CSE-361:Compiler Design

Parsing : Part-II

Predictive Parsing

- The goal is to construct a top-down parser that never backtracks
- Always leftmost derivations
- We must transform a grammar in two ways:
  - eliminate left recursion
  - perform left factoring
- These rules eliminate most common causes for backtracking although they do not guarantee a completely backtrack-free parsing

Left Recursion: Infinite Looping problem

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

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

- **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 :

$E \rightarrow E + T \mid T \longrightarrow \begin{cases} E \rightarrow TE' \\ E' \rightarrow +TE' \mid \in \end{cases}$

$T \rightarrow T * F \mid F \longrightarrow \begin{cases} T \rightarrow FT' \\ T' \rightarrow *FT' \mid \in \end{cases}$

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

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.

$A \rightarrow S\underline{d}$

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

$A \rightarrow A\underline{fd} \mid \underline{bd}$

The modified grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid A\underline{fd} \mid \underline{bd} \mid \underline{e}$

Now eliminate immediate left recursion involving A.

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid \underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \underline{\varepsilon}$

Left Recursion in More Than One Step: Elimination

The Original Grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e}$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

Left Recursion in More Than One Step: Elimination

The Original Grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e}$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

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

Left Recursion in More Than One Step: Elimination

The Original Grammar:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e}$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

$B \rightarrow A\underline{g} \mid A\underline{fh} \mid \underline{bh} \mid \underline{k}$

Substitute, using the rules for "S"

$A\underline{f}... \mid \underline{b}...$

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:

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid BeA'$

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

$B \rightarrow Ag \mid Afh \mid bh \mid k$

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:

$S \rightarrow Af \mid b$

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

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

So Far:

$S \rightarrow Af \mid b$

$A \rightarrow bdA' \mid BeA'$

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

$B \rightarrow bdA'g \mid BeA'g \mid bdA'fh \mid BeA'fh \mid bh \mid k$

Substitute, using the rules for "A"

$bdA' \ldots \mid BeA' \ldots$

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:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow A\underline{c} \mid S\underline{d} \mid B\underline{e} \mid C$

$B \rightarrow A\underline{g} \mid S\underline{h} \mid \underline{k}$

$C \rightarrow B\underline{km}A \mid AS \mid \underline{j}$

If there is another nonterminal, then do it next.

So Far:

$S \rightarrow A\underline{f} \mid \underline{b}$

$A \rightarrow \underline{bd}A' \mid B\underline{e}A' \mid CA'$

$A' \rightarrow \underline{c}A' \mid \underline{fd}A' \mid \varepsilon$

$B \rightarrow \underline{bd}A'\underline{g}B' \mid \underline{bd}A'\underline{fh}B' \mid \underline{bh}B' \mid \underline{k}B' \mid CA'\underline{g}B' \mid CA'\underline{fh}B'$

$B' \rightarrow \underline{e}A'\underline{g}B' \mid \underline{e}A'\underline{fh}B' \mid \varepsilon$

Algorithm for Eliminating Left Recursion

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

(In the example: S, A, B)

for each nonterminal A_i (for i = 1 to N) do

  for each nonterminal A_j (for j = 1 to i-1) do

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

    if there is a rule of the form

      $A_i \rightarrow A_j\alpha$

    then replace it by

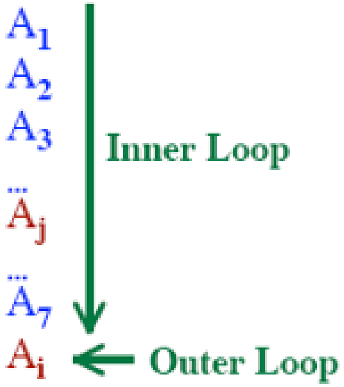
      $A_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \beta_3\alpha \mid \ldots \mid \beta_N\alpha$

    endIf

  endFor

  Eliminate immediate left recursion among the A_i rules

endFor



Left Factoring: Common Prefix Problem

Problem : Uncertain which of 2 rules to choose:

$$stmt \rightarrow \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt$$
$$\mid \textbf{if } expr \textbf{ then } stmt$$

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:

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

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

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

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

$$\Downarrow$$

$A \rightarrow aA' \mid b$

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

$$\Downarrow$$

$A \rightarrow aA' \mid b$

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

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

Reading Materials

- Chapter -4 of your Text book:
  - Compilers: Principles, Techniques, and Tools

The End