CS51 PS06 Writeup

Matthew Chung

May 3 2022

In the final project, I implemented a lexical evaluator for my extension. In many ways, the lexical evaluator was the same as the dynamical evaluator, so most of the expressions were implemented the same as the dynamical evaluator. These include Var, Unop, Binop, Conditional, Let, Num, Bool, Unassigned, and Raise. In particular, Unop, Binop, and Conditional were implemented with helpers that worked for all the evaluators, including substitution. The changes that had to be made were to Fun, Letrec, and App. In a lexical evaluator, a function is created with the environment at the time, and is ran with variables within it according to the environment when it was created. This differs from dynamical, as the function is ran with the current environment, not the one from the function's creation. In order to accomplish this, a Closure value was used in order to pair the expression along with the environment at creation when evaluating a Fun expression. The next change was to Letrec, where we begin by extending the environment with the function name and an Unassigned expression. After evaluating the function expression, we remap the evaluated function to the function name and then we can use that updated environment when evaluating the body. For App, we were now extracting the function from a Closure that contained the environment upon its creation, and we use that environment to evaluate the function application rather than the current environment.

The best way to demonstrate the difference between the lexical evaluator and the dynamical evaluator would be to use the example below.

```
let x = 1 in let f = fun y -> x + y in let x = 2 in f 3 ;;
Num(4)
```

Here is an expression that sets variable x to 1 and creates a function f that sums its input y and variable x and sets variable x to 2 and applies 3 to f. In a lexical evaluator, the function would be created with the environment where x is set to 1 and applying 3 to f would evaluate to 4, despite the current environment having x set to 2, since it is evaluated with x being 1, as that was the environment when the function was created.

```
let x = 1 in let f = fun y -> x + y in let x = 2 in f 3 ;;
Num(5)
```

However, in a dynamical evaluator with the exact same expression being evaluated, the result comes out to 5, since the function is evaluated with the current environment where x is set to 2, no longer 1. That concludes my lexical evaluator.