Noncanonical Extensions of LR Parsing Methods*

Michael D. Hutton
University of Waterloo†

August, 1990

Abstract

Bottom up LR-parsing is the most powerful, practical method of linear time parsing known. Various attempts have been made to enhance the power and flexibility of the LR method, without losing the property of linear time parsing, or introducing inordinately large constant factors which prohibit practicality.

In this paper, we survey the particular line of this work which adds power through allowing noncanonical operation of the LR parser — reducing non-leftmost phrases. We show that for some natural extensions, the use of noncanonical parsing is beneficial in enlarging the class of accepted grammars or languages, while for others, the class of resulting grammars constitute a non-recursive set.

The original goal of this work was to investigate the particular class of grammars parsable by noncanonical LR parsers using LALR lookahead. Unfortunately, this particular class of grammars is not believed to be recursively enumerable in its full generality. However, by weakening the noncanonical extension of the parser to constant sized paths, analagous to Knuth's LR(k, t) grammars, we can achieve a recursive class of grammars, which yield linear time, practical parsers.

1 Introduction

The well known LR(k) parsing algorithm of Knuth [10] and its two major variations SLR(k) and LALR(k) due to DeRemer [5] are *canonical* in that they reduce only leftmost phrases or handles, with a constant (k) symbols of terminal lookahead. Such parsers can be implemented by a single stack machine with a constant sized parse table (a pushdown automaton).

A noncanonical extension would be to allow the parser to perform parsing/reduction during the lookahead phase, and use the resulting nonterminal symbols as lookahead characters for the temporarily put-off action. Such reductions would not involve leftmost phrases. This modification does extend the class of grammars accepted [14, 15], as well as allowing a more expressive and natural notation for grammar specification [13]; preventing the well known problem of "contorting" the grammer to make it fit the class of the parser generator.

Obviously such an extended class of grammars is useful only if membership in the class is decidable – ie. if a correct parser generator exists to accept the language, and output a parser. When treated less formally, it may be acceptable to allow for an incorrect but "close" parser generator for a non-recursive set, in the case where it can be shown that the "problem" elements of the set do not arise in practice, but we will not consider this case here.

A second criteria would be that the resulting parser has linear time complexity. Szymanski and Williams [14] have shown that this property follows whenever the grammar is unambiguous. This is promising, however it

*This document was prepared as a term paper for CS 744 at the University of Waterloo, in August 1990. It should be considered a limited circulation draft paper, as it has not been reviewed other than by the author. M.H. 10/92.

†Authors current address is: Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4. mdhutton@cs.toronto.edu.

proves only the *existance* of such a parser, and does not yield an algorithm for its discovery (exactly because such algorithms do not always exist).

A third criteria is that the linear time parser have a practical realization. Some parsing methods fail this criteria; in particular Williams' noncanonical extension BCP(m, n) [16] to the Bounded Context parsing method of Floyd [7] is the largest known class of recursive, linear-time parsable grammars, but it requires the existance of a very large (but constant) sized parse table, and is not of practical use in the design of efficient compilers.

In this paper, we first survey the known methods of providing noncanonical extensions to LR parsers, then give a new class of grammars which provides bounded path LALR(1) lookahead to a noncanonical LR parsing automaton. This class is the obvious restriction of the noncanonical LR(k, t) class of Knuth [10]. The new work consists of an algorithm for constructing parsers for this class, definition of the properties and expressive power of the class, an extension of the NLR(k) undecidability proof of Szymanski and Williams to intuitively prove/conjecture that the full generalization, the set of NLALR(k) grammars, is not recursively enumerable. As well, we show some new containments with regard to the NSLR(k) and NLALR(k) classes.

Sections 2 and 3 review formal language notation, and LR parsing respectively. Section 4 introduces the LR-regular grammars of Culik and Cohen [4]. In section 5, we overview the Bounded Context parsing method of Floyd [7], and its noncanonical extension, the BCP(m, n) grammars of Williams [16]. Section 6 introduces noncanonical extensions of LR parsers: NLR(k) and LR(k, t) classes and their properties; NSLR(k) grammars, their properties and a construction algorithm for NSLR(1) prasers; and NLALR(k) grammars and LALR(k, t) grammars.

In some ways, the historical presentation is not in the natural evolutionary order, but it serves better our purpose of discussing the evolution of noncanonical parsing. We have chosen not to include precedence-type parsing methods in our discussion; the reader is referred to the works cited by Tai [15] and Szymanski and Williams [14].

2 Preliminaries

A *context free grammar (CFG)* G is a a quadruple (N, T, P, S), where N is a set of *nonterminal* symbols, T is a set of *terminal* symbols, P a set of *productions*, and S the starting nonterminal symbol. N and T together comprise the *vocabulary* V of G. We use the standard notational conveniences unless explicitly stated otherwise: $A, \dots, F, S \in N$; $a, \dots, h \in T$; $i, \dots, n$ are integers or indices; $p, \dots, t$ are states in a finite state machine or FSM; $X, Y, Z \in V$; $u, v, \dots, z \in T^*$ (the reflexive transitive closure of T) are called *words* or *sentences* of the language $L(G)$ of G; $\alpha, \beta, \gamma, \delta, \omega, \in V^*$; ε is the *empty word* (terminal string with null contents); and $|\gamma|$ is the integer *length* of γ.

The *produces* relation "$\Rightarrow$" is defined on V^* such that $\alpha A \gamma \Rightarrow \alpha\omega\gamma$ whenever $\alpha, \gamma, \omega \in V^*$, and $A \rightarrow \omega \in P$. The transitive and reflexive transitive closures of $\Rightarrow$ are correspondingly denoted $\Rightarrow^+$ and $\Rightarrow^*$. All three are read as "produces". If $S \Rightarrow^* \alpha$ then α is a *sentential form* derivable from S, and if $\alpha \in T^*$ then it is a *sentence* (*word*). A series of zero or more consecutive productions, $S \Rightarrow \alpha_1 \Rightarrow \cdots \alpha_k \Rightarrow w$ is called a *derivation*. In a (canonical) rightmost derivation, the rightmost nonterminal of a given sentential form α_i is always the one expanded to generate the next sentential form α_{i+1}. A substring β of a sentential form α is called a *phrase* if there exists a derivation $S \overset{*}{\Rightarrow} \delta A \gamma \Rightarrow \delta\beta\gamma = \alpha$. If β is the *leftmost* (complete) phrase of α it is called a *handle*.

A grammar G is *ε-free* if it contains no productions $A \rightarrow \varepsilon$. G is *unambiguous* if every $x \in L(G)$ has a unique leftmost (rightmost) derivation, and *ambiguous* otherwise (there exists some sentence x in $L(G)$ generated by two or more distinct leftmost (rightmost) derivations).

The subclass of the ε-free CFG's we will consider comprise a lattice under the order imposed by containment. A lattice is a partially ordered set such that every two elements (here classes) A and B have a unique least upper bound (class C containing $A \cup B$) and greatest lower bound (class D contained in both A and B). A *class* of grammar is a set of grammars under some "defined property", such as "LR-regular" or "are ambiguous". If $C_1 \subset C_2$ then C_2 has greater *expressive power*, and if neither $C_1 \subseteq C_2$ nor $C_2 \subseteq C_1$ holds, then the two classes are *incomparable*.

The class of languages generated by these grammars as well form a lattice under inclusion, although not necesarily the same one for the underlying grammar lattice. The *language* of a grammar G is denoted $L(G)$, and consists of all sentences generated by G. The language class of a grammar class C is denoted $\mathcal{L}_C$, and comprises the set of all languages generated by the set of grammars in G. If $\mathcal{L}_{C_1} \subset \mathcal{L}_{C_2}$ then C_2 has greater *generative power*, and if neither $\mathcal{L}_{C_1} \subseteq \mathcal{L}_{C_2}$ nor $\mathcal{L}_{C_2} \subseteq \mathcal{L}_{C_1}$ holds, then the two classes are *incomparable*. Note that if $C_1 \subseteq C_2$, then any C_1 grammar is also C_2, so $\mathcal{L}_{C_1} \subseteq \mathcal{L}_{C_2}$.

Two grammar classes C_1 and C_2 are *equal* if $G \in C_1$ if and only if $G \in C_2$. This necessarily implies $\mathcal{L}_{C_1} = \mathcal{L}_{C_2}$. However the converse does not apply, as we will later see with the grammatical classes LR(k) and BRC(m, n), which are not equal, but have the same generative power (yield the same class of languages). Several basic classes of grammars are: UCFG (unambiguous CFG), DCFG (determnistic CFG), and $\emptyset$ (the empty class); which generate CFL $= \mathcal{L}_{\text{CFG}}$, UCFL $= \mathcal{L}_{\text{UCFG}}$, DCFL $= \mathcal{L}_{\text{LR(k)}} = \mathcal{L}_{\text{DCFG}} = \mathcal{L}_{\text{BRC(m,n)}}$, and $\emptyset$ respectively.

3 LR parsing

The opertation of an LR "parser", also known as a *bottom up* or *shift-reduce* parser, is to scan an input word from left to right, and construct the reverse of the rightmost derivation.

More formally, a CFG *parsing automaton* $M(G)$ for G is a finite state machine with a single stack, represented by $(Q, N, T, S, \textbf{Next}, \textbf{Reduce})$, where Q is the finite set of states of $M(G)$, N, T and S are as in G, **Next** is the transition function of $M(G)$ and **Reduce** is the reduction function of G. **Next** and **Reduce** comprise the parse table of $M(G)$. The transition $p \xrightarrow{X} q$ is represented by an entry $\textbf{Next}(p, X) = q$, and is called a *terminal transition* if $X \in T$, and a *nonterminal transition* if $X \in N$.

