹)

Knuth's algorithm determines whether, for a given k and G , $\mathcal{T}_k(G)$ exists, and constructs it if possible. It does this by specifying the entries of the row of $\mathcal{T}_k(G)$ corresponding to a state $\mathcal{S}$ as a function of the partial states composing $\mathcal{S}$. If this specification leads to conflicting actions for the same lookahead string, then $\mathcal{T}_k(G)$ does not exist. The algorithm will attempt to construct the row of $\mathcal{T}_k(G)$ corresponding to state $\mathcal{S}$ if either $\mathcal{S}$ is the start state $\mathcal{S}_0$ or $\mathcal{S}$ already appears in the GOTO column for some state.

We define $\text{CS}(k)$ grammars by a similar method. Let G be a context-sensitive grammar and $k \geq 0$. G is $\text{CS}(k)$ iff the table $\mathcal{T}_k(G)$, as defined below, exists. The following statement is true for any context-free grammar G and any k : G is $\text{LR}(k)$ iff it is $\text{CS}(k)$, and the tables $\mathcal{T}_k(G)$ produced by Knuth's definition and by the one below are identical if they exist.¹⁰

Definition of $\mathcal{T}_k(G)$

Step 1. Define the start state $\mathcal{S}_0 \triangleq \{[0, 0, A]\}$.

Steps 2-7 specify the row of $\mathcal{T}_k(G)$ corresponding to any state $\mathcal{S}$.

Step 2. When the $\text{CS}(k)$ processor is in a state containing $[p, j, \theta]$ with $j < n_p$, the processor must be prepared to encounter as a prefix of the R pushdown list, a string derived from $X_{p,j+1} \dots X_{pn_p} \theta$. If this derivation applies

⁹ For the case where $m_p = 1$, the subject of rule p will be pushed onto R , and then *shifted* onto L under the direction of $\mathcal{S}_{n-n_p}$ in the next step. This is clearly equivalent to the steps called for by Knuth's formulation: the subject of rule p is immediately pushed onto the bottom track of the stack, and $G(\mathcal{S}_{n-n_p}, Y_{p1})$ is pushed onto the top track.

¹⁰ If G contains any A -rules, this statement applies to the grammar obtained by removing the A -rules from G in the usual way.

rule q at the left, then the processor must also be prepared to encounter $X_{q1}...X_{qn_q}$. $\mathcal{S}''$ is defined as the set of all partial states $[q, 0, \theta']$ that need be considered in this way:

$$\begin{aligned}
 \mathcal{S}'' &\triangleq \{[q, 0, \theta'] \mid \theta' \in V^* \#^* \& \mid \theta' \mid = k \\
 &\quad \& (\exists [p, j, \theta] \in \mathcal{S})(\exists \alpha \in V^*)(\exists \gamma, \delta \in V^* \#^*)(\exists \omega \in L(G)) \\
 &\quad (S' \Rightarrow \alpha Y_{p1} \dots Y_{pn_p} \theta \gamma \\
 &\quad \xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p,j+1} \dots X_{pn_p} \theta \gamma \\
 &\quad \Rightarrow \alpha X_{p1} \dots X_{pj} Y_{q1} \dots Y_{qn_q} \theta' \delta \\
 &\quad \xrightarrow{q} \alpha X_{p1} \dots X_{pj} X_{q1} \dots X_{qn_q} \theta' \delta \\
 &\quad \Rightarrow \omega \#^k \text{ is a rightmost derivation})\}.
 \end{aligned}$$

Step 3. Define $\mathcal{S}' \triangleq \mathcal{S} \cup \mathcal{S}''$.

Step 4. Define the set of lookahead strings that have the action *shift*:

$$\begin{aligned}
 L_0(\mathcal{S}) &\triangleq \{x\eta \mid \mid x\eta \mid = k \& (\exists [p, j, \theta] \in \mathcal{S})(\exists \alpha \in V^*)(\exists \gamma, \delta \in V^* \#^*)(\exists \omega \in L(G)) \\
 &\quad (S' \Rightarrow \alpha Y_{p1} \dots Y_{pn_p} \theta \gamma \\
 &\quad \xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p,j+1} \dots X_{pn_p} \theta \gamma \\
 &\quad \Rightarrow \alpha X_{p1} \dots X_{pj} x\eta \delta \\
 &\quad \Rightarrow \omega \#^k \text{ is a rightmost derivation with the first step, if any, in} \\
 &\quad \alpha X_{p1} \dots X_{pj} x\eta \delta \Rightarrow \omega \#^k \text{ not wholly in } \eta \delta)\}.
 \end{aligned}$$

This says that $x\eta \in L_0(\mathcal{S})$ due to $[p, j, \theta] \in \mathcal{S}$ if there is some rightmost derivation of a sentence that uses rule p , and, when the processor is reconstructing this derivation and has already shifted $X_{p1}...X_{pj}$ onto the L pushdown list, $x\eta$ are the next k symbols after $X_{p1}...X_{pj}$. The requirement that $\alpha X_{p1}...X_{pn_p} \theta \gamma \Rightarrow \alpha X_{p1}...X_{pj} x\eta \delta$ not involve any symbols in $\alpha X_{p1}...X_{pj}$ reflects the fact that $X_{p1}...X_{pj}$ are already on the L pushdown list, and hence no reduction can be performed within them before shifting x . The requirement that the first step in $\alpha X_{p1}...X_{pj} x\eta \delta \Rightarrow \omega \#^k$ involve a symbol of $\alpha X_{p1}...X_{pj} x$ reflects the fact that we are looking for a rightmost derivation, and hence any steps wholly to the right of x would have been performed

before reaching the line of the derivation associated with the present configuration.

Step 5. For each p , $1 \leq p \leq \pi$, define the set of lookahead strings that have the action *reduce* p :

$$L_p(\mathcal{S}) \triangleq \{\theta \mid \exists [p, n_p, \theta] \in \mathcal{S}'\}.$$

That is, $A(\mathcal{S}, \theta) = \text{reduce } p$ if the topmost n_p symbols on L are $X_{p1} \dots X_{pn_p}$ and the lookahead string can follow $Y_{p1} \dots Y_{pn_p}$ in a derivation of a sentence.¹¹

Step 6. If $A(\mathcal{S}, \theta)$ is unique for all $\theta \in V^i \#^{k-i}$ —i.e., if the $L_i(\mathcal{S})$ are pairwise disjoint for $0 \leq p \leq \pi$ —, then continue. Otherwise, G is not CS(k).

Step 7. Define the set of stack symbols and the goto states by

$$S(\mathcal{S}) \triangleq \{x \mid (\exists [p, j, \theta] \in \mathcal{S}')(x = X_{p,j+1})\}$$

and

$$G(\mathcal{S}, x) \triangleq \{[p, j+1, \theta] \mid [p, j, \theta] \in \mathcal{S}' \ \& \ x = X_{p,j+1}\}.$$

For $k > 0$, $S(\mathcal{S})$ can also be defined by

$$S(\mathcal{S}) = \{x \mid (\exists \eta)(x\eta \in L_0(\mathcal{S}))\}.$$

Steps 2–7 are repeated for each state appearing in the GOTO column of $\mathcal{S}$ whose row has not yet been computed.

Step 8. Let $\mathcal{S}_F$ denote the unique state¹² containing $[0, 1, A]$. Set $A(\mathcal{S}_F, \#^k) = \text{accept}$.

EXAMPLE. Let $G = (\{S, A, B, C, D, E, F\}, \{a, b, d, e, f\}, S, \mathcal{P})$, where $\mathcal{P}$ is

$$\begin{array}{lll} 0 \ S' \rightarrow S\# & 3 \ S \rightarrow Ed & 5 \ DBC \rightarrow dbe \\ 1 \ S \rightarrow ABC & 4 \ A \rightarrow D & 6 \ DBF \rightarrow abd \\ 2 \ S \rightarrow fABF & & 7 \ E \rightarrow f. \end{array}$$

¹¹ In the definition of $L_p(\mathcal{S})$, we could use $\mathcal{S}$ instead of $\mathcal{S}'$, since $n_p > 0$ for all p , and all members of $\mathcal{S}' - \mathcal{S}$ are of the form $[g, 0, \theta]$. Using $\mathcal{S}'$ makes the definition formally correct for the LR(k) case, when n_p may be zero.

