Recitation #5

Administrative
- Any questions on Assignment 2?
- Exam is on 02/22 at normal class times
- Worksheet → solution from lab 4 updated with replacement of non-immediate left recursion and one erroneous state transition for precedence grammar

Exam Review
- More theoretical, less code work
- Look at Dr. Hughes’ promises on his website

Assignment 3
- Preparation for Midterm #1

FIRST and FOLLOW set
- Important for both top-down as well as bottom-up parsers
- Allow to choose which production to apply, based on next input symbol
- \textit{FIRST}(\alpha), where \alpha is any string of grammar symbols, to be the set of terminals that begin strings derived from \alpha
  - If \alpha \Rightarrow^* \varepsilon, then \varepsilon is part of \textit{FIRST}(\alpha)
  - E.g. \textit{A} \Rightarrow^* \textit{c} \gamma, so \textit{c} is in \textit{FIRST}(\textit{A})
  - To compute, see page 293 in Knights Book
  - Add to \textit{FIRST}(X_1X_2\ldots X_n) all non-\varepsilon symbols of \textit{FIRST}(X_1). Also add the non-\varepsilon symbols of \textit{FIRST}(X_2), if \varepsilon is in \textit{FIRST}(X_1); the non-\varepsilon symbols of \textit{FIRST}(X_3), if \varepsilon is in \textit{FIRST}(X_1) and \textit{FIRST}(X_2); and so on. Finally, add \varepsilon to \textit{FIRST}(X_1X_2\ldots X_n) if, for all \textit{i}, \varepsilon is in \textit{FIRST}(X_\textit{i})
  - If \textit{t} is a terminal, \textit{FIRST}(\textit{t}) = \{\textit{t}\}
- \textit{FOLLOW}(\textit{A}), for nonterminal \textit{A}, to be the set of terminals \textit{a} that can appear immediately to the right of \textit{A} in some sentential form
  - i.e. there exists a derivation of the form \textit{S} \Rightarrow^* \alpha \textit{Aa} \beta, for some \alpha and \beta
  - There may have been symbols between \textit{A} and \textit{a} at some time during derivation, but they derived to \varepsilon
  - If \textit{A} can be the rightmost symbol in some form, then \textit{S} is in \textit{FOLLOW}(\textit{A}) (\textit{S} is a special “endmarker” symbol that is assumed not to be a symbol of any grammar)
  - If \textit{t} is a terminal, \textit{FOLLOW}(\textit{t}) = \emptyset
  - Steps:
1. Place $ in FOLLOW(S)$, where $S$ is the start symbol, and $S$ is the input right endmarker.
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST($\beta$) except $\varepsilon$ is in FOLLOW($B$)
3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$, where FIRST ($\beta$) contains $\varepsilon$, then everything in FOLLOW($A$) is in FOLLOW($B$).

• Example: Grammar $G = (\{E, E', T, T', F\}, \{, \}, \{, \}, id, E, P)$

$E \rightarrow TE'$
$E' \rightarrow +TE' | \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' | \varepsilon$
$F \rightarrow (E) | id$

Then the FIRST and FOLLOW sets (listed in the order they can be most readily assigned) are:

FIRST(F) = {\{, id\}
FIRST(T) = {\{, id\}
FIRST(E) = {\{, id\}
FIRST(E') = {+, \varepsilon\}
FIRST(T') = {*, \varepsilon\}

Because $E$ is the start symbol of the grammar and production 5, we can deduce with the help of rule (1):

FOLLOW(E) = {\{, $\}$

For $E'$, notice that it only appears on the right side of $E$ productions. Following rule (2):

FOLLOW(E') = FOLLOW(E) = {\{, $\}$

Considering $T$, it is part of production 2 and rule (2) applies, so:

FOLLOW(T) = FIRST(E') \{\varepsilon\} = {+}

In addition, in productions 1 and 2, $T$ is followed by $E'$ and FIRST($E'$) contains $\varepsilon$, so following rule (3):

