Homework #3

CS231

Due by the end of the day on February 11. You should submit three OCaml files and one PDF file.

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. Implement the operational semantics for the untyped lambda calculus — the language of Section 2 of the cheat sheet, but without the type annotation on formal parameters — augmented with the `true` value, in OCaml. See the file `hw3a.ml` for details.

2. Implement the operational semantics for that same language in OCaml again, now using a fancy technique called *higher-order abstract syntax*. See the file `hw3b.ml` for details.

3. Implement a typechecker for the simply-typed lambda calculus augmented with the `true` value, using our original OCaml representation of terms from Problem 1. See the file `hw3c.ml` for details.

4. Consider the simply typed lambda calculus augmented with our language of booleans and numbers. For each term `t` below, provide a Γ and `T` such that $\Gamma \vdash$ `t : T` and provide the associated derivation tree. If there is no such Γ and `T`, just say so. Here both `f` and `x` denote variables.

   (a) `f (if x then 0 else 1)`

   (b) `f (if (f x) then 0 else 1)`

   (c) `f (if (f x) then (f 0) else 1)`

5. Consider the simply-typed lambda calculus in Section 2 of the cheat sheet, and assume it is augmented to also include the type `Bool` (but no new terms or values). We include the type `Bool` just so that the grammar of types is not vacuous.

   (a) Prove the Progress theorem:
   Theorem: If $\emptyset \vdash$ `t : T`, then either `t` is a value or there exists some term `t'` such that `t` $\longrightarrow$ `t'`.
   Note that $\emptyset$ denotes the empty type environment.

   (b) Suppose we allow an arbitrary type environment in the Progress theorem:
   Theorem: If $\Gamma \vdash$ `t : T`, then either `t` is a value or there exists some term `t'` such that `t` $\longrightarrow$ `t'`.
   Is the theorem still true? If so, just say so. If not, provide a counterexample.

   (c) Prove the Preservation theorem:
   Theorem: If $\emptyset \vdash$ `t : T` and `t` $\longrightarrow$ `t'`, then $\emptyset \vdash$ `t' : T`.
   Your proof may make use of the following *type substitution* lemma, which formalizes the correctness of modular typechecking: it suffices to typecheck a function body once, and that ensures that the function body is type-correct for all possible instantiations of the function's formal parameter.
   Lemma: If Γ, `x : T1` $\vdash$ `t : T2` and $\Gamma \vdash$ `v : T1`, then $\Gamma \vdash$ `[x` $\mapsto$ `v]t : T2`.

6. Consider the simply typed lambda calculus (Section 2 of the cheat sheet) with integers and booleans (Section 1 of the cheat sheet), and we'll also add the unit value `()`, the type `Unit`, and this typing rule:

$$\frac{}{\texttt{() : Unit}}$$

In this problem you will add a new term `for` $\texttt{t}_1$ `do` $\texttt{t}_2$ for bounded loops. The semantics of this construct is as follows: $\texttt{t}_1$ should evaluate to a number `n`, and $\texttt{t}_2$ should be executed `n` times if `n` is nonnegative and 0 times otherwise. We assume that $\texttt{t}_2$ is just executed for its side effects, so it (and the entire loop) should have type `Unit`.

Define the small-step operational semantics for this new term. Also define a typing rule for the new term.