Compiler Design

LEFT RECURSION AND LEFT FACTORING

LEFT RECURSION: INFINITE LOOPING PROBLEM

A grammar is left-recursive if it has a non-terminal A, such that there is a derivation :

$$A \overset{+}{\Rightarrow} A\alpha, \quad \text{for some } \alpha.$$

Top-Down parsing can't reconcile this type of grammar, since it could consistently make choice which wouldn't allow termination.

$$A \Rightarrow A\alpha \Rightarrow A\alpha\alpha \Rightarrow A\alpha\alpha\alpha \ldots \text{etc.} \quad A \rightarrow A\alpha \mid \beta$$

So we have to convert our left-recursive grammar into an equivalent grammar which is not left-recursive.

The left-recursion may appear in a single step of the derivation (*immediate left-recursion*), or may appear in more than one step of the derivation.

Immediate Left Recursion

- $A \rightarrow A\,\alpha \mid \beta$ where β does not start with A

$\Downarrow$ eliminate immediate left recursion

$A \rightarrow \beta\, A'$ where A' is a new nonterminal

$A' \rightarrow \alpha\, A' \mid \varepsilon$ an equivalent grammar

More General (but still immediate):

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots$$

Transform into:

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots \mid \varepsilon$$

Immediate Left Recursion Elimination: Example

Our Example :

$E \rightarrow E + T \mid T$ → $\begin{cases} E \rightarrow TE' \\ E' \rightarrow + TE' \mid \epsilon \end{cases}$

$T \rightarrow T * F \mid F$ → $\begin{cases} T \rightarrow FT' \\ T' \rightarrow * FT' \mid \epsilon \end{cases}$

$F \rightarrow (E) \mid id$ → $F \rightarrow (E) \mid id$

Left Recursion in More Than One Step

- A grammar cannot be immediately left-recursive, but it still can be left-recursive.
- By just eliminating the immediate left-recursion, we may not get a grammar which is not left-recursive.

Example:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Sd \mid e$

Is A left recursive? Yes.

Is S left recursive? Yes, but not immediate left recursion. $S \Rightarrow Af \Rightarrow Sdf$

Left Recursion in More Than One Step: Elimination

Approach:

Look at the rules for S only (ignoring other rules)... No left recursion.

Look at the rules for A...

Do any of A's rules start with S? Yes.

$A \rightarrow Sd$

Get rid of the S. Substitute in the righthand sides of S.

$A \rightarrow Afd \mid bd$

The modified grammar:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Afd \mid bd \mid e$

Now eliminate immediate left recursion involving A.

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid eA'$

$A' \rightarrow cA' \mid fdA' \mid \varepsilon$

Left Recursion in More Than One Step: Elimination

The Original Grammar:

$S \rightarrow Af \mid b$

$A \rightarrow Ac \mid Sd \mid Be$

$B \rightarrow Ag \mid Sh \mid k$

So Far:

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid BeA'$

$A' \rightarrow cA' \mid fdA' \mid \varepsilon$

Left Recursion in More Than One Step: Elimination

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → Ag | Sh | k

Look at the B rules next; Does any righthand side start with "S"?

Left Recursion in More Than One Step: Elimination

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → Ag | Afh | bh | k

Substitute, using the rules for "S"
Af... | b...

Left Recursion in More Than One Step: Elimination

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → Ag | Afh | bh | k

Does any righthand side start with "A"?

Left Recursion in More Than One Step: Elimination

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'g | BeA'g | Afh | bh | k

Substitute, using the rules for "A"
bdA'... | BeA'...

Left Recursion in More Than One Step: Elimination

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'g | BeA'g | bdA'fh | BeA'fh | bh | k

Substitute, using the rules for "A"
bdA'... | BeA'...

Left Recursion in More Than One Step: Elimination

The Original Grammar:

S → Af | b
A → Ac | Sd | Be
B → Ag | Sh | k

So Far:

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'g | BeA'g | bdA'fh | BeA'fh | bh | k

Finally, eliminate any immediate Left recursion involving "B"

Next Form

S → Af | b
A → bdA' | BeA'
A' → cA' | fdA' | ε
B → bdA'gB' | bdA'fhB' | bhB' | kB'
B' → eA'gB' | eA'fhB' | ε

Left Recursion in More Than One Step: Elimination

The Original Grammar:

$S \to Af \mid b$

$A \to Ac \mid Sd \mid Be \mid C$

$B \to Ag \mid Sh \mid k$

$C \to BkmA \mid AS \mid j$

If there is another nonterminal, then do it next.

So Far:

$S \to Af \mid b$

$A \to bdA' \mid BeA' \mid CA'$

$A' \to cA' \mid fdA' \mid \varepsilon$

$B \to bdA'gB' \mid bdA'fhB' \mid bhB' \mid kB' \mid CA'gB' \mid CA'fhB'$

$B' \to eA'gB' \mid eA'fhB' \mid \varepsilon$

Algorithm for Eliminating Left Recursion

Assume the nonterminals are ordered $A_1, A_2, A_3, \ldots$

(In the example: S, A, B)

<u>for</u> <u>each</u> nonterminal A_i (for i = 1 to N) <u>do</u>

  <u>for</u> <u>each</u> nonterminal A_j (for j = 1 to i-1) <u>do</u>

    Let $A_j \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_N$ be all the rules for A_j

    <u>if</u> there is a rule of the form

      $A_i \rightarrow A_j\alpha$

    <u>then</u> replace it by

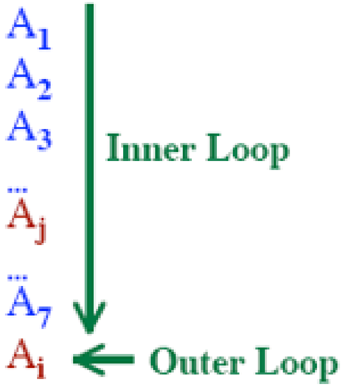
      $A_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \beta_3\alpha \mid \ldots \mid \beta_N\alpha$

    <u>endIf</u>

  <u>endFor</u>

  Eliminate immediate left recursion among the A_i rules

<u>endFor</u>



Left Factoring: Common Prefix Problem

Problem : Uncertain which of 2 rules to choose:

$$\textit{stmt} \rightarrow \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt}$$
$$\quad | \ \textbf{if } \textit{expr} \textbf{ then } \textit{stmt}$$

When do you know which one is valid ?

What's the general form of *stmt* ?

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

α : **if** *expr* **then** *stmt*

β_1: **else** *stmt* $\quad \beta_2$: $\in$

Transform to:

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 \mid \beta_2$

EXAMPLE:

$\textit{stmt} \rightarrow \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \ \textit{rest}$

$\textit{rest} \rightarrow \textbf{else } \textit{stmt} \mid \in$

Left Factoring : Example

$A \rightarrow abB \mid aB \mid cdg \mid cdeB \mid cdfB$

$\Downarrow$

$A \rightarrow aA' \mid cdg \mid cdeB \mid cdfB$

$A' \rightarrow bB \mid B$

$\Downarrow$

$A \rightarrow aA' \mid cdA''$

$A' \rightarrow bB \mid B$

$A'' \rightarrow g \mid eB \mid fB$

Left Factoring : Example

$A \rightarrow ad \mid a \mid ab \mid abc \mid b$

$\Downarrow$

$A \rightarrow aA' \mid b$

$A' \rightarrow d \mid \varepsilon \mid b \mid bc$

$\Downarrow$

$A \rightarrow aA' \mid b$

$A' \rightarrow d \mid \varepsilon \mid bA''$

$A'' \rightarrow \varepsilon \mid c$

The End