FOLLOW(T) = FIRST(E') \{\varepsilon\} \cup FOLLOW(E') = {+, \{, $\}
For $T'$, see production 3 and rule (3), so:

$$\text{FOLLOW}(T') = \text{FOLLOW}(T) = \{+, \), $\}$$

In the case of $F$, in productions 3 and 4, it is followed by $T'$. Rule (2) dictates, that:

$$\text{FOLLOW}(F) = \text{FIRST}(T') \setminus \{\epsilon\} = \{\ast\}$$

In addition, FIRST($T'$) contains $\epsilon$, so following rule (3):

$$\text{FOLLOW}(F) = \text{FIRST}(T') \setminus \{\epsilon\} \cup \text{FOLLOW}(T) \cup \text{FOLLOW}(T') = \{\ast, +, \), $\}$$

**LL(1) Grammars**

- Predictive parsers can be constructed for LL(1) grammars, since the proper production to apply for a nonterminal can be selected by looking only at the current input symbol
- First “L” for scanning input from left-to-right, second “L” for producing a leftmost derivation, and the “1” for using one input symbol of lookahead
- LL(1) grammars cover most programming constructs
- LL(1) parsers operate in linear time!
- **No left-recursive or ambiguous** grammar can be LL(1).
- Can’t be left-recursive, because it could go in infinite loop
- Can’t be ambiguous, because parser wouldn’t know what to choose
- A grammar $G$ is LL(1) if and only if whenever $A \rightarrow \alpha | \beta$ are two distinct productions of $G$:
  - FIRST($\alpha$) and FIRST($\beta$) are disjoint sets, i.e. FIRST($\alpha$) $\cap$ FIRST($\beta$) $= \emptyset$
  - If $\epsilon$ is in FIRST($\beta$), then FIRST($\alpha$) and FOLLOW($A$) are disjoint sets
  - If $\epsilon$ is in FIRST($\alpha$), then FIRST($\beta$) and FOLLOW($A$) are disjoint sets.
  - Note that due to the first condition, at most one of $\alpha$ or $\beta$ can derive $\epsilon$
- Flow-control statements, e.g. if, while, for, generally satisfy LL(1) constraints
- Not all grammars are LL(1), eg. $S \rightarrow aS | a$ is not LL(1) because FIRST($aS$) =FIRST($a$) = $\{a\}$ → common prefixes violate LL(1) property, that’s why we need left factoring

**LL(1) parsing table (predictive parsing table)**

- **Idea:** The production $A \rightarrow \alpha$ is chosen if the next input symbol $a$ is in $\text{FIRST}(\alpha)$. The only complication occurs when $\alpha = \epsilon$ or, more generally, $\alpha \Rightarrow^* \epsilon$. In this case, we should again choose $A \rightarrow \alpha$, if the current input symbol is in FOLLOW($A$), or if the $\$$ on the input has been reached and $\$$ is in FOLLOW($A$).
- **Algorithm:** For each production $A \rightarrow \alpha$ of the grammar, do the following:
  1. For each terminal $a$ in $\text{FIRST}(A)$, add $A \rightarrow \alpha$ to $M[A,a]$. 

3
2. If \( \epsilon \) is in FIRST(\( \alpha \)), then for each terminal \( b \) in FOLLOW(\( A \)), add \( A \rightarrow \alpha \) to \( M[A, b] \). If \( \epsilon \) is in FIRST(\( \alpha \)) and \( \$ \) is in FOLLOW(\( A \)), add \( A \rightarrow \alpha \) to \( M[A, \$] \) as well.

- If, after performing the above, there is no production at all in \( M[A, a] \), then set \( M[A, a] \) to “error” (normally represented by empty entry in the table).
- Example: (from grammar above)

<table>
<thead>
<tr>
<th>Non-Terminals</th>
<th>Input Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>( E \rightarrow TE' )</td>
</tr>
<tr>
<td>( E' )</td>
<td>( E' \rightarrow +TE' )</td>
</tr>
<tr>
<td>( T )</td>
<td>( T \rightarrow FT' )</td>
</tr>
<tr>
<td>( T' )</td>
<td>( T' \rightarrow \epsilon )</td>
</tr>
<tr>
<td>( F )</td>
<td>( F \rightarrow id )</td>
</tr>
</tbody>
</table>

