GENERALIZED BOTTOM-UP PARSING

Thomas Gregory Szymanski

TR 73-168

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University for the Degree of

Doctor of Philosophy

May, 1973

This research was supported by the National Science Foundation under Grant GJ-28176 and GJ-33171X.

GENERALIZED BOTTOM-UP PARSING

Thomas Gregory Szymanski, Ph.D.
Cornell University, 1972

ABSTRACT

In this thesis we present a decision procedure for testing the correctness of a broad class of bottom-up parsing machines. Motivated by the work of Colmerauer and Williams, we allow our parsers to find any simple phrase (not necessarily the leftmost) in the input string. Such a parser can be implemented on an automaton using two pushdown stores and can in fact produce a complete parse for an input string in linear time with respect to the length of the input.

Certain restrictions are necessary in order for the method to work, but nevertheless it is sufficiently general to handle most existing classes of parsers. Moreover, the adjustment of certain parameters to this decision procedure gives rise in a natural way to such classes of grammars as the LR(k) class of Knuth, the BRC class of Floyd and the BCP class of Williams. Further adjustment of these parameters suggests other, more general, classes of parsable grammars which we investigate here for the first time.

Among these new classes of grammars is one first suggested by Knuth and given the name LR(k,t). This class is a generalization of the LR(k) method and intuitively is that

class of grammars for which it is possible, in any sentential form, to find one of the t leftmost simple phrases given only that portion of the string to the left of the phrase and the first k characters to its right. We give an exact construction for parsers of this class and present the surprising fact that these grammars can be parsed using a deterministic pushdown automaton.

We also investigate a class herein called LR(k,∞) in which we completely relax the condition that the selected phrase be in any certain location. This latter class, which represents the ultimate left to right bottom-up parser, is shown to be too general to have "nice" decidability properties.

A final class of grammars investigated is designated FPFAP(k), that is, the class of grammars which are parsable in a left to right fashion by a finite state automaton using k characters of lookahead. This class is shown to lie strictly between the LR(k,t) and LR(k,∞) classes.

We conclude by demonstrating the relationships between these and other classes of grammars, not only from the point of view of the grammars themselves, but also with regard to the classes of languages induced by the grammars.

BIOGRAPHICAL SKETCH

Thomas Gregory Szymanski was born in Cleveland, Ohio, on December 13, 1946, the son of Mr. and Mrs. Walter Anthony Szymanski. He was graduated from Case Western Reserve University in June, 1968, with the degree of Bachelor of Science, _magna cum laude_, having majored in Engineering. He then spent two years at the University of Wisconsin, obtaining a Master of Science degree in Computer Science. In September of 1970 he entered the graduate school at Cornell University to further his education in Computer Science.

Mr. Szymanski is a member of Tau Beta Pi and the Association for Computing Machinery. He is married to the former Karen White.

to my Grandmother

ACKNOWLEDGEMENTS

I would first of all like to express my gratitude to Professor John H. Williams, my thesis advisor, for his continued guidance and encouragement not only during the preparation of this dissertation, but also during those arduous months spent searching for a topic. I am also indebted to Professors Juris Hartmanis and Robert Walker who served on my special committee.

The National Science Foundation partially supported this work through grant GJ-33171. Gayle Mykita did a fine job of preparing the final manuscript.

Finally, and most importantly, I thank my wife, Karen, whose love and confidence constituted the real driving force behind this thesis.

TABLE OF CONTENTS

PAGE REFERENCES

TERMS

Notation

Chapter 0 PRELIMINARIES

0.1 Introduction and Background

The subject matter of this dissertation is the parsing of sentences generated by context free grammars. A suitable formalism and adequate solutions already exist for this general problem; see, for instance, Hopcroft and Ullman [HU 69] and Earley [Ea 70]. However, since the main motivation for studying this problem is to facilitate the design of computer languages and, above all, to develop techniques for the quick, and automatic, "understanding" of such languages, we feel that continued attention to this problem is in order.

Historically, the parsing problem has been attacked from two different directions, which we will call "low level" and "high level" approaches. The "low level" approach has resulted in a large collection of special case techniques which work for various restricted subclasses of the context free grammars. These methods are always intended to be efficient, practical and applicable to most "garden variety" programming languages. As a consequence, these approaches usually are applicable only to subcases of the deterministic context free grammars. The reader is referred to Aho and Ullman [AU 72] for a detailed description of many of these techniques.

The "high level" approach on the other hand attempts to devise algorithms for the general case, namely the parsing of arbitrary context free grammars. These approaches tend to lose efficiency because of their ability to handle those pathological conditions (such as ambiguity) which cannot be allowed to occur in programming languages.

In this thesis we have tried to attain a sort of "middle ground" in which we develop techniques for classes of grammars which are much broader than the deterministic grammars and yet sufficiently restricted to avoid those pathologies referred to above.

Our ideas are drawn from essentially four sources. Floyd [Fl 64] in his paper on bounded context (BC) grammars, proved the correctness of his methods by doing a formal analysis of all the ways in which his method could err, and then showing that none of these errors could actually arise. We will extend this idea in Chapter 1 to produce a general model of bottom-up parsing which turns out to be sufficiently powerful to model almost all existing "low level" parsing techniques. Furthermore, the model suggests new parsing techniques which will be investigated at various points throughout the dissertation.

Another wellspring of ideas was furnished by Knuth [Kn 65] when he showed that it was possible to develop parsing methods (viz. the LR(k) method) which use an unbounded amount of context and yet still could be implemented in an efficient manner. In this same paper he foreshadowed the concept of "non-canonical parsing" or parsing in which reductions can be made at any place in the input string instead of at the leftmost place where a reduction is possible. This idea of non-canonical parsing was developed by both Colmerauer [Co 70] and Williams [Wi 69] and shown to be a useful addition to the "bag of tricks" used to construct parsers.

In Chapter 2 we will extend the LR(k) method to incorporate non-canonical parsing. The technique so developed (the LR(k,∞) method) will still be within the framework of the model of Chapter 1 and will provide us with a formalization which will aid us throughout the rest of this work. In the rest of Chapter 2 and all of Chapter 3, we will produce several new parsing techniques which are suggested by both the parsing model of Chapter 1 and by the study of the drawbacks of the LR(k,∞) method.

Finally in Chapter 4 we will investigate several different kinds of relationships between the parsing techniques which we have developed.

0.2 Grammars, Languages and Phrases

In this section we review basic terminology and exhibit our notation. We assume that the reader is familiar with such concepts as strings, formal grammars, formal languages, finite state automata (fsa), push down automata (pda), regular expressions, decidability and undecidability.

We denote a context free grammar (cfg) by a quadruple (V,V_T,P,S) where

V is the vocabulary of the grammar

$V_T \subseteq V$ is the terminal vocabulary

$P \subseteq (V-V_T) \times V^*$ is a finite set of productions

$S \in V-V_T$ is the start symbol

Individual productions of a grammar are written $A_i \rightarrow x_i$ where A_i designates whatever non-terminal is the left side of the $i^{\underline{th}}$ production and x_i designates that string of V^* which is the right side of the $i^{\underline{th}}$ production. We allow empty right sides and refer to such productions as ε-rules. We also use the symbol $\rightarrow$ (or $\underset{G}{\rightarrow}$ when we wish to indicate which grammar is being used) to denote the application of a production to a string. Thus if $\phi=wA_iy$ and $\psi=wx_iy$, we will write $\phi \rightarrow \psi$. An asterisk (*) written above a relation denotes the reflexive and transitive completion of that relation and a plus sign (+) the transitive completion. The special endmarkers $\vdash$ and $\dashv$ are used to delimit the ends of strings. In certain places we will assume multiple copies of the endmarkers, but such usage will always be clear from the context in which it is necessary. Both $\vdash$ and $\dashv$ are considered elements of V_T.

We designate by SF(G) the set of <u>sentential forms</u> derivable according to the grammar G, that is,

$$SF(G) = \{w \in V^* \mid S \overset{*}{\rightarrow} w\}.$$

The <u>language</u> of G, written L(G), is the set of terminal strings derivable by G, that is,

$$L(G) = SF(G) \cap V_T^*.$$

We will occasionally have need to discuss certain restricted forms of derivations. A derivation $S \rightarrow w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n$ is said to be a <u>rightmost derivation</u> if at each step the production being applied is applied to the rightmost non-terminal in the current sentential form w_i. We symbolize a rightmost derivation step by $\underset{rm}{\rightarrow}$. Similarly, leftmost derivations are denoted by $\underset{lm}{\rightarrow}$.

Let us introduce here an example grammar.

G_1: $S \rightarrow LSR$

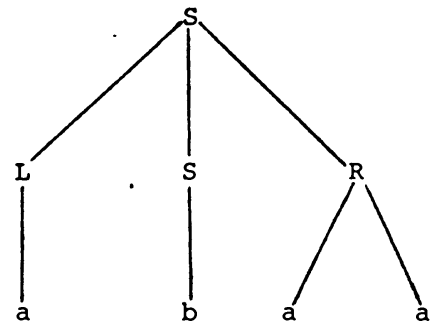
$S \rightarrow b$

$L \rightarrow a$

$R \rightarrow aa$

Thus $L(G_1) = \{a^n ba^{2n} \mid n \geq 0\}$.

We say that x_i is a <u>phrase</u> of a sentential form wx_iy if there is a derivation $S \overset{*}{\rightarrow} wA_iy \rightarrow wx_iy$. Other authors have used the more cumbersome term "simple phrase" to denote this same concept. Clearly, if G is unambiguous, then there is a unique set of phrases corresponding to any sentential form of G. Thus for G_1, the sentential form "abaa" has three phrases, namely (from left to right) "a", "b" and "aa". Phrases can be visualized in terms of the derivation tree corresponding to each sentential form. Phrases then correspond to the sons of a node A such that all sons of A are leaves. Thus for example, the following G_1 tree corresponds to "abaa".



The phrases corresponding to this tree are now clearly defined.

The leftmost phrase of a sentential form is called a <u>handle</u>.

0.3 Parsing

The problem to which this thesis is directed is the problem of determining the structure of strings generated by context free grammars. Suppose that G is a CFG and that $S = w_0 \rightarrow w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n$ is a derivation according to G. This derivation can be represented by n pairs of integers (p_0,ℓ_0), $(p_1,\ell_1),\ldots,(p_{n-1},\ell_{n-1})$ where each pair (p_j,ℓ_j) tells us that w_{j+1} can be produced from w_j by applying production p_j to the non-terminal in location ℓ_j of w_j. The reverse of this sequence is called a generalized bottom-up parse [AU 72]. We distinguish this concept from the term "bottom-up parse" which has come to imply a parse corresponding to a rightmost derivation. From now on we will simply refer to bottom-up parsing when in fact we actually mean generalized bottom-up parsing.

A reduction is the process of taking a grammar G and a sentential form $\phi=X_1\ldots X_n$ and performing the following operations:

1) find two integers p and ℓ such that $x_p = X_{\ell-k+1}\ldots X_{\ell-1}X_\ell$ where $k = |x_p|$.
2) output the pair (p,ℓ).
3) replace the characters $X_{\ell-k+1}\ldots X_\ell$ in ϕ with A_p and renumber the characters of ϕ accordingly.

A reduction sequence is a sequence of pairs (p_1,ℓ_1), $(p_2,\ell_2),\ldots,(p_m,\ell_m)$ obtained by iterating the process of making reductions on ϕ and concatenating the output. Successive instances of ϕ are designated $w_1, w_2, \ldots, w_{m+1}$. A complete reduction sequence is a reduction sequence which reduces a string to the point where no further reductions can be made.

Lemma 0.1 Let $\Pi = (p_1, \ell_1) \ldots (p_m, \ell_m)$ be a complete reduction sequence for some string w_0. Then Π is a bottom-up parse if and only if $w_{m+1} = S$.

Proof: $\Rightarrow$ Since Π is a bottom-up parse, the reversal of Π is a derivation which be definition, starts with the distinguished sentential form S. Hence $w_{m+1} = S$.
$\Leftarrow$ Clearly the reversal of a reduction sequence is a derivation. Since $w_{m+1} = S$, the reversal of Π corresponds to a derivation according to G. But now the reversal of this derivation (i.e. Π) is by definition a bottom-up parse.

q.e.d.

The crucial step in parsing a sentence of some grammar is step 1 of the reduction algorithm, namely finding a "correct" instance of x_i. For example, consider the string ϕ = aabaaaa $\in L(G_1)$. Applying the reduction algorithm indiscriminantly to ϕ can yield such outputs as (3,4), (4,2) or (4.6) which would reduce ϕ respectively to aabLaaa, Rbaaaa or aabaRa. Obviously, none of these strings is what we had intended to produce.

Accordingly, the essential challenge of bottom-up parsing is to select a good "choice function" for use in the reduction algorithm. Traditionally this choice function is implemented through examination of the context of a suspected phrase, that is, the characters immediately adjacent to it. We shall see that the amount and format of the context so used characterize most parsing strategies.

0.4 Description Languages

Although the derivation tree is probably the clearest description of a sentential form, it frequently turns out that the formal manipulation of such trees is a cumbersome process. Accordingly, we will introduce an alternative notation for describing string structure. Suppose we are dealing with a grammar $G = (V, V_T, P, S)$ with p productions. We pick a set B of p new terminal characters and denote the members of B by $]_i$ for $1 \leq i \leq p$. We then define a new grammar.

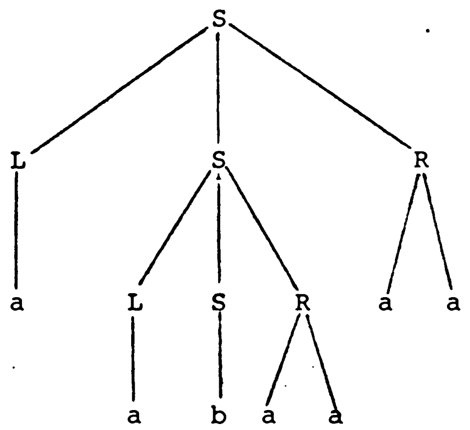
$G' = (V \cup B, V_T \cup B, P', S)$ where now P' also has

p productions and for every production $A_i \rightarrow x_i$ occuring in P, P' contains the production $A_i \rightarrow x_i]_i$.

For example, we now have

$$
\begin{aligned}
G_1': \quad & S \rightarrow LSR]_1 \\
& S \rightarrow b]_2 \\
& L \rightarrow a]_3 \\
& R \rightarrow aa]_4
\end{aligned}
$$

The important point is that every application of a production in a derivation according to G' leaves "its mark" in the string produced. Thus by examining the brackets appearing in any sentential form of G', we can determine exactly its structure. What we have actually done here is to define a linear representation of a derivation tree. Thus corresponding to the G_1 tree appearing below



we have the following string which appears in $L(G_1)$:

$$a]_3 a]_3 b]_2 aa]_4]_1 aa]_4]_1$$

<u>Lemma 0.2</u> If G is an arbitrary context free grammar and G' is formed as described above, then G' is an unambiguous context free grammar.

<u>Proof</u>: G' is clearly BC(0,0) and hence unambiguous.

q.e.d.

We will call SF(G') the <u>description language</u> for G and abbreviate it DL(G). Notice that DL(G) includes bracketed representations not only for every terminal string of G but also for every sentential form of G. Hence we call the elements of DL(G) the <u>bracketed sentential forms</u> of G.

In this dissertation, we will have frequent need to convert back and forth between sentential forms and bracketed sentential forms. We will do this by an erasing mapping $m: V \cup B \to V$ defined by

$$m(X) = X \text{ if } X \in V$$
$$= \varepsilon \text{ if } X \in B$$

Notice that m is a homomorphism with respect to concatenation. Accordingly we will henceforth treat m as being defined on $(V \cup B)^*$. We will also need to consider the image of sets of strings under m and will write m(S) to denote such an image. It will always be clear from the context whether the argument to m is a string or set of strings.

In bottom-up parsing, we are actually concerned only with those places in a string where a phrase occurs and care not about the structure of the derivation tree on higher levels. Thus, in parsing the string ϕ = aabaaaa of G_1 we are not concerned about the <u>complete structure</u> of ϕ which is the string $\phi' = a]_3a]_3b]_2aa]_4]_1aa]_4]_1 \in DL(G)$, but rather with the <u>phrase structure</u> of ϕ which is simply the string $\phi'' = a]_3a]_3b]_2aa]_4aa]_4$. Notice that ϕ'' can be easily produced from ϕ' by erasing those brackets which do not indicate phrases (i.e. the right hand side of the production is not fully reduced, or in tree terms, the sons of the node in question are not all leaves).

To facilitate this sort of construction we will define a new context free language in which brackets will designate exactly those places where phrases occur in the sentential forms of some grammar G. This language will be called the <u>phrase language of G</u> and written PL(G). It is a simple matter to define PL(G) in terms of a generalized sequential machine g applied to DL(G). In particular, g erases all brackets in its input string except for those which immediately follow the characters of the right hand side of the corresponding production. We can also define PL(G) grammatically, which we do below:

Suppose $G = (V, V_T, P, S)$ has p productions. Let $\hat{V} = \{\hat{A} \mid A \in V\}$ and $B = \{]_i \mid 1 \leq i \leq p\}$ be sets of new symbols. We define $\hat{G} = (V \cup \hat{V} \cup B,\ V \cup B,\ \hat{P},\ \hat{S})$ where $\hat{P}$ includes all productions of the form

$$\hat{A}_i \to x_i]_i$$

and of the form

$$\hat{A}_i \to \hat{x}_i$$

where $\hat{x}_i$ is the same as x_i except at least one character has been given a "hat" whenever $A_i \to x_i$ was a production in P.

Then we define $PL(G) = L(\hat{G}) \cup \{S]_0\}$.

Note carefully that we have included the distinguished sentential form in a special way. For example,

$\hat{G}_1$: $\hat{S} \to LSR]_1$

$$\hat{S} \rightarrow \hat{L}SR \mid L\hat{S}R \mid LS\hat{R}$$

$$\hat{S} \rightarrow L\hat{S}\hat{R} \mid \hat{L}S\hat{R} \mid \hat{L}\hat{S}R$$

$$\hat{S} \rightarrow \hat{L}\hat{S}\hat{R}$$

$$\hat{S} \rightarrow b]_2 \mid \hat{b}$$

$$\hat{L} \rightarrow a]_3 \mid \hat{a}$$

$$\hat{R} \rightarrow aa]_4 \mid \hat{a}a \mid a\hat{a} \mid \hat{a}\hat{a}$$

$PL(G_1)$ thus contains, for example, the desired string $a]_3a]_3b]_2aa]_4aa]_4$ where now each bracket designates a phrase.

The reader should verify that the "bracket eraser" m has the following effect.

$$SF(G) = m(DL(G)) = m(PL(G)).$$

It is also true that G is ambiguous if and only if m applied to PL(G) is not a one-to-one mapping.

0.5 Miscellaneous Notation

In this section we present a smorgasboard of notational devices which will be of use to us in various parts of this dissertation.

We record here our conventions (which are slightly non-standard) for naming strings:

$a,b,c,d,\ldots$	are terminal characters
A,C,D,E,F,S	are non-terminal characters
$U,V,\ldots,Z$	are either terminal or non-terminal characters
$\alpha,\beta,\gamma,\ldots$	are elements of $(V \cup B)^*$
$u,v,\ldots,z$	are elements of V^*

B will never be used to denote a character or string because its use is reserved for denoting the set of brackets corresponding to some grammar. In addition to the above, we will frequently use ϕ and ψ to be either sentential forms or bracketed sentential forms.

The notation $|\alpha|$ denotes the number of characters in the string α whereas the function $\mathrm{TRUNC}_k(\alpha)$ yields the first k characters of α, that is,

$$\begin{aligned}\mathrm{TRUNC}_k(\alpha) &= \alpha \text{ if } |\alpha| \leq k \\ &= \beta \text{ if } \exists\beta,\gamma \text{ such that } \alpha = \beta\gamma \text{ and } |\alpha| = k.\end{aligned}$$

The function $\mathrm{NEXT}_k(A)$ yields a set of k letter strings each of which can follow the non-terminal A in some sentential form. More specifically,

$$\mathrm{NEXT}_k(A) = \{y \in V^k \mid \exists w,z \text{ st. } S \overset{*}{\Rightarrow} wAyz\}.$$

Frequently we will want those elements of $\mathrm{NEXT}_k(A)$ which occur in derivation trees which are fully "pruned" to the right of A, that is, those k letter strings which represent fully reduced right contexts for the non-terminal A. Accordingly, we offer as our next definition:

$$\mathrm{REDNEXT}_k(A) = \{y \in V^k \mid \exists w,z \text{ st. } S \underset{lm}{\overset{*}{\Rightarrow}} wAyz\}.$$

In places we will need to introduce a special marker into strings. This marker is written # and is always taken to be disjoint from the vocabulary or brackets for any grammar under consideration. The map $h\colon V \cup B \cup \{\#\} \to V \cup B$ defined by

$$\begin{aligned} h(X) &= X \text{ if } X \in V \cup B \\ &= \varepsilon \text{ if } X = \# \end{aligned}$$

is a homomorphism with respect to concatenation and will be used to insert or delete the special marker.

Finally, let us warn the reader of a possible source of confusion in our notation. We frequently will use expressions such as LR(k) and BRC(m,n) which involve parameters. We will almost always intend these parameters to be free in the sense of standing for all possible applicable values. Thus when we refer to "the class of LR(k) grammars", we actually mean the class of all grammars which are LR(k) for some value of k, or symbolically, $\{G \mid \exists k \text{ st. } G \text{ is } LR(k)\}$. When we intend the parameters to be bound to a specific value, we will state explicitly that the parameters in question are fixed.

Chapter 1 PARSING WITH PATTERNS

1.1 Finding Phrases with Patterns

In this section we will develop sufficient groundwork for the presentation of a decision procedure for the correctness of bottom-up parsers. Such parsers can be characterized by the set of strings, or contexts, surrounding those occurences of phrases which are reduced by the parser. For example, a Bounded Context parsing machine as envisioned by Floyd [Fl 64] can be viewed as a collection of pairs of finite strings, each pair of which corresponds to an acceptable context for the reduction of some phrase. Similarly, an LR(k) parsing machine can be viewed as a collection of pairs of regular sets and finite strings, each pair again specifying an acceptable context for the reduction of the leftmost phrase of a right sentential form. We now state a few definitions to make this concept more precise.

A <u>reduction pattern</u> for the <u>ith</u> production of some grammar G is a pair (R,i) such that $R \subseteq V^* x_i \# V^*$. Such a pattern is said to <u>apply</u> to a string ϕ if ϕ is of the form $\ldots w x_i y \ldots$ and $w x_i \# y \in R$. Any such application induces a <u>reduction</u> on ϕ which is a new string ϕ' of the form $\ldots w A_i y \ldots$ where the indicated occurences of w and y in ϕ' are the same occurences as indicated in ϕ.

Certain formats of patterns occurs frequently enough to warrant special names. Many existing parsing methods for example, restrict themselves to some fixed amount of lookahead (or lookbehind) whereas others allow complete knowledge of whatever amount of the input string has already been read.

We say that a reduction pattern (R,i) is <u>k bounded on the left</u> (for some integer k) if $wx_i\#y \in R$ implies $|w| \leq k$. (R,i) is said to be <u>spanning on the left</u> if $wx_i\#y \in R$ implies $w \in \vdash V^*$. We similarly define right bounded and right spanning patterns.

In order to be able to utilize a reduction pattern for parsing, we must observe the following condition. In any application of a pattern to sentential form, it must always be the case that in every possible bracketing of that sentential form, the instance of x_i discovered by the pattern must actually be an immediate A_i derivative. In other words, the reduction pattern should avoid making reductions of things which might not be phrases. We formalize this requirement as follows.

A <u>parsing pattern</u> is a reduction pattern (R,i) such that if $\phi = vwx_iyz \in SF(G)$ and $wx_i\#y \in R$ then

$$\forall\psi \in m^{-1}(\phi) \cap PL(G)\ ,\ \psi \text{ can be written } \psi_1\psi_2x_i]_i\psi_3\psi_4$$

$$\text{where } m(\psi_1) = v$$
$$m(\psi_2) = w$$
$$m(\psi_3) = y$$
$$m(\psi_4) = z.$$

Let us pause for a moment and consider some examples.

Let G_2 be defined by

$$S \rightarrow LSR \mid M \mid N$$

$$L \rightarrow a$$

$$R \rightarrow aa$$

$$M \rightarrow b$$

$$N \rightarrow b$$

Consider the reduction pattern ({a#b},4). This pattern is a parsing pattern because the center of any sentential form of G_2 is "marked" by an occurence of either S,M,N or b. Since <u>any</u> occurence of an a to the left of center is an immediate L derivative, the reduction pattern always functions correctly.

Similarly, the pattern ({a#L},4) is a parsing pattern because an L can occur only in the left portion of a sentential form and therefore any instance of an a still further to the left must also be an immediate L derivative.

Consider now the pattern ({aa#},5). Here we are attempting to reduce the production R → aa without the benefit of any context. As expected, many things can go wrong. This pattern can apply to the left side of a sentential form as well as to the right side. We would certainly not want to reduce aabaaaa to Rbaaaa. We could even misapply the pattern to the right side to produce aabaRa. This last pattern failed because it could be applied in the wrong place. A pattern can also fail even if it is applied in the correct place! For instance, the pattern ({b#},6) always locates a character which is a phrase but this phrase could be either an M derivative or an N derivative. For instance the string abaa has two possible phrase bracketings which reflect this difficulty, namely $a]_4b]_6aa]_5$ and $a]_4b]_7aa]_5$. It is through detection of this sort of trouble that we will be able to guarantee the unambiguity of the grammar for any parser which we decide is correct.

In order to transform the definition of parsing patterns into a more tractable form, it is necessary to examine the ways in which a reduction pattern can fail to be a parsing pattern. In Floyd [Fl 64] and Williams [Wi 69] this was done in a somewhat more restricted setting by examining a series of 16 relations on strings. We will take an alternate approach.

Observe from the definition that a reduction pattern can fail to be a parsing pattern only if some application of the pattern to a sentential form locates an occurence of x_i such that in some phrase bracketing of that sentential form, that instance of x_i is not immediately followed by a $]_i$. In the case of ε-rules, which have a null x_i, a mistake has been made if no instance of $]_i$ occurs anywhere between the rightmost character of left context and the leftmost character of right context.

We will illustrate these ideas by showing that ({a#a},4) is not a parsing pattern for G_2. Our starting point will be the set $PL(G_2)$ which of course tells us where the phrases occur in any sentential form. Next we insert a # in each string of $PL(G_2)$ to which the pattern in question applies. We do this by computing the set

$$S = h^{-1}(PL(G_2)) \cap m^{-1}(V^*a\#aV^*).$$

This intersection provides us with a set of bracketed strings, each of which contains exactly one #. Furthermore, the # is always located at a place where the pattern applies. Since the above process is performed in all possible ways, we are left with a representation of every way in which the pattern could be applied to any sentential form. In particular, S contains such strings as

$$a\#]_4a]_4b]_6aa]_5aa]_5$$
$$a]_4a]_4b]_6a\#a]_5aa]_5$$
$$a]_4a]_4b]_6aa\#]_5aa]_5$$

The first string listed above certainly corresponds to a valid reduction (i.e. reducing aabaaaa to Labaaaa) but the other two do not. If we now compute the set

$$S' = S - (V \cup B)^*V\#]_4(V \cup B)^*$$

we will discard all instances of correct reductions. S' thus tells us exactly how the pattern ({a#a},4) can make errors. More specifically, S' tells us that the pattern should not be applied at either position 4 or 5 of the string aabaaaa.

We can recast these ideas to produce a test for whether an arbitrary reduction pattern is a parsing pattern.

Theorem 1.1 (R,i) is a parsing pattern for G if and only if

$$(*) \quad \left[h^{-1}(PL(G))\right] \cap \left[(V \cup B)^* m^{-1}(R)(V \cup B)^*\right] \cap \left[(V \cup B)^* V\#(B -]_i)^* V(V \cup B)^*\right]$$

is empty.

Proof: Assume that no pattern begins or ends with a # (i.e. we always use at least one character of context). We will prove the contrapositives, referring informally to (*) as the "set of mistakes".

