NON-CANONICAL EXTENSIONS OF BOTTOM-UP PARSING TECHNIQUES[†]

Thomas G. Szymanski[††]

John H. Williams[†††]

TR 75-226

January 1975

Department of Computer Science
Cornell University
Ithaca, New York 14853

[†]This research was supported in part by the National Science Foundation under Grants GJ 42-512 and GJ 33171X.

[††]Thomas G. Szymanski, Princeton University, Princeton, NJ.

[†††]John H. Williams, Cornell University, Ithaca, NY.

NON-CANONICAL EXTENSIONS OF BOTTOM-UP PARSING TECHNIQUES[†]

Thomas G. Szymanski[††]

John H. Williams[†††]

TR 75-226

January 1975

Department of Computer Science
Cornell University
Ithaca, New York 14853

[†]This research was supported in part by the National Science Foundation under Grants GJ 42-512 and GJ 33171X.

[††]Thomas G. Szymanski, Princeton University, Princeton, NJ.

[†††]John H. Williams, Cornell University, Ithaca, NY.

NON-CANONICAL EXTENSIONS OF
BOTTOM-UP PARSING TECHNIQUES[†]

Thomas G. Szymanski[††]

John H. Williams[†††]

Abstract

A bottom-up parsing technique which can make non-leftmost possible reductions in sentential forms is said to be <u>non-canonical</u>. Nearly every existing parsing technique can be extended to a non-canonical method which operates on larger classes of grammars and languages than the original technique. Moreover, the resulting parsers run in time linearly proportional to the length of their input strings.

Several such extensions are defined and analyzed from the points of view of both power and decidability. The results are presented in terms of a general bottom-up parsing model which yields a common decision procedure for testing membership in many of the existing and extended classes.

[†] This research was supported in part by the National Science Foundation under Grants GJ 42-512 and GJ 33171X.

[††] Thomas G. Szymanski, Princeton University, Princeton, NJ.

[†††] John H. Williams, Cornell University, Ithaca, NY.

Section 1 Introduction

Since 1965 when Knuth suggested that the power of formal parsing techniques might be increased by allowing them to reduce not only handles but other phrases of sentential forms as well, there have been few investigations exploring this possibility. In this paper we consider this technique which we call non-canonical parsing. We present a descriptive framework in which to consider bottom-up parsing techniques in general and non-canonical techniques in particular, and we use this framework to describe some existing parsing techniques and to suggest their non-canonical extensions.

Very simply stated, the class of grammars that can be parsed by bottom-up reduction methods operating under a left to right scan has been progressively approximated by the BRC notion of Floyd [FL 64], then the LR(k) notion of Knuth [Kn 65] and finally by the LR-regular notion of Cohen and Culik [CC 73] (we overlook the right to left transduction of the latter method in making this oversimplified summary). Essentially each of these methods generalizes its predecessor in that it enlarges the class of grammars (and sometimes the class of languages) to which it is applicable. This generality has been obtained by discovering ways to increase the amount of context used in making decisions without increasing the complexity of the decisions so much that the parser can no longer parse in linear time. Note that each of these methods has required the parser to produce a canonical parse.

Our approach has been to remove the restriction that a parser must always reduce the handle of a sentential form and to study the properties of the resulting non-canonical parsers. In Sections 2 and 3 we present basic definitions and develop a general model of bottom-up parsing techniques with which we can describe and characterize some of the existing methods. Then, in Section 4, we consider the non-canonical extensions of these methods and consider to what degree these extensions possess the useful properties of i) membership in the class being decidable, and ii) membership in the class implying the existence of a linear time parser for the grammar. Finally, we demonstrate that the non-canonical versions of the various methods considered are applicable to more grammars than their canonical counterparts, and we compare the various classes of languages thus obtained.

In order to motivate the more formal treatment of Section 3 we will attempt to give the flavor of non-canonical techniques by considering the non-LR-regular language.

$L_1 = \{a^n b^n c^m d^{m+\ell} \mid n,m,\ell \geq 1\} \cup \{a^n b^{2n} c^m d^m \mid n,m \geq 1\}$

and constructing a BCP grammar for it. A BCP grammar is essentially a grammar which can be parsed by a non-canonical, bounded context parser [Wi 72]. We emphasize that this is intended to be merely an intuitive discussion; the formal definition appears in Section 4.

Intuitively, L_1 is not LRR since any grammar for L_1 will require the handle of some sufficiently long sentence to be at the a-b interface, and the context required to distinguish between the two alternative parses, namely whether there are more d's than c's in the remaining portion of the string, cannot be determined by a partition into regular sets. L_1 is a BCP language however, since the grammar G_1 given in Figure 1 is a BCP(1,1) grammar for it.

Intuitively, G_1 allows a non-canonical parser to postpone any decisions about the a's and b's until the c's and d's have been reduced. It is then a simple matter to tell whether there were originally more d's than c's. This information is then transmitted back to the a-b interface via the B and B' one-productions. In Figure 1, we associate with each production, $A_i \rightarrow x_i$, of G_1 a set of "parsing contexts", i.e. contexts in which it is always correct to reduce x_i to A_i.

	production	parsing contexts
G_1:	1) $S \rightarrow XY$	$(\vdash, \dashv)$
	2) $S \rightarrow X'Y'$	$(\vdash, \dashv)$
	3) $X \rightarrow aXB$	(Λ, Λ)
	4) $X \rightarrow aB$	(Λ, Λ)
	5) $X' \rightarrow aX'B'B'$	(Λ, Λ)
	6) $X' \rightarrow aB'B'$	(Λ, Λ)
	7) $Y \rightarrow Yd$	(Λ, Λ)
	8) $Y \rightarrow Zd$	(b, Λ)
	9) $Y' \rightarrow Z$	$(\Lambda, \dashv)$
	10) $Z \rightarrow cZd$	(Λ, Λ)
	11) $Z \rightarrow cd$	(Λ, Λ)
	12) $B \rightarrow b$	(Λ, B) , (Λ, Y)
	13) $B' \rightarrow b$	(Λ, B') , (Λ, Y')

Figure 1. A BCP grammar for L_1.

step	string	applicable reduction
1)	$\vdash aabbc\underline{cd}dd \dashv$	11
2)	$\vdash aabb\underline{cZd}d \dashv$	10
3)	$\vdash aabb\underline{Zd} \dashv$	8
4)	$\vdash aab\underline{b}Y \dashv$	12
5)	$\vdash aa\underline{b}BY \dashv$	12
6)	$\vdash a\underline{aB}BY \dashv$	4
7)	$\vdash \underline{aXB}Y \dashv$	3
8)	$\vdash \underline{XY} \dashv$	1
9)	$\vdash S \dashv$	

Figure 2. A sample parse of a sentence in $L(G_1)$.

For example, any occurrence of the string Yd can always be reduced to a Y (production 7), but the string b can be reduced to a B (production 12) only when it occurs next to a B or a Y.

Actually, the parsing contexts listed in Figure 1 are only a subset of those produced by the general BCP analyzer. For example, the string Zd could be reduced to a Y (production 8) not only when it follows a b but also whenever it follows a B. The subset we have presented is sufficient to allow the correct parsing of any sentence of G_1. We illustrate this by considering the parse of the sentence

$$a^2b^2c^2d^3$$

in Figure 2. At each step we have underlined the leftmost string which occurs in a parsing context, and we indicate the appropriate reduction.

