Left Recursion

Lecture 6

Section 4.3.3

Rokan Uddin Faruqui

Associate Professor
Dept of Computer Science and Engineering
University of Chittaong, Bangladesh
Email: *rokan@cu.ac.bd*

1. Problems with Recursive Descent
2. Left Recursion
3. Eliminating Left Recursion
4. Advantages of Left Recursion
5. Assignment

Outline

A Problem with Recursive Descent Parsers

- Suppose the grammar were

$$
\begin{aligned}
S &\to AB \mid CD \\
A &\to BC \mid CA \mid \mathbf{a} \\
B &\to CA \mid DB \mid \mathbf{b} \\
C &\to BA \mid AD \mid \mathbf{a} \\
D &\to AC \mid BD \mid \mathbf{b}
\end{aligned}
$$

- How could a top-down parser decide which production for S to use to derive **babbb**?
- Indeed, can **babbb** be derived?

A Problem with Recursive Descent Parsers

- Suppose the grammar were

$$\begin{aligned} S &\to AB \mid CD \\ A &\to BC \mid CA \mid \mathbf{a} \\ B &\to CA \mid DB \mid \mathbf{b} \\ C &\to BA \mid AD \mid \mathbf{a} \\ D &\to AC \mid BD \mid \mathbf{b} \end{aligned}$$

- How could a top-down parser decide which production for S to use to derive **babbb**?
- Indeed, can **babbb** be derived?

Another Problem with Recursive Descent Parsers

```
S() {
 S();
S();
 }
```

- Suppose the grammar were

$$S \to S\ S \mid a$$

- How could the parser decide how many times to use the production $S \to S\ S$ before using the production $S \to a$?

Futile Attempt

Futile Attempt

```
void S()        // Match S -> S S | a
{
    if (token == a)
        match(a);
    else
    {
        S();
        S();
    }
    return;
}
```

Outline

Left Recursion

Definition (Left recursive production)

A production is left recursive if it is of the form

$$A \to A\alpha.$$

for some nonterminal A and some string α.

Definition (Left recursive grammar)

A grammar is left recursive if there is a derivation

$$A \stackrel{+}{\Rightarrow} A\alpha$$

for some nonterminal A and some string α.

Left Recursion

Left Recursion

```
void A()       // Match A -> A



{
    A();
//  Process



    return;
}
```

- Attempting to match the left-recursive production $A \rightarrow A\alpha$.

Left Recursion

$$S \to AB \mid CD$$
$$A \to BC \mid CA \mid \mathbf{a}$$
$$B \to CA \mid DB \mid \mathbf{b}$$
$$C \to BA \mid AD \mid \mathbf{a}$$
$$D \to AC \mid BD \mid \mathbf{b}$$

- Is this grammar left recursive?

Left Recursion

- Recall that in the earlier example, we added the production

$$S' \rightarrow SS' \mid \varepsilon,$$

not the production

$$S' \rightarrow S'S \mid \varepsilon.$$

- Why?
- Are they equivalent as far as the language of the grammar is concerned?

Outline

Eliminating Left Recursion

- Left recursion in a production may be removed by transforming the grammar in the following way.
- Replace

$$A \to A\alpha \mid \beta$$

with

$$A \to \beta A'$$
$$A' \to \alpha A' \mid \varepsilon.$$

where A' is a new nonterminal.

Eliminating Left Recursion

- Under the original productions, a derivation of $\beta\alpha\alpha\alpha$ is

$$
\begin{aligned}
A &\Rightarrow A\alpha \\
&\Rightarrow A\alpha\alpha \\
&\Rightarrow A\alpha\alpha\alpha \\
&\Rightarrow \beta\alpha\alpha\alpha.
\end{aligned}
$$

Eliminating Left Recursion

- Under the new productions, a derivation of $\beta\alpha\alpha\alpha$ is

$$
\begin{aligned}
A &\Rightarrow \beta A' \\
&\Rightarrow \beta\alpha A' \\
&\Rightarrow \beta\alpha\alpha A' \\
&\Rightarrow \beta\alpha\alpha\alpha A' \\
&\Rightarrow \beta\alpha\alpha\alpha.
\end{aligned}
$$

Example

Example (Eliminating Left Recursion)

- Consider the left recursive grammar

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid \mathbf{id}$$

Example

Example (Eliminating Left Recursion)

- Apply the transformation to E:

$$E \to T\ E'$$
$$E' \to +\ T\ E' \mid \varepsilon.$$

- Then apply the transformation to T:

$$T \to F\ T'$$
$$T' \to *\ F\ T' \mid \varepsilon.$$

Example

Example (Eliminating Left Recursion)

- Apply the transformation to E:

$$E \rightarrow T\ E'$$
$$E' \rightarrow +\ T\ E' \mid \varepsilon.$$

- Then apply the transformation to T:

$$T \rightarrow F\ T'$$
$$T' \rightarrow *\ F\ T' \mid \varepsilon.$$

Example

Example (Eliminating Left Recursion)

- Now the grammar is

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +\ T\ E' \mid \varepsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \varepsilon \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$

Eliminating Left Recursion

Eliminating Left Recursion

```
void Eprime()       // Match E' -> + T E'
{
    if (token == PLUS)
    {
        match(PLUS);
        T();
        Eprime();
    }
    return;
}
```

- This is the function for E'.

Outline

Advantages of Left Recursion

- A left recursive grammar is often more intuitive than the transformed grammar.
- A left recursive grammar will match expressions earlier, leading to shallower recursion.
- Bottom-up parsing takes advantage of the benefits of left recursion.

Example

- Consider the simple grammar

$$E \rightarrow E\ +\ E \mid \mathbf{id}$$

- Convert it to

$$E \rightarrow \mathbf{id}\ E'$$
$$E' \rightarrow +\ E\ E' \mid \varepsilon$$

Outline

Assignment

Homework

- 4.2.1, 4.2.2.
- 4.4.1, 4.4.3, 4..4.4
- The grammar

$$R \to R \cup R \mid R\,R \mid R^{*} \mid (\,R\,) \mid \mathbf{a} \mid \mathbf{b}$$

  generates all regular expressions over the alphabet $\{\mathbf{a}, \mathbf{b}\}$.
  - Rewrite the grammar to reflect the precedence rules.
  - Eliminate left recursion.