¹² $\mathcal{S}_F = G(\mathcal{S}_0, S)$ unless $L(G) = \emptyset$, in which case $L_i(\mathcal{S}_0) = \emptyset$ for $0 \leq i \leq \pi$ and $\mathcal{S}_0$ is the only state.

G is CS(1). There are three rightmost derivations:

(1)	(2)	(3)
S	S	S
1 $\underline{ABC}$	2 $f\underline{ABF}$	3 $\underline{Ed}$
4 $\underline{DBC}$	4 $f\underline{DBF}$	7 fd
5 dbe	6 $fabd$	

The parsing table $\mathcal{T}_1(G)$ is given by Table I.

TABLE I

Parsing Table $\mathcal{T}_1(G)$						
State	Members of $\mathcal{S}$	Members of $\mathcal{S}''$	Lookahead	Action	Stack	Goto
$\mathcal{S}_0$	$[0, 0, A]$		S	Shift	S	$\mathcal{S}_F = [0, 1, A]$
		$[1, 0, \#]$	A	Shift	A	$\mathcal{S}_1 = [1, 1, \#]$
		$[2, 0, \#]$	f	Shift	f	$\mathcal{S}_2 = [2, 1, \#], [7, 1, d]$
		$[3, 0, \#]$	E	Shift	E	$\mathcal{S}_3 = [3, 1, \#]$
		$[4, 0, B]$	D	Shift	D	$\mathcal{S}_4 = [4, 1, B]$
		$[5, 0, \#]$	d	Shift	d	$\mathcal{S}_5 = [5, 1, \#]$
		$[7, 0, d]$				
$\mathcal{S}_F$	$[0, 1, A]$		$\#$	Accept		
$\mathcal{S}_1$	$[1, 1, \#]$		B	Shift	B	$\mathcal{S}_6 = [1, 2, \#]$
$\mathcal{S}_2$	$[2, 1, \#]$		A	Shift	A	$\mathcal{S}_7 = [2, 2, \#]$
	$[7, 1, d]$		d	Reduce 7		
		$[4, 0, B]$	D	Shift	D	$\mathcal{S}_4 = [4, 1, B]$
		$[6, 0, \#]$	a	Shift	a	$\mathcal{S}_8 = [6, 1, \#]$
$\mathcal{S}_3$	$[3, 1, \#]$		d	Shift	d	$\mathcal{S}_9 = [3, 2, \#]$
$\mathcal{S}_4$	$[4, 1, B]$		B	Reduce 4		
$\mathcal{S}_5$	$[5, 1, \#]$		b	Shift	b	$\mathcal{S}_{10} = [5, 2, \#]$
$\mathcal{S}_6$	$[1, 2, \#]$		C	Shift	C	$\mathcal{S}_{11} = [1, 3, \#]$
$\mathcal{S}_7$	$[2, 2, \#]$		B	Shift	B	$\mathcal{S}_{12} = [2, 3, \#]$
$\mathcal{S}_8$	$[6, 1, \#]$		b	Shift	b	$\mathcal{S}_{13} = [6, 2, \#]$
$\mathcal{S}_9$	$[3, 2, \#]$		$\#$	Reduce 3		
$\mathcal{S}_{10}$	$[5, 2, \#]$		e	Shift	e	$\mathcal{S}_{14} = [5, 3, \#]$
$\mathcal{S}_{11}$	$[1, 3, \#]$		$\#$	Reduce 1		
$\mathcal{S}_{12}$	$[2, 3, \#]$		F	Shift	F	$\mathcal{S}_{15} = [2, 4, \#]$
$\mathcal{S}_{13}$	$[6, 2, \#]$		d	Shift	d	$\mathcal{S}_{16} = [6, 3, \#]$
$\mathcal{S}_{14}$	$[5, 3, \#]$		$\#$	Reduce 5		
$\mathcal{S}_{15}$	$[2, 4, \#]$		$\#$	Reduce 2		
$\mathcal{S}_{16}$	$[6, 3, \#]$		$\#$	Reduce 6		

This example shows the inadequacy of some alternative possible definitions of $\mathcal{S}''$ and $L_0(\mathcal{S})$. One of these alternatives, which is more analogous to Knuth's definitions than the ones we have adopted, is

$$\begin{aligned}
 S'' &= \{[g, 0, \theta'] \mid (\exists [p, j, \theta] \in \mathcal{S}') \\
 &\quad [X_{p,j+1} = Y_{q1} \ \& \ (\exists \alpha \in V^*)(\exists \gamma, \delta \in V^* \#^*)(\exists \omega \in L(G)) \\
 &\quad (S' \Rightarrow \alpha Y_{p1} \dots Y_{pm_p} \theta \gamma \\
 &\quad \xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p,j+1} X_{p,j+2} \dots X_{pn_p} \theta \gamma \\
 &\quad \Rightarrow \alpha X_{p1} \dots X_{pj} X_{p,j+1} Y_{q2} \dots Y_{qm_q} \theta' \delta \\
 &\quad \xrightarrow{q} \alpha X_{p1} \dots X_{pj} X_{q1} \dots X_{qn_q} \theta' \delta \\
 &\quad \Rightarrow \omega \#^k \text{ is a rightmost derivation})]\}.
 \end{aligned}$$

$$\begin{aligned}
 L_0(\mathcal{S}) &= \{x\eta \mid |x\eta| = k \ \& \ (\exists [p, j, \theta] \in \mathcal{S}') \\
 &\quad [x = X_{p,j+1} \ \& \ (\exists \alpha \in V^*)(\exists \gamma, \delta \in V^* \#^*)(\exists \omega \in L(G)) \\
 &\quad (S' \Rightarrow \alpha Y_{p1} \dots Y_{pm_p} \theta \gamma \\
 &\quad \xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p,j+1} X_{p,j+2} \dots X_{pn_p} \theta \gamma \\
 &\quad \Rightarrow \alpha X_{p1} \dots X_{pj} X_{p,j+1} \eta \delta \\
 &\quad \Rightarrow \omega \#^k \text{ is a rightmost derivation with the first step} \\
 &\quad \text{if any, in } \alpha X_{p1} \dots X_{pj} x \eta \delta \Rightarrow \omega \#^k \text{ not wholly} \\
 &\quad \text{within } \eta \delta)]\}.
 \end{aligned}$$

Using these definitions, a member $[p, j, \theta]$ of $\mathcal{S}'$ generates as additional members of $\mathcal{S}''$ those $[q, 0, \theta']$ that have $X_{p,j+1} = Y_{q1}$ such that $X_{p,j+1}$ was carried down from the line resulting from rule p to the line in which rule q was used. Members of $L_0(\mathcal{S})$ are generated from each member of $\mathcal{S}'$, and due to $[p, j, \theta] \in \mathcal{S}'$ we get only lookahead strings beginning with $X_{p,j+1}$.

