Code Review 9: April 17, 2020

Environment Semantics

Hakeem Angulu, hangulu@college.harvard.edu

Learning Goals

- Compare environment semantics to substitution semantics
- Understand the differences between dynamic scoping and lexical scoping
- Understand how refs are affected by the models

Overview

Up to this point, our study of semantics has focused on the **substitution model**, an excellent model for evaluation that is commonly used in programming theory. However, its treatment of variables, and in particular, references, makes it unsuitable for more complex evaluation.

The substitution model is eager: it substitutes for every occurrence of a variable, even if that occurrence is never used. Take:

```
let x = 42 in e
```

The substitution model must crawl through the entirety of `e` to find `x`. But, `x` may never appear, or may appear in a small branch of `e` that is never used in evaluation, wasting time.

An improvement over this is a lazy substitution, where the interpreter only substitutes for a value when the variable is needed for evaluation. This is the idea behind the **environment model**.

Environments

An **environment** is a mapping from variables to values. When a variable is encountered during the course of evaluation, its value is looked up in the environment.

Notation

`{x1 -> v1}` is the environment that binds `x1` to `v`. `E(x)` represents the binding of `x` in the environment. `E{x -> v}` represents the environment `E` with the variable `x` additionally bound to the value `v`.

Dynamic vs. Lexical Scoping

A central question in the use of environment semantics is, what mapping is in play during evaluation.

Exercise 1: Consider the below expression. What does it evaluate to?

```
let x = 5 in
let f = fun y -> x + y in
let x = 3 in
f 4;;
```

The value of the expression changes depending on whether we use Dynamic Environment Semantics or Lexical Environment Semantics. The central difference between these semantics is which environment is used during the application of a function.

In dynamic environment semantics, the function is evaluated using the environment at the time of application. In the above example, that means `x` will be `3`.

In lexical environment semantics, functions are evaluated in the environment at the time of function definition. In the above example, that means `x` will be `5`.

To achieve lexical environment semantics, we need to be able to take a snapshot of the world at the point in time at which a function is defined and store that snapshot in the environment along with the function. This pair, variable to function + snapshot mapping, is called a closure.

We define a full set of rules for lexical and dynamic scoping in the attached PDF.

Stores

To implement references, we need a model of memory.

Unlike the environment, this abstract model of memory, called a **store** can be modified without changing the mapping of variables to values. In evaluating references, we add to the environment a mapping from a variable to a location, and we add to the store a mapping from that location to value.

Expressions are evaluated in the context of an environment and a store. The semantics specify when the environment and store are modified. As evaluations can essentially always potentially modify the store, each step in an evaluation returns a (potentially) modified store. This new store is used in the succeeding step.

We define a full set of rules for lexical environment semantics with stores in the attached PDF. Stores could also be implemented with dynamic environment semantics, but we do not provide these semantics.

Practice

Exercise 2:

Evaluate the following using lexical environment semantics with stores.

```
let x = ref 42 in (x := !x - 21; !x) + !x ;;
```

The solutions to all of the above exercises will be available on Sunday at noon.

For this and future code reviews, please do not hesitate to reach out to me with any questions or concerns. My email can be found at the top of this document.