G52MAL
Machines and Their Languages
Lecture 16

Recursive-Descent Parsing: Elimination of Left Recursion

Henrik Nilsson

University of Nottingham, UK
This Lecture

- The problem of recursive-descent parsing and left recursive grammars.
- Elimination of left recursion.
Left Recursion

Consider: \[ A \rightarrow Aa \mid \epsilon \]
Left Recursion

Consider: \( A \rightarrow Aa | \epsilon \)

Parsing function:

```haskell
parseA :: [Token] -> Maybe [Token]
parseA ts =
    case parseA ts of
        Just ('a' : ts') -> Just ts'
        _ -> Just ts
```
Left Recursion

Consider: $A \rightarrow Aa \mid \epsilon$

Parsing function:

```haskell
parseA :: [Token] -> Maybe [Token]
parseA ts =
  case parseA ts of
    Just ('a' : ts') -> Just ts'
    _ -> Just ts
```

Any problem?
Left Recursion

Consider: $A \rightarrow Aa | \epsilon$

Parsing function:

```haskell
parseA :: [Token] -> Maybe [Token]
parseA ts =
    case parseA ts of
      Just ('a' : ts') -> Just ts'
      _ -> Just ts
```

Any problem?

Would loop! Recursive-descent parsers cannot deal with left-recursive grammars.
A grammar is \textit{left-recursive} if there is some non-terminal $A$ such that $A \Rightarrow A\alpha$. 
Elimination of Left Recursion (1)

- A grammar is **left-recursive** if there is some non-terminal $A$ such that $A \Rightarrow A\alpha$.
- Certain parsing methods **cannot** handle left-recursive grammars.
Elimination of Left Recursion (1)

- A grammar is *left-recursive* if there is some non-terminal $A$ such that $A \Rightarrow A\alpha$.
- Certain parsing methods *cannot* handle left-recursive grammars.
- If we want to use such a parsing method for parsing a language $L = L(G)$ given by a left-recursive grammar $G$, then the grammar first has to be transformed into an *equivalent* grammar $G'$ that is *not* left-recursive.
Recap: Equivalence of Grammars

Two grammars $G_1$ and $G_2$ are equivalent iff $L(G_1) = L(G_2)$.

Example:

$G_1$:  

- $S \rightarrow \epsilon \mid A$
- $A \rightarrow a \mid aA$

$L(G_1) = \{a\}^* = L(G_2)$

$G_2$:  

- $S \rightarrow A$
- $A \rightarrow \epsilon \mid Aa$

(The equivalence of CFGs is in general undecidable.)
We will first consider *immediate* left recursion; i.e., productions of the form

\[ A \rightarrow A\alpha \]

where \( \alpha \) cannot derive \( \epsilon \).
We will first consider *immediate* left recursion; i.e., productions of the form

\[ A \rightarrow A\alpha \]

where \( \alpha \) cannot derive \( \epsilon \).

Key idea: \( A \rightarrow \beta \mid A\alpha \) and \( A \rightarrow \beta(\alpha)^* \) are equivalent.
We will first consider *immediate* left recursion; i.e., productions of the form

\[ A \rightarrow A\alpha \]

where \( \alpha \) cannot derive \( \epsilon \).

Key idea: \( A \rightarrow \beta \mid A\alpha \) and \( A \rightarrow \beta(\alpha)^* \) are equivalent.

The latter can be expressed as:

\[ A \rightarrow \beta A' \]
\[ A' \rightarrow \alpha A' \mid \epsilon \]
Exercise

- The following grammar $G_1$ is immediately left-recursive:

$$A \rightarrow b \mid Aa$$

Draw the derivation tree for $baa$ using $G_1$.

- The following is a non-left-recursive grammar $G'_1$ equivalent to $G_1$:

$$A \rightarrow bA'$$

$$A' \rightarrow aA' \mid \epsilon$$

Draw the derivation tree for $baa$ using $G'_1$. 
Elimination of Left Recursion (3)

For each nonterminal $A$ defined by some left-recursive production, group the productions for $A$

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

such that no $\beta_i$ begins with an $A$. 
Elimination of Left Recursion (3)

For each nonterminal $A$ defined by some left-recursive production, group the productions for $A$

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

such that no $\beta_i$ begins with an $A$.

