Lecture 16: Parsing

CSCI 131
Fall, 2008

Kim Bruce

Rewrite Grammar

```
      <exp> ::= <term> <termTail>                       (1)
  <termTail> ::= <addop> <term> <termTail>              (2)
               | ε                                      (3)
     <term> ::= <factor> <factorTail>                   (4)
<factorTail> ::= <mulop> <factor> <factorTail>          (5)
               | ε                                      (6)
   <factor> ::= ( <exp> )                               (7)
               | NUM                                    (8)
               | ID                                     (9)
    <addop> ::= + | -                                  (10)
    <mulop> ::= * | /                                  (11)
```

No left recursion.
How do we know which production to take?

FIRST

- *Intuition*: b ∈ First(X) iff there is a derivation X →* bω for some ω.

1. First(b) = {b} for b a terminal or the empty string
2. If have X ::= ω_1 | ω_2 | ... | ω_n then
   First(X) = First(ω_1) ∪ ... ∪ First(ω_n)
3. For any right hand side $u_1u_2...u_n$
   - First(u_1) ⊆ First($u_1u_2...u_n$)
   - if all of u_1, u_2..., u_{i-1} can derive the empty string then also First(u_i) ⊆ First($u_1u_2...u_n$)
   - empty string is in First($u_1u_2...u_n$) iff all of u_1, u_2..., u_n can derive the empty string

Follow

- *Intuition:* A terminal b ∈ Follow(X) iff there is a derivation S →* vXbω for some v and ω.

1. If S is the start symbol then put EOF ∈ Follow(S)
2. For all rules of the form A ::= wXv,
   a. Add all elements of First(v) to Follow(X)
   b. If v can derive the empty string then add all elts of Follow(A) to Follow(X)

- Follow(X) only used if can derive empty string from X.

Follow for Arithmetic

FOLLOW(<exp>) = { EOF,) }
FOLLOW(<termTail>) = FOLLOW(<exp>) = { EOF,) }
FOLLOW(<term>) = FIRST(<termTail>) ∪ FOLLOW(<exp>) ∪ FOLLOW(<termTail>)
= { +, -, EOF,) }
FOLLOW(<factorTail>) = { +, -, EOF,) }

$\left.\begin{array}{l}\textit{FOLLOW(<factor>) = \{ *, /, +, -, EOF \}}\\ \textit{FOLLOW(<addop>) = \{ (, NUM, ID \}}\\ \textit{FOLLOW(<mulop>) = \{ (, NUM, ID \}}\end{array}\right\}$ *Not needed!*

Predictive Parsing

- Want at most one production per entry.
  - unambiguous choice of production
  - may require rewriting of grammar!
- Rules:
  - If A ::= α_1 |...| α_n then for all i ≠ j, First(α_i) ∩ First(α_j) = ∅.
  - If X →* ε, then First(X) ∩ Follow(X) = ∅.

Build Table

- Create table to guide parsing.
  - Rows are non-terminals, columns are terminals
  - Put production X ::= w in entry (X,b) iff
    - b ∈ First(w) or
    - empty string is in First(w) and b ∈ Follow(X)
- Production in entry (X,b) iff applying production can eventually lead to string starting with b.

First for Arithmetic

FIRST(<exp>) = { (, NUM, ID }
FIRST(<termTail>) = { +, -, ε }
FIRST(<term>) = { (, NUM, ID }
FIRST(<factorTail>) = { *, /, ε }
FIRST(<factor>) = { (, NUM, ID }
FIRST(<addop>) = { +, - }
FIRST(<mulop>) = { *, / }

Parse Table for Arithmetic

Non-terminals	*ID*	*NUM*	*Addop*	*Mulop*	*(*	*)*	*EOF*
<exp>	1	1			1		
<termTail>			2			3	3
<term>	4	4			4		
<factTail>			6	5		6	6
<factor>	9	8			7		
<addop>			10				
<mulop>				11			

See ML Recursive Descent Parser

Table-Driven Stack-based Parser

- http://en.wikipedia.org/wiki/LL_parser
- Start with S $ on stack and input $ to be recognized.
- Use table to replace non-terminals on top of stack.
- If terminal on top of stack matches next input then erase both and proceed.
- Success if end up clearing stack and input
- Show with ID * (NUM + NUM)

Another alternative

- LR(1) parsers -- bottom up, gives right-most derivation.
- YACC is LR(1). ANTLR is LL(1).
- k in LL(k) and LR(k) indicates how many letters of look ahead are necessary -- e.g. length of strings in columns of table.
- Compiler writers are happiest with k=1 to avoid exponential blow-up of table. May have to rewrite grammars.