Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Environment Semantics; Axiomatic Semantics; Dependent Types
Section and Practice Problems

Section 11

1 Environment Semantics

For Homework 5, the monadic interpreter you will be using uses environment semantics, that is, the operational semantics of the language uses a map from variables to values instead of performing substitution. This is a quick primer on environment semantics.

An environment ρ maps variables to values. We define a large-step operational semantics for the lambda calculus using an environment semantics. A configuration is a pair $\langle e, \rho\rangle$ where expression e is the expression to compute and ρ is an environment. Intuitively, we will always ensure that any free variables in e are mapped to values by environment ρ.

The evaluation of functions deserves special mention. Configuration $\langle \lambda x.\, e, \rho\rangle$ is a function $\lambda x.\, e$, defined in environment ρ, and evaluates to the *closure* $(\lambda x.\, e, \rho)$. A closure consists of code along with values for all free variables that appear in the code.

The syntax for the language is given below. Note that closures are included as possible values and expressions, and that a function $\lambda x.\, e$ is *not* a value (since we use closures to represent the result of evaluating a function definition).

$$\begin{aligned} e &::= x \mid n \mid e_1 + e_2 \mid \lambda x.\, e \mid e_1\ e_2 \mid (\lambda x.\, e, \rho) \\ v &::= n \mid (\lambda x.\, e, \rho) \end{aligned}$$

Note than when we apply a function, we evaluate the function body using the environment from the closure (i.e., the lexical environment, ρ_{lex}), as opposed to the environment in use at the function application (the dynamic environment).

$$\frac{}{\langle x, \rho\rangle \Downarrow \rho(x)} \qquad \frac{}{\langle n, \rho\rangle \Downarrow n} \qquad \frac{\langle e_1, \rho\rangle \Downarrow n_1 \quad \langle e_2, \rho\rangle \Downarrow n_2}{\langle e_1 + e_2, \rho\rangle \Downarrow n}\ n = n_1 + n_2$$

$$\frac{}{\langle \lambda x.\, e, \rho\rangle \Downarrow (\lambda x.\, e, \rho)} \qquad \frac{\langle e_1, \rho\rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho_{lex}[x \mapsto v_2]\rangle \Downarrow v}{\langle e_1\ e_2, \rho\rangle \Downarrow v}$$

For convenience, we define a rule for let expressions.

$$\frac{\langle e_1, \rho\rangle \Downarrow v_1 \quad \langle e_2, \rho[x \mapsto v_1]\rangle \Downarrow v_2}{\langle \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2, \rho\rangle \Downarrow v_2}$$

(a) Evaluate the program $\mathsf{let}\ f = (\mathsf{let}\ a = 5\ \mathsf{in}\ \lambda x.\, a + x)\ \mathsf{in}\ f\ 6$. Note the closure that f is bound to.

(b) Suppose we replaced the rule for application with the following rule:

$$\frac{\langle e_1, \rho\rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho[x \mapsto v_2]\rangle \Downarrow v}{\langle e_1\ e_2, \rho\rangle \Downarrow v}$$

That is, we use the dynamic environment to evaluate the function body instead of the lexical environment.

What would happen if you evaluated the program $\mathsf{let}\ f = (\mathsf{let}\ a = 5\ \mathsf{in}\ \lambda x.\, a + x)\ \mathsf{in}\ f\ 6$ with this modified semantics?

2 Axiomatic semantics

(a) Consider the program

$$c \equiv \mathsf{bar} := \mathsf{foo}; \textbf{while } \mathsf{foo} > 0 \textbf{ do } (\mathsf{bar} := \mathsf{bar} + 1; \mathsf{foo} := \mathsf{foo} - 1).$$

Write a Hoare triple $\{P\}\ c\ \{Q\}$ that expresses that the final value of bar is two times the initial value of foo.

(b) Prove the following Hoare triples. That is, using the inference rules from Section 1.3 of Lecture 19, find proof trees with the appropriate conclusions.

(i) $\vdash \{\mathsf{baz} = 25\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}$

(ii) $\vdash \{\textbf{true}\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}$

(iii) $\vdash \{\mathsf{baz} + \mathsf{quux} = 42\}\ \mathsf{baz} := \mathsf{baz} - 5; \mathsf{quux} := \mathsf{quux} + 5\ \{\mathsf{baz} + \mathsf{quux} = 42\}$

(iv) $\vdash \{\textbf{true}\}\ \textbf{if } \mathsf{y} = 0 \textbf{ then } \mathsf{z} := 2 \textbf{ else } \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}$

(v) $\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while } \mathsf{y} > 0 \textbf{ do } \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{z} = 55\}$

(vi) $\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while } \mathsf{y} > 0 \textbf{ do } (\mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1)\ \{\mathsf{z} = 55\}$

3 Dependent Types

(a) Assume that boolvec has kind $(x : \textbf{nat}) \Rightarrow \textbf{Type}$ and init has type $(n : \textbf{nat}) \to \textbf{bool} \to \textbf{boolvec}\ n)$.
Show that the expression init 5 true has type **boolvec** 5,
That is, prove

$$\Gamma \vdash \mathsf{init}\ 5\ \mathsf{true} : \textbf{boolvec}\ 5$$

where

$$\Gamma = \mathsf{boolvec} :: (x : \textbf{nat}) \Rightarrow \textbf{Type}, \mathsf{init} : (n : \textbf{nat}) \to \textbf{bool} \to \textbf{boolvec}\ n.$$

(b) Show that the types **boolvec** $(35 + 7)$ and **boolvec** $((\lambda y : \textbf{nat}.\ y)\ 42)$ are equivalent.
That is, prove that

$$\Gamma \vdash \textbf{boolvec}\ (35 + 7) \equiv \textbf{boolvec}\ ((\lambda y : \textbf{nat}.\ y)\ 42) :: \textbf{Type}$$

where

$$\Gamma = \mathsf{boolvec} :: (x : \textbf{nat}) \Rightarrow \textbf{Type}.$$

(c) Suppose we had a function double that takes a **boolvec** and returns a **boolvec** that is twice the length. Write an appropriate type for double. (Note that you will need make sure that the type of the **boolvec** argument is well formed! Hint: take a look at the type of join, mentioned in the Lecture 20 notes, for inspiration.)

4 Coq and Dafny (Optional!)

Note that for this course we do not expect you to be deeply familiar with Dafny or Coq. So this section question is for those that are interested in finding out more about these tools.

You can play around with Dafny online at `https://rise4fun.com/dafny`. A tutorial (on which the class demo was based) is available at `https://rise4fun.com/Dafny/tutorial/Guide`.

The Coq website is `https://coq.inria.fr/`. The easiest way to install Coq is via opam, OCaml's package manager. See `https://coq.inria.fr/opam/www/using.html`. In lecture, Prof. Chong was using Proof General (an extension to Emacs) to interact with Coq: `https://proofgeneral.github.io/`.

The Software Foundations series (`https://softwarefoundations.cis.upenn.edu/`) is a programming-languages oriented introduction to using Coq.