CS 221/321	Programming Languages	Homework Solution 3
Fall 2009		Due October 29, 2009

1. Termination in Arith.

Prop: $forall e \in \text{Arith}.\ \vdash e\texttt{ok} \Rightarrow \exists n \in \texttt{Nat}.\, e \stackrel{!}{\longmapsto} \texttt{num}(n)$.

Proof: We define a function *size* mapping Arith expressions to natural numbers. *size* simply counts the number of syntax constructors in the expression:

$$\begin{aligned}
\texttt{size}(\texttt{num}(n)) &= 1 \\
\texttt{size}(\texttt{var}(x)) &= 1 \\
\texttt{size}(\texttt{plus}(e_1, e_2)) &= \texttt{size}(e_1) + \texttt{size}(e_2) + 1 \\
\texttt{size}(\texttt{times}(e_1, e_2)) &= \texttt{size}(e_1) + \texttt{size}(e_2) + 1 \\
\texttt{size}(\texttt{let}(e_1, x.e_2)) &= \texttt{size}(e_1) + \texttt{size}(e_2) + 1
\end{aligned}$$

Now we prove a lemma saying that if $e \mapsto e'$, the size of e' is strictly less than the size of e.

Lemma. $(\mapsto e, e') \ \Rightarrow\ \texttt{size}(e') < \texttt{size}(e)$. **Proof**: By rule induction the judgement $e \mapsto e'$.

Case (base): $e \mapsto e'$ by rule E1. Then $e = \texttt{plus}(\texttt{num}(m), \texttt{num}(n))$ and $e' = \texttt{num}(p)$, where $p = m + n$. So $\texttt{size}(e) = 3$ and $\texttt{size}(e') = 1$, hence $\texttt{size}(e') < \texttt{size}(e)$.

Case (base): $e \mapsto e'$ by rule E2. Same as the previous case, with `plus` replaced by `times`.

Case (base): $e \mapsto e'$ by rule E3. Then $e = \texttt{let}(\texttt{num}(n), x.e_2)$ and $e' = \{\texttt{num}(n)/x\}e_2$.

Claim: $\texttt{size}(\{\texttt{num}(n)/x\}e_2) = \texttt{size}(e_2)$. This is proved by a straightforward induction over the structure of e_2. Intuitively, the substitution results in all $\texttt{var}(x)$ subterms of e_2 being replaced by $\texttt{num}(n)$. Since $\texttt{var}(x)$ and $\texttt{num}(n)$ are both of size 1, the substitution does not change the size of the term.

Since $\texttt{size}(e) = \texttt{size}(e_1) + \texttt{size}(e_2) + 1$, and by the Claim, $\texttt{size}(e') = \texttt{size}(e_2)$, we have $\texttt{size}(e') < \texttt{size}(e)$.

Case (ind): $e \mapsto e'$ by rule E4. Then $e = \texttt{plus}(e_1, e_2)$ and $e' = \texttt{plus}(e_1', e_2)$, where (by inversion), $e_1 \mapsto e_1'$.

(IH) $\texttt{size}(e_1') < size(e_1)$.

Hence

$$\begin{aligned}
\texttt{size}(e') &= \texttt{size}(e_1') + \texttt{size}(e_2) + 1 \\
&< \texttt{size}(e_1) + \texttt{size}(e_2) + 1 \\
&= \texttt{size}(e)
\end{aligned}$$

Cases (ind): $e \mapsto e'$ by rule E5, E6, E7, and E8 are essentially the same as E4 with minor variations.

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 $\Gamma \ \vdash\ \{e_1/x\}e_2\ \texttt{ok}$.

Proof: By induction on the derivation of $\Gamma \cup \{x\} \ \vdash\ e_2\ \texttt{ok}$. [Note: it is tempting to try induction on the the structure of e_2, but this approach gets hung up on the `let` case.

Case (base): By rule S1, with $e_2 = \texttt{var}(x)$. Then $\{e_1/x\}e_2 = e_1$, so $\Gamma \ \vdash\ \{e_1/x\}e_2\ \texttt{ok}$ by assumption.

Case (base): By rule S1, with $e_2 = \texttt{var}(y)$ where $y \neq x$. Then $\{e_1/x\}e_2 = e_2 = y$, so the fact that $\Gamma \cup \{x\} \vdash e_2$ ok implies that $y \in \Gamma \cup \{x\}$, but since $y \neq x$ this means that $y \in \Gamma$. Hence $\Gamma \vdash \{e_1/x\}e_2$ ok by the var rule (S1).

Case (base): By rule S2. Then $e_2 = \texttt{num}(n)$. Then $\{e_1/x\}e_2 = \texttt{num}(n)$ by the definition of substitution, and $\Gamma \vdash \{e_1/x\}e_2$ ok by the num rule (S2).

