CS480

Top Down Parsing

Ch 4 p 161-165, 181-195

February 24, 2013

Parsing

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

```
#include <stdio.h>

int main()
{
   x ++;
}
```

Top Down Parsing

- Find left most derivation of a string
- Backtracking
- Predictive

```
S -> cBd
B -> ab | a
```

a) Is this grammar ambiguous?

b) Is this grammar left-recursive?

c) Show $S \underset{lm}{\Rightarrow}^{+}$ 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

```
expr   -> expr op term | term
op     -> + | -
term   -> term mulop factor |
          factor
mulop  -> *
factor -> ( expr ) | 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

```
E  -> TE'
E' -> +TE' | ε
T  -> FT'
T' -> *FT' | ε
F  -> (E)  | id
```

FIRST

"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 α and β." 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]
  }
}
```

Parse Table

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

FOLLOW(E) = FOLLOW(E') = {**$,)** }
FOLLOW(T) = FOLLOW(T') = { **$, +,)**}
FOLLOW(F) = {**$, *, +,)** }

```
1E  -> TE'
2E' -> +TE'
3E' -> ε
4T  -> FT'
5T' -> *FT'
6T' -> ε
7F  -> (E)
8F  -> id
```

	+	*	(	)	id	$
E						
E'						
T						
T'						
F						

Error Handling

- Goals
  - Report presence oferrors clearly and accurately
  - Recover quickly to find more errors
  - Should not slow down processing

- LL & LR detect errors quickly
  - Viable-prefix property: detect an error as soon as a prefix is seen that is not a viable prefix for any string in the language.

Error Recovery

- Poor error handling adds spurious errors
  - Syntactic or semantic

- Panic Mode

- Phase level recovery

- Error Productions

- Global corrections