If we were to apply these definitions to G , we would have $[6, 0, \#] \in \mathcal{S}''_0$ as follows: $[1, 0, \#] \in \mathcal{S}''_0$ due to $[0, 0, A] \in \mathcal{S}_0$ via derivation (1); $[4, 0, B] \in \mathcal{S}''_0$ due to $[1, 0, \#] \in \mathcal{S}''_0$ via derivation (1); $[6, 0, \#] \in \mathcal{S}''_0$ due to $[4, 0, B] \in \mathcal{S}''_0$ via derivation (2). Due to $[6, 0, \#] \in \mathcal{S}''_0$, we would have $a \in L_0(\mathcal{S}_0)$, $A(\mathcal{S}_0, a) = \text{shift}$, and $G(\mathcal{S}_0, a) = \{[6, 1, \#]\} = \mathcal{S}_{13}$. No sentence will ever use this line

of $\mathcal{T}_1(G)$, and Theorem 1 will show that the presence of this line does not affect the validity of $\mathcal{T}_1(G)$ as a recognizer. However, the possibility of such spurious lines in a table allows the possibility that a spurious conflict would make a grammar falsely appear not to be $\text{CS}(k)$. This actually happens for G . Using the alternative definitions, we obtain $[5, 0, \#] \in \mathcal{S}_8''$ since $[4, 0, B] \in \mathcal{S}_8''$ due to $[2, 1, \#] \in \mathcal{S}_8$ via derivation (2) and $[5, 0, \#] \in \mathcal{S}_8''$ due to $[4, 0, B] \in \mathcal{S}_8''$ via derivation (1). Due to $[5, 0, \#] \in \mathcal{S}_8''$ we have $d \in L_0(\mathcal{S}_8)$ via derivation (1); hence, $L_0(\mathcal{S}_8) \cap L_7(\mathcal{S}_8) \neq \emptyset$, and G would not be considered to be $\text{CS}(1)$. Formally, there is nothing wrong with this, since we would be using a different definition. Esthetically, however, this seems very wrong, since the line of the table that causes the conflict is never used for any sentence.

In the example above, the reason that $[6, 0, \#]$ is a member of $\mathcal{S}_0''$ under the new definitions is that derivation (2) is used to justify $[6, 0, \#] \in \mathcal{S}_0''$ but derivation (2) cannot be used to justify $[4, 0, B] \in \mathcal{S}_0''$. More generally, this definition generates members of $\mathcal{S}''$ from members of $\mathcal{S}'$, whereas the definitions we have adopted require that members of $\mathcal{S}''$ be generated only from members of $\mathcal{S}$.

Another Definition of $L_0(\mathcal{S})$

The definition of $L_0(\mathcal{S})$ given above considers $L_0(\mathcal{S})$ to be composed of contributions from the different members of $\mathcal{S}$. It will be useful later to have $L_0(\mathcal{S})$ divided even more finely—into contributions from each member of $\mathcal{S}'$.

LEMMA 1. $L_0(\mathcal{S}) = L_0'(\mathcal{S}) \cup L_0''(\mathcal{S})$, where

$$L_0'(\mathcal{S}) \triangleq \{x\eta \mid |x\eta| = k \ \& \ (\exists [p, j, \theta] \in \mathcal{S})$$

$$[x = X_{p,j+1} \ \& \ (\exists \alpha \in V^*)(\exists \gamma, \delta \in V^* \#^*)(\exists \omega \in L(G))$$

$$(S' \Rightarrow \alpha Y_{p1} \dots Y_{pn_p} \theta \gamma$$

$$\xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p,j+1} \underline{X_{p,j+2} \dots X_{pn_p} \theta \gamma}$$

$$\Rightarrow \alpha X_{p1} \dots X_{pj} X_{p,j+1} \eta \delta$$

$$\Rightarrow \omega \#^k \text{ is a rightmost derivation with the first step,}$$

$$\text{if any, in } \alpha X_{p1} \dots X_{p,j+1} \eta \delta \Rightarrow \omega \#^k \text{ not wholly}$$

$$\text{within } \eta \delta \}}];$$

$$\begin{aligned}
L_0''(\mathcal{S}) &\triangleq \{x\eta \mid |x\eta| = k \ \& \ (\exists [p, j, \theta] \in \mathcal{S})(\exists [q, 0, \theta'] \in \mathcal{S}'') \\
&\quad [x = X_{q1} \ \& \ (\exists \alpha \in V^*)(\exists \gamma, \delta, \zeta \in V^* \#^*)(\exists \omega \in L(G)) \\
&\quad (S' \Rightarrow \alpha Y_{p1} \dots Y_{pm_p} \theta \gamma \\
&\quad \xrightarrow{p} \alpha X_{p1} \dots X_{pj} \underline{X_{p,j+1} \dots X_{pn_p} \theta \gamma} \\
&\quad \Rightarrow \alpha X_{p1} \dots X_{pj} Y_{q1} \dots Y_{qm_q} \theta' \delta \\
&\quad \xrightarrow{q} \alpha X_{p1} \dots X_{pj} \underline{X_{q1} X_{q2} \dots X_{qn_q} \theta' \delta} \\
&\quad \Rightarrow \alpha X_{p1} \dots X_{pj} X_{q1} \eta \delta \\
&\quad \Rightarrow \omega \#^k \text{ is a rightmost derivation with the first step,} \\
&\quad \text{if any, in } \alpha X_{p1} \dots X_{pj} X_{q1} \eta \delta \Rightarrow \omega \#^k \text{ not wholly} \\
&\quad \text{within } \eta \delta)]\}.
\end{aligned}$$

Proof. If $x\eta \in L_0(\mathcal{S})$, either $x = X_{p,j+1}$ and, in the derivation justifying $x\eta \in L_0(\mathcal{S})$, x was carried down from the line $\alpha X_{p1} \dots X_{pn_p} \theta \gamma$ to the line $\alpha X_{p1} \dots X_{pj} x \eta \delta$ or $x \neq X_{p,j+1}$ or it was not carried down. In the former case $x\eta \in L_0'(\mathcal{S})$, and in the latter case, $x\eta \in L_0''(\mathcal{S})$. Conversely, the derivation establishing that $x\eta \in L_0'(\mathcal{S})$ or $x\eta \in L_0''(\mathcal{S})$ also establishes that $x\eta \in L_0(\mathcal{S})$. $\square$

The Extended Table $\bar{\mathcal{T}}_k(G)$

Let G be a $CS(k)$ grammar, and let $\mathcal{T}_k(G)$ be its parsing table. We will denote by $\bar{\mathcal{T}}_k(G)$ any parsing table that satisfies the definition below. Informally, any table $\bar{\mathcal{T}}_k(G)$ is a “superset” of $\mathcal{T}_k(G)$ —for every state $\mathcal{S}, L_i(\mathcal{S})$ in $\bar{\mathcal{T}}_k(G)$ is a superset of $L_i(\mathcal{S})$ in $\mathcal{T}_k(G)$, and $\bar{\mathcal{T}}_k(G)$ may contain states not present in $\mathcal{T}_k(G)$. $\mathcal{T}_k(G)$ itself is always a $\bar{\mathcal{T}}_k(G)$.

Definition of $\bar{\mathcal{T}}_k(G)$

- Step 1. Define $\mathcal{S}_0 \triangleq \{[0, 0, A]\}$.
- Step 2. Define $\mathcal{S}''$ as for $\mathcal{T}_k(G)$. Let $\bar{\mathcal{S}}''$ be any superset of $\mathcal{S}''$ such that $[p, j, \theta] \in \bar{\mathcal{S}}'' \supset p > 0 \ \& \ j = 0$.
- Step 3. Define $\mathcal{S}' \triangleq \mathcal{S} \cup \mathcal{S}''$ and $\bar{\mathcal{S}}' = \mathcal{S} \cup \bar{\mathcal{S}}''$.
- Step 4. Define $L_0(\mathcal{S})$ as for $\mathcal{T}_k(G)$. Let $\bar{L}_0(\mathcal{S})$ be any superset of $L_0(\mathcal{S})$ such that $x\eta \in \bar{L}_0(\mathcal{S}) \supset |x\eta| = k \ \& \ (\exists [p, j, \theta] \in \bar{\mathcal{S}}')(x = X_{p,j+1})$.

Step 5. Define $L_p(\mathcal{S})$, $1 \leq p \leq \pi$, as for $\mathcal{T}_k(G)$.

Step 6. If $A(\mathcal{S}, \theta)$ is unique for all θ , continue. Otherwise, the supersets chosen so far cannot lead to a $\mathcal{T}_k(G)$.

Step 7. Define $S(\mathcal{S})$ and $G(\mathcal{S}, x)$ as for $\mathcal{T}_k(G)$, using $\bar{\mathcal{S}}'$ instead of $\mathcal{S}'$.

Step 8. Define $\mathcal{S}_F$ as for $\mathcal{T}_k(G)$, with $A(\mathcal{S}_F, \#^k) = \text{accept}$.