The two basic actions are to *read* an additional character of input (from **Next**) on to the parse stack, or *reduce* a handle on the parse stack to the nonterminal symbol which generates it. The parse ends when the parse stack contains the start symbol and the input is exhausted. The rightmost derivation generated by the operation is the *parse* of the input word. A CFG is LR(k) if such a parser exists which can always make the decision to reject, shift or reduce (and which nonterminal to reduce to) with a constant k symbols of *lookahead* into the remaining terminal string, ie. if it can be dedided by a deterministic finite state automaton with one stack. For further details on LR parsing, or the construction of the LR parser the reader is refered to a Compilers text such as Aho, Sethi and Ullman [1]. We will review by simply stating the definitions of the interesting components of the development, and giving an informal algorithm for the construction of the LR(k) parsing machine. Note that the "k-sets" below are truncated to smaller lengths when appropriate (ie. when they are derived from a string of length $< k$). We also have a concatanation operator $\oplus_k$ which concatanates to a maximum length of k.

The set of *first* terminals in a string $\alpha \in V^*$:

$$T\_FIRST_k(\alpha) = \left\{ x \in T^k \mid \alpha \overset{*}{\Rightarrow} x\beta \right\}.$$

The set of *last* elements is defined analogously:

$$T\_LAST_k(\alpha) = \left\{ x \in T^k \mid \alpha \overset{*}{\Rightarrow} \beta x \right\}.$$

We assume that the input word is always delimited by begin/end markers — \$, and that the starting production is unique. The latter is usually accomplished with the addition of a production $S' \to S$, and S' replacing S as the starting symbol.

The set of following terminals of a nonterminal A, in a sentential form of G:

$$T\_FOLLOW_k(A) = \left\{ x \in T^k \cup T^{k-1}\$ \mid \$S'\$ \overset{*}{\Rightarrow} \beta A x \gamma,\ \beta \in \{\$V^*\},\ \gamma \in \{V^*\$\} \cup \{\varepsilon\} \right\}.$$

We denote the *items* of an LR machine $M(G)$, with respect to each production $A \rightarrow \alpha$ as

$$[A \rightarrow \beta \bullet \gamma, u],$$

where $\alpha = \beta\gamma$. Each item represents a partition of the right hand side of a produtions. Informally, the item represents a production which is "partially recognized", so we call items as above where $\beta = \alpha \quad (\gamma = \varepsilon)$ *complete* items, denoted $[A \rightarrow \alpha\bullet, u]$. The *lookahead* string u denotes possible terminal strings to follow the nonterminal A of this item.

$M(G)$ states are elements of the power set of items. There are two types of items in a state, *kernel* items, which define the state, and *closure* items, which complete the state, and are derived through a closure operation on the kernel items: The *closure* of a state q is defined recursively as the smallest set $q' \supseteq q$ such that:

$$[X \rightarrow \bullet\omega, FIRST_k(\beta u)] \in q' \text{ if } [A \rightarrow \alpha \bullet X\beta, u] \in q' \text{ and } X \rightarrow \omega \in P.$$

The transition, or GOTO function on the CFSM is defined by:

$$GOTO(q, X) = \{[A \rightarrow \alpha X \bullet \beta, u] \mid [A \rightarrow \alpha \bullet X\beta, u] \in q\}.$$

We can extend this function to its closure, by allowing an arbitrary string in V^* to replace X:

$$\begin{aligned} GOTO(q, \varepsilon) &= q \\ GOTO(q, X\alpha) &= GOTO\,(GOTO(q, X), \alpha). \end{aligned}$$

The inverse of the GOTO function is the PRED relation, which defines the predecessors $\{q\}$ of a state p under a series of GOTO's.

$$PRED(p, \alpha) = \{q \mid p \in GOTO(q, \alpha)\}$$

The definitions GOTO and closure define a DFA for the recognition of "viable prefixes" of G in the parsing of a word x; in essense a handle recognizer. The machine contains a special operation to change its input and move back through machine on a reduce move[1]. To reduce $A \rightarrow \alpha$, the machine "backs up" $|\alpha|$ steps in its path through the FSM path, inserts an "A" in the input, and continues in order to find the next handle. A shift is a standard transition in the FSM, and recognition of the start nonterminal symbol triggers the accept state.

The construction of the LR(k) machine M proceeds as follows:

```
Add $[S' \rightarrow \alpha\bullet, \$]$ to $q_0$.
Compute $q_0 \leftarrow$ closure($q_0$), and add it to the list of incomplete states.
For each incomplete state $q$, do:
    remove $q$ from the list (mark it complete).
    for every $X \in V$ such that some $[A \rightarrow \beta \bullet X\gamma, u] \in q$, do:
        $q' \leftarrow \{[A \rightarrow \beta X \bullet \gamma, u] \mid [A \rightarrow \beta \bullet X\gamma, u] \in q\}$.
        $q' \leftarrow$ closure($q'$).
        if $q'$ already exists in $Q(M)$,
            add $q \xrightarrow{X} q'$ to GOTO($M$).
        else
            add $q'$ to $Q(M)$, $q \xrightarrow{X} q'$ to GOTO($M$), and mark $q'$ incomplete.
Continue until no more incomplete states.
```

An *inadequate* state in M is one for which the machine cannot deterministically decide what move to make: These fall into two classes, as illustrated in Figure 1: q_1 is an LR(1) inadequate state because of the *reduce-reduce conflict* – the machine cannot deterministically decide, given one character of lookahead, whether to reduce the phrase α to the nonterminal A or the phrase β to the nonterminal B. State q_2 is inadequate as a result of a *shift-reduce conflict* – the machine cannot deterministically decide whether to reduce α to A, or shift to a new state on the terminal symbol a.

[1]In reality, the machine requires a stack to do this, so it is implicitly the pushdown automaton described earlier

$$q_1 \quad \boxed{\begin{array}{l}[A \rightarrow \alpha\bullet, a] \\ [B \rightarrow \beta\bullet, a]\end{array}} \qquad\qquad q_2 \quad \boxed{\begin{array}{l}[A \rightarrow \alpha\bullet, a] \\ [B \rightarrow \gamma \bullet a\beta, b]\end{array}}$$

Figure 1: Inadequate parser states; reduce-reduce and shift-reduce conflicts.

In a *canonical* parse, only handles are reduced, so the lookahead strings contain only terminals. In a *noncanonical* parse, other phrases may be also reduced, so the lookahead strings can contain any vocabulary symbols (including nonterminals).

Although containing a strong theoretical background, and generating all of the deterministic languages, the LR(k) parsing method fails in the third criteria we previously mentioned. The constructed parser contains a very large, constant sized table, much of which would not be required to deterministically parse words of the input grammar. This applies even to the LR(1) class, and increases dramatically for $k > 1$. As such, LR parsing was initially considered to be of theoretical interest only..

Two restrictions on LR(k) parsing, due to DeRemer [6], have made LR parsing practical and attractive, with minimal reduction in parsing power. The first adds k characters of lookahead to the LR(0) machine (the characteristic finite state machine – CFSM) and is called SLR(k), or *simple* LR(k). The resulting parser has the same number of states as the CFSM, but gains some of the power of the full LR(k) machine. The second type is called LALR(k) (*lookahead* LR(k)), and is constructed from the LR(k) machine by merging states to gain the smaller (same size as LR(0)) machine.

It should come as no surprise that the power of these classes is a proper ordering:

$$\mathrm{LR}(k) \supset \mathrm{LALR}(k) \supset \mathrm{SLR}(k)$$

3.1 SLR lookahead

To construct an SLR(k) parser, we first precompute the T_FIRST_k and T_FOLLOW_k sets as previously defined, then the LR(0) CFSM is defined using the LR construction algorithm with null lookahead sets.

Since items of the CFSM have no lookahead, we will simplify their definition to $[A \rightarrow \beta \bullet \gamma]$.

A CFSM state is inadequate if it contains more than one complete item, since there is no lookahead to distinguish betweeen multiple reductions.

The SLR(1) machine attempts to resolve the conflicts through the adding of lookahead. SLR(1) lookahead is computed, on the complete items of each state, as follows:

$$LA_k(q, A \rightarrow \alpha) = \{x \mid x \in T - FOLLOW_k(A)\}$$

The correctness follows because LA(q,$A \rightarrow \alpha$) simply states that parsing can continue by reducing α to A only if the following symbol in the input stream could possibly arise after A in some derivation from S', obviously necessary for the result to be a sentential form deriving a word $x \in L(G)$.

Note that the simple LR lookahead for $A \rightarrow \alpha$ is independent of the state in which it is being applied (other than that the state contains the production $A \rightarrow \alpha$). This differs from the full LR lookahead, which enforces that parsing continue with reduction $A \rightarrow \alpha$ only if the following symbol could possibly arise after A in some derivation *consistent with the current state of the parsing automaton.* The construction of full LR lookahead, and similarly the LALR lookahead to be discussed presently, is tied to the construction of the parsing automaton itself.

3.2 LALR lookahead

The method we will present for calculating LALR lookahead is a variation proposed by Park, Choe and Chang [11] on the standard "recursive lookback" method. More efficient algorithms do exist, the most notable being DeRemer and Penello's graph traversal algorithm [6], which was also improved by Park, Choe and Chang's variation [11]. We chose the former, less efficient, version because it extends to the noncanonical case without hiding noncanonical operation in the more complicated lookahead calculation.

The extension of Park, Choe, and Chang is to re-define the LALR calculation iteratively, so as to remove the implicit recursion from the calculation of closures. Using this, it can be shown that the LALR lookahead really depends only upon the kernel items, rather than the entire states, under a suitable formalism which they introduce.

Note that the following definitions are applicable to LR parsing in general, and are not yet specific to LALR calculation.