Notice that even though only one character of left and right context is ever used, the important information can be passed back to the crucial a-b interface by rippling back over the b's via the one-production, $B \rightarrow b$. This is the technique that gives the non-canonical methods their additional power. It will be seen in the next section that this increase in power has not been achieved at the expense of the two important properties mentioned earlier, namely decidability of membership in the class and linear time parsing.

We want to re-emphasize that the purpose of this example was to give an intuitive feeling for the formal treatment to follow. In particular it will be seen that BCP parsers can be applied to arbitrary sentential forms as well as sentences. This simplification was made here in order to expose the essential features of non-canonical parsing.

Section 2 Basic Definitions

In this section we exhibit our notation. First, we need the usual grammatical concepts.

Def. 2.1: A context free grammar G is a quadruple (V,Σ,P,S) where V and Σ are finite sets called, respectively, the vocabulary and terminals of G (the set $N = V - \Sigma$ is called the nonterminals of G), P is a finite subset of $N \times V^*$ called the productions of G and $S \in N$ is called the start symbol of G.

We adhere to the usual convention of using $A,B,C,\ldots$ to denote elements of N, $X,Y,Z,\ldots$ to denote elements of V and $\alpha,\beta,\gamma,\ldots$ to represent elements of V^*.

Def. 2.2: We define the relation $\Rightarrow \subseteq V^* \times V^*$ by saying that $\alpha \Rightarrow \beta$ iff $\alpha = \alpha_1 A \alpha_2$, $\beta = \alpha_1 \beta_1 \alpha_2$ and $A \rightarrow \beta_1$ is in P for some $A \in N$ and $\alpha_1, \alpha_2, \beta_1 \in V^*$. The set of sentential forms of G is the set $SF(G) = \{\alpha \in V^* \mid S \overset{*}{\Rightarrow} \alpha\}$. The language of G is the set of terminal sentential forms, that is, $L(G) = SF(G) \cap \Sigma^*$.

In dealing with parsers, we need the concept of a phrase. We assume that the reader is familiar with the concept of a derivation tree for a grammar. The next definition is basic to this paper.

Def. 2.3: Let T be a derivation tree for some sentential form $\beta\alpha\gamma$ of the context free grammar G. We say that the ordered pair $(A \rightarrow \alpha, i)$ is a phrase of T if $A \rightarrow \alpha \in P$ and $i = |\beta\alpha|$ and there exists a derivation corresponding to T of the form $S \overset{*}{\Rightarrow} \beta A \gamma \Longrightarrow \beta\alpha\gamma$.

Notice that phrases are defined only in terms of derivation trees. If G is an unambiguous grammar, then and only then will it make sense to talk about the phrases of a sentential form.

We next linearize the concept of a derivation tree in two ways. A description language for a grammar will be a set of bracketed strings denoting a (unique) derivation tree and the phrase language for a grammar will describe the location of all the phrases of a sentential form relative to a given derivation tree.

Def. 2.4: Let $G = (V,\Sigma,P,S)$ be a CFG. Let $\mathscr{B} = \{]_i \mid 1 \leq i \leq |P|\}$ be a set of new characters. We call $\mathscr{B}$ the set of brackets for G. Let $\bar{N} = \{\bar{A} \mid A \in N\}$ also be a new set of characters. The description language for G is the language $DL(G)$ generated by the grammar

$G' = (\bar{N} \cup V \cup \mathscr{B}, V \cup \mathscr{B}, P', \bar{S})$ where

$$P' = \{\bar{A} \to Y_1 \ldots Y_{n_i}]_i \mid A \to X_1 \ldots X_{n_i} \text{ is the i-th production of } P \text{ and } Y_j = X_j \text{ or } \bar{X}_j \text{ for } 1 \leq j \leq n_i\}$$

The phrase language $PL(G)$ for G is defined by the context free grammar $G'' = (\bar{N} \cup V \cup \mathscr{B}, V \cup \mathscr{B}, P'', \bar{S})$ where

$$P'' = \{\bar{A} \to Y_1 \ldots Y_{n_i} \mid A \to X_1 \ldots X_{n_i} \text{ is the i-th production of } P,\ Y_j = X_j \text{ or } \bar{X}_j \text{ for } 1 \leq j \leq n_i \text{ and at least one } Y_j = \bar{X}_j\}$$
$$\cup$$
$$\{\bar{A} \to X_1 \ldots X_{n_i}]_i \mid A \to X_1 \ldots X_{n_i} \text{ is the i-th production of } P\}.$$

For example, consider the grammar whose productions are $S \to SS \mid A$ and $A \to a$. $DL(G)$ is defined by the set of productions

$\bar{S} \to \bar{S}\bar{S}]_1 | \bar{S}S]_1 | S\bar{S}]_1 | SS]_1$

$\bar{S} \to A]_2 | \bar{A}]_2$

$\bar{A} \to a]_3$

PL(G) is defined by the productions

$\bar{S} \to \bar{S}\bar{S} | \bar{S}S | S\bar{S} | SS]_1$

$\bar{S} \to \bar{A} | A]_2$

$\bar{A} \to a]_3$

Consider now the following derivation tree according to G:

```
        S
       / \
      S   S
     / \  |
    S   S A
        | |
        A a
        |
        a
```

This tree is represented by the strings

$\phi_1 = Sa]_3]_2]_1a]_3]_2]_1 \in DL(G)$

and

$\phi_2 = Sa]_3a]_3 \in PL(G)$

Notice that the brackets in a string of DL(G) capture the complete structure of some derivation tree whereas the brackets of a string in PL(G) describe only the locations of the phrases for that tree. Notice also that if G is ambiguous, there may exist more than one string in DL(G) or PL(G) corresponding to a given sentential form of G. The mapping between SF(G) and DL(G) or PL(G) is provided by the string homomorphism defined below.

Def. 2.5: Let $G = (V,\Sigma,P,S)$ be a CFG and $\mathscr{B}$ be the bracket set for G. Let $\#$ be a special character not in $V \cup \mathscr{B}$. Define $m: (V \cup \mathscr{B} \cup \{\#\})^* \to (V \cup \{\#\})^*$ by

$$m(\alpha) = \begin{cases} Y \cdot m(\gamma) & \text{if } \alpha = Y\gamma,\ Y \in V \cup \{\#\} \\ m(\gamma) & \text{if } \alpha = Y\gamma, Y \in \mathscr{B} \\ \Lambda & \text{if } \alpha = \Lambda \end{cases}$$

define $h: (V \cup \mathscr{B} \cup \{\#\})^* \to (V \cup \mathscr{B})^*$ by

$$h(\alpha) = \begin{cases} Y \cdot h(\gamma) & \text{if } \alpha = Y\gamma, Y \in V \cup \mathscr{B} \\ h(\gamma) & \text{if } \alpha = Y\gamma, Y = \# \\ \Lambda & \text{if } \alpha = \Lambda \end{cases}$$

The maps m and h are respectively referred to as the bracket-erasing and marker erasing homomorphisms. The reader may verify that the restrictions of m to $DL(G)$ or $PL(G)$ are one-to-one if and only if G is unambiguous. The reader may further verify that $m(DL(G) \cup \{S\}) = m(PL(G) \cup \{S\}) = SF(G)$.

Section 3 A Model of Bottom-Up Parsing

In the sequel it will be necessary to discuss the operation of parsers which make non-canonical reductions. For this purpose we define a general model of bottom-up parsing which emphasizes the context used by a parser when making reductions. No restriction will be placed on where in a subject string the parser chooses to make a reduction.