Recognition of $CS(k)$ Grammars

We will show that the $CS(k)$ processor, when directed by an appropriate table, is a valid recognizer. This will be established by Theorems 1, 2, and 3. Lemmas 2 and 3 are needed for Theorem 1, and Lemma 4 is needed for Theorem 2.

LEMMA 2. *If the processor for $\bar{\mathcal{T}}_k(G)$ arrives at a configuration $\mathcal{S}_0 \mathcal{S}_{(1)} \dots \mathcal{S}_{(n)} \mid \beta$, $n \geq 0$, then for every $[p, j, \theta] \in \mathcal{S}'_{(n)}$, either $j = 0$ or $x_1 \dots x_n = X_{p1} \dots X_{pj}$.*

Proof. By induction on stack length.

Basis. $n = 0$. $\mathcal{S}_0 = \{[0, 0, A]\}$ and every $[p, j, \theta] \in \mathcal{S}'_0$ has $j = 0$ by definition.

Induction. Assume that the lemma is true for stacks of length n . Show that it is true for stacks of length $n + 1$: $\mathcal{S}_0 \mathcal{S}_{(1)} \dots \mathcal{S}_{(n)}$. If $[p, j, \theta] \in \bar{\mathcal{T}}'_{(n)}$, then $j = 0$, by definition. So, suppose $[p, j, \theta] \in \mathcal{S}_{(n)}$ and $j > 0$. x_n could only have been put on the stack by performing a shift in state $\mathcal{S}_{(n-1)}$, and we must have $G(\mathcal{S}_{(n-1)}, x_n) = \mathcal{S}_{(n)}$. But Step 7 in the definitions of $\bar{\mathcal{T}}_k(G)$ and $\mathcal{T}_k(G)$ insures that $[p, j - 1, \theta] \in \bar{\mathcal{T}}'_{(n-1)}$, and $x_n = X_{p,j}$. The induction hypothesis then shows that $x_{n-j+1} \dots x_{n-1} = X_{p1} \dots X_{p,j-1}$. Hence, $x_{n-j+1} \dots x_n = X_{p1} \dots X_{pj}$. $\square$

LEMMA 3. *Let $\alpha \in V^*$, and suppose the processor for $\bar{\mathcal{T}}_k(G)$ performs $T > 0$ operations (shift's, reduce's, accept or reject) during the processing of $\alpha \#^k$. Let the configuration of the processor after performing t operations ($0 \leq t \leq T$) be denoted*

$$\begin{array}{c} \mathcal{S}_{0,t} \quad \mathcal{S}_{1,t} \dots \mathcal{S}_{\ell_t,t} \\ \bullet \quad x_{1,t} \dots x_{\ell_t,t} \end{array} \mid x_t \theta_t \beta_t, \quad \text{where } |x_t \theta_t| = k.$$

Then $(\forall t_1, t_2)(0 \leq t_1 \leq t_2 \leq T)$

$$x_{1,t_2} \dots x_{\ell_{t_2},t_2} x_{t_2} \theta_{t_2} \beta_{t_2} \Rightarrow x_{1,t_1} \dots x_{\ell_{t_1},t_1} x_{t_1} \theta_{t_1} \beta_{t_1}.$$

Proof. Informally, we just note that a *shift* action does not change the string $x_1, t \dots x_{\ell_t}, t x_t \theta_t \beta_t$, and Lemma 2 implies that a *reduce* action replaces the RHS of a rule by its subject. This can be easily formalized as an induction on $t_2 - t_1$. $\square$

THEOREM 1. *Let $\alpha \in V^*$. If the processor for $\bar{\mathcal{T}}_k(G)$ accepts $\alpha \#^k$, then $S \Rightarrow \alpha$.*

Proof. First we note that the processor accepts only in the configuration $\begin{smallmatrix} \mathcal{S}_0 \mathcal{S}_F \\ \bullet S \end{smallmatrix} \Big| \#^k$. This is true since (1) the action *accept* occurs only in $\mathcal{S}_F$ for the lookahead string $\#^k$; (2) since $[0, 1, A] \in \mathcal{S}_F$, S is on the bottom track of L by Lemma 2; (3) there must be a state below $\mathcal{S}_F$ on the stack, since $\mathcal{S}_F$ can be on the stack only due to a *shift* action; and the state below $\mathcal{S}_F$ must be $\mathcal{S}_0$, since only $\mathcal{S}_0$ contains $[0, 0, A]$; (4) this occurrence of $\mathcal{S}_0$ must be at the bottom of the stack, since no state has $\mathcal{S}_0$ in its GOTO column.

Now, suppose the processor for $\bar{\mathcal{T}}_k(G)$ accepts $\alpha \#^k$ after T operations. The processor started in the configuration $\begin{smallmatrix} \mathcal{S}_0 \\ \bullet \end{smallmatrix} \Big| \alpha \#^k$ after 0 operations. Applying Lemma 3 with $t_1 = 0$ and $t_2 = T$, we have $S \#^k \Rightarrow \alpha \#^k$, and hence $S \Rightarrow \alpha$. $\square$

Lemma 4 is a technical lemma for the proof of Theorem 2.

LEMMA 4. *Let Σ be a nonnull sequence of states, with state $\mathcal{S}$ being the rightmost, and let $\nu \in V^*$ with $|\Sigma| = |\nu| + 1$. Let $\alpha \in V^*$, $\beta \in V^* \#^k$, $\zeta \in V^* \#^*$, and let $[p, j, \theta] \in \mathcal{P}'$.*

If there is some $\rho \in V^*$ such that the processor for $\bar{\mathcal{T}}_k(G)$ reaches the configuration $\begin{smallmatrix} \Sigma \\ \bullet \nu \end{smallmatrix} \Big| \beta$ when started in $\begin{smallmatrix} \mathcal{S}_0 \\ \bullet \end{smallmatrix} \Big| \rho \#^k$ and there is some $\omega \in L(G)$ having a rightmost derivation of the form,

$$\begin{aligned}
 S' &\Rightarrow \alpha Y_{p1} \dots Y_{pn_p} \theta \zeta \\
 &\xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p, j+1} \dots X_{pn_p} \theta \zeta \\
 &= \nu \underline{X_{p, j+1} \dots X_{pn_p}} \theta \zeta \\
 &\Rightarrow \nu \beta \\
 &\Rightarrow \omega \#^k \text{ with the first step, if any, in } \nu \beta \Rightarrow \omega \#^k \text{ not wholly to the right of} \\
 &\quad \text{the first symbol of } \beta,
 \end{aligned}$$

then the processor will reach the configuration,

$$\begin{array}{c} \Sigma \\ \bullet \nu \end{array} \Big| X_{p,j+1} \delta,$$

where $\nu X_{p,j+1} \delta$ is the last line of the derivation $\nu X_{p,j+1} \dots X_{pn_p} \theta \zeta \Rightarrow \nu \beta$ in which $X_{p,j+1}$ has been carried down from the line $\nu X_{p,j+1} \dots X_{pn_p} \theta \zeta$.

Proof. The proof is rather long, and is given in Appendix A.

THEOREM 2. For all $\alpha \in V^*$, if α is a line in a rightmost derivation of a sentence such that the first replacement in α , if any, involves its first symbol, then the processor for $\bar{\mathcal{T}}_k(G)$ accepts $\alpha \#^k$.

Proof. Suppose that the processor for $\bar{\mathcal{T}}_k(G)$ starts in the configuration $\begin{array}{c} \mathcal{S}_0 \\ \bullet \end{array} \Big| \alpha \#^k$. Since $S \Rightarrow \alpha$ iff $S' \Rightarrow \alpha \#^k$, and rule 0 is the only rule with S' as its subject, a rightmost derivation of $\alpha \#^k$ from S' exists and has the form, $S' \rightarrow S \#^k \Rightarrow \alpha \#^k$. Applying Lemma 4 with $\alpha = \theta = \zeta = \Lambda$, $p = j = 0$, $\rho = \alpha$, $\nu = \Lambda$, $\Sigma = \mathcal{S} = \mathcal{S}_0$, $\beta = \alpha \#^k$, the processor will reach the configuration, $\begin{array}{c} \mathcal{S}_0 \\ \bullet \end{array} \Big| S \#^k$. The processor will perform a shift since $S \#^{k-1} \in L_0(\mathcal{S}_0)$, and will reach the configuration $\begin{array}{c} \mathcal{S}_0 \mathcal{S}_F \\ \bullet S \end{array} \Big| \#^k$. It will then *accept*, since $A(S_F, \#^k) = \text{accept}$. $\square$

THEOREM 3. Let $\alpha \in V_T^*$. The processor for $\bar{\mathcal{T}}_k(G)$ accepts $\alpha \#^k$ iff $\alpha \in L(G)$.

