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: \( A \to Aa | \epsilon \)

Parsing function:

\[
\text{parseA} :: \text{[Token]} \rightarrow \text{Maybe \text{[Token]}}
\]

\[
\text{parseA ts} =
\begin{cases}
  \text{case parseA ts of} & \\
  \text{Just (a : ts')} \rightarrow \text{Just ts'} & \\
  \_ & \rightarrow \text{Just ts}
\end{cases}
\]

Any problem?
Would loop! Recursive-descent parsers cannot deal with 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:

$G_1$: 
- $S \rightarrow \epsilon \mid A$
- $A \rightarrow a \mid aA$

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

(The equivalence of CFGs is in general undecidable.)

Elimination of Left Recursion (2)

- 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 $A$

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

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

Then replace the $A$ productions by

$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$
$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$

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

Consider the (immediately) left-recursive grammar:

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

Terminal strings derivable from \( B \) include:

\( b, bee, beeee, beeeeee \)

Terminal strings derivable from \( A \) include:

\( a, aa, aadd, aaadd, aaadddd, abc, aabc, abeec, aabeec, abeecbeec, aabeeecddbeec \)

Elimination of Left Recursion (5)

Let us do a leftmost derivation of \( aabeeecddbeec \):

\[ S \Rightarrow A \\
\Rightarrow ABc \\
\Rightarrow AAddBc \\
\Rightarrow aAddBc \\
\Rightarrow aABcddBc \\
\Rightarrow aaBcddBc \\
\Rightarrow aaBeeecddBc \\
\Rightarrow aaBeeecddBc \\
\Rightarrow aabeeecddBee \\
\Rightarrow aabeeecddbeec \]

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:

\[ S \rightarrow A \mid B \\
A \rightarrow aA' \mid aaA' \\
B \rightarrow bB' \mid \epsilon \\
A' \rightarrow BcA' \mid AddA' \mid \epsilon \\
B' \rightarrow eeB' \mid \epsilon \]

Elimination of Left Recursion (7)

Derivation of \( aabeeecddbeec \) in the new grammar:

\[ S \Rightarrow A \Rightarrow aA' \Rightarrow aAddA' \Rightarrow aaA'ddA' \\
\Rightarrow aaBcA'ddA' \\
\Rightarrow aabB'cA'ddA' \\
\Rightarrow aabeeB'cA'ddA' \\
\Rightarrow aabeeecB'cA'ddA' \\
\Rightarrow aabeeecddA' \\
\Rightarrow aabeeecddBcA' \\
\Rightarrow aabeeecddBbB'cA' \\
\Rightarrow aabeeecddbeeB'cA' \\
\Rightarrow aabeeecddbeecA' \Rightarrow aabeeecddbeec \]

G52MAL 2012/13: Lecture 14 – p.9/17

G52MAL 2012/13: Lecture 14 – p.10/17

G52MAL 2012/13: Lecture 14 – p.11/17

G52MAL 2012/13: Lecture 14 – p.12/17
General Left Recursion (1)

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
\[
A \to Ba \\
B \to Ab | Ac | \epsilon
\]
is first transformed into the immediately left-recursive grammar
\[
A \to Aba \\
A \to Aca \\
A \to a
\]

Exercise

Transform the following generally left-recursive grammar
\[
A \to BaB \\
B \to Cb | \epsilon \\
C \to Ab | Ac
\]
into an equivalent immediately left-recursive grammar.
Solution

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:

\[ A \rightarrow AbbaAbb \mid AcbaAbb \mid aAbb \]
\[ \mid AbbaAcb \mid AcbaAcb \mid aAcb \]
\[ \mid Abba \mid Acba \mid a \]