Lecture 8: Recursive-descent parsing

- Recursive-descent formalized
- FIRST sets
- LL(1) condition
- Transformations to LL(1) form

Grammars for expressions - a difficult case

(Next week: LR(1) parsing, ocamlyacc)

Top-down parsing

- For each non-terminal with productions:

\[ A \rightarrow \vec{X} \mid \vec{Y} \mid \ldots \mid \vec{Z} \]

define parseA:

\[
\begin{align*}
\text{parseA toklis} & = \text{choose production based on hd toklis:} \\
& \quad \text{if } A \rightarrow \vec{X} \text{ chosen: handle } \vec{X} \\
& \quad \quad \text{else if } A \rightarrow \vec{Y} \text{ chosen: handle } \vec{Y}, \\
& \quad \quad \quad \text{else if } \text{etc.} \\
& \quad \text{handle } X_1 X_2 \ldots X_n : \text{handle } X_1; \text{ handle } X_2; \ldots; \text{ handle } X_n \\
& \quad \text{where handle } t : \text{if } \text{hd toklis} = t \\
& \quad \quad \text{then remove } t \text{ and continue} \\
& \quad \quad \quad \text{else error} \\
& \quad \text{handle } B : \text{parseB toklis}
\end{align*}
\]
“choose production based on hd toklis”

- Need to formalize some things...
- Recall definitions of: $\Rightarrow$, $\Rightarrow^+$, $\Rightarrow^*$, sentential form, sentence.
- Define: $\vec{X}$ is nullable if it can derive $\epsilon$.

“choose production based on hd toklis” (cont.)

- Define: $\text{FIRST}(\vec{X}) = \{ t \in T | \vec{X} \Rightarrow^* t\alpha \text{ for some } \alpha \} \cup \{ \cdot | \vec{X} \text{ nullable} \}$.
- Define: $\text{FOLLOW}(A) = \{ t \in T | \exists \text{ a sentential form } \alpha At\beta \}$.

There are well-known algorithms for calculating FIRST and FOLLOW sets, but we will consider only simple cases where they can be calculated by inspection.
Top-down parsing revisited

Ignoring $\epsilon$-productions for the moment, we can be more precise in constructing parser.

$A \rightarrow \vec{X} | \vec{Y} | \ldots | \vec{Z}$

parseA toklis = let t = hd toklis in
   if t $\in$ FIRST($\vec{X}$) then handle $\vec{X}$
   else if t $\in$ FIRST($\vec{Y}$) then handle $\vec{Y}$
   ...
   else if t $\in$ FIRST($\vec{Z}$) then handle $\vec{Z}$
   else error

handle $X_1, X_2, \ldots, X_n$ : handle $X_1$; handle $X_2$; ...; handle $X_n$

handle t : if hd toklis = t
   then remove t
   else error

handle B : parseB toklis

“choose production based on hd toklis” (cont.)

- Define: G is left-recursive if $\exists A : A \Rightarrow^+ A\alpha$ for some $\alpha$.
- Define: G is LL(1) if
  1. G is not left-recursive, and
  2. For all non-terminals $A$, if the productions of $A$ are $A \rightarrow \alpha_1 | \ldots | \alpha_n$:
     (a) The sets FIRST($\alpha_1$), ..., FIRST($\alpha_n$) are pairwise disjoint.
         ($\forall i, j. i \neq j \Rightarrow \text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$.)
     (b) If $A$ is nullable, then suppose $\alpha_i$ is the unique right-hand side such that $\bullet \in \text{FIRST}(\alpha_i)$. Then, for all $j \neq i$, \text{FIRST}(\alpha_j) \cap \text{FOLLOW}(A) = \emptyset$. 

