Generalised recursive descent
Part 2: Some underlying theory

Draft

Adrian Johnstone
A.Johnstone@rhbnc.ac.uk

Elizabeth Scott
E.Scott@rhbnc.ac.uk

Technical Report

CSD–TR–97–19

October 31, 1997

Royal Holloway
University of London

Department of Computer Science
Egham, Surrey TW20 0EX, England

1 Introduction

This report is the second in a two part series in which we consider a parsing technique which we call *generalised recursive descent* (GRD). In the first report [JS97] there is an extensive discussion of the influence of parsing techniques on language design, a full description of the GRD parsing technique, and a description of a tool, GRDP, which generates GRD parsers for suitable grammars.

There are essentially two classes of GRD parsers: the full *prototyping* versions which are admitted by any non-left recursive grammar, and the more efficient *production* versions which are only guaranteed to be admitted (see below) by non-left recursive follow-determined grammars. In this report we shall consider theoretical aspects of follow-determined grammars and the languages which can be specified with them.

- We shall show that any non-trivially left recursive grammar is follow-non-determined and that the only follow-determinism allowable in an LR(1) grammar is that which arises from left recursion. This gives some insight into the question of why certain grammars are not LR(1).
- We shall demonstrate the surprising result that the standard left recursion removal algorithm is not robust with respect to follow-determinism. That is, the left recursion removal algorithm can, and for some grammars must, introduce additional follow-non-determinism. This means that it is possible to apply the left recursion removal algorithm to an LR(1) grammar and end up with a grammar which is no longer LR(1). Since all LL(1) grammars are also LR(1) and the left recursion removal algorithm is often used as part of a process of constructing an LL(1) grammar, this is a serious problem.
- We shall show that there are non-left recursive follow-determined languages for which there is no LR(1) grammar. Thus the production version of GRDP can produce parsers for some languages which cannot be handled using the standard bottom-up techniques.
- We shall show that there is a non-left recursive follow-determined language for which there is no left factored grammar.

Although this report is essentially independent of the companion report [JS97], much of the motivation for the work in this report is contained there and we strongly recommend that the reader reads that material before looking in detail at the theoretical issues discussed below, and this report has been written with that recommendation in mind.

2 Follow determined grammars

It is well known that the LL(1) class of grammars admit parsers which use basic recursive descent techniques. The class of LL(1) grammars is quite restricted but it is possible to extend the basic recursive descent technique to allow backtracking. A common technique is to explore each alternate in a grammar rule, and to return either the first match encountered to the input string or the longest such match. However, with these techniques follow-determinism is still a problem. First match works if, with one token of lookahead, there is only one possible match, and this is guaranteed if the grammar is follow-determined. We begin by looking at LL(1) grammars, and then in more detail at follow-determinism.

2.1 Formal grammars

We use a slightly unconventional definition of a grammar rule. A grammar rule is a mapping from the set of non-terminals to sets of strings of terminals and non-terminals. In the examples in this paper the sets will be finite, so grammar rules can be thought of in the usual EBNF way,

$$A\texttt{::=}\{aBb, aCC, \epsilon\}$$

is equivalent to

$$A\texttt{::=}aBb \mid aCC \mid \epsilon.$$

However, much of what we say applies to grammars where the right hand sides of grammar rules can be infinite sets, and in a future paper we shall be discussing such grammars. Thus we use the set based notation here so that this work will not need to be reformulated later.

A *context free grammar* $\Gamma = (U, T, S, P)$ is a set U of symbols, a set $T \subset U$ of terminals, a start symbol $S \in U \backslash T$, and a set P of rules $A\texttt{::=}\tau_A$, where $\tau_A \subseteq U^*$ (the set of strings on U), $A \in U \backslash T$, and there is only one grammar rule for each A. The elements in τ_A are called *alternates* of the rule $A\texttt{::=}\tau_A$.

A *derivation step* is of the form

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

where $\alpha, \beta \in U^*$, $A \in U \backslash T$, and $\gamma \in \tau_A$.

A *derivation* is a sequence of derivation steps

$$\alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \alpha_k = \beta$$

where $\alpha_{i-1} \Rightarrow \alpha_i$ is a derivation step, $1 \leq i \leq k$. In this case we write $\alpha \stackrel{*}{\Rightarrow} \beta$, or $\alpha \stackrel{k}{\Rightarrow} \beta$ if we wish to make the number of steps explicit.

We define $L(\Gamma)$, the *language generated by* Γ, to be the strings of terminals which can be derived from S. So

$$L(\Gamma) = \{u \in T^* \mid S \stackrel{*}{\Rightarrow} u\}.$$

A parser P can be thought of as a map from the set of strings in a language to the set $\{success, failure\}$. The idea is that a string is input to P and after a finite amount of time P should terminate and return either *success* or *failure*. In other words, if u is the input string and $P(u)$ is the result of inputting u to P, we want $P(u) = success$ if u is in the language and $P(u) =$*failure* otherwise.

Formally, a grammar Γ *admits* a parser P if

- for all u in $L(\Gamma)$, $P(u) = success$,
- for all u not in $L(\Gamma)$, $P(u) =$*failure*.

2.1.1 Reachable symbols, useful strings and left recursion

We need a grammar to have certain properties if it is to admit a particular parsing technique. However, we do not need these properties to hold for parts of the grammar which will never appear in a derivation. For example, in a grammar whose productions are

S::=$\{a, Ab\}$
A::=$\{A, bA, bC\}$
B::=$\{Ba\}$
C::=$\{CC\}$

there is left recursion (see below). However, the production A::=A adds nothing to the language generated by the grammar, the production B::=Ba will never be used, and the productions A::=bC and C::=CC will never be used in the derivation of a sentence.

A symbol $x \in U$ is *reachable* if there is a derivation

$$S \stackrel{*}{\Rightarrow} \alpha x \beta,$$

for some $\alpha, \beta \in U^*$.

A string $\alpha \in U^*$ is *useful* if there is a derivation

$$S \stackrel{*}{\Rightarrow} \beta\alpha\gamma \stackrel{*}{\Rightarrow} a_1 \ldots a_q \in T^*.$$

We say that α is *useless* if it is not useful.

We note that standard recursive descent based parsers require the grammar to be free of left recursion, as left recursion can cause the parser to go in to an infinite loop.

A non-terminal A is *left recursive* if for some $\alpha \in U^*$, and some $m \geq 1$, $A \stackrel{m}{\Rightarrow} A\alpha$.

A grammar Γ is *left recursive* if for some reachable $A \in U$, A is left recursive.

GRD requires its input grammars to be non-left recursive. However, as there is a standard left recursion removal algorithm [ASU86] any context free language can be parsed using a prototype GRD parser. Unfortunately, as we shall see below, this algorithm can introduce follow-non-determinism, so currently left recursion is an unavoidable problem for production version GRD parsers.

2.1.2 FIRST sets and left factored grammars

In order for the parser to be more efficient it is useful to be able to tell, by looking at the current input symbol, which alternate of τ_A to use to replace A in the next derivation step. The idea is that if the current input symbol is a then we use an alternate α only if it is possible to derive a string beginning with a from α. Thus we are interested in the so-called FIRST sets of strings.

The FIRST set of a string $\alpha \in U^*$ is the set of terminals which can begin a string derivable from α, together with the empty string if $\alpha \stackrel{*}{\Rightarrow} \epsilon$. So

$$\mathrm{FIRST}_T(\alpha) = \{a \in T \mid \text{for some } \beta,\ \alpha \stackrel{*}{\Rightarrow} a\beta\}$$

and

$$\mathrm{FIRST}(\alpha) = \begin{cases} \mathrm{FIRST}_T(\alpha) \cup \{\epsilon\}, & \text{if } \alpha \stackrel{*}{\Rightarrow} \epsilon \\ \mathrm{FIRST}_T(\alpha), & \text{otherwise.} \end{cases}$$

If we are to be able to decide which alternate to use on the basis of the next input symbol, a say, then it has to be the case that at most one alternate in the grammar rule has a in its FIRST set. This motivates the following definition.

A grammar Γ is *left factored* if for every reachable non-terminal $A \in U$ and for every pair $\alpha, \beta \in \tau_A$ with $\alpha \neq \beta$, we have $\mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta) = \emptyset$.

This is the so-called lookahead property. Many of the recent attempts to extend recursive descent parsing techniques [Par95] have focused on relaxing this property, for example by allowing the parser to choose a production on the basis of the next k input symbols – LL(k) parsers. Our parser generator does not require the grammar to be left factored, or indeed to be left factored for any value of k. Of course, if the grammar is not left factored then the generated parser will, in general, be less efficient than an LL(1) parser. But a key point is that if the grammar *is* left factored then the parser produced will have LL(1) efficiency.

