Due: Monday 12 February 2007 13:00

**Exercise 1**

Draw the complete inference tree for the following statement

\[ \langle x := 1; \text{while } x < y \text{ do } x := 2 \times x, \sigma \rangle \rightarrow \sigma' \]

where \( \sigma(x) = 0 \) and \( \sigma(y) = 3 \). Explain what \( \sigma' \) is.

**Exercise 2**

Let \( c_0 \sim c_1 \) be defined by: for all \( \sigma, \sigma' \in \Sigma \), \( \langle c_0, \sigma \rangle \rightarrow \sigma' \) iff \( \langle c_1, \sigma \rangle \rightarrow \sigma' \). Prove:

a) if then c else c \sim c.

b) while b do c \sim if b then (c; while b do c) else skip.

You may use, without a proof, the totality of the transition system for Boolean expressions. That is, you can assume that for any \( b \in \text{Bexp} \) and any state \( \sigma \) one can infer either \( \langle b, \sigma \rangle \rightarrow \text{true} \) or \( \langle b, \sigma \rangle \rightarrow \text{false} \).

**Exercise 3**

The transition system for \textbf{Imp} evaluates expressions all at once. An alternative approach would be to compute value of expression one step at a time, using transitions of the form \( \langle e, \sigma \rangle \rightarrow_1 \langle e', \sigma \rangle \). In this transition \( e' \) is the result of executing single, leftmost atomic operation in expression \( e \). For example the rules for arithmetic addition would look as follows.

\[
\begin{align*}
\langle a_0, \sigma \rangle \rightarrow_1 \langle a'_0, \sigma' \rangle & \quad \text{and} \quad \langle a_1, \sigma \rangle \rightarrow_1 \langle a'_1, \sigma' \rangle \\
\langle a_0 + a_1, \sigma \rangle \rightarrow_1 \langle a'_0 + a'_1, \sigma' \rangle & \quad \text{and} \quad \langle n_0 + a_1, \sigma \rangle \rightarrow_1 \langle n_0 + a'_1, \sigma' \rangle \\
\langle n_0 + n_1, \sigma \rangle \rightarrow_1 \langle n_0 + n_1, \sigma \rangle
\end{align*}
\]

Note that in the last rule above, the plus sign to the left is a “syntactic” plus sign from the \textbf{Imp} language, while the plus sign to the right is a mathematical plus sign. Thus the rule turns an expression intended to compute the sum of two numbers into the single number which is the actual sum.

Current variable values are accessed using the following rule

\[ \langle x, \sigma \rangle \rightarrow_1 \langle \sigma(x), \sigma \rangle \]
A computation of an arithmetic expression $a$ is done as a sequence of transitions

$$
\langle a, \sigma \rangle \rightarrow_1 \langle a_1, \sigma_1 \rangle \rightarrow_1 \cdots \rightarrow_1 \langle a_{k-1}, \sigma_{k-1} \rangle \rightarrow_1 \langle n, \sigma' \rangle
$$

where $a$ is an Imp arithmetic expression and $n$ the number it computes.
Eventually each expression is reduced to a single constant and there are no rules for further reduction of constants.

Now for the actual problem:
Give a complete system of single step evaluation rules for all arithmetic and Boolean expressions of Imp. Give a sequence of single step transitions of your system representing evaluation of the following expression

$$
e \equiv (3 \times x) - y < 0 \text{ and not}(y = 0)
$$

in a state $\sigma$ where $\sigma(x) = 2$ and $\sigma(y) = 3$.

**Exercise 4**

We will write $\langle e, \sigma \rangle \rightarrow^k_1 \langle e', \sigma' \rangle$ to express the fact that there exists a sequence of $k$ single step transitions

$$
\langle e, \sigma \rangle \rightarrow_1 \langle e_1, \sigma_1 \rangle \rightarrow_1 \cdots \rightarrow_1 \langle e_{k-1}, \sigma_{k-1} \rangle \rightarrow_1 \langle e', \sigma' \rangle
$$

such that each transition in the sequence can be derived using the single step evaluation rules. We write $\langle e, \sigma \rangle \rightarrow^1_1 \langle e', \sigma' \rangle$ if $\langle e, \sigma \rangle \rightarrow^k_1 \langle e', \sigma' \rangle$ for some $k$.

Let $a \in Aexp$. Prove the following

(a) If $\langle a, \sigma \rangle \rightarrow_1 \langle a', \sigma \rangle$ can be derived using single step evaluation rules and $\langle a', \sigma \rangle \rightarrow n$ can be derived using "big step" evaluation rules, then also $\langle a, \sigma \rangle \rightarrow n$ can be derived in "big step" semantics.

(b) If $\langle a, \sigma \rangle \rightarrow^n_1 \langle n, \sigma \rangle$ then $\langle a, \sigma \rangle \rightarrow n$ can be derived in "big step" semantics. Hint: use induction over the number of small step transitions.

**Exercise 5**

Modify the ML implementation of the operational semantics to implement the small-step semantics above. (It is sufficient that you implement small-step semantics of arithmetic expressions.) Submit a printout of the ML file with your modifications marked using a pen. Include a printout of an ML run where you demonstrate that the changes work by executing some simple example programs of your own invention.