1) Assume that (R,i) is not a parsing pattern. Then there exists a sentential form ϕ and a phrase bracketing ψ which cannot be written in the desired format. Since ψ uniquely determines ϕ, we can forget about ϕ and concentrate on the structure of ψ. Let V_1 and V_2 be the characters immediately adjacent to the # in the matching pattern string. Hence there exist

$$\alpha_1, \alpha_2, \gamma_1, \gamma_2 \in (V \cup B)^*$$
$$V_1, V_2 \in V$$
$$\beta \in B^*$$

such that $\psi = \alpha_1\alpha_2V_1\beta V_2\gamma_1\gamma_2 \in PL(G)$ and $m(\alpha_2V_1\#V_2\gamma_1) \in R$.

By hypothesis, ψ cannot be written in the desired way. If x_i is non-empty then $]_i$ cannot be the initial character of β since we could then take $\psi_1 = \alpha_1$, $\psi_2 x_i = \alpha_2 V_1$, $\psi_3 = \beta' V_2 \gamma_1$ and $\psi_4 = \gamma_2$ (where β' is the rest of β) to produce the desired format. Similarly, if production i is an ε-rule, we can conclude that $]_i$ occurs nowhere in β. In either case we have $\beta \in (B-]_i)^*$. But clearly then

$$\begin{aligned} \alpha_1\alpha_2 V_1 \# \beta V_2 \gamma_1 \gamma_2 &\in h^{-1}(PL(G)) \\ '' \qquad\qquad '' &\in (V \cup B)^* m^{-1}(R)(V \cup B)^* \\ '' \qquad\qquad '' &\in (V \cup B)^* V\#(B-]_i)^*(V \cup B)^* \end{aligned}$$

Hence $\alpha_1\alpha_2 V_1 \# \beta V_2 \gamma_1 \gamma_2 \in (*)$ and the set of mistakes is non-empty.

2) Assume that (*) is non-empty. Let ψ' be any string in (*). ψ' can then be written as $\alpha_1\alpha_2 V_1 \# \beta V_2 \gamma_1 \gamma_2$ where $\alpha_1, \alpha_2, \gamma_1, \gamma_2 \in (V \cup B)^*$

$$V_1, V_2 \in V$$
$$\beta \in (B -]_i)^*$$

and $m(\alpha_2 V_1 \# \beta V_2 \gamma_1) \in R$.

Clearly then, $\psi = \alpha_1\alpha_2 V_1 \beta V_2 \gamma_1 \gamma_2 \in PL(G)$ because $\psi' \in (*)$. However since R obviously applies to $m(\psi)$ and β contains no $]_i$, we cannot partition ψ as required. Hence R is not a parsing pattern.

q.e.d.

<u>Corollary 1.2</u> It is decidable whether a regular reduction pattern is a parsing pattern for G.

Proof: The set (*) is context free if R is regular and the emptiness problem for context free languages is well known to be decidable.

q.e.d.

Although we can decide the correctness of a regular pattern used for finding phrases, we shall see in Chapter 2 that this problem is undecidable for more general sets of patterns.

1.2 Parsing with Patterns

A single parsing pattern allows us to reduce only one production of the grammar in question. Accordingly, we need a collection of parsing patterns, one for each production, before we can start parsing. Another requirement is even more important. It must be the case that if we are given any sentential form which arises during parsing, we must have some pattern available which applies to it. Since the set of sentential forms which arise during parsing is not necessarily a cleanly structured set, we will simply require that our parsing methods be powerful enough to find a phrase in <u>any</u> sentential form.

A <u>parsing scheme</u> for a grammar G is a finite collection of reduction patterns such that

1) each reduction pattern is a parsing pattern,
2) for every sentential form of G other than $\vdash S \dashv$, there exists some pattern in the collection which applies to it.

A parsing scheme is potentially a very powerful device as demonstrated by

<u>Theorem 1.3</u> There exists a parsing scheme for G if and only if G is unambiguous.

<u>Proof</u>: $\Rightarrow$ Suppose G is ambiguous and S is a parsing scheme for G. Let $A = \{(\phi,\psi) \mid \phi \neq \psi,\ \phi,\ \psi \in DL(G)$ and $m(\phi) = m(\psi)\}$. Thus A is simply the set of pairs of

different bracketings for the same sentential form. Let (ϕ', ψ') be a pair in A such that the number of brackets in $\phi'\psi'$ is a minimum over A. Clearly ϕ' and ψ' have no common phrase. But now some pattern will apply to $m(\phi')$ and whatever phrase it purports to find in ϕ' will be wrong in ψ'. Hence the pattern used cannot be a parsing pattern.

$\Leftarrow$ Suppose that G is unambiguous. Then every sentential form has a unique phrase bracketing and the entire string on either side of any phrase will form a parsing pattern for that phrase. Thus is x_i is the indicated phrase of $wx_iy \in SF(G)$ then $wx_i\#y$ is certainly a parsing pattern for the ith production of G. All possible parsing patterns so generated form a context free language which can easily be derived from G. Simply take the grammar for SF(G), add the production $A_i \rightarrow x_i\#$ and intersect the resulting language with $V^*\#V^*$.

q.e.d.

Corollary 1.4 It is undecidable whether a parsing scheme for G exists.

Proof: Such a test is equivalent (by Theorem 1.3) to testing G for ambiguity, a problem well known to be unsolvable.

Lemma 1.5 Let G be a context free grammar with p productions. Let $P = \{(R_1,1),\ldots,(R_p,p)\}$ be a collection of reduction patterns. Then P is a parsing scheme for G if and only if

1) each (R_i, i) is a parsing pattern for G.

2) $SF(G) - \{\vdash S \dashv\} \subseteq h(\bigcup_{1 \le i \le p} V^* R_i V^*)$

Proof: The above conditions are simply a rewording of the definition of parsing schemes. The V^*'s are used to "fill out" the patterns to match the entire string being parsed and the homomorphism h, as always, removes the # from the pattern strings.

q.e.d.

Theorem 1.6 [The Correctness Problem for Bottom-up Parsers]
Let G be a context free grammar and P be a finite collection of regular reduction patterns. Then it is decidable whether P is a parsing scheme for G.

Proof: We must test the two conditions mentioned in Lemma 1.5. Corollary 1.2 tells us that condition 1 is decidable. Condition 2 on the other hand, is simply a test for the containment of a context free language in a regular language, a problem well known to be decidable.

q.e.d.

This practical restriction to regular parsing patterns is not a serious restriction. As a matter of fact, most existing parsing methods utilize principles which correspond to the application of regular patterns. We will elaborate on this idea in section 1.3.

So far, we have used the entire set of sentential forms as the "domain of discourse" for our parsers. We have used this set because it is closed under any reductions made by a correct parser. That is, if a phrase is found in a sentential form and we replace that phrase with the left hand side of the corresponding production, then the resulting string is also a sentential form. This fact guarantees that we can iterate the application of parsing patterns until we either arrive at the distinguished sentential form $\vdash S \dashv$ or we arrive at some string to which no pattern applies. In the former case we have successfully found a parse corresponding to the input string and in the latter case, we can conclude that the input string was not a valid sentential form. We prove this fact explicitly in the next lemma, which guarantees that a parsing scheme can never make an error such as rejecting a valid sentence or accepting a non-sentence. In particular, the lemma implies that if at some stage in the parsing process we have a choice of several ways to apply patterns to the input string, we need never fear that the "wrong choice" will lead to disaster.

<u>Lemma 1.7</u> Let G be a context free grammar and P be a parsing scheme for G. Let Π be a complete reduction sequence induced by P on some $w \in V^*$. Then the following are equivalent:

1) $w \in L(G)$
2) the fully reduced string ϕ corresponding to Π is $\vdash S \dashv$
3) Π is a bottom-up parse.

Proof: We first observe that a single application of a parsing pattern always reduces a sentential form to another sentential form and a non-sentential form to a non-sentential form. By induction on the length of Π, ϕ must therefore be a sentential form if and only if w is a sentential form. However, the only sentential form to which no pattern applies is $\vdash S \dashv$. Thus w is a sentential form if and only if ϕ is $\vdash S \dashv$. Hence 1) and 2) are equivalent. The equivalence of 2) and 3) on the other hand follows from Lemma 0.1.

q.e.d.

1.3 A General Model of Bottom-up Parsing

The test of Theorem 1.1 makes no use of any "preferred order" of application of reduction patterns. Such a preferred order is in fact used by most parsing methods because of their intrinsic left to right nature.

Suppose that $S = \{(R_i, i) \mid 1 \leq i \leq p\}$ and $\hat{S} = \{(\hat{R}_i, i) \mid 1 \leq i \leq p\}$ are sets of reduction patterns for grammar G. We say that $\hat{S}$ is a set of <u>leftmost patterns corresponding to</u> S if whenever $\hat{R}_i$ applies to $w \in V^*$ at position n then

1) R_i applies to w at position n

2) no R_j applies to w at any position < n.

In other words, $\hat{S}$ performs the reductions of S in a strictly leftmost possible fashion. The reader may easily verify that if S is a regular set of patterns, so too is $\hat{S}$ and furthermore that $\hat{S}$ can be easily constructed from S by simple manipulations performed on the state diagrams for S.

We will now indicate a more general approach which could have been used in the development of Theorems 1.1 and 1.6. We have required our methods to be able to find a phrase in <u>any</u> sentential form. An alternative approach would restrict itself to some subset of all the sentential forms. For example, the formal derivation of the LR(k) technique concerns itself only with right sentential forms as does the BRC technique. Accordingly we will indicate how this more general approach proceeds.

We must first select some subset D of sentential forms as the domain of the parser in question. We then insert indexed brackets around all phrases in D which we wish to consider as candidates for reduction. Let us call the resulting set D'. If we now utilize D' instead of PL(G) in Theorem 1.1, we have a procedure for verifying that the parser never makes a reduction which is not explicitly permitted by D'. In particular, we could select as D' the set of right sentential forms with one bracket per string designating a handle of that string. The test of Theorem 1.1 will now decide whether the parsing machine will ever incorrectly reduce a handle or "miss" a handle by reducing something to its right. Moreover, the test will preclude the possibility of alternate phrase bracketings existing in D' for the same string in D.

A second requirement must be made of D, namely D must be closed under the reductions induced by the parser. Thus if $\phi \in D$ is reduced to ϕ', then ϕ' must also be in D. This closure property guarantees that we can iterate the reduction process until a complete parse is produced.

Again the set of right sentential forms satisfies this requirement if we assure that only the leftmost phrase (i.e. handle) is ever reduced. This is true because pruning the leftmost phrase of a right sentential form yields another right sentential form.

Finally, we must make sure that every element of D is reducible by the purported parser. We do this by performing the second test in Lemma 1.5 using the set D instead of SF(G).

A certain amount of discretion is necessary in making these tests because they might very well involve solving unsolvable problems.

By varying the subset of sentential forms and the format of the parsing patterns, it is possible to generate many of the existing parsing methods and furthermore, to suggest other methods which have not yet been studied.

Suppose that we choose the desired subset of sentential forms to be the set of right sentential forms and restrict ourselves to patterns which are simultaneously both right and left bounded. If we now test the correctness of the set of leftmost patterns corresponding to our original patterns, then we have a decision procedure for the correctness of Bounded Right Context (BRC) parsers. This class was first studied by Floyd [Fl 64].

Again, if we look at right sentential forms and allow our parsing patterns to be arbitrary right bounded sets, we arrive at a model of the LR(k) parsers. This class was first investigated by Knuth [Kn 65] who showed that the patterns used are in fact regular sets. Continuing with the set of right sentential forms, if we allow arbitrary left context sets and require right context sets to be regular, we obtain a model of the LRR grammars of Cohen and Čulík [CC 71]. We point out here that in this case the left context sets once again will turn out to be regular.

If we mark <u>all</u> phrases in <u>all</u> sentential forms with brackets then we have the set PL(G) as described earlier.

Since m(PL(G)) = SF(G), we have the desired property of closure under reductions. Restricting ourselves to patterns which are simultaneously right and left bounded yields the class of Bounded Context Parsable (BCP) grammars which was studied by Williams [Wi 69].

Knuth [Kn 65] has suggested a class of grammars in which we have the option of bypassing a "questionable" phrase and reducing a phrase further to the right. He suggested, however, that this decision postponement be restricted to occur some fixed number of times before a positive decision be made. He called the class of grammars parsable by this method the LR(k,t) grammars. In this class we require that in any sentential form, it is possible to distinguish one of the leftmost t phrases purely on the basis of its left context (all the way back to the start of the string) and the first k characters of the right context.

We can generate a decision procedure for this class by building a special set L(t)PL(G) in which only the <u>l</u>eftmost <u>t</u> <u>p</u>hrases of the <u>l</u>anguage of <u>G</u> are delimited by brackets. Restricting parsing patterns to be k bounded on the right yields a decision procedure for the LR(k,t) parsers. As this class of grammars has not yet been studied, we investigate it in detail in Sections 2.5 and 2.6. In particular we will show that the parsing patterns for an LR(k,t) grammar are always regular sets and furthermore, that a parser for such a grammar can always be implemented on a DPDA.

Additional classes of parsers will arise naturally as we continue to alter the parameters. If we allow ourselves the luxury of finding any phrase in any sentential form and at the same time restrict ourselves to regular parsing patterns which are k right bounded, we come up with a class which we will call FPFAP(k), that is, the class of grammars for which a finite phrase finding automaton can find a phrase in any sentential form by using a left to right scan with k characters of lookahead.

If we allow the parsing patterns to be arbitrary sets but maintain the bounded right context requirement, we arrive at a class in which some phrase can be found in any sentential form purely by examination of its left context and first k characters of right context. Such a class is the natural generalization of the LR(k,t) concept and will accordingly be called LR(k,∞). This class will be studied extensively in Chapter 2.

Most of the above methods have involved bounded right context. Our decision procedure has no such intrinsic restriction. It will work equally well, for example, on parsing patterns which are simply regular sets, perhaps extending on both sides to the ends of the string. This class will be called RPP, that is, regular pattern parsable. We do not, however, study it extensively here because it does not lend itself cleanly to modeling by traditional automata.

1.4 Implementing Parsing Patterns

So far in this thesis, we have concentrated our attention on the development of classes of patterns which can be used to find a single phrase in a sentential form. In this section we will be concerned with the process of iterating these reductions to produce a complete parse. In order to discuss the speed of these parsers, it is necessary to be able to place a bound on the number of reductions necessary to produce a complete parse. Equivalently we show

Lemma 1.8 Let G be an unambiguous context free grammar with p non-terminals and q = length of the longest right hand side. Let $\phi \in SF(G)$ with $|\phi| = n$. Then $S \overset{*}{\rightarrow} \phi$ in at most $p(1 + q^{p+1})n$ steps.

Proof: Examine the (unique) derivation tree corresponding to ϕ. Mark all nodes of the tree which include a terminal among their descendants. Call this set of marked nodes the "core" of the tree. Some nodes of the core have two or more direct descendants which are marked. However, there can never occur a sequence of p+1 nodes, each being the sole marked son of its father, for then such a sequence would contain a repetition of some non-terminal. The portion of the core between the repeated occurences could then be deleted to produce an alternate derivation tree for ϕ in contradiction to our assumption that G is unambiguous. Thus, the core contains at most pn nodes.

Since each node represents the application of a production, we can have at most pqn nodes which are direct descendants of nodes in the core. Some of these must be gotten rid of by application of ε-rules.

Clearly, however, the height of the subtree derived from such a node can be at most p because otherwise we could derive the empty string in more than one way. Thus each such subtree contains at most q^p nodes. Thus an upper bound on the number of productions used to derive ϕ is $pn + pqn(q^p)$ which is linear in n.

q.e.d.

Thus it is that our parsing algorithms will make at most a linear number of reductions with respect to the length of the input string.

A certain class of automata will prove valuable to us as a model of the parsing process. Such an automaton has a finite control and two pushdown stores. The next move function depends on the top symbol of each stack and the state of the finite control. At each step, the machine can pop one symbol off either stack or push any symbol onto either stack. At each step, the machine can, in addition, change the state of its internal control. Initially, the finite control is in some designated start state, stack one contains a distinguished "bottom marker" and stack two contains the input string which is also delimited with a special "bottom marker". Such a device will be called a <u>two push</u>

down store automaton (2PDA). No formal definition will be given beyond what has already been said.

Lemma 1.9 Let G be a CFG and S be a parsing scheme for G, all of whose patterns are regular sets. Then S can be implemented on a 2PDA. Furthermore, such an implementation runs in time $O(n^2)$ where n is the length of the input.

Proof: Suppose (P,i) is one of the regular parsing patterns of S. Then P can be written as some finite number of subpatterns $L_1\#R_1,\ldots,L_k\#R_k$ where k is the number of states necessary to recognize P and each of the sets L_i and R_i is regular. More specifically, suppose that $M = (\{q_1,\ldots,q_k\},V,\delta,q_1,F)$ is the finite automaton which recognizes P. Then

$$L_i = \{w|\delta(q_1,w) = q_i\}$$
$$R_i = \{w|\delta(q_i,\#w) \in F\}$$

We now build M', a 2PDA to implement S. M' does the following process, in parallel for each (P,i). M' reads the input from left to right one symbol at a time. At each step M' records (in its other stack) which of the L_i sets the current initial segment of the input is in. At the end of this first pass, M' changes direction and tries to recognize the reverse of the sets R_i. M' proceeds from right to left in this manner until it comes to a cell for which the string to its left is in L_i and the string to its right is in R_i (for the same value of i). At this point M' has determined that P is applicable to its input string and furthermore the current position of the machine corresponds to the position of the "#" in the pattern which matches the input. Hence, M' can make the reduction implied by P by popping off the symbols corresponding to the right hand side of the production in question and replacing the missing symbols with the left hand side of this production. M' then moves all symbols back into one stack and essentially starts over. Each such reduction clearly takes an amount of time bounded by 2n, where n is the length of the input.

Since S is a parsing scheme for G we know by Theorem 1.3 that G is unambiguous. Thus Lemma 1.8 applies and the number of reductions necessary to parse a string is O(n). Since each reduction takes O(n) time, the entire parsing process takes at most $O(n^2)$ time.

q.e.d.

Unfortunately, we are not really interested in $O(n^2)$ parsing schemes but rather in techniques which run quickly. Accordingly, we will concentrate on methods which can produce a complete parse in linear time. These methods will usually involve a left-to-right scan of the input string, using a fixed, finite amount of lookahead.

This process can be modeled with a k right bounded parsing scheme. In particular, the process of matching these patterns to the input string can be viewed most conveniently by considering a phrase finding automaton (PFA). Such a device will always be able to find a phrase in any sentential form by scanning it from left to right using k characters of lookahead. More specifically, a PFA has a set K of states and a transition function δ: $K \times V^k \rightarrow K$. We will permit the set K to be infinite because this will allow us to model some interesting parsing algorithms. One of the distinguishing features of "practical" parsers then will be their utilization of a finite state set.

A single move is made as follows: first, the leading character of the input string is deleted. Then the next k characters of input are examined and a transition is made to a new state based on the current state and the k character segment of input. If the new state is a terminal state, the machine halts and indicates that a reduction should be made at the current position of the input.

If the new state is not a terminal state, then the move cycle is iterated until a terminal state is reached. We will represent terminal states in diagrams by #i where the i is an integer designating the production to be reduced. Non-terminal states will be designated i where the i is the name or number of the state. Transitions will be represented by arrows between states, each arrow being labeled with the appropriate k letter string.

The input string will always be padded to the left with k "⊢"s and to the right with k "⊣"s. We will now give an example of a grammar and its corresponding PFA.

G_3:

$$
\begin{aligned}
S &\rightarrow Aa \mid Bb \\
A &\rightarrow dA\bar{A} \mid c \\
B &\rightarrow dB\bar{B} \mid c \\
\bar{A} &\rightarrow d \\
\bar{B} &\rightarrow dd
\end{aligned}
$$

Machine M in Figure 1.1, as the reader may verify, is a PFA for G_3 using a value of $k = 1$.

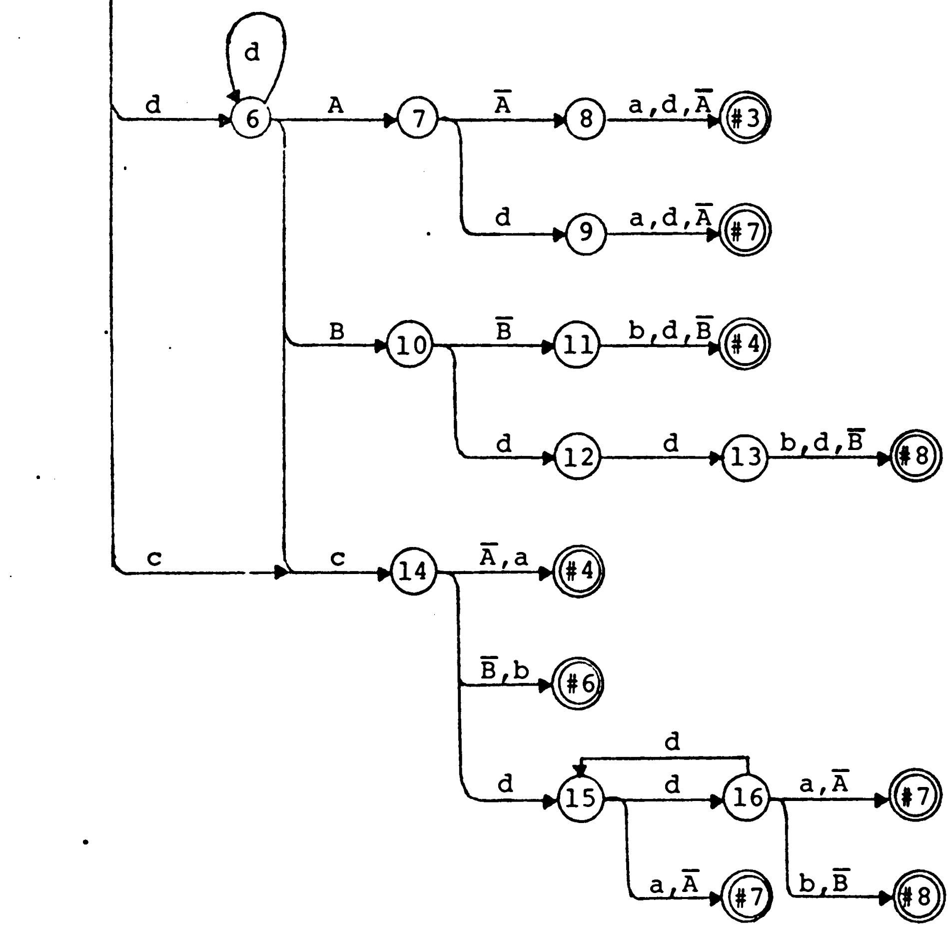


Figure 1.1 A PFA for grammar G_3

Thus the input string w = ddcdda, when padded out to ⊢ddcdda⊣ causes M to go through the following sequence of states: 1,6,6,14,15,16,#7 which indicates that the last d of w should be reduced according to production 7. Thus w becomes ddcd$\bar{A}$a.

Note carefully that the operation of the PFA involves a state transition made <u>before</u> the shift over the next input character is made. Thus we never have to "back up" to get to the phrase being reduced. Note further that we will never use a value of k = 0 in our parsing methods. This is because all of our parsing methods degenerate to the LR(0) method when this value is chosen for k. For larger values of k, our methods will turn out to be much more powerful than the corresponding LR(k) methods.

Suppose then that M is a PFA for G and that we apply M to some sentential form ϕ. Let us assume that $\phi = vwx_iy$ where x_i is the phrase found by M and $|w| = k$. Thus we reduce ϕ to $\psi = vwA_iy$. Notice that if we now apply M to ψ, M will not detect any phrases in v because the context of nothing in v has changed. Thus when M makes a reduction, we need only back up k characters (to the beginning of w) before resuming our left to right scan. Furthermore, this backing up can be done very easily using a stack containing the characters which we have already read and the states that M was in at the time when it read each of these characters.

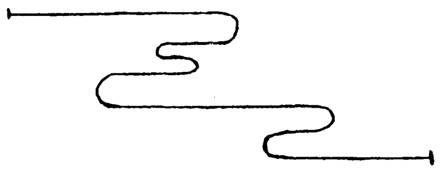
These ideas lead us to

<u>Theorem 1.10</u> Let G be a CFG and M be a PFA for G. Then we can construct a 2PDA M' which parses G in linear time.

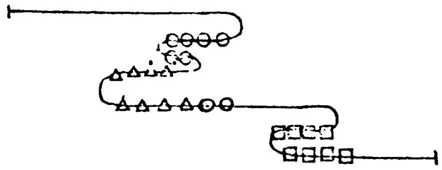
<u>Proof</u>: The essential ideas have already been described. M' starts with its input in stack two. First M' pops off the first k characters of input from stack two and stores them in a buffer in its finite control. This buffer will always represent the current status of the lookahead string. M' now simulates M. At each step M' stacks in stack one, the state of M and the oldest character in its buffer. The buffer is then refilled from stack two.

Eventually M will signal that a reduction is to be made. This is done by popping off the characters of the right hand side of the indicated production from stack one and replacing them with the left hand side of the production. M' then backs up by storing the buffer contents back in stack two and refilling the buffer from stack one. At the same time, the appropriate state of M can be recovered from stack one and we can resume our left to right scan. Thus it is that the concatenated contents of stacks one and two always represent a valid sentential form.

We must show that M' runs in linear time. We do this by noting that whenever M' makes a reduction and backs up k steps, it must at some time in the future return to the current point of the scan. Accordingly, we can charge each reduction with 2k units of work, k for backing up and k for returning to the current point. Assume that the position of the parsing machine varies on its input as shown below.



Then we indicate how these "backtracks" are charged by covering the segments corresponding to each reduction, with similar symbols.



Thus the total time used by M' is bounded above by 2k times the number of reductions made, plus the time used for a single left to right pass over the input. By Theorem 1.3 and Lemma 1.8, we conclude that this time is linear with respect to the length of the input.

q.e.d.

We emphasize the importance of the above theorem, for in the rest of this dissertation we will show only how to construct PFA's and rely on Theorem 1.10 to construct a machine for doing the actual parsing of the grammar in question.

Let us consider an example of the construction of 2PDA's from PFA's. We will write configurations of the 2PDA used for parsing as triples, consisting of, respectively, the contents of stack one, the current state of the finite control, and the content of stack two. Stack contents will be written with the tops toward the middle, that is in $\vdash$ABC,3,DEF$\dashv$, C and D are the top symbols of their stacks and the machine is in state 3.

Now let us parse a string of SF(G_3). Suppose the input is ddcdda. Then the machine undergoes the sequence of transitions indicated in Figure 1.2.

$\vdash,\ 1,ddcdda\dashv$

$\vdash 1d,\ 6,dcdda\dashv$

$\vdash 1d6d,\ 6,cdda\dashv$

$\vdash 1d6d6c,14,dda\dashv$

$\vdash 1d6d6c14d,15,da\dashv$

$\vdash 1d6d6c14d15d,16,a\dashv$

$\vdash 1d6d6c14d,15,\bar{A}a\dashv$

$\vdash 1d6d6c,14,\overline{\bar{A}\bar{A}}a\dashv$

$\vdash 1d6d,\ 6,A\overline{\bar{A}\bar{A}}a\dashv$

$\vdash 1d6d6A,\ 7,\overline{\bar{A}\bar{A}}a\dashv$

$\vdash 1d6d6A7\bar{A},\ 8,\bar{A}a\dashv$

$\vdash 1d,\ 6,A\bar{A}a\dashv$

$\vdash 1d6A,\ 7,\bar{A}a\dashv$

$\vdash 1d6A7\bar{A},\ 8,a\dashv$

$\vdash,\ 1,Aa\dashv$

$\vdash 1A,\ 2,a\dashv$

$\vdash 1A2a,\ 3,\dashv$

$\vdash,\ 1,S\dashv$

$\vdash 1S,17,\dashv$

halts

Figure 1.2 Successive configurations of a 2PDA while parsing ddcdda according to the PFA for G_3.