We begin with a re-definition of the calculation of closure with a new operator δ on sets of items:

$$\delta\left(\{[A \rightarrow \alpha \bullet X\beta, u]\}\right) = \{[X \rightarrow \bullet\omega, T\_FIRST_k(\beta u)] \;\mid\; X \rightarrow \omega \in P\},$$

($\delta(\{I\}) = \emptyset$ if I is complete or $X \in T$). So the closure of a set of items (state q) can be defined by:

$$CLOSURE(q) = \delta^*(q).$$

The recursiveness of this operator, and the definition of CLOSURE itself, arises from the sequences of productions of the form:

$$\begin{array}{rcl} B & \rightarrow & B_1\beta_1 \\ B_1 & \rightarrow & B_2\beta_2 \\ & \vdots & \\ B_{n-1} & \rightarrow & B_n\beta_n, \end{array}$$

which induces a relation[2] on the B_i's,

$$B_1 \rhd B_2 \text{ iff } B_1 \rightarrow B_2\beta_2 \in P.$$

If we define

$$PATH_k(B, C) = \bigcup\{T\_FIRST_k(\beta_n \cdots \beta_1) \;\mid\; B_0 = B, B_n = C\}$$

for the B_i productions a just shown, we can re-define δ in terms of $\rhd$ and $PATH_k$:

$$\begin{array}{rl} \delta\,\{\,[\,A \rightarrow & \alpha \bullet B\beta, u\,]\} \\ & = \left\{[C \rightarrow \bullet\gamma, PATH_k(B, C) \oplus_k T\_FIRST(\beta u)] \;\mid\; B \rhd^{n-1} C, C \rightarrow \gamma \in P\right\}. \end{array}$$

The $\rhd$-graph is a digraph induced by $\rhd$ with the edges marked by the terminal path of the β_i's,

$$\begin{tikzcd} B = B_0 \arrow[r, "\beta_1"] & B_1 \arrow[r, dash, "\beta_2"] & \cdots \arrow[r, "\beta_n"] & C = B_n \end{tikzcd}$$

so the $PATH_k$ notion above represents the set of first k terminal symbols derivable on this path.

The conclusions of this derivation, as proven in [11], are that the closure operation can be calculated iteratively as

$$\delta^+\,\{[A \rightarrow \alpha \bullet B\beta, u]\} = \{[C \rightarrow \bullet\gamma, PATH_k(b, c) \oplus_k FIRST_k(\beta u)] \;\mid\; B \rhd^* C,\;\; C \rightarrow \gamma \in P\},$$

[2] Park, Choe and Chang denote this relation "L". We have chosen to use $\rhd$ which makes it more readable as an operator, and also enforces its non-reflexive nature.

using only the kernel items, and a single traversal of the $\triangleright$-graph.

Under this formalism, we can now define $LALR$ lookahead for an item in state p by:

$$\begin{aligned} LA_k\,(p,[A\rightarrow\alpha_1\bullet\alpha_2]) \;=\; \{\, x \mid\; & x\in PATH_k(A',A) \\ & \quad\oplus_k\, FIRST_k(\beta_2) \\ & \quad\oplus_k\, LA_k\,(q,[B\rightarrow\beta_1\bullet A'\beta_2])\,, \\ & q\in PRED(p,\alpha_1), \\ & A'\,\triangleright^*A, \\ & [B\rightarrow\beta_1\bullet A'\beta_2]\in kernel(q)\,\}\,. \end{aligned}$$

Intuitively, the lookahead for $[A\rightarrow\alpha_1\bullet\alpha_2]$ in state p is propagated from any state q which is a predecesor of p in the CFSM under a path α_1. The lookahead consists of any closure induced lookahead first, then the string β_2 following the nonterminal A' in the production of predecessor state q, and then the lookahead from that item itself, all truncated at k characters. So, $LA_k\,(p,[A\rightarrow\alpha_1\bullet\alpha_2])$ can also be defined as:

$$\bigcup_{q\in PRED(p,\alpha_1)} \;\; \bigcup_{\substack{\mathrm{A'}\,\triangleright^*\,\mathrm{A},\\ [\mathrm{B}\rightarrow\beta_1\bullet\mathrm{A'}\beta_2]\,\in\,\mathrm{kernel(q)}}} PATH_k(A',A)\;\oplus_k\; FIRST_k(\beta_2)\;\oplus_k\; LA_k(q,[B\rightarrow\beta_1\bullet A'\beta_2]).$$

The computation of LALR lookahead proceeds as follows:

```
LALR($p$, $[A \rightarrow \alpha_1 \bullet \alpha_2]$ =
{
    LALR $\leftarrow$ {}
    for $q \in$ PRED($p, \alpha_1$) do
        for $[B \rightarrow \beta_1 \bullet A'\beta_2] \in$ kernel($q$), where $A' \triangleright^* A$, do
            LALR $\leftarrow$ LALR $\cup$ PATH($A', A$);
            if $\varepsilon \in$ PATH($A', A$) do
                LALR $\leftarrow$ LALR $\cup$ FIRST($\beta_2$);
                if $\varepsilon \in$ FIRST($\beta_2$) do
                    LALR $\leftarrow$ LALR $\cup$ LALR($q, [B \rightarrow \beta_1 \bullet A'\beta_2]$)
    return LALR;
}
```

This is a straightforward implementation of the formula above. It can be shown [11] that this formalization generates a significant improvement over previous recursive methods, since the multiple calculation of closure is eliminated, and replaced with a single traversal of the $\triangleright$-graph, linear time with depth first search.

4 LR-Regular Parsing

An LR-Regular grammar [4] is one which can be parsed deterministicaly on an LR(0) machine with a finite number of regular lookahead sets to resolve state conflicts. The LR-Regular class is interesting to the discussion at hand, as it is one of the first extensions to canonical LR parsing, and it is a formally interesting class with which to compare the properties of the noncanonical classes to be covered.

More formally: a *regular partition* of a set X is a partition of X into n disjoint regular sets, whose union is X; a grammar $G=(N,T,P,S)$ is *LR-Regular* if there exists some regular partition $\pi=\{R_1,R_2,\ldots,R_n\}$ of T^*, and the handle in any right sentential form is uniquely determined by its left context, and some (appropriate) R_i.

A language $L\in\mathcal{L}_{\mathrm{LRR}}$ if there exists some $G\in$ LRR such that $L(G)=L$.

An LRR parser operates by first making a right to left scan of the input, and "compressing" the lookahead information by adding tags to certain terminals indicating the regular sets to follow. The LR(0)–like parser then operates left to right, using the lookahead tags to resolve nondeterministic options in the LR(0) states. Since the number of regular lookahead sets is constant, the first pass requires linear time, and the linearity of the second pass follows from the correctness of LR parsing.

Any LR(k) grammar is obviously also an LRR grammar, as any finite language (such as an LR(k) lookahead set) is a regular language. The language class containment follows as well from this property. To show that the containment is proper, we consider the following grammar G_1, and the language $L_1 = L(G_1)$:

$$\begin{array}{llll} G_1: & S & \rightarrow & aCb \mid bCa \mid aDa \mid bDb \\ & C & \rightarrow & aCaa \mid b \\ & D & \rightarrow & aDa \mid b \end{array}$$

$$\begin{array}{lll} L_1 & = & \{aa^nba^{2n}b\} \\ & \cup & \{ba^nba^{2n}a\} \\ & \cup & \{aa^nba^na\} \\ & \cup & \{ba^nba^nb\} \end{array}$$

G_1 is LRR with respect to the partition $\pi = \{T^*a, T^*b\}$, (ie. the lookahead needs only to distinguish strings ending with a or ending with b in order to resolve state conflicts), so $L_1 \in \mathcal{L}_{\text{LRR}}$. G_1 is not however LR(k) for any k, and L_1 is not deterministic, so no other LR(k) grammar can define L. Thus the classes of LR(k) grammars and LR(k) languages (the DCFL's) are properly contained in the classes of LRR grammars and LRR languages respectively.

We can also note, by definition, that $\mathcal{L}_{\text{LRR}}$ contains only unambiguous languages. Otherwise, the parser could not be deterministic. Furthermore, the well known unambiguous language of even length palindromes,

$$EPAL = \left\{ ww^R \,|w \in \{a,b\}^* \right\},$$

is not LRR, since the lookahead sets are necesarily non-regular.

Culik and Cohen's introduction to LR-Regular grammars includes a formal study of their containment and closure properties. Of particular interest is that "every LR(π) [LRR with respect to some regular partition π] language is obtainable by a homomorphism from some $L \in \mathcal{L}_{\text{DCFL}}$". This directly implies nearly all of the closure properties of the (well studied) deterministic context free languages (eg. $\cap_{\text{REG}}$); making the LRR lagnuages more interesting, from a theoretical point of view, then many of the other more ad hoc extensions we will discuss.

Culik and Cohen conjectured that the question of membership in LRR would be undecidable, unless the partition π was supplied as part of the problem. They did show that the problem was equivalent to determining a regular separating set for each of the constant number of rules (productions) of G. The question was settled (affirmatively) by Odgen (unpublished); a "similar" proof is contained in Szymanski and Williams [14], which directly proves that regular separability is not r.e. (implied by the proof that membership in LRR is not r.e.).

The major drawback of LR-Regular grammars is that this problem forces the user to supply the regular partition (ie. Finite State Machines to recognize the regular lookahead sets) with the grammar; no easy task, since their very existence is not decidable. Even in the "obviously" possible cases, the computation would be quite difficult, and would no doubt preclude its use. The correctness of their parser generator algorithm shows decidability when given the FSM's.

Baker [2] lists a second criticizm of LRR grammars — that the right to left scanning process is not viable on current hardware, which is targeted directly at left to right reading of files. However, this is probably of minor significance, given that practical inputs can be easily manipulated in memory.