Def. 3.1: Let $G = (V,\Sigma,P,S)$ be a CFG. A reduction pattern for G is a pair $(R, A \to \alpha)$ where $A \to \alpha \in P$ and $R \subseteq V^*\alpha\#V^*$. The reduction pattern is said to apply to the string

$\phi = \beta_1\beta_2\alpha\beta_3\beta_4$ if $\beta_2\alpha\#\beta_3 \in R$. The string $\phi' = \beta_1\beta_2 A\beta_3\beta_4$ is said to be a reduction of ϕ implied by $(R, A \rightarrow \alpha)$.

The set R thus specifies those contexts in which a given reduction may be made.

Note that no restrictions have been made on the set R as far as finiteness or even recursiveness are concerned. Of course the interesting cases will involve "nice" sets in the sense of being regular, bounded to the right of the #, etc.

In order for a reduction pattern to be useful, we must place some additional qualifications upon it. In particular, let us require that an implied reduction must be correct in the sense that the substring "pruned" by the pattern from a sentential form ϕ is in fact a phrase of ϕ relative to all derivation trees for ϕ.

Def. 3.2: Let $(R, A \rightarrow \alpha)$ be a reduction pattern for a CFG $G = (V, \Sigma, P, S)$ in which $A \rightarrow \alpha$ is the i-th production. We say that $(R, A \rightarrow \alpha)$ is a parsing pattern for G if

1) $\psi \in PL(G)$

2) $m(\psi) = \beta_1\beta_2\alpha\beta_3\beta_4$

3) $\beta_2\alpha\#\beta_3 \in R$

imply $\psi \in m^{-1}(\beta_1\beta_2)\alpha]_i m^{-1}(\beta_3\beta_4)$.

Consider for example the grammar G whose productions are $S \rightarrow SS|A$ and $A \rightarrow a$. The reduction pattern $(\{a\#\}, A \rightarrow a)$ is a parsing pattern because any "a" can only be derived from an "A". On the other hand, the pattern $(\{SS\#\}, S \rightarrow SS)$ is not a parsing pattern because the pair of S's in the sentential form SSA is not always an immediate S derivative. This is captured by the definition in that both $\psi_1 = SS]_1 A]_2$ and $\psi_2 = SSA]_2$ are elements of PL(G), yet SS is not a phrase of SSA relative to <u>all</u> derivation trees (eg. $\psi_2 \not\in m^{-1}(A)SS]_1\, m^{-1}(A)$).

At this point we are ready to characterize parsing patterns in such a way as to yield an effective procedure for testing whether a large class of reduction patterns are in fact also parsing patterns.

<u>Lemma 3.3</u>: Let G be a CFG and $\mathcal{B}$ be the set of brackets for G. Let $A \to \alpha$ be the i-th production of G and $(R, A \to \alpha)$ be a reduction pattern for G. Define the set

$$M = h^{-1}(PL(G)) \cap m^{-1}(V^*RV^*) \cap m^{-1}(V^*)V\#(\mathcal{B}-]_i)^* V\, m^{-1}(V^*) .$$

Then $(R, A \to \alpha)$ is a parsing pattern for G if and only if $M = \emptyset$.

<u>Proof</u>: Loosely speaking, M is the set of "mistakes" committed by the reduction pattern, and a parsing pattern will be a reduction pattern whose set of mistakes is empty. More formally:

only if: Suppose $M \neq \emptyset$. We must show that $(R, A \to \alpha)$ is not a parsing pattern.

Let $\psi' \in M$.

ψ' can be uniquely written as $\phi_1 X_1 \# \gamma X_2 \phi_2$ where $\phi_1, \phi_2 \in m^{-1}(V^*)$, $X_1, X_2 \in V$, $\gamma \in (\mathcal{B}-]_i)^*$.

Let $\psi = h(\psi')$. Then $\psi \in PL(G)$.

By definition of M, $\exists\ \beta_1, \beta_2, \beta_3, \beta_4 \in V^*$ such that $m(\phi_1 X_1) = \beta_1\beta_2\alpha$, $m(X_2\phi_2) = \beta_3\beta_4$, and $\beta_2\alpha\#\beta_3 \in R$.

(Note that since $\psi \in PL(G)$, $m(\psi) = \beta_1\beta_2\alpha\beta_3\beta_4 \in SF(G)$).

Notice that X_1 is the last character of $\beta_1\beta_2\alpha$ and that X_2 is the first character of $\beta_3\beta_4$.

If $\psi = \phi_1 X_1 \gamma X_2 \phi_2 \in m^{-1}(\beta_1\beta_2)\alpha]_i m^{-1}(\beta_3\beta_4)$ then γ would have to contain $]_i$. But by hypothesis, $\gamma \in (\mathscr{B} -]_i)^*$.
Hence $\psi \notin m^{-1}(\beta_1\beta_2)\alpha]_i m^{-1}(\beta_3\beta_4)$ and we conclude that $(R, A \to \alpha)$ is not a parsing pattern.

if: Suppose $(R, A \to \gamma)$ is not a parsing pattern. We must show that $M \neq \emptyset$.

Since $(R, A \to \alpha)$ is not a parsing pattern, $\exists\ \psi \in PL(G)$ such that $m(\psi) = \beta_1\beta_2\alpha\beta_3\beta_4$ with $\beta_2\alpha\#\beta_3 \in R$ but $\psi \notin m^{-1}(\beta_1\beta_2)\alpha]_i m^{-1}(\beta_3\beta_4)$.
By picking X_1 = last character of $\beta_1\beta_2\alpha$ and X_2 = first character of $\beta_3\beta_4$ we can uniquely write $\psi = \phi_1 X_1 \gamma X_2 \phi_2$ with $\phi_1, \phi_2 \in (V \cup \mathscr{B})^*$, $X_1, X_2 \in V$, $\gamma \in \mathscr{B}^*$, $m(\phi_1 X_1) = \beta_1\beta_2\alpha$, and $m(X_2\phi_2) = \beta_3\beta_4$.
Moreover, since $\psi \notin m^{-1}(\beta_1\beta_2)\alpha]_i m^{-1}(\beta_3\beta_4)$ we must have $\gamma \in (\mathscr{B} -]_i)^*$. (Note that in PL(G), any occurrence of $]_i$ must be immediately preceded by α with no intervening brackets.)

Now consider the string $\psi' = \phi_1 X_1 \# \gamma X_2 \phi_2$.
Since $h(\psi') = \psi$ and $\psi \in PL(G)$, we have $\psi' \in h^{-1}(PL(G))$.
Since $m(\psi') = \beta_1\beta_2\alpha\#\beta_3\beta_4$ and $\beta_2\alpha\#\beta_3 \in R$ by hypothesis, we have $\psi' \in m^{-1}(V^*RV^*)$.
Finally, since $\phi_1, \phi_2 \in m^{-1}(V^*)$, $X_1, X_2 \in V$ and $\gamma \in (\mathscr{B} -]_i)^*$, we have $\psi' \in m^{-1}(V^*)V\#(\mathscr{B} -]_i)^* V m^{-1}(V^*)$.
Thus $\psi' \in M$ and $M \neq \emptyset$ as was to be shown. ∎

Since the set M defined in Lemma 3.3 is a context-free set whenever the set R is regular, we have

<u>Corollary 3.4:</u> It is decidable whether a regular reduction pattern is a parsing pattern.

