Homework #1

CS231

Due by the end of the day on January 21. You should submit two files:
`hw1.ml` for Problem 1, and `hw1.pdf` for the rest.

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.

Latex'ed solutions are preferred. To facilitate this, I've posted Benjamin Pierce's style file `bcprules.sty` with this homework, which has macros for formatting inference rules, along with an example of how to use it in `example.tex`.

1. Recall the small-step operational semantics for the simple language of booleans and integers from class:

$$\frac{}{\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2} \qquad \text{(E-IfTrue)}$$

$$\frac{}{\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3} \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

$$\frac{n_1 \; [\![+]\!] \; n_2 = n}{n_1 + n_2 \longrightarrow n} \qquad \text{(E-Plus)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1 + t_2 \longrightarrow t_1' + t_2} \qquad \text{(E-Plus1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1 + t_2 \longrightarrow v_1 + t_2'} \qquad \text{(E-Plus2)}$$

$$\frac{n_1 \; [\![>]\!] \; n_2 = v}{n_1 > n_2 \longrightarrow v} \qquad \text{(E-GT)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1 > t_2 \longrightarrow t_1' > t_2} \qquad \text{(E-GT1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1 > t_2 \longrightarrow v_1 > t_2'} \qquad \text{(E-GT2)}$$

In the `hw1.ml` file I've defined the type `t` of terms for this language. I've also provided a function `isval` to determine whether a term is a value. Your job is to implement two functions described below; currently these functions just raise an `ImplementMe` exception.

(a) Implement the function `step`, which takes one step of execution on a given term. In other words, `(step t)` should return `t'` if and only if `t` $\longrightarrow$ `t'` according to our small-step semantics above. Your function should raise the `NormalForm` exception if `t` is already in normal form (i.e., `t` cannot step according to our small-step semantics).

(b) Implement the function `eval` which uses your `step` function above to execute a given term `t` until it reaches a normal form (either a value or a stuck expression). The `eval` function should return the final normal form term that is reached.

You can use OCaml's `assert` function to test your code. For example, `assert(step(If(True, False, True)) = False)` will be silent if the specified equality holds and will signal an assertion failure otherwise. So you can put a bunch of these asserts at the bottom of your file to serve as a test suite.

2. Consider this program P:

```
if (1 + 2) > 3 then 4+5 else 6+7
```

(a) Provide a derivation tree, using our rules above, for one step of execution of the program P.

(b) List the sequence of terms that P steps to according to our rules, up to and including the final normal form. You do not need to provide the derivation trees for each step.

3. Consider the subset of the language above that only contains booleans (also in Figure 3-1 of the book):

```
t  ::=  true | false | if t then t else t
v  ::=  true | false
```

$$\frac{}{\texttt{if true then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_2} \qquad \text{(E-IFTRUE)}$$

$$\frac{}{\texttt{if false then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{t}_3} \qquad \text{(E-IFFALSE)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \longrightarrow \texttt{if } \texttt{t}_1' \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3} \qquad \text{(E-IF)}$$

Prove the following theorem, which is often called a "progress" theorem, since it says that terms that are not values can always make progress in their evaluation.

Theorem: For every term `t`, either `t` is a value or there exists a term `t'` such that `t` $\longrightarrow$ `t'`.

Clearly state your induction hypothesis before the proof.

4. Suppose we want to change the evaluation strategy of our language of booleans above so that the `then` and `else` branches of an `if` expression are evaluated (in that order) before the guard is evaluated. Provide a new small-step operational semantics for the language that has this behavior.

5. Given the original small-step semantics for booleans defined in Problem #3 above, prove the following theorem, which says that the semantics is deterministic.

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

   The notion of equality used above is *syntactic equality*: $\texttt{t} = \texttt{t}'$ if and only if $\texttt{t}$ and $\texttt{t}'$ are the exact same term.

   Clearly state your induction hypothesis before the proof.

6. Consider the original small-step semantics for the language of booleans and integers, as well as a version with booleans modified as in Problem #4 above. Recall that a term is *stuck* if it is a normal form but is not a value.

   (a) Are there any terms that are stuck in the original semantics that are not stuck in the modified version? If yes, provide one such term. If no, just say so.

   (b) Are there any terms that are stuck in the modified semantics that are not stuck in the original version? If yes, provide one such term. If no, just say so.

7. Consider the original small-step semantics for the language of booleans and integers, as well as a version with booleans modified as in Problem #4 above. Recall that a term $\texttt{t}$ is *eventually stuck* if there exists a term $\texttt{t}'$ such that $\texttt{t} \longrightarrow^* \texttt{t}'$ and $\texttt{t}'$ is stuck.

   (a) Are there any terms that are *eventually stuck* in the original semantics that are not eventually stuck in the modified version? If yes, provide one such term. If no, just say so.

   (b) Are there any terms that are *eventually stuck* in the modified semantics that are not eventually stuck in the original version? If yes, provide one such term. If no, just say so.