2.1.3 FOLLOW sets and follow-determined grammars

The other property required of an LL(1) grammar is that the parser should know 'when to stop' matching a production. This may seem a little strange, but essentially we mean that if the parser has matched a portion of the input to the right hand side of a production, it must be clear whether it should continue to try to get a longer match, or stop and begin matching the next production. This is the basis of a new concept which allows us to extend our understanding of the restrictions of current parsing techniques and associated language design. The idea is that the if next input symbol, a say, can appear in a sentential form (string derivable from the start symbol) after a non-terminal A then A cannot derive both u and uaw, so if we match u we know that we can stop because there is no possibility of also matching something of the form uaw. In order to formally define this notion we need the concept of a FOLLOW set.

The FOLLOW set of a non-terminal A is the set of terminals which can appear immediately after A in a sentential form, together with a special end of file symbol \$ if A can appear at the end of a sentential form. So

$$\mathrm{FOLLOW}_T(A) = \{a \in T \mid \text{for some } \alpha, \beta,\ S \stackrel{*}{\Rightarrow} \alpha A a \beta\}$$

and

$$\mathrm{FOLLOW}(A) = \begin{cases} \mathrm{FOLLOW}_T(A) \cup \{\$\}, & \text{if for some } \alpha,\ S \stackrel{*}{\Rightarrow} \alpha A \\ \mathrm{FOLLOW}_T(A), & \text{otherwise.} \end{cases}$$

A grammar Γ is *follow-determined* if for every $A \in U$, if $A \stackrel{*}{\Rightarrow} u$ and $A \stackrel{*}{\Rightarrow} uaw$, for some $a \in T$, $u, w \in T^*$, then $a \notin \mathrm{FOLLOW}(A)$.
(Note: if A is not reachable then its FOLLOW set is empty, so this condition is only of interest for reachable non-terminals.)

There are other parser generators which use a generalised form of recursive descent [BB95]. Such parsers usually explore all possibilities at each step in the derivation but then select only one to proceed with. A common selection procedure is to choose the so-called 'longest-match', i.e. the possibility which matches the longest section of the input string. This process is not guaranteed to produce correct parsers for all grammars. Prototype version GRD parsers return the set of all possible matches at each step and continue with each of them. This allows us to produce correct parsers for all non-left recursive grammars, and all possible derivations in the case where a grammar is ambiguous. The more efficient production version GRD parsers also select just one alternate to proceed with, being the first match with a allowable follow symbol. This modification provides similar parsing power to the traditional longest-match approach, and is guaranteed to produce a correct parser for a follow-determined grammar. A follow-determined grammar will also admit a longest-match based parser. However, a follow-determinism based parser is generally more efficient than longest-match one because we can abort an alternate as soon as we find a follow-determined match, whereas longest match requires all strings to be checked and compared for length.

2.2 LL(1) grammars and follow-determinism

Follow-determinism is not the condition usually used in the definition of LL(1) grammars, but it turns out that the weaker condition usually stated is equivalent to follow-determinism in the presence of left factoring.

Γ is *simply follow-determined* if for all A such that $A \stackrel{*}{\Rightarrow} \epsilon$, if $\beta \in \tau_A$ then $\mathrm{FIRST}(\beta) \cap \mathrm{FOLLOW}(A) = \emptyset$.

A grammar Γ is said to be LL(1) if

- $L(\Gamma) \neq \emptyset$.
- Γ is left factored.

◇ Γ is simply follow-determined.

Theorem 1 (below) shows that all LL(1) grammars are follow-determined. Production version GRD parsers require only that the input grammar be non-left recursive and follow-determined.

The following lemma is used in the proof of Theorem 1.

Lemma 1 *Let Γ be an LL(1) grammar. If $S\stackrel{*}{\Rightarrow}\alpha_1\alpha\alpha_2 \in U^*$ and if for some $u \in T^*$, $a \in T$ and $\gamma \in U^*$ we have*

$$\alpha\stackrel{*}{\Rightarrow}u \qquad \alpha\stackrel{*}{\Rightarrow}ua\gamma$$

then there exists a non-terminal A such that $\alpha = \alpha' A\omega$, where $\omega\stackrel{*}{\Rightarrow}\epsilon$ and $a \notin$ FOLLOW(A).

Proof The proof is by induction on the length of a left-most derivation $\alpha\underset{lm}{\stackrel{m}{\Rightarrow}}u$, of u from α.

If $m = 0$, so $\alpha = u$, then $\alpha\stackrel{*}{\Rightarrow}\beta$ if and only if $\beta = u$, so there is nothing to prove.

If $m = 1$, so $\alpha \Rightarrow u$, then $\alpha = u_1Bu_3$ where for some $u_2 \in \tau_B$, $u = u_1u_2u_3$. Also, since $\alpha\stackrel{*}{\Rightarrow}ua\gamma$, we have $\beta' \in \tau_B$ such that $\beta'\stackrel{*}{\Rightarrow}\delta$ where $u_1\delta u_3 = u_1u_2u_3a\gamma$. Since $u_2u_3 \stackrel{*}{\not\Rightarrow} u_2u_3a\gamma$ we have $u_2 \neq \beta'$ and so, since Γ is left factored,

$$\text{FIRST}(u_2) \cap \text{FIRST}(\beta') = \emptyset.$$

Considering the lengths of strings we see that we must have $\delta = u_2\eta$, where $\eta \neq \epsilon$, so we must have $u_2 = \epsilon$, $\beta' \stackrel{*}{\not\Rightarrow}\epsilon$ and $\beta' u_3\stackrel{*}{\Rightarrow}u_3a\gamma$. Now,

$$S\stackrel{*}{\Rightarrow}\alpha_1\alpha\alpha_2 = \alpha_1u_1Bu_3\alpha_2$$

so, unless $u_3 = \epsilon$,

$$\text{FIRST}(u_3) \subseteq \text{FOLLOW}(B) \cap \text{FIRST}(\beta') = \emptyset$$

since Γ is simply follow-determined. Thus we must have $u_3 = \epsilon$ and $\delta = a\gamma$. Then $\alpha = u_1B$, $B\stackrel{*}{\Rightarrow}a\gamma$, and Γ is simply follow-determined, so we must have $a \notin$ FOLLOW(B), as required.

Now suppose that the result is true for left most derivations of length less than $m \geq 2$. Then we have $\alpha = u_1B\eta$, $u_1 \in T^*$, and $\beta, \beta' \in \tau_B$ such that

$$\alpha = u_1B\eta \Rightarrow u_1\beta\eta\underset{lm}{\stackrel{m-1}{\Rightarrow}}u_1u_2u_3 = u$$

$$\alpha = u_1B\eta \Rightarrow u_1\beta'\eta\underset{lm}{\stackrel{*}{\Rightarrow}}u_1u_2u_3a\gamma, \quad \text{where } \beta\stackrel{*}{\Rightarrow}u_2, \eta\stackrel{*}{\Rightarrow}u_3.$$

We show that we must have $\beta = \beta'$, and that in this case the result follows by induction.

Suppose that $\beta \neq \beta'$, then, since Γ is left factored, we must have

$$\text{FIRST}(\beta) \cap \text{FIRST}(\beta') = \emptyset.$$

If $\beta, \beta' \overset{*}{\Rightarrow} \epsilon$ then $\epsilon \in \text{FIRST}(\beta) \cap \text{FIRST}(\beta')$. If neither of β, β' derive ϵ then $\text{FIRST}(u_2) \subseteq \text{FIRST}(\beta) \cap \text{FIRST}(\beta')$. So the only possibility is that exactly one of β, β' derives ϵ.

If $\beta' \overset{*}{\Rightarrow} \epsilon$ then β does not, so $u_2 \neq \epsilon$ and $\text{FIRST}(u_2) \subseteq \text{FIRST}(\beta)$. Thus $\text{FIRST}(u_2) \cap \text{FIRST}(\beta') = \emptyset$ and so $\eta \overset{*}{\Rightarrow} u_2 u_3 a \gamma$. But then

$$S \overset{*}{\Rightarrow} \alpha_1 \alpha \alpha_2 = \alpha_1 u_1 B \eta \alpha_2 \overset{*}{\Rightarrow} \alpha_1 u_1 B u_2 u_3 a \gamma \alpha_2$$

and $\text{FIRST}(u_2) \subseteq \text{FOLLOW}(B)$, contrary to the simple follow-determinism of Γ.

If $\beta \overset{*}{\Rightarrow} \epsilon$ then β' does not, so $\text{FIRST}(u_2 u_3 a \gamma) \subseteq \text{FIRST}(\beta')$. If $u_2 u_3 \neq \epsilon$ then

$$\text{FIRST}(u_2 u_3) \cap \text{FIRST}(\beta) = \emptyset$$

so $\eta \overset{*}{\Rightarrow} u_2 u_3$,

$$S \overset{*}{\Rightarrow} \alpha_1 \alpha \alpha_2 = \alpha_1 u_1 B \eta \alpha_2 \overset{*}{\Rightarrow} \alpha_1 u_1 B u_2 u_3 a \gamma \alpha_2$$

and $\text{FIRST}(u_2 u_3) \in \text{FOLLOW}(B)$, contrary to the simple follow determinism of Γ. Thus the only possibility is $u_2 = u_3 = \epsilon$ and $a \in \text{FIRST}(\beta')$, then $\alpha = u_1 B \eta$, $\eta \overset{*}{\Rightarrow} u_3 = \epsilon$, and $a \notin \text{FOLLOW}(B)$, since Γ is simply follow-determined.