Notice that all commonly used parsing methods use reduction patterns which are in fact regular sets. This observation is easily verified for any of the precedence or bounded context algorithms. The LR(k) method of Knuth [Kn 65] essentially uses patterns which are regular sets but which have only k or fewer characters following the #. The LR-regular technique of Cohen and Culik [CC 73] uses a pattern set which is of unbounded length to either side of the # but which is always regular. All of the above methods however are strictly canonical in operation. Our model thus not only includes them but also allows for their extension to noncanonical operation.

A single parsing pattern allows us to make reductions for but one production of the grammar at hand. In order to parse arbitrary sentences we need a pattern for each production along with some guarantee that the parsing process will not "block". In other words, the set of parsing patterns must cover the set of sentential forms which are generated by the parsing process itself while processing strings of L(G). Since this set of sentential forms is not necessarily a "cleanly" structured set we will substitute a simpler requirement, namely the ability to cover the <u>entire</u> set of sentential forms. We thus are led to

<u>Def. 3.5</u>: 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 S , there

exists some pattern in the collection which applies to it. This concept left unrestricted is sufficient to allow us to parse any unambiguous context-free grammar. Moreover,

Theorem 3.6: Let $G = (V,\Sigma,P,S)$ be a CFG in which $S \not\overset{+}{\Rightarrow} S$. Then there exists a parsing scheme for G if and only if G is unambiguous.

Proof: only if: suppose we have a parsing scheme $\mathcal{S}$ for G but that G is ambiguous. Therefore there exists some sentential form ϕ with two distinct derivation trees T_1 and T_2. We may further assume that T_1 and T_2 have no common phrases (if they do, we can "prune" the common phrases until the desired condition occurs.). The sentential form ϕ cannot be S (because we have hypothesized that $S \not\overset{+}{\Rightarrow} S$) and therefore some reduction pattern in $\mathcal{S}$ applies to ϕ. The "phrase" located by this pattern cannot be a phrase of both T_1 and T_2, and thus the reduction pattern fails to be a parsing pattern. Thus $\mathcal{S}$ could not have been a parsing scheme after all.

if: If G is unambiguous we can meaningfully talk about the phrases of a sentential form instead of just the phrases of a derivation tree. So now we take as a reduction pattern for the i-th production, $A \to \alpha$, the set $R_i = \{\beta\alpha\#\gamma \mid S \overset{+}{\Rightarrow} \beta A\gamma\}$. This (context-free) pattern is a parsing pattern for G because whenever $\phi = \beta\alpha\gamma \in SF(G)$ and $\beta\alpha\# \in R_i$, then there is but one derivation tree for ϕ and $(i,|\beta\alpha|)$ must be a phrase of that tree. Furthermore the collection of parsing patterns $\{R_i | 1 \leq i \leq |P|$ clearly covers every sentential form except for S. ∎

An immediate and somewhat unfortunate corollary to the above theorem is

Corollary 3.7: It is undecidable whether there exists a parsing scheme for an arbitrary context free grammar.

Thus is appears as if our original conception of a parsing scheme is too powerful to work with. The obvious restriction (with an eye toward meaningful implementation) is to require that each reduction pattern be a regular set. This restriction yields a class of parsing schemes which can be tested for correctness as described in

Theorem 3.8: [Decidability Theorem for correctness of bottom-up parsers] Let G be a CFG and $\mathcal{S}$ be a finite collection of reduction patterns for G . Then it is decidable whether $\mathcal{S}$ is a parsing scheme for G .

Proof: We first check that each member of the finite collection $\mathcal{S}$ is a parsing pattern. Corollary 3.4 tells us that this is decidable. We must next check that some pattern applies to each sentential form. Let $\mathcal{R}$ be the regular set $\bigcup_{R \in \mathcal{S}} V^*RV^*$. Then every sentential form is reducible if $SF(G) - \{S\} \subseteq \mathcal{R}$. Since the set on the left of the $\subseteq$ is a context-free set and $\mathcal{R}$ is regular, this question is also decidable. ∎

In the next few paragraphs let us consider the implementation of a parser based on regular reduction patterns. If we require that only canonical reductions be made, then we essentially have a model of the LR-regular parsing process [CC 73] and can implement the parser on a one stack DPDA which is allowed to perform an initial right-to-left finite state transduction of its input. If non-canonical reductions are allowed, then one stack is no longer sufficient, and we are forced to turn to a two stack implementation. We thus have

Theorem 3.9: Let G be a CFG and $\mathscr{P}$ be a parsing scheme for G, all of whose patterns are regular. Then $L(G)$ can be parsed by a deterministic two stack pushdown transducer. Furthermore, there exists a constant c depending only on G such that any string of length n can be parsed in time cn^2.

Proof: The two stacks are used to hold those portions of the current sentential form which are respectively to the left or right of the current "point of interest". Each reduction can be made by a left-to-right sweep followed by a right-to-left sweep over the sentential form recording states of the regular patterns as we go. Thus a two stack DPDA implementation works.

To establish the time bound, recall that if a grammar is unambiguous then there exists a constant c_1 such that any derivation of a string of $L(G)$ of length n takes at most $c_1 n$ steps. Furthermore no intermediate sentential form of this derivation is of length longer than $c_1 n$. Hence the parsing process described above takes at most time $(c_1 n)^2$. ∎

If we limit ourselves to reduction patterns which are bounded on one side of the #, we can establish a linear time bound for parsing. Such patterns can be implemented in the simple automaton of

<u>Def. 3.10</u>: Let $G = (V,\Sigma,P,S)$ be a CFG. A <u>reducing automaton</u> for G is a quadruple (Q,k,δ,q_0) where

Q is a (not necessarily finite) set of <u>states</u>,

k, the <u>lookahead factor</u>, is an integer ≥ 1,

δ, the <u>move function</u>, is a map $\delta: Q \times V^k \rightarrow Q \cup \{i | 1 \leq i \leq |P|\}$,

$q_0 \in Q$ is the start state.

The extension of δ to $Q \times V^kV^*$ is defined inductively as follows:

$$\hat{\delta}(p,\alpha\beta X) = \begin{cases} \delta(\hat{\delta}(p,\alpha\beta),\beta X) & \text{if } |\beta X| = k \text{ and } \hat{\delta}(p,\alpha\beta) \in Q \\ \text{undefined otherwise} \end{cases}$$

A reducing automaton functions in a manner similar to that of a finite state automaton except 1) each move consults the next k characters of input and 2) the machine stops as soon as the move function yields the index of a production instead of a new state.

A natural collection of reduction patterns can be associated with any reducing automaton. If this collection constitutes a parsing scheme, then we call the automaton a <u>parsing automaton</u>. More formally,

<u>Def. 3.11</u>: Let $A = (Q,k,\delta,q_0)$ be a reducing automaton for $G = (V,\Sigma,P,S)$. Let $R_i = \{\beta \# \gamma | \beta \in V^*, \gamma \in V^k \text{ and } \hat{\delta}(q_0,\beta\gamma) = i\}$. Let $\mathcal{S} = \{R_i | 1 \leq i \leq |P|\}$. A is said to be a <u>parsing automaton</u> for G iff $\mathcal{S}$ is a parsing scheme for G.

Restricting a parser to a fixed amount of lookahead beyond the right end of a phrase yields a linear time parser.

<u>Theorem 3.12</u>: Let A be a parsing automaton for G. Then there exists a deterministic pushdown transducer which parses sentences of G in time $O(n)$ where n is the length of the input string.

