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

Algebraic structures; Axiomatic semantics
Section and Practice Problems

Apr 3–6, 2018

---

1 Haskell

(a) Install the Haskell Platform, via `https://www.haskell.org/platform/`.

(b) Get familiar with Haskell. Take a look at `http://www.seas.harvard.edu/courses/cs152/2018sp/resources.html` for some links to tutorials.

In particular, get comfortable doing functional programming in Haskell. Write the factorial function. Write the append function for lists.

(c) Get comfortable using monads, and the bind syntax. Try doing the exercises at `https://wiki.haskell.org/All_About_Monads#Exercises` (which will require you to read the previous sections to understand `do` notation, and their previous examples).

(d) Also, look at the file `http://www.seas.harvard.edu/courses/cs152/2018sp/sections/haskell-examples.hs`, which includes some example Haskell code (that will likely be covered in Section).

2 Algebraic structures

(a) Show that the option type, with *map* defined as in the lecture notes (Lecture 18, Section 2.2) satisfy the functor laws.

(b) Consider the list type, τ **list**. Define functions *return* and *bind* for the list monad that satisfy the monad laws. Check that they satisfy the monad laws.

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

4 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?