Tutorial 1b

September 23, 2004
1 Regular Expressions

Express the following as regular expressions:

1. All binary strings
   
   
   \((0 + 1)^*\)

2. All binary strings such that the number of 1’s is divisible by 5
   
   \((0^*10^*10^*10^*1)^*0^*\)

3. All strings over \(\{a\}\) of length \(3 + 4 \times k\), \(k \geq 0\)
   
   \(aaa(aa)^*\)

4. All strings over \(\{a, b\}\), such that the substring “ab” occurs exactly once
   
   \(b^*a^*abb^*a^*\)
2 CFG

<table>
<thead>
<tr>
<th>Desc</th>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary prefix operators</td>
<td>+, -</td>
<td>N/A</td>
</tr>
<tr>
<td>Multiplicative operators</td>
<td>*, /, %</td>
<td>Left</td>
</tr>
<tr>
<td>Additive operators</td>
<td>+, -</td>
<td>Left</td>
</tr>
<tr>
<td>Conditional (ternary)</td>
<td>? :</td>
<td>None</td>
</tr>
</tbody>
</table>

Rules:

- decreasing order of precedence, with all operators at a given level having the same precedence

- assume the non-terminal <simple-expr> is defined and handles literals, variables, etc.

- handle parentheses

Note:

-- left/right associativity means left/right recursion of the given nonterminal

-- higher precedence means the given nonterminal is lower down in the nonterminal recursion

-- have parentheses, to indicate when default associativity or precedence is to be overridden in a given expression.
<expr> ::= <cond-expr>
<cond-expr> ::= <add-expr>
               | <add-expr> ? <add-expr> : <add-expr>
<add-expr> ::= <mult-expr>
              | <add-expr> <add-op> <mult-expr>
<mult-expr> ::= <unary-expr>
              | <mult-expr> <mult-op> <unary-expr>
<unary-expr> ::= <primary-expr>
              | <unary-op> <primary-expr>
<primary-expr> ::= <simple-expr> | ( <expr> )
<add-op> ::= + | -
<mult-op> ::= * | / | %
<unary-op> ::= + | -

Draw parse trees for:

a) 3 * - 2 * ( 5 + 4 )

b) a ? b * 3 + 2 : c / + 3
3 ambiguity

\[
\text{<E> ::= if <E> then <E> | if <E> then <E> else <E>}
\]

This grammar is ambiguous and somewhat famous. The ‘‘dangling else ambiguity’’

The expression if e1 then if e2 then e3 else e4 has two parse trees.

Develop a grammar that generates the same strings, but is not ambiguous.

A possible solution:

\[
\text{<E> ::= <match> | <nomatch>}
\]
\[
\text{<match> ::= if <E> then <match> else <match>}
\]
\[
\text{<nomatch>::= if <E> then <match> else <unmatch> | if <E> then <E>}
\]

Draw a parse tree for if e1 then if e2 then e3 else e4 again, note the difference.