Proof: Suppose that $\phi = \beta_1\beta_2\alpha\gamma_1\gamma_2$ is a string with $|\beta_2| = |\gamma_1| = k$. Suppose further that $\hat{\delta}(q_0,\beta_1\beta_2\alpha\gamma_1) = i$ and the i-th production is $A \to \alpha$. This means that $\hat{\delta}(q_0,\beta_1\beta_2)$ is well defined and is a member of Q. If we now reduce ϕ to $\phi' = \beta_1\beta_2 A\gamma_1\gamma_2$ and re-apply A to find another phrase, the found phrase must be to the right of β_1. Thus in implementing A as a parser, after each reduction we need only "back up" k characters in the sentential form before continuing a left to right scan.

The total time spent during parsing can thus be divided into three parts: t_1 = the time spent backing up, t_2 = the time spent moving forward over regions of the string already scanned and t_3 = the time spent advancing over previously unscanned characters. The time t_3 is clearly equal to n. The time t_1 is bounded above by $c_1n(k+\ell)$ where c_1n bounds the total number of reductions made (see proof of Theorem 3.9) and ℓ is the length of the longest right side of a production. Similarly t_2 is at most $c_1n(k+1)$. In any case, since c_1 and ℓ depend only on G, we conclude that $t_1 + t_2 + t_3 = O(n)$. ∎

Thus the parsing automaton is just the model which we need to develop the desired linear time, non-canonical extensions of existing parsing algorithms.

Section 4 Non-canonical extensions

In this section we will show how several existing parsing methods can be extended to operate non-canonically while remaining within the framework of the theory developed in section 3. We will then analyze the generative power of these extensions with respect to both classes of parsable grammars and classes of recognizable languages.

Let us start with Floyd's Bounded Context grammars [Fl 64]. A grammar is m,n Bounded Context (BC(m,n)) if every phrase of every sentential form is uniquely distinguished by the m characters to its left and the n characters to its right. Extending this idea, a grammar is said to be m,n Bounded Context Parsable (BCP(m,n)) if at least one phrase of every sentential form is uniquely distinguished by the m characters to its left and n characters to its right [Wi 72]. This extension is actually non-canonical because a BC parser was intended to operate in a strictly left-to-right fashion.

An equivalent definition may be phrased in the terminology of section 3. Thus a grammar is BCP(m,n) iff there exists a parsing scheme for it such that every reduction pattern takes the form of an m character string, followed by the right side of the production in question, followed by a # and then an n character string.

The results of section 3 immediately yield

Theorem 4.1: It is decidable whether a grammar is BCP(m,n) for fixed m and n.

Proof: One simply generates all possible sets of reduction patterns which fit the BCP condition and tests their correctness using

Theorem 3.8. Such a procedure is guaranteed to halt by the fact that there are but a finite number of such patterns. ∎

A simple example of a BCP(1,1) grammar is G_1. It has also been shown that the BCP grammars are sufficiently powerful to generate a proper superset of the deterministic languages [Wi 72].

We mention in passing that the work of Colmerauer [Co 70] was intended to be a non-canonical extension of simple precedence parsing. It has been shown however [AU 72] that the resulting class of grammars is too powerful in the sense of including some ambiguous grammars. This will never be the case with our parsers as was shown in Theorem 3.6.

Let us consider next the non-canonical extension of LR(k) parsing. Knuth [Kn 65] suggested a partially non-canonical extension by defining an <u>LR(k,t) grammar</u> as an unambiguous CFG in which every sentential form has the property that one of its t leftmost phrases is uniquely distinguished by its left context and first k characters of right context. An example of an LR(1,2) grammar is provided by G_2 :

$$S \rightarrow A\bar{A} \mid B\bar{B}$$
$$A \rightarrow a$$
$$B \rightarrow a$$
$$\bar{A} \rightarrow b\bar{A} \mid b\bar{B}$$
$$\bar{B} \rightarrow b\bar{B}c \mid bc$$

The reader should note that G_2 is not an LR(k) grammar or even an LR-regular grammar.

Parsers for LR(k,t) grammars can be constructed using a technique similar to that used for building LR(k) parsers (see [AU 72] for details of this latter construction). The only major difference is that the lookahead string associated with an individual LR(k) item is allowed to contain non-terminals as well as terminals. Whenever an inadequate item set (i.e. one in which the correct parsing action is undefined due to the presence of conflicting items) is reached, the parser postpones any implied reduction(s) and shifts to a new set of items. This new item set includes in its closure any items produced by expanding the leading non-terminal in the lookahead string of any postponed items. Complete details of this construction appear in [Sz 73]. The construction also serves as a test for LR(k,t)-ness.

Since the process of postponing an individual item can be repeated at most t times, no lookahead string need have a length which exceeds kt. Hence the above construction is finite and we have

Theorem 4.2: It is decidable whether an arbitrary CFG is LR(k,t) for fixed values of k and t. Furthermore, an LR(k,t) grammar can be parsed in linear time.

Proof: The construction sketched above gives rise to a set of regular reduction patterns whose adequacy can be tested by Theorem 3.8. The linear time result then follows immediately from Theorem 3.12. However, since an LR(k,t) parser has to back up at most t times in succession and since each back up is of distance k, we can claim a stronger result, namely that LR(k,t) grammars can be parsed by a DPDA. ∎

Note that parsing with a DPDA implies that all LR(k,t) languages are in fact deterministic languages; that is, every LR(k,t) grammar is equivalent to some LR(k) grammar. This suggests that we should consider instead the fully non-canonical generalization of the LR(k) grammars.

--

Accordingly, we define an LR(k,∞) grammar as an unambiguous CFG in which every sentential form has some phrase which can be uniquely distinguished by its left context and first k characters of right context. G_1 is an example of an LR($1,\infty$) grammar which is not LR(k,t) for any k and t.

The generalization actually introduces more parsing power than intended. More specifically, if one considers the parsing patterns induced by an LR(k,∞) parser, one finds that these sets can in fact be non-regular context-free sets. This in turn implies the need for a PA with infinitely many states in order to do parsing. Even more serious is the fact that

Theorem 4.3: The class of LR(k,∞) grammars is not recursively enumerable.

Proof: Consider the set $C = \{(G',G'') \mid G'$ and G'' are LR(k) and $L(G') \cap L(G'') = \phi\}$. It is easy to show that C is not recursively enumerable.

We next show that given any two LR(k) grammars G_1 and G_2, we can effectively construct a new grammar $\bar{G}(G_1,G_2)$ such that $\bar{G}$ is LR(k,∞) iff $(G_1,G_2) \in C$. Accordingly, let $G_1 = (V_1, \Sigma, P_1, S_1)$ and $G_2 = (V_2, \Sigma, P_2, S_2)$. Without loss of generality, assume that G_1 and G_2 have disjoint sets of non-terminals. For each $\sigma_i \in \Sigma$, we replace all occurrences of σ_i in P_1 by a new non-terminal σ_i' and all occurrences of σ_i in P_2 by a new non-terminal σ_i''. At the same time we add $\sigma_i' \to \sigma_i$ to P_1 and $\sigma_i'' \to \sigma_i$ to P_2, and call the resulting sets of productions P_1' and P_2'. Now we let $\bar{G}$ be the grammar

$$\bar{G} = (V_1 \cup V_2 \cup \Sigma' \cup \Sigma'' \cup \{S\}, \Sigma, P_1' \cup P_2' \cup \{S \to S_1 | S_2\}, S).$$

It is well known that with this standard construction $\bar{G}$ is unambiguous iff $L(G_1) \cap L(G_2) = \phi$. We claim that $\bar{G}$ is LR(k,∞) iff $\bar{G}$ is unambiguous. The only if direction is obvious. For the if direction, note that if $\bar{G}$ is unambiguous then $L(G_1)$ and $L(G_2)$ are disjoint. Therefore in every sentence of $L(\bar{G})$ the last character, σ_i, will always be able to be reduced to a σ_i' or a σ_i'' by considering the left context (the entire rest of the sentence) and the right context (the $\dashv$). Once a sentential form contains a non-terminal, that non-terminal will serve as sufficient context for its neighbor until the entire string has been reduced to a word in either Σ' or Σ''. At this point the parsing schemas of the original LR(k) grammars can complete the reduction to $\vdash S_1 \dashv$ or $\vdash S_2 \dashv$.

