02/17/2010

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
- **FIRST(α)**, where α is any string of grammar symbols, to be the **set of terminals** that begin strings derived from α
    - If $\alpha \Rightarrow^* \varepsilon$, then ε is part of FIRST(α)
    - E.g. $A \Rightarrow^* c\gamma$, so c is in FIRST(A)
    - To compute, see page 293 in Knights Book
    - Add to $\text{FIRST}(X_1 X_2 \ldots X_n)$ all non-ε symbols of FIRST(X_1). Also add the non-ε symbols of FIRST(X_2), if ε is in FIRST(X_1); the non-ε symbols of FIRST(X_3), if ε is in FIRST(X_1) and FIRST(X_2); and so on. Finally, add ε to $\text{FIRST}(X_1 X_2 \ldots X_n)$ if, for all i, ε is in FIRST(X_i)
    - If t is a terminal, $\text{FIRST}(t) = \{t\}$
- **FOLLOW(A)**, for nonterminal A, to be the set of terminals a that can appear immediately to the right of A in some sentential form
    - i.e. there exists a derivation of the form $S \Rightarrow^* \alpha A a \beta$, for some α and β
    - There may have been symbols between A and a at some time during derivation, but they derived to ε
    - If A can be the rightmost symbol in some form, then $\$$ is in FOLLOW(A) ($ is a special "endmarker" symbol that is assumed not to be a symbol of any grammar)
    - If t is a terminal, $\text{FOLLOW}(t) = \varnothing$
    - Steps:

1. Place $\$$ in FOLLOW(S), where S is the start symbol, and $\$$ is the input right endmarker.
2. If there is a production $A \to \alpha B \beta$, then everything in FIRST(β) except ε is in FOLLOW (B)
3. If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$, where FIRST (β) contains ε, then everything in FOLLOW (A) is in FOLLOW (B).

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

$$E \to TE'$$
$$E' \to +TE' \mid \varepsilon$$
$$T \to FT'$$
$$T' \to *FT' \mid \varepsilon$$
$$F \to (E) \mid id$$

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

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

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

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

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

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

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

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

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

$$\text{FOLLOW}(T) = \text{FIRST}(E') \setminus \{\varepsilon\} \cup \text{FOLLOW}(E') = \{+,), \$\}$$

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:

$$\text{FOLLOW}(F) = \text{FIRST}(T') \setminus \{\varepsilon\} = \{*\}$$

In addition, FIRST(T') contains ε, so following rule (3):

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

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 $A \to \alpha \mid \beta$ are two distinct productions of G:
  - FIRST(α) and FIRST(β) are disjoint sets, i.e. FIRST$(\alpha) \cap$ FIRST$(\beta) = \varnothing$
  - If ε is in FIRST(β), then FIRST(α) and FOLLOW(A) are disjoint sets
  - If ε is in FIRST(α), then FIRST(β) and FOLLOW(A) are disjoint sets.
  - Note that due to the first condition, at most one of α or β can derive ε
- Flow-control statements, e.g. if, while, for, generally satisfy LL(1) constraints
- Not all grammars are LL(1), eg. $S \to aS \mid a$ 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 $A \to \alpha$ is chosen if the next input symbol a is in FIRST(α). The only complication occurs when $\alpha = \varepsilon$ or, more generally, $\alpha \Rightarrow^* \varepsilon$. In this case, we should again choose $A \to \alpha$, 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 $A \to \alpha$ of the grammar, do the following:
    1. For each terminal a in FIRST(A), add $A \to \alpha$ to $M[A, a]$.

2. If ε is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \to \alpha$ to $M[A,b]$. If ε is in FIRST(α) and \$ is in FOLLOW(A), add $A \to \alpha$ 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)

Non-Terminals	Input Symbol					
	ID	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

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] → optionally include string s
  - {s} → repeat string s 0 or more times
  - (…) → grouping
- **How to convert grammar to EBNF**?
  - First, remove left recursion and do left factoring
  - Use quotes for terminals
  - Replace → with ::=
  - Replace ε 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. □

```
set ip to point to the first symbol of w;
set X to the top stack symbol;
while ( X ≠ $ ) { /* 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 → Y1 Y2 ··· Yk ) {
        output the production X → Y1 Y2 ··· Yk;
        pop the stack;
        push Yk, Yk−1, ... , Y1 onto the stack, with Y1 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):

$$E \underset{lm}{\Rightarrow} TE' \underset{lm}{\Rightarrow} FT'E' \underset{lm}{\Rightarrow} \mathbf{id}\,T'E' \underset{lm}{\Rightarrow} \mathbf{id}\,E' \underset{lm}{\Rightarrow} \mathbf{id} + TE' \underset{lm}{\Rightarrow} \cdots$$

MATCHED	STACK	INPUT	ACTION
	$E\$$	**id** + **id** * **id**\$	
	$TE'\$$	**id** + **id** * **id**\$	output $E \to TE'$
	$FT'E'\$$	**id** + **id** * **id**\$	output $T \to FT'$
	id $T'E'\$$	**id** + **id** * **id**\$	output $F \to$ **id**
id	$T'E'\$$	+ **id** * **id**\$	match **id**
id	$E'\$$	+ **id** * **id**\$	output $T' \to \epsilon$
id	$+\,TE'\$$	+ **id** * **id**\$	output $E' \to +\,TE'$
id +	$TE'\$$	**id** * **id**\$	match +
id +	$FT'E'\$$	**id** * **id**\$	output $T \to FT'$
id +	**id** $T'E'\$$	**id** * **id**\$	output $F \to$ **id**
id + **id**	$T'E'\$$	* **id**\$	match **id**
id + **id**	$*\,FT'E'\$$	* **id**\$	output $T' \to *\,FT'$
id + **id** *	$FT'E'\$$	**id**\$	match *
id + **id** *	**id** $T'E'\$$	**id**\$	output $F \to$ **id**
id + **id** * **id**	$T'E'\$$	\$	match **id**
id + **id** * **id**	$E'\$$	\$	output $T' \to \epsilon$
id + **id** * **id**	\$	\$	output $E' \to \epsilon$