Culik and Cohen suggest various extensions to their work, which contribute to the Extended LR (XLR) method of Baker [2] and LAR(m) method of Bermudez and Schrimpf [3]; both of which are "unbounded lookahead" parsing methods (the automaton postpones parsing for an arbitrary lookahead in the input to resolve a conflict). These are in order to overcome the decidability problem of the regular partitions. In particular, they suggest to "forget" all but a bounded amount of the pushdown store while scanning the LR(0) automaton, and construct *regular approximations* or *envelopes* to the known context free lookahead. Through successive refinement, they conjecture that such envelopes can eliminate the nondeterminism in the LR(0) machine.

5 Bounded Context Parsing

Floyd's notion of Bounded Context [7] provided much of the framework for the discovery of the LR(k) languages. A grammar G is said to be of *bounded context order* (m, n), BC(m,n), if every phrase of a sentential form is uniquely distinguished by the m (terminal) symbols to its left, and the n (terminal) symbols to its right.

If the restriction is weakened to specify that at least the leftmost phrase has bounded context, then the class, BRC(m,n), of *bounded right context grammars* is a proper subclass of the $LR(n)$ grammars, as LR(n) grammars have right context at most n, and unbounded left context. However, classes are equivalent when viewed in language space, ie. generative power: Knuth [10] showed that $\mathcal{L}_{\mathrm{LR(k)}} = \mathcal{L}_{\mathrm{BC(m,n)}} = \mathcal{L}_{\mathrm{DCFL}}$.

BRC parsing is canonical, in the LR sense, as only leftmost phrases are reduced. The obvious noncanonical extension, provided by Williams [16] is the class of *Bounded Context Parsable* grammars, BCP(m,n) — grammars G such that *some* phrase of a sentential form is uniquely distinguished by the m (vocabulary) symbols to its left, and the n (vocabulary) symbols to its right. BCP(m,n) is currently the most powerful of all classes of grammars/languages parsable in linear time, and for which membership in the class is decidable.

We wish to show the position of $\mathcal{L}_{\mathrm{BCP(m,n)}}$ in the containment hierarchy: As in the LRR case, the parsing automaton is deterministic, by definition, so BCP languages are unambiguous. As well, no BCP(m,n) grammar can describe EPAL, so $\mathcal{L}_{\mathrm{BCP(m,n)}} \subset \mathcal{L}_{\mathrm{UCFG}}$.

Szymanski and Williams [14] have shown the containment of the LRR languages within the BCP(m,n) languages. This is intuitive, because the lookahead DFAs for the LR-regular grammar have a constant number of configurations, so these can be incorporated into the vocabulary of some BCP grammar which accepts the language of the LRR grammar. For proper containment, we have the language

$$L_2 = \{a^n b^n c^m d^{m+l} \quad | \quad l, m, n \geq 1\} \cup \{a^n b^{2n} c^m d^m \quad | \quad m, n \geq 1\}$$

which cannot be LRR, as regular languages cannot "count" — the lookahead for the first handle must test membership in $\{c^m d^m\}$, which is non-regular. However the BCP(1,1) grammar G_2 (below) recognizes L_2, so L_2 is BCP(m,n). The language class hierarchy is shown in Figure 2.

The grammar G_1 of the previous section is not BCP(m,n) for any m and n, as the lookahead string is of unbounded length. Thus, the LR-Regular grammars are not contained in the BCP(m,n) grammars. The following BCP(1,1) grammar (taken from [14]), G_2, is not LRR, so the classes are incomparable. Similarly, G_2 is not LR(k), and it is an easy matter to find some LR(k) grammar which requires unbounded left context to parse, so LR(k) and BCP(m,n) are incomparable. The lattice of containment is shown in Figure 2. Note that noncanonical parsing is appropriate only for grammars which do not generate the empty word, so the poset is applicable only for ε-free languages.

				Context
G_2 :	S	$\rightarrow$	CD	$(\$,\$)$
	S	$\rightarrow$	$C'D'$	$(\$,\$)$
	C	$\rightarrow$	aCb	$(\varepsilon,\varepsilon)$
	C	$\rightarrow$	aB	$(\varepsilon,\varepsilon)$
	C'	$\rightarrow$	$aC'B'B'$	$(\varepsilon,\varepsilon)$
	C'	$\rightarrow$	$aB'B'$	$(\varepsilon,\varepsilon)$
	D	$\rightarrow$	Dd	$(\varepsilon,\varepsilon)$
	D	$\rightarrow$	Ad	(b,ε)
	D'	$\rightarrow$	A	$(\varepsilon,\$)$
	A	$\rightarrow$	cAd	$(\varepsilon,\varepsilon)$
	A	$\rightarrow$	cd	$(\varepsilon,\varepsilon)$
	B	$\rightarrow$	b	$(\varepsilon,\{B,D\})$
	B'	$\rightarrow$	b	$(\varepsilon,\{B',D'\})$

The above grammar works by first consuming/reducing all of the $c^i d^i$ substring into the nonterminal A, which is identifiable without context. When no c's are left, the operation decides based upon one symbol of context on either side (\$ or b) whether the number of c's and d's are the same, and decides deterministically which half of the grammar (D or D') is applicable, using the disjoint lookahead sets $\{B,D\}$ and $\{B',D'\}$.

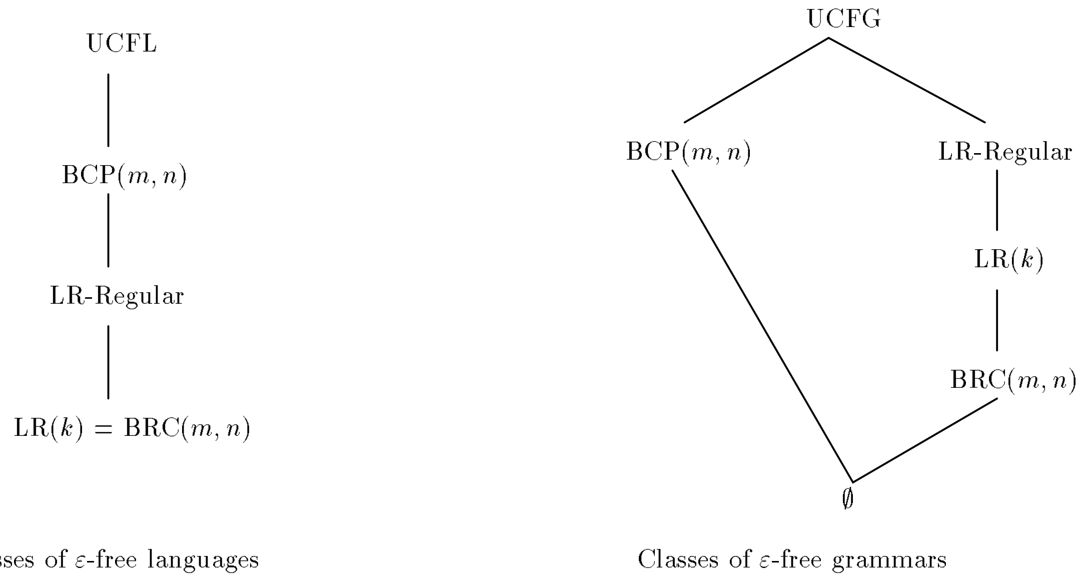


Figure 2: Containment of LR-Regular (LRR) and BCP classes

For given m and n, it is decidable whether a grammar G is BCP(m,n). However, it is not possible to determine whether G is in the union over the class; ie. the predicate

$$\text{“}\exists m,n \mid G \text{ is BCP}(m,n)\text{”}$$

is undecidable [16, 10].

The major problem with Bounded Context Parsable grammars has already been stated in the introduction. Although membership is decidable for given m and n, the parser requires creation of a very large table, representing the contexts possible in any sentential form. Although this table is of constant size (proportional to grammar, not input), it is very large, for even small m and n, so the parsing method is not applicable to any feasible compiler-generator. It is interesting to note that this is the same problem which plagued LR(1) parsing, but was solved by DeRemer's SLR and LALR methods.

6 Noncanonical Extensions to LR Parsing

Knuth [10] defines the first noncanonical extension to LR parsing as a sideline of its introduction: A grammar G is LR(k,t), if one of the first t leftmost phrases of any sentential form derivable from G can be (deterministically) reduced with its left context and only k characters of right context (lookahead).

The class of LR(k,t) grammars obviously includes the LR(k) grammars (ie. LR$(k,1)$) as a subset. To show that the containment is proper, consider the following grammar:

$$
\begin{array}{rlcl}
G_3: & S & \rightarrow & A\bar{A} \mid B\bar{B} \\
 & A & \rightarrow & a \\
 & B & \rightarrow & a \\
 & \bar{A} & \rightarrow & b\bar{A} \mid b\bar{B} \\
 & \bar{B} & \rightarrow & b\bar{B}c \mid bc
\end{array}
$$

$$L(G_3) = \{ab^n b^m c^m; m \geq 1, n \geq 0\}$$

In order to reduce the leftmost phrase, which is one of $A \rightarrow a$ or $B \rightarrow b$, we need to know whether there are more b's than c's, which requires unbounded lookahead. Thus G_3 is not LR(k) for any k. It is also not LR-Regular, as the lookahead requires memory to count b's. G_3 is, however, LR(1,2), as the second leftmost phrase must reduce to $\bar{B}$, which is deterministic without any lookahead. Given $\bar{A}$ or $\bar{B}$ as already reduced, the machine can decide whether to reduce the initial a to A or B with this single *noncanonical* lookahead.

We note that the language of G_3 is not inherently nondeterministic. A deterministic pushdown automata could easily recognize it, so is is also described by some LR(k) grammar.

The lookahead for an LR(k,t) grammar consists of both nonterminal and terminal symbols. The LR(k,t) method solves the reduce-reduce conflict on the first phrase by postponing it until the required (nonterminal) lookahead is available; here it postpones only once. The state resolving the nondeterminism looks like that of Figure 3.

$$
\boxed{\begin{array}{l}
[B \rightarrow a\bullet,\ \ \bar{B}] \\
[A \rightarrow a\bullet,\ \ \bar{A}] \\
\
\end{array}}
$$

Figure 3: Noncanonical lookahead removes nondeterminism.