Hence $(G_1, G_2) \in C$ iff $\bar{G}(G_1, G_2)$ is LR(k,∞). Since C is not recursively enumerable, the class of LR(k,∞) grammars cannot be recursively enumerable either. □

Corollary 4.4: It is undecidable whether a CFG is LR(k,∞) even for fixed k.

Suppose then that we try to capture the favorable aspects of the LR(k,∞) method, i.e. the ability to reduce arbitrary phrases of sentential forms, while still preserving the applicability of the theory developed in the previous section. This suggests that we must weaken the discriminatory power of the left contexts by restricting them to be regular sets. If a grammar is parsable (perhaps non-canonically) by using regular reduction patterns that are k-bounded on the right, then we say that it is FSPA(k) (the abbreviation follows from the fact that such a grammar has a finite state parsing automaton using k characters of lookahead). The results of Section 3 guarantee that

Theorem 4.5: An FSPA(k) grammar is unambiguous and is parsable in linear time on a IPDA.

Proof: Directly from Theorems 3.6 and 3.12. □

Let us next investigate whether this restriction to regular parsing patterns yields a decidable class of grammars. The BCP(m,n) grammars formed a recursive class (for fixed m and n) because of the fact that only a bounded number of potential reduction patterns needed to be checked via Theorem 3.8. On the other hand, since there are arbitrarily many regular reduction patterns that are candidates for being FSPA(k) parsing patterns, such an argument will no longer work. In fact, the question of whether an arbitrary CFG is FSPA(k) for any fixed value of k will be shown to be undecidable by the next two theorems.

The argument turns on the following general problem for deterministic languages:

Is it decidable of two arbitrary deterministic languages, L_1 and L_2, whether they are regularly separable, i.e. whether there exists a regular set R such that

$$L_1 \subseteq R \text{ and } L_2 \subseteq \bar{R}?$$

Regular separability implies the existence of a f.s.a. which accepts all strings from L_1, rejects all strings from L_2, and can do what it pleases with strings in $\Sigma^* - (L_1 \cup L_2)$. Obviously, a necessary condition for regular separability is that L_1 and L_2 be disjoint.

The regular separability problem can be shown to be undecidable by modifying Ogden's proof [Og 71] that it is undecidable whether an arbitrary CFG is LR-regular.

Theorem 4.6: It is undecidable whether two arbitrary deterministic CFL's are regularly separable.

Proof: Let M be an arbitrary Turing machine with tape alphabet Σ and state set Q. Let $id_i \in \Sigma^* \cdot (Q\times\Sigma) \cdot \Sigma^*$ represent the i-th instantaneous description of M when started on blank tape. Using well established techniques [Ha 67, RS 70] we may define the following disjoint DCFL's.

$$L_1 = \{w_1\#w_2\#\ldots\#w_{2k}\#a^k \mid k \geq 1,\ w_i \in \Sigma^*\cdot(Q\times\Sigma)\cdot\Sigma^* \text{ for } 1 \leq i \leq 2k,\ w_{2i-1} \vdash_M w_{2i}^R \text{ for } 1 \leq i \leq k\}$$

$$L_2 = \{w_1\#w_2\#\ldots\#w_{2k}\#a^{2k} \mid k \geq 1,\ w_1 = id_0,\ w_i \in \Sigma^*\cdot(Q\times\Sigma)\cdot\Sigma^* \text{ for } 2 \leq i \leq 2k,\ w_{2i}^R = w_{2i+1} \text{ for } 1 \leq i \leq k-1\}$$

To establish the theorem, we will show that L_1 and L_2 are regularly separable if and only if M never halts. We will thus have reduced the halting problem to the regular separability problem. The essential idea to be used here is that if M diverges, L_1 and L_2 will have arbitrarily long common prefixes which will "confuse" any f.s.a. which attempts to separate them.

case 1: Suppose M diverges. Assume A is an n-1 state f.s.a. which separates L_1 from L_2. Consider the string

$$z = id_0\#id_1^R\#id_1\#id_2^R\#\ldots\#id_{n!-1}\#id_{n!}^R\#$$

Then $w_1 = za^{n!} \in L_1$, and $w_2 = za^{2n!} \in L_2$.

Suppose that $\delta_A(q_0,z) = p$ and $\delta_A(p,a^{n!}) = r$.

Then by the usual repeated state arguments, there exists s such that $1 \le s \le n$ and $\delta_A(p,a^{n!+is}) = r$ for any value of i.

In particular, choose $i = n! \div s$ to see that $\delta_A(p,a^{2n!}) = r$.

Thus $\delta_A(q_0,w_1) = \delta_A(q_0,w_2)$ and hence A cannot possibly separate L_1 and L_2.

<u>case 2</u>: Suppose M halts in n steps. Thus $id_0 \vdash id_1 \vdash \ldots \vdash id_n \not\vdash id_{n+1}$ and the length of each $id_i \le n+1$.

Suppose $z = w_1\#w_2\#\ldots\#w_{2k}\#a^j \in L_1 \cup L_2$.

Then z can be classified by means of the following algorithm:

```
if k ≤ n then
    if j = k then z ∈ L1 else z ∈ L2
  else
    for i = 1 to n+1 do
        if w(2i-1) ≠ id(i-1) then z ∈ L1.
        if w(2i) ≠ id(i)^R then z ∈ L2
    end
```

The above algorithm can be performed by a f.s.a., and hence L_1 and L_2 are regularly separable. ☐

In Theorem 4.3 we constructed a grammar which was LR(k,∞) iff its two "halves" were disjoint (i.e. separable by a context-free set). Exactly this same construction can be used to combine two LR(1) grammars G_1 and G_2 into a new grammar G which is FSPA(1) iff $L(G_1)$ and $L(G_2)$ are regularly separable. We thus conclude

<u>Theorem 4.7</u>: It is undecidable (even for fixed k) whether or not an arbitrary grammar is FSPA(k).

The classes of LR(k), FSPA(k) and LR(k,∞) constitute a natural hierarchy. Each class is more powerful than its predecessor (with respect to both grammars and languages). The price paid for this power is the loss of manageable decision procedures. Thus the classes of LR(k), FSPA(k) and LR(k,∞) grammars are respectively recursive, recursively enumerable and non-r.e.

If we consider context-free grammars without Λ - rules, we can show that the classes are related as depicted in the set inclusion lattice of Figure 3. For the most part, these inclusions are reasonably straightforward consequences of the definitions. It is also not difficult to show that each inclusion is proper. One simply constructs a grammar that requires the additional parsing power of the higher class.