Then replace the $A$ productions by

$$A \to \beta_1A' \mid \beta_2A' \mid \ldots \mid \beta_nA'$$

$$A' \to \alpha_1A' \mid \alpha_2A' \mid \ldots \mid \alpha_mA' \mid \epsilon$$

Assumption: no $\alpha_i$ derives $\epsilon$. 
Elimination of Left Recursion (4)

Consider the (immediately) left-recursive grammar:

\[
\begin{align*}
S & \rightarrow A \mid B \\
A & \rightarrow ABc \mid AAdd \mid a \mid aa \\
B & \rightarrow Bee \mid b
\end{align*}
\]

Terminal strings derivable from \( B \) include:

\( b, bee, beee, beeeee \)

Terminal strings derivable from \( A \) include:

\( a, aa, aadd, aaadd, aaadddd, abc, aabc, abeec, aabeec, abeeceecbeec, aabeeeeecddbeec \)
Elimination of Left Recursion (5)

Let us do a leftmost derivation of $aabeeeecddbeec$:

$$
\begin{align*}
S & \Rightarrow A \\
& \Rightarrow ABc \\
& \Rightarrow AAddBc \\
& \Rightarrow aAddBc \\
& \Rightarrow aABcddBc \\
& \Rightarrow aaBcddBc \\
& \Rightarrow aaBeecddBc \\
& \Rightarrow aaBeeecdddBc \\
& \Rightarrow aabeeecdddBc \\
& \Rightarrow aabeeecdddBeec \\
& \Rightarrow aabeeecdddbeec
\end{align*}
$$
Elimination of Left Recursion (6)

Here is the grammar again:

\[
\begin{align*}
S & \rightarrow A \mid B \\
A & \rightarrow ABc \mid AAdd \mid a \mid aa \\
B & \rightarrow Bee \mid b
\end{align*}
\]
Elimination of Left Recursion (6)

Here is the grammar again:

\[ S \rightarrow A \mid B \]
\[ A \rightarrow ABc \mid AAdd \mid a \mid aa \]
\[ B \rightarrow Bee \mid b \]

An equivalent right-recursive grammar:

\[ S \rightarrow A \mid B \]
\[ A \rightarrow aA' \mid aaA' \]
\[ A' \rightarrow BcA' \mid AddA' \mid \epsilon \]
\[ B \rightarrow bB' \]
\[ B' \rightarrow eeB' \mid \epsilon \]
Elimination of Left Recursion (7)

Derivation of $aabeeeecddbeec$ in the new grammar:

$$
S' \Rightarrow A \Rightarrow aA' \Rightarrow aAddA' \Rightarrow aaA'ddA'
$$
$$
\Rightarrow aaBcA'ddA'
$$
$$
\Rightarrow aabB'cA'ddA'
$$
$$
\Rightarrow aabbeeB'cA'ddA'
$$
$$
\Rightarrow aabeeeecA'ddA'
$$
$$
\Rightarrow aabeeeecddA'
$$
$$
\Rightarrow aabeeeecddBcA'
$$
$$
\Rightarrow aabeeeecddBbB'cA'
$$
$$
\Rightarrow aabeeeecddbeeB'cA'
$$
$$
\Rightarrow aabeeeecddbeecA' \Rightarrow aabeeeecddbeec
$$
To eliminate *general* left recursion:

- first transform the grammar into an *immediately* left-recursive grammar through systematic substitution
- then proceed as before.
Substitution

- An occurrence of a non-terminal in a right-hand side may be replaced by the right-hand sides of the productions for that non-terminal if done in all possible ways.
- All productions for non-terminals that, as a result, cannot be reached from the start symbol, can be eliminated.

(See e.g. Aho, Sethi, and Ullman (1986) for details.)
General Left Recursion (2)

For example, the generally left-recursive grammar

\[
\begin{align*}
A & \rightarrow Ba \\
B & \rightarrow Ab \mid Ac \mid \epsilon
\end{align*}
\]

is first transformed into the immediately left-recursive grammar

\[
\begin{align*}
A & \rightarrow Aba \\
A & \rightarrow Aca \\
A & \rightarrow \epsilon
\end{align*}
\]
Exercise

Transform the following generally left-recursive grammar

\[ A \rightarrow B a B \]
\[ B \rightarrow C b \mid \epsilon \]
\[ C \rightarrow A b \mid A c \]

into an equivalent immediately left-recursive grammar.

Then eliminate the left recursion.
Solution (1)

First:

\[ A \rightarrow BaB \]
\[ B \rightarrow Abb \mid Acb \mid \epsilon \]

Then:

\[ A \rightarrow AbbaB \mid AcbaB \mid aB \]
\[ B \rightarrow Abb \mid Acb \mid \epsilon \]

Or, eliminating \( B \) completely:

\[ A \rightarrow AbbaAbb \mid AcbaAbb \mid aAbb \]
\[ \quad \mid AbbaAcb \mid AcbaAcb \mid aAcb \]
\[ \quad \mid Abba \mid Acba \mid a \]
Solution (2)

Let’s go with the smaller version (fewer productions):

\[
A \rightarrow AbbaB \mid AcbaB \mid aB \\
B \rightarrow Abb \mid Acb \mid \epsilon
\]

Only productions for \(A\) are immediately left-recursive. Applying the elimination transformation:

\[
A \rightarrow aBA' \\
A' \rightarrow bbaBA' \mid cbaBA' \mid \epsilon \\
B \rightarrow Abb \mid Acb \mid \epsilon
\]

Note: \(A\) appears to the left in \(B\)-productions; yet grammar no longer left-recursive. Why?