Thus we may suppose that $\beta = \beta'$, so

$$\alpha = u_1 B \eta \Rightarrow u_1 \beta \eta \underset{lm}{\overset{m-1}{\Rightarrow}} u_1 u_2 u_3 = u, \qquad \text{and } \beta \eta \underset{lm}{\overset{m-1}{\Rightarrow}} u_2 u_3$$

$$\alpha = u_1 B \eta \Rightarrow u_1 \beta \eta \underset{lm}{\overset{m-1}{\Rightarrow}} u_1 u_2 u_3 a \gamma, \qquad \text{and } \beta \eta \underset{lm}{\overset{m-1}{\Rightarrow}} u_2 u_3 a \gamma.$$

Then, by induction, $\beta \eta = \mu A \omega'$ where $\omega' \overset{*}{\Rightarrow} \epsilon$ and $a \notin \text{FOLLOW}(A)$. If $\eta = \mu' A \omega'$ then $\alpha = u_1 B \mu' A \omega'$, as required. If $\beta = \mu A \omega'_1$ then $\eta \overset{*}{\Rightarrow} \epsilon$, so we only need to show that $a \notin \text{FOLLOW}(B)$. But if $a \in \text{FOLLOW}(B)$ then

$$S \overset{*}{\Rightarrow} \delta_1 B a \delta_2 \overset{*}{\Rightarrow} \delta_1 \mu A \omega'_1 a \delta_2 \overset{*}{\Rightarrow} \delta_1 \mu A a \delta_2$$

so $a \in \text{FOLLOW}(A)$, which is a contradiction. Thus the result is proved.

Theorem 1 *1. If Γ is follow-determined then Γ is simply follow-determined.*

2. If Γ is left factored and simply follow-determined then Γ is follow-determined.

Proof (1) Suppose that Γ is not simply follow determined. Then there is a reachable $A \in U$ and there is $\beta \in \tau_A$ such that

$$A \overset{*}{\Rightarrow} \epsilon \qquad A \Rightarrow \beta \overset{*}{\Rightarrow} a \beta'$$

for some $a \in \text{FOLLOW}(A)$. But then taking $u = \epsilon$ we have

$$A \overset{*}{\Rightarrow} u, \qquad A \overset{*}{\Rightarrow} u a \beta',$$

so Γ is not follow determined.

(2) Suppose that for some reachable A and some $u \in T^*$, $a \in T$, $\gamma \in U^*$ we have $A \overset{*}{\Rightarrow} u$ and $A \overset{*}{\Rightarrow} u a \gamma$. Then by Lemma 1 we have $a \notin \text{FOLLOW}(A)$, as required.

Corollary 1 *LL(1) grammars are left factored and follow-determined.*

2.3 Left recursion and follow-determinism

A non-terminal A is *non-trivially left recursive* if there is a derivation $A \stackrel{*}{\Rightarrow} Aau$ for some $a \in T$ and some $u \in T^*$.

Theorem 2 *If a grammar Γ contains a reachable useful non-trivially left recursive non-terminal A then Γ is follow-non-determined.*

Proof Since A is useful we have $A \stackrel{*}{\Rightarrow} v$ for some $v \in T^*$. Then, since A is non-trivially left recursive we have

$$A \stackrel{*}{\Rightarrow} Aau \stackrel{*}{\Rightarrow} vau \qquad A \stackrel{*}{\Rightarrow} Aau \stackrel{*}{\Rightarrow} Aauau \stackrel{*}{\Rightarrow} vauau.$$

Since A is reachable we have

$$S \stackrel{*}{\Rightarrow} \alpha A \beta \stackrel{*}{\Rightarrow} \alpha A a u \beta.$$

Hence $a \in \mathrm{FOLLOW}(A)$ and thus Γ is not follow-determined.

3 Follow-determinism and LR(1) grammars

When a grammar for a language has been constructed, the author often wants to use the grammar to create a parser or compiler for the language. A common action is to run the grammar through a parser generator. If the grammar does not have the properties required by the particular method on which the parser generator is based, then the parser generator will fail. In the case of an LR parser generator, the failure is usually detected by finding multiple entries in a parse table. However, the author wants to know what the problem is in terms of the grammar rules, not in terms of the parse table, so that the grammar can be modified to remove the problem. The definition of an LL(1) grammar is in terms of properties of the grammar rules, and hence LL(1) parser generators are able to give suitable diagnostics when they are presented with a non-LL(1) grammar. It is well known that ambiguous grammars (grammars which allow two essentially different derivations of some sentence) do not admit LR(1) parsers, but there are grammars which are neither ambiguous nor LR(1). In order to pin-point the problem areas in such grammars it is useful to have grammar rule level properties which must be satisfied by LR(1) grammars. Follow-determinism can be used in this way.

In this section we shall show that any follow-non-determinism in an LR(1) grammar must arise from left recursion, and we shall give an example of a follow-determined grammar for which there is no equivalent LR(1) grammar. In addition, in Section 4 we shall show that applying the standard left recursion removal algorithm to an LR(1) can result in a grammar which is no longer LR(1).

The following definition is taken from [AU72]:

Given a context-free grammar $\Gamma = (U, T, S, \mathcal{P})$ form the augmented grammar $\Gamma' = (U', T, S', \mathcal{P}')$ by adding a new start symbol S' and a new production

S'::=$\{S\}$. Then Γ is an *LR(1)-grammar* if whenever there are two derivations

$$S' \underset{rm}{\overset{*}{\Rightarrow}} \alpha A a u \underset{rm}{\Rightarrow} \alpha\beta a u$$

$$S' \underset{rm}{\overset{*}{\Rightarrow}} \gamma B x \underset{rm}{\Rightarrow} \alpha\beta a y$$

where $u, x, y \in T^*$, $a \in T$, $A, B \in U$ and $\alpha, \beta, \gamma \in U^*$ then

$$\gamma B x = \alpha A a y.$$

(Here $\underset{rm}{\Rightarrow}$ means a right-most derivation, that is a derivation in which at each step the right-most non-terminal is replaced. For example,

$$S \Rightarrow aABb \Rightarrow aAbb \Rightarrow acABbb \Rightarrow acAbb \Rightarrow acabbb$$

is a right-most derivation.)

We begin by giving an equivalent definition of follow non-determinism in terms of right-most derivations.

Proposition 1 *A context-free grammar Γ is not follow-determined if and only if there exists a reachable non-terminal $E \in U$, useful strings $\alpha, \gamma, \delta \in U^*$ and $b \in T$ such that*

$$D \underset{rm}{\overset{+}{\Rightarrow}} \alpha u, \quad D \underset{rm}{\overset{+}{\Rightarrow}} \alpha u b v, \quad \gamma D \delta \in \tau_E, \quad b \in FIRST(\delta).$$

(Here $\underset{rm}{\overset{+}{\Rightarrow}}$ means a derivation of length at least 1).

Proof If the conditions in the statement of the proposition hold then, since α, δ are useful and E is reachable

$$\gamma \overset{*}{\Rightarrow} w \in T^*, \qquad S \overset{*}{\Rightarrow} \eta E \mu \Rightarrow \eta\gamma D \delta \mu,$$

so $b \in \mathrm{FOLLOW}(D)$ and

$$D \overset{*}{\Rightarrow} w u, \qquad D \overset{*}{\Rightarrow} w u b v.$$

Thus Γ is not follow-determined.

Suppose now that Γ is not follow-determined. Then there exists $A \in U$, $u, v \in T^*$, $b \in T$ such that

$$A \overset{*}{\Rightarrow} u, \qquad A \overset{*}{\Rightarrow} u b v, \qquad b \in \mathrm{FOLLOW}(A).$$

(Since u, ubv are sentences there are non-trivial derivations

$$A \underset{rm}{\overset{+}{\Rightarrow}} u, \qquad A \underset{rm}{\overset{+}{\Rightarrow}} u b v.)$$

Suppose also that $S \overset{k}{\Rightarrow} \eta A \mu$, where $b \in \mathrm{FIRST}(\mu)$. We prove the result by induction on k.

If $k = 1$ then $\eta A \mu \in \tau_S$ we can take $\alpha = \epsilon$, $D = A$, and $E = S$.

Now suppose that $k \geq 2$, that the result is true if $S \stackrel{k-1}{\Rightarrow} \eta A\mu$, and that

$$S \stackrel{k-1}{\Rightarrow} \beta C\psi \Rightarrow \beta\rho\psi = \eta A\mu.$$

If $\beta = \eta A\mu'$ or if $\psi = \eta' A\mu$ then $S \stackrel{k-1}{\Rightarrow} \beta C\eta' A\mu$ or $S \stackrel{k-1}{\Rightarrow} \eta A\mu' C\psi$ and in either case the result follows from the inductive hypothesis. Thus we assume that $\rho = \eta' A\mu'$.