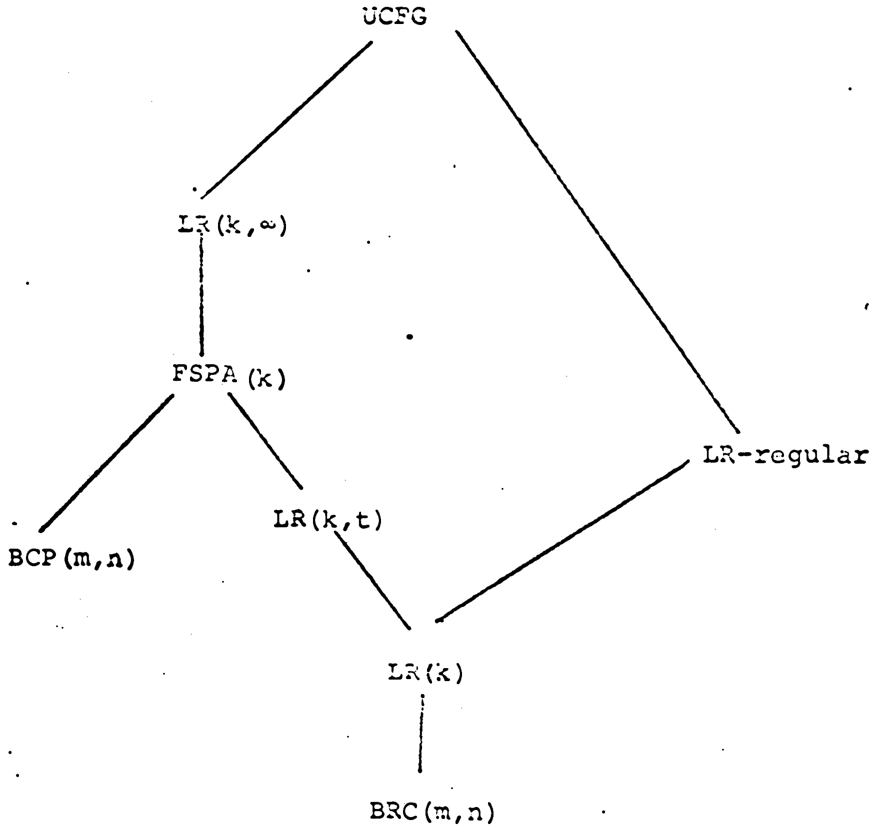


FIGURE 3: Inclusion diagram for Λ - free grammars

Let us next turn our attention to a comparison of the generative power of these classes of grammars. We say that a context-free language L is of type x if there exists some grammar of type x which generates L. It frequently turns out that unequal classes of grammars generate equal classes of languages. For example,

Theorem 4.8: The following classes of languages are equivalent:

a) Deterministic CFL's

b) LR(k) languages

c) BRC languages

d) LR(k,t) languages

Proof: The equivalence of a,b and c is due to [Kn 65]. The reason for the equivalence of a and d was alluded to in the proof of Theorem 4.2 . Details may be found in [Sz 73]. □

Closely related to the deterministic languages are the LR-regular languages which are essentially those languages which can be mapped into LR(0) languages by means of a right to left gsm transduction. Cohen and Culik have shown that the LR-regular languages properly include the deterministic languages [CC 73]. The LR-regular languages in turn are properly contained within the BCP languages, a fact which is proven below.

Theorem 4.9: The class of BCP languages properly contains the class of LR-regular languages.

Proof: a) We will first demonstrate containment.

Let L be any LR-regular language and g be its associated right to left gsm. Suppose that the alphabet of L is Σ and the set of states of g is Q. Suppose further that the start state q_0 of g is not in the range of the move function δ_g. Define a new language L' with alphabet $Q\times\Sigma\times Q$ by

$$L' = \{(p_0,a_1,p_1)(p_1,a_2,p_2)\dots(p_{k-1},a_k,p_k) \mid k\geq 1,\ a_1a_2\dots a_k \in L,\ p_k = q_0,\ \delta_g(p_i,a_i) = p_{i-1} \text{ for } 1\leq i\leq k\}$$

L' then is an LR(0) language. Williams has shown that every deterministic language is a BCP language [Wi 72]. So, let $G' = (N', Q\times\Sigma\times Q, P', S')$ be a BCP grammar generating L'.

We next define a new grammar

$$G'' = (N' \cup Q\times\Sigma\times Q,\ \Sigma,\ P' \cup P'',\ S')$$

by adding some productions to G'. More specifically we let

$$P'' = \{(p,a,q) \rightarrow a \mid \delta_g(q,a) = p\}.$$

Clearly $L(G'') = L$. We also claim that G" is a BCP grammar. The necessary reduction patterns are

(a) for $\pi \in P'$ the appropriate pattern for parsing G'.

(b) for $(p,a,q) \rightarrow a \in P''$ then $\{a\#(q,b,s) \mid b \in \Sigma,\ s \in Q\}$ if $q \neq q_0$
$\{a\#\dashv\}$ if $q = q_0$.

Thus G" is a BCP grammar for L.

b) To see that the containment is proper, consider $L(G_1)$ from Section 1. $L(G_1)$ has been shown not to be LR-regular [CC 73], yet clearly is BCP(1,1) as demonstrated in Section 1. ☐

Thus the ability to parse non-canonically significantly broadens the class of recognizable languages beyond the deterministic languages. It is obvious from the definitions that the BCP languages are contained within the FSPA languages. However, we have not been able to prove or disprove that this containment is proper.

We turn now to analyzing the power of the $LR(k,\infty)$ languages. Our results are presented in a sequence of lemmas.

Lemma 4.10: Every FSPA(k) language is an $LR(k,\infty)$ language.

Proof: Recall that the FSPA(k) grammars were defined by a restriction applied to the $LR(k,\infty)$ grammars. Hence every FSPA(k) grammar is an $LR(k,\infty)$ grammar and the lemma follows immediately. ☐

Lemma 4.11: The sets $\{a^n c\, b^n | n \geq 1\}$ and $\{a^n c\, b^{2n} | n \geq 1\}$ are not separable by any regular set.

Proof: Left to the reader.

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

Proof: This result was first shown by Hennie [He 65]. ☐

Lemma 4.13: There exist $LR(1,\infty)$ languages which are not FSPA(k) for any value of k.

Proof: Consider $L = \{a^n c\, b^n | n \geq 1\} \cup \{a^n c\, b^{2n} | n \geq 1\}$. L can easily be shown to have an $LR(1,\infty)$ grammar. We will show that no grammar for L can be FSPA(k).

Suppose that some grammar G for L were FSPA(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 non-terminal 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, some sentential form of 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 $2 \times |y|$ we produce a string which certainly is not in L.

2) If α is a sentential form of G then there can be at most one non-terminal in α 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. We claim that either $n = m$ or $n = 2m$. Using 1 above and the fact that G is reduced, there exist 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 α which derives a string in $\{a^n c b^n \mid n \geq 1\}$.

Suppose otherwise. Thus $\alpha = \beta_1 A \beta_2$ and $\alpha \overset{*}{\Rightarrow} a^n c b^n$ for some n. Then there exist q,r,s,t such that $\beta_1 \overset{*}{\Rightarrow} a^q$, $A \overset{*}{\Rightarrow} a^r c b^s$ and $\beta_2 \overset{*}{\Rightarrow} b^t$. Furthermore, $q + r = s + t$. Since $\beta_1 A \beta_2$ is a sentential form, so are $a^q A b^t$, $a^q a^{km} A b^{2km} b^t$, and $a^q a^{km} a^r c b^s b^{2km} b^t$. Examining 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.

