Problem Set 3: Semantics & Induction

Problem 1
Suppose $\sigma_1 = \{(x,1), (y,2), (z,3)\}$, $\sigma_2 = \{(y,5)\}$, and $\sigma_3 = \{(w,1)\}$. What are the results of the following operations?

- $\sigma_1 \cup \sigma_2$
- $\sigma_1 \cup \sigma_3$
- $\sigma_2 \cup \sigma_3$
- $\emptyset \cup \sigma_2$

Problem 2
Prove the validity of the following Hoare triple using the proof rules shown in class:

\{z=4\}
\begin{align*}
x &= 3; \\
y &= 6; \\
\text{if } (x > z) & \quad r = x; \\
\text{else & else} & \quad r = y; \\
\text{else & else} & \quad r = z; \\
\{x=3, y=6, z=4, r=6\}
\end{align*}

Problem 3
Using the set builder notation $\{x \in A \mid P(x) = \text{True} \}$ define the following sets:

1. The set of all integers greater than 50 and smaller than 300.
2. The set of all upper-case ASCII letters.
3. The set of all pairs $(p, q)$, where $p$ is an element of the set specified in 1., and $q$ being an element specified in 2.
Problem 4

1. Define inductively the set of all strings, called $A^*$, which can be generated from the alphabet $A = \{0, 1\}$. Define appropriate base- and inductive clauses.

2. Define inductively a list that consists of characters of the alphabet $A = \{0, 1\}$. You should use the pair notation.

FOLLOW THE GRAMMAR:

When defining a program based on structural induction, the structure of the program should be patterned after the structure of the data.

This means that (i) each LHS is encoded in a separate procedure, (ii) in each procedure, check to which production the input to the procedure belongs, and (iii) for each non-terminal on the RHS specify a recursive call to the procedure for that non-terminal.

Problem 5

Consider the following BNF specification:

\[
\langle \text{Expression} \rangle \quad ::= \quad \text{(true)}
\]

\[
\quad | \quad \text{(not} \langle \text{Expression} \rangle \text{)}
\]

\[
\quad | \quad \langle \text{Expression} \rangle \text{and} \langle \text{Expression} \rangle
\]

Where true, not, and and are keywords.

Define recursively the procedure (is-expression? obj) that returns #t if obj is an element of <Expression>, otherwise the procedure should return #f. For example, the call (is-expression? '((not (true)) and ((true) and (true)))) returns #t.

Problem 6

Consider the following BNF specification:

\[
\langle \text{LambdaExp} \rangle \quad ::= \quad \langle \text{Identifier} \rangle
\]

\[
\quad | \quad \text{(lambda} \langle \text{Identifier} \rangle \langle \text{LambdaExp} \rangle \text{)}
\]

\[
\quad | \quad \langle \text{LambdaExp} \rangle \langle \text{LambdaExp} \rangle
\]

Implement the procedure (is-lambda-exp? term) that returns #t if term is an element of <LambdaExp>, otherwise the procedure should return #f. In order to simplify the definition of the procedure is-lambda-exp?, consider a decomposition of the problem into smaller sub-problems. For example,

> (is-lambda-exp? 'x)

#t

> (is-lambda-exp? 'lambda)

#f
> (is-lambda-exp? '(lambda (x) x))
#t
> (is-lambda-exp? '(lambda (x) lambda))
#f
> (is-lambda-exp? '(((lambda (x) x) x)))
#t
> (is-lambda-exp? '(((lambda (x) x) x) lambda))
#f
> (is-lambda-exp? '(((lambda (x) x) (lambda (y) y))))
#t

**Problem 7**
Consider the following BNF specification:

```
<s-list> ::= ( {<symbol-expression>}* )
<symbol-expression> ::= <symbol>
  | <s-list>
```

Define the procedure `(count-occurrences slist symbol)` that returns the number of occurrences of `symbol` in `slist`. For examples,

```scheme
> (count-occurrences '(((f x) y (((x z) x))) 'x)
3
> (count-occurrences '(((f x) y (((x z) () x))) 'x)
3
> (count-occurrences '(((f x) y (((x z) x))) 'y)
1
> (count-occurrences '(((f x) y (((x z) x))) 'w)
0
```

**Submission deadline: Tuesday, February 13, 2007, 2:10 p.m.**  
**Submission procedure: on paper in class.**