Proof. Immediate from Theorems 1 and 2. $\square$

Theorem 4 shows that the processor for $\bar{\mathcal{T}}_k(G)$ operates by constructing, in reverse order, a rightmost derivation of the input string.

THEOREM 4. Let $\alpha \in V^*$, and suppose that the processor for $\bar{\mathcal{T}}_k(G)$ accepts $\alpha \#^k$ after performing T operations. The T -th operation is *accept*; suppose that the r_1 -th, r_2 -th, ..., r_R -th operations are *reduce* p_1 , *reduce* p_2 , ..., *reduce* p_R , for $1 \leq r_1 < r_2 < \dots < r_R = T - 2$ and $1 \leq p_i \leq \pi$ for $1 \leq i \leq R$, and the other operations are *shift*. Let the configuration after t operations ($0 \leq t \leq T$)

be denoted $\begin{array}{c} \mathcal{S}_0 \mathcal{S}_{1,t} \dots \mathcal{S}_{t,t} \\ \bullet \beta_t \end{array} \Big| \theta_t \gamma_t$ where $|\theta_t| = k$. Then

(1) The sequence

$$S \#^k = \beta_{r_R} \theta_{r_R} \gamma_{r_R}, \beta_{r_{R-1}} \theta_{r_{R-1}} \gamma_{r_{R-1}}, \dots, \beta_{r_1} \theta_{r_1} \gamma_{r_1}, \beta_0 \theta_0 \gamma_0 = \alpha \#^k$$

is a rightmost derivation of $\alpha \#^k$ from $S \#^k$,

(2) The i -th step of this derivation applies rule p_{R-i+1} to the first $m_{p_{R-i+1}}$ symbols of $\theta_{r_{R-i+1}}$ to produce the $i + 1$ -st line of the derivation, for $1 \leq i \leq R$.

Proof. Lemma 3 shows that for $1 \leq i \leq T - 1$, $\beta_i \theta_i \gamma_i = \beta_{i-1} \theta_{i-1} \gamma_{i-1}$ if the i -th operation is *shift*, and $\beta_i \theta_i \gamma_i \xrightarrow{p} \beta_{i-1} \theta_{i-1} \gamma_{i-1}$ if the i -th operation is *reduce* p . This shows that the sequence (1) is a derivation of $\alpha \#^k$ from $S \#^k$. To show that it is a rightmost derivation, consider the r_i -th operation, *reduce* p_i . The n_{p_i} -th state popped from the stack contained $[p_i, 1, \theta_{r_i-1}]$; hence, $[p_i, 0, \theta_{r_i-1}] \in \mathcal{S}_{\ell_{r_i}, r_i}''$ since the n_{p_i} -th state popped must have been pushed onto the stack by performing a shift in state $\mathcal{S}_{\ell_{r_i}, r_i}$. The derivation showing that $[p_i, 0, \theta_{r_i-1}] \in \mathcal{S}_{\ell_{r_i}, r_i}''$ also shows that the k -symbol prefix of $Y_{p_i 1} \dots Y_{p_i m_{p_i}} \theta_{r_i-1}$ is a member of $L_0(\mathcal{S}_{\ell_{r_i}, r_i})$. Since

$$Y_{p_i 1} \dots Y_{p_i m_{p_i}} \theta_{r_i-1} \gamma_{r_i-1} = \theta_{r_i} \gamma_{r_i}$$

(which establishes statement (2) of the theorem), the processor will perform a *shift* immediately after each reduce, and the derivation it is producing is a rightmost one. $\square$

Properties of $CS(k)$ Grammars

We will prove that $CS(k)$ grammars are unambiguous, and we will prove three undecidability results. Lemma 5 will be needed to prove unambiguity, and itself gives some insight into the table $\mathcal{T}_k(G)$. The proofs of Lemma 5 and Theorem 5 are rather involved, and are given in Appendix B.

LEMMA 5. *If the processor for $\mathcal{T}_k(G)$ reaches the configuration,*

$$\left. \begin{array}{c} \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \\ \bullet x_1 \dots x_n \end{array} \right| x_{n+1} \dots x_t,$$

where $x_{t-l+1} \dots x_t = \#^k$, and if there exist $\gamma \in V^* \#^*$ and $\omega \in L(G)$ such that

$$\begin{aligned} S' &\Rightarrow x_1 \dots x_{n-j} Y_{p1} \dots Y_{p m_p} \theta \gamma \\ &\xrightarrow{p} x_1 \dots x_{n-j} X_{p1} \dots X_{p n_p} \theta \gamma \\ &= x_1 \dots x_{n-j} \dots x_n \underline{X_{p, j+1} \dots X_{p n_p} \theta \gamma} \\ &\Rightarrow x_1 \dots x_n x_{n+1} \dots x_t \\ &\Rightarrow \omega \#^k \text{ is a rightmost derivation,} \end{aligned}$$

then $[p, j, \theta] \in \mathcal{S}_n'$.

THEOREM 5. $CS(k)$ grammars are unambiguous.

Theorems 6 and 7 and their corollary establish three undecidability results for $\text{CS}(k)$ grammars.

THEOREM 6. *The problem of deciding, for a given context-sensitive grammar G , whether or not there exists an integer k such that G is $\text{CS}(k)$ is recursively unsolvable.*

Proof. Knuth proves that it is undecidable, for an arbitrary context-free grammar G , whether there exists any k such that G is $\text{LR}(k)$. His proof is valid even if G is constrained to have no Λ -rules. But every such grammar is a context-sensitive grammar, and is $\text{CS}(k)$ iff it is $\text{LR}(k)$. $\square$

THEOREM 7. *For each $k \geq 0$, there is no algorithm that will decide whether or not an arbitrary context-sensitive grammar G is $\text{CS}(k)$, and construct $\mathcal{T}_k(G)$ if it exists.*

Proof. We show that if such an algorithm existed, the emptiness problem for context-sensitive grammars would be solvable, which it is not. First, note that for any G such that $L(G) = \emptyset$, $\mathcal{T}_k(G)$ exists, and hence G is $\text{CS}(k)$ for every k : $\mathcal{T}_k(G)$ consists of the single state $\mathcal{S}_0$ with $A(\mathcal{S}_0, \theta) = \text{reject}$ for all $\theta \in (V_T')^k$. Now, suppose that for some integer k the algorithm mentioned in the theorem existed. Apply this algorithm to an arbitrary grammar G : if G is not $\text{CS}(k)$, then $L(G) \neq \emptyset$; if G is $\text{CS}(k)$, the algorithm constructs $\mathcal{T}_k(G)$, and it can be decided whether or not $\mathcal{T}_k(G)$ is the unique table for a grammar generating the empty language. $\square$

COROLLARY. *For each $k \geq 0$, let $A_k(G, \mathcal{T})$ be a predicate that is true iff G is $\text{CS}(k)$ and $\mathcal{T}$ is $\mathcal{T}_k(G)$. Then A_k is undecidable.*

Proof. There are only finitely many tables that could be $\mathcal{T}_k(G)$ for each G and k . If A_k were decidable, Theorem 7 would be violated. Alternatively, let $\mathcal{T}_0$ be the unique table for any grammar generating the empty language. As in Theorem 7, $\mathcal{T}_0$ contains only the state $\mathcal{S}_0$, with $A(\mathcal{S}_0, \theta) = \text{reject}$ for all $\theta \in (V_T')^*$. For an arbitrary grammar G , $A_k(G, \mathcal{T}_0)$ is true iff $L(G) = \emptyset$. Hence $A_k(G, \mathcal{T}_0)$, and *a fortiori*, $A_k(G, \mathcal{T})$, are undecidable.

