Class Information

REMINDERS

• Deadline extension for second homework: Monday, October 2, 11:59pm. No late submissions.

• Don’t forget to work on your C and Linux skills (ilab).
Top-Down Parsing - LL(1)

Basic Idea:

- The parse tree is constructed from the root, expanding **non-terminal** nodes on the tree’s frontier following a left-most derivation.

- The input program is read from left to right, and input tokens are read (consumed) as the program is parsed.

- The next **non-terminal** symbol is replaced by one of its rules. The particular choice has to be unique, and uses parts of the input (partially parsed program), for instance the first **token** of the remaining input.
Top-Down Parsing - LL(1) (cont.)

How can we parse (automatically construct a left-most derivation) an input string, for example $a a a b b b$, using a PDA (push-down automaton) and only the first symbol of the remaining input?

Example:

$S ::= a \ S \ b \ | \ \epsilon$

INPUT: $a a a b b b \ \text{eof}$
Predictive Parsing

Basic idea:

For any two productions $A ::= \alpha \mid \beta$ with $\alpha \in (T \cup N)^*$ and $\beta \in (T \cup N)^*$, we would like a distinct way of choosing the correct production to expand.

For $\alpha \in (T \cup N)^*$, define $\text{FIRST}(\alpha)$ as the set of tokens that appear as the first token in some string derived from $\alpha$.

That is

$a \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* a \gamma$ for some $\gamma \in (T \cup N)^*$ and $a$ is a token ($x \in T$), and $\epsilon \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* \epsilon$

For a non-terminal $A$, define $\text{FOLLOW}(A)$ as the set

$a \in \text{FOLLOW}(A)$ iff $S \Rightarrow^* \alpha A a \gamma$ for some $\alpha, \gamma \in (T \cup N)^*$, $a \in T$, and $S$ the start symbol.

Thus, a non-terminal’s FOLLOW set specifies the tokens that can legally appear after it.

FOLLOW sets are not defined for terminal symbols.

$\text{FIRST}$ and $\text{FOLLOW}$ sets can be constructed automatically.
Predictive Parsing (cont.)

Key Property:
Whenever two productions $A ::= \alpha$ and $A ::= \beta$ both appear in the grammar, we would like

- $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$, and
- if $\alpha \Rightarrow^* \epsilon$ then $\text{FIRST}(\beta) \cap \text{FOLLOW}(A) = \emptyset$
- Analogue case for $\beta \Rightarrow^* \epsilon$. Note: due to first condition, at most one of $\alpha$ or $\beta$ can derive $\epsilon$.

This would allow the parser to make a correct choice with a lookahead of only one symbol!
LL(1) Grammar

Define $FIRST^+(\delta)$ for rule $A ::= \delta$

- $FIRST(\delta) - \{\epsilon\} \cup \text{Follow}(A)$, if $\epsilon \in FIRST(\delta)$
- $FIRST(\delta)$ otherwise

A grammar is LL(1) iff

$(A ::= \alpha$ and $A ::= \beta)$ implies

$FIRST^+(\alpha) \cap FIRST^+(\beta) = \emptyset$
Back to Our Example

\[
S ::= a \, S \, b \mid \epsilon
\]

\[
FIRST(aSb) = \{a\}
\]

\[
FIRST(\epsilon) = \{\epsilon\}
\]

\[
FOLLOW(S) = \{\text{eof, b}\}
\]

\[
\begin{align*}
FIRST^+(aSb) &= \{a\} \\
FIRST^+(\epsilon) &= (FIRST(\epsilon) - \{\epsilon\}) \cup FOLLOW(S) = \\
&= \{\text{eof, b}\}
\end{align*}
\]

Is the grammar LL(1)?
Table-Driven LL(1) Parser

LL(1) parse table

Example:
$$S ::= a\, S\, b \mid \epsilon$$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>eof</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>aSb</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>error</td>
</tr>
</tbody>
</table>

How to parse input a a a b b b ?
Table-driven predictive parsing algorithm

Input: a string \( w \) and a parsing table \( M \) for \( G \)

\[
\begin{align*}
\text{push } & \text{eof} \\
\text{push } & \text{Start Symbol} \\
token & \leftarrow \text{next\_token()} \\
X & \leftarrow \text{top\_of\_stack} \\
\text{repeat} & \\
\quad & \text{if } X \text{ is a terminal then} \\
\quad & \quad \text{if } X = \text{token then} \\
\quad & \quad \quad \text{pop } X \\
\quad & \quad \quad \text{token } \leftarrow \text{next\_token()} \\
\quad & \quad \quad \text{else error()} \\
\quad & \quad \text{else /* } X \text{ is a non-terminal */} \\
\quad & \quad \quad \text{if } M[X,\text{token}] = X \rightarrow Y_1Y_2\cdots Y_k \text{ then} \\
\quad & \quad \quad \quad \text{pop } X \\
\quad & \quad \quad \quad \text{push } Y_k, Y_{k-1}, \cdots, Y_1 \\
\quad & \quad \quad \quad \text{else error()} \\
\quad & \quad X \leftarrow \text{top\_of\_stack} \\
\text{until } & X = \text{eof} \\
\quad & \text{if token } \neq \text{eof} \text{ then error()} \\
\end{align*}
\]

See also Aho, Lam, Sethi, and Ullman, Figure 4.20, page 227
Next Lecture

Syntax-directed translation

Four examples: Interpreter, compiler, type checker, static performance predictor

Things to do:
Start programming in C. Check out the web for tutorials.