CS480

Syntax Analysis

Ch 4 p 159-195

February 23, 2009

CS310 Problems

- Describe (in English) the language denoted by the regular expression ((ε|0)1*)*

- Write regular definitions for:
  - all strings that begin with an aa
  - all strings that contain aa
  - all strings that do not contain aa
  - All are over the alphabet {a,b}.
- Construct an NFA for the regular expression ((ε| a)b*)*

CFGs

```
expr -> expr op expr | ( expr ) |
number | id
```

```
op -> + | - | *
```

- Backus-Naur Form

```
< expr > ::= < expr > < op > < expr > |
 (< expr >) | NUMBER
```

```
< op > ::= + | - | *
```

Notation from the Book

- Terminals
- Nonterminals
- String of terminals
- Greek Letters
- Alternate Forms
- Start production

Derivations

- =>
  - can derive with one application of a production
- =>*
  - can derive with zero or more applications of any productions

```
E -> ( E ) | a
```

Does E =>* ((a))?

Does E => ((a))?

Does E =>* (a)(a)?

Grammars

- G1: A -> Aa | a
- G2: B -> aB | a

- Do G1 and G2 describe the same language?
- Are both G1 and G2 equivalent to a*?
- Are they ambiguous?
  - How fix?
- Right or Left recursive?
  - What problems could arise?
- Does A =>* ε

More...

- Give a CFG which generates sequences of one or more statements (s) separated by ;
  - (i.e. L(G) = {s s;s s;s;s ...})

- Give a CFG which generates sequences of one or more statements where the semicolon is a terminator and not a separator (i.e. L(G) = {s; s;s; s;s;s; ...})

Parsing!

```
expr -> expr op expr | ( expr ) |
number

op -> + | - | * | /
```

- Problem?

  1 + 3 * 8

  Left most? Right most?

- Ambiguity:
  – Get rid of it **OR**
  – Use rules to limit its impact

More..

```
expr -> expr op expr | term
op -> + | - | *
term -> number
```

- Ambiguous?
  - Why or why not?
  - Precedence?

More still...

```
stmt -> ifstmt | other
ifstmt -> if ( expr ) stmt |
 if ( expr ) stmt else stmt
expr -> T | F
```

- Thoughts?

- Fixes?

Immediate Left Recursion

- Immediate Left Recursion

```
E -> E + T | T
T -> T * F | F
F -> (E) | id
```

Differences?
Why is this important?

- Nonimmediate Left Recursion:

```
S -> Aa | b
A -> Ac | Sd | e
```

How do you remove each type?

Practice

```
S -> Ba | b
B -> Sa | a
```

- What is the language?
- Eliminate all the left recursion
  - Algorithm 4.1 on p 177