Top-down parsing re-revisited

The full story (with $\epsilon$-productions): If $G$ is LL(1), construct $\text{parseA}$ for non-terminal $A$ with productions

$$A \rightarrow \vec{X} \mid \vec{Y} \mid \ldots \mid \vec{Z}$$

$$\text{parseA toklis} = \text{let } t = \text{hd toklis in}$$

$$\text{if } t \in \text{FIRST}(\vec{X}) \text{ then handle } \vec{X}$$

$$\text{else if } t \in \text{FIRST}(\vec{Y}) \text{ then handle } \vec{Y}$$

$$\ldots \text{else if } t \in \text{FIRST}(\vec{Z}) \text{ or } (\bullet \in \text{FIRST}(\vec{Z}) \text{ and } t \in \text{FOLLOW}(A))$$

$$\text{ then handle } \vec{Z} \text{ (the unique nullable right-hand side of } A, \text{ if any)}$$

$$\text{else error}$$

$$\text{handle } X_1, X_2, \ldots, X_n : \text{handle } X_1; \text{ handle } X_2; \ldots; \text{ handle } X_n$$

$$\text{handle } t : \text{if hd toklis } = t$$

$$\text{then remove } t$$

$$\text{else error}$$

$$\text{handle } B : \text{parseB toklis}$$

Transformation to LL(1)

- **Left refactoring:**
  $$A \rightarrow \alpha \beta \mid \alpha \gamma$$
  $$\Rightarrow A \rightarrow \alpha B$$
  $$B \rightarrow \beta \mid \gamma$$

- **Left-recursion removal:**
  $$A \rightarrow A \alpha \mid \beta$$
  $$\Rightarrow A \rightarrow \beta B$$
  $$B \rightarrow \epsilon \mid \alpha B$$
Example

- Consider non-LL(1) grammar 3 from the previous class:

  \[
  A \rightarrow \text{id} \mid '(' B ')' \\
  B \rightarrow A \mid A \ ' + ' B
  \]

- Grammar 3 transformed to LL(1) form:

  \[
  A \rightarrow \text{id} \mid '(' B ')' \\
  B \rightarrow A \ C \\
  C \rightarrow \ '+' B \mid \epsilon
  \]

Ambiguity

- No test for ambiguity
- Recursive descent and LR(1) parsing not applicable to ambiguous grammar (possible to “cheat” with LR parser - will see how next week)
Expression grammars

- Expressions are challenging for several reasons:
  - Grammar should enforce precedence, if possible
  - Grammar should enforce associativity, if possible
  - Grammar shouldn’t be ambiguous
  - Should be easy to construct abstract syntax tree

- Especially hard to write LL(1) parser for expressions. Not so hard for LR(1).
Enforcing associativity

Some expression grammars

\[ G_A: \quad E \rightarrow id \mid E \cdot E \mid E \star E \]

\[ G_B: \quad E \rightarrow id \mid id - E \mid id \star E \]

\[ G_C: \quad E \rightarrow id \mid E - id \mid E \star id \]

\[ G_D: \quad E \rightarrow T \cdot E \mid T \]
\[ \quad T \rightarrow id \mid id \star T \]

\[ G_E: \quad E \rightarrow E - T \mid T \]
\[ \quad T \rightarrow id \mid T \star id \]

\[ G_F: \quad E \rightarrow T \cdot E' \]
\[ \quad E' \rightarrow \epsilon \mid - E \]
\[ \quad T \rightarrow id \cdot T' \]
\[ \quad T' \rightarrow \epsilon \mid \star T \]
\[ G_A: \ E \rightarrow \text{id} \mid \ E - \ E \mid \ E \ast \ E \]

\[ G_B: \ E \rightarrow \text{id} \mid \text{id} - \ E \mid \text{id} \ast \ E \]

- \( x - y \ast z \)
- \( x - y - z \)
- \( x - y \ast z \)
- \( x - y \ast z - w \)
- \( x \ast y - z \)
- \( x \ast y \ast z \)
\[ G_C : E \rightarrow id \mid E - id \mid E * id \]

\[ G_D : E \rightarrow T - E \mid T \]

\[ T \rightarrow id \mid id * T \]

- \[ x - y * z \]
- \[ x - y * z - w \]

- \[ x * y - z \]

- \[ x * y - z \]
\[ G_E: \ E \rightarrow E - T \mid T \]
\[ T \rightarrow id \mid T * id \]

- \( x - y * z \)
- \( x * y - z \)
- \( x - y - z \)

\[ G_F: \ E \rightarrow T E' \]
\[ E' \rightarrow \epsilon \mid - E \]
\[ T \rightarrow id T' \]
\[ T' \rightarrow \epsilon \mid * T \]