Class Information

REMINDERS

• **Deadline extension** for second homework:
  Tuesday, September 29, before class.
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:**
$$S ::= a \ S \mid b \ S \mid \epsilon$$

INPUT: $a \ a \ a \ b \ b \ b$ eof

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

INPUT: $a \ a \ a \ b \ b \ b$ 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

$x \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* x\gamma$ for some $\gamma \in (T \cup N)^*$ and $x$ 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 of terminals that can appear immediately to the right of $A$ in some sentential form.

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

A terminal symbol has no FOLLOW set.

$\text{FIRST}$ 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

- $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$, and
- if $\alpha \Rightarrow^* \epsilon$ then $FIRST(\beta) \cap FOLLOW(A) = \emptyset$
- 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

\[
(A ::= \alpha \text{ and } A ::= \beta) \text{ implies } FIRST^+(\alpha) \cap FIRST^+(\beta) = \emptyset
\]
Back to Our Example

\[ S ::= aSb \mid \epsilon \]

\[ FIRST(aSb) = \{a\} \]
\[ FIRST(\epsilon) = \{\epsilon\} \]
\[ FOLLOW(S) = \{eof, b\} \]

\[ FIRST^+(aSb) = \{a\} \]
\[ FIRST^+(\epsilon) = (FIRST(\epsilon) - \{\epsilon\}) \cup FOLLOW(S) = \{eof, b\} \]

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

LL(1) parse table

Example:
\[ S ::= aSb \mid \epsilon \]

<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 \texttt{a a a b b b}?
Table-driven predictive parsing algorithm

Input: a string $w$ and a parsing table $M$ for $G$

push $\text{eof}$
push $\text{Start Symbol}$
token $\leftarrow$ next_token()

$X \leftarrow$ top-of-stack
repeat
  if $X$ is a terminal then
    if $X$ = token then
      pop $X$
      token $\leftarrow$ next_token()
    else error()
  else /* $X$ is a non-terminal */
    if $M[X,\text{token}] = X \rightarrow Y_1Y_2\cdots Y_k$ then
      pop $X$
push $Y_k,Y_{k-1},\cdots,Y_1$
    else error()

  $X \leftarrow$ top-of-stack
until $X$ = $\text{eof}$

if token $\neq$ $\text{eof}$ then error()

See also Aho, Lam, Sethi, and Ullman, Figure 4.20, page 227
Predictive Parsing

Now, a predictive parser looks like:

Rather than writing code, we build tables.

Building tables can be automated!
Generating a Table-Driven Parser

A parser generator system often looks like:

This is true for both top down and bottom up parsers.

**LL(1):** left to right, leftmost derivation, lookahead(1)

**LR(1):** left to right, reverse rightmost derivation, lookahead(1)
Recursive descent parser

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.