CS480

Top Down Parsing
Ch 4 p 181-195

March 2, 2009
Parsing

• Will the following code parse?
• Is it valid C code?

```c
#include <stdio.h>

int main()
{
    x ++;
}
```
Top Down Parsing

- Find left most derivation of a string

- Backtracking

- Predictive
a) Is this grammar ambiguous?
b) Is this grammar left-recursive?
c) Show $S_{lm} \Rightarrow^+ \text{cad}$
   Is backtracking necessary?
d) Can this grammar be left factored?
Top Down Parsing

• Recursive decent
  – no backtracking
  – no left-recursion (left factored)

• LL(1) parsing
  – L: Left to right
  – L: Left most derivation
  – (1): One lookahead token
Grammar

\[ \text{expr} \rightarrow \text{expr op term} \mid \text{term} \]
\[ \text{op} \rightarrow + \mid - \]
\[ \text{term} \rightarrow \text{term mulop factor} \mid \text{factor} \]
\[ \text{mulop} \rightarrow * \]
\[ \text{factor} \rightarrow ( \text{expr} ) \mid \text{num} \]
Parse Tables

• Not all grammars are good for recursive descent
  – backtracking can be expensive

• LL(1) uses a stack instead of recursion

• Use FIRST and FOLLOW to build predictive parse tables
Example Grammar

\[
\begin{align*}
E & \rightarrow TE' \\
E' & \rightarrow +TE' \mid \varepsilon \\
T & \rightarrow FT' \\
T' & \rightarrow *FT' \mid \varepsilon \\
F & \rightarrow (E) \mid \text{id}
\end{align*}
\]
“Let FIRST(α) be the set of terminals that begin the strings derived from α. If α =>* ε, then ε is also in FIRST(α).” Aho p 188

FIRST(E)
FIRST(E')
FIRST(F)
FIRST(T)
FIRST(T')

FIRST(EXPRESSION) is to be used in your top-down parser to identify the beginning of an expression or ε
FOLLOW

• "FOLLOW(N), for nonterminal N, is the set of terminals t that can appear immediately to the right of N in some sentential form, that is, the set of terminals t such that there exists a derivation of the form $S \Rightarrow^* \alpha N t \beta$ for some $\alpha$ and $\beta."$ Aho, p 189

FOLLOW(E)
FOLLOW(E')
FOLLOW(F)
FOLLOW(T)
FOLLOW(T')
Parse Table Construction

for (each nonterminal N and production possibility N → α)
{
  for (each token t in the FIRST(α))
  {
    Add N → α to TBL[N,t]
  }

  if (ε is an element of FIRST(α))
  {
    for (each token a in the FOLLOW(N),
    Add N → α to TBL[N,t]
  }
}