HW6 – Functional Languages

CS 476, Fall 2023

1 Instructions

Begin by downloading the file `hw6-base.ml` from the course website and renaming it to `hw6.ml`. Then fill in your answers to the problems, adding or modifying definitions as you see fit. Some questions will ask you to solve a problem and put your answer in the comments, while others will ask you to write some code. Submit your completed `hw6.ml` via Gradescope. As always, please don't hesitate to ask for help on Piazza (`https://piazza.com/class/lkwp62qwo734i9`).

2 Evaluating Functional Programs

The file `hw6-base.ml` defines a type `exp` of expressions for a simple OCaml-like language. It also defines a function `eval` that takes an expression and returns the evaluated version of the expression. In other words, `eval` e should return `Some` v exactly when $e \Downarrow v$. In this assignment, you will experiment with the `eval` function and complete it using the semantics of the language.

1. (3 points) Use the semantics of lambda calculus to evaluate each of the following terms for as many steps as possible. Write your answers in the space provided in the comments near the bottom of `hw6.ml`.

   (a) $(\lambda x.\ x)\ y$

   (b) $(\lambda x.\ (\lambda y.\ x))\ z$

   (c) $(\lambda x.\ (\lambda y.\ y)\ x)\ (\lambda z.\ z)$

2. (3 points) Define variables `expa`, `expb`, and `expc` of type `exp` that correspond to terms (a), (b), and (c) respectively in our simple functional programming language. It may help to refer to the provided definition of `lam1`, which represents the lambda-term $\lambda x.\ (\lambda y.\ x\ y)$. Then use the `eval` function to check your answers to problem 1. If the results you get are different from what you wrote, see if you can figure out why. In the space provided in the comments, write any differences you noticed.

3. (3 points) The rules for sum values are:

$$\frac{e \Downarrow v}{\texttt{inl}\ e \Downarrow \texttt{inl}\ v} \qquad \frac{e \Downarrow v}{\texttt{inr}\ e \Downarrow \texttt{inr}\ v}$$

Add cases to `eval` for the `Inl` and `Inr` expressions according to these rules.

Once you have completed this problem, `eval (Inr (Add (Int 3, Int 4)))` should return `Some (Inr (Int 7))`.

4. (6 points) The rules for match expressions are:

$$\frac{e \Downarrow \texttt{inl}\ v \qquad [x_1 \mapsto v]e_1 \Downarrow v'}{(\texttt{match}\ e\ \texttt{with inl}\ x_1\ \texttt{->}\ e_1\ \texttt{|}\ \texttt{inr}\ x_2\ \texttt{->}\ e_2) \Downarrow v'}$$

$$\frac{e \Downarrow \texttt{inr}\ v \qquad [x_2 \mapsto v]e_2 \Downarrow v'}{(\texttt{match}\ e\ \texttt{with inl}\ x_1\ \texttt{->}\ e_1\ \texttt{|}\ \texttt{inr}\ x_2\ \texttt{->}\ e_2) \Downarrow v'}$$

where $[x \mapsto v]e$ is implemented by `subst x v e`. Add a case for the `Match` expression to `eval` according to these rules. Make sure that your case implements both rules, depending on the constructor of the condition.

Once you have completed this problem, `eval (Match (Inr (Bool false), "i", Var "i", "b", If (Var "b", Int 1, Int 0)))` should return `Some (Int 0)`.

5. (for graduate students) The rules for tuples and their associated functions are:

$$\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{(e_1, e_2) \Downarrow (v_1, v_2)} \qquad \frac{e \Downarrow (v_1, v_2)}{\texttt{fst}\ e \Downarrow v_1} \qquad \frac{e \Downarrow (v_1, v_2)}{\texttt{snd}\ e \Downarrow v_2}$$

Add constructors for tuple expressions (`Tuple`, `Fst`, and `Snd`) to the `exp` type. Then add cases to `vars` and `subst` for tuples, and extend `eval` to handle tuples according to the rules.

Once you have completed this problem, `eval (Tuple (Snd (Tuple (Int 5, Bool true)), Fst (Tuple (Int 5, Bool true))))` should return `Some (Tuple (Bool true, Int 5))`.