Decidability of parser construction follows from the LR method, as the changes made can increase the size of the parser by no more than a constant size (because it is limited to t expansions). The additions are that the lookahead strings can have nonterminals as well as terminals, and the closure of an item includes any items generated by expanding the leading nonterminal in the lookahead string of the postponed items.

Szymanski and Williams [14] point out, however, that because of this constant bound, any LR(k,t) language can be parsed in linear time by a DPDA. Since the number of 'backups' and 'lookaheads' made during the parse is constant they can be kept in finite control, and hence performed by a single stack machine. Thus the LR(k,t) grammars have greater expressive power than the LR(k) grammars, but the same generative power (the DCFL's).

We have previously shown that G_3 is LR(k,t), but not LRR. In fact, the two grammatical classes are incomparable; since the following grammar, G_4, is LRR under lookahead partition $\pi = \{T^*a, T^*b\}$ (ie. we only need to know whether the string ends in a or b, and don't care what comes first – obviously regular), but not LR(k,t) for any k and t:

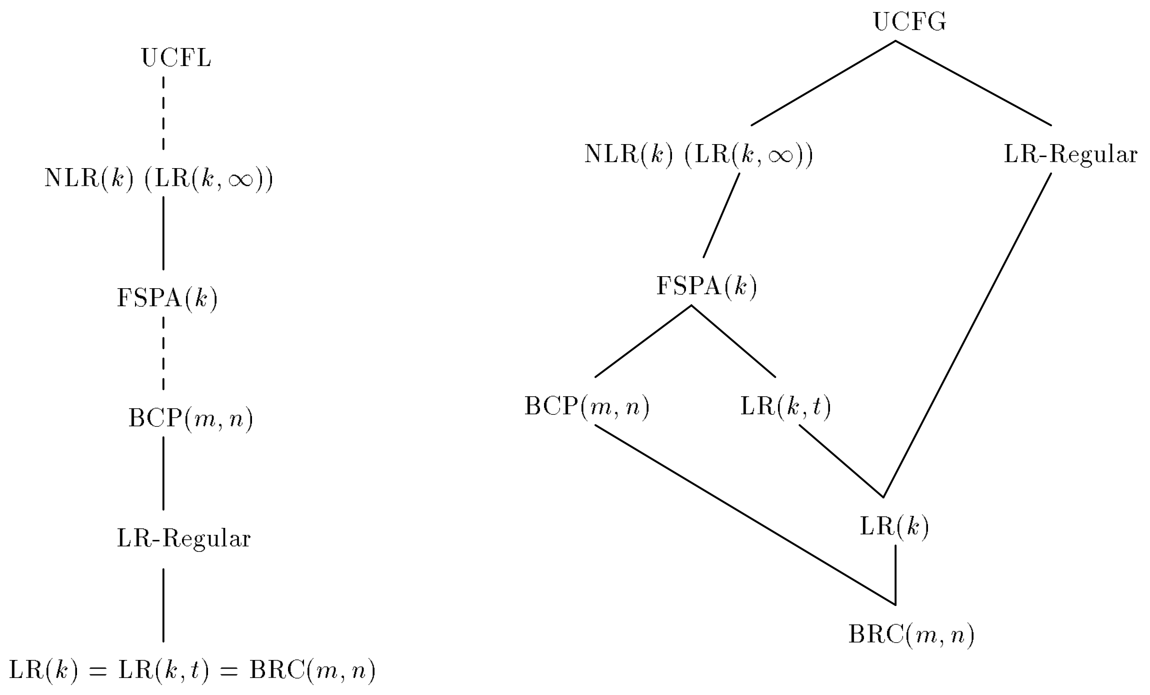
$$\begin{array}{llll} G_4: & S & \rightarrow & aAa \ \mid \ bAb \ \mid \ aBb \ \mid \ bBa \\ & A & \rightarrow & \bar{A}A \ \mid \ c \\ & B & \rightarrow & \bar{B}B \ \mid \ c \\ & \bar{A} & \rightarrow & c \\ & \bar{B} & \rightarrow & c \end{array}$$

$$L(G_4) = \{ac^+a\} \cup \{bc^+b\} \cup \{ac^+b\} \cup \{bc^+a\}$$

G_4 is not LR(k,t), because an unbounded number of complete phrases ($\bar{A}$'s or $\bar{B}$'s) must be scanned before they can be resolved by the final lookahead symbol. We could get the same result by asserting that $L(G_4)$ is non-DCFL (non-LR(k)), using the previous result.

Furthermore, G_3 is not BCP(m,n) for any m and n, as counting of an unbounded number of symbols is required on either side of the "first" reducable phrase, and G_2 of Section 5 is not LR(k,t) an unbounded number of B' or B handles occur before the reduce-reduce conflict can be resolved.

These relationships are added to our previous hierarchy, and shown in Figure 4.



Classes of ε-free languages Classes of ε-free grammars

Figure 4: Hierarchy with the addition of NLR(k) and LR(k,t) classes.

The obvious extension is to remove the constant t from the picture, allowing the parser to be fully noncanonical, which we now address.

6.1 NLR(k) Parsing

Definition 6.1 (Szymanski, Williams) *An unambiguous CFG in which every sentential form has some phrase which can be uniquely distinguished by its left context, and the first k characters of its right context* *is defined to be an* **LR(k,∞)**, *or* **NLR(k)**[3] *grammar.*

The NLR(k) class properly contains the LR(k,t) grammars, and it should also be clear that any BCP(m,n) grammar is necesarily $NLR(n)$. For proper containment,we have the following, which is NLR(k), but not BCP(m,n) for any m and n, as all b's and c's must be seen to make the first reduction at the a,b–interface (ie. unbounded context on either side is required):

$$\begin{array}{llcl} G_5: & S & \rightarrow & A \mid B \\ & A & \rightarrow & aA\bar{A} \mid a\bar{A} \\ & B & \rightarrow & aB\bar{B} \mid a\bar{B} \\ & \bar{A} & \rightarrow & b \\ & \bar{B} & \rightarrow & bb \end{array}$$

$$L(G_5) = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

G_5 is also not LR(k,t) as it cannot be accepted by a DPDA.

By definition, the NLR(k) grammars are unambiguous, and the following unambiguous grammar for EPAL is non NLR(k), so thir containment in UCFG is proper.

$$G_6: \quad S \quad \rightarrow \quad aSa \mid bSb \mid aa \mid bb$$

Since G_2 is BCP(1,1), it is also NLR(k), but as previously mentioned, it is non LRR. We already know the G_1 on page 8 is LRR. G_1 is not NLR(k) because the first phrase to be reduced must be either $C \rightarrow b$ or $D \rightarrow b$, and neither is identifiable with a constant sized lookahead string. Thus the grammar classes LRR and NLR(k) are incomparable.

As mentioned previously, we have a serious problem with the usability of the NLR(k) grammars, in the form of the following theorem.

Theorem 6.2 (Szymanski and Williams) *For given $k > 0$ the class of NLR(k) grammars is not recursively enumerable.*

Proof: We reduce membership in DCFG-EMPTY-INTERSECTION, a well known uncomputable problem [9], to the problem at hand. This shows that computation of membership in NRL(k) implies an algorithm to enumerate

$$C = \{(G_1, G_2) \mid L(G_1) \cap L(G_2) = \emptyset; \quad G_1, G_2 \in DCFG\},$$

which is not recursively enumerable.

Let $G_1 = (N_1, \Sigma, P_1, S_1)$ and $G_2 = (N_2, \Sigma, P_2, S_2)$ be arbitrary LR(k) DCFG's, with $N_1 \cap N_2 = \emptyset$ (or else just rename the violating nonterminals).

For every terminal a_i in Σ: Replace all occurances of a_i in P_1 by A_i, and all occurances of a_i in P_2 by B_i, where A_i and B_i are new nonterminals. Also add new productions $A_i \rightarrow a_i$ to P_1, and $B_i \rightarrow a_i$ to P_2. Call the resulting new grammars $G_1' = (N_1', \Sigma, P_1', S_1)$ and $G_2' = (N_2', \Sigma, P_2', S_2)$.

Define a new grammar,

$$G = (N_1' \cup N_2' \cup \{S\}, \Sigma, P_1' \cup P_2' \cup \{S \rightarrow S_1' \mid S_2\}, S).$$

Now G_1 and G_2 are unambiguous, as they are DCFG, so G is unambiguous iff $L(G_1) \cap L(G_2) = \emptyset$. We can also relate this to membership in NLR(k) with the following claim:

Claim 6.3 *G is unambiguous iff $G \in NLR(k)$.*

[3]The new term is ours, for compatibility with NSLR and NLALR to be discussed shortly.

Proof: The only if holds by definition, as all NLR(k) grammars are unambiguous.

For the if part, assume the G is unambiguous; then $L(G_1 \cap L(G_2) = \emptyset$ as just mentioned. In any word $x = x'a_i$, the last terminal can be reduced by either $A_i \rightarrow a_i$ or $B_i \rightarrow a_i$ using the left context only, since L(G_1) and L(G_2) are disjoint. So, $x = ya_i$ is reduced to $yA_i = za_jA_i$. With the A_i for lookahead, the previous terminal a_j can be reduced, and so on until the entire string has been reduced to a string in either $\{A_i\}^+$ or $\{B_i\}^+$. Now the parser for one of the LR(k) grammars G_1 or G_2 can deterministically reduce the sentence to S_1 or S_2. ∎

We have that $(G_1, G_2) \in C$ iff $G \in NLR(k)$. Thus the existance of an algorithm to compute membership in NLR(k) implies a method to enumerate C which is not recursively enumerable. It follows that $\{G \mid G \in NLR(k)\}$ is not r.e. ∎

Szymanski and Williams suggest a restriction of the NLR(k) grammars, to achieve decidability, which they call the FSPA(k) class. A grammar is FSPA(k) if it is (noncanonically) parsable by a "regular reduction pattern" which is k bounded to the right.

