CS 221/321 **Programming Languages** **Homework 3**
Fall 2009 **Due: Oct 27, 2009**

1. [20 points] Termination in Arith. Define a size function for expressions in the abstract syntax for Arith.

$$e ::= \texttt{num}(n) \mid \texttt{var}(x) \mid \texttt{plus}(e,e) \mid \times(e,e) \mid \texttt{let}(e, x.e)$$

where n ranges over natural number constants and x ranges over a set of variables. For instance, the size function might count the number syntax constructors in the expression. It should be obvious that the size of an expression is strictly greater than the size of a proper subexpression. Use the size function to prove that the SOS evaluation of any closed expression terminates in a finite number of steps, by showing that if $e \mapsto e'$ then $\texttt{size}(e) > size(e')$.

Challenge Question. Suppose that we replace the two rules for let expressions with the following single rule

$$\frac{}{\texttt{let}(e_1, x.e_2) \;\mapsto\; \{e_1/x\}e_2} \tag{1}$$

The question is whether, using this "by-name" treatment of let, does the evaluation of every expression still terminate. The answer is yes, but the proof idea used above doesn't work any more. See if you can find a termination proof for this new version of evaluation.

2. [20 points] Prove the following version of the Substitution Lemma for Arith: Lemma: If $\Gamma \vdash e_1 \texttt{ ok}$ and $\Gamma \cup \{x\} \vdash e_2 \texttt{ ok}$, then $\Gamma \vdash \{e_1/x\}e_2 \texttt{ ok}$.

3. [15 points] Write out the Induction Principle for proofs of properties of Arith expression as a logical formula. This is a bit of a trick question. The problem is how to treat the abstractor argument of `let`, which is not quite an expression.

4. [15 points] Consider an alternate, more flexible, way of treating the primitive arithmetic operations plus and times in Arith. Instead of making plus and times basic syntactic forms of the language, we have a more general form $\texttt{op}(o, e_1, e_2)$ with a new syntactic category

$$o \;::=\; + \mid \times$$

$$e \;::= \texttt{num}(n) \mid \texttt{var}(v) \mid \texttt{op}(o, e, e) \mid \texttt{let}(e, x.e)$$

Give rules for the static ($\Gamma \vdash e \texttt{ ok}$) and dynamic ($e \mapsto e'$) semantics for this alternative abstract syntax. You should unify the instruction rules for arithmetic operations by assuming a mapping from operator symbols ($+$, $\times$) to the corresponding arithmetic operations. We could denote this mapping as $M(o)$, so $M(+)$ denotes the real arithmetic addition operation.

5. [20 points]. Programming exercise. Modify the code in the file arith-SOS.sml to use the modified abstract syntax from Problem 4. Make sure your code compiles without error, and test it.