CS101 Homework 2

Due: Friday, Oct 21, 1PM (sharp). Last updated: October 17, 2005 14 : 56

Collaboration: You are allowed and encouraged to work together and collaborate with others. However, your submission must be your own; you must write up your homework without referring to material developed with other groups.

You may use the WWW for reference material. You may use the material you found to develop your understanding, but your submission must be your own.

Summary: you may use any and all resources at your disposal, but your submission must be your own work.

Part I: λ-Calculus

1. Write a proof of the (*Apply*) case of the induction step of the substitution lemma. Try to be as precise as possible and give a detailed and rigorous proof — at least as detailed as the (*Fun*) case writeup available from the course web page.

2. Suppose we extend the language of the λ-Calculus with booleans. Namely, the types are extended with `bool`, and the expressions are extended with constants `true`, and `false`, as well as the "`if` e_1 `then` e_2 `else` e_3" operation.

a. Write the large-step evaluation rule(s) that need to be added to λ-Calculus in order to support booleans.

b. Write the small-step evaluation rule(s) (in style that was presented in Lecture 3) that need to be added to λ-Calculus in order to support booleans.

c. Rewrite the small-step evaluation rule(s) that you wrote in part (b) in the "evaluation environment" style (reminder: this was presented in Lecture 4). Namely, write how the definition of the evaluation environment needs to change and what rule(s) need to be added.

How to submit: Bring it to class on Friday. If you are unable to come to class, put it into Aleksey Nogin's mailbox on the second floor of Jorgensen.

Part II: OCaml

Start by downloading the code stubs file `homework2.tgz` and unpacking it. Make sure you can compile it by running either `make` or `omake` (OMake is a build system developed by Jason Hickey with some contributions from me — see `http://omake.metaprl.org/` if you are interested in more information.)

In this homework you should be modifying the `cs101_hw2.ml` file. You are not allowed to modify any other files, except for extending the `lc_error` type in the `cs101_hw2_types.ml` file.

3. Write a λ-Calculus type checker (or, rather, type discovery) function `lc_typecheck`. Represent the Γ typing environment with a value of type `type_environment` — in other words, use the `VarMap` module defined for you to maintain the map from variable names to types. Note that if you use the `VarMap` module correctly, it will take care of variable shadowing without any extra effort on your part.

Report all type errors by raising the `LcError` exception. Extend the `lc_error` type as necessary for accommodating all the type errors that may occur.

Hint:

```
let rec lc_get_type gamma e =
   match e with
      ExpInt _ ->
         TyInt
    | ExpLambda (v, t, e) ->
         let gamma' = VarMap.add v t gamma in
         let ty_res = lc_get_type gamma' e in
            TyFun(t, ty_res)
    | ...

let lc_type_check = lc_get_type VarMap.empty
```

4. Write an evaluator (interpreter) for the λ-Calculus. When evaluating λ-Calculus expressions, instead of using an explicit implementation of substitution, maintain an evaluation environment, mapping from variable names to values (in the code, use maps of the type `value_environment` similar to how you have used the `type_environment` in problem 3) instead of performing substitutions explicitly. Each functional value will need to be kept together with its appropriate environment in a "closure". This corresponds to the following evaluation semantics:

$$\frac{}{\Gamma \vdash n \longrightarrow \vdash n}$$

$$\frac{}{\Gamma \vdash \lambda x : t.e \longrightarrow \Gamma \vdash \lambda x : t.e}$$

$$\frac{\Gamma \vdash e_1 \longrightarrow \Gamma' \vdash n_1 \quad \Gamma \vdash e_2 \longrightarrow \Gamma'' \vdash n_2 \quad n = n_1 \; op \; n_2}{\Gamma \vdash e_1 \; op \; e_2 \longrightarrow \vdash n}$$

$$\frac{\Gamma \vdash e_1 \longrightarrow \Gamma_1 \vdash \lambda x : t.e_1' \quad \Gamma \vdash e_2 \longrightarrow \Gamma_2 \vdash v \quad \Gamma_1; x = (\Gamma_2 \vdash v) \vdash e_1' \longrightarrow \Gamma_3 \vdash v'}{\Gamma \vdash (e_1 \; e_2) \longrightarrow \Gamma_3 \vdash v'}$$

Here Γ and Δ stand for arbitrary sequences of the form $x_1 = (\Gamma_1 \vdash v_1); x_2 = (\Gamma_2 \vdash v_2); \ldots x_n = (\Gamma_n \vdash v_n)$ that map variables to values; $\Gamma \vdash e$ represents an expression e (that may contain free variables) whose variables are mapped to values according to Γ.

Make sure your evaluator starts by making sure that the expression is well-typed. After that, handle all errors cases by raising the `InternalError` exception (the type safety theorem for the λ-Calculus guarantees that these cases should never be actually encountered, unless your code has a bug).

The evaluation semantics stated above returns a "closure" in case of a functional value. This means that you will need to expand the closure by performing all the relevant substitutions in order to turn in back into `lc_exp`.

How to submit: Make sure your code compiles without any warnings (**non-compiling code or code that compiles with warnings will not receive any credit**). Then email the `cs101_hw2.ml` and `cs101_hw2_types.ml` files containing your solutions to `nogin@cs.caltech.edu`.