We will not define reduction patterns here. Informally a grammar is FSPA(k) if, for any sentential form, a reduction can be made given k characters of context on the right, and some regular set "remembering" the left context. Hence the machine is allowed to forget all but a regular representation of the (possibly non-regular) left context.

Unfortunately, the membership problem for these grammars is also undecidable (even for fixed k), due to the same problem suffered by the LR-Regular grammars (regular separation). The set is, however, recursively enumerable.

The FSPA(k) class can be shown to properly contain both the BCP(m, n) and $LR(k, t)$ classes, and to be properly contained in NLR(k). The language class it generates contains (not known to be proper) BCP(m, n) and is properly contained by NLR(k). The reader is referred to the proofs in [14] for details. The containment of the NLR(k) and FSPA(k) classes is shown in Figure 4 on page 12.

6.2 NSLR(k) Parsing

The NSLR method is a restriction of the NLR parsing just defined, analagous to the SLR restriction on the canonical LR grammars. A grammar is NSLR(k) if it can be parsed by a LR(0) machine with noncanonical SLR(k) lookahead.

We present a modified NSLR(1) construction algorithm due to Salomon and Cormack [13], which incorporates several fixes or enhancements to Tai's original algorithm [15]. We then proceed to anlayze the expressive power of the NLSR(k) grammars relative to the classes mentioned thus far in the paper.

6.2.1 NSLR(1) construction

To construct the NSLR(1) machine, we precompute the following sets over $V(G)$: Recall our notational conventions: $X, Y \in V$, $x \in T^*$, $\alpha, \beta \in V^*$.

$$\begin{aligned}
VISIBLE &= \left\{X \mid X \overset{*}{\Rightarrow} x \mid x \neq \varepsilon\right\} \\
FIRST(X) &= \left\{Y \mid X \overset{*}{\Rightarrow} Y\alpha\right\} \\
LAST(X) &= \left\{Y \mid X \overset{*}{\Rightarrow} \alpha Y\right\} \\
FOLLOW(A) &= \left\{X \mid B \Rightarrow \alpha Y \beta Z \gamma, A \in LAST(Y), \beta \overset{*}{\Rightarrow} \varepsilon, X \in FIRST(Z)\right\} \\
NEEDED\_FOLLOW(A) &= \left\{X \mid B \Rightarrow \alpha Y \beta X \gamma, A \in LAST(Y), \beta \overset{*}{\Rightarrow} \varepsilon\right\} \\
UNRESOLVABLE(A) &= \left\{X \in VISIBLE \mid A \Rightarrow \alpha X \beta, \alpha \overset{+}{\Rightarrow} \varepsilon\right\}
\end{aligned}$$

VISIBLE represents the set of nonterminals that can generate a non-empty word, and FIRST and LAST are simply the noncanonical versions of $FIRST_1$ and $LAST_1$ discussed previously. The NEEDED_FOLLOW of a nonterminal A is the set of vocabulary symbols which immediately follow A; in the simplest derivation, which differs from FOLLOW(A) which is a closure of NEEDED_FOLLOW – the set of FIRST symbols in the NEEDED_FOLLOW of A. The list of UNRESOLVABLE lookahead nonterminals for a given nonterminal arises from the method of Cormack and Salomon for handline invisible nonterminals in conflict resolution. Its purpose is to reject grammars that require state expansion to occur recursively on some state. This slightly weakens the class, but the lost grammars are deemed to be few and "not important enought to justify the additional complication" to handle them in the algorithm.

The construction algorithm is as follows:

```
Construct the CFSM.
Add noncanonical lookheads LA($A$) = FOLLOW($A$) $\cap$ VISIBLE
                    to complete items in states of the CFSM.
For each state $q$ containing a conflict on symbol $X$ do
    For each conflicting item $[A \rightarrow \alpha \bullet, LA_i]$ do
        if $X \in$ UNRESOLVABLE($A$)
                    or $X \in$ NEEDED_FOLLOW($B$) for some $B \in LA_i$
            ABORT — G is not NSLR(1).
        For each rule $B \rightarrow X\beta$, $B \in LA_i$ do
            add $[B \rightarrow \bullet X\beta]$ to $q$
            remove $X$ from $LA_i$
Stop.
```

Ignoring the problem of ε productions and invisible nonterminals, we are simply including nonterminals in the lookahead sets, and whenever a symbol X is conflicting in state q, we eliminate it by adding $[B \rightarrow \bullet X\beta]$ where B is a nonterminal lookahead symbol for the given item. We choose the "lowest" B in the $\triangleright$-graph, to resolve the conflict with the minimum possible noncanonical operation (ie. don't expand all nonterminals, just the ones which shift out the lookahead character). An example of the NSLR fix to a conflicting lookahead symbol is shown in Figure 5. The first state is inadequate in the SLR(1) machine for the non-SLR(1), non LR(k) grammar G_7. The second state shows the noncanonical NSLR(1) expansion of the state, which resolves the nondeterminism.

$$\begin{array}{llll} G_7: & S' & \rightarrow & S \\ & S & \rightarrow & Aa \mid Bb \\ & A & \rightarrow & \bar{A}A \mid \bar{A} \\ & B & \rightarrow & \bar{B}B \mid \bar{B} \\ & \bar{A} & \rightarrow & c \\ & \bar{B} & \rightarrow & c \end{array}$$

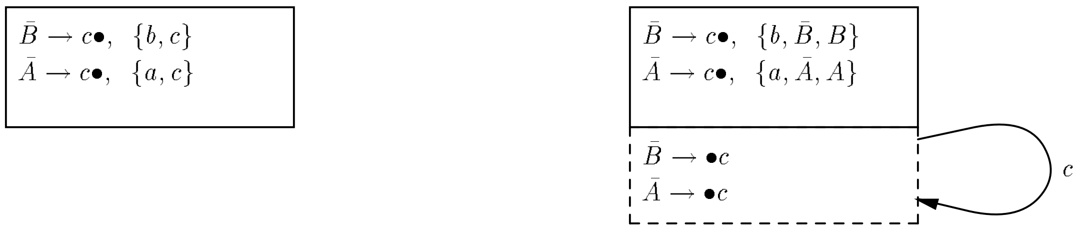
$$L(G_7) = \{c^n a\} \cup \{c^n b\}$$

6.2.2 Expressive power of NSLR(k) grammars

Tai has shown that NSLR(k) properly contains the LR(k) grammars.

It should be clear that any NSLR(k) grammar is necessarily NLR(k). For proper containment, consider the grammar G_4 on page 12, which is NLR(1), but non-NSLR(k). Figure 6 shows a part of the SLR(1) machine; the conflict in state q is a result of the CFSM, not an inadequacy in the lookahead, so the problem cannot be repaired by additional canonical or noncanonical lookahead.

Lemma 6.4 *The NSLR(k) and LR-Regular classes are incomparable.*



$\bar{B} \to c\bullet, \ \{b, c\}$
$\bar{A} \to c\bullet, \ \{a, c\}$
$\bar{B} \to c\bullet, \ \{b, \bar{B}, B\}$
$\bar{A} \to c\bullet, \ \{a, \bar{A}, A\}$
$\bar{B} \to \bullet c$
$\bar{A} \to \bullet c$
c

SLR(1) inadequate state for G_7 NSLR(1) fix

Figure 5: NSLR(1) fix to inadequate SLR(1) machine.

Proof: Recall that G_1 on page 8 is LR-Regular, but not NLR(k) for any k and hence not NSLR(k) for any k. As well, the grammar G_3 on page 11 is not LR-Regular, but is NSLR(1), as evidenced by the NSLR(1) machine shown in Figure 7. It follows that NSLR(k) and LRR are incomparable. ■

Recall that the NLR(k) class properly contains the BCP(m, n) class. We conjecture that this does not hold for NSLR(k), due to the loss of left context from using only the LR(0) states, and that BCP(m, n) and NSLR(k) are incomparable. One direction is shown by the grammar G_3, which is NSLR(1) (as just shown) but not BCP(m, n) (see page 11).

The relationship between NSLR and FSPA classes is not known, nor is any intuitive information available.

With regard to language inclusions, it is clear that NLR(k) properly contains NSLR(k), from the grammar containiment just shown. The relationship between $\mathcal{L}_{\text{NSLR}(k)}$ and the $\mathcal{L}_{\text{LRR}}$ or $\mathcal{L}_{\text{BCP}(m,n)}$ languages is unknown.

The hierarchy with the known information filled in is shown in Figure 8.

6.3 NLALR(k) Parsing

The NLALR(k) class is the obvious extension to the previous work on the NLR(k) and NSLR(k) classes. The containments of the grammar and language classes are obvious, but we are unable to produce a proof that some NLR(k) grammar is non-NLALR(k) or that some NLALR(k) grammar is non-NSLR(k), mostly due to the effort required to check candidates by hand.

Unfortunately, it may not be possible to construct NLALR(k) parsers in general, since membership in the class is conjectured to be non-r.e. as we show in the next section (6.3.1). However, the LR(k, t) method of Knuth can be used to provide greater expressive power to LALR gramamrs with t-bounded noncanonical operation, which we describe in Section 6.3.2.

6.3.1 Decidability

We believe that membership in NLALR(k) is not computable. For a formal proof, it would be necessary to give a method to construct a gramamar G' for any input G which is NLALR(k) iff G is NLR(k). However, no such reduction is known. Instead we provide an intuitive proof only, in the form of the following (weaker) claim and its justification.

Conjecture 6.5 *The class of NLALR(k) grammars is not decidable.*

The justification for this conjecture is as follows: Assume that membership in NLALR(k) is decidable. It should be clear that an algorithm for the construction of a NLALR(k) parser then exists.

