Homework #1

CS231

Due by the end of the day on October 7. 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. This style file has nice macros for formatting inference rules. The file contains several examples at the top that illustrate the usage of these macros.

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

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

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

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

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

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

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

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

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

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

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 $\mathtt{t}'$ if and only if $\mathtt{t} \longrightarrow \mathtt{t}'$ 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.

2. 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
```

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

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

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

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 the abstract machine.

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

Clearly state your induction hypothesis before the proof.

Induction hypothesis: For every term $\mathtt{t_0}$ that is a subterm of `t`, either $\mathtt{t_0}$ is a value or there exists a term $\mathtt{t_0}'$ such that $\mathtt{t_0} \longrightarrow \mathtt{t_0}'$.

Proof: By structural induction on `t`. Case analysis of the form of `t`.

- Case `t` is `true`. Then `t` is a value.
- Case `t` is `false`. Then `t` is a value.
- Case `t` has the form $\mathtt{if\ t_1\ then\ t_2\ else\ t_3}$. By the induction hypothesis, either $\mathtt{t_1}$ is a value or there exists some $\mathtt{t_1'}$ such that $\mathtt{t_1} \longrightarrow \mathtt{t_1'}$. If $\mathtt{t_1}$ is a value, then by the definition of values $\mathtt{t_1}$ is either `true` or `false`. If the former, then by E-IfTrue we have $\mathtt{t} \longrightarrow \mathtt{t_2}$. If the latter, then by E-IfFalse we have $\mathtt{t} \longrightarrow \mathtt{t_3}$. Otherwise, there exists some $\mathtt{t_1'}$ such that $\mathtt{t_1} \longrightarrow \mathtt{t_1'}$. Then by E-If we have $\mathtt{t} \longrightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}$.

3. 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.

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

$$\frac{\mathtt{t}_3 \longrightarrow \mathtt{t}_3'}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{t}_3 \longrightarrow \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{t}_3'}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{v}_3 \longrightarrow \mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{v}_3}$$

$$\frac{}{\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{v}_3 \longrightarrow \mathtt{v}_2}$$

$$\frac{}{\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{v}_3 \longrightarrow \mathtt{v}_3}$$

4. Given the original small-step semantics for booleans defined in Problem #2 above, prove the following theorem, which says that the semantics is deterministic. The notion of equality used below is syntactic equality of terms.

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

   Clearly state your induction hypothesis before the proof.

   Induction Hypothesis: If $\mathtt{t}_0 \longrightarrow \mathtt{t}_0'$ and $\mathtt{t}_0 \longrightarrow \mathtt{t}_0''$ and $\mathtt{t}_0 \longrightarrow \mathtt{t}_0'$ is a subderivation of $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\mathtt{t}_0' = \mathtt{t}_0''$.

   Proof: By induction on the derivation of $\mathtt{t} \longrightarrow \mathtt{t}'$. Case analysis on the last rules used in the derivations of $\mathtt{t} \longrightarrow \mathtt{t}'$ and $\mathtt{t} \longrightarrow \mathtt{t}''$. (We could do a case analysis on one of them, and then an inner case analysis on the other, but this way will be more succinct.)

   - Case both derivations end with the rule E-IfTrue. Then $\mathtt{t}$ has the form $\mathtt{if\ true\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{t}'' = \mathtt{t}_2$.
   - Case both derivations end with the rule E-IfFalse. Then $\mathtt{t}$ has the form $\mathtt{if\ false\ then\ t_2\ else\ t_3}$ and $\mathtt{t}' = \mathtt{t}'' = \mathtt{t}_3$.
   - Case both derivations end with the rule E-If. Then $\mathtt{t}$ has the form $\mathtt{if\ t_1\ then\ t_2\ else\ t_3}$ and $\mathtt{t}_1 \longrightarrow \mathtt{t}_1'$ and $\mathtt{t}'$ has the form $\mathtt{if\ t_1'\ then\ t_2\ else\ t_3}$ and $\mathtt{t}_1 \longrightarrow \mathtt{t}_1''$ and $\mathtt{t}''$ has the form $\mathtt{if\ t_1''\ then\ t_2\ else\ t_3}$. By the induction hypothesis, $\mathtt{t}_1' = \mathtt{t}_1''$, so also $\mathtt{t}' = \mathtt{t}''$.
   - Case the derivations end with different rules. If one derivation uses E-IfTrue and the other uses E-IfFalse then we have that $\mathtt{t}$ has the form $\mathtt{if\ true} \cdots$ as well as $\mathtt{if\ false} \cdots$, which is a contradiction. If one derivation uses E-IfTrue and the other uses E-If then we have that $\mathtt{t}$ has the form $\mathtt{if\ true} \cdots$ and $\mathtt{true} \longrightarrow \mathtt{t}_1'$ for some term $\mathtt{t}_1'$. Since $\mathtt{true}$ does not step according to any of our rules, we have a contradiction. Finally, if one derivation uses E-IfFalse and the other uses E-If then we have that $\mathtt{t}$ has the form $\mathtt{if\ false} \cdots$ and $\mathtt{false} \longrightarrow \mathtt{t}_1'$ for some term $\mathtt{t}_1'$. Since $\mathtt{false}$ does not step according to any of our rules, we have a contradiction.

5. For the full language of booleans and integers shown in Problem #1 above, provide a BNF grammar for a new metavariable s that characterizes exactly the stuck expressions. You can introduce other metavariables as needed.

```
s  ::=  if n then t else t | if s then t else t
     |  s + t | v + s | b + v | v + b
     |  s > t | v > s | b > v | v > b
b  ::=  true | false
```

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 #3 above.

(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.

Yes. An example is `if 0 then 1+2 else true`.

(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.

Yes. An example is `if true then 0 else (1 + true)`.

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 #3 above.

(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.

No.

(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.

Yes. An example is `if true then 0 else (1 + true)`.