APPENDIX A: PROOF OF LEMMA 4

LEMMA 4. *Let Σ be a nonnull sequence of states, with state $\mathcal{S}$ being the rightmost, and let $\nu \in V^*$ with $|\Sigma| = |\nu| + 1$. Let $\alpha \in V^*$, $\beta \in V^*\#\#^k$, $\zeta \in V^*\#\#^*$, and let $[p, j, \theta] \in \mathcal{P}'$.*

If there is some $\rho \in V^*$ such that the processor for $\mathcal{T}_k(G)$ reaches the configuration $\frac{\Sigma}{\bullet_v} \Big| \beta$ when started in $\frac{\mathcal{S}_0}{\bullet} \Big| \rho \#^k$ and there is some $\omega \in L(G)$ having a rightmost derivation of the form,

$$\begin{aligned}
 S' &\Rightarrow \alpha Y_{p1} \dots Y_{pm_p} \theta \zeta \\
 &\xrightarrow{p} \alpha X_{p1} \dots X_{pj} X_{p,j+1} \dots X_{pn_p} \theta \zeta \\
 &= \underline{\nu X_{p,j+1} \dots X_{pn_p} \theta \zeta} \\
 &\Rightarrow \nu \beta \\
 &\Rightarrow \omega \#^k \text{ with the first step, if any, in } \nu \beta \Rightarrow \omega \#^k \text{ not wholly to the right of} \\
 &\quad \text{the first symbol of } \beta,
 \end{aligned}$$

then the processor will reach the configuration, $\frac{\Sigma}{\bullet_v} \Big| X_{p,j+1} \delta$, where $\nu X_{p,j+1} \delta$ is the last line of the derivation $\nu X_{p,j+1} \dots X_{pn_p} \theta \zeta \Rightarrow \nu \beta$ in which $X_{p,j+1}$ has been carried down from the line $\nu X_{p,j+1} \dots X_{pn_p} \theta \zeta$.

Proof. By induction on m = number of lines of the derivation after $\nu X_{p,j+1} \delta$ up to and including the line $\nu \beta$.

Basis. $m = 0$. Hence, $\nu X_{p,j+1} \delta = \nu \beta$ and $X_{p,j+1} \delta = \beta$, so the processor is already in the desired configuration.

Induction Step. Assume that the lemma is true if there are fewer than $m \geq 1$ lines after $\nu X_{p,j+1} \delta$ up through $\nu \beta$. Suppose there are $m \geq 1$ such lines. The proof has seven steps:

- (1) The part of the derivation being considered must have the form,

$$\begin{aligned}
 &\nu X_{p,j+1} \underline{X_{p,j+2} \dots X_{pn_p} \theta \zeta} \\
 &\Rightarrow \nu \underline{X_{p,j+1} \xi \eta \beta_2} = \nu X_{p,j+1} \delta \\
 &\xrightarrow{q} \nu \underline{\gamma \eta \beta_2} \\
 &\Rightarrow \nu \beta_1 \beta_2,
 \end{aligned}$$

where $\delta = \xi \eta \beta_2$, $\beta = \beta_1 \beta_2$, $X_{p,j+1} \xi \rightarrow \gamma$ is the q -th rule, and $\nu \gamma \eta \beta_2 \Rightarrow \nu \beta_1 \beta_2$ has m lines.

(2) Let $\gamma = g_1 g_2 \dots g_{n_q}$. Let μ be the k -symbol prefix of $\eta \beta_2$. ($\eta \beta_2$ ends in $\#^k$, hence μ exists.) Then $[q, 0, \mu] \in \mathcal{S}''$ due to $[p, j, \theta] \in \mathcal{S}$.

(3) Consider the m -line derivation $\nu\gamma\eta\beta_2 \Rightarrow \nu\beta_1\beta_2$. This derivation is $\nu g_1 \dots g_{n_r} \eta \beta_2 \Rightarrow \nu\beta_1\beta_2$ by Step (2). There is a unique sequence of integers, $(i_1, i_2, \dots, i_r)$, $r \geq 0$, $0 \leq i_1 < i_2 < \dots < i_r < n_q$ such that: The first replacement of any g_i 's replaces g_{i_r+1} plus possibly $g_{i_r+2}, \dots$; the next replacement that uses any of $g_1, \dots, g_{i_r}$ uses $g_{i_{r-1}+1}$ but not $g_{i_{r-1}}$, etc. The derivation has the form,

$$\begin{aligned}
 & \nu g_1 \dots g_{i_r} \underline{g_{i_r+1} \dots g_{n_q}} \eta \beta_2 = \nu \gamma \eta \beta_2 \\
 & \Rightarrow \nu g_1 \dots g_{i_{r-1}} \underline{g_{i_{r-1}+1} \dots g_{i_r} \beta'_r} \beta_2 \\
 & \Rightarrow \nu g_1 \dots g_{i_{r-2}+1} \dots g_{i_{r-1}} \beta'_{r-1} \beta_2 \\
 & \Rightarrow \nu g_1 \dots g_{i_1} \underline{g_{i_1+1} \dots g_{i_2} \beta'_2} \beta_2 \\
 & \Rightarrow \nu g_1 \dots g_{i_1} \beta_1' \beta_2 = \nu \beta_1 \beta_2.
 \end{aligned}$$

Thus, we have that

$\nu g_1 \dots g_{i_1}$, but not g_{i_1+1} , was carried down from $\nu\gamma\eta\beta_2$ to $\nu\beta_1\beta_2$;

$\nu g_1 \dots g_{i_2}$, but not g_{i_2+1} , was carried down from $\nu\gamma\eta\beta_2$ to the first line in which g_{i_1+1} was replaced;

$\nu g_1 \dots g_{i_\ell}$, but not $g_{i_\ell+1}$, was carried down from $\nu\gamma\eta\beta_2$ to the first line in which $g_{i_{\ell-1}+1}$ was replaced;...

$\nu g_1 \dots g_{i_r}$, but not $g_{i_{r+1}}$, is not involved in the first replacement.

[Note that the first step, if any, in the derivation $\nu\gamma\eta\beta_2 \Rightarrow \nu\beta_1\beta_2$ must use part of γ , since β_2 is not involved in the derivation, and any step wholly within η would have been performed as part of $\nu X_{p,j+1} \dots X_{pn_p} \theta \zeta \Rightarrow \nu X_{p,j+1} \xi \eta \beta_2$ (see Step (1)) since the derivation is rightmost.]

(4) Starting in the configuration $\begin{smallmatrix} \Sigma \\ \bullet \nu \end{smallmatrix} \Big| \beta_1 \beta_2 = \begin{smallmatrix} \Sigma \\ \bullet \nu \end{smallmatrix} \Big| g_1 \dots g_{i_1} \beta_1' \beta_2$, the processor for $\mathcal{T}_k(G)$ will perform i_1 shifts, reaching the configuration, $\begin{smallmatrix} \Sigma \mathcal{S}_{(i_1)} \dots \mathcal{S}_{(i_1)} \\ \bullet \nu g_1 \dots g_{i_1} \end{smallmatrix} \Big| \beta_1' \beta_2$, where $[q, i, \mu] \in \mathcal{S}_{(i)}$ for $1 \leq i \leq i_1$.

[Proof. If $i_1 = 0$, there is nothing to prove. Otherwise, the k -symbol prefix of $g_1 \dots g_{i_1} \beta_1' \beta_2 = \beta$ is a member of $L_0(\mathcal{S})$, due to $[p, j, \theta] \in \mathcal{S}$, since the first step, if any, in $\nu\beta \Rightarrow \omega \#^k$ is not wholly to the right of g_1 (the first symbol of β). Therefore, $[q, 1, \mu] \in G(\mathcal{S}, g_1) = \mathcal{S}_{(1)}$. If $i_1 > 1$, then for each i , $1 \leq i < i_1$, the k -symbol prefix of $g_{i+1} \dots g_{i_1} \beta_1' \beta_2$ is a member of

$L_0(\mathcal{S}_{(i)})$ due to $[q, i, \mu] \in \mathcal{S}_{(i)}$, since the first step, if any, in $\nu\beta \Rightarrow \omega\#^k$ cannot be wholly to the right of g_{i+1} since it is not wholly to the right of g_1 , as seen above. Therefore, $[q, i+1, \mu] \in G(\mathcal{S}_{(i)}, g_{i+1}) = \mathcal{S}_{(i+1)}$.]