If $b \in \text{FIRST}(\mu')$ then we have the required result with $D = A$ and $E = C$. Otherwise, we have $\mu' \underset{rm}{\stackrel{\epsilon}{\Rightarrow}}$ and $b \in \text{FIRST}(\psi)$. Furthermore, since

$$A \stackrel{*}{\Rightarrow} u \qquad A \stackrel{*}{\Rightarrow} ubv$$

we have

$$C \Rightarrow \eta' A\mu' \stackrel{*}{\Rightarrow} \eta' u, \qquad C \Rightarrow \eta' A\mu' \stackrel{*}{\Rightarrow} \eta' ubv.$$

Since $S \stackrel{k-1}{\Rightarrow} \beta C\psi$ and $b \in \text{FIRST}(\psi)$ we can apply the induction hypothesis to the non-terminal C, and hence the result holds.

Theorem 3 *If Γ is an LR(1) grammar and for some $\nu \in U^*$, $A \underset{rm}{\stackrel{*}{\Rightarrow}} \nu$, $A \underset{rm}{\stackrel{*}{\Rightarrow}} \nu av$ and $S \underset{rm}{\stackrel{*}{\Rightarrow}} \mu Aaw$, then $A \underset{rm}{\stackrel{+}{\Rightarrow}} Aav$.*

Proof Suppose that

$$A \underset{rm}{\stackrel{n}{\Rightarrow}} \nu, \qquad A \underset{rm}{\stackrel{m}{\Rightarrow}} \nu av, \text{ where } n \geq 0, m \geq 1.$$

If $n = 0$ then $\nu = A$ and $A \underset{rm}{\stackrel{m}{\Rightarrow}} Aav$, as required. So we shall suppose that $n \geq 1$ and that

$$A \underset{rm}{\stackrel{n-1}{\Rightarrow}} \alpha Bq \underset{rm}{\Rightarrow} \alpha\beta q = \nu, \text{ where } q \in T^*.$$

We prove the result by induction on m.

If $m = 1$ then

$$A \Rightarrow \nu av.$$

So, in the augmented grammar Γ' we have

$$S' \underset{rm}{\stackrel{*}{\Rightarrow}} \mu Aaw \underset{rm}{\stackrel{*}{\Rightarrow}} \mu\alpha Bqaw \Rightarrow \mu\alpha\beta qaw$$

$$S' \underset{rm}{\stackrel{*}{\Rightarrow}} \mu Aaw \Rightarrow \mu\nu avaw = \mu\alpha\beta qavaw.$$

If $q \neq \epsilon$ then let $q = bq'$, then

$$\mu\alpha\beta qaw = \mu\alpha\beta b(q'aw), \quad \mu\nu avaw = \mu\alpha\beta b(q'avaw).$$

If $q = \epsilon$ then

$$\mu\alpha\beta qaw = \mu\alpha\beta aw, \quad \mu\nu avaw = \mu\alpha\beta a(vaw).$$

In either case, since Γ is LR(1) we must have

$$\mu\alpha Bqavaw = \mu Aaw.$$

But this is not possible, so if $m = 1$ we must have $n = 0$ and the result follows from above.

Now suppose that $m \geq 2$, that the result holds for $m - 1$, and that

$$A \overset{m-1}{\underset{rm}{\Rightarrow}} \eta D p \underset{rm}{\Rightarrow} \eta \delta p = \nu a v, \quad p \in T^*.$$

Then, in the augmented grammar Γ' we have

$$S' \overset{*}{\underset{rm}{\Rightarrow}} \mu A a w \overset{*}{\underset{rm}{\Rightarrow}} \mu \alpha B q a w \Rightarrow \mu \alpha \beta q a w$$

$$S' \overset{*}{\underset{rm}{\Rightarrow}} \mu A a w \overset{m-1}{\underset{rm}{\Rightarrow}} \mu \eta D p a w \Rightarrow \mu \eta \delta p a w = \mu \alpha \beta q a v a w.$$

Since Γ is LR(1) we must have

$$\mu \alpha B q a v a w = \mu \eta D p a w.$$

As $p \in T^*$ we must have $p = qav$, then $D = B$ and $\eta = \alpha$, so

$$A \overset{m-1}{\underset{rm}{\Rightarrow}} \alpha B q a v, \quad A \overset{*}{\underset{rm}{\Rightarrow}} \alpha B q$$

so the result follows by induction.

Theorem 4 *There is a language L which has a follow-determined context-free grammar but which is not LR. In particular there is no LR(1) grammar for L.*

Proof Let

$$L = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}.$$

It is known, see [AU72] that L is not LR, i.e. cannot be generated by an LR grammar.

Clearly the grammar

$$S ::= \{aAb, aBb^2\}, \ A ::= \{aAb, \epsilon\}, \ B ::= \{aBb^2, \epsilon\}$$

generates L. By Proposition 1, if Γ is not follow determined then we must have

$$A \overset{*}{\underset{rm}{\Rightarrow}} \alpha u, \qquad A \overset{*}{\underset{rm}{\Rightarrow}} \alpha u b v$$

or

$$B \overset{*}{\underset{rm}{\Rightarrow}} \alpha u, \qquad B \overset{*}{\underset{rm}{\Rightarrow}} \alpha u b v$$

for some $\alpha \in U^*$ and $u, v \in T^*$. But

$$A \overset{*}{\underset{rm}{\Rightarrow}} a^n A b^n \text{ or } A \overset{*}{\underset{rm}{\Rightarrow}} a^n b^n$$

so we cannot have

$$A \overset{*}{\underset{rm}{\Rightarrow}} \alpha u, \qquad A \overset{*}{\underset{rm}{\Rightarrow}} \alpha u b v.$$

Similarly,

$$B \overset{*}{\underset{rm}{\Rightarrow}} a^n B b^{2n} \text{ or } B \overset{*}{\underset{rm}{\Rightarrow}} a^n b^{2n}$$

so we cannot have

$$B \overset{*}{\underset{rm}{\Rightarrow}} \alpha u, \qquad B \overset{*}{\underset{rm}{\Rightarrow}} \alpha u b v.$$

Thus Γ is follow determined.

4 Follow determinism and left recursion removal

In this section we consider the circumstances under which left recursion removal can introduce new follow non-determinism. In particular, we look at cases in which applying left recursion removal to an LR(1) grammar can result in a grammar which is no longer LR(1).

4.1 The standard left recursion removal algorithm

The standard left recursion algorithm [ASU86, AU72] requires the non-terminals to be ordered, say $A_1, \ldots, A_h$. Then for each non-terminal, A_i, in order, replace each production A_i::=$A_j\alpha$ with productions $\{A_i$::=$\gamma\alpha \mid \gamma \in \tau_{A_j}\}$, for $j = 1, 2, \ldots, i-1$, in order. Then it is necessary to replace all the productions A_i::=$A_i\alpha$ by introducing a new non-terminal. So replace

$$A_i ::= \ \{A_i\alpha_1,\ \ldots,\ A_i\alpha_n,\ \beta_1,\ \ldots,\ \beta_m\}$$

with

$$\begin{aligned} A_i &::= \ \{\beta_1 X_i,\ \ldots,\ \beta_m X_i\} \\ X_i &::= \ \{\alpha_1 X_i,\ \ldots,\ \alpha_n X_i,\ \epsilon\} \end{aligned}$$

where X_i is a new non-terminal symbol and the β_j do not begin with A_i.

Later in this section we shall show that the left recursion removal algorithm described above is not robust with respect to follow determinism, in the sense that whether the resulting grammar is follow-determined will usually depend on the order which has been placed on the non-terminals. Surprisingly, it is possible (and it could be argued natural) to choose an ordering for the non-terminals so that the result of applying the left recursion removal algorithm to an LR(1) grammar is a grammar which is no longer LR(1). Since one of the things that the left recursion removal algorithm is used for is as a step towards making a grammar LL(1) this observation is an important concern.

Before discussing the general case we shall look at a specific example.

4.2 An example

```
S ::= {Ba,  bB}
B ::= {Sd, e}
```

In the next section we shall show that this grammar is LR(1). Here we show that any application of the standard left recursion removal algorithm, and indeed back substitution, will result in a grammar which is not LR(1).

Suppose first that we order the non-terminals $S < B$. The left recursion removal algorithm gives the following steps:

```
1. S ::= {Ba, bB}
   B ::= {Bad, bBd, e}
```

```
2. S ::= {Ba, bB}
   B ::= {bBdX, eX}
   X ::= {adX, epsilon}
```

The final grammar is not LR(1) because it is not left recursive but it is follow-non-determined. We have $X \stackrel{*}{\Rightarrow} ad$, $X \stackrel{*}{\Rightarrow} adad$ and $S \stackrel{*}{\Rightarrow} eXa$, so $a \in FOLLOW(X)$.