**Extended Backus-Naur Form (EBNF)**

- Just one way of writing a grammar in a “standardized” format.
- I showed you the Wikipedia standard in Recitation #1.
- Dr. Hughes’ format:
  - non-terminal ::= rhs
  - rhs can include quoted terminals (e.g. “1” “AND” ‘1’ ‘.’), non-terminals, designated keywords (e.g. PROGRAM), and special symbols
  - [s] \rightarrow \text{optionally include string } s
  - {s} \rightarrow \text{repeat string } s 0 \text{ or more times}
  - (...) \rightarrow \text{grouping}
- How to convert grammar to EBNF?
  - First, remove left recursion and do left factoring
  - Use quotes for terminals
  - Replace \( \rightarrow \) with ::=
  - Replace \( \epsilon \) with optional expressions
  - Reduce grammar until there is a single production rule for each non-terminal symbol, using optional expressions where necessary
Stack-based non-recursive predictive parsing

- Maintain a stack explicitly, rather than implicitly via recursive calls
- Mimics a leftmost derivation

**Algorithm 4.34**: Table-driven predictive parsing.

**INPUT**: A string $w$ and a parsing table $M$ for grammar $G$.

**OUTPUT**: If $w$ is in $L(G)$, a leftmost derivation of $w$; otherwise, an error indication.

**METHOD**: Initially, the parser is in a configuration with $w$ in the input buffer and the start symbol $S$ of $G$ on top of the stack, above $\$. The program in Fig. 4.20 uses the predictive parsing table $M$ to produce a predictive parse for the input. □

```plaintext
set ip to point to the first symbol of $w$;
set $X$ to the top stack symbol;
while ($X \neq \$) { /* stack is not empty */
    if ( $X$ is a ) pop the stack and advance ip;
    else if ( $X$ is a terminal ) error();
    else if ( $M[X, a]$ is an error entry ) error();
    else if ( $M[X, a] = X \rightarrow Y_1Y_2 \cdots Y_k$ ) {
        output the production $X \rightarrow Y_1Y_2 \cdots Y_k$;
        pop the stack;
        push $Y_k, Y_{k-1}, \ldots, Y_1$ onto the stack, with $Y_1$ on top;
    }
    set $X$ to the top stack symbol;
}
```

**Example 4.35**: Consider grammar (4.28); we have already seen its the parsing table in Fig. 4.17. On input $id + id * id$, the nonrecursive predictive parser of Algorithm 4.34 makes the sequence of moves in Fig. 4.21. These moves correspond to a leftmost derivation (see Fig. 4.12 for the full derivation):

$$E \rightarrow TE' \rightarrow TE' \rightarrow FT'E' \rightarrow id T'E' \rightarrow id E' \rightarrow id + T'E' \rightarrow \cdots$$

<table>
<thead>
<tr>
<th>MATCHED</th>
<th>STACK</th>
<th>INPUT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>id + id + id$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$TE$</td>
<td>id + id + id$</td>
<td></td>
<td>output $E \rightarrow TE'$</td>
</tr>
<tr>
<td>$FT'E$</td>
<td>id + id + id$</td>
<td></td>
<td>output $T \rightarrow FT'$</td>
</tr>
<tr>
<td>id $T'E$</td>
<td>id + id + id$</td>
<td></td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id $TE$</td>
<td>id + id + id$</td>
<td></td>
<td>match id</td>
</tr>
<tr>
<td>id $E$</td>
<td>id + id + id$</td>
<td></td>
<td>output $T' \rightarrow \epsilon$</td>
</tr>
<tr>
<td>id $+TE$</td>
<td>id + id + id$</td>
<td></td>
<td>output $E' \rightarrow + TE'$</td>
</tr>
<tr>
<td>id $+TE'$</td>
<td>id + id + id$</td>
<td></td>
<td>match +</td>
</tr>
<tr>
<td>id $+FT'E$</td>
<td>id + id + id$</td>
<td></td>
<td>output $T \rightarrow FT'$</td>
</tr>
<tr>
<td>id $+id T'E$</td>
<td>id + id + id$</td>
<td></td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$T'E$</td>
<td>id + id + id$</td>
<td>match id</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$*FT'E$</td>
<td>id + id + id$</td>
<td>output $T' \rightarrow * FT'$</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$FT'E$</td>
<td>id + id + id$</td>
<td>match $*$</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$id T'E$</td>
<td>id + id + id$</td>
<td>output $F \rightarrow id$</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$id E$</td>
<td>id + id + id$</td>
<td>match id</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$id T'E$</td>
<td>id + id + id$</td>
<td>output $T' \rightarrow \epsilon$</td>
</tr>
<tr>
<td>id $+id$</td>
<td>$id$</td>
<td>id + id + id$</td>
<td>output $E' \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

5