CSCI 384 Fall 2021: Programming Languages
Homework #8

Due: December 13, 2021, 11:59pm

1. For this problem you will write some Standard ML code. Consider a subset of MiniML's abstract syntax with only the constructors plus, times, and num. Recall our stack-based rules we gave for evaluating terms:

$$\begin{aligned}
\Sigma \rhd \mathsf{num}(n) &\longrightarrow \Sigma \lhd \mathsf{num}(n) \\
\Sigma \rhd \mathsf{plus}(e_1, e_2) &\longrightarrow \Sigma; \mathsf{plus}(\circ, e_2) \rhd e_1 \\
\Sigma; \mathsf{plus}(\circ, e) \lhd v &\longrightarrow \Sigma; \mathsf{plus}(v, \circ) \rhd e \\
\Sigma; \mathsf{plus}(\mathsf{num}(n_1), \circ) \lhd \mathsf{num}(n_2) &\longrightarrow \Sigma \lhd \mathsf{num}(n_1 + n_2) \\
\Sigma \rhd \mathsf{times}(e_1, e_2) &\longrightarrow \Sigma; \mathsf{times}(\circ, e_2) \rhd e_1 \\
\Sigma; \mathsf{times}(\circ, e) \lhd v &\longrightarrow \Sigma; \mathsf{times}(v, \circ) \rhd e \\
\Sigma; \mathsf{times}(\mathsf{num}(n_1), \circ) \lhd \mathsf{num}(n_2) &\longrightarrow \Sigma \lhd \mathsf{num}(n_1 \cdot n_2)
\end{aligned}$$

   In the above, we use $\circ$ to represent a "hole". The sequence Σ acts as a stack of these hole terms with a single hole. Let's work to express these rules in Standard ML.
   • **Start with** this definition of a datatype for these MiniML terms:

```
datatype term =
  plus of term * term
| times of term * term
| num of int
```

   • **Now develop a datatype** `hterm` that models a term with a hole. It should have four cases: an addition where there is a hole on the left, an addition where there is a hole on the right, a multiplication where there is a hole on the left, and a multiplication where there is a hole on the right. (*Hint:* the hole should not be represented in these. Instead, each should have one subterm with no holes.)
   • **Now add this datatype definition** to represent the $\rhd$ and $\lhd$ symbols in the stack-based semantics.

```
datatype does = awaits | returns
```

   • **Finally, write a function** `stackStep` that performs a stack-based semantics step on this subset of MiniML. The function should be of type `[hterm] -> does -> term -> ([hterm] * does * term)`. Its first argument is a stack of terms with holes, expressed as a list with that stack's terms laid out in reverse order. The head term in the list is the last one pushed onto the stack. The second argument gives the direction of information flow: is the stack awaiting a result? Or is the stack being sent a result? The third argument is the term right of that direction. It is either the term whose ultimate result is awaited by the stack, or the one being returned back to the stack. Finally, the triple returned by `stackStep` is the same three things: a stack of terms with holes, a `does` direction, and the term being waited for or returned back.

2. Consider adding abstract syntax terms nil, cons(e_1, e_2), head(e), tail(e), and null(e) to our MiniML terms. These correspond to the construction of an empty list, construction of a list from a head element and a tail list, extracting the head or the tail from a list value, and checking whether or not a list value is empty.

   Extend our stack-based semantics to include these terms. To do this, you need to specify three things:

   - The rules that dictate whether a term using these constructs is already a value.
   - The new kinds of terms with holes (i.e. contexts) that can occur during evaluation.
   - The rules for rewriting a stack context that awaits or returns a value.

3. Consider adding abstract syntax terms rec(f, x, r) to our MiniML terms. This describes a recursive function that takes a parameter x and then refers to itself as f inside its defining rule r. This means that f and x are variables that normally appear freely within r. As an example, the term

   $$\mathsf{rec}(c, n, \mathsf{if}(\mathsf{equals}(n, \mathsf{num}(0)), \mathsf{nil}, \mathsf{cons}(n, \mathsf{apply}(c, \mathsf{minus}(n, \mathsf{num}(1))))))$$

   describes a recursive function c that builds a list of values from n down to 1. Its defining rule is like the SML code

   ```
   if n=0 then nil else n::(c (n-1))
   ```

   Give the stack-based semantics for this term. In the semantics a rec should be a value term, like an fn term is. This means that your term rewrite rule(s) should describe how apply behaves when its left subterm is a rec term.

4. Give programs whose types correspond to this facts in classical logic:

   $$(A \wedge B) \Rightarrow ((\neg A \vee \neg B) \Rightarrow \bot)$$
   $$(A \vee B) \Rightarrow ((\neg A \wedge \neg B) \Rightarrow \bot)$$
   $$(\neg A \Rightarrow A) \Rightarrow A$$

   BONUS: also try

   $$\neg(A \wedge B) \Rightarrow (\neg A \vee \neg B)$$
   $$\neg(A \vee B) \Rightarrow (\neg A \wedge \neg B)$$