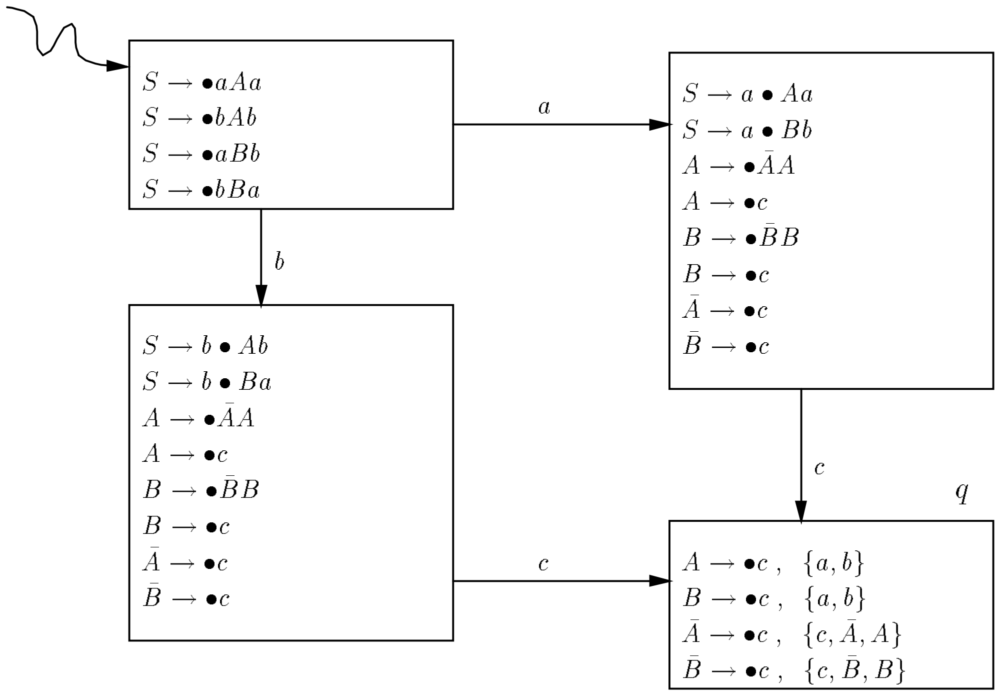


Figure 6: G_4 is not NSLR(k) for any k.

Recall that the NLR(k) class was non r.e. as a result of the undetectable ambiguity possible in the parser, rather than simply the inability to resolve a conflict with lookahead. However, given that we can construct an inadequate NLALR(k) parser for G, we can proceed to construct a NLR(k) parser for G by state splitting in a method similar to in Pager's algorithm for LR(k) parser construction (see [8]). Since the state splitting algorithm "should" work for the noncanonical case as well as for canonical lookahead, we would then have an algorithm to test membership in NLR(k).

The crux of the conjecture lies in the belief that if G is NLR(k), then it's inadequate NLALR(k) parser would be sufficient to test for NLR(k)-ness.

6.3.2 LALR(k, t) Parsing

The decidability of construction of an LALR(k, t) parser follows from the decidability of the LR(k, t) parser of Knuth. We will outline the construction of an LALR($1, t$) parser:

We assume the noncanonical extensions of the NSLR construction just outlined in Section 6.2.1. We also need the analagous noncanonical concepts to the standard LALR(1) constrution, which simply consist of defining PATH and δ of Section 3.2 over V instead of T.

$$\begin{aligned}\delta(\{[A \to \alpha \bullet X\beta, \gamma]\}) &= \{[X \to \bullet\omega, FIRST_k(\beta\gamma)] \mid X \to \omega \in P\} \\ &\cup \{[X \to \bullet\omega, REST_k(\gamma)] \mid X \in FIRST(\gamma)\}\,,\end{aligned}$$

and

$$PATH_k(B, C) = \bigcup \{FIRST_k(\beta_n \cdots \beta_1) \mid B_0 = B, B_n = C\}\,.$$

The more notable change is that the $\rhd$ graph must be defined to take into account for state expansions:

$$B_1 \rhd B_2 \text{ iff } B_1 \to B_2\beta_2 \in P \text{ or } B_2 \in FIRST(B_1).$$

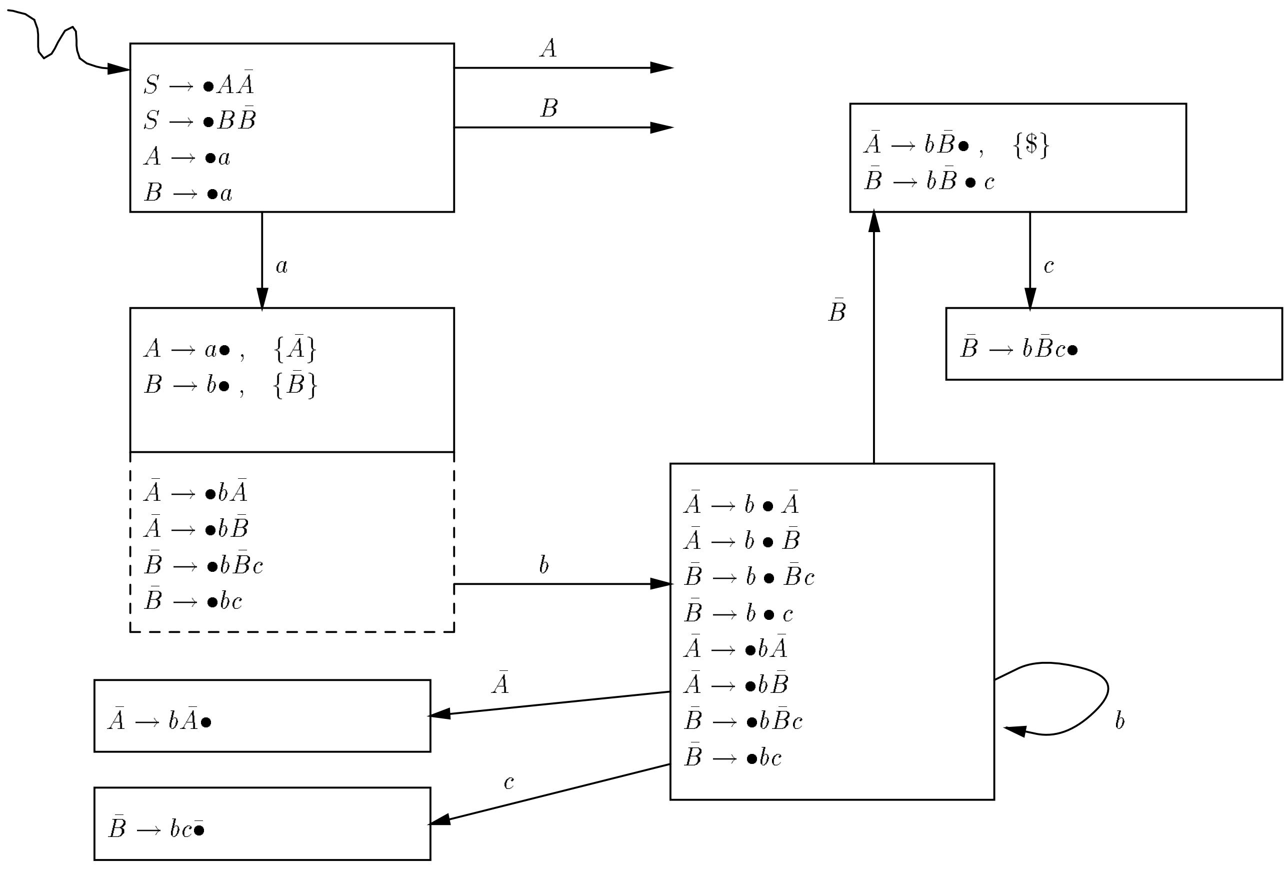

$S \to \bullet A\bar{A}$
$S \to \bullet B\bar{B}$
$A \to \bullet a$
$B \to \bullet a$
A
B
a
$A \to a\bullet\,,\quad \{\bar{A}\}$
$B \to b\bullet\,,\quad \{\bar{B}\}$
$\bar{A} \to \bullet b\bar{A}$
$\bar{A} \to \bullet b\bar{B}$
$\bar{B} \to \bullet b\bar{B}c$
$\bar{B} \to \bullet bc$
b
$\bar{A} \to b \bullet \bar{A}$
$\bar{A} \to b \bullet \bar{B}$
$\bar{B} \to b \bullet \bar{B}c$
$\bar{B} \to b \bullet c$
$\bar{A} \to \bullet b\bar{A}$
$\bar{A} \to \bullet b\bar{B}$
$\bar{B} \to \bullet b\bar{B}c$
$\bar{B} \to \bullet bc$
b
$\bar{A}$
$\bar{A} \to b\bar{A}\bullet$
c
$\bar{B} \to bc\bullet$
$\bar{B}$
$\bar{A} \to b\bar{B}\bullet\,,\quad \{\$\}$
$\bar{B} \to b\bar{B} \bullet c$
c
$\bar{B} \to b\bar{B}c\bullet$

Figure 7: G_3 is NSLR(1).

We now have the definition of noncanonical LALR(1) lookahead as:

$$\begin{aligned} LA\,(p, [A \to \alpha_1 \bullet \alpha_2]) \;=\; \{\, \gamma \;\mid\; & \gamma \in PATH_t(A', A) \\ & \quad \oplus_t\ FIRST_t(\beta_2) \\ & \quad \oplus_t\ LA_t\,(q, [B \to \beta_1 \bullet A'\beta_2])\,, \\ & q \in PRED(p, \alpha_1), \\ & A' \;\triangleright^*\, A, \\ & [B \to \beta_1 \bullet A'\beta_2] \in kernel(q)\ \}\,. \end{aligned}$$

This can be implemented efficiently in the same manner as the LALR algorithm of Section 3.2, and is "almost" correct. It should be pointed out that a great deal more complexity is added if we attempt to determine the LALR(k,t) lookahead with $k \neq 1$, as the required set-size is the product of k and t, and it would become important to know 'which' lookahead sets were being concatanated.