To confirm that it is not LR(1) we observe

$$S \Rightarrow Ba \Rightarrow eXa \Rightarrow ea$$

$$S \Rightarrow Ba \Rightarrow eXa \Rightarrow eadXa \Rightarrow eada$$

and use the definition given at the beginning of Section 3.

Now suppose that we order the non-terminals $B < S$. The left recursion removal algorithm gives the following steps:

```
1. S ::= {Sda, ea, bB}          2. S ::= {eaX, bBX}
   B ::= {Sd, e}                   B ::= {Sd, e}
                                   X ::= {daX, epsilon
```

Again the final grammar is not LR(1) because it is not left recursive but it is follow-non-determined. We have $X \stackrel{*}{\Rightarrow} da$, $X \stackrel{*}{\Rightarrow} dada$ and $S \stackrel{*}{\Rightarrow} beaXdX$, so $d \in FOLLOW(X)$.

To confirm that it is not LR(1) we observe

$$S \Rightarrow bBX \Rightarrow bB \Rightarrow bSd \Rightarrow beaXd \Rightarrow bead$$

$$S \Rightarrow bBX \Rightarrow bB \Rightarrow bSd \Rightarrow beaXd \Rightarrow beadaXd \Rightarrow beadad$$

Now suppose that we try back substitution and then left recursion removal:

```
1. S ::= {Sda, ea, bSd, be}          2. S ::= {eaX, bSdX, beX}
                                        X ::= {daX, epsilon}
```

In the final grammar we have $X \stackrel{*}{\Rightarrow} da$, $X \stackrel{*}{\Rightarrow} dada$, and $S \stackrel{*}{\Rightarrow} beaXdX$ so $d \in$ FOLLOW(X), and the grammar is not LR(1).

4.3 The Example is SLR(1)

We use a standard algorithm, see for example [ASU86], to construct the SLR(1) parse table for the augmented version of the grammar given at the beginning of the previous section. The parse table has no repeated entries, showing that the grammar is SLR(1) and hence LR(1).

```
0. T ::= S
1. S ::= Ba
2. S ::= bB
3. B ::= Sd
4. B ::= e
```

The following Ij are the states formed from the sets of grammar items.

```
I0 = [T-> .S ]                          FOLLOW(S) = {d,$}
     [S-> .Ba]                          FOLLOW(B) = {a,d,$}
     [S-> .bB]
     [B-> .Sd]
     [B-> .e ]

I1 = g(0,S) = [T->S. ]
              [B->S.d]


I2 = g(0,B) = [S->B.a]


I3 = g(0,b) = [S->b.B ]= g(3,b)
              [B-> .Sd]
              [B-> .e ]
              [S-> .Ba]
              [S-> .bB]

I4 = g(0,e) = [B->e.]  = g(3,e)

I5 = g(1,d) = [B->Sd.] = g(7,d)

I6 = g(2,a) = [S->Ba.] = g(8,a)

I7 = g(3,S) = [B->S.d]

I8 = g(3,B) = [S->bB.]
              [S->B.a]
```

This gives the following action table

	a	b	d	e	$
I0		s3		s4	
I1			s5		acc
I2	s6				
I3		s3		s4	
I4	r4		r4		r4
I5	r3		r3		r3
I6			r1		r1
I7			s5		
I8	s6		r2		r2

4.4 The general case

We now consider the effect of various applications of the left recursion removal algorithm on a general system of rules.

4.4.1 The rule system and conditions

Suppose that the grammar Γ has a left recursive non-terminal A. There is a sequence of productions

$$\begin{aligned} A &::= B_1\gamma_1 \\ B_1 &::= B_2\gamma_2 \\ &\vdots \\ B_{k-1} &::= B_k\gamma_k \\ B_k &::= A\gamma_0 \end{aligned}$$

To avoid reasoning about A separately from the B_i, we set $B_0 = A$.

We call a system of rules of this form a recursive loop. If $k \geq 1$ we say that the loop is *indirect*. We say that a non-terminal symbol B_h is *reachable from outside the loop* if there is a derivation

$$S \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_m \Rightarrow \delta_1 B_h \delta_2$$

in which none of the steps is involves replacing a B_i, $0 \leq i \leq k$, i.e. there are no steps of the form

$$\gamma_j = \delta B_i \nu \Rightarrow \delta\mu\nu = \gamma_{j+1}.$$

The point here is that if the grammar is modified by changing the grammar rules in the recursive loop the above derivation will still be a derivation in the new grammar and B_h will still be reachable.

Suppose that

1. $k \geq 1$
2. for $0 \leq i \leq k$ there exists $u_i \in T^*$ such that $\gamma_i \stackrel{*}{\Rightarrow} u_i$
3. for some $i \geq 1$, $u_i \neq \epsilon$, so $u = u_k \ldots u_1 \neq \epsilon$
4. A is reachable from outside the loop
5. for some l, there exists $\alpha \in \tau_{B_l}$ such that $\alpha \stackrel{*}{\Rightarrow} v \in T^*$ and α does not begin with any B_i, $0 \leq i \leq k$.

We shall discuss these conditions below, but first we look at the result of applying the standard left recursion removal algorithm to the rule system above.

4.4.2 Ordering $A < B_1 < \ldots < B_k$

Suppose that we order the non-terminals $A = B_0 < B_1 < \ldots < B_k$ and apply the substitution part of the left recursion algorithm. We get

$$\begin{aligned} A &::= B_1\gamma_1 \mid \ldots \\ B_1 &::= B_2\gamma_2 \mid \ldots \\ &\vdots \\ B_l &::= B_{l+1}\gamma_{l+1} \mid \alpha \mid \ldots \\ &\vdots \\ B_{k-1} &::= B_k\gamma_k \mid \ldots \\ B_k &::= B_k\gamma_k \ldots \gamma_0 \mid \alpha\gamma_l \ldots \gamma_0 \mid \ \ldots \end{aligned}$$

Then removing the immediate left recursion using a new non-terminal symbol X gives

$$\begin{aligned} A &::= B_1\gamma_1 \mid ... \\ B_1 &::= B_2\gamma_2 \mid ... \\ &\vdots \\ B_l &::= B_{l+1}\gamma_{l+1} \mid \alpha \mid ... \\ &\vdots \\ B_{k-1} &::= B_k\gamma_k \mid ... \\ B_k &::= \alpha\gamma_l \ldots \gamma_0 X \mid \ ... \\ X &::= \gamma_k \ldots \gamma_0 X \mid \ \ldots \ \mid\epsilon \end{aligned}$$

Note that, since α does not begin with any B_i it is unaffected by any of the steps in the left recursion removal algorithm.

Now, in the new grammar just constructed,

$$B_k \Rightarrow \alpha\gamma_l \ldots \gamma_0 X \Rightarrow \alpha\gamma_l \ldots \gamma_0 \stackrel{*}{\Rightarrow} vu_l \ldots u_0$$

$$B_k \stackrel{*}{\Rightarrow} \alpha\gamma_l \ldots \gamma_0\gamma_k \ldots \gamma_0 X \Rightarrow \alpha\gamma_l \ldots \gamma_0\gamma_k \ldots \gamma_1\gamma_0 \stackrel{*}{\Rightarrow} vu_l \ldots u_0 a u' u_0$$

where $u_k \ldots u_1 = u = au'$.

In the original grammar A, and hence all the B_i were reachable. In the new grammar A is still reachable because A was chosen to be reachable from outside the loop and the left recursion removal algorithm only modifies grammar rules whose left hand sides are one of the B_i. Hence all the B_i are still reachable. (Note, it need not be the case that all the B_i are reachable in the new grammar if a different ordering is used for the non-terminals. This is essentially what allows some applications of left recursion removal to avoid the introduction of additional follow non-determinism, see below.)

Since A is reachable we have, in the new grammar,

$$S \stackrel{*}{\Rightarrow} \delta_1 A \delta_2 \Rightarrow \delta_1 B_1 \gamma_1 \delta_2 \Rightarrow \ldots \Rightarrow \delta_1 B_k \gamma_k \ldots \gamma_1 \delta_2 \stackrel{*}{\Rightarrow} \delta_1 B_k a u' \delta_2$$

so $a \in \mathrm{FOLLOW}(B_k)$ and the new grammar is not follow-determined.

4.4.3 Analysis of the conditions

Next we consider the implications of the five conditions from Section 4.4.1 that we assumed the grammars rules in question satisfied.

Condition 1

If $k = 0$ then rule system becomes

$$A ::= A\gamma_0 \mid \alpha \mid ...$$

and we have immediate left recursion. In this case there is only one non-terminal to consider and there is no ordering of non-terminals to be applied. The left recursion removal algorithm generates the following rules

$$\begin{aligned} A &::= \alpha X \mid ... \\ X &::= \gamma_0 X \mid ... \mid \epsilon. \end{aligned}$$

and no new follow non-determinism is introduced. (Here FOLLOW(X) = FOLLOW(A), but this new FOLLOW(A) won't contain $a \in$ FIRST(γ_0) unless a can follow A as a result of some other grammar rule in the original grammar.)