(5) Since the derivation $\nu\gamma\eta\beta_2 \Rightarrow \nu\beta_1\beta_2$ has m lines, there are fewer than m lines after the line $\nu g_1 \dots g_{i_1+1} \dots g_{i_2} \beta_2' \beta_2$. Applying the induction hypothesis (with ν being $\nu g_1 \dots g_{i_1}$), we reach the configuration,

$$\begin{array}{c} \Sigma \mathcal{S}_{(1)} \dots \mathcal{S}_{(i_1)} \\ \bullet \nu g_1 \dots g_{i_1} \end{array} \left| g_{i_1+1} \dots g_{i_2} \beta_2' \beta_2 \right.$$

since $g_{i_1+1} \dots g_{i_2}$ was carried down to the line $\nu g_1 \dots g_{i_1} \dots g_{i_2} \beta_2' \beta_2$. As in Step (4), the processor will then perform $i_2 - i_1$ shifts.

(6) After performing Step (4) once and performing Step (5) $r \geq 0$ times, we reach the configuration, $\begin{array}{c} \Sigma \mathcal{S}_{(1)} \dots \mathcal{S}_{(n_q)} \\ \bullet \nu g_1 \dots g_{n_q} \end{array} \left| \eta\beta_2 \right.$, where $[q, i, \mu] \in \mathcal{S}_{(i)}$ for $1 \leq i \leq n_q$.

The processor will now perform a reduce q action, since $[q, n_q, \mu] \in \mathcal{S}_{(n_q)}$ and μ is a prefix of $\eta\beta_2$ (step 2)).

(7) After the *reduce* q , the processor is in the configuration, $\begin{array}{c} \Sigma \\ \bullet \nu \end{array} \left| X_{p,j+1} \xi \eta \beta_2 \right.$, which proves the lemma. $\square$

APPENDIX B: PROOFS OF LEMMA 5 AND THEOREM 5

LEMMA 5. *If the processor for $\mathcal{T}_k(G)$ reaches the configuration,*

$$\begin{array}{c} \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \\ \bullet x_1 \dots x_n \end{array} \left| x_{n+1} \dots x_t \right.,$$

where $x_{t-k+1} \dots x_t = \#^k$, and if there exist $\gamma \in V^* \#^*$ and $\omega \in L(G)$ such that

$$\begin{aligned} S' &\Rightarrow x_1 \dots x_{n-j} Y_{p1} \dots Y_{pm_p} \theta \gamma \\ &\xrightarrow{p} x_1 \dots x_{n-j} X_{p1} \dots X_{pn_p} \theta \gamma \\ &= x_1 \dots x_{n-j} \dots x_n \underline{X_{p,j+1} \dots X_{pn_p} \theta \gamma} \\ &\Rightarrow x_1 \dots x_n x_{n+1} \dots x_t \\ &\Rightarrow \omega \#^k \text{ is a rightmost derivation,} \end{aligned}$$

then $[p, j, \theta] \in \mathcal{S}_n'$.

Proof. By induction on n .

Basis. $n = 0$. Then $j = 0$. Suppose

$$S' \Rightarrow Y_{p1} \dots Y_{pm_p} \theta \gamma \xrightarrow{p} X_{p1} \dots X_{pn_p} \theta \gamma \Rightarrow x_1 \dots x_t \Rightarrow \omega \#^k$$

is a rightmost derivation of some $\omega \in L(G)$. If $p = 0$, there is nothing to prove. Otherwise, this derivation shows that $[p, 0, \theta] \in \mathcal{S}''_0$.

Induction. Assume the lemma is true for stacks containing n entries ($\mathcal{S}_0 \dots \mathcal{S}_{n-1}$). Suppose the processor reaches the configuration assumed in the lemma with $n \geq 1$. Consider the most recent configuration in which the L pushdown list was $\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_{n-1}$. The R pushdown list must have been $\bullet x_1 \dots x_{n-1}$. $x_n y_1 \dots y_u$ and $A(\mathcal{S}_{n-1}, x_n y_1 \dots y_{k-1})$ must have been *shift*, with

$$G(\mathcal{S}_{n-1}, x_n) = \mathcal{S}_n;$$

if the action were *reduce*, the stack would have become shorter, and this would not be the most recent configuration as assumed, while if $G(\mathcal{S}_{n-1}, x_n) \neq \mathcal{S}_n$, the hypothesis of the lemma could never be true. Since

$$\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_{n-1} \mid \begin{array}{c} x_n y_1 \dots y_u \\ \bullet x_1 \dots x_{n-1} \end{array}$$

is the most recent configuration with the stack as shown, the processor must have gone from the next configuration,

$$\mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \mid \begin{array}{c} y_1 \dots y_u \\ \bullet x_1 \dots x_n \end{array} \quad \text{to} \quad \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_n \mid \begin{array}{c} x_{n+1} \dots x_t \\ \bullet x_1 \dots x_n \end{array}$$

without popping $\mathcal{S}_n$. By Lemma 3 and this observation,

$$x_1 \dots x_n \underline{x_{n+1} \dots x_t} \Rightarrow x_1 \dots x_n y_1 \dots y_u.$$

Now, suppose the derivation assumed in the lemma exists.

If $j > 0$. This derivation and the fact that $x_1 \dots x_n$, and hence $x_1 \dots x_{n-1}$, are not involved in $x_1 \dots x_t \Rightarrow x_1 \dots y_u$, together with the induction hypothesis, show that $[p, j-1, \theta] \in \mathcal{S}'_{n-1}$. Since $G(\mathcal{S}_{n-1}, x_n) = \mathcal{S}_n$ and $x_n = X_{pj}$, $[p, j, \theta] \in \mathcal{S}_n$.

If $j = 0$. From the hypothesis of the lemma,

$$\begin{aligned} S' &\Rightarrow x_1 \dots x_n Y_{p1} \dots Y_{pm_p} \theta \gamma \xrightarrow{p} x_1 \dots x_n \underline{X_{p1} \dots X_{pn_p} \theta \gamma} \\ &\Rightarrow x_1 \dots x_n x_{n+1} \dots x_t \Rightarrow \omega \#^k \text{ is a rightmost derivation.} \end{aligned}$$

Consider the rightmost derivation $S' \Rightarrow x_1 \dots x_n Y_{p1} \dots Y_{pm_p} \theta \gamma$. Since $n > 0$, it must have the form

$$\begin{aligned} S' &\Rightarrow x_1 \dots x_{n-\ell} Y_{r1} \dots Y_{rm_r} \theta' \delta \\ &\xrightarrow{r} x_1 \dots x_{n-\ell} X_{r1} \dots X_{rm_r} \theta' \delta \\ &= x_1 \dots x_{n-\ell} \dots x_n \underline{X_{r,\ell+1} \dots X_{rm_r} \theta' \delta} \\ &\Rightarrow x_1 \dots x_n Y_{p1} \dots Y_{pm_p} \theta \gamma. \end{aligned}$$

Hence, $x_{n-\ell+1} \dots x_n = X_{r1} \dots X_{r\ell}$, with $\ell \geq 1$.