1.5 Limitations of the Model

At this point, it is worthwhile to point out a weakness in the theory being developed. We have required that reduction patterns work correctly on <u>every</u> sentential form to which they can be applied, and that <u>every</u> sentential form be "covered" by some pattern. These requirements are stronger than we would like to use. What is actually needed, is that reduction patterns work correctly on every sentential form which <u>can arise in parsing</u> (i.e. which can be produced from an element of L(G) by applying some sequence of reductions allowed by the set of patterns being used). Unfortunately, such a set need not have a nice structure (indeed we can even recognize non-context-free sets in this manner), a fact of life which diminishes hope of deriving a decision procedure for testing correctness of a parser. However, as we have already seen, there are certain restricted sets of sentential forms (such as right sentential forms) which are context free and which are closed under application of the parsing rules. An example will clarify these points. Consider grammar G_4 and the set of reduction patterns shown below.

G_4:	$S \to ASA$	ASA#
	$S \to BSB$	BSB#
	$S \to aa$	$\vdash (A \cup B)^* aa\#(A \cup B)^* \dashv$
	$S \to bb$	$\vdash (A \cup B)^* bb\#(A \cup B)^* \dashv$

$\bar{A} \rightarrow \bar{A}$ $\quad \vdash (A \cup B)^* A (a \cup b)^+ \bar{A} \# (A \cup B)^* \dashv$

$\qquad \vdash (A \cup B)^* \bar{A} \# (a \cup b)^+ \bar{A} (A \cup B)^* \dashv$

$\bar{A} \rightarrow a$ $\quad \vdash (A \cup B)^* a \# (a \cup b)^+ a (A \cup B)^* \dashv$

$\qquad \vdash (A \cup B)^* \bar{A} (a \cup b)^+ a \# (A \cup B)^* \dashv$

$B \rightarrow \bar{B}$ $\quad (A \cup B)^* B (a \cup b)^+ \bar{B} \# (A \cup B)^*$

$\qquad (A \cup B)^* \bar{B} \# (a \cup b)^+ \bar{B} (A \cup B)^*$

$\bar{B} \rightarrow b$ $\quad (A \cup B)^* b \# (a \cup b)^+ b (A \cup B)^*$

$\qquad (A \cup B)^* \bar{B} (a \cup b)^+ b \# (A \cup B)^*$

The grammar above generates the language $\{ww^r \mid w \in (a \cup b)^+\}$ and the reduction patterns listed will successfully parse any string of $L(G_4)$. The patterns accomplish this task by "pairing off" matching characters at the extreme ends of the string. For example, the string abaaba would be successively reduced as follows:

$abaaba \rightarrow \bar{A}baaba \rightarrow \bar{A}baab\bar{A} \rightarrow Abaab\bar{A} \rightarrow AbaabA \rightarrow A\bar{B}aabA \rightarrow$
$A\bar{B}aa\bar{B}A \rightarrow ABaa\bar{B}A \rightarrow ABaaBA \rightarrow ABSBA \rightarrow ASA \rightarrow S$

This behavior results in a parsing speed of $O(n^2)$ when implemented on a 2PDA (see Lemma 1.9 for the details of the construction). Notice however, that only a certain restricted subset of the sentential forms arises in parsing $L(G_4)$, namely those with essentially equal length strings of A's and B's at the ends of the form. Thus it is that the sentential form ϕ = AAaaaA would be incorrectly reduced by the above set of reduction patterns. However, this should not concern us, since ϕ can never occur by applying reductions to a string in $L(G_4)$.

We thus arrive at what appears to be a crucial tradeoff. If we are first presented with a parser, and then define its domain as those sentential forms which can arise from iteratively reducing members of L(G), then we have a quite powerful parser and probably will not be able to prove its correctness without resorting to "ad hoc" techniques.

On the other hand, if we first pick the domain of the parser as some well behaved subset of sentential forms, then we have a good chance of automatically proving the correctness of the parser by using the techniques of Theorem 1.6. The parser so tested however, derives all of its power from pattern matching and can use no information implied by the sequence in which reductions are applied. Thus it is that the reduction patterns of the previous example fail to satisfy the definition of parsing patterns although they form a working method for parsing strings in $L(G_4)$.

Chapter 2 EXACT CONTEXT METHODS

The parsing model of the previous chapter is essentially symmetric in that it does not depend on a preferred direction of application of patterns. Most "real" parsers, however, are intrinsically asymmetric in the sense of reading all input from left to right and always making the leftmost possible reduction. In this chapter we apply the idea of non-canonical parsing to the left-to-right approach to produce a parsing technique which is, in a sense, the "ultimate" left-to-right parser.

2.1 Non-canonical Parsing

Knuth [Kn 65] has suggested a class of grammars which he has called LR(k,t). This class has the property that in any sentential form, one of the leftmost t phrases can be identified (as a phrase) by examination of the entire string to its left and the first k characters (either terminal or nonterminal) to its right. A simple example will help to motivate this definition.

Consider the grammar G_5:

$$S \rightarrow A\bar{A} \mid B\bar{B}$$

$$A \rightarrow d$$

$$B \rightarrow d$$

$$\bar{A} \rightarrow e\bar{A} \mid a$$

$$\bar{B} \rightarrow e\bar{B} \mid b$$

The set of sentential forms of G_5 is as follows:

$Ae*\bar{A}$	$Ae*a$	$de*\bar{A}$	$de*a$
$Be*\bar{B}$	$Be*b$	$de*\bar{B}$	$de*b$

This grammar fails to be LR(k) for any value of k because in any sentential form belonging to de*a or de*b, we need to look all the way to the extreme right of the string in order to determine whether the phrase d should be reduced to an A or to a B. On the other hand, if we postpone this reduction and move over to the <u>second</u> phrase in the string (either a or b) we can immediately reduce either $\bar{A} \to a$ or $\bar{B} \to b$. We can then continue to reduce the second phrase of the resulting sentential form (either $e\bar{A}$ or $e\bar{B}$) until we have reduced the string to either $d\bar{A}$ or $d\bar{B}$. At this point we can go back and reduce the leftmost phrase because the $\bar{A}$ or $\bar{B}$ gives us the necessary context information.

The important thing to notice is that if the decision to reduce a phrase must be postponed, then it must be the case that in the corresponding sentential forms with all phrases to the right of the place in question reduced, the first k characters of fully reduced context tell us what decision to make. In the above example, the lookahead strings of e*a and e*b became respectively $\bar{A}$ and $\bar{B}$ when fully reduced, and thus served to differentiate the cases in question.

Let us consider another example.

$$G_6: \quad \begin{aligned} S &\rightarrow A\overline{A} \mid B\overline{B} \\ A &\rightarrow cAc \mid d \\ B &\rightarrow cBc \mid d \\ \overline{A} &\rightarrow e\overline{A} \mid a \\ \overline{B} &\rightarrow e\overline{B} \mid b \end{aligned}$$

This example is similar to the previous one except that now A and B can embed themselves in "parentheses" (actually c's) before generating d. In this case, the ability to postpone the reduction $A \rightarrow d$ or $B \rightarrow d$ does us no good because even if the rest of the sentential form was fully reduced, we would still be shielded from the necessary context by arbitrarily many c's. Hence the grammar is not LR(k,t) for any k or t. This observation is of crucial importance to all schemes with bounded right lookahead. If a "questionable" phrase can embed itself arbitrarily deeply, then the embedding characters themselves (when fully reduced) must provide us with the necessary context to make the correct reduction.

We will return to the LR(k,t) grammars in Section 2.5, but first we will present a class of grammars which represent the "ultimate" in left-to-right bottom-up parsing. This class is the naturally suggested generalization of LR(k,t) and will be designated LR(k,∞). In it, we require that in every sentential form, there must exist <u>some</u> phrase which can be identified given only its left context and first k characters of right context. (Notice that this phrase can occur anywhere in the sentential form.)

The grammar of the previous example can be "reworked" to yield an example of this concept. Consider the grammar

$$G_7: \qquad \begin{aligned} S &\to A\bar{A} \mid B\bar{B} \\ A &\to cAA' \mid d \\ B &\to cBB' \mid d \\ A' &\to c \\ B' &\to c \\ \bar{A} &\to e\bar{A} \mid a \\ \bar{B} &\to e\bar{B} \mid b \end{aligned}$$

G_7 generates exactly the same language as G_6 and as a matter of fact, any parse tree of a string generated by G_6 is the homomorphic image of a parse tree produced by G_7.

The embedding problem in G_6 has been avoided by the introduction of the "nonterminal parenthesis" A' and B'. It is now the case that the fully reduced right context character for the phrase d is either an A' or a B' and hence the proper reduction can be made. However, there is a price to be paid for this privilege, namely we must be able to postpone the decision of whether to reduce c to A' or B' an unbounded number of times. Consider how the string ccccdccccceea would be parsed. The only parsable phrase is the rightmost phrase, namely $\bar{A} \to a$. After that reduction is made, we can "swallow up" the e's and then "ripple the right context" back through the c's by successively reducing each one to an

A'. Finally we can reduce the d to A. The complete process is illustrated in Figure 2.1.

c c c c d c c c c e e a

c c c c d c c c c e e $\overline{A}$

c c c c d c c c c e $\overline{A}$

c c c c d c c c c $\overline{A}$

c c c c d c c c A'$\overline{A}$

c c c c d c c A'A'$\overline{A}$

c c c c d c A'A'A'$\overline{A}$

c c c c d A'A'A'A'$\overline{A}$

c c c c A A'A'A'A'$\overline{A}$

c c c A A'A'A'$\overline{A}$

c c A A'A'$\overline{A}$

c A A'$\overline{A}$

A $\overline{A}$

S

Figure 2.1 A non-canonical parse according to G_7.

2.2 Informal Development of LR(k,∞) Parsers

Let us now attempt to generate a parser for grammar G_5. We will do this by generating a PFA with a potentially infinite number of states. This automaton will have the ability that given any element of $SF(G_5)$, it will scan this element from left to right using one lookahead character, and halt when it finds a phrase which it "knows" can be reduced. Once we have constructed this PFA, we will be able to apply Theorem 1.10 to generate a 2PDA parser for G_5. To construct this automaton, let us first consider the set $PL(G_5)$. This set (with 1 trailing ⊣ appended) is:

$$S]_0\dashv$$

$$A\bar{A}]_1\dashv \qquad B\bar{B}]_2\dashv$$

$$Ae^+\bar{A}]_5\dashv \qquad Be^+\bar{B}]_7\dashv$$

$$Ae^*a]_6\dashv \qquad Be^*b]_8\dashv$$

$$d]_3\bar{A}\dashv \qquad d]_4\bar{B}\dashv$$

$$d]_3e^+\bar{A}]_5\dashv \qquad d]_4e^+\bar{B}]_7\dashv$$

$$d]_3e^*a]_6\dashv \qquad d]_4e^*b]_8\dashv$$

(We have chosen G_5 for this example because it yields a regular set and has a simple presentation. In general this will not be the case.)

What we propose to do is to match characters of our input string against the set $PL(G_5)$ in a left to right fashion until we find a phrase. As successive characters are scanned, we will modify $PL(G_5)$ accordingly. The set so produced will be written $T(w)$, where w is that portion of input which has already been read.

Suppose then, that our input string is da. We start with $T(\varepsilon) = PL(G_5)\dashv$. When we read the first character of input, namely d, we select all elements of $T(\varepsilon)$ which begin with a d. We then remove the leading d from each of these strings to produce $T(d)$. In this case, $T(d)$ =

$$\begin{array}{ll}]_3\bar{A}\dashv &]_4\bar{B}\dashv \\ ]_3e^+\bar{A}]_5\dashv &]_4e^+\bar{B}]_7\dashv \\ ]_3e^*a]_6\dashv &]_4e^*b]_8\dashv \end{array}$$

What $T(d)$ represents of course, is the set of all possible phrase bracketings for strings which are valid suffixes for those sentential forms beginning with a d. In general, $T(w)$ will tell us exactly 1) how we can complete the string w to produce a sentential form, and 2) what the phrase structure of each of these "completing strings" is.

Notice now that only one string in $T(d)$ begins with an a, which is our second input character. This element is $]_3a]_6\dashv$. Rather than shift off the a and continue scanning, we will instead do something much better. We will halt and signal that production 3 should be reduced. This decision is correct because $T(d)$ contains exactly those strings which describe the phrase structure of all sentential forms starting with a d. Since there is only one such phrase structure for sentential forms whose second character is an a, we can conclude that $]_3 a]_6 \dashv$ represents the only possible bracketing of the string under consideration.

Alternatively, let us suppose that our input string was dea. This time when we compare our second input character (an e) with T(d), we come up with several possible phrase structures for the termination strings. These are

$$]_3 e^{+}\bar{A}]_5 \dashv \qquad]_4 e^{+}\bar{B}]_7 \dashv$$

$$]_3 e^{+}a]_6 \dashv \qquad]_4 e^{+}b]_8 \dashv$$

In this case, since we are only using one lookahead character, we cannot tell to which of these 4 classes our input string belongs. In other words, we cannot validly reduce either production 3 or 4 based on our present knowledge of the string. So we shift off the e from these items (along with the brackets $]_3$ and $]_4$ which no longer are of use to us) and continue looking for a reducible phrase. Thus we arrive at

$$T(de) = e{*}\bar{A}]_5 \dashv \qquad e{*}\bar{B}]_7 \dashv$$

$$e{*}a]_6 \dashv \qquad e{*}b]_8 \dashv$$

(Notice how all the +'s become *'s when we shift off the leading "e".)

The procedure which we have been following has two phases. At each step we have a set T(w) and a lookahead character U. We first select all items from T(w) whose first "real" character is a U. If all of these items share a common initial bracket, then we halt and indicate that the reduction corresponding to the common bracket should be made. Otherwise, we perform phase 2 of the step by discarding all leading brackets and the initial U of each string, to produce the set T(wU).

Thus, T(dea) becomes the single string $]_6\dashv$ and we reduce production 6 (i.e. $\bar{A} \to a$) as soon as we read the next character of input, which should be a $\dashv$.

This procedure works correctly for any grammar, in that only correct reductions are ever made. It remains to be shown that a reduction is made in every sentential form. As it turns out, this will be true if and only if the grammar in question is LR(k,∞).

Let us examine how the procedure will fail for non-LR(k,∞) grammars. Suppose that our grammar is ambiguous, for example,

$$G: \qquad S \to SS \mid a$$

Now if we apply our procedure to SSS we will have the following bad things happen:

$$\{SS]_1S\dashv,\ SSS]_1\dashv\} \subseteq T(\varepsilon)$$

$$\{S]_1S\dashv,\ SS]_1\dashv\} \subseteq T(S)$$

$$\{]_1 S\dashv,\ S]_1\dashv\} \subseteq T(SS)$$

$$\{\dashv,\ ]_1\dashv\} \subseteq T(SSS)$$

Thus at no point in the scan of SSS will we be able to make a reduction and know that it is correct.

Unambiguous grammars can also fail to be LR(k,∞) for any finite value of k. Recall grammar G_6 of section 2.1 and examine the string $c^k dc^k \overline{B}$. Since G_6 is unambiguous, there is only one phrase bracketing corresponding to the string, namely $c^k d]_4 d^k \overline{B}$. Unfortunately however, $T(c^k d)$ includes both

$$]_4 c^k \overline{B}\dashv \quad \text{and} \quad]_6 c^k \overline{A}\dashv .$$

Thus this phrase cannot be reduced on the basis of its left context and first k characters of right context. Furthermore, postponing the decision and continuing to scan for other phrases doesn't work because there aren't any more phrases in $c^k dc^k \overline{B}$. Moreover, even if our input string were $c^k dc^k b$, which does contain a second phrase reducible by its context, we would still have no hope of ever being able to reduce the first phrase of the string. This is because the k characters of right context for the d are fully reduced, and already convey as much context information as they are capable of holding. This leads us to the recognition of the essential nature of LR(k,∞) grammars. Let us suppose that we are given a grammar G and some initial portion w of an input string. Suppose further that the next k characters of input are the string y and that we have been unable to reduce any previous phrases in wy. Let T' be the set of strings in T(w) whose first k "real" characters are y. If T' contains some item I which begins with a bracket and which is fully reduced (i.e. contains no complete bracketed phrases) then the present moment is our last chance for making the reduction implied by I. Thus if some other item in T' prevents us from doing this, then the grammar is not LR(k,∞). These ideas will be presented rigorously in the following section.

The procedure we have just presented is of course totally unapplicable in its present form. The most serious obstacle to be overcome is that right from the very start (i.e. at the $T(\varepsilon)$ stage) we are performing manipulations on infinite sets. If these sets were always regular there might be some hope of implementing the method, but of course, in general this will not be the case. However, the previous sentence contains an important idea. If we could store all characters of these strings (except for the first k) in a <u>fully reduced</u> form then these sets of strings would in fact be regular sets. Furthermore as we scan from left to right we could recover "on the fly" any sentential forms which we have missed, by the simple process of applying the productions of G to the k<u>th</u> character of each item. Let us rework the previous example for G_5 using this new approach which will be developed formally in section 2.3.

Once again we will be concerned with items which tell us how possible "completions" of the input string could be bracketed. However this time we will never generate a complete phrase to the right of the <u>kth</u> character in the item. V(w) will designate the set of items under consideration after reading the string w. We will once again use a look-ahead of one character (i.e. k = 1).

Thus $V(\varepsilon)$ consists exactly of the items

$$S]_0\dashv$$
$$A\bar{A}]_1\dashv$$
$$B\bar{B}]_2\dashv$$
$$d]_3\bar{A}\dashv$$
$$d]_4\bar{B}\dashv$$

Compare this set with $T(\varepsilon)$. Clearly, $V(\varepsilon) \subseteq T(\varepsilon)$. Furthermore, any element of $T(\varepsilon) - V(\varepsilon)$ can be generated by picking the appropriate element of $V(\varepsilon)$ and applying productions to those characters to the right of the <u>kth</u> character (in this case, to the second, third, etc. characters).

Reading a d as input selects the last two items in $V(\varepsilon)$ as possible descriptions of the input string. Since these two items do not imply a reduction we shift off the d to produce $\{]_3\bar{A}\dashv,\ ]_4\bar{B}\dashv\}$ as the basis of our next set. To generate V(d) we take the "closure" of this basic set by applying productions to the k = 1st "real" character of each item (i.e. either $\bar{A}$ or $\bar{B}$). Thus V(d) =

$$]_3\bar{A}\dashv \qquad]_4\bar{B}\dashv$$

$$]_3 e\bar{A}]_5\dashv \qquad]_4 e\bar{B}]_7\dashv$$

$$]_3 a]_6\dashv \qquad]_4 b]_8\dashv$$

Once again, we can recover T(d) - V(d) by applying productions to the second, third, etc. characters of each item in V(d). Furthermore, T(d) implies a reduction if and only if V(d) does. This is true because the application of productions beyond the kth character cannot affect the identity of the first k characters or the brackets contained therein.

We can use these sets V(w) to generate a PFA for G_5. The name of the state assumed by the automaton after reading the string w will be V(w). If V(w) = V(y) for w $\neq$ y, so much the better, for if we are lucky, enough such collapsing will take place to yield a finite automaton. As a matter of fact, this is exactly what happens for G_5. We present the state diagram for this PFA along with the sets corresponding to each state in Figures 2.2 and 2.3. We can now invoke Theorem 1.10 to construct a 2PDA which parses G_5 in linear time.

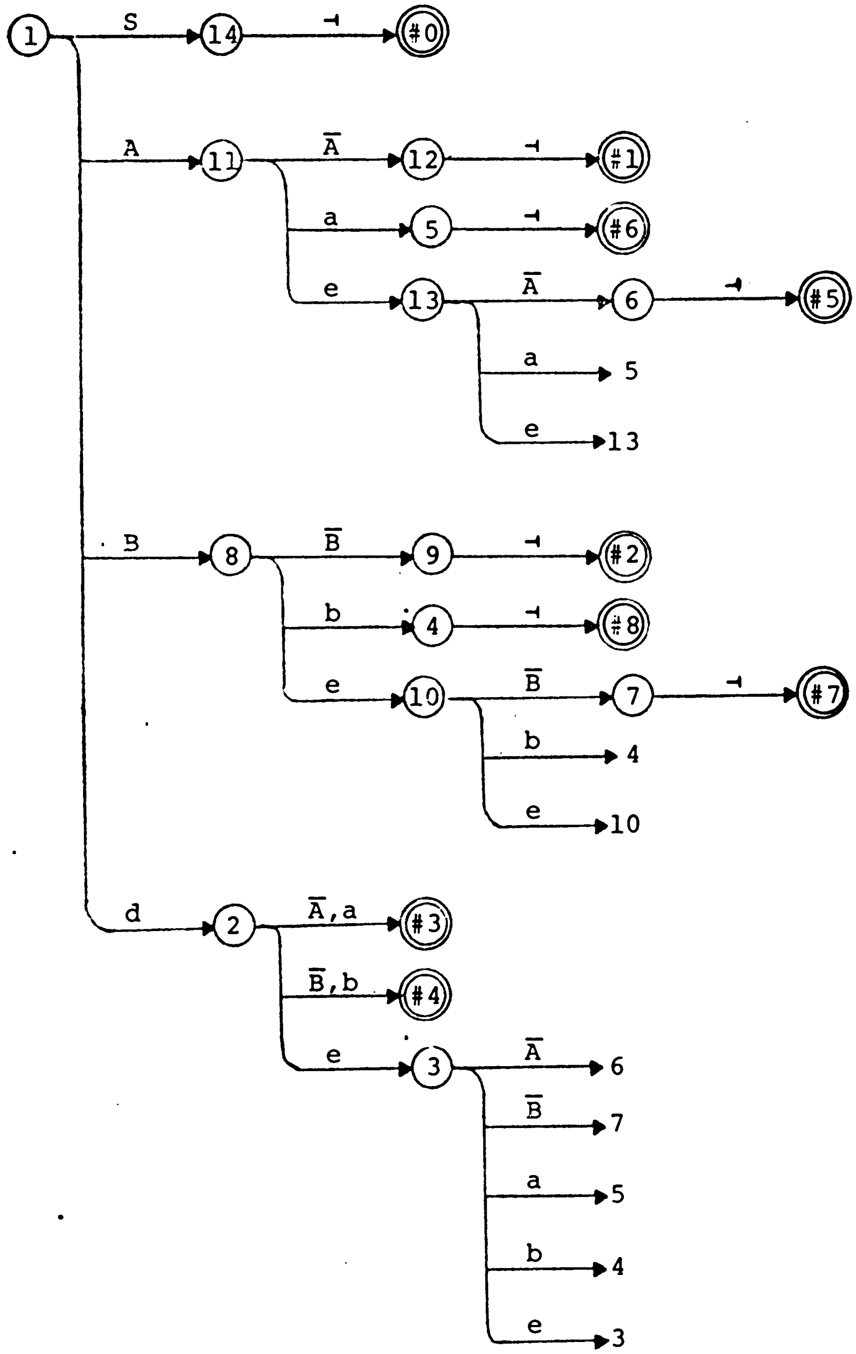


Figure 2.2 A PFA for G_5.

(1) $S\dashv$
$A\bar{A}]_1\dashv$
$B\bar{B}]_2\dashv$
$d]_3\bar{A}\dashv$
$d]_4\bar{B}\dashv$

(2) $]_3\bar{A}\dashv$
$]_4\bar{B}\dashv$
$]_3a]_6\dashv$
$]_4b]_8\dashv$
$]_3e\bar{A}]_5\dashv$
$]_4e\bar{B}]_7\dashv$

(3) $\bar{A}]_5\dashv$
$\bar{B}]_7\dashv$
$a]_6\dashv$
$b]_8\dashv$
$e\bar{A}]_5\dashv$
$e\bar{B}]_7\dashv$

(4) $]_8\dashv$

(5) $]_6\dashv$

(6) $]_5\dashv$

(7) $]_7\dashv$

(8) $\bar{B}]_2\dashv$
$b]_8\dashv$
$e\bar{B}]_7\dashv$

(9) $]_2\dashv$

(10) $\bar{B}]_7\dashv$
$b]_8\dashv$
$e\bar{B}]_7\dashv$

(11) $\bar{A}]_1\dashv$
$a]_6\dashv$
$e\bar{A}]_5\dashv$

(12) $]_1\dashv$

(13) $\bar{A}]_5\dashv$
$a]_6\dashv$
$e\bar{A}]_5\dashv$

(14) $\dashv$

Figure 2.3 The sets of strings corresponding to the states of the PFA in Figure 2.2.

2.3 Formal Development of LR(k,∞) Parsers

In this section we will present a parsing procedure for LR(k,∞) grammars. The method as first presented fails to be an algorithm because the generated states may contain infinitely many items. However, we will show later how this difficulty may be avoided. We first need several definitions.

Let w and y be elements of V^*. Then a bracket $]_i$ is said to be an unambiguous bracket with respect to w and y if (w#y,i) is a parsing pattern for G. In other words, any possible bracketing of any sentential form ...wy... must contain a $]_i$ between the indicated occurences of w and y. We will usually take w to be the entire string to the left of the point currently being scanned in the input string. Similarly, y will usually be the next k characters of input.

An item will be a string of brackets and vocabulary symbols containing at least k vocabulary symbols. Items will be written $\beta_1 V_1 \ldots \beta_k V_k \gamma$ with $\beta_i \in B^*$, $V_i \in V$ and $\gamma \in (V \cup B)^*$. We will call an item $\beta_1 Y_1 \ldots \beta_k Y_k \gamma$ valid for $X_1 \ldots X_n Y_1 \ldots Y_k$ iff

$$\exists \alpha_1, \ldots, \alpha_n \in B^* \ni 1)\ \alpha_1 X_1 \ldots \alpha_n X_n \beta_1 Y_1 \ldots \beta_k Y_k \gamma \in PL(G)$$

$$2)\ \gamma \text{ contains no complete phrases}$$

Thus an item is a possible way to "complete" the string already read so as to form a sentential form. We also require that the right context string be completely reduced. Thus the string γ can contain at most one bracket and that bracket must appear "near" the left end of γ. We present below a procedure for computing the set of items which are valid for $X_1 \ldots X_n Y_1 \ldots Y_k$. We will call the set so computed $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k)$.

<u>Procedure 2.1</u> [Computation of $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k)$]

1 [recurse] if $n=0$ then set $V' = \{\vdash^{k+1} S]_0 \dashv^k\}$
otherwise set $V' = V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_{k-1})$

2 [shift] set $V'' = \{\beta_1 V_1 \ldots \beta_k V_k \gamma \mid \exists \alpha \ni \alpha X_n \beta_1 V_1 \ldots \beta_k V_k \gamma \in V'\}$

3 [close] set $V''' = V'' \cup \{\beta_1 V_1 \ldots \beta_k x_i]_i \pi(\gamma) \mid$
$\beta_1 V_1 \ldots \beta_k V_k \gamma \in V'''$ and $V_k = A_i\}$

4 [intersect] set $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k) = \{\beta_1 V_1 \ldots \beta_k V_k \gamma \in V'' \mid$
$V_1 = Y_1, \ldots, V_k = Y_k\}$

Before proving any facts about this procedure, we will discuss a few of its features. Since we always wish to have available all possible combinations of k character strings which can validly follow the current position in the input string, it is necessary to perform the closure operation on the <u>kth</u> character of each item instead of on the first. This then causes an initialization problem which can be remedied by requiring our input to be left padded with k $\vdash$'s. (The alternative, namely initially closing on each of the first k characters of each item, was rejected as being too "messy".) Finally, although step 1 used k+1 $\vdash$'s if n=0, one of these is immediately discarded by the shift operation to produce an item with a normal complement of k endmarkers.

Theorem 2.2 An item is in $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k)$ if and only if it is valid for $X_1 \ldots X_n Y_1 \ldots Y_k$.

Proof By induction on n.

1) If $n = 0$ and $Y_1 = \ldots = Y_n = \vdash$ then the procedure yields the single item $\vdash^k S]_0 \dashv^k$. Taking $\beta_1 = \ldots = \beta_k = \varepsilon$, $\gamma = S\dashv^k$, we clearly have a valid item. Furthermore, it is clear that no other item can be valid for $\vdash^k$, because any such item would contain a complete phrase. If $n = 0$ and some $Y_i \neq \vdash$ then the procedure yields $\emptyset$ which clearly is correct since every sentential form starts with a series of k $\vdash$'s.

2) Accordingly let us assume the theorem is true for n-1. That is, $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_{k-1})$ contains any and all items valid for $X_1 \ldots X_n Y_1 \ldots Y_{k-1}$.