Condition 2

If, for some i, u_i is useless then we can remove the string $B_i\gamma_i$ from $\tau_{B_{i-1}}$ and this particular instance of left recursion will no longer exist.

Condition 3

If each of the γ_i only derives ϵ then in some sense the indirect nature of the left recursion is spurious. This case is essentially the same as that in Condition 1. Indeed, direct left recursion can always be turned into indirect left recursion by adding new non-terminals and taking the γ_i to be empty. So, for example, $A ::= A\gamma_0$ can become

$$A ::= X_1, \qquad X_1 ::= X_2, \qquad X_2 ::= A\gamma_0$$

Condition 4

If A is not reachable then nor are any of the B_i and the whole rule system can be removed from the grammar without changing the language generated. If A is reachable then at least one of the B_i must be reachable from outside the loop. If A is not reachable from outside the loop then we can rename the non-terminals to make A the non-terminal which is reachable from outside. The above discussion then applies under this new naming.

Condition 5

If, for $0 \leq i \leq k$, $\gamma \in \tau_{B_i}$ implies that γ begins with some B_j, where $0 \leq j \leq k$, then if $A \overset{*}{\Rightarrow} \alpha$ we have that α must begin with some B_i, thus $\alpha \notin T^*$. So A is useless and we can just remove it and its grammar rule from the grammar. So again this instance of left recursion can be removed.

4.4.4 Other non-terminal orderings

Next we look at the effect of chosing a different ordering of the non-terminals when applying the algorithm. Ordering the non-terminals so that A, the one reachable from outside the loop, is larger than all the others

$$B_1 < \ldots < B_k < A$$

then applying the left recursion algorithm gives

$$\begin{aligned}
A &::= \alpha\gamma_l \ldots \gamma_1 X \mid ... \\
X &::= \gamma_0\gamma_k \ldots \gamma_2\gamma_1 X \mid \ \ldots \ \mid \epsilon \\
B_1 &::= B_2\gamma_2 \mid ... \\
&\ \vdots \\
B_l &::= B_{l+1}\gamma_{l+1} \mid \alpha \mid ...
\end{aligned}$$

$$\begin{aligned} &\vdots \\ B_{k-1} &::= B_k\gamma_k \mid ... \\ B_k &::= A\gamma_0 \end{aligned}$$

If none of the B_i, $1 \leq i \leq k$, was reachable from outside the loop in the original grammar then they are all unreachable in the new grammar and the left recursion removal algorithm has not introduced any new follow non-determinism in this case. If B_h, say, was reachable from outside the original loop then we have, in the new grammar,

$$S \stackrel{*}{\Rightarrow} \delta_1 B_h \delta_2 \stackrel{*}{\Rightarrow} \delta_1 A\gamma_0\gamma_k \ldots \gamma_{h+1}\delta_2$$

so

$$\mathrm{FIRST}(\gamma_0\gamma_k \ldots \gamma_{h+1}) \subseteq \mathrm{FOLLOW}(A) = \mathrm{FOLLOW}(X)$$

Since

$$A \stackrel{*}{\Rightarrow} \alpha\gamma_l \ldots \gamma_1, \qquad A \stackrel{*}{\Rightarrow} \alpha\gamma_l \ldots \gamma_1\gamma_0\gamma_k \ldots \gamma_1$$

if $\mathrm{FIRST}(\gamma_0\gamma_k \ldots \gamma_{h+1}) \not\subseteq \{\epsilon\}$ then new follow non-determinism has been introduced into the grammar.

Applying the left recursion algorithm using the following ordering

$$B_k < \ldots < B_1 < A$$

gives

$$\begin{aligned} A &::= \alpha\gamma_l \ldots \gamma_1 X \mid ... \\ X &::= \gamma_0\gamma_k \ldots \gamma_2\gamma_1 X \mid \ldots \mid \epsilon \\ B_1 &::= A\gamma_0\gamma_k \ldots \gamma_2 \mid \alpha\gamma_l \ldots \gamma_2 \mid ... \\ &\vdots \\ B_l &::= A\gamma_0\gamma_k \ldots \gamma_{l+1} \mid \alpha \mid ... \\ &\vdots \\ B_{k-1} &::= A\gamma_0\gamma_k \mid ... \\ B_k &::= A\gamma_0 \end{aligned}$$

If none of the B_i, $1 \leq i \leq k$, was reachable from outside the loop in the original grammar then they are all unreachable in the new grammar and again the left recursion removal algorithm has not introduced any new follow non-determinism. If B_h, say, was reachable from outside the original loop then we have, in the new grammar,

$$S \stackrel{*}{\Rightarrow} \delta_1 B_h \delta_2 \stackrel{*}{\Rightarrow} \delta_1 A\gamma_0\gamma_k \ldots \gamma_{h+1}\delta_2$$

and again if $\mathrm{FIRST}(\gamma_0\gamma_k \ldots \gamma_{h+1}) \not\subseteq \{\epsilon\}$ then new follow non-determinism has been introduced into the grammar.

4.4.5 Summary

If any of the conditions 2, 4, or 5 given in Section 4.4.1 does not hold then aspects of rule system are not required in the sense that removing them would not change the language generated by the grammar. So the left recursion is essentially spurious.

Conditions 1 and 3 apply in cases where the left recursion is essentially direct, and in these cases the left recursion removal algorithm need not introduce new follow non-determinism.

If the non-terminals are ordered so that A is larger than all the other B_i, and if $\mathrm{FIRST}(\gamma_0\gamma_k\ldots\gamma_{h+1}) \subseteq \{\epsilon\}$ for any $1 \leq h \leq k$ such that B_h was reachable from outside the loop in the original grammar, then no new follow non-determinism need be introduced by the left recursion removal algorithm.

4.5 Another example

The grammar

$$S ::= \{A, Ab\} \qquad A ::= \{Aba, a\}$$

is LR(1). We construct its parse table (see [ASU86]) as follows.

$$I_0 = [S' \to .S, \$][S \to .A, \$][S \to .Ab, \$][A \to .Aba, \$/b][A \to .a, \$/b]$$

$$goto(I_0, S) = I_1 = [S' \to S., \$]$$

$$goto(I_0, A) = I_2 = [S \to A., \$][S \to A.b, \$][A \to A.ba, \$/b]$$

$$goto(I_0, a) = I_3 = [A \to a., \$/b]$$

$$goto(I_2, b) = I_4 = [S \to Ab., \$][A \to Ab.a, \$/b]$$

$$goto(I_4, a) = I_5 = [A \to Aba., \$/b]$$

	a	b	$
0	s3		
1			acc
2		s4	r1
3		r3	r3
4	s5		r2
5		r4	r4

Applying the standard left recursion removal algorithm to the grammar gives:

$$S ::= \{A, Ab\} \quad A ::= \{aX\} \quad X ::= \{baX, \epsilon\}$$

then

$$S' \Rightarrow S \Rightarrow Ab \Rightarrow aXb \Rightarrow ab$$

$$S' \Rightarrow S \Rightarrow A \Rightarrow aX \Rightarrow abaX \Rightarrow aba$$

If we don't use the standard algorithm but replace head with tail recursion we get:

$$S ::= \{A, Ab\} \quad A ::= \{abA, a\}$$

then

$$S' \Rightarrow S \Rightarrow Ab \Rightarrow ab$$

$$S' \Rightarrow S \Rightarrow A \Rightarrow abA \Rightarrow aba$$

so its still not LR(1).

In fact, both of these grammars is not LL(k) for any k.

However, we can left factor the second of these grammars to get

$$S::=\{AX\} \quad X::=\{\epsilon, b\} \quad A::=\{aY\} \quad Y::=\{bA, \epsilon\}$$

Now

$$S' \Rightarrow S \Rightarrow AX \Rightarrow Ab \Rightarrow aYb \Rightarrow ab$$

$$S' \Rightarrow S \Rightarrow AX \Rightarrow A \Rightarrow aY \Rightarrow abA \Rightarrow abaY \Rightarrow aba$$

so the new grammar is still not LR(1) and it is also not follow-determined. However, this grammar is LL(2) (see [ASU86] for a definition of LL(2) grammars). It is clearly left factored: what is stopping it from being LL(1), and hence LR(1), is that the follow-determinism needs two symbols of look-ahead!

5 Follow-determinism and left factoring

Recursive descent based parsers are more efficient on grammars which are left factored. However, we shall now show that there exist follow-determined grammars for which there are no equivalent left factored grammars, thus the capability of production version GRD parsers to deal with non-left factored grammars increases the number of languages they can handle.

