Problem Set 7:

Problem 1

Consider the following BNF specification:

\[
\text{<BExpression> ::= True}
\]
\[
| \quad \text{not <BExpression>}
\]
\[
| \quad <\text{BExpression}> \text{ and } <\text{BExpression}>
\]

Where true, not, and and are keywords.

We can define the rank of an expression \( E \in \text{<BExpression>} \) by the following definition:

- \( \text{rank}(\text{true}) = 0 \)
- \( \text{rank}(\text{not } E) = \text{rank}(E) + 1 \)
- \( \text{rank}(E_1 \text{ and } E_2) = \max\{\text{rank}(E_1), \text{rank}(E_2)\} + 1 \)

Prove by structural induction that every expression \( E \in \text{<BExpression>} \) has a rank greater than or equal to 0.

Problem 2

Start with “xmlscheme-10.scm”, which can be found on the course page. Add a “let*” construct to the defined language:

\[
\text{<expression> ::= "<let*" <declarations> <expressions> "/"}
\]
\[
\text{let-star-exp (decls expr)}
\]

The “let*” should exhibit the same behavior as Scheme’s “let*” declaration.
Problem 3

Start with “xmlscheme-10.scm” or the language developed in Problem 2.
Define the procedure modulo for positive numbers in XMLScheme.

<invoke <reference value = modulo />  
  <arguments <integer value = 13 /> <integer value = 4 /> /> />  
⇒ 1
<invoke <reference value = modulo />  
  <arguments <integer value = 12 /> <integer value = 4 /> /> />  
⇒ 0
<invoke <reference value = modulo />  
  <arguments <integer value = 12 /> <integer value = 0 /> /> />  
⇒/: division by zero

Problem 4

Start with “xmlscheme-10.scm” or the language developed in Problem 2 again. Now add modulo to the initial environment of the interpreter. That is, add a binding from the name “modulo” to a function (or closure) implementing the behavior of the modulo function to the initial environment init-env. Test your implementation with the samples given in Problem 3.

Submission deadline: Tuesday, April 10, 2007, 2:10 p.m.
Submission procedure: on paper in class.