Case (ind): By rule S3, with $e_2 = \texttt{plus}(e_3, e_4)$. Then the induction hypotheses are:

(IH1) $\Gamma \vdash \{e_1/x\}e_3$ ok and
(IH2) $\Gamma \vdash \{e_1/x\}e_4$ ok

But $\{e_1/x\}e_2 = \texttt{plus}(\{e_1/x\}e_3, \{e_1/x\}e_4)$, and $\Gamma \vdash \texttt{plus}(\{e_1/x\}e_3, \{e_1/x\}e_4)$ ok by the plus rule (S3).

Case (ind): By rule S4, $e_2 = \times(e_3, e_4)$. Similar to the previous case.

Case (ind): By rule S5 (let rule). Then $e_2 = \texttt{let}(e_3, y.e_4)$. We can assume that the bound variable y is not x (if necessary by α-converting the bound variable). Then $\{e_1/x\}e_2 = \texttt{let}(\{e_1/x\}e_3, y.(\{e_1/x\}e_4))$. By inversion of the second assumption, we have

(1) $\Gamma \cup \{x\} \vdash e3\ ok$
(2) $(\Gamma \cup \{x\}) \cup \{y\} \vdash e4\ ok$

and (2) can be rephrased as

(3) $\Gamma' \cup \{x\}) \vdash e4\ ok$

where $\Gamma' = \Gamma \cup \{y\}$. Noting that $\Gamma' \vdash e_1$ (a "Weakening Lemma" allows extraneous variables to be added to the context in an ok judgement). We have the induction hypotheses:

(IH1) $\Gamma \vdash \{e_1/x\}e_3)$ ok
(IH2) $\Gamma' \vdash \{e_1/x\}e_4)$ ok

But from these if follows by the let rule (S5) that

$$\Gamma \vdash \texttt{let}(\{e_1/x\}e_3, y.(\{e_1/x\}e_4))\ \texttt{ok}$$

Hence $\Gamma \vdash \{e_1/x\}e_2$ 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.

Here is the simple version: Let P(e) be a predicate over Arith expressions.

$$\begin{array}{rcl}
\forall n.P(\texttt{num}(n))\ \& & & \\
\forall x.P(\texttt{var}(x))\ \& & & \\
\forall e_1.\forall e_2.\,(P(e_1)\&P(e_2) & \Rightarrow & P(\texttt{plus}(e_1, e_2)))\ \& \\
\forall e_1.\forall e_2.\,(P(e_1)\&P(e_2) & \Rightarrow & P(\texttt{times}(e_1, e_2)))\ \& \\
\forall e_1.\forall e_2.\,(P(e_1)\&P(e_2) & \Rightarrow & P(\texttt{let}(e_1, x.e_2))) \\
\Rightarrow\ \forall e.\,P(e) & &
\end{array}$$

The problem with this formulation is that it may not work for a predicate P which depends on taking proper account of free variables. Note that the last clause for let expressions has an inductive hypothesis $P(e_2)$ where e_2 has an additional free variable.

In some cases, one needs to design the predicate so that it relates an epression to a context Γ of free variables. Even this approach may not succeed – take the previous problem as an example.

Most of our proofs will involve inductions not directly over the structure of expressions, but over derivations of judgments for typing or transition relations, which relate expressions to contexts, or closed expressions to other closed expressions.

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 $\texttt{oper}(o, e_1, e_2)$ with a new syntactic category

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

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

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.

New static rule:

$$\text{(S3/S4)} \qquad \frac{\Gamma \vdash e_1 \ \texttt{ok} \quad \Gamma \vdash e_2 \ \texttt{ok}}{\Gamma \vdash \texttt{oper}(o, e_1, e_2) \ \texttt{ok}}$$

New dynamic rules:

$$\text{(E1/E2)} \qquad \frac{(p = M(o)(m, n))}{\texttt{oper}(o, \texttt{num}(m), \texttt{num}(n)) \mapsto \texttt{num}(p)}$$

$$\text{(E4/E6)} \qquad \frac{e_1 \mapsto e_1'}{\texttt{oper}(o, e_1, e_2) \mapsto \texttt{oper}(o, e_1', e_2)}$$

$$\text{(E5/E7)} \qquad \frac{e_2 \mapsto e_2'}{\texttt{oper}(o, \texttt{num}(n), e_2) \mapsto \texttt{oper}(o, \texttt{num}(n), e_2')}$$

where as suggested, M(o) denotes the mathematical operation that the operator symbol o represents, e.g. it maps "+" to the plus operation on natural numbers. The (E1/E2) rule replaces the instruction rules for plus and times. The other two rules replace pairs of search rules for plus and times.

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.

See the file arith-SOS-oper.sml (linked from the class web page).