2a) Suppose $I = \beta_1 Y_1 \ldots \beta_k Y_k \gamma$ is valid for $X_1 \ldots X_n Y_1 \ldots Y_k$. We must show that I will be generated by the procedure. Since I is valid for $X_1 \ldots X_n Y_1 \ldots Y_k$,

$$\exists \alpha_1 \ldots \alpha_n \ni \phi = \alpha_1 X_1 \ldots \alpha_n X_n \beta_1 Y_1 \ldots \beta_k Y_k \gamma \in PL(G) \text{ and } \gamma \text{ contains no complete phrases.}$$

2a1) Suppose Y_k is not the initial symbol of a phrase in ϕ. (That is, $Y_k\gamma$ contains no complete phrase.) Then $I' = \alpha_n X_n \beta_1 Y_1 \ldots \beta_k Y_k \gamma$ is a valid item for $X_1 \ldots X_n Y_1 \ldots Y_{k-1}$ and by the inductive hypothesis, is in $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_{k-1}) = V'$.

Step 2 clearly generates I in V" as a result of the presence of I' in V'. Step 3 puts I into V‴ and Step 4 puts I into $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k)$.

2a2) Suppose Y_k is the first character of a phrase in ϕ. Then there exists an item $I'' = \beta_1 Y_1 \ldots \beta_{k-1} Y_{k-1} \beta_k \overline{Y}_k \overline{\gamma}$ valid for $X_1 \ldots X_n Y_1 \ldots Y_{k-1} \overline{Y}_k$ such that $\dot{\overline{Y}}_k$ is not the initial character of a phrase in $\overline{Y}_k \overline{\gamma}$ and such that $\overline{Y}_k \overline{\gamma} \overset{*}{\underset{lm}{\rightarrow}} Y_k \gamma$. (I" is produced by successively pruning the rightmost phrase of ϕ.) I" will be generated in V" as described in case 2a1. Step 3 then will produce I in V‴ and Step 4 will put I into $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k)$. Thus all valid items are generated in $V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k)$.

1/2 q.e.d.

2b) We must also show that all items generated by the procedure are valid. We will do this by working forward through the procedure, showing that each step preserves validity. By induction, any item in V' is valid for $X_1 \ldots X_n Y_1 \ldots Y_{k-1}$. Thus at the end of step 1, the following is true:

> if $I = \beta_1 V_1 \ldots \beta_k V_k \gamma \in V'$ then I is valid for
> $X_1 \ldots X_n Y_1 \ldots Y_{k-1}$

Step 2 is simply a shift operation, hence we conclude that if $I = \beta_1 Y_1 \ldots \beta_{k-1} V_{k-1} \beta_k \overline{Y}_k \gamma \in V''$ then I is valid for $X_1 \ldots X_n Y_1 \ldots Y_{k-1} \overline{Y}_k$. A simple induction argument (on the number of steps necessary to produce an item in V''' from V'') will show that validity is preserved by Step 3:

> Suppose $\beta_1 Y_1 \ldots \beta_{k-1} Y_{k-1} \beta_k Z \gamma \in V''$ is valid for $X_1 \ldots X_n Y_1 \ldots Y_{k-1} Z$. Then if $Z \rightarrow x_i$ is production i of G, it clearly is true that
> $\beta_1 Y_1 \ldots \beta_{k-1} Y_{k-1} \beta_k x_i]_i m(\gamma)$ is valid for
> $X_1 \ldots X_n Y_1 \ldots Y_{k-1} W$ where $W = \mathrm{TRUNC}_1(x_i m(\gamma))$.

Hence if $I = \beta_1 Y_1 \ldots \beta_{k-1} Y_{k-1} \beta_k \overline{Y}_k \gamma \in V'''$ then I is valid for $X_1 \ldots X_n Y_1 \ldots Y_{k-1} \overline{Y}_k$.

Step 4 assures that $\overline{Y}_k = Y_k$ and hence at the end of the procedure,

> $I = \beta_1 Y_1 \ldots \beta_k Y_k \gamma \in V_{k,\infty}(X_1 \ldots X_n Y_1 \ldots Y_k) \Rightarrow$
> I is valid for $X_1 \ldots X_n Y_1 \ldots Y_k$.

Thus all items generated are valid.

q.e.d.

If the grammar G contains left recursive productions, then the procedure given above will attempt to generate an infinite set and will, of course, never terminate. However, careful observation will reveal that even if the set of valid items is infinite, that set still has a finite representation as a regular set. For example, suppose G contains the production $E \to E + T$. Let us also suppose that $E\dashv$ is a valid item for some string $X_1 \ldots X_n E$. Then $E + T]\dashv$ is also a valid item, as is $E + T] + T\dashv$, $E + T] + T + T\dashv$, etc. More simply, any and all such items can be described by the expression

$$\{E\dashv\} \cup \{E + T](+T)^*\dashv\} .$$

The procedure above can accordingly be modified to use regular sets of strings as its items instead of strings. Clearly Theorem 2.2 still holds, but now we can claim that the procedure is an algorithm since we can, by the manipulation of regular expressions (which are themselves finite) describe all valid items for a given input string.

We will not modify our algorithms to reflect this change because such modifications are straightforward and tedious. Armed with the ability to compute the set of valid items for any string, we can construct an automaton (possibly with infinitely many states) for recognizing a phrase in any sentential form. The states of the automaton will simply be the sets computed by the algorithm described above. More specifically, after reading the first n characters of the string

$X_1 \ldots X_n \ldots$ the states assumed by the automaton will be $V_{k,\infty}(X_1 \ldots X_{n+k})$. The automaton about to be constructed will read a string and either

a) find a phrase,

b) state that the string is not a sentential form, or

c) report that the grammar is not LR(k,∞).

Algorithm 2.3 [Finding phrases of LR(k,∞) grammars]

Let the input be $X_1 \ldots X_m \in \vdash^k V^* \dashv^k$.

1. set $n = 0$.
2. compute $V_{k,\infty}(X_1 \ldots X_{n+k})$.
3. if $V_{k,\infty}(X_1 \ldots X_{n+k}) = \emptyset$ then the string is not a sentential form.
4. if all items in $V_{k,\infty}(X_1 \ldots X_{n+k})$ share an initial $]_i$ in common then halt indicating production i should be reduced at position n in the string.
5. if some item in $V_{k,\infty}(X_1 \ldots X_{n+k})$ contains no brackets, then halt and indicate that G is not LR(k,∞).
6. if $n+k = m$ then halt indicating that G is ambiguous and not LR(k,∞).
7. set $n = n+1$ and go to step 2.

The above algorithm is simply a loop in which the counter n is stepped from 0 to m-k. At each iteration $V_{k,\infty}(X_1 \ldots X_{n+k})$ is computed. Since the computation of $V_{k,\infty}$ is always accomplished in a finite amount of time, it is clear that:

<u>Lemma 2.4</u> Algorithm 2.3 always terminates.

<u>Theorem 2.5</u> Algorithm 2.3 is correct.

<u>Proof</u> By the above lemma, the algorithm always halts.

Suppose it halts in step 3 with $n = n'$. Then if $n' = 0$, it must be the case that the input string does not start with $\vdash^n$. Such a string obviously is not a sentential form. Suppose $n' \neq 0$. Then clearly $V_{k,\infty}(X_1 \ldots X_{n'-1+k})$ is non-empty (otherwise, we would already have halted). Furthermore, not all items in $V_{k,\infty}(X_1 \ldots X_{n'-1+k})$ contain a common initial bracket, otherwise the algorithm would have halted in step 4 with $n = n'-1$. Hence, it must be the case that $X_1 \ldots X_{n'+k}$ is not an initial segment of some sentential form. Thus if the algorithm halts in step 3, the input is not a sentential form.

Suppose the algorithm halts in step 4. That is all items in $V_{k,\infty}(X_1 \ldots X_{n'+k})$ contain a common initial bracket. Since by Theorem 2.2 $V_{k,\infty}(X_1 \ldots X_{n'+k})$ is the set of all valid items for $X_1 \ldots X_{n'+k}$, we conclude that in any sentential form starting with $X_1 \ldots X_{n'+k}$, phrase i occurs at position n'. Hence the decision of the algorithm is correct.

Suppose the algorithm halts in step 5 with $n = n'$. Then $\exists$ some item $I = X_{n'+1} \ldots X_{n'+k}\gamma$ in $V_{k,\infty}(X_1 \ldots X_{n'+k})$ such that γ contains no brackets. This in turn implies that in the sentential form $X_1 \ldots X_{n'+1} \ldots X_{n'+k}\gamma$, there are no phrases which can be recognized given only their left context and first k characters of right context. Thus G certainly is not LR(k,∞).

Suppose the algorithm halts in step 6. Then all items in $V_{k,\infty}(X_1 \ldots X_m)$ must be of the form $\beta\dashv^k$ where $\beta \in B^*$. As a matter of fact, since we "made it" past step 5, we can conclude that $\beta \in B^+$. Furthermore, since we "made it" past step 4 there must be other items in $V_{k,\infty}(X_1 \ldots X_m)$. This guarantees that there are at least two different bracketings possible for $X_1 \ldots X_m$ and hence G is ambiguous and certainly not LR(k,∞).

Thus the decision made by the algorithm is always correct.

q.e.d.

As a consequence of this result we conclude:

<u>Corollary 2.6</u> Let G be any CFG and k be a fixed integer. Then G is LR(k,∞) if and only if Algorithm 2.3 halts in step 4 whenever it is applied to an element of SF(G).

The above results have concerned finding phrases in sentential forms of LR(k,∞) grammars. As always, we rely on Theorem 1.10 to convert the PFA so produced into an actual parsing machine.

We will now study examples which put the LR(k,∞) parsing algorithm "thru its paces".

Let G_8 be

$$
\begin{aligned}
S &\rightarrow A\bar{A} \mid B\bar{B} \\
A &\rightarrow eAe \mid d \\
B &\rightarrow eBe \mid d \\
\bar{A} &\rightarrow f \\
\bar{B} &\rightarrow f
\end{aligned}
$$

If we apply the algorithm to a non-sentential form such as eedf, we go through the following succession of states:

$$
\begin{aligned}
V_{k,\infty}(\vdash) &= \{\vdash S]_0\dashv\} \\
V_{k,\infty}(\vdash e) &= \{eAe]_3\bar{A}\dashv,\ eBe]_5\bar{B}\dashv\} \\
V_{k,\infty}(\vdash ee) &= \{eAe]_3e\bar{A}\dashv,\ eBe]_5e\bar{B}\dashv\} \\
V_{k,\infty}(\vdash eed) &= \{d]_4ee\bar{A}\dashv,\ d]_6ee\bar{B}\dashv\} \\
V_{k,\infty}(\vdash eedf) &= \emptyset
\end{aligned}
$$

and the algorithm halts in step 3 as expected because no sentential form of G_8 starts with the characters eedf.

If we apply the algorithm to a sentential form such as eeAeef, we go through the following states:

$$
\begin{aligned}
V_{k,\infty}(\vdash) &= \{\vdash S]_0\dashv\} \\
V_{k,\infty}(\vdash e) &= \{eAe]_3\bar{A}\dashv,\ eBe]_5\bar{B}\dashv\} \\
V_{k,\infty}(\vdash ee) &= \{eAe]_3e\bar{A}\dashv,\ eBe]_5e\bar{B}\dashv\} \\
V_{k,\infty}(\vdash eeA) &= \{Ae]_3e\bar{A}\dashv\} \\
V_{k,\infty}(\vdash eeAe) &= \{e]_3e\bar{A}\dashv\} \\
V_{k,\infty}(\vdash eeAee) &= \{]_3e\bar{A}\dashv\}
\end{aligned}
$$

and hence we reduce production 3 after reading $\vdash$eeAe with a lookahead of e. Thus the algorithm halts in step 4.

Suppose we apply ourselves to the string edef. Then

$$
\begin{aligned}
V_{k,\infty}(\vdash) &= \{\vdash S]_0 \dashv\} \\
V_{k,\infty}(\vdash e) &= \{eAe]_3\bar{A}\dashv,\ eBe]_5\bar{B}\dashv\} \\
V_{k,\infty}(\vdash ed) &= \{d]_4e\bar{A}\dashv,\ d]_6e\bar{B}\dashv\} \\
V_{k,\infty}(\vdash ede) &= \{]_4e\bar{A}\dashv,\ ]_6e\bar{B}\dashv\} \\
V_{k,\infty}(\vdash edef) &= \{f]_7\dashv,\ f]_8\dashv\} \\
V_{k,\infty}(\vdash edef\dashv) &= \{]_7\dashv,\ ]_8\dashv\}
\end{aligned}
$$

and the algorithm halts in step 6 indicating that G_8 is ambiguous and hence not LR(k,t)..

Finally, suppose we apply the algorithm to the sentential form eedeef. We get

$$
\begin{aligned}
V_{k,\infty}(\vdash) &= \{\vdash S]_0 \dashv\} \\
V_{k,\infty}(\vdash e) &= \{eAe]_3\bar{A}\dashv,\ eBe]_5\bar{B}\dashv\} \\
V_{k,\infty}(\vdash ee) &= \{eAe]_3e\bar{A}\dashv,\ eBe]_5e\bar{B}\dashv\} \\
V_{k,\infty}(\vdash eed) &= \{d]_4ee\bar{A}\dashv,\ d]_6ee\bar{B}\dashv\} \\
V_{k,\infty}(\vdash eede) &= \{]_4ee\bar{A}\dashv,\ ]_6ee\bar{B}\dashv\} \\
V_{k,\infty}(\vdash eedee) &= \{e\bar{A}\dashv,\ e\bar{B}\dashv\}
\end{aligned}
$$

and the algorithm halts in step 5. This is because the phrase d (which should either be reduced to an A or a B) can bury itself deeply in e's which effectively shield it from the necessary context information (i.e. either $\bar{A}$ or $\bar{B}$).

2.4 Properties of LR(k,∞) Grammars

In this section we will investigate the properties of LR(k,∞) grammars. The first and most basic question deals with the format of the parsing patterns necessary for parsing this class of grammars. It will turn out that these patterns are in fact context-free sets which are k right bounded. In order to prove this result we must first establish several lemmas, some of which are important in their own right.

Lemma 2.7 G is pattern parsable with a set of k right bounded patterns iff G is LR(k,∞).

Proof If G is LR(k,∞), then we can easily generate these k right bounded patterns from the action of the LR(k,∞) parsing algorithm. It is not clear that the resulting sets are context-free, but that is unimportant for our present purposes.

Conversely, if G is pattern parsable with a set of k right bounded patterns, then by definition of pattern parsable we can find a phrase in any sentential form using only its left context and first k characters of right context. This however, is exactly what we mean when we say that G is LR(k,∞).

q.e.d.

Lemma 2.8 If G is LR(k,∞) then G is unambiguous.

Proof Immediate from Lemma 2.7 and Theorem 1.3.

Thus if G is a LR(k,∞) grammar, the "phrases of a sentential form of G" is a well-defined concept.

Lemma 2.9 Let G be LR(k,∞) and $\phi = wx_iy \in SF(G)$. If

1) x_i is the only phrase in ϕ
2) $w \overset{*}{\rightarrow} w'$
3) $y' = TRUNC_k(y)$

then

$w'x_i\#y'$ is a parsing pattern for i.

Proof Let us apply the LR(k,∞) parsing algorithm to $\phi' = w'x_iy$. Since x_i is the only phrase in ϕ, we know that the parsing algorithm will never read past the end of x_i in ϕ' without reducing x_i. Furthermore, in any new strings produced by the reductions of the parsing algorithm, it will also be true that the algorithm never reads past the end of x_i without reducing it. This is true because the algorithm cannot expect to find any phrases out there. Eventually however, x_i will be reduced. Since all reductions made are correct independent of the string following y', we conclude that this reduction could have been performed in the first place on ϕ'. Thus $w'x_i\#y'$ is a parsing pattern for i.

q.e.d.

The previous lemma says, in effect, that the rightmost phrase in a sentential form of an LR(k,∞) grammar can always be reduced by examining its left context and first k characters of right context. (This need not be true of those phrases which are not the rightmost.)

<u>Theorem 2.10</u> G is LR(k,∞) iff G is pattern parsable by a finite set of context-free, k right bounded patterns.

<u>Proof</u> $\Leftarrow$ Follows from Lemma 2.7.

$\Rightarrow$ We will first define a reduction pattern for each production. We then show that each such pattern is actually a parsing pattern and finally that the set of all such patterns covers SF(G).

1) Let $S_i = PL(G) \cap (V \cup B)^*]_i V^*$. Let g represent a generalized sequential machine which when presented with an input string w, erases all brackets except the rightmost, changes the rightmost bracket to a "#" and finally truncates w so that exactly k characters remain to the left of #. Thus the string $T^*F]_3 + I]_5{}^*F$ would become $T^*F + I\#*$ if a value of k = 1 were used. We now pick as the <u>ith</u> reduction pattern, the pattern $R_i = (g(S_i),i)$. This pattern is obviously context-free since it is defined in terms of a g.s.m. map applied to a CFL. Furthermore, by construction of g, the pattern is k right bounded.

2) We now show that every string in R_i is a parsing pattern.

Since G is LR(k,∞) we can talk about phrases in sentential forms. Suppose $wx_i\#y \in R_i$. Then $\exists z \in V^*$ such that $\phi = wx_iyz \in SF(G)$ and x_i is the rightmost phrase in ϕ. Furthermore we can repeatedly prune (using an oracle) all phrases in ϕ other than the indicated occurence of x_i. This produces a sentential form $\phi' = w'x_iyz$ where $w' \overset{*}{\rightarrow} w$. We now can apply Lemma 2.9 to conclude that if $w' \overset{*}{\rightarrow} w''$ then $w''x_i\#y$ is a parsing pattern for i. In particular, this is true for $wx_i\#y$, which was what needed to be shown.

3) We show that $\{R_i \mid 1 \leq i \leq p\}$ covers SF(G) by a direct proof.

Let $\phi = wx_iy \in SF(G)$ where the indicated x_i is the rightmost phrase in ϕ. Clearly $wx_i\#y' \in g(S_i)$ where $y' = TRUNC_k(y)$. Hence R_i applies to ϕ.

q.e.d.

The fact that the parsing patterns for LR(k,∞) grammars are context-free gives us an indication that the class is too general to be of protracted interest. As a matter of fact, we can't decide whether a grammar is LR(k,∞) even if k is a fixed integer, a fact which we prove in

<u>Theorem 2.11</u> [The LR(k,∞) decision problem]

Let G be an arbitrary CFG and k be a fixed integer. Then it is undecidable whether G is LR(k,∞).

<u>Proof</u> We can reduce this problem to the intersection problem

for deterministic context free languages. Suppose then that $G_1 = (V_1, \Sigma, P_1, S_1)$ and $G_2 = (V_2, \Sigma, P_2, S_2)$ are LR(k) grammars with disjoint sets of non-terminals, i.e. $V_1 \cap V_2 = \Sigma$.

Suppose that $\Sigma = \{\sigma_i \mid 1 \leq i \leq n\}$. Let $\hat{\Sigma} = \{\hat{\sigma}_i \mid 1 \leq i \leq n\}$ and $\bar{\Sigma} = \{\bar{\sigma}_i \mid 1 \leq i \leq n\}$ be fresh characters. We form a new grammar

$$G = (V_1 \cup V_2 \cup \hat{\Sigma} \cup \bar{\Sigma} \cup \{S\}, \Sigma, \mathscr{P}, S)$$

in which S is a new symbol. The productions $\mathscr{P}$ of G are defined in 4 "batches" as follows:

$$\mathscr{P}_1 = \{\hat{\sigma}_i \to \sigma_i \mid 1 \leq i \leq n\}$$

$$\mathscr{P}_2 = \{\bar{\sigma}_i \to \sigma_i \mid 1 \leq i \leq n\}$$

$$\mathscr{P}_3 = \{S \to S_1, S \to S_2\}$$

$$\mathscr{P}_4 = \{A \to y_1 \ldots y_m \mid A \to x_1 \ldots x_m \in P_1 \cup P_2$$

$$y_1 = \begin{cases} x_i & \text{if } x_i \notin \Sigma \\ \hat{x}_i & \text{if } x_i \in \Sigma,\ A_i \in V_1 \\ \bar{x}_i & \text{if } x_i \in \Sigma,\ A_i \in V_2 \end{cases}$$

Sentential forms of G can be immediately identified as "belonging" to either G_1 or G_2 by simple examination of their non-terminals. On the other hand, $L(G) = L(G_1) \cup L(G_2)$ cannot be separated so easily. We will show that G is LR(k,∞) iff $L(G_1) \cap L(G_2) = \emptyset$.

Suppose $L(G_1) \cap L(G_2) \neq \emptyset$. Pick any word, say w, in this intersection. Then w is a sentential form of G derivable in two different ways from S. Hence G is ambiguous and cannot be LR(k,∞).

Conversely, suppose $L(G_1) \cap L(G_2) = \emptyset$. Since G_1 and G_2 are each LR(k) we know there is a finite set of patterns $\mathcal{S}_1$ (written over $\hat{\Sigma} \cup V_1 - \Sigma$) which will perform a reduction on any string in $SF(G) \cap (\hat{\Sigma} \cup V_1 - \Sigma)^*$. A similar set of patterns $\mathcal{S}_2$ will suffice for parsing $SF(G) \cap (\bar{\Sigma} \cup V_2 - \Sigma)^*$. The question remains, how do we reduce strings which contain an element of Σ? Consider the following patterns:

$$(L(G_1) \cap \Sigma^*\sigma_i)\#\dashv \qquad \text{for} \quad \hat{\sigma}_i \to \sigma_i$$

and

$$(L(G_2) \cap \Sigma^*\sigma_i)\#\dashv \qquad \text{for} \quad \bar{\sigma}_i \to \sigma_i$$

which completely cover $SF(G) \cap \Sigma^*$. Since $L(G_1) \cap L(G_2)$ are disjoint, so too are the patterns.

Similarly the patterns

$$\begin{array}{lll} \sigma_i\#(\hat{\Sigma} \cup V_1 - \Sigma) & \text{for} & \hat{\sigma}_i \to \sigma_i \\ \sigma_i\#(\bar{\Sigma} \cup V_2 - \Sigma) & \text{for} & \bar{\sigma}_i \to \sigma_i \\ (\hat{\Sigma} \cup V_1 - \Sigma)\sigma_i\# & \text{for} & \hat{\sigma}_i \to \sigma_i \\ (\bar{\Sigma} \cup V_2 - \Sigma)\sigma_i\# & \text{for} & \bar{\sigma}_i \to \sigma_i \end{array}$$

cover that subset of $SF(G) - \Sigma^*$ in which each string contains at least one symbol in Σ. The reader can easily verify that all of the above patterns are parsing patterns and that in conjunction with $\mathcal{S}_1$ and $\mathcal{S}_2$ they completely cover SF(G). Hence G is pattern parsable using right bounded patterns and by Lemma 2.7 is LR(k,∞).

q.e.d.

In the final two theorems of this section we establish limits on the "power" of LR(k,∞) grammars by showing that they lie strictly between the LR(k) grammars and the class of unambiguous context-free grammars. We also show that the LR(k,∞) languages properly contain the deterministic context-free languages.

Theorem 2.12 There are unambiguous context-free grammars which are not LR(k,∞) for any k.

Proof Consider G_9 below:

$$S \rightarrow Aa \mid Bb$$
$$A \rightarrow Ac \mid d$$
$$B \rightarrow Bc \mid d$$

G_9 is linear, and hence G_9 is LR(k,∞) iff G_9 is LR(k). But it is easy to show that G_9 is not LR(k) because $dc^k a$ cannot be distinguished from $dc^k b$ until it is "too late".

q.e.d.

Theorem 2.13 Every deterministic language has an LR(k,∞) grammar. However, there are LR(k,∞) grammars which generate languages which are not deterministic.

Proof The first statement is trivial because every deterministic language has an LR(k) grammar. This grammar is of course also LR(k,∞).

As an example of the second statement, consider G_{10} below:

$$S \to A \mid B$$
$$A \to aA\overline{A} \mid ac\overline{A}$$
$$B \to aB\overline{B}\,\overline{B} \mid ac\overline{B}\,\overline{B}$$
$$\overline{A} \to b$$
$$\overline{B} \to b$$

$$L(G_{10}) = \{a^n cb^n \mid n \geq 1\} \cup \{a^n cb^{2n} \mid n \geq 1\}$$

This language is easily shown to not be deterministic. However, the following parsing patterns suffice to show that G_{10} is LR(k,∞).

$(\{\vdash A\#\dashv\}, 1)$	$(\{\vdash B\#\dashv\}, 2)$
$(\{aA\overline{A}\#\}, 3)$	$(\{ac\overline{A}\#\}, 4)$
$(\{aB\overline{B}\,\overline{B}\#\}, 5)$	$(\{ac\overline{B}\,\overline{B}\#\}, 6)$
$(\{Ab\#, \overline{A}b\#, b\#\overline{A}\}, 7)$	$(\{Bb\#, \overline{B}b\#, b\#\overline{B}\}, 8)$
$(\{a^n cb^n\# \mid n \geq 1\}, 7)$	$(\{a^n cb^{2n}\# \mid n \geq 1\}, 8)$

q.e.d.

2.5 Development of LR(k,t) Parsers

Let us now return to LR(k,t) grammars and their parsers. We recall from section 2.1, that a grammar is LR(k,t) if in any sentential form one of the leftmost t phrases can be identified solely by its left context and first k characters of right context. The LR(k,∞) parsing algorithm will provide us with a basis for constructing an LR(k,t) parsing algorithm. Once again, we will be concerned with an automaton whose states will be items. This time however, an individual item will consist not only of a bracketed string which gives a possible "local" structure of the sentential form, but also of a counter, which specifies how many phrases have been bypassed in the bracketing corresponding to the particular item being considered. Furthermore, we will truncate the right context information in such a way that it will have a bounded length. We will illustrate these ideas with a few simple examples.

Let G_{11} be

$$S \rightarrow AA\bar{A} \mid B\bar{B}$$
$$A \rightarrow c$$
$$B \rightarrow cc$$
$$\bar{A} \rightarrow e\bar{A} \mid a$$
$$\bar{B} \rightarrow e\bar{B} \mid b$$

We will construct an automaton for finding phrases in $SF(G_{11})$. Since no non-terminal is self-embedding, our right context information will always be of bounded length. Let us compute the successive sets of items which describe the structure of the string cceea. If we have not yet examined the first character of the sentential form, we must include the item $S]_0\dashv 0$ as a possible description of the form (the trailing zero indicates that no phrases have been bypassed so far, or more specifically, no brackets have been shifted off the left end of the item). Furthermore we must include also the items $AA\bar{A}]_1\dashv 0$ and $B\bar{B}]_2\dashv 0$. Notice we are simply taking the "closure" of the item set by applying productions to the k=1st character of each item. Finally we produce the items $c]_3A\bar{A}\dashv 0$ and $cc]_4\bar{B}\dashv 0$.

Our initial item set then is

1 $S]_0\dashv 0$
$AA\bar{A}]_1\dashv 0$
$B\bar{B}]_2\dashv 0$
$c]_3A\bar{A}\dashv 0$
$cc]_4\bar{B}\dashv 0$

These items describe any and all sentential forms which have no complete phrase beyond the first character. Furthermore, any other sentential form can be derived from one of the above forms by applying one or more productions to characters other than the first.

We now look at the initial input character and discover that it is a c. Accordingly, we shift to the following set of items

$]_3 A\bar{A}\dashv 0$

$c]_4 \bar{B}\dashv 0$

which after closing becomes

2 $]_3 A\bar{A}\dashv 0$

$c]_4 \bar{B}\dashv 0$

$]_3 c]_3 \bar{A}\dashv 0$

The second input character is also a c which means we cannot decide whether or not to reduce production 3. Accordingly, we shift to the item set

$]_4 \bar{B}\dashv 0$

$]_3 \bar{A}\dashv 1$

Notice that in the second item, we have increased the count of bypassed phrases to reflect the fact that we have discarded a possible bracket. These items indicate clearly that in any sentential form beginning with two c's, the first c is a phrase only in those sentential forms which lie in the "A-half" of the grammar.

We can continue this process as before. If we look at all possible input strings, we find that only a finite number of states is generated. In Figures 2.4 and 2.5 we display the completed PFA for G_{11} along with the sets of items corresponding to each state.