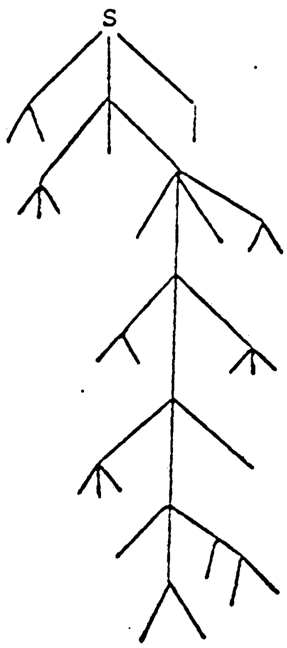
$V_1 = \{A \in V | L(G_A) \text{ is finite}\}$

$V_2 = \{A \in V | L(G_A) \text{ is infinite but A is not recursive}\}$

$V_3 = \{A \in V | \text{A is recursive and } A \overset{+}{\Rightarrow} a^m A b^m \text{ for some m}\}$

$V_4 = \{A \in V | \text{A is recursive and } A \overset{+}{\Rightarrow} a^m A b^{2m} \text{ for some m}\}$

6) There 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 FSPA(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 non-terminal 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 $\{a^n cb^n | 1 \leq n \leq q/2\}$. Thus M' separates $\{a^n cb^n \; n \geq 1\}$ from $\{a^n cb^{2n} | n \geq 1\}$. 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 non-terminals 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 $\{a^n cb^n | n \geq 1\}$ from $\{a^n cb^{2n} | n \geq 1\}$.

This, of course, contradicts Lemma 4.11 so we conclude that G could not have existed in the first place. Hence no grammar for G is FSPA(k). ∎

These lemmas can be combined to give us the desired result, namely,

<u>Theorem 4.14</u>: The class of LR(k,∞) languages properly includes the FSPA(k) languages.

<u>Proof</u>: Inclusion follows from Lemma 4.10. The fact that this inclusion is proper is a consequence of Lemma 4.13. ∎

The relationships between the classes of languages induced by the parsing methods we have studied is depicted in Figure 4. Solid lines indicate proper containment whereas dashed lines indicate a containment which has not yet been shown to be proper. It is interesting to note that the fairly "messy" lattice of inclusions of grammar classes shown in Figure 3 collapses into a linear order when viewed in language space.

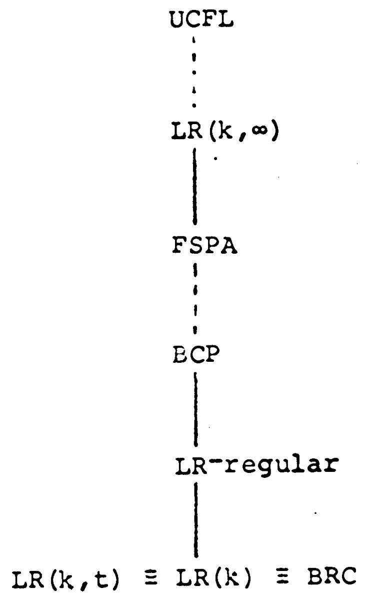


FIGURE 4: Inclusion lattice for some classes of Λ - free languages.

Section 4 Conclusions and Open Problems

We have attempted to construct a general framework for bottom-up parsers and, within that framework, to examine the non-canonical extensions of some existing methods. It has been shown that the only non-canonical extension to enjoy both linear time parsability and decidability is the BCP technique. We intend to consider further restrictions to the FSPA method to see if we can achieve decidability with some class significantly more general than the BCP grammars. Even so, we conjecture that in language space the two classes will prove to be identical, probably for the same reason that BRC languages are the same as LR(k) languages.

Additional extensions are possible within the current framework. Define a grammar G as being Regular Pattern Parsable (RPP) if there exists a parsing scheme for G, each of whose reduction patterns is a regular set (possibly unbounded on both sides of the #). The class of RPP grammars is essentially the non-canonical extension of the LR-regular grammars. This class is non-recursive because it contains the FSPA(k) grammars yet certainly is recursively enumerable (Theorem 3.8). Several interesting questions can now be asked. Does every unambiguous context-free language have an RPP grammar? Can RPP grammars be parsed in linear time and if so, what sort of computer is needed to achieve this bound?

We finally point out that the requirement that a parser be able to locate a phrase in any sentential form is overly restrictive. What is really desired is that every sentential form which arises during parsing be reducible. Is it possible to refine the theoretical framework of Section 3 to incorporate this idea? As an example, one can easily construct a grammar G for $\{ww^R | w \in \{a,b\}^*\}$ and a parser for G which utilizes the order in which reductions are made to produce a parser while still only using a finite number of states.

APPENDIX

Grammars used in constructing the lattice of Figure 3.

Lemma A.1: G_1 of the paper is BCP but not LR(k,t) nor LR-regular.

Lemma A.2: G_2 of the paper is LR(k,t) but not LR-regular.

Lemma A.3: G_3 below is LR(1,∞) but not FSPA(k).

G_3: $S \rightarrow A \mid B$

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

$B \rightarrow aB\bar{B} \mid a\bar{B}$

$\bar{A} \rightarrow b$

$\bar{B} \rightarrow bb$

Lemma A.4: G_4 below is LR-regular and FSPA(1) but not BCP nor LR(k,t).

G_4: $S \rightarrow aAa \mid bAb \mid aBb \mid bBa$

$A \rightarrow \bar{A}A \mid c$

$B \rightarrow \bar{B}B \mid c$

$\bar{A} \rightarrow c$

$\bar{B} \rightarrow c$

Lemma A.5: G_5 below is unambiguous but neither LR(k,∞) nor LR-regular.

G_5: $S \rightarrow aSa \mid bSb \mid aa \mid bb$

BIBLIOGRAPHY

[AU 72] Aho, A.V., and J.D. Ullman, The Theory of Parsing, Translation and Compiling, Volume 1 and 2, Prentice-Hall.

[CC 73] Culik, K., II., and R. Cohen, "LR-Regular Grammars - An Extension of LR(k) Grammars", Journal of Computer and System Sciences 7, pp. 66-96.

[Co 70] Colmerauer, A., "Total Precedence Relations", Journal of the Association for Computing Machinery, 17:1, pp. 14-30.

[Fl 64] Floyd, R.W., "Bounded Context Syntactic Analysis", Communications of the Association for Computing Machinery 7:2, pp. 62-67.

[Ha 67] Hartmanis, J., "Context Free Languages and Turing Machine Computations", Proceedings of Symposium in Applied Mathematics, Volume 19, American Mathematical Society.

[He 65] Hennie, F.C., "One-tape, Off-line Turing Machine Computations", Information and Control, 8:6, pp. 553-578.

[Kn 65] Knuth, D.E., "On the Translation of Languages from Left to Right", Information and Control, 8:6, pp. 607-639.

[Og 71] Ogden, W.F., Unpublished memorandum dated Dec. 1971.

[RS 70] Rosenkrantz, D.J., and R.E. Stearns, "Properties of Deterministic Top Down Grammars", Information and Control 17:3, pp. 226-256.

[Sz 73] Szymanski, T.G., "Generalized Bottom-up Parsing", Ph.D. Thesis, Dept. of Computer Science, Cornell University, Ithaca, NY.

[Wi 72] Williams, J.H., "Bounded Context Parsable Grammars", Dept. of Computer Science, Cornell University Technical Report 72-127.