Let $\Gamma = (U, T, S, \mathcal{P})$ be a context free grammar. For $a \in T$ and $\alpha \in U^*$ define $L_a(\alpha)$ to be the set of strings of terminals beginning with a which are derivable from α. So

$$L_a(\alpha) = \{av \mid \text{for some } v \in T^*,\ \alpha \overset{*}{\Rightarrow} av\}.$$

The idea is to show that if there is a non-terminal B in Γ such that

$$L_a(B) = \{aba, a^2b^2, a^3b^3a, a^4b^4, \ldots\} = L, \text{ say}$$

then Γ cannot be left factored. We then demonstrate that this language L can be generated by a follow-determined context free grammar, and since it must be $L_a(S)$ for the start symbol of any grammar which generates it, L cannot be generated by a left factored grammar.

The main part of the proof relies on the following lemma.

Lemma 2 *Suppose that $\Gamma = (U, T, S, \mathcal{P})$ is a left factored context-free grammar.*

1. *If there exists $B \in U \backslash T$, $a, b \in T$, and $J \geq 0$ such that*

$$L_a(B) = \{ab^{J+1}a, a^2b^{J+2}, a^3b^{J+3}a, a^4b^{J+4}, \ldots\} = X_J, \text{ say,}$$

then there exists $C \in U$ such that

$$L_a(C) = \{ab^{J+2}, a^2b^{J+3}a, a^3b^{J+4}, a^4b^{J+5}a, \ldots\} = Y_{J+1}, \text{ say.}$$

2. If there exists $B \in U \backslash T$, $a, b \in T$, and $J \geq 0$ such that

$$L_a(B) = \{ab^{J+1}, a^2b^{J+2}a, a^3b^{J+3}, a^4b^{J+4}a, \ldots\} = Y_J, \text{ say,}$$

then there exists $C \in U$ such that

$$L_a(C) = \{ab^{J+2}a, a^2b^{J+3}, a^3b^{J+4}a, a^4b^{J+5}, \ldots\} = X_{J+1}, \text{ say.}$$

Proof (1). Let $B_1, \ldots, B_f$ be the non-terminals in Γ such that $L_a(B_i) = X_J$, $1 \leq i \leq f$. Choose i such that B_i has a shortest derivation of $ab^{J+1}a$ and set $B = B_i$. So

$$B \overset{n}{\Rightarrow} ab^{J+1}a$$

and

$$B_j \overset{m}{\Rightarrow} ab^{J+1}a \quad \text{implies} \quad m \geq n.$$

Suppose first that there exist $\alpha, \beta \in \tau_B$ such that $\alpha \overset{*}{\Rightarrow} av_1$ and $\beta \overset{*}{\Rightarrow} av_2$, for some $v_1, v_2 \in T^*$. Then $a \in \mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta)$, and, since the grammar is left factored, $\alpha = \beta$. Thus there exists $\alpha = x_1 \ldots x_l \in \tau_B$ such that

$$\text{if} \quad B \Rightarrow \gamma \overset{*}{\Rightarrow} w \in L_a(B) = X_J \quad \text{then} \quad \gamma = \alpha.$$

Furthermore, there exist $v_i \in T^*$ such that

$$B \Rightarrow x_1 \ldots x_l \overset{n-1}{\Rightarrow} v_1 \ldots v_l,$$

$$\text{where} \quad x_i \overset{*}{\Rightarrow} v_i, \quad \text{and} \quad v_1 \ldots v_l = ab^{J+1}a.$$

Suppose that

$$v_{m+1} \ldots v_l = p, \quad v_m = u_2bq, \quad v_1 \ldots v_{m-1} = u_1, \quad qp = a,$$

so that

$$x_1 \ldots x_{m-1} \overset{s}{\Rightarrow} u_1, \quad x_m \overset{t}{\Rightarrow} u_2bq, \quad u_1u_2bqp = ab^{J+1}a, \quad s + t \leq n - 1.$$

First we note that we can essentially ignore $x_{m+1} \ldots x_l$ because it can only derive the empty string. If

$$x_{m+1} \ldots x_l \overset{*}{\Rightarrow} w \in T^*$$

$$\text{then} \quad B \Rightarrow x_1 \ldots x_{m-1}x_mx_{m+1} \ldots x_l \overset{*}{\Rightarrow} u_1u_2bqw = ab^{J+1}qw \in L_a(B).$$

From the structure of the elements of $L_a(B)$ we see that we must have $ab^{J+1}qw = ab^{J+1}a$ and so $w = p$. Then, since all the elements in $L_a(B)$ must be derived using $x_1 \ldots x_l$, they must all end in p. Since $a^2b^{J+2} \in L_a(B)$, we have $p \neq a$ and hence $p = \epsilon$, as required.

Note, we have also shown that $q = a$, i.e. $x_m \overset{*}{\Rightarrow} u_2ba$.

In the remainder of the proof we repeatedly use the consequence of our reasoning so far, that

$$w \in L_a(B) \text{ if and only if } w = aw' = w_1w_2 \text{ and } x_1 \ldots x_{m-1} \overset{*}{\Rightarrow} w_1, \; x_m \overset{*}{\Rightarrow} w_2.$$

Next we aim to show that $L_a(x_1 \dots x_{m-1}) = \{a\}$. We begin by showing that this set is non-empty.

If $L_a(x_1 \dots x_{m-1}) = \emptyset$ then $B \Rightarrow \alpha \stackrel{*}{\Rightarrow} aw$ if and only if $x_m \stackrel{*}{\Rightarrow} aw$, so

$$L_a(x_m) = L_a(B) = X_J,$$

and x_m is one of the B_i, $1 \le i \le f$. In addition, $u_1 = \epsilon$ so $x_m \stackrel{t}{\Rightarrow} u_2 b = ab^{J+1}a$, where $t \le n-1$. This is contrary to the choice of B, hence there exists some $v_0 \in T^*$ such that

$$x_1 \dots x_{m-1} \stackrel{*}{\Rightarrow} av_0.$$

We now show that all the elements of $L_a(x_1 \dots x_{m-1})$ must be of the form $a^{2r-1}b^h$, and then that the only possibility is $r = 1$ and $h = 0$.

If $u_1 = \epsilon$ then $u_2 = ab^J$ and so

$$B \Rightarrow \alpha \stackrel{*}{\Rightarrow} av_0 x_m \stackrel{*}{\Rightarrow} av_0 u_2 ba = av_0 ab^{J+1} a \notin L_a(B).$$

Thus for some k we must have

$$u_1 = ab^k, \qquad u_2 = b^{J-k}.$$

If $x_1 \dots x_{m-1} \stackrel{*}{\Rightarrow} av$ we have

$$B \Rightarrow \alpha \stackrel{*}{\Rightarrow} avx_m \stackrel{*}{\Rightarrow} avu_2 ba = avb^{J-k+1}a \in L_a(B).$$

By the structure of the elements of $L_a(B)$ we must have, for some $r \ge 1$,

$$v = a^{2r-2}b^{k+2r-2}.$$

We shall show that in fact the only possibility is $r = 1$ and $h = 0$.

Since $a^2 b^{J+2} \in L_a(B)$ there must exist w_1, w_2 such that

$$x_1 \dots x_{m-1} \stackrel{*}{\Rightarrow} w_1, \quad x_m \stackrel{*}{\Rightarrow} w_2, \quad w_1 w_2 = a^2 b^{J+2}.$$

If $w_1 = \epsilon$ then $w_2 = a^2 b^{J+2}$ and

$$B \stackrel{*}{\Rightarrow} x_1 \dots x_{m-1} x_m \stackrel{*}{\Rightarrow} av_0 a^2 b^{J+2} \notin L_a(B).$$

Thus $w_1 \ne \epsilon$ and so there must exist $s \ge 1$ such that

$$w_1 = aa^{2s-2}b^{k+2s-2}.$$

But then

$$a^2 b^{J+2} = a^{2s-1} b^{k+2s-2} w_2 \ \text{ implies } \ s = 1, \ k = 0, \ w_2 = ab^{J+2}.$$

If $av \in L_a(x_1 \dots x_{m-1})$ then

$$B \stackrel{*}{\Rightarrow} x_1 \dots x_{m-1} x_m \stackrel{*}{\Rightarrow} avw_2 = a^{2r-1} b^{2r-2} ab^{J+2} \in L_a(B)$$

and hence, by the structure of $L_a(B)$, $r = 1$. Thus we have shown that

$$L_a(x_1 \dots x_{m-1}) = \{a\}.$$

Clearly,

$$Y_{J+1} = \{aw \mid w \in T^*,\ a^2w \in X_J\}.$$

We have shown

$$a^2w \in X_J = L_a(B) \ \Leftrightarrow\ a^2w = w_1w_2, x_1 \ldots x_{m-1} \stackrel{*}{\Rightarrow} w_1, x_m \stackrel{*}{\Rightarrow} w_2 \ \Leftrightarrow\ w_1 = a, x_m \stackrel{*}{\Rightarrow} aw$$

so

$$L_a(x_m) = \{aw \mid w \in T^*, x_m \stackrel{*}{\Rightarrow} aw\} = \{aw \mid w \in T^*, a^2w \in X_J\} = Y_{J+1}$$

and we can take $C = x_m$.