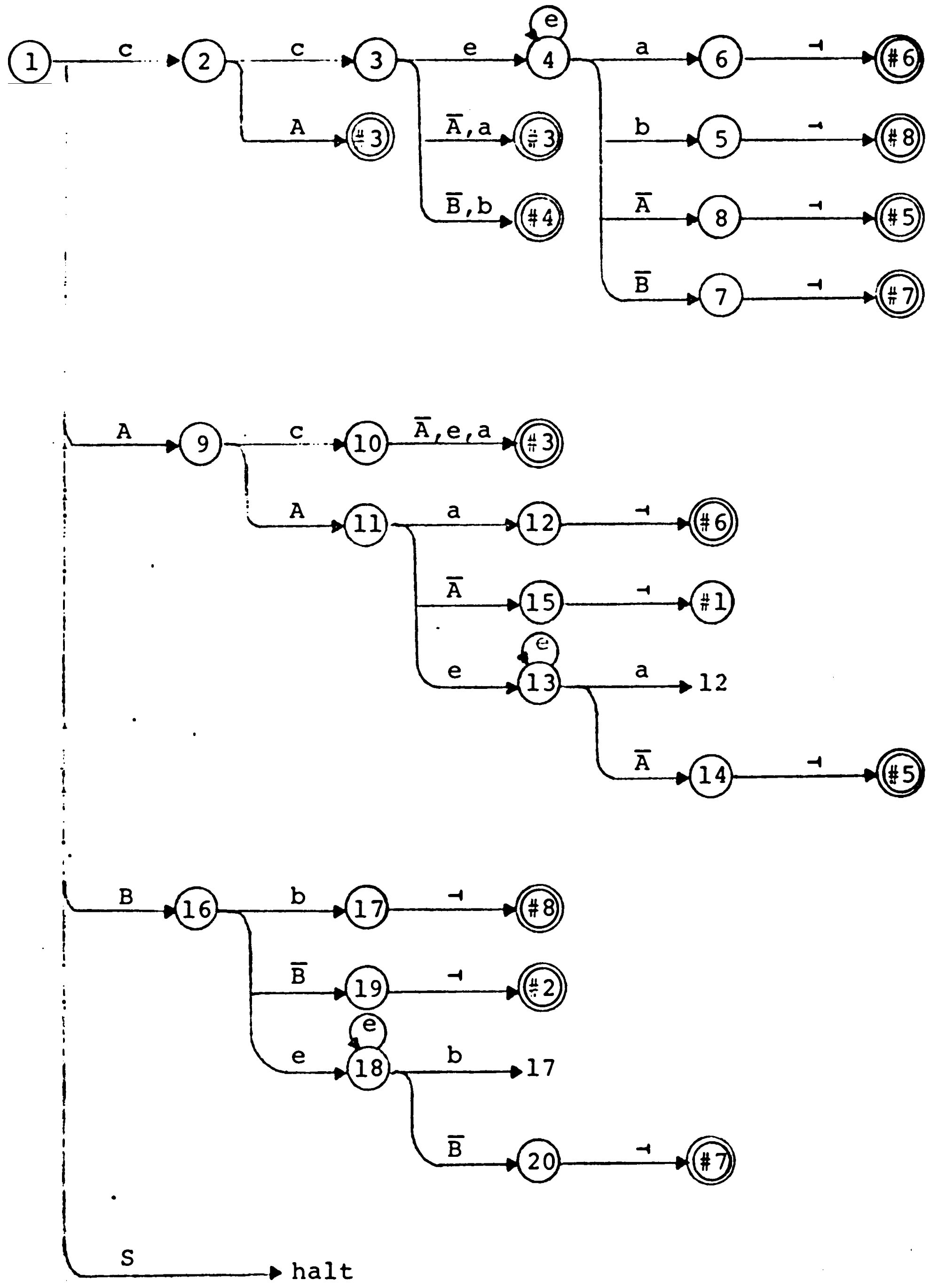


Figure 2.4 A PFA for G_{11}

(1) $S \dashv 0$
$AA\bar{A}]_1 \dashv 0$
$B\bar{B}]_2 \dashv 0$
$c]_3 A\bar{A} \dashv 0$
$cc]_4 \bar{B} \dashv 0$

(2) $]_3 A\bar{A} \dashv 0$
$c]_4 \bar{B} \dashv 0$
$]_3 c]_3 \bar{A} \dashv 0$

(3) $]_4 \bar{B} \dashv 0$
$]_3 \bar{A} \dashv 1$
$]_4 e\bar{B}]_7 \dashv 0$
$]_3 e\bar{A}]_5 \dashv 1$
$]_4 b]_8 \dashv 0$
$]_3 a]_6 \dashv 1$

(4) $\bar{B}]_7 \dashv 1$
$\bar{A}]_5 \dashv 2$
$e\bar{B}]_7 \dashv 1$
$e\bar{A}]_5 \dashv 2$
$b]_8 \dashv 1$
$a]_6 \dashv 2$

(5) $]_8 \dashv 1$

(6) $]_6 \dashv 2$

(7) $]_7 \dashv 1$

(8) $]_5 \dashv 2$

(9) $A\bar{A}]_1 \dashv 0$
$a]_3 \bar{A}]_1 \dashv 0$

(10) $]_3 \bar{A}]_1 \dashv 0$
$]_3 e\bar{A}]_5 \dashv 0$
$]_3 a]_6 \dashv 0$

(11) $\bar{A}]_1 \dashv 0$
$e\bar{A}]_5 \dashv 0$
$a]_6 \dashv 0$

(12) $]_6 \dashv 0$

(13) $\bar{A}]_5 \dashv 0$
$e\bar{A}]_5 \dashv 0$
$a]_6 \dashv 0$

(14) $]_5 \dashv 0$

(15) $]_1 \dashv 0$

(16) $\bar{B}]_2 \dashv 0$
$e\bar{B}]_7 \dashv 0$
$b]_8 \dashv 0$

(17) $]_8 \dashv 0$

(18) $\bar{B}]_7 \dashv 0$
$e\bar{B}]_7 \dashv 0$
$b]_8 \dashv 0$

(19) $]_2 \dashv 0$

(20) $]_7 \dashv 0$

Figure 2.5 Item sets corresponding to the states of the PFA in Figure 2.4.

Examination of this machine reveals that we need never bypass more than two phrases while parsing. Since we used only one lookahead character in building the automaton we conclude that G_{11} is LR(1,2).

It is our intention to use the above process as a basis for the construction of LR(k,t) parsers. Obviously, if we ever generate an item in which more than t phrases have been bypassed, we can conclude that the grammar in question is not LR(k,t). By looking at the sequence of states passed through while accessing the "forbidden" state, we will be able to produce a series of different bracketings for the initial segment of the input string, which demonstrates why no phrase can ever be reduced based solely on its left context and first k characters of right context. Furthermore, one of these bracketings will contain t "imparsable" phrases and thus serve as a counterexample to any claim that the given grammar is LR(k,t) for those values of k and t under consideration.

One difficulty remains. If the grammar in question contains productions which are self-embedding or left-recursive, then the size of the right context strings can grow without bounds. However, we can show that it is never necessary to retain more than kt characters of right context if the grammar in question is to be LR(k,t). This in turn implies that not only will the size of each item be bounded but also the number of items in a state will be bounded. Thus the generated parsing automaton will have a finite number of states.

Let us define an LR(k,t) item as being an ordered pair (γ, h) such that γ is a string $\in (B \cup V)^*$ containing at least k occurences of characters in V and h is an integer such that $0 \leq h \leq t$.

An item $(\beta_1 Y_1 \ldots \beta_k Y_k \gamma, h)$ is a valid LR(k,t) item for the string $X_1 \ldots X_n Y_1 \ldots Y_k$ iff

$\exists \psi \in V^*, \alpha_1, \ldots, \alpha_n \in B^* \ni$

1) $\alpha_1 X_1 \ldots \alpha_n X_n \beta_1 Y_1 \ldots \beta_k Y_k \gamma \psi \in PL(G)$
2) $|\alpha_1 \ldots \alpha_n| = h$
3) $\gamma\psi$ contains no complete phrases
4) γ contains at most kt characters to the right of the rightmost bracket.

Lemma 2.14 The set of all valid LR(k,t) items for a given grammar G and fixed values of k and t is a finite set.

Proof Let m equal the length of the longest rhs of a production in G. Since γ can contain no complete phrases, γ can have at most m-1 characters to the left of its one and only bracket (if it has a bracket at all). Thus the total number of characters of V occurring in an item is at most k+m-1+kt. Furthermore, in the absence of ε-rules, we can have at most k+1 brackets occuring in an item. Thus an item is a string of at most 2k+m+kt characters over $B \cup V$ and an integer between 0 and t. Clearly there are but a finite number of such items.

q.e.d.

Suppose that $I = (\phi]_i\psi, h)$ is an LR(k,t) item and that ψ contains no brackets (i.e. $]_i$ is the rightmost bracket in the item). Then ψ is called the <u>right context string</u> of I.

Two LR(k,t) items $I = (\gamma, h)$ and $I' = (\gamma', h')$ are called <u>equivalent</u> if $h = h'$ and γ is a proper prefix of γ'.

By concentrating on equivalence classes of LR(k,t) items we will avoid the difficulty of having to carry around similar items differing only in amount of right context.

We will next present an algorithm for computing the set of equivalence classes of valid LR(k,t) items for the string $X_1 \ldots X_n Y_1 \ldots Y_k$. The output of this algorithm will be called $V_{k,t}$.

<u>Algorithm 2.15</u> [Computation of $V_{k,t}(X_1 \ldots X_n Y_1 \ldots Y_k)$]

1. if $n = 0$ set $V' = \{(\vdash^{k+1} S]_0 \dashv^k, 0)\}$
   otherwise set $V' = V_{k,t}(X_1 \ldots X_n Y_1 \ldots Y_{k-1})$

2. set $V'' = \{(\beta_1 V_1 \ldots \beta_k V_k \gamma, h+|\alpha|) \mid \exists \alpha \ni (\alpha X_n \beta_1 V_1 \ldots \beta_k V_k \gamma, h) \in V'\}$

3. set $V''' = V'' \cup \left\{ \begin{array}{l} (\beta_1 V_1 \ldots \beta_k x_i]_i z, h) \mid z \in \mathrm{TRUNC}_{kt}(m(\gamma)), \\ (\beta_1 V_1 \ldots \beta_k V_k \gamma, h) \in V''' ,\ V_k = A_i \end{array} \right\}$

4. set $V_{k,t}(X_1 \ldots X_n Y_1 \ldots Y_k) = \{(\beta_1 V_1 \ldots \beta_k V_k \gamma, h) \in V''' \mid$
   $V_1 = Y_1, \ldots, V_k = Y_k\}$

5. if any item ends in a $\dashv$ but has fewer than kt characters of right context, append $\dashv$'s to the item until there are either k $\dashv$'s or kt right context characters.

Given this algorithm, we proceed directly to the LR(k,t) parsing algorithm before proving the correctness of our current work.

Algorithm 2.16 [Phrase finding for LR(k,t) grammars]

Let the input string be $\phi = X_1 \ldots X_m \in \vdash^k V^* \dashv^k$

1. Set n = 0.
2. Compute $S = V_{k,t}(X_1 \ldots X_{n+k})$.
3. If $S = \emptyset$, then halt because $\phi \notin SF(G)$.
4. If all items in S share $]_i$ as a common initial bracket then halt, indicating that reduction i should be applied at position n of ϕ.
5. If some item in S contains no brackets, then halt, G is not LR(k,t).
6. If some item (α,h) in S has $h > t$ then halt, G is not LR(k,t).
7. If $n + k = m$, then halt, G is not LR(k,t).
8. Set $n = n + 1$ and go to step 2.

The above algorithms are correct for much the same reasons as Algorithms 2.1 and 2.3. We must however verify that the shift operation never "gobbles up" so many characters that insufficient context is left for parsing. In other words, we must show that at least k characters are always maintained in an item. First a definition.

Let $(\phi]_i\psi,\ h)$ be an item. Then the <u>count</u> of the indicated bracket is the number of brackets to its left in any bracketing of a sentential form corresponding to this item. This value is simply $h+|\phi|-|m(\phi)|$. (h tells how many brackets have been shifted off, and $|\phi|-|m(\phi)|$ tells how many brackets are in ϕ.

<u>Lemma 2.17</u> If G is LR(k,t), then any and all items which occur during parsing according to Algorithm 2.16 contain at least one bracket.

<u>Proof</u> Directly from step 5 of the parsing algorithm.

<u>Lemma 2.18</u> Let $(\phi]_i\psi,\ h)$ be an item in $V_{k,t}(X_1 \ldots X_n Y_1 \ldots Y_k)$. Then ψ either

1) contains one or more $\dashv$'s
2) is of length $\geq k(t - \text{count}(]_i))$.
$= k(t - h - |\phi| + |m(\phi)|)$.

<u>Proof</u> We will show that the above property is true of the item $(\vdash^k S]_0 \dashv^k, 0)$ and that it is preserved by both the shift operation and the closure operation.

Consider the item $(\vdash^k S]_0 \dashv^k, 0)$. The lemma holds true of this item since $]_0$ is the only bracket in the item and it is immediately followed by k $\dashv$'s.

Assume the lemma is true of some item $I = (\beta X \phi]_i\psi, h)$. Shifting produces a new item $I' = (\phi]_i\psi,\ h+|\beta|)$. Suppose ψ in I contained one or more

$\dashv$'s. Then obviously ψ in I' contains one or more $\dashv$'s and the lemma holds true. On the other hand, suppose $|\psi| \geq k(t - h - |\beta X\phi| + |m(\beta X\phi)|) =$
$k(t - h - |\beta| - |\phi| + |m(\phi)|) =$
$k(t - (h + |\beta|) - |\phi| + |m(\phi)|)$. Thus ψ in I' is of the correct length and the lemma holds. Thus the shift operation preserves the lemma.

For the closure operation we must show

1) the property is preserved for existing brackets in the item, and
2) the property holds true for any new brackets introduced into the item

1) So assume the lemma is true of $I = (\phi]_i\psi,h)$.

<u>Case 1</u> $|m(\phi)| < k$. This means that the closure operation is performed to the right of $]_i$ to produce a new item $I' = (\phi]_i\psi',h)$.

If truncation occured in the production of ψ', then we clearly have at least kt characters of context for $]_i$ and the lemma is true.

If truncation did not occur, and ψ contains one or more $\dashv$'s, then certainly ψ' contains these same $\dashv$'s. Finally if truncation did not occur and ψ contains no $\dashv$'s, then we must have $|\psi| \geq k(t - \text{count}(]_i))$. But in the absence of ε-rules and truncation, $|\psi'| \geq |\psi|$. Hence $|\psi'| \geq k(t - \text{count}(]_i))$.

<u>Case 2</u> $|m(\phi)| \geq k$. Then $]_i$ is removed from the item as it no longer designates a phrase. Hence nothing further need be proven.

2) Suppose $I' = (\phi x_i]_i\psi\ , h)$ is produced from $I = (\phi A_i\psi, h)$. We will show the property holds for $]_i$.

<u>Case 1</u> ψ contains a bracket. Then $I = (\phi A_i\psi_1]_j\psi_2, h)$ where $\psi' = \mathrm{TRUNC}_{kt}(\psi_1\psi_2)$ and neither ψ_1 nor ψ_2 contain any brackets.

If $|\psi'| = kt$ then the lemma certainly holds for $]_i$ in I' because the right context string ψ is sufficiently long.

If $|\psi'| < kt$ then no truncation took place and $\psi' = \psi_1\psi_2$.

If ψ_2 contains any $\dashv$'s then so does ψ' and we are done.

If ψ_2 contains no $\dashv$'s then

$$|\psi_2| \geq k(t - h - |\phi A_i\psi_1| + |m(\phi A_i\psi_1)|)$$

$$= k(t - h - |\phi| + |m(\phi)|).$$

But since $|\psi'| \geq |\psi_2|$, we have $|\psi'| \geq k(t - h - |\phi| + |m(\phi)|)$ which is exactly what we needed.

<u>Case 2</u> ψ contains no brackets. Since I must contain a bracket somewhere, we conclude that it must be hidden in ϕ. Accordingly we will concentrate on the rightmost such bracket. So $I = (\phi_1]_j\phi_2A_i\psi, h)$ where $\phi_1]_j\phi_2 = \phi$

and neither ϕ_2 nor ψ contain any brackets.

No truncation can occur in this case since $|\phi_2 A_i \psi| \leq kt \Rightarrow |\psi| \leq kt$. Thus $\psi' = \psi$.

If ψ contains any $\dashv$'s then we are done. Otherwise, $|\phi_2 A_i| + |\psi| = |\phi_2 A_i \psi| \geq k(t - h - |\phi_1| + |m(\phi_1)|)$. Since $k \geq |\phi_2 A_i|$, $|\psi| \geq k(t - h - |\phi_1| + |m(\phi_1)|) - k$

$$= |\phi_1| - |m(\phi_1)| + |]_j| - |m(]_j)| + |\phi_2| - |m(\phi_2)|$$
$$= |\phi_1| - |m(\phi_1)| + 1,$$

$|\psi| \geq k(t - h - |\phi| + |m(\phi)|$ which is exactly what was needed.

q.e.d.

Theorem 2.19 Let (γ, h) be an item generated when Algorithm 2.15 is applied to some string. If $h < t$ then γ contains at least k occurences of characters from V.

Proof Follows directly from step 6 of the algorithm and lemma 2.18.

q.e.d.

Theorem 2.20 Let G be an LR(k,t) grammar. Then $V_{k,t}(X_1 \ldots X_n Y_1 \ldots Y_k)$ as computed by Algorithm 2.15 contains one item for each equivalence class of the valid LR(k,t) items for $X_1 \ldots X_n Y_1 \ldots Y_k$. Furthermore, for each item valid for $X_1 \ldots X_n Y_1 \ldots Y_k$, there is an equivalent item in $V_{k,t}(X_1 \ldots X_n Y_1 \ldots Y_k)$.

Proof Follows the same lines as Theorem 2.2 but also uses

Theorem 2.19 to show that truncated right context is sufficient for generating the representatives of each equivalence class.

Theorem 2.21 Let G be a CFG and k and t be fixed. Then Algorithm 2.16 terminates when applied to any string. Furthermore, any reduction made by the Algorithm is correct.

Proof Immediate from the algorithm and the previous theorem.

q.e.d.

2.6 Properties of LR(k,t) Grammars

In this section we will investigate some of the properties of LR(k,t) grammars, especially those which are related to the implementation of their parsers. We start by showing that the LR(k,t) decision problem is solvable.

Theorem 2.22 [The LR(k,t) Decision Procedure]

Let G be a CFG and k and t be fixed integers. Then it is decidable whether G is LR(k,t).

Proof By lemma 2.14, the set of valid LR(k,t) items for G is a finite set. Furthermore the maximum size of this set is known a priori and is bounded above by

$$(1 + t)(|V| + p + 2)^{2k+m+kt}$$

where p is the number of productions in G and m is the longest righthand side length of a production in G. Call the above number n. Since states of an LR(k,t) parser are sets of items, the bound on the number of states for such a parser is 2^n.

To test whether G is LR(k,t) or not, simply compute the transitions for all possible states and see if any state containing an item whose count is equal to t or which contains no brackets is accessible. (See Algorithm 2.16 for details of constructing these transitions.)

q.e.d.

Inherent in the above proof is the fact that the parsing automaton for an LR(k,t) grammar is a finite state automaton. Theorem 1.10 therefore applies and we conclude that these grammars are parsable on a 2PDA in linear time. However, it is possible to do even better than this. Since each reduction causes us to backup at most k characters in our input stream, and the total number of consecutive backups is clearly bounded by t, we can actually do the backup operation by utilizing a buffer of size kt characters. Hence we only need one stack for parsing, a fact with two immediate consequences.

<u>Theorem 2.23</u> An LR(k,t) grammar can be parsed in linear time on a deterministic pushdown automaton.

<u>Theorem 2.24</u> If G is an LR(k,t) grammar, then L(G) is a deterministic context-free language.

This latter theorem suggests that adding "bounded postponements" to our parsers does not buy us nearly so much power as we had hoped. Clearly, what is needed is an "unbounded postponement" capability (such as is used by the LR(k,∞) parsers). This approach will be investigated further in Chapter 3.

Before leaving the realm of LR(k,t) grammars, we record several routine theorems. In particular, we show that although the LR(k,t) decision problem is solvable for fixed k and t, this same problem is unsolvable if either k or t is free, a result which is entirely analogous to the situation for LR(k) grammars. We show that both k and t represent "useful parsing resources", that is, increasing either parameter yields a larger class of parsable grammars. Finally, we show that the LR(k,∞) grammars are strictly more powerful than the LR(k,t) grammars, a fact suggested by the fact that a more powerful automaton is necessary for parsing the former class.

Theorem 2.25 Let G be a CFG and t be a fixed integer. Then it is recursively undecidable whether there exists a $k \geq 0$ such that G is LR(k,t).

Proof In [Kn 65], Knuth shows that it is undecidable of an arbitrary linear grammar G', whether or not there exists some k such that G' is LR(k). Since a linear grammar is one in which there is at most one phrase in any sentential form, we conclude that a linear grammar is LR(k) iff it is LR(k,t) iff it is LR(k,∞). The theorem follows immediately.

q.e.d.

Theorem 2.26 Let k be some fixed integer and G be a CFG. Then it is undecidable whether $\exists$t such that G is LR(k,t).

Proof Consider the Partial Correspondence Problem [Kn 65]. That is, given a finite set of ordered pairs of nonempty strings $\{(\alpha_i,\beta_i) \mid 1 \leq i \leq n\}$, do there exist for all $p > 0$, ordered p tuples of integers $(i_1,i_2,\ldots,i_p)$ such that the first p characters of $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_p}$ match the first p characters of $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_p}$. Knuth has shown this problem to be undecidable.

Accordingly, let $\{(\alpha_i,\beta_i) \mid 1 \leq i \leq n\}$ be an instance of the partial correspondence problem over some alphabet Σ.

Let $\Sigma = \sigma_1,\sigma_2,\ldots,\sigma_m$. Let $a,b,c_1,\ldots,c_n$ be new symbols.

Let $\bar{\sigma}_1,\bar{\sigma}_2,\ldots,\bar{\sigma}_m,\bar{\bar{\sigma}}_1,\ldots,\bar{\bar{\sigma}}_m,A,B,S$ be new symbols.

Construct G =

$$S \to Aa \mid Bb$$

$$A \to c_jA\bar{\sigma}_{j_1}\ldots\bar{\sigma}_{j_{k_j}} \mid c_j\bar{\sigma}_{j_1}\ldots\bar{\sigma}_{j_{k_j}}$$

where $\alpha_j = \sigma_{j_1}\ldots\sigma_{j_{k_j}}$

$$B \to c_jB\bar{\bar{\sigma}}_{j_1}\ldots\bar{\bar{\sigma}}_{j_{k_j}} \mid c_j\bar{\bar{\sigma}}_j\ldots\bar{\bar{\sigma}}_{j_{k_j}}$$

where $\beta_j = \sigma_{j_1}\ldots\sigma_{j_{k_j}}$

$$\bar{\sigma}_j \to \sigma_j$$

$$\bar{\bar{\sigma}}_j \to \sigma_j$$

Clearly G is unambiguous. We will show that G is LR(k,t) for some t iff the partial correspondence problem has a negative answer. The theorem will then follow immediately.

Accordingly,

suppose the answer is affirmative; that is

$(\forall p > 0)(\exists i_1 \ldots i_p) \ni \mathrm{TRUNC}_p(\alpha_{i_1} \ldots \alpha_{i_p}) = \mathrm{TRUNC}_p(\beta_{i_1} \ldots \beta_{i_p})$.

Now suppose that G is LR(k,t) for some particular t. Let $q = k + t$. Thus there are particular

$$i_1, \ldots, i_q \ni \mathrm{TRUNC}_q(\alpha_{i_1} \ldots \alpha_{i_q}) = \mathrm{TRUNC}_q(\beta_{i_1} \ldots \beta_{i_q}).$$

Examine the two sentential forms

$$w_1 = c_{i_q} \ldots c_{i_1} \alpha_{i_1} \ldots \alpha_{i_q} a \qquad \text{and}$$

$$w_2 = c_{i_q} \ldots c_{i_1} \beta_{i_1} \ldots \beta_{i_q} b.$$

Since G is unambiguous we can talk about phrases of sentential forms without danger of confusion.

Notice that the j<u>th</u> phrase of w_1 is exactly the j<u>th</u> character of $\alpha_{i_1} \ldots \alpha_{i_q}$ and that the j<u>th</u> character of $\beta_{i_1} \ldots \beta_{i_q}$ is exactly the j<u>th</u> phrase of w_2. Furthermore for $1 \leq j \leq t$ these phrases belong either to the class $\{\bar{\sigma} \rightarrow \sigma\}$ or $\{\bar{\bar{\sigma}} \rightarrow \sigma\}$ depending on which "half" of the grammar the sentential form belongs to.

It should be equally clear that for $1 \leq j \leq t$, the j<u>th</u> phrase of w_1 has the same left context and first k characters of right context as the j<u>th</u> phrase of w_2. Hence the j<u>th</u> phrase cannot be reduced and G is not LR(k,t). Since the above argument can be used for any value of t, we conclude that G is not in the class of LR(k,t) grammars (for this value of k).

Let us now suppose that the partial correspondence problem has a negative answer. That is there exists some $p \ni \forall i_1 \ldots i_p$,

$$\mathrm{TRUNC}_p(\alpha_{i_1} \ldots \alpha_{i_p}) \neq \mathrm{TRUNC}_p(\beta_{i_1} \ldots \beta_{i_p}).$$

Then the following reduction patterns are parsing patterns for G.

$c{*}c_{i_p} \ldots c_{i_1} w\#$ for $\bar{\sigma}_j \to \sigma_j$ if $w = \mathrm{TRUNC}_p(\alpha_{i_1} \ldots \alpha_{i_p})$ and σ_j is the pth character of w.

$c{*}c_{i_p} \ldots c_{i_1} w\#$ for $\bar{\bar{\sigma}}_j \to \sigma_j$ if $w = \mathrm{TRUNC}_p(\beta_{i_1} \ldots \beta_{i_p})$ and σ_j is the pth character of w.

$\sigma_j \# \bar{\bar{\sigma}}_k$ for $\bar{\bar{\sigma}}_j \to \sigma_j$

$\sigma_j \# \bar{\sigma}_k$ for $\bar{\sigma}_j \to \sigma_j$

$c_j \bar{\sigma}_{j_1} \bar{\sigma}_{j_2} \ldots \bar{\sigma}_{j_{k_j}} \#$ for $A \to c_j \bar{\sigma}_{j_1} \ldots \bar{\sigma}_{j_{k_j}}$

etc.

Furthermore, these patterns always reduce one of the first p phrases and hence G is LR(k,p).

q.e.d.

Theorem 2.27 For any fixed k and t,

1) $\exists$ a grammar which is LR(k+1,t) but not LR(k,t).

2) $\exists$ a grammar which is LR(k,t+1) but not LR(k,t).

Proof 1) Suppose k is a fixed integer. Let G be

$$S \to Ae^k a \mid Be^k b$$

$$A \to c$$

$$B \to c$$

G is linear and therefore LR(k,t) iff it is LR(k). G however is certainly not LR(k) because the two sentential forms $ce^k a$ and $ce^k b$ both have c as their handle. This handle has the same k characters of right context and hence can't be reduced. It is equally clear however, that G is LR(k+1) and hence LR(k+1,t) for all t.

2) Suppose t is a fixed integer. Let G be

$$S \to A^t \bar{A} \mid B^t \bar{B}$$

$$A \to c$$

$$B \to c$$

$$\bar{A} \to e\bar{A}e \mid A$$

$$\bar{B} \to e\bar{B}e \mid b$$

We first show that G is LR(k,t) for no value of k. To do this, we need simply consider the sentential forms $c^t e^k a e^k$ and $c^t e^k b e^k$. Each of the c's is a phrase and should be reduced to either an A or a B depending on whether the embedded character is an a or b. However, this embedded character can be surrounded by arbitrarily many e's and is thus effectively shielded from the questionable phrases. Put more simply, the first t phrases of both sentential forms look the same from the viewpoint of a bounded lookahead parser.

On the other hand, G is certainly LR(1,t+1) because the t+1 phrase of any sentential form of G is "self identifying" and can be reduced until it is adjacent to the series of c's.

q.e.d.

Theorem 2.28 There exist LR(k,∞) grammars which are not LR(k,t) for any finite k or t.