By "almost" correct above, we mean two things: The first is that we are expanding, unnecessarily, for all noncanonical lookahead. This is wasteful, because most states will not be in conflict. As seen with the LSLR grammars (predecessor to NSLR in Tai's paper [15]) this will also cause unnecessary new conflicts, and cause the grammatical class to be smaller. The second is that we have not described the removal of conflicts in the noncanonical expansion (since conflicting lookahead still exists – it was never deleted as in the NSLR algorithm).

We now give the additions required for the correct algorithm:

To construct the LALR($1,t$) automaton, first construct the LALR(t) automaton without expanding states (ie. ignoring the second component of δ). We now have, essentially a LALR(t) parser, where the lookahead

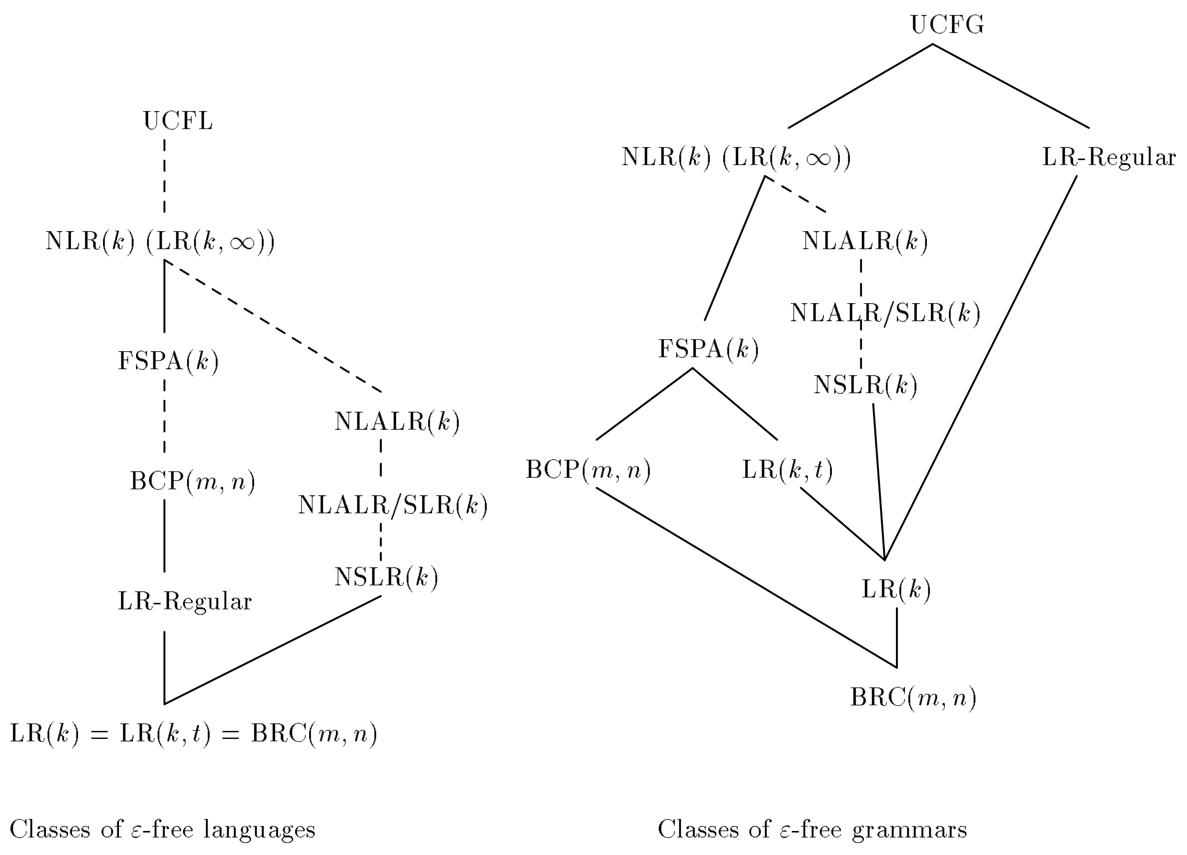


Figure 8: The known containment hierarchy.

string contains terminals or nonterminals, and may conflict. Define a conflict to be a state with a reduce-reduce or shift-reduce conflict using only the first terminal or first marked nonterminal character of lookahead (all nonterminals initially unmarked). For each inadequate state q, conflicting on lookahead symbol Y, and each item $[A \rightarrow \alpha\bullet, LA_i]$, expand as per the second part of δ only the closure items beginning with Y:

$$[X \rightarrow \bullet Y\beta, REST(\gamma) \text{ for all } \gamma \in LA_i | FIRST(\gamma) = X]\,;$$

remove the conflicting lookahead Y and mark the nonterminals X. Note that the expansion process "uses up" one character of extra lookahead at each step (from the "REST"), and no new characters are ever created, so it is finite (halting). If no lookahead is available, then the grammar is not LALR$(1, t)$, so quit.

The details of this construction are left out, but it should be clear that it is correct (from the correctness of Park,Choe and Chang's LALR algorithm, and Salomon's NSLR(1) algorithm) and halts (as just mentioned).

Recall that the LR(k, t) class is more expressive than the LR(k) class, but generates the same set. Obviously LALR(k, t) cannot then generate non-deterministic languages. However, it is possible to extend the definition to expand t levels of noncanonical LALR lookahead, and conclude with a recursive NSLR lookahead (which is decidable). This would give a class of gramamars at least as expressive and generative as the NSLR(k) set. This hybrid idea is not new; it is mentioned by Salomon and Cormack [13], but only for the LALR/SLR$(1, 1)$/NSLR(1) case.

7 Conclusions

Noncanonical versions of LR parsers prove to be effective in increasing both the class of grammars, and the class of languages supported by the accepted. However, in some cases, the power introduced by the extension may go so far as to render the set of grammars undecidable, and hence un-usable.

We have discussed the various forms of known noncanonical extensions to LR parsing, including the successful NSLR(k) method of Tai [15], and the analogous, but undecidable LR(k) and (believed undecidable) NLALR(k) extensions. We have shown the existance of a recursive set of grammars, the class NLALR(k, t) which provides expressive power above the LR(k) grammars, but generates no more than the deterministic languages. This class can be combined with the NSLR(k) class to yield a more expressive hybrid class, NLALR/SLR(k, t) which includes the NSLR(k) grammars as a subset, and admits linear time, viable parsers for its members. Although it is not "pleasing" that the user must specify such an arbitrary path bound on the action of the parser generator, it is, in fact, necessary for the correctness of the algorithm, and in practice it is likely that NLALR/SLR($1, 1$) or NLALR/SLR($1, 2$) will suffice.

Further research into noncanonical extensions of grammars is useful, not only for increasing the class of grammars, but for use in syntax error recovery and more convenient and natural "notations" for specification of context free grammars [13].

One area which we have not investigated fully is to determine a charactarization of the "problematic" grammars in the NLR(k) class. Identification of these could allow for a new class to be defined which contains the useful noncanonical properities, and decidable membership. Proof of some of the open class containments of Section 6 could possibly help with this characterization.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, “Compilers: Principles, Techniques, and Tools”. Addison Wesley, Reading, Mass. (1986).

[2] Theodore P. Baker, “Extending Lookahead for LR Parsers”. J.Comp.Sys.Sci 22 (1981), 243-259.

[3] Manuel E. Bermudez and Karl M. Schimpf, “A Practical Arbitrary Look-ahead LR Parsing Technique”. ACM SIGPLAN Notices (1986), 136-144.

[4] K. Culik and R. Cohen, “LR-regular grammars — an extension of LR(k) grammars”. J.Comp.Sys.Sci 7 (1973), 66-96.

[5] Frank DeRemer, “Simple LR(k) grammars”. CACM 14, 7 (1971), 453-460.

[6] Frank DeRemer and Thomas Pennello, “Efficient Computation of LALR(1) Look-Ahead Sets”. ACM TOPLAS, Vol.4, No.4 (1982), 615-649.

[7] R.W. Floyd, “Bounded context syntactic analysis”. CACM 7,2 (1964), 62-67.

[8] Charles N. Fischer and Richard J. LeBlanc Jr., “Crafting a Compiler”. Benjamin / Cummings, Menlo Park, California. (1988).

[9] John E. Hopcroft and Jeffrey D. Ullman, “Introduction to Automata, Lanugages and Computation”. Addison Wesley 1979.

[10] Donald Knuth, “On the translation of languages from left to right”. Inf. Control 8 (1965), 607-639.

[11] Joseph C.H. Park, K.M. Choe and C.H. Chang , “A New Analysis of LALR Formalisms”. ACM TOPLAS, Vol.7, No.1, (Jan. 1985), 159-175.

[12] Danial J. Salomon, “Metalanguage Enhancements and Parser-Generation Techniques for Scannerless Parsing of Programming Languages”. Ph.D. Thesis, University of Waterloo. Tech.Report CS-89-65 (1989).

[13] Danial J. Salomon and Gordon V. Cormack, “Scannerless NSLR(1) Parsing of Programming Languages”. ACM SIGPLAN Notices, (June 1989), 170-178.

[14] T.G. Syzmanski and J.H. Williams, “Noncanonical extensions of bottom up parsing techniques”. SIAM J.Computing 5, 2 (June 1976), 321-250.

[15] Kuo-Chung Tai, “Noncanonical SLR(1) Grammars”. ACM TOPLAS, Vol.1, No.2, (Oct. 1979), 295-320.

[16] J.H. Williams, “Bounded context parsable grammars”. Inf. Control 28,4 (1975), 314-334.