We can apply the induction hypothesis to the configuration

$$\bullet \begin{array}{c} \mathcal{S}_0 \dots \mathcal{S}_{n-1} \\ x_1 \dots x_{n-1} \end{array} \left| x_n y_1 \dots y_u, \right.$$

since

$$\begin{aligned} S' &\Rightarrow x_1 \dots x_{n-\ell} Y_{r1} \dots Y_{rm_r} \theta' \delta \\ &\xrightarrow{r} x_1 \dots x_{n-\ell} X_{r1} \dots X_{rm_r} \theta' \delta \\ &= x_1 \dots x_{n-1} \underline{X_{r\ell} \dots X_{rm_r} \theta' \delta} \\ &\Rightarrow x_1 \dots x_{n-1} \underline{x_n x_{n+1} \dots x_t} \\ &\Rightarrow x_1 \dots x_{n-1} x_n y_1 \dots y_u \\ &\Rightarrow \omega \#^k \end{aligned}$$

is a rightmost derivation and hence $[r, \ell - 1, \theta'] \in \mathcal{S}'_{n-1}$.

Since $A(\mathcal{S}_{n-1}, x_n y_1 \dots y_{k-1}) = \text{shift}$, $G(\mathcal{S}_{n-1}, x_n) = \mathcal{S}_n$ and $X_{r\ell} = x_n$, we have $[r, \ell, \theta'] \in \mathcal{S}_n$. Hence $[p, 0, \theta] \in \mathcal{S}_n''$. $\square$

THEOREM 5. *CS(k) grammars are unambiguous.*

Proof. Let G be a CS(k) grammar for some value of k , and hence $\mathcal{T}_k(G)$

exists. Suppose G is ambiguous. Let ω be any ambiguous sentence. Let $\{D_i \mid i = 1, 2, \dots\}$ be all rightmost derivations of ω , where

$$D_i = \langle [p_{i,s_i}, r_{i,s_i}], [p_{i,s_i-1}, r_{i,s_i-1}], \dots, [p_{i,1}, r_{i,1}] \rangle.$$

Each $[p_{i,j}, r_{i,j}]$ represents the $s_i - j + 1$ -st step of the derivation: Apply rule $p_{i,j}$ to the $r_{i,j} + 1$ -st through $r_{i,j} + m_{p_{i,j}}$ -th symbols of the $s_i - j + 1$ -st line of the derivation.

Let $u \geq 1$ be the greatest integer such that the last $u - 1$ steps of all the D_i are the same. Let D_1 be the D_i with the least value of $r_{i,u} + n_{p_{i,u}}$ (any one of these, if more than one), and let D_2 be the D_i with the next smallest value (the same value if possible), but such that D_2 and D_1 have different values of $[p_{i,u}, r_{i,u}]$. Let $r_{1,u} = a$, $r_{2,u} = b$, $p_{1,u} = p$ and $p_{2,u} = q$.

D_1 and D_2 are as follows:

D_1 .

$$\begin{array}{c} S' \\ \vdots \\ x_1 \dots x_a \underline{Y_{p1} \dots Y_{pm_p}} x_{a+n_p+1} \dots x_t, \end{array} \quad (\alpha_1)$$

$$\begin{array}{c} x_1 \dots x_a X_{p1} \dots X_{pn_p} x_{a+n_p+1} \dots x_t. \\ \vdots \\ \omega \#^k \end{array} \quad (\beta)$$

D_2 .

$$\begin{array}{c} S' \\ \vdots \\ x_1 \dots x_b \underline{Y_{q1} \dots Y_{qm_q}} x_{b+n_q+1} \dots x_t \end{array} \quad (\alpha_2)$$

$$\begin{array}{c} x_1 \dots x_b X_{q1} \dots X_{qn_q} x_{b+n_q+1} \dots x_t \\ \vdots \\ \omega \#^k \end{array} \quad (\beta)$$

where β and the $u - 1$ lines below it in D_1 are the same as in D_2 , obtained by the same steps, but $\alpha_1 \rightarrow \beta$ and $\alpha_2 \rightarrow \beta$ used rules p and q as shown. We have $a + n_p \leq b + n_q$.

Suppose $\mathcal{T}_k(G)$ is started with $\omega \#^k$. It will accept it, by Theorem 2, and will do so by finding a rightmost derivation, by Theorem 4. Consider the configuration of the processor just after performing $u - 1$ reduce's: If $u = 1$, the processor is in its initial configuration. If $u > 1$, the stack contains the prefix

of β up to the subject of the rule used in going from β to the next line—i.e., L contains the first $r_{i,u-1}$ symbols of β , for any i (since all derivations are the same at this point).

But, since all D_i are rightmost derivations, $r_{j,u-1} \leq r_{j,u} + n_{p_{j,u}}$ for any j —in particular, this is true for the derivation D_i that $\mathcal{F}_k(G)$ is producing. Therefore, $r_{i,u-1} \leq a + n_p$. Let $r_{i,0} = 0$ for all i . The processor for $\mathcal{F}_k(G)$ will do $a + n_p - r_{i,u-1}$ *shift*'s, regardless of which D_i it is generating, and reach the configuration

$$\bullet x_1 \dots x_a \mathcal{S}_0 \mathcal{S}_1 \dots \mathcal{S}_{a+n_p} \left| x_{a+n_p+1} \dots x_t \right.$$

By Lemma 5,

$$[p, n_p, x_{a+n_p+1} \dots x_{a+n_p+k}] \in \mathcal{S}'_{a+n_p}. \quad (*)$$

We consider two cases: (1) $a + n_p \geq b$, and (2) $a + n_p < b$.

Case 1. Have $b \leq a + n_p \leq b + n_q$. By Lemma 5,

$$[q, a + n_p - b, x_{b+n_q+1} \dots x_{b+n_q+k}] \in \mathcal{S}'_{a+n_p}.$$

But we have $A(\mathcal{S}_{a+n_p}, x_{a+n_p+1} \dots x_{a+n_p+k}) = \text{reduce } p$ due to (*), and $A(\mathcal{S}_{a+n_p}, x_{a+n_p+1} \dots x_{a+n_p+k}) = \text{reduce } q$ if $b + n_q = a + n_p$ or *shift* if $b + n_q > a + n_p$. Therefore, there is a conflict in $\mathcal{F}_k(G)$, since $p \neq q \vee a \neq b$.

Case 2. Have $a + n_p < b$. For this case, we consider the form of D_2 . Let δ be the line in D_2 such that x_{a+n_p+1} was carried down from δ to α_2 , but was not carried down to δ . δ was generated from the line above it by using some rule r , such that $x_{a+n_p+1} = X_{r,i}$ for some i , $1 \leq i \leq n_r$. Thus, D_2 has the form,

$$\begin{array}{l} S' \\ \vdots \\ x_1 \dots x_{a+n_p+1-i} \underline{Y_{r1} \dots Y_{rn_r} \theta \zeta} \quad \text{where } |\theta| = k \\ x_1 \dots x_{a+n_p+1-i} X_{r1} \dots X_{rn_r} \theta \zeta \\ x_1 \dots x_{a+n_p+1} \underline{X_{r,i+1} \dots X_{rn_r} \theta \gamma} \\ \vdots \\ x_1 \dots x_{a+n_p+1} \dots x_b \underline{Y_{q1} \dots Y_{qn_q} x_{b+n_q+1} \dots x_t} \quad (\alpha_2) \\ x_1 \dots x_b X_{q1} \dots X_{qn_q} x_{b+n_q+1} \dots x_t \quad (\beta) \\ \vdots \\ \omega \#^k. \end{array}$$

To show that D_2 has this form, we only must show that no step involves $x_1 \dots x_{a+n_p+1}$ in going from δ to α_2 . The rest is general. But this follows from x_{a+n_p+1} being carried down from δ to α_2 , and rule q being applied to α_2 where shown. If any step had been applied to $x_1 \dots x_{a+n_p}$ during $\delta \Rightarrow \alpha_2$, we could not have applied rule q in a rightmost derivation.

By Lemma 5, we have $[r, i - 1, \theta] \in \mathcal{S}'_{a+n_p}$. Therefore, we have

$$A(\mathcal{S}_{a+n_p}, x_{a+n_p+1} \dots x_{a+n_p+k}) = \text{shift},$$

conflicting with (*), contradicting the existence of $\mathcal{T}_k(G)$. $\square$

RECEIVED: September 15, 1969; Revised: February 25, 1970.

REFERENCE

KNUTH, D. E. (1965), On the translation of languages from left to right. *Information and Control* 8, 607–639.