Proof Two proofs can be given, each of which lends a different insight into the LR process.

Direct Proof Consider the grammar G_{12}:

$$S \rightarrow Aa \mid Bb$$
$$A \rightarrow \bar{A}A \mid \bar{A}$$
$$B \rightarrow \bar{B}B \mid \bar{B}$$
$$\bar{A} \rightarrow c$$
$$\bar{B} \rightarrow c$$

Suppose that G_{12} were LR(k,t) for some finite values of k and t. Let us examine the sentential forms $c^{k+t}a$ and $c^{k+t}b$. Each of the initial c's is a phrase but none of the first t such phrases can be reduced based on its k rightmost neighbors (they're all c's!). Hence G is not LR(k,t) for any k and t.

G_{12} is however LR(1,∞) and in fact is also FPFAP(1). The parsing process steps over all handles until it reaches the rightmost phrase (i.e. c), which it reduces to either $\bar{A}$ or $\bar{B}$ as appropriate. The remaining c's can then be reduced (from right to left). After this parsing proceeds trivially.

Indirect Proof In Theorem 2.13 we gave an example of an LR(k,∞) grammar G_{10} which generated a language which was not a deterministic language. Clearly then by Theorem 2.24 G_{10} could not be LR(k,t) for any finite values of k or t.

q.e.d.

We point out here that the above proof also shows that the LR(k,∞) languages strictly include the LR(k,t) languages.

Chapter 3 INEXACT CONTEXT METHODS

The biggest drawback of the parsing techniques discussed so far is that they tend to produce PFA's with large numbers of states. This is essentially due to the fact that they at all times maintain exact "knowledge" of the structure of the string they are processing. It frequently is the case, however, that not all this information is necessary. Many existing parsing schemes are quite "myopic" and yet produce successful parsers. The simple precedence method [WW 66], for example, bases all decisions on the relationship between successive pairs of characters and retains no other knowledge of the string being parsed. Accordingly, we devote this chapter to the investigation of certain simplifications which can be made to our methods while still retaining significant parsing power.

3.1 DeRemer's Simplification of the LR(k) Technique

Attempts to apply the LR(k) method in its original form to "practical" grammars are usually doomed to failure unless a value of k = 0 is used. This is because the upper bound on the number of states in an LR(k) parsing machine varies exponentially with k. Thus, for example, attempts to generate LR(1) parsers for ALGOL 60 usually are aborted after many thousands of states have been generated [An 72]. LR(0) parsers for practical grammars on the other hand, tend to have a reasonable number of states (usually a number of the same order as the number of productions). Unfortunately very few grammars are LR(0) or even can be modified to fit into the LR(0) mold. This realization led to the work of DeRemer [De 69] which can be viewed as a technique which "beefs up" the power of the LR(0) parsing algorithm in such a way as to make it applicable to a wide class of practical grammars. The method enjoys the same advantage as the LR(0) technique, namely it employs a small set of states.

The essential idea of the SLR(k) construction is as follows. We start by building the LR(0) machine for the grammar in question. Certain of the states of this machine will be inadequate, that is, the machine will not be able to tell whether or not to make a certain reduction based only on the string seen so far. (Remember, k = 0 means that no right context is used.) We now try to resolve the difficulty by peeking at the right context and comparing it with those characters which can legitimately follow (according to the NEXT relation) the left hand side of the "questionable" reduction. To illustrate this point, consider

G_{13}: $S \to aAa$

$S \to bAb$

$S \to acc$

$A \to c$

The LR(0) machine for G_{13} has the following states (using Earley's notation):

(1) $S' \to .S$
$S \to .aAa$
$S \to .bAb$
$S \to .acc$

(2) $S \to a.Aa$
$S \to a.cc$
$A \to .c$

(3) $S \to b.Ab$
$A \to .c$

(4) $S \to ac.c$
$A \to c.$

(5) $S \to acc.$

(6) $S \to aA.a$

(7) $S \to aAa.$

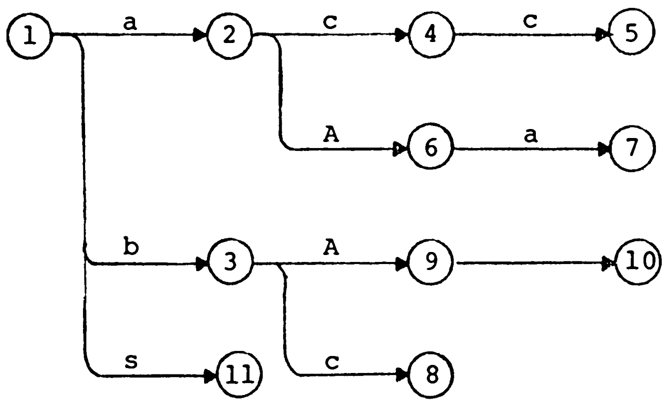
(8) $A \to c.$

(9) $S \to bA.b$

(10) $S \to bAb.$

(11) $S' \to S.$

The transition diagram for this machine is



State 4 is an inadequate state, because in a sentential form ϕ = ac..., we cannot tell whether to reduce the c to an A. However, the NEXT relation can help us. A necessary condition for the reduction $A \to c$ being correct in ϕ is that the next character of ϕ be an element of $NEXT_1(A)$. Furthermore, this condition is sufficient to resolve the inadequacy of state 4, because the only other transition out of state 4 is made on reading the character c which is not an element of $NEXT_1(A) = \{a,b\}$. We present the final form of the SLR(1) PFA for G_{13} in Figure 3.1.

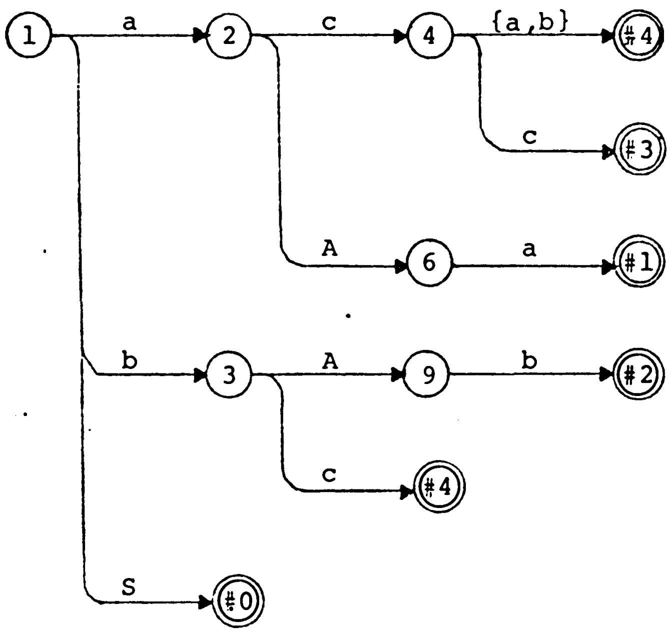


Figure 3.1 The SLR(1) PFA for G_{13}.

We have changed the names of the terminal states to reflect the reduction to be applied at that point. Furthermore, a set appearing on a transition arrow indicates a so-called "lookahead transition", that is, in state 4, if the next character of input is either an a or a b we do not advance the scanner but instead simply go to the reduce state for production 4.

It is appropriate at this point to introduce a formal definition of SLR(k) grammars. Unfortunately, no clean grammatical definition seems to exist because SLR(k) grammars are defined operationally, that is, an SLR(k) grammar is one for which a certain construction yields a parser. For our purposes, an intuitive definition is really all that we need. The reader should, however, be aware of our laxness when we say

> A grammar G is <u>SLR(k)</u> if in any right sentential form we can uniquely determine the handle by
>
> a) examining its left context to produce a list of productions which could possibly be reduced at the present point.
>
> b) choosing between these alternatives by comparing the next k characters of input with the sets of strings $NEXT_k(A_i)$ for each i on our list of suspected handles.

Notice that in some sense, the left and right contexts are used independently here. $NEXT_k(A_i)$ can only provide us with an <u>approximation</u> to the set of characters which can actually occur in the sentential form being parsed. The inexactness of this approximation is the main reason why the SLR(k) parsing technique is less general than the LR(k) technique. Let us examine the complete set of parsing patterns induced by the machine of Figure 3.1. These patterns are:

$(\vdash ac\#a,4)$

$(\vdash ac\#b,4)$

$(\vdash acc\#,3)$

$(\vdash aAa\#,1)$

$(\vdash bab\#,2)$

$(\vdash bc\#,4)$

$(\vdash S\#,0)$

Notice in particular that the second pattern $(\vdash ac\#b,4)$ can never apply to a sentential form of G_{13}. This phenomenon occurs because of the independence of the left and right contexts for state 4. Keep in mind that this does not make this pattern incorrect in the sense of Theorem 1.1. Since the pattern is never applicable to a sentential form, it can never make an error on a sentential form. The SLR(k) technique then, is our first example of a class of parsing methods which we call inexact context methods because they "lose track" of the exact structure of their input strings. Such a "forgetfulness" is simultaneously to their advantage and to their disadvantage. Fewer states are used by these methods because of their inexactness. On the other hand, we occasionally throw away so much information about the string being parsed that we no longer know whether or not to make a reduction.

As an example of these ideas consider the grammar

G_{14}:

$$S \rightarrow aEa$$
$$S \rightarrow bEb$$
$$S \rightarrow aDb$$
$$S \rightarrow bDa$$
$$D \rightarrow c$$
$$E \rightarrow c$$

This grammar is LR(1) but not SLR(1) because in the sentential form acb the context characters a on the right side and b on the left are used independently to determine whether to reduce the c to a D or E. In effect, the SLR(1) machine has "forgotten" the initial a because the strings ac and bc both take the machine into the same state (the reader should verify that this does not occur in the LR(1) machine).

Let us record here, a well known fact.

<u>Theorem 3.0</u> Every SLR(k) grammar is an LR(k) grammar but not conversely.

In the next few sections, we will apply the simplifying ideas of the SLR(k) technique to the parsing algorithms developed in Chapter 2.

3.2 The SLR(k,t) and SLR(k,∞) Methods

Let us attempt to extend the SLR technique to allow non-canonical parsing, that is, the reduction of phrases which might not be handles. Accordingly, we compare the definitions of LR(k), SLR(k) and LR(k,t) grammars to derive the following crude definition.

A grammar is SLR(k,t) if in any sentential form one of the leftmost t phrases is uniquely distinguished by

1) examining its left context to determine which phrases might occur at the current point.
2) choosing between these alternatives by comparing the first k characters of right context with the sets of string which stand in the $NEXT_k$ relation to the left hand sides of those productions selected in part one.

As always, if any uncertainty arises we postpone the reduction decision and continue scanning in the hope of being able to make a reduction elsewhere. We can, of course, continue the analogy by defining SLR(k,∞) grammars in which we require only that some phrase somewhere in the string be uniquely identifiable. The above definitions seem to be the "natural" extension of the SLR(k) concept. Unfortunately, as we will soon see, this generalization yields rather disappointing results.

Let us consider

G_{15}:

$$S \to aEA$$
$$S \to bEB$$
$$S \to aD\overline{B}$$
$$S \to bD\overline{A}$$
$$D \to c$$
$$E \to c$$
$$A \to a$$
$$\overline{A} \to a$$
$$B \to b$$
$$\overline{B} \to b$$

Notice that $L(G_{15}) = L(G_{14})$ and as a matter of fact G_{15} covers G_{14}. Suppose that we wish to parse the string acb using the SLR(1,2) technique. Reading the string from left to right, we first scan an a and then a c. At this point, the left context (i.e. the string we have already scanned) is ac. This tells us immediately that either production 5 or 6 should be reduced here (i.e. the c should become either a D or an E). However, the SLR framework requires us to choose between these alternatives by comparing the next input character (viz. b) with $NEXT_1(D)$ and $NEXT_1(E)$. Since b is an element of both these sets, we cannot make a reduction now and must instead scan on. Accordingly, we step over the b and our left context becomes acb which immediately tells us that the b should be reduced to a $\overline{B}$. Accordingly, we make this reduction and the sentential form becomes $ac\overline{B}$. Now that the right context of the c has changed, we can resolve the difficulty of which way to reduce it, because $\overline{B} \in NEXT_1(D)$ but $\overline{B} \notin NEXT_1(E)$. So we reduce the string to $AD\overline{B}$ and then to S and we are done parsing. Thus G_{15} is SLR(1,2) and everything seems rosy.

Unfortunately, a formal implementation of this method will reveal a glaring flaw. We have seen that it is always necessary to have exact knowledge or control over the left context string. In order to compute this information we must maintain almost as much information in our item sets as do the LR(k,t) or LR(k,∞) methods. This is because whenever a reduction is postponed, the inexact right context information provided by the $NEXT_k$ relation must be discarded and replaced instead with exact context information. This exact information can only be provided by using "complete" item sets. Thus it is that for the most part, our parser must maintain exact right context information in its item sets, even though the general philosophy of the method forbids full utilization of this knowledge. This criticism does not, of course, apply to the SLR(k) family of parsers because the difficulty only arises when the reduction of a suspected phrase is postponed. Since we never make postponements in the SLR(k) method, the difficulty simply never arises.

We illustrate these ideas by presenting

Algorithm 3.1 [The SLR(k,∞) Parsing Algorithm]

Assume the input is a string $\phi = X_1 \ldots X_m \in \vdash^k V^* \dashv^k$.

1	[Initialize]	set $n = 0$, $W = \{\vdash^{k+1} S]_0 \dashv^k\}$.
2	[Shift]	set $n = n + 1$. if $n + k > m$ then halt, G is not SLR(k,∞). set $W' = \{\gamma \mid \exists\alpha \in B^*,\ Y \in V,\ \alpha Y\gamma \in W\}$.
3	[Close]	set $W'' = W' \cup \{\beta_1 V_1 \ldots \beta_k x_i]_i m(\gamma) \mid \beta_1 V_1 \ldots \beta_k V_k \gamma \in W''$ and $V_k = A_i\}$.
4	[Intersect]	set $W = \{\beta_1 V_1 \ldots \beta_k V_k \gamma \in W'' \mid V_1 = X_{n+1}, \ldots, V_k = X_{n+k}\}$.
5	[Error?]	if $W = \emptyset$, halt, $\phi \notin SF(G)$.
6	[SLR(k,∞)?]	if $W \cap V^* \neq \emptyset$, halt, G is not SLR(k,∞).
7	[Lookahead]	set $Z_i = NEXT_k(A_i)$ if $W \cap]_i (V \cup B)^* \neq \emptyset$ $= \emptyset$ otherwise set $\overline{Z} = TRUNC_k(W \cap V(V \cup B)^*)$.
8	[Continue?]	if $X_{n+1} \ldots X_{n+k} \in \overline{Z}$, go to 2. if $X_{n+1} \ldots X_{n+k} \in Z_i \cap Z_j$ for $i \neq j$, go to 2.
9	[Reduce]	since $X_{n+1} \ldots X_{n+k} \in Z_i$ for a unique i, halt indicating production i should be reduced at position n.

This algorithm is essentially a combined version of algorithms 2.1 and 2.3 modified to utilize the SLR(k) strategy. The sets W, W' and W" represent sets of items describing the local structure of the string. As parsing proceeds, these sets can actually become infinite. This is of little concern to us as these sets will in fact be regular and the necessary computations upon them can be performed in a finite amount of time.

In step 6, we test for the presence of an item which contains no further brackets, indicating we have bypassed all parsing opportunities in ϕ. This test was, of course, used in the same way in the LR(k,∞) and LR(k,t) parsing algorithms.

In step 7, we are applying the essential SLR(k) idea, namely using our left context to predict which productions might have phrases at the position n in ϕ. Thus Z_i is nonempty if and only if production i might so generate a phrase. In step 8, we compare the first k characters of right context with the appropriate sets of strings. Notice how the $NEXT_k$ relation has been used here. Note further that this test can never be as "sharp" as the test used by the LR(k,∞) method, namely, direct comparisons with the items in W. As always, conflicting results cause us to postpone these reductions and continue scanning.

Several routine theorems concerning this algorithm are evident.

<u>Theorem 3.2</u> Algorithm 3.1 always terminates and is correct.

<u>Proof</u> By "correct" we mean here that the algorithm never makes an incorrect reduction in the sense of Theorem 1.1. We also mean that the algorithm halts in step 5 only if $\phi \notin SF(G)$. The formal proof of the theorem follows the lines of Theorems 2.4 and 2.5.

<u>Theorem 3.3</u> Every SLR(k,∞) grammar is LR(k,∞). However, there are LR(k,∞) grammars which are not SLR(k,∞).

<u>Proof</u> The inclusion of SLR(k,∞) in LR(k,∞) is obvious from Algorithm 3.1. To see that the inclusion is proper, one need only consider grammar G_{14} which is LR(1,∞) but not SLR(k,∞) for any value of k.

q.e.d.

It should be obvious that the SLR(k,∞) algorithm can be "trimmed down" to produce an SLR(k,t) parsing algorithm. This process essentially is the same as was used to produce the LR(k,t) algorithm from the LR(k,∞) algorithm. However, a few minor points merit discussion. In particular, we need never maintain any characters to the right of the <u>tth</u> bracket in any item. (Since we will never step over this bracket, we will never need exact context information to its right. It might well be the case that we will need approximate context information, but this will, as always, be provided by the $NEXT_k$ relation.) Using the above strategy, it may happen that some items will be generated having a length less than k. Such items will always end in a bracket (which will be the tth bracket). For purposes of making the necessary right context tests, we can recover (approximately) the missing characters by using the $NEXT_k$ relation for the left hand side of whatever production corresponds to this final bracket.

We point out here that the SLR(k,t) algorithm produced above becomes exactly the SLR(k) method of DeRemer [De 69] when a value of $t = 1$ is chosen.

We will not bother to even list any theorems about the SLR(k,∞) and SLR(k,t) classes because for the most part they and their proofs just carry over from Sections 2.4 and 2.6. However, it is appropriate to comment upon the fundamental unsoundness of generalizing the SLR(k) method in the manner done in this section.

In the case of the SLR(k,∞) algorithm, the generated PFA will have a set of states which will, in general, properly include the state set of the LR(k,∞) PFA for the same grammar and value of k. However, as Theorem 3.3 shows the SLR(k,∞) method is actually weaker in power than the LR(k,∞) method. Accordingly, we must include the SLR method in our collection of nice ideas which do not generalize nicely.

Things are not quite so bad in the case of SLR(k,t) parsers. Here we can in fact save a few states over the corresponding LR(k,t) parser simply because we do not need to carry around the right context for the tth phrase. However, whenever $t > 1$, the same general criticism can be made as was made of the SLR(k,∞) method. That is, the SLR(k,t) item sets actually contain more accurate information than is provided by the $NEXT_k$ relation. To ignore this information simply to stay within the "spirit" of the SLR(k) method is foolhardy. Accordingly, we must continue our search for a "clean" application of DeRemer's ideas to non-canonical parsing.

3.3 The REDNEXT Method

In this section we will present another parsing method motivated by the SLR technique. The intuitive idea is to construct sets of states similar to LR(0) state sets as we scan the input from left to right. Upon reaching an inadequate state, we attempt to resolve the local ambiguity by using the $NEXT_k$ relation. So far, our method is just the SLR(k) method. However, if it now turns out that the state is still inadequate, then we postpone the reduction and move on to phrases further to the right. The crucial problem is deciding what characters can follow the "ambiguous" phrase. In the $LR(k,\infty)$ and $LR(k,t)$ methods (as well as the $SLR(k,\infty)$ and $SLR(k,t)$ methods) this problem was solved by examining the "reduced right context" string associated with each item in the state and thereby determining all possible continuations of the sentential form being parsed. In the method about to be presented, we will approximate these reduced right context strings by using the REDNEXT relation. We will thus achieve a saving with respect to the amount of information stored in each state and consequently be able to generate parsers which have the ability to postpone an unbounded number of reductions while still using a bounded number of states. The price paid for this privilege will as always, be a loss of parsing generality.

The closure process used to produce item sets will proceed as follows. Suppose, that $\phi A_i \psi$ is an item to be "closed".

We generate new items $\phi x_i]_i y$ where $y \in \text{REDNEXT}_k(A_i)$. Notice that we have completely thrown away the right context string ψ. It will always be true however, that

$m(\psi) \in \bigcup_{1 \leq j \leq k} \text{REDNEXT}_j(A_i)$. Thus we have approximated the "true" right context of A_i by a bigger set of "possible" right contexts. It will become clear later that this process can never cause us to make an erroneous reduction but can in fact cause us to bypass correct reductions. This observation is also true for other "inexact context" methods such as SLR(k), SLR(k,t) and SLR(k,∞) and is the chief reason why they lose parsing power.

We offer G_{16} below as an example of a grammar parsable by this technique.

G_{16}:

$$S \rightarrow Ag \mid Bh$$
$$A \rightarrow aAG \mid aG$$
$$B \rightarrow aBHH \mid aHH$$
$$G \rightarrow b$$
$$H \rightarrow b$$

$$L(G_{16}) = \{a^n b^n g \mid n \geq 1\} \cup \{a^n b^{2n} h \mid n \geq 1\}$$

Let us use a lookahead factor of k = 1 and list the relations we will need.

	NEXT_k	REDNEXT_k
S	$\dashv$	$\dashv$
A	g,G,b	g,G

	$NEXT_k$	$REDNEXT_k$
B	h,H,b	h,H
G	g,G,b	g,G
H	h,H,b	h,H

The first item set for our parser will contain the item $S]_0\dashv$. Proceeding with the closure, we generate $Ag]_1\dashv$ and $Bh]_2\dashv$ (because $REDNEXT_k(S) = \{\dashv\}$, a singleton set). Closing the item $Ag]_1\dashv$ yields $aAG]_3g$ and $aAG]_3G$ as well as $aG]_4g$ and $aG]_4G$ because both G and g are in $REDNEXT_1(A)$. Notice how we have already introduced "spurious" items into the item set, namely $aAG]_3G$ and $aG]_4G$ which describe the initial characters of no sentential form. Such is the danger of approximation. We present below the complete first item set.

1 $S]_0\dashv$

$Ag]_1\dashv$	$Bh]_2\dashv$
$aAG]_3g$	$aBHH]_5h$
$aAG]_3G$	$aBHH]_5H$
$aG]_4g$	$aHH]_6h$
$aG]_4G$	$aHH]_6H$

If our input character is an a then we shift and produce the basis set for item set 2 (shown above the line in the figure below). The complete item set for state 2 is:

2

$AG]_3 g$	$BHH]_5 h$
$AG]_3 G$	$BHH]_5 H$
$G]_4 g$	$HH]_6 h$
$G]_4 G$	$HH]_6 H$

$b]_7 g$	$b]_8 h$
$b]_7 G$	$b]_8 H$
$aAG]_3 g$	$aBHH]_5 h$
$aAG]_3 G$	$aBHH]_5 H$
$aG]_4 g$	$aHH]_6 h$
$aG]_4 G$	$aHH]_6 H$

Once again, spurious items have crept into the item set. For example, 2 contains the item $b]_8 h$ but no sentential form of G_{16} begins with abh. Scanning a b in state 2 results in item set 3 whose basis set is described above the line in the figure below.

3

$]_7 g$	$]_8 h$
$]_7 G$	$]_8 H$

$]_7 b]_7 g$	$]_8 b]_8 h$
$]_7 b]_7 G$	$]_8 b]_8 H$

If the next input character were a g we would of course reduce production 7 since all items in 3 whose first real character is a g agree that this reduction should be made. On the other hand, if our next input character were a b we would have to postpone our decision to reduce because we do not know whether to reduce production 7 or 8. Continuing the scan, the next basis set would then be

$$]_7 g \qquad]_8 h$$
$$]_7 G \qquad]_8 H$$

which is exactly the basis set for item set 3 . Hence the parsing machine enters a loop, stepping over successive b's until finally a g,G,h or H is encountered. The complete parsing automaton induced by this technique is displayed in Figure 3.2. The corresponding sets of items are shown in Figure 3.3.

We will call the grammars parsable by this technique the RNP(k) grammars (i.e. REDNEXT parsable using k characters of lookahead). A grammar will fail to be RNP(k) if in some generated inadequate item set, there is an item whose first k real characters are fully reduced. This is the same failure criterion used for the LR(k,∞) grammars and corresponds to the case in which bypassing a reduction will do us no good because the right context information for the suspected phrase is already fully reduced and hence will never change as a result of a reduction performed still further to its right.

Intuitively, it is easier for a grammar to be LR(k,∞) than RNP(k) because the spurious items introduced into the RNP item sets can cause conflicts to occur and postpone those reductions which the LR(k,∞) method is in fact "smart" enough

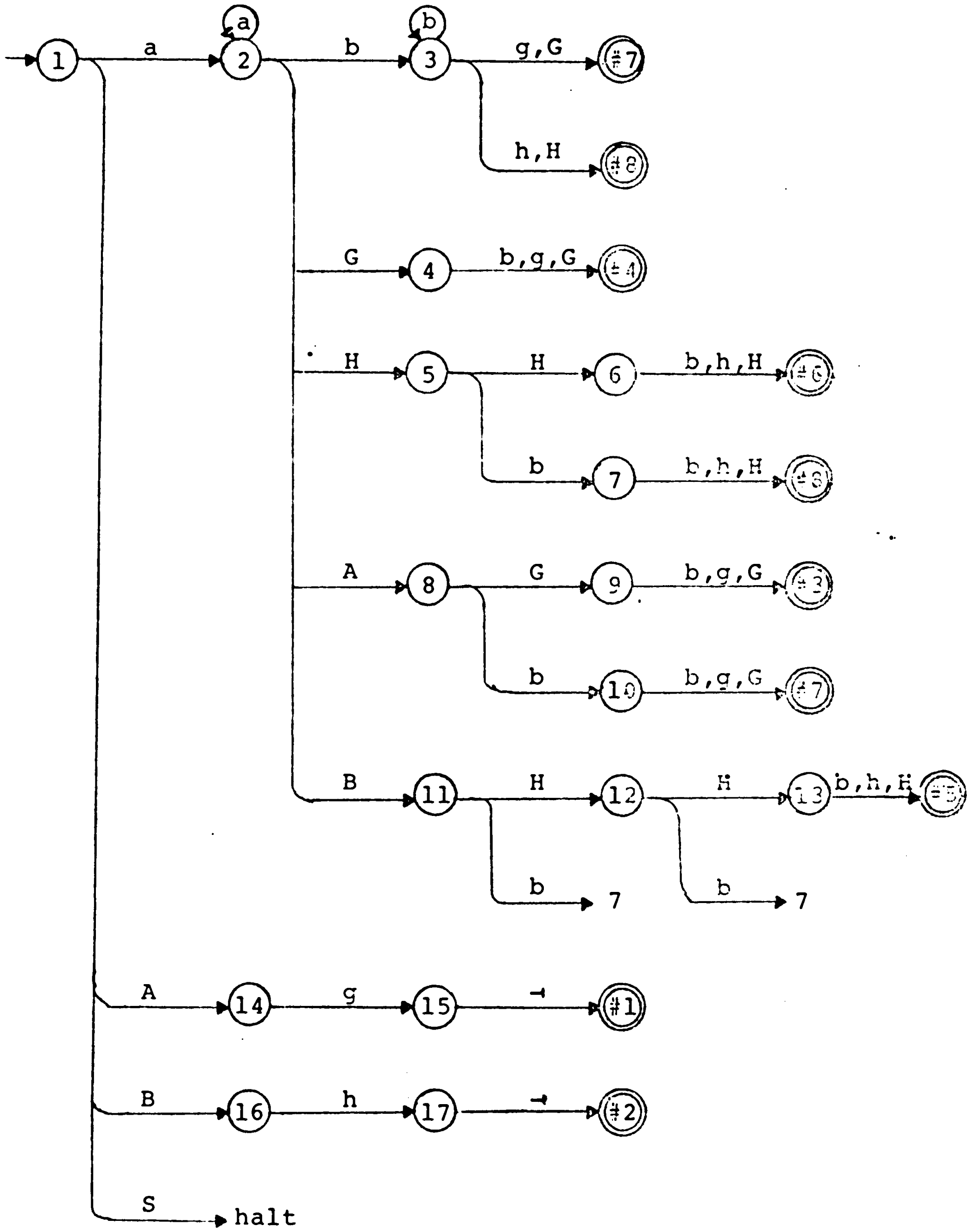


Figure 3.2 A PFA for G_{16}

(1)

$S]_0\dashv$	
$Ag]_1\dashv$	$Bh]_2\dashv$
$aAG]_3g$	$aBHH]_5h$
$aAG]_3G$	$aBHH]_5H$
$aG]_4g$	$aHH]_6h$
$aG]_4G$	$aHH]_6H$

(2)

$AG]_3g$	$BHH]_5h$
$AG]_3G$	$BHH]_5H$
$G]_4g$	$HH]_6h$
$G]_4G$	$HH]_6H$
$b]_7g$	$b]_8h$
$b]_7G$	$b]_8H$
$aAG]_3g$	$aBHH]_5h$
$aAG]_3G$	$aBHH]_5H$
$aG]_4g$	$aHH]_6h$
$aG]_4G$	$aHH]_6H$

(3)

$]_7g$	$]_8h$
$]_7G$	$]_8H$
$]_7b]_7g$	$]_8b]_8h$
$]_7b]_7G$	$]_8b]_8H$

(4)

$]_4g$	$]_4G$
$]_4b]_7g$	$]_4b]_7G$

(5)

$H]_6h$	$H]_6H$
$b]_8h$	$b]_8H$

(6)

$]_6h$	$]_6H$
$]_6b]_8h$	$]_6b]_8H$

(7)

$]_8h$	$]_8H$
$]_8b]_8h$	$]_8b]_8H$

(8)

$G]_3g$	$G]_3G$
$b]_7g$	$b]_7G$

(9)

$]_3g$	$]_3G$
$]_3b]_7g$	$]_3b]_7G$

(10)

$]_7g$	$]_7G$
$]_7b]_7g$	$]_7b]_7G$

(11)

$HH]_5h$	$HH]_5H$
$b]_8h$	$b]_8H$

(12)

$H]_5h$	$H]_5H$
$b]_8h$	$b]_8H$

(13)

$]_5h$	$]_5H$
$]_5b]_8h$	$]_5b]_8H$

(14) $g]_1\dashv$ (15) $]_1\dashv$

(16) $h]_2\dashv$ (17) $]_2\dashv$

Figure 3.3 Item sets corresponding to the states of the machine shown in Figure 3.2.

to make. Let us reiterate the point that this phenomenon occurs because the RNP(k) technique is an <u>inexact context</u> method and accordingly is "less sure" of the structure of its input string. This "uncertainty" results in more reduction postponements being made and in the extreme, gives rise to the parser postponing <u>all</u> reduction decisions. Thus the parser fails to work.

