Homework #2

CS231

Due by the end of the day on October 14. You should submit three files:
`hw2a.ml`, `hw2b.ml`, and `hw2.pdf`.

Remember that you are encouraged to work in pairs on homework assignments. See the course syllabus for details. Also remember the course's academic integrity policy. In particular, you must credit other people and other resources that you consulted. Again, see the syllabus for details.

1. Like C, OCaml includes terms of the form $\texttt{t}_1$ `&&` $\texttt{t}_2$, which represent short-circuited "and" for booleans: the second boolean expression is evaluated only if necessary. In this problem we add terms of this form to our language of booleans and numbers (see the Homework #2 Cheat Sheet).

   (a) Add rules to the small-step operational semantics to support the new kind of term. You should give a *direct* semantics for this new term rather than simply "desugaring" it to some other kind of term (we'll do that later).

   (b) Add rules to the type system to support the new operator. Give a name to each new rule.

   (c) Provide only the cases specific to the new $\texttt{t}_1$ `&&` $\texttt{t}_2$ term for the proof of the Progress theorem:

   Theorem: If $\texttt{t} : \texttt{T}$, then either $\texttt{t}$ is a value or there exists some term $\texttt{t}'$ such that $\texttt{t} \longrightarrow \texttt{t}'$.

   Assume that the proof is performed by induction on the derivation of $\texttt{t} : \texttt{T}$. If your proof uses a Canonical Forms lemma, state the lemma clearly before your proof (but you need not prove the lemma).

   (d) Provide only the cases specific to the new $\texttt{t}_1$ `&&` $\texttt{t}_2$ term for the proof of the Preservation theorem:

   Theorem: If $\texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t}' : \texttt{T}$.

   Assume that the proof is performed by induction on the derivation of $\texttt{t} : \texttt{T}$.

2. (a) It turns out that terms of the form $\texttt{t}_1$ `&&` $\texttt{t}_2$ can be treated as "syntactic sugar," or shorthands for other terms in the language of booleans and numbers. Demonstrate this by providing a new operational semantics for these terms, which consists of a single rule with no premises.

   (b) Consider an eager version of `&&`, in which both operands are evaluated (in order from left to right) before producing the overall value of the term. Is this version still a syntactic sugar? If so, provide the semantics. If not, just say so.

3. Consider each of the following changes to the simple language of booleans and integers. For each change, say whether it invalidates Progress, Preservation, both, or neither. Also provide a counterexample to each invalidated theorem.

   (a) Remove the rule E-IfFalse.

   (b) Add the following axiom to the type system:

$$\frac{}{\texttt{0} : \texttt{Bool}}$$

(c) Add the following axiom to the operational semantics:

$$\frac{}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow t_2}$$

(d) Add the following rules:

$$\frac{}{\texttt{false + false} \longrightarrow \texttt{false}}$$

$$\frac{}{\texttt{true + true} \longrightarrow \texttt{true}}$$

$$\frac{t_1\texttt{:Bool} \qquad t_2\texttt{:Bool}}{t_1 \texttt{ + } t_2\texttt{:Int}}$$

(e) Add the following rules:

$$\frac{}{\texttt{if 0 then } t_2 \texttt{ else } t_3 \longrightarrow t_2}$$

$$\frac{t_1\texttt{:Int} \qquad t_2\texttt{:T} \qquad t_3\texttt{:T}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3\texttt{:T}}$$

4. Consider again the language of just booleans and integers.

   (a) Consider this "reverse progress" theorem:
   Theorem: If $\texttt{t}' : \texttt{T}$, then there exists some term $\texttt{t}$ such that $\texttt{t} \longrightarrow \texttt{t}'$.
   Is this a true theorem? If so, prove it. If not, give a counterexample.

   (b) Consider this "reverse preservation" theorem:
   Theorem: If $\texttt{t}' : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\texttt{t} : \texttt{T}$.
   Is this a true theorem? If so, prove it. If not, give a counterexample.

5. Implement the call-by-value lambda calculus (see the Homework #2 Cheat Sheet) in OCaml. See the file `hw2a.ml` for details.

6. Implement the call-by-value lambda calculus in OCaml again, now using a fancy technique called *higher-order abstract syntax*. See the file `hw2b.ml` for details.

7. In class we saw the rules for multi-step evaluation:

$$\frac{}{\texttt{t} \longrightarrow^* \texttt{t}} \qquad \text{(E-REFL)}$$

$$\frac{\texttt{t} \longrightarrow \texttt{t}'}{\texttt{t} \longrightarrow^* \texttt{t}'} \qquad \text{(E-STEP)}$$

$$\frac{\texttt{t} \longrightarrow^* \texttt{t}'' \qquad \texttt{t}'' \longrightarrow^* \texttt{t}'}{\texttt{t} \longrightarrow^* \texttt{t}'} \qquad \text{(E-TRANS)}$$

Prove the following theorem for the call-by-value lambda calculus, which says that the term known as Ω only multi-steps to itself.

Theorem: If `((function x -> x x) (function x -> x x))` $\longrightarrow^*$ `t`,
then `t` = `((function x -> x x) (function x -> x x))`.