(2). The proof in this case is essentially the same as the proof of (1), but it is included for completeness.

Let $B_1, \ldots, B_f$ be the non-terminals in Γ such that $L_a(B_i) = Y_J$, $1 \le i \le f$. Choose i such that B_i has a shortest derivation of ab^{J+1} and set $B = B_i$. So

$$B \stackrel{n}{\Rightarrow} ab^{J+1}$$

and

$$B_j \stackrel{m}{\Rightarrow} ab^{J+1} \quad \text{implies} \quad m \ge n.$$

If there exist $\alpha, \beta \in \tau_B$ such that $\alpha \stackrel{*}{\Rightarrow} av_1$ and $\beta \stackrel{*}{\Rightarrow} av_2$, for some $v_1, v_2 \in T^*$, then $a \in \mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta)$, and $\alpha = \beta$. Thus there exists $\alpha = x_1 \ldots x_l \in \tau_B$ such that

$$\text{if} \quad B \Rightarrow \gamma \stackrel{*}{\Rightarrow} w \in L_a(B) = X_J \quad \text{then} \quad \gamma = \alpha.$$

Furthermore, there exist $v_i \in T^*$ such that

$$B \Rightarrow x_1 \ldots x_l \stackrel{n-1}{\Rightarrow} v_1 \ldots v_l,$$

$$\text{where} \quad x_i \stackrel{*}{\Rightarrow} v_i, \quad \text{and} \quad v_1 \ldots v_l = ab^{J+1}.$$

Suppose that for $m + 1 \le j \le l$,

$$v_i = \epsilon, \quad u_2b, \quad v_1 \ldots v_{m-1} = u_1,$$

so that

$$x_1 \ldots x_{m-1} \stackrel{s}{\Rightarrow} u_1, \quad x_m \stackrel{t}{\Rightarrow} u_2b, \quad u_1u_2b = ab^{J+1}, \quad s + t \le n - 1.$$

First we show that $x_{m+1} \ldots x_l$ can only derive the empty string. If

$$x_{m+1} \ldots x_l \stackrel{*}{\Rightarrow} w \in T^*$$

$$\text{then} \quad B \Rightarrow x_1 \ldots x_{m-1}x_mx_{m+1} \ldots x_l \stackrel{*}{\Rightarrow} u_1u_2bw = ab^{J+1}w \in L_a(B).$$

From the structure of the elements of Y_J we see that we must have $ab^{J+1}w = ab^{J+1}$ and so $w = \epsilon$.

In the remainder of the proof we repeatedly use the consequence of our reasoning so far, that

$$w \in L_a(B) \ \text{ if and only if } \ w = aw' = w_1w_2 \text{ and } x_1 \ldots x_{m-1} \stackrel{*}{\Rightarrow} w_1,\ x_m \stackrel{*}{\Rightarrow} w_2.$$

We aim to show that $L_a(x_1 \dots x_{m-1}) = \{a\}$. We begin by showing that this set is non-empty.

If $L_a(x_1 \dots x_{m-1}) = \emptyset$ then $B \stackrel{*}{\Rightarrow} aw$ if and only if $x_m \stackrel{*}{\Rightarrow} aw$, so

$$L_a(x_m) = L_a(B) = Y_J,$$

and x_m is one of the B_i, $1 \le i \le f$. In addition, $u_1 = \epsilon$ so $x_m \stackrel{t}{\Rightarrow} u_2 b = ab^{J+1}$, where $t \le n-1$. This is contrary to the choice of B, hence there exists some $v_0 \in T^*$ such that

$$x_1 \dots x_{m-1} \stackrel{*}{\Rightarrow} av_0.$$

We now show that all the elements of $L_a(x_1 \dots x_{m-1})$ must be of the form $a^{2r-1}b^h$, and then that the only possibility is $r = 1$ and $h = 0$.

If $u_1 = \epsilon$ then $u_2 = ab^J$ and so

$$B \Rightarrow \alpha \stackrel{*}{\Rightarrow} av_0 x_m \stackrel{*}{\Rightarrow} avu_2 b = av_0 ab^{J+1} \notin L_a(B).$$

Thus for some k we must have

$$u_1 = ab^k, \qquad u_2 = b^{J-k}.$$

If $x_1 \dots x_{m-1} \stackrel{*}{\Rightarrow} av$ we have

$$B \Rightarrow \alpha \stackrel{*}{\Rightarrow} avx_m \stackrel{*}{\Rightarrow} avu_2 b = avb^{J-k+1} \in Y_J.$$

By the structure of the elements of Y_J we must have, for some r,

$$v = a^{2r-2}b^{k+2r-2}.$$

We shall show that the only possibility is $r = 1$ and $h = 0$.

Since $a^2b^{J+2} \in L_a(B)$ there must exist w_1, w_2 such that

$$x_1 \dots x_{m-1} \stackrel{*}{\Rightarrow} w_1, \quad x_m \stackrel{*}{\Rightarrow} w_2, \quad w_1 w_2 = a^2 b^{J+2} a.$$

If $w_1 = \epsilon$ then $w_2 = a^2b^{J+2}a$ and

$$B \stackrel{*}{\Rightarrow} av_0 a^2 b^{J+2} a \notin L_a(B).$$

So there must exist $s \ge 1$ such that

$$w_1 = aa^{2s-2}b^{k+2s-2}.$$

But then

$$a^2b^{J+2}a = a^{2s-1}b^{k+2s-2}w_2 \ \text{ implies } \ s = 1,\ k = 0,\ w_2 = ab^{J+2}a.$$

If $av \in L_a(x_1 \dots x_{m-1})$ then

$$B \stackrel{*}{\Rightarrow} avw_2 = a^{2r-1}b^{2r-2}ab^{J+2}a \in L_a(B)$$

and hence, by the structure of $L_a(B)$, $r = 1$. So

$$L_a(x_1 \dots x_{m-1}) = \{a\}.$$

Thus we have shown

$$a^2w \in Y_J = L_a(B) \quad \Leftrightarrow \quad a^2w = aw_2, x_m \stackrel{*}{\Rightarrow} w_2 \quad \Leftrightarrow \quad x_m \stackrel{*}{\Rightarrow} aw$$

so

$$L_a(x_m) = \{aw \mid w \in T^*, x_m \stackrel{*}{\Rightarrow} aw\} = \{aw \mid w \in T^*, a^2w \in Y_J\} = X_{J+1}$$

and again we can take $C = x_m$.

Theorem 5 *There exists a non-left recursive, follow-determined grammar for which there is no corresponding left factored context free grammar.*

Proof Let $L = \{a^{2n-1}b^{2n-1}a, a^{2n}b^{2n} \mid n \geq 1\}$.

$$S\texttt{::=}\,\{aAba,\ aBb\}, \qquad A\texttt{::=}\,\{aBb,\ \epsilon\} \qquad B\texttt{::=}\,\{aAb\}$$

is a follow-determined grammar which is not left recursive and which generates L.

Suppose that $\Gamma = (U, T, S, \mathcal{P})$ is a left factored context free grammar which generates L. Then clearly,

$$L_a(S) = \{aba, a^2b^2, a^3b^3a, a^4b^4, \ldots\} = X_0.$$

So, by Lemma 2(1), there exists a non-terminal C_1 such that

$$L_a(C_1) = \{ab^2, a^2b^3a, a^3b^4, \ldots\} = Y_1.$$

Suppose that there are non-terminals $C_1, \ldots, C_{2n-1}$ such that

$$L_a(C_i) = \begin{cases} Y_j, & \text{if } i = 2j - 1 \\ X_j, & \text{if } i = 2j. \end{cases}$$

Then, by Lemma 2(2) there exists a non-terminal C_{2n} such that $L_a(C_{2n}) = X_n$. Then, by Lemma 2(1) there exists a non-terminal C_{2n+1} such that $L_a(C_{2n+1}) = Y_{n+1}$.

Since the X_i and Y_i are all different we must have $C_r \neq C_s$ if $r \neq s$ and thus Γ has infinitely many distinct non-terminals. So Γ cannot be a context free grammar.

References

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles techniques and tools.* Addison-Wesley, 1986.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1 (Parsing). Prentice-Hall Inc., 1972.

[BB95] Peter T. Breuer and Jonathan P. Bowen. A PREttier Compiler-Compiler: Generating higher-order parsers in C. *Software Practice and Experience*, 25(11):1263–1297, November 1995.

[JS97] Adrian Johnstone and Elizabeth Scott. Generalised recursive descent. Part 1: language design and parsing. Technical Report TR-97-18, Royal Holloway, University of London, Computer Science Department, October 1997.

[Par95] Terence John Parr. *Language Translation Using PCCTS and C++ (A Reference Guide)*. Parr Research Corporation, June 1995.