Let us now define an algorithm for parsing RNP(k) grammars. Individual items in the item set for this algorithm will be written $\beta_1 V_1 \cdots \beta_p V_p$ where each $\beta_i \in B^*$ and $V_i \in V$.

<u>Algorithm 3.4</u> [The RNP(k) parsing algorithm]

Suppose the input string is $\phi = X_1 \ldots X_m \in \vdash^k V^* \dashv^k$.

1. set $n = 0$ and $W = \{\vdash^{k+1} S]_0 \dashv^k\}$.

2. set $n = n + 1$.
   if $n + k > m$, the halt (G is not RNP(k)).
   set $W' = \{\beta_2 V_2 \cdots \beta_p V_p \mid \exists \beta_1, V_1 \ni \beta_1 V_1 \cdots \beta_p V_p \in W\}$.

3. set $$W'' = W' \cup \left\{ \begin{array}{c} \beta_1 V_1 \cdots \beta_{k-1} x_i]_i y \mid \exists p, \beta_k, \ldots, \beta_p, V_{k+1}, \ldots, V_p \\ \text{such that } \beta_1 V_1 \cdots \beta_{k-1} A_i \beta_k V_{k+1} \cdots \beta_p V_p \in W'' \\ y \in \mathrm{REDNEXT}_k(A_i) \end{array} \right\}$$

4. set $W = \{\beta_1 V_1 \cdots \beta_p V_p \in W'' \mid V_1 = X_{n+1}, \ldots, V_k = X_{n+k}\}$.

5. if $W = \emptyset$, then halt ($\phi \notin SF(G)$).

6. if every item in W contains the common initial bracket $]_i$ (i.e. for each item, $\beta_1 \in B^*]_i B^*$) then halt indicating

that production i should be reduced at position n of ϕ.

7 if $\exists \beta_1 V_1 \ldots \beta_p V_p \in W$ such that $\beta_1 = \ldots = \beta_p = \varepsilon$ then halt (G is not RNP(k)).

8 go to step 2.

Lemma 3.5 Suppose G is RNP(k) and W is an item set which arises during the execution of Algorithm 3.4. Let I = $\beta_1 V_1 \ldots \beta_p V_p \in W$. Then $p \leq 2k + q - 1$ where q is the length of the longest right hand side of a production in G.

Proof By induction on the number of steps used to produce I. The only place in the algorithm where a longer item can be produced is step 3 where the closure is being taken. However, the closure operation by its very definition always produces exactly $2k + q - 1$ characters where q is now the length of the right hand side of the production being applied.

q.e.d.

We can now proceed to bound the size of each β_i which occurs. In the absence of ε- rules, clearly each β_i is of length 0 or 1. In the presence of ε-rules however, the β's can in fact grow unboundedly in length. We point out here that we never need include more than 1 instance of any $]_i$ in a β because we are only interested in whether or not $]_i$ can occur between two particular characters of the input string.

Thus if we have p productions in G, none of the β's need ever grow larger than p characters in length. Thus we have

<u>Lemma 3.6</u> The total number of RNP(k) items for a grammar G is bounded by

$$[2^p(v+1)]^{2k+q-1}$$

where p = the number of productions in G

v = the number of vocabulary symbols in G

q = the length of the longest right side of a production in G.

<u>Theorem 3.7</u> The RNP(k) parsing algorithm can be implemented on a 2PDA which parses in linear time. Furthermore, it is decidable whether an arbitrary grammar G is RNP(k) for a given value of k.

<u>Proof</u> Since the number of RNP(k) items is bounded (see previous lemma), so too is the number of possible RNP(k) item <u>sets</u>. Hence, we can construct all transitions between these item sets, reduce the resulting finite state automaton, and test if a non-RNP(k) item set (as defined by steps 7 and 8) is accessible from the start state. This gives us the desired decision procedure. The first part of the theorem (dealing with implementation issues) follows from Theorem 1.10.

q.e.d.

<u>Theorem 3.8</u> The RNP(k) parsing algorithm is correct.

<u>Proof</u> The proof essentially mirrors Theorem 2.5. The crucial observation is that the RNP(k) item sets contain all valid items (plus some invalid items) describing the local structure of the input string. Hence the RNP(k) algorithm will never make a reduction prohibited by the presence of conflicting valid items. Thus any parse produced by the algorithm is correct. For the same reason, if the algorithm ever generates an empty set of items and halts in step 5, then the input string is not a sentential form. If it were, there would be a representative of some phrase bracketing for the string present in the item set.

q.e.d.

3.4 The FPFAP(k) Technique

Although the RNP(k) technique is rather simple conceptually, it frequently fails to yield parsers for some rather simple grammars. Consider for example, grammar

G_{17}:

$$S \rightarrow A \mid B$$
$$A \rightarrow EEA \mid EE$$
$$B \rightarrow OOB \mid O$$
$$E \rightarrow a$$
$$O \rightarrow a$$

$$L(G_{17}) = (aa)^* \cup (aa)^*a$$

Let us attempt to construct a RNP(1) parser for G_{17}. We list the pertinent relations for reference.

	$NEXT_1$	$REDNEXT_1$
S	$\dashv$	$\dashv$
A	$\dashv$	$\dashv$
B	$\dashv$	$\dashv$
E	$E,A,a,\dashv$	$E,A,\dashv$
O	$O,B,a,\dashv$	$O,B,\dashv$

Item set construction proceeds as follows:

1

$S]_0\dashv$	
$A]_1\dashv$	$B]_2\dashv$
$EEA]_3(E,A,\dashv)$	$OOB]_5(O,B,\dashv)$
$EE]_4(E,A,\dashv)$	$O]_6(O,B,\dashv)$
$a]_7(E,A,\dashv)$	$a]_8(O,B,\dashv)$

Scanning an a, we proceed to state (2) whose basis is:

$]_7(E,A,\dashv)$ $]_8(O,B,\dashv)$

Clearly this state will be inadequate if the next character is $\dashv$. Furthermore, the reader may easily verify that using larger values of k will not help either. Thus G_{17} is not RNP(k) for any finite k. This failure is disappointing because it can be remedied by the simple expediency of counting modulo 2, the number of a's already scanned. Unfortunately, the RNP(k) method does not do this, its right context is completely independent of its left context.

On the other hand, the LR(k,∞) method, when applied to G_{17} generates only a finite number of item sets as illustrated in Figure 3.4. These item sets correspond to the states of the PFA shown in Figure 3.5. The crucial point here is that although the RNP(k) construction has failed, there does in fact exist a simple parser for G_{17}. What we would like to do next is to characterize the class of grammars for which there are "simple" parsers. More specifically, we wish to devise a parsing scheme which combines the good points and avoids the faults of some of the previous parsing methods examined. In particular, a left to right scan using some bounded amount of lookahead should be employed as this will guarantee linear parsing time. Decision postponement (i.e. non-canonical parsing) seems to be advantageous although it does necessitate the use of two stacks for parsing. Finally and most impor-

(1) $S]_0\dashv$

$A]_1\dashv$ $B]_2\dashv$

$EEA]_3\dashv$ $OOB]_5\dashv$

$EE]_4\dashv$ $O]_6\dashv$

$a]_7EA\dashv$ $a]_8OB\dashv$

$a]_7E\dashv$ $a]_8\dashv$

(2) $]_7EA\dashv$ $]_8OB\dashv$

$]_7E\dashv$ $]_8\dashv$

$]_7a]_7A\dashv$ $]_8a]_8B\dashv$

$]_7a]_7\dashv$

(3) $]_7A\dashv$ $]_8B\dashv$

$]_7\dashv$

$]_7EEA]_3\dashv$ $]_8OOB]_6\dashv$

$]_7EE]_4\dashv$ $]_8O]_6\dashv$

$]_7a]_7EA\dashv$ $]_8a]_8OB\dashv$

$]_7a]_7E\dashv$ $]_8a]_8\dashv$

(4) $OB]_5\dashv$ $]_6\dashv$

$a]_8B\dashv$

(5) $]_8B\dashv$

$]_8OOB]_5\dashv$ $]_8O]_6\dashv$

$]_8a]_8OB\dashv$ $]_8a]_8\dashv$

(6) $B]_5\dashv$

$OOB]_5\dashv$ $O]_6\dashv$

$a]_8OB\dashv$ $a]_8\dashv$

(7) $]_8OB\dashv$ $]_8\dashv$

$]_8a]_8B\dashv$

(8) $]_5\dashv$

(9) $EA]_3\dashv$ $E]_4\dashv$

$a]_7A\dashv$ $a]_7\dashv$

(10) $]_7A\dashv$ $]_7\dashv$

$]_7EEA]_3\dashv$ $]_7EE]_4\dashv$

$]_7a]_7EA\dashv$ $]_7a]_7E\dashv$

(11) $A]_3\dashv$ $]_4\dashv$

$EEA]_3\dashv$ $EE]_4\dashv$

$a]_7EA\dashv$ $a]_7E\dashv$

(12) $]_7EA\dashv$ $]_7E\dashv$

$]_7a]_7A\dashv$ $]_7a]_7\dashv$

(13) $]_3\dashv$ (14) $]_1\dashv$

(15) $]_2\dashv$ (16) $]_0\dashv$

Figure 3.4 The LR(1,∞) item sets for G_{17}.

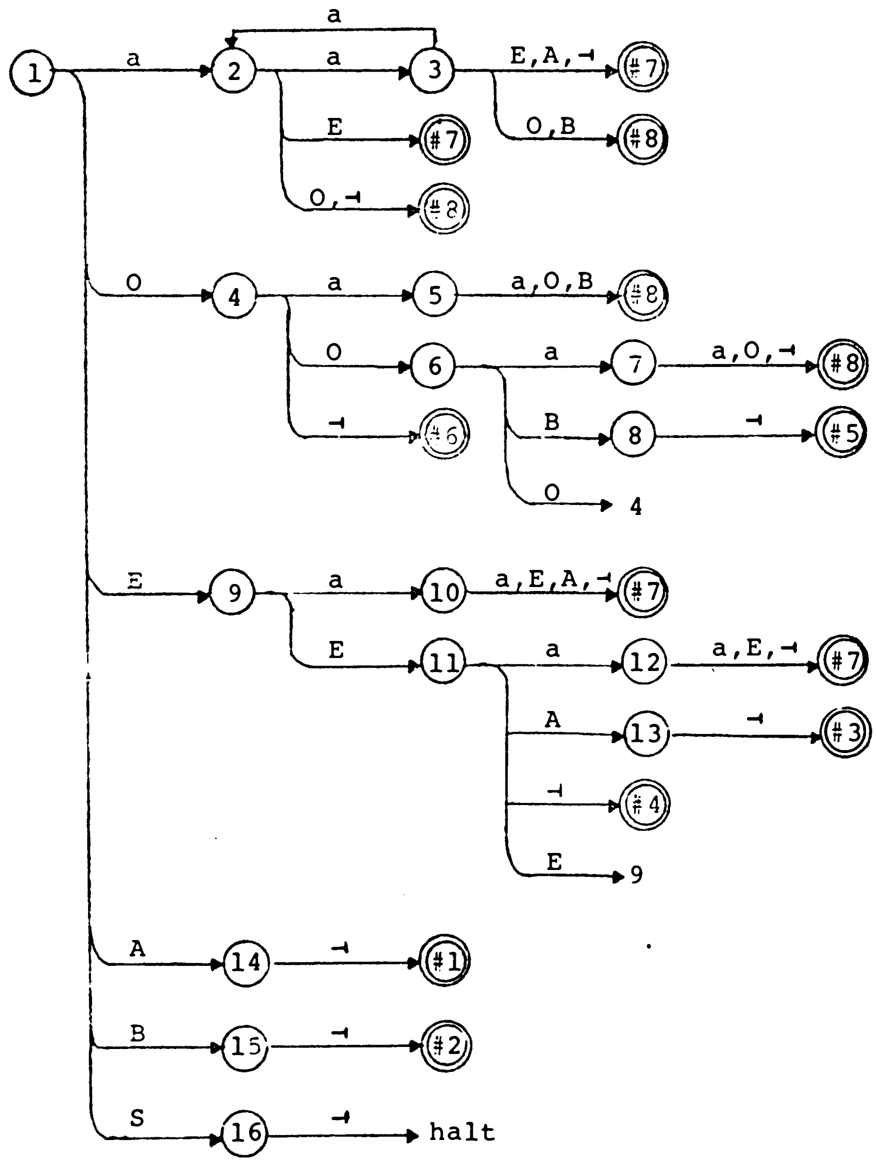


Figure 3.5 The LR(1,∞) PFA for G_{17}.

tantly, we wish our parser to utilize a finite number of states, or expressed differently, the parsing patterns induced by the method should be regular sets. We define the class formally by

> A grammar G is <u>f</u>inite <u>p</u>hrase <u>f</u>inding <u>a</u>utomaton <u>p</u>arsable using <u>k</u> characters of lookahead (FPFAP(k)) if and only if there exists a parsing scheme for G which uses a finite collection of regular k right bounded patterns.

Grammar G_{17} is an FPFAP(k) grammar as is any grammar for which the LR(k,∞) algorithm happens to yield a PFA with a finite number of states. However, the FPFAP(k) class is much broader than this as will be shown by the next example. Consider

$$
G_{18}: \quad \begin{aligned}
S &\to aAa \mid bAb \\
S &\to aDb \mid bDa \\
A &\to cA\bar{A} \mid c\bar{A} \\
D &\to cD\bar{D} \mid c\bar{D} \\
\bar{A} &\to d \\
\bar{D} &\to d
\end{aligned}
$$

$$L(G_{18}) = \{(a \cup b)c^n d^n (a \cup b) \mid n \geq 1\}.$$

The reader can easily verify that G_{18} is LR(k,∞) and furthermore that the PFA induced by the LR(k,∞) algorithm has infinitely many states. Furthermore, the RNP(k) method and the SLR(k,∞) method both fail on G_{18} because of the necessity to utilize interdependent context, that is, the parser must know whether the first and last characters of its input string are alike or different. There does however, exist a right bounded regular parsing scheme for G_{18}. The appropriate reduction patterns are:

$$(aAa\#,\ 1)$$
$$(bAb\#,\ 2)$$
$$(aDb\#,\ 3)$$
$$(bDa\#,\ 4)$$
$$(cA\bar{A}\#,\ 5)$$
$$(c\bar{A}\#,\ 6)$$
$$(cD\bar{D}\#,\ 7)$$
$$(c\bar{D}\#,\ 8)$$
$$(ac^{+}d^{+}\#(a \cup \bar{A}),\ 9)$$
$$(bc^{+}d^{+}\#(b \cup \bar{A}),\ 9)$$
$$(ac^{+}d^{+}\#(b \cup \bar{D}),\ 10)$$
$$(bc^{+}d^{+}\#(a \cup \bar{D}),\ 10)$$

A PFA corresponding to these patterns is displayed in Figure 3.6.

Loosely speaking, the FPFAP(k) technique is the ultimate "practical" bottom-up parsing method. Unfortunately we can not at this point give a terminating construction for generating FPFAP(k) parsers. We can however show

<u>Theorem 3.9</u> Let G be a context-free grammar and k be a fixed integer. Then there is a procedure which emits an

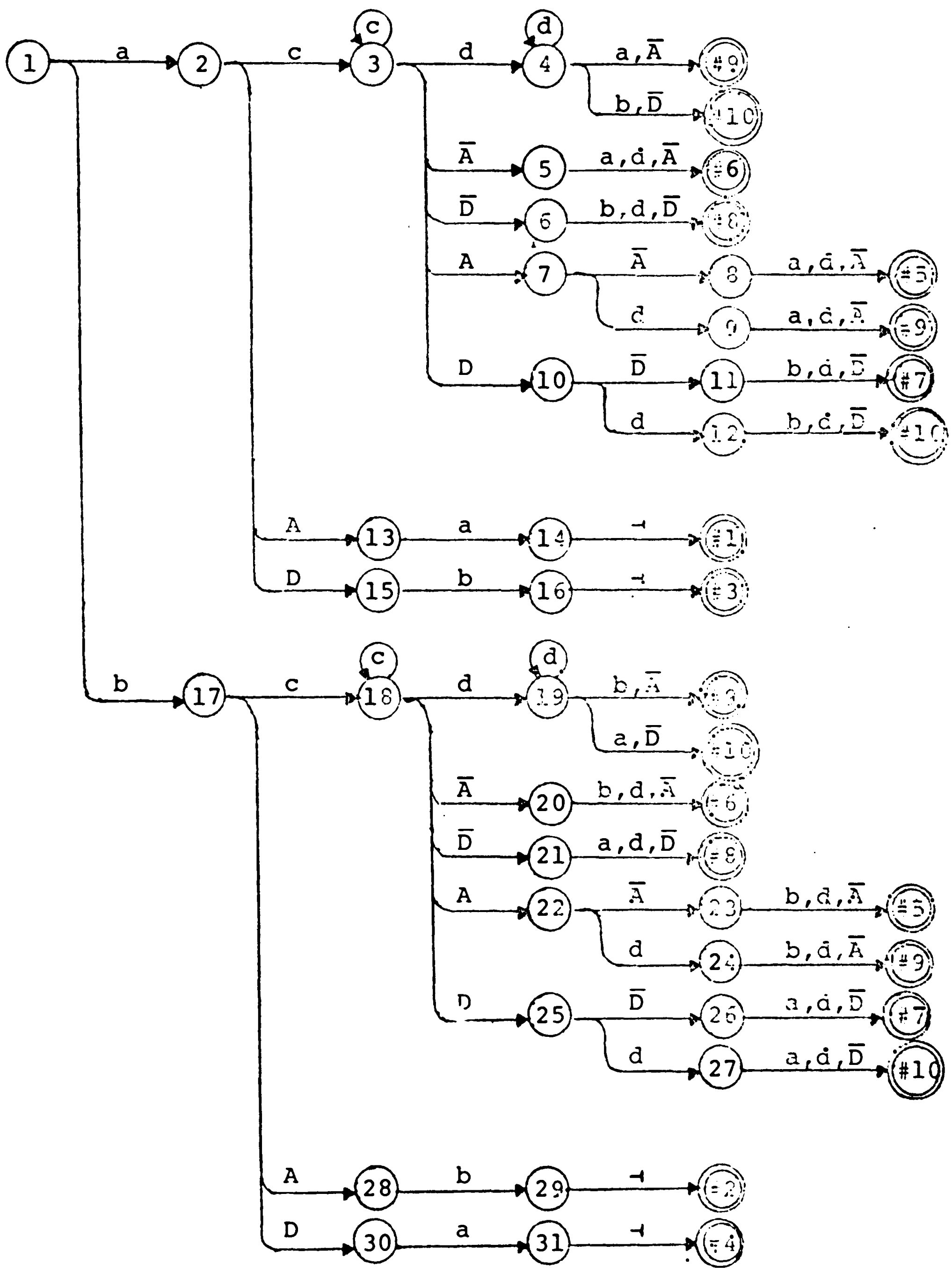


Figure 3.6 A PFA for G_{18}.

FPFAP(k) parser for G if one exists, and runs forever otherwise.

Proof The procedure simply enumerates all regular k-bounded reduction schemes using Theorem 1.6 to test each one for being a parsing scheme.

q.e.d.

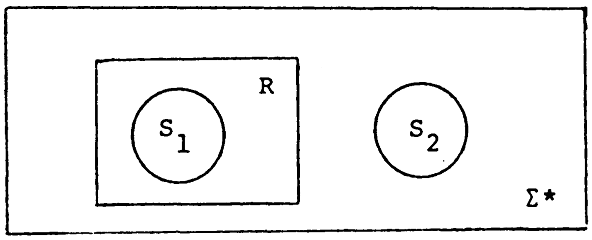
In order to guarantee termination of the above process, we need to answer the following

Open Question 3.10 [The FPFAP(k) decision problem]

Is it decidable for fixed integer k and arbitrary context-free grammar G whether G is FPFAP(k)?

Let us make a tangential approach to the FPFAP decidability problem.

Two sets S_1 and S_2 are said to be regularly separable if there exists a regular set R such that $S_1 \subseteq R$ and $S_2 \cap R = \emptyset$. Pictorially, the situation is



If S_1 and S_2 are taken to be context-free languages, then it is easy to discover instances of pairs of sets which either are or are not regularly separable. A problem which is not so readily cracked is

Open Question 3.11 [The Regular Separability Problem]

Is it decidable of two arbitrary context-free languages (two arbitrary deterministic context-free languages) whether they are regularly separable?

This question has arisen in other places [CC 70] and bears a strong relationship to the problem at hand as shown below.

Theorem 3.12 If the Regular Separability Problem for deterministic context-free languages is unsolvable, then the FPFAP(k) decision problem is unsolvable.

Proof. Suppose G_1 and G_2 are LR(k) grammars for the languages in question. We apply exactly the same construction as in Theorem 2.11 to produce a new grammar G. We next demonstrate that G is FPFAP(k) if and only if $L(G_1)$ and $L(G_2)$ are regularly separable by some set R.

Accordingly, suppose G is FPFAP(k) using some set of reduction patterns $\{R_i \mid 1 \leq i \leq p\}$. Consider now the regular set $R = \bigcup_j h(R_j)\Sigma^*$ where the j's are chosen only for those productions in class $\mathscr{P}_1$. Clearly, $L(G_1) \subseteq R$. It is also clear that $L(G_2) \cap R = \emptyset$ because then a "wrong" reduction would be implied by some pattern R_i. Hence, R separates $L(G_1)$ and $L(G_2)$.

Conversely, suppose R separates $L(G_1)$ and $L(G_2)$. This time we can use the patterns

$$(R \cap \Sigma^{*}\sigma_i \#\dashv) \quad \text{for} \quad \hat{\sigma}_i \rightarrow \sigma_i$$

and

$$(R \cap \Sigma^{*}\sigma_i \#\dashv) \quad \text{for} \quad \bar{\sigma}_i \rightarrow \sigma_i$$

in conjunction with the other patterns listed in the proof of Theorem 2.11 to parse G. But these patterns are all regular k-bounded patterns and hence G is FPFAP(k).

q.e.d.

Although the FPFAP(k) decision problem remains open for fixed k, if we allow k to be a free variable, we can prove a routine result.

Theorem 3.13 Let G be a context-free grammar. Then it is undecidable whether there exists a k such that G is FPFAP(k).

Proof Simply observe that in the space of linear grammars, the notions of LR(k) and FPFAP(k) are identical. However, it is well known [Kn 65] that it is undecidable whether a linear context free grammar is LR(k) for free k.

q.e.d.

Finally, let us put the FPFAP(k) grammars in perspective by stating several theorems related to their utility and limitations.

<u>Theorem 3.14</u> If G is FPFAP(k), then G can be parsed in linear time on a 2PDA with a finite number of states.

<u>Proof</u> Direct from the definition of FPFAP(k) and Theorem 1.10.

<u>Theorem 3.15</u> If G is LR(k,t) or RNP(k), then G is FPFAP(k).

<u>Proof</u> The parsing patterns which correspond to the PFA for either an LR(k,t) grammar or an RNP(k) grammar are k right-bounded regular sets as was shown in the proofs of Theorems 2.23 and 3.7. But then by definition, such a grammar is FPFAP(k).

q.e.d.

<u>Theorem 3.16</u> Every FPFAP(k) grammar is an LR(k,∞) grammar. However, there exist LR(k,∞) grammars which are not FPFAP(k) for any value of k.

<u>Proof</u> If G is FPFAP(k), G is pattern parsable by using a finite collection of regular k bounded patterns. But then, by Lemma 2.7 , G is also LR(k,∞). To see that this inclusion is proper, consider the grammar G_{10}. G_{10} was shown to be LR(k,∞) in Theorem 2.13. On the other hand, we can demonstrate that G_{10} is not FPFAP(k) for any finite value of k. We could prove this directly using the idea of regular separability. Instead, we will wait until Lemma 4.13 in which we will prove that $L(G_{10})$ is generated by no FPFAP(k) grammar. This fact

will, of course, complete the present theorem.

q.e.d.

Chapter 4 RELATIONSHIPS BETWEEN THE CLASSES

In this chapter we will compare the properties of the various parsing techniques defined in this thesis. In particular, we will discuss inclusions between classes of grammars and classes of languages. We will also elaborate upon certain decidability and constructability properties which apply to some of the classes.

4.1 Relations in Grammar Space

The power or generality of the various classes of grammars can be explained most easily in terms of a lattice. A class of grammars A is "greater" in the lattice sense than a class B if A $\supset$ B and the inclusion is proper. The relation of all the grammar classes mentioned in this dissertation is summarized in Figure 4.1. Notice that we restrict our attention to grammars which contain no ε-rules. The reason for this will be explained in Section 5.1.

The lattice of Figure 4.1 summarizes a myriad of relationships. Rather than prove every one of them, we instead present Figure 4.2 which provides the reader with sufficient "hints" to construct his own proofs. One simply looks up the table entry corresponding to the classes of interest. This table entry refers to the necessary theorems and example grammars for proving either strict containment or incomparability. If one class is an upper bound but not a least upper bound of the other (i.e. there is a chain of one or

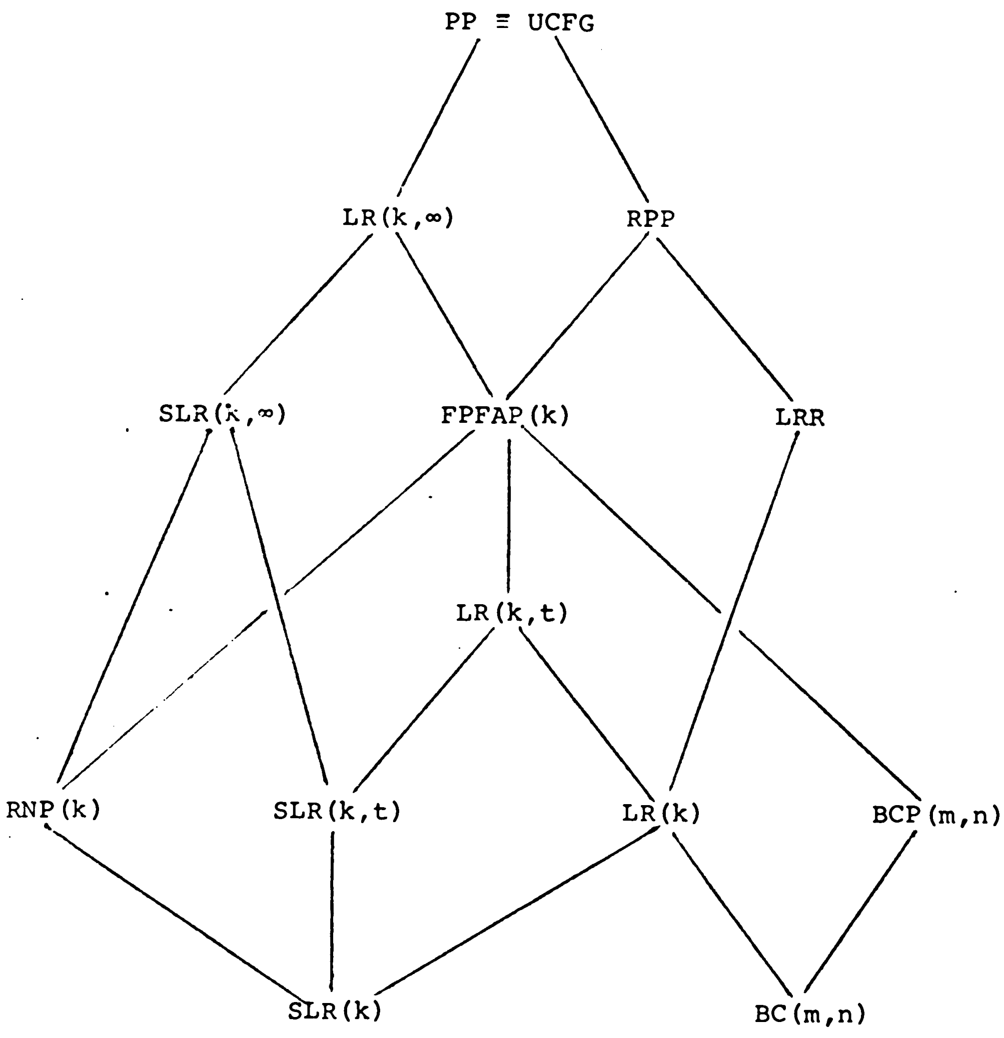


Figure 4.1 Inclusion lattice for classes of grammars without ε-rules.

	PP	LR (k,∞)	SLR (k,∞)	RPP	FPFAP	BCP	RNP	LR (k,t)	SLR (k,t)	LRR	LR(k)	SLR (k)	BC
PP													
LR (k,∞)	2.6 2.12												
SLR (k,∞)	—	3.3											
RPP	DEF G_{10}	G_{19} G_{10}	G_{19} G_{10}										
FPFAP	—	3.16	G_{14} G_{10}	DEF G_{19}									
BCP	—	—	G_{14} G_{10}	—	4.4 G_{17}								
RNP	—	—	4.2 G_{17}	—	3.15 G_{14}	G_{14} G_{21}							
LR (k,t)	—	—	G_{14} G_{10}	—	3.15 G_{14}	G_{12} G_{21}	G_{14} G_{12}						
SLR (k,t)	—	—	DEF G_{17}	—	—	G_{14} G_{21}	G_{22} G_{12}	4.6 G_{14}					
LRR	—	G_{19} G_{20}	G_{14} G_{19}	4.3 G_{20}	G_{19} G_{20}	G_{19} G_{20}	G_{14} G_{20}	G_{19} G_{20}	G_{14} G_{20}				
LR(k)	—	—	3.0 G_{10}	—	—	G_{20} [illegible]	G_{14} G_{20}	DEF G_{20}	G_{14} G_{20}	4.3 G_{19}			
SLR (k)	—	—	—	—	—	G_{14} G_{21}	4.7 G_{20}	—	DEF G_{20}	—	3.0 G_{14}		
BC	—	—	G_{14} G_{10}	—	—	4.5 G_{20}	G_{14} G_{20}	—	G_{14} G_{20}	—	4.5 G_{21}	G_{14} G_{21}	

Figure 4.2 Summarized proofs of the relationships in Figure 4.1.

more classes between them) then the corresponding table entry is a dash.

We conclude this section by listing those example grammars, facts and theorems referenced in Figure 4.2. At the end of the section we present Figure 4.3 which specifies for each relevant grammar exactly which classes it is a member of.

Theorem 4.1 Not all unambiguous context-free grammars are parsable using regular patterns.

Proof If every unambiguous context-free grammar were RPP (regular pattern parsable), then we could recursively enumerate the set of unambiguous context-free grammars by using Theorem 1.6. However, since this set is known to be non-R.E., we arrive at a contradiction which proves our theorem.

q.e.d.

Theorem 4.2 Every RNP(k) grammar is SLR(k,∞) for the same value of k.

Proof Compare Algorithms 3.1 and 3.4. Quite clearly, the RNP(k) method will never make a reduction which would be bypassed by the SLR(k,∞) method. Thus, if $w \in SF(G)$ and the RNP(k) method performs a reduction on w, then the SLR(k,∞) method will also perform a reduction on w although it may in fact be a different reduction (i.e. performed further to the left). Since both algorithms are correct in the sense of never making wrong reductions, we reason that if G is RNP(k), then Algorithm 3.4 will make a correct reduction on every element of SF(G). Therefore, Algorithm 3.1 also will make a correct reduction on every sentential form of G which means that G is SLR(k,∞).

q.e.d.

Theorem 4.3 Every LR(k) grammar is an LRR grammar. Every LRR grammar is parsable using a finite set of regular patterns.

Proof See Cohen and Čulík [CC 71].

Theorem 4.4 Every BCP grammar is an FPFAP grammar.

Proof Suppose G is BCP(m,n) for some fixed values of m and n. Therefore, G is parsable using a collection of m left bounded, n right bounded patterns. These same patterns are obviously regular, n right bounded patterns and so G is FPFAP(n).

q.e.d.

Fact 4.5 Every BC grammar is an LR(k) grammar. Every BC grammar is a BCP grammar.

Proof The first statement was shown in Knuth [Kn 65]. The second fact was proven by Williams [Wi 69].

Fact 4.6 Every SLR(k,t) grammar is an LR(k,t) grammar for the same values of k and t.

Proof Obvious from definitions. Formal proof parallels Theorems 3.0 and 3.3.

Theorem 4.7 Every SLR(k) grammar is an RNP(k) grammar for the same value of k.

Proof As we commented in Section 3, the RNP(k) and SLR(k) methods function identically until the first postponement is made. However, if G is SLR(k) and we apply the RNP(k) algorithm to it, then no postponements will ever have to be made. Hence, the RNP(k) algorithm will also parse the string.

q.e.d.

G_{19}:

$$S \to Aa \mid Bb$$
$$A \to cAc \mid d$$
$$B \to cBc \mid d$$

G_{20}:

$$S \to A\bar{A} \mid B\bar{B}$$
$$A \to d$$
$$B \to d$$
$$\bar{A} \to a\bar{A} \mid a\bar{B}$$
$$\bar{B} \to a\bar{B}a \mid b$$

G_{21}:

$$S \to aA \mid bB$$
$$A \to cA \mid d$$
$$B \to cB \mid d$$

G_{22}:

$$
\begin{aligned}
S &\rightarrow aBG \mid aAD \mid bAE \\
A &\rightarrow c \\
B &\rightarrow c \\
D &\rightarrow dD \mid f \\
E &\rightarrow dE \mid f \\
G &\rightarrow dG \mid g
\end{aligned}
$$

	PP	LR (k,∞)	SLR (k,∞)	RPP	FPFAP	BCP	RNP	LR (k,t)	SLR (k,t)	LRR	LR(k)	SLR (k)	BC
G_{10}	Y	Y	Y	N	N	N	N	N	N	N	N	N	N
G_{12}	Y	Y	Y	Y	Y	Y	Y	N	N	Y	N	N	N
G_{14}	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y	N	Y
G_{17}	Y	Y	Y	Y	Y	N	N	N	N	Y	N	N	N
G_{18}	Y	Y	N	Y	Y	N	N	N	N	Y	N	N	N
G_{19}	Y	N	N	Y	N	N	N	N	N	Y	N	N	N
G_{20}	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N
G_{21}	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	N
G_{22}	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	N	N	N

Figure 4.3 Classification of various example grammars.

4.2 Relations in Language Space

Suppose we are given some subset S of the context-free grammars. Corresponding to S will be a subset S' of the context-free languages. S' will be defined as $\{L(G) \mid G \in S\}$. We will call S' the set of languages induced by S. In this section we will investigate the relationships between the classes of languages induced by the classes of grammars defined earlier in this dissertation.

One fact quickly becomes apparent when working in language space, namely proving the appropriate relations is much more difficult than was the case in grammar space. The reason for this is essentially that there are infinitely many grammars which generate any one language. Thus, in order to prove that a certain language is not induced by a given parsing technique, one must show that none of the corresponding grammars is parsable by the method in question. Similarly, in order to prove inclusion of one class by another, we must show that for every language in class A, there exists a grammar generating it which is a member of class B.

We will restrict ourselves in this section to the classes of languages which correspond to the parsing schemes which we find most interesting, namely the LR(k), LR(k,t), FPFAP(k), BCP and LR(k,∞) methods. Accordingly, we present their relationship in Figure 4.4.

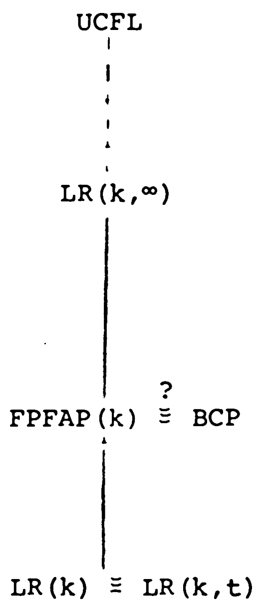


Figure 4.4 The inclusion lattice for classes of languages.

The class UCFL is the class of unambiguous context-free languages or as it is more commonly called, the class of languages which are not inherently ambiguous. The relation between this class and the class of languages for which there exists an LR(k,∞) grammar is shown as a dotted line because we have been unable to show that this inclusion is proper. Let us prove the facts summarized by this lattice.

The first few proofs are quite easy. It is well known [Kn 65] that the class of LR(k) languages is exactly the class of deterministic context-free languages, that is, the set of languages accepted by deterministic push down automata. However, we have already shown in Theorem 2.24 that the

LR(k,t) languages are all deterministic context-free languages. Hence, in language space LR(k,t) $\subseteq$ LR(k). On the other hand, since every LR(k) grammar is an LR(k,t) grammar (simply take t = 1) we also have the converse relation and can conclude that

<u>Theorem 4.8</u> The classes of deterministic context-free languages, LR(k) languages and LR(k,t) languages are identical.

Turning our attention to the FPFAP languages, we first note that every LR(k) grammar is an FPFAP(k) grammar and hence the LR(k) languages are all FPFAP languages. To see that this inclusion is proper, we consider

$$
G_{23}: \quad
\begin{aligned}
S &\to Aa \mid Bb \\
A &\to cA\bar{A} \mid d \\
B &\to cB\bar{B} \mid d \\
\bar{A} &\to c \\
\bar{B} &\to cc
\end{aligned}
$$

$$L(G_{23}) = \{c^n dc^n a \mid n \geq 0\} \cup \{c^n dc^{2n} b \mid n \geq 0\}$$

The reader can easily verify both that G_{23} is FPFAP(1) and that $L(G_{23})$ is not a deterministic context-free language. Hence, we have

<u>Theorem 4.9</u> The class of FPFAP(k) languages strictly includes the class of LR(k) languages.

To produce our next result, we must first present a series of lemmas.

Lemma 4.10 Every FPFAP(k) language is an LR(k,∞) language.

Proof Immediate from Theorem 3.16.

Lemma 4.11 The sets $\{a^n cb^n \mid n \geq 1\}$ and $\{a^n cb^{2n} \mid n \geq 1\}$ are not separable by any regular set.

Proof Suppose some finite state automaton M accepted the former set and rejected the latter. Consider the set of congruence classes induced on a^* by M. Pick any n and m such that $n \geq 2m$ and $a^n \equiv a^m$. Such n and m exist because there certainly must be an infinite congruence class. If we now apply Nerode's Theorem [HU 69] we conclude successively that

$$a^n \equiv a^m$$
$$a^n a^{n-2m} \equiv a^m a^{n-2m}$$
$$a^{2(n-m)} \equiv a^{n-m}$$

However, since $a^{2(n-m)} cb^{2(n-m)} \in T(M)$, we also have $a^{n-m} cb^{2(n-m)} \in T(M)$ in contradiction to our assumption.

q.e.d.

Lemma 4.12 [Hennie] If a one-tape off line Turing machine M performs all of its computations within the time Kn where K is a constant and n is the length of its input string, then the set accepted by M is regular.

<u>Proof</u> This lemma was proved in [He 65].

<u>Lemma 4.13</u> There exist LR(k,∞) languages for which no grammar is FPFAP(k). (Both instances of k are free.)

<u>Proof</u> Consider $L = \{a^n cb^n \mid n \geq 1\} \cup \{a^n cb^{2n} \mid n \geq 1\}$. L is generated by G_{10} which we have already shown to be an LR(k,∞) grammar (see Theorem 2.13 for the details). Hence, L is certainly an LR(k,∞) language.

Suppose that some grammar G for L were FPFAP(k) for some k. Let us analyze the properties of this grammar. Assume without loss of generality, that G is reduced.

1) If A is a nonterminal of G which can generate infinitely many strings, then it must be the case that any terminal string produced by A contains a c. If this were not the case, SF(G) would contain a string such as wAy where w and y are strings of terminals and y (say) contains a c. By letting A derive a string of length 2 × |y| we produce a string which certainly is not in L.

2) If $\phi \in$ SF(G) then there can be at most one nonterminal in ϕ which generates an infinite set because otherwise we could produce a string with multiple c's.

3) Suppose that A is a non-terminal and that $A \overset{+}{\rightarrow} a^m A b^n$ for some values of m and n. Then either n = m or n = 2m. Using 1 above and the fact that G is reduced,

$\exists q, r, s, t$ such that $S \overset{+}{\rightarrow} a^q A b^t$ and $A \overset{+}{\rightarrow} a^r c b^s$. Therefore, since $A \overset{+}{\rightarrow} a^{km} A b^{kn}$ for all k, we conclude that $a^q a^{km} a^r c b^s b^{kn} b^t \in L(G)\ \forall k$. Examine the ratio of number of b's to number of a's in such a form. This ratio is $\frac{(s+t) + kn}{(q+r) + km}$ and must for any value of k be either 1 or 2. The only way that this can happen is for n = m or for n = 2m.

4a) Let $A \overset{+}{\rightarrow} a^m A b^{2m}$ for some m. Then A never can occur in a sentential form ϕ which derives a string in $\{a^n c b^n \mid n \geq 1\}$.

Suppose otherwise. Thus $\phi = \psi_1 A \psi_2$ and $\phi \overset{*}{\rightarrow} a^n c b^n$ for some n. Then $\exists q, r, s, t$, such that $\psi_1 \overset{*}{\rightarrow} a^q$ and $A \overset{*}{\rightarrow} a^r c b^s$ and $\psi_2 \overset{*}{\rightarrow} b^t$. Furthermore, q + r = s + t. Since $\psi_1 A \psi_2$ is a sentential form, so are $a^q A b^t$

$$a^q a^{km} A b^{2km} b^t$$

$$a^q a^{km} a^r c b^s b^{2km} b^t$$

examing the ratio of b's to a's in this latter sentence, we have

$$f = \frac{s + t + 2km}{q + r + km}$$

In particular, letting k = q + r and recalling that s + t = q + r,

$$f = \frac{(q+r) + 2(q+r)m}{(q+r) + (q+r)m} = \frac{1 + 2m}{m}$$

but then 2 > f > 1 and the string could not have been in L(G).

4b) Similarly, if $A \overset{+}{\rightarrow} a^m A b^m$ for some m, then no sentential form containing an A can ever derive a string of the form $a^n c b^{2n}$.

5) We can now partition the vocabulary of G into four subsets.

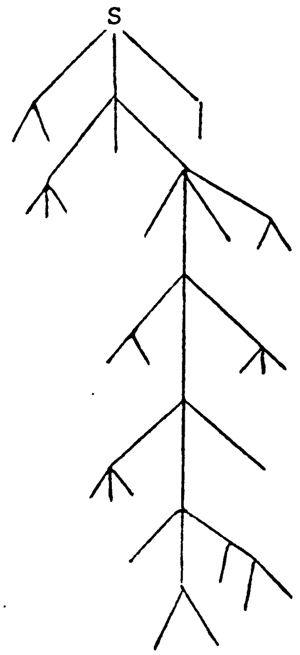
$V_1 = \{A \in V \mid L(G_A) \text{ is finite}\}$

$V_2 = \{A \in V \mid L(G_A) \text{ is infinite but } A \text{ is not recursive}\}$

$V_3 = \{A \in V \mid A \text{ is recursive and } A \overset{+}{\rightarrow} a^m A b^m \text{ for some } m\}$

$V_4 = \{A \in V \mid A \text{ is recursive and } A \overset{+}{\rightarrow} a^m A b^{2m} \text{ for some } m\}$

6) $\exists q$ such that any sentential form whose length is greater than q can only be derived using recursive non-terminals. Pictorially then, a tree for a long string looks like



where only elements of $V_2 \cup V_3$ occur along the "core" or else elements of $V_2 \cup V_4$ occur along the "core". All characters on the side branches are elements of V_1.

7) Suppose G is FPFAP(k) with machine M (a 2PDA) as its parser. We will generate a two stack acceptor M' which will simulate M on any input whose length is greater than q. However, the first time that M produces an element of V_3 or V_4, M' will halt. Furthermore, M' will indicate acceptance of its input if and only if the nonterminal which caused the termination is in V_3. On the other hand, if the input string has length $\leq$ q, then M' accepts it if and only if it is of the form $\{a^n cb^n \mid 1 \leq n \leq q/2\}$. Notice now that the set accepted by M' separates $\{a^n cb^n \mid n \geq 1\}$ from $\{a^n cb^{2n} \mid n \geq 1\}$. Furthermore, M' does this in linear time on a 2PDA.

8) M' can be simulated by a Turing machine M" which also runs in linear time. When M" makes a reduction such as A → BCD on its tape it will actually change the instance of BCD to Abb. Thus M" does not attempt to "shrink" its tape as the two stack device M' does. Since the side trees occurring on a derivation tree belonging to G are bounded in size (i.e. width) by some value p, we can never produce more than 2p blanks in a row before producing one of the nonterminals in V_3 or V_4. Thus, M" runs slower than M' by at most a constant factor, namely 2p.

9) T(M") is a regular set by Lemma 4.12. However, since T(M") = T(M'), we conclude that T(M") is a regular set which separates $\{a^n cb^n \mid n \geq 1\}$ from $\{a^n cb^{2n} \mid n \geq 1\}$.

This in turn contradicts Lemma 4.11 so we conclude that M could not have existed in the first place. Thus no grammar for L can be parsed by a finite state automaton or, more simply, L is not a member of the class of FPFAP(k) languages for any k.

q.e.d.

Combining Lemmas 4.10 and 4.13 we have

Theorem 4.14 The class of LR(k,∞) languages properly includes the FPFAP(k) languages.

Two relationships remain to be investigated. It is clear that every LR(k,∞) language is unambiguous and that every BCP language is FPFAP. On the other hand, we cannot prove whether or not these inclusions are proper. We offer our opinions however in

Conjecture 4.15 The classes of BCP and FPFAP languages are identical.

Conjecture 4.16 Not all unambiguous context free languages are LR(k,∞) languages.

Supporting (but not conclusive) evidence for the former theorem is a proof due to Williams [Wi 69] that the BCP languages include all deterministic languages. Two languages which we feel are strong candidates for establishing the latter conjecture are

$$\{ww^{reversed} \mid w \in (a \cup b)^*\}$$

and

$$\{a^n cb^m \mid n \leq m \leq 2n\}.$$

4.3 Decidability Properties

When the classes of grammars presented in this dissertation are viewed from the standpoint of decidability, three categories quickly distinguish themselves. We will define these categories as

Category 1: Pattern Parsable (PP) grammars
$LR(k,\infty)$ grammars
$SLR(k,\infty)$ grammars

Category 2: Regular Pattern Parsable (RPP) grammars
FPFAP(k) grammars
LR-Regular (LRR) grammars

Category 3: LR(k,t) grammars
LR(k) grammars
SLR(k,t) grammars
SLR(k) grammars
RNP(k) grammars
Bounded Context Parsable (BCP(m,n)) grammars
Bounded Right Context (BRC(m,n)) grammars
Bounded Context (BC(m,n)) grammars

If we take an arbitrary grammar G and pick any class from category 3 with fixed parameters (i.e. pick definite, fixed integer values for k, t, m or n) then it is decidable whether G is a member of the class in question. Furthermore, constructions can be given for building parsers for each of the category 3 classes.

On the other hand, even for fixed parameter values in the classes of category 1, it is undecidable whether an arbitrary grammar is in that class. Although algorithms can be given for parsing these grammars, no construction can be given for building parsers (in the sense of a 2PDA) for these grammars. The reason for this is that such a parser might very well have infinitely many states. An additional common property of this category is that it is undecidable whether a given set of reduction patterns is a parsing scheme for a grammar of this class.

Category 2 is somewhat of a hybrid. Parsers for grammars in these classes always have a finite number of states although it seems as if there is no a priori bound on how large this number can get (notice the similarity here to the Regular Separability Problem). We can certainly use Theorem 1.6 to test a set of patterns for correctness with respect to category 2 classes, but it is not presently known whether we can decide (for fixed parameters) whether a grammar is in one of these classes.

We can summarize the differences of these categories in a succinct way as follows. Let all parameters be fixed. Then category 1 classes are not recursively enumerable, category 2 classes are recursively enumerable (but probably not recursive) and category 3 classes are recursive sets.

Chapter 5 CONCLUSIONS

5.1 Future Work

There are essentially three areas in which the work of this thesis is incomplete.

First, we have yet to establish a decision procedure (or prove the non-existence thereof) for whether a grammar is FPFAP(k) for fixed k. As was mentioned in Chapter 3, this problem seems intimately related to the unsolved Regular Separability Problem. The decidability question also remains open for the LRR and RPP grammars.

Second, our diagram of the lattice for language space (Figure 4.4) is incomplete, probably because of our inability to choose the right counter-examples.

Finally, our parsing model of Chapter 1 has considerable room for further development. Although our theorems make provision for handling grammars with ε-rules, and are still correct in their presence, a certain anomaly does occur in grammar space. Consider the example,

G_{24}:

$$S \rightarrow Aa \mid Bb$$
$$A \rightarrow dAE \mid dE$$
$$B \rightarrow dBE \mid dE$$
$$E \rightarrow \varepsilon$$

$$L(G_{24}) = d^{+}a \cup d^{+}b$$

G_{24} is an LR(k) grammar and even an SLR(k) grammar, but it is not, according to our definitions, an LR(k,t) grammar!

The reason for this is that we have required our parsers to be able to handle any sentential form. In particular, our LR(k,t) parser cannot handle $\{d^nE^na \mid n \geq 1\} \subseteq SF(G_{24})$ or $\{d^nE^nb \mid n \geq 1\} \subseteq SF(G_{24})$. The LR(k) parser, on the other hand, needs only to concern itself with right sentential forms, which eliminates from its scope of action the troublesome cases described above. We now make an important observation. If we apply our LR(k,t) parser to a string in $L(G_{24})$, the string will be parsed correctly. This is because our parser, which always makes the leftmost possible reduction, will only generate right sentential forms while parsing. Hence our parsers will handle G_{24} even though G_{24} is not technically an LR(k,t) grammar.

This anomaly can never occur in the absence of ε-rules because then any k letter lookahead string always contains as much information as k characters of terminals. With ε-rules present however, a right context string can "shrink away" to nothing, giving us absolutely no additional information for making parsing decisions. In G_{24} for example, in the string "ddddEEEEa", the series of E's shield us from the piece of information which we really need to know, namely the presence of the final "a". Thus in the absence of ε-rules, if we are given a parser M which works on the set of right sentential forms, then we can easily produce a new parser M' which works on all sentential forms. Furthermore, M' will have the same number of states as M and can be easily constructed from the transition table for M.

The ε-rule anomaly mentioned above is merely one aspect of a problem which we have already discussed extensively in sections 1.4 and 1.5, namely the fact that we need to restrict our parsers to certain domains which are actually proper subsets of the set of all sentential forms. These proper subsets arise because every parser makes reductions in a specific order. Our present model, however, prevents a parser from deriving any useful information from the order in which it performs reductions. Said another way, all parsing information which arises from the right end of the input string must be transmitted back by making reductions and thereby changing the context of other phrases. Our model can not use sequencing in any way, shape or form to aid it in parsing. What we would like to suggest for further study is an investigation of the ways in which this sort of sequencing can be used profitably by a parser.

5.2 Summary

We feel that there are essentially four main contributions which we have made in this dissertation.

The parsing model of Chapter 1 provides a "global" view of the parsing process in that it reveals that the differences between methods can usually be characterized in terms of differences in either their parsing patterns or in the mode of application (leftmost, rightmost, etc.) of these patterns. Proving the correctness of arbitrary parsers by searching for their mistakes is a novel approach which merits future study in the directions indicated in the previous section. In particular this same approach can be used to justify optimizations and transformations performed on practical parsers for the purpose of increasing their efficiency or range of applicability.

The LR(k,∞) technique developed in Chapter 2 serves as the basis for all left to right non-canonical parsers. In particular, other parsers of this style can be viewed as operating with a set of states which is a homomorphic image of the state set of the LR(k,∞) PFA. Thus, the operation of the less general methods can be explained in terms of the structure of the item sets which are merged together by this homomorphism. The distinction between exact and inexact context methods is important because it reveals how the process of context approximation affects the trade off between parsing power and state set size.

The concept of FPFAP parsing, introduced in Chapter 3, is possibly the most general existing model of "practical" parsing within the traditional framework of automata. Much work is needed here to bound the applicability of this technique.

Finally, our investigations of language space in Chapter 4 have revealed that many facts can sometimes be ascertained about the structure of all grammars generating a particular language. These facts can then provide a basis for proving facts concerning the application of various parsing techniques to this language.

BIBLIOGRAPHY

[An 72] Anderson, T. "Syntactic Analysis of LR(k) Languages". Ph.D. Thesis, University of Newcastle upon Tyne, Newcastle upon Tyne, England. 1972.

[CC 71] Cohen, R.S. and K. Culik II. "LR-Regular Grammars - an Extension of LR(k) Grammars". IEEE Conference Record of 12th Symposium on Switching and Automata Theory, pp. 153-165. 1971.

[Co 70] Colmerauer, A. "Total Precedence Relations". JACM, 17:1, pp. 14-30. January 1970.

[De 69] DeRemer, F.L. "Practical Translators for LR(k) Languages". Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, Mass. October 1969.

[De 71] DeRemer, F.L. "Simple LR(k) Grammars". CACM, 14:7, pp. 453-460. July 1971.

[Ea 70] Earley, J. "An Efficient Context-free Parsing Algorithm". CACM, 13:2, pp. 94-102. February 1970.

[Fl 64] Floyd, R.W. "Bounded Context Syntactic Analysis". CACM, 7:2, pp. 62-67. February 1964.

[He 65] Hennie, F.C. "One-tape, Off-line Turing Machine Computations". Inf. and Control, 8:6, pp. 553-578. 1965.

[HU 69] Hopcroft, J.E. and J.D. Ullman. Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass. 1969.

[Kn 65] Knuth, D.E. "On the Translation of Languages from Left to Right". Inf. and Control, 8:6, pp. 607-639. 1965.

[Wi 69] Williams, J.H. "Bounded Context Parsable Grammars". Technical Report #58, Computer Science Department, University of Wisconsin, Madison, Wisconsin. 1969.

[WW 66] Wirth, N. and H. Weber. "Euler - A Generalization of Algol and its Formal Definition, Parts 1 and 2". CACM, 9:1, pp. 13-23. CACM, 9:2, pp. 89-99. January and February 1966.