University of Massachusetts Lowell — Comp 3010: Organization of Programming Languages

Comp 3010. OPL Practice Midterm

October 2024

---

1. Consider the following inference rules that define the relation $\rightsquigarrow$, which is a subset of $\mathbb{N} \times \mathbb{N}$.

$$\frac{a \rightsquigarrow b}{c \rightsquigarrow d}\, c = a + 1 \text{ and } d = b + 2 \qquad\qquad \frac{}{0 \rightsquigarrow 0}$$

Which *one* of the following statements is true?

a) For all $(a, b) \in \rightsquigarrow$, b is equal to $2^a - 1$.

b) For all $(a, b) \in \rightsquigarrow$, b is even.

c) The relation $\rightsquigarrow$ is reflexive.

d) For all $(a, b) \in \rightsquigarrow$, a is greater than or equal to b.

2. Consider the set **GG** given by the following inference rules.

$$\frac{}{\mathsf{raas} \in \mathbf{GG}} \qquad \frac{}{\mathsf{fusion} \in \mathbf{GG}} \qquad \frac{d \in \mathbf{GG} \quad e \in \mathbf{GG}}{(\mathsf{senior}, d, e) \in \mathbf{GG}}$$

Suppose we want to prove that some property P holds of every element of **GG**.

Which *one* of the following is an appropriate inductive reasoning principle?

a) If $\forall a_1, a_2 \in \mathbf{GG}.\ P(a_1) \wedge P(a_2) \implies P((\mathsf{senior}, a_1, a_2))$ then $\forall a \in \mathbf{GG}.\ P(a)$.

b) If $P(\mathsf{raas})$ and $P(\mathsf{fusion})$ and $P((\mathsf{senior}, \mathsf{raas}, \mathsf{fusion}))$ then $\forall a \in \mathbf{GG}.\ P(a)$.

c) If $P(\mathsf{raas})$ and $P(\mathsf{fusion})$ and $\forall a_1, a_2 \in \mathbf{GG}.\ P(a_1) \wedge P(a_2) \implies P((\mathsf{senior}, a_1, a_2))$ then $\forall a \in \mathbf{GG}.\ P(a)$.

d) If $\forall a_1, a_2 \in \mathbf{GG}.\ P(\mathsf{raas}) \wedge P(\mathsf{fusion}) \wedge P(a_1) \wedge P(a_2) \implies P((\mathsf{senior}, a_1, a_2))$ then $\forall a \in \mathbf{GG}.\ P(a)$.

3. Let $\longleftrightarrow^*$ be the reflexive symmetric transitive closure of the small-step IMP relation $\longrightarrow$. That is, $\longleftrightarrow^*$ is the smallest relation such that all the following properties hold:

- (Contains $\longrightarrow$) For all $\langle c, \sigma\rangle$ and $\langle c', \sigma'\rangle$, if $\langle c, \sigma\rangle \longrightarrow \langle c', \sigma'\rangle$ then $\langle c, \sigma\rangle \longleftrightarrow^* \langle c', \sigma'\rangle$.
- (Reflexive) For all $\langle c, \sigma\rangle$, we have $\langle c, \sigma\rangle \longleftrightarrow^* \langle c, \sigma\rangle$.
- (Symmetric) For all $\langle c, \sigma\rangle$ and $\langle c', \sigma'\rangle$, if $\langle c, \sigma\rangle \longleftrightarrow^* \langle c', \sigma'\rangle$ then $\langle c', \sigma'\rangle \longleftrightarrow^* \langle c, \sigma\rangle$.
- (Transitive) For all $\langle c, \sigma\rangle$, $\langle c', \sigma'\rangle$, and $\langle c'', \sigma''\rangle$, if $\langle c, \sigma\rangle \longleftrightarrow^* \langle c', \sigma'\rangle$ and $\langle c', \sigma'\rangle \longleftrightarrow^* \langle c'', \sigma''\rangle$ then $\langle c, \sigma\rangle \longleftrightarrow^* \langle c'', \sigma''\rangle$.

Let σ_0 be the store that maps all program variables to zero. (See Appendix B for the small-step operational semantics of IMP.)

Which *one* of the following relationships does **NOT** hold?

a) $\langle \mathsf{foo} := 42, \sigma_0\rangle \longleftrightarrow^* \langle \mathsf{foo} := 35 + 7 + 0, \sigma_0\rangle$

b) $\langle \textbf{if } 0 < \mathsf{foo} \textbf{ then } \mathsf{bar} := 42 \textbf{ else skip}, \sigma_0\rangle \longleftrightarrow^* \langle \textbf{skip}, \sigma_0\rangle$

c) $\langle \mathsf{bar} := 20 + 7, \sigma_0\rangle \longleftrightarrow^* \langle \mathsf{bar} := 5 + 22, \sigma_0\rangle$

d) $\langle \textbf{if } 0 < \mathsf{foo} \textbf{ then } \mathsf{bar} := 42 \textbf{ else } \mathsf{foo} := 42, \sigma_0\rangle \longleftrightarrow^* \langle \textbf{if } 0 < \mathsf{foo} \textbf{ then } \mathsf{foo} := 42 \textbf{ else } \mathsf{bar} := 42, \sigma_0\rangle$

4. Consider the large-step evaluation of the IMP command **while** foo $<$ bar **do** foo $:=$ foo $+ 1$. Which *one* of the following statements is true? (See Appendix B for the large-step operational semantics of IMP.)

a) There do not exist stores σ and σ' such that $\langle \textbf{while } \mathsf{foo} < \mathsf{bar} \textbf{ do } \mathsf{foo} := \mathsf{foo} + 1, \sigma\rangle \Downarrow \sigma'$.

b) There exists a store σ such that for all stores σ' it is not the case that $\langle \textbf{while } \mathsf{foo} < \mathsf{bar} \textbf{ do } \mathsf{foo} := \mathsf{foo} + 1, \sigma\rangle \Downarrow \sigma'$.

c) For all stores σ and σ' there is a derivation that $\langle \textbf{while } \mathsf{foo} < \mathsf{bar} \textbf{ do } \mathsf{foo} := \mathsf{foo} + 1, \sigma\rangle \Downarrow \sigma'$.

d) For all stores σ there exists a store σ' such that $\langle \textbf{while } \mathsf{foo} < \mathsf{bar} \textbf{ do } \mathsf{foo} := \mathsf{foo} + 1, \sigma\rangle \Downarrow \sigma'$.

5. Consider two languages: (1) the language of arithmetic expressions (see Appendix A) and (2) the IMP language (see Appendix B). Circle all the ways to prove the determinism of their large-step evaluation.

Determinism for (1): For all expressions $e \in$ **Exp**, stores $\sigma, \sigma_1, \sigma_2 \in$ **Store** and numbers $n_1, n_2 \in$ **Int**, if $\langle e, \sigma\rangle \Downarrow \langle n_1, \sigma_1\rangle$ and $\langle e, \sigma\rangle \Downarrow \langle n_2, \sigma_2\rangle$ then $n_1 = n_2$ and $\sigma_1 = \sigma_2$.

Determinism for (2): For all commands $c \in$ **Com** and stores $\sigma, \sigma_1, \sigma_2 \in$ **Store**, if $\langle c, \sigma\rangle \Downarrow \sigma_1$ and $\langle c, \sigma\rangle \Downarrow \sigma_2$ then $\sigma_1 = \sigma_2$.

a) By structural induction over the expression for (1).

b) By structural induction over the command for (2).

c) By induction over one large-step derivation for (1).

d) by induction over one large-step derivation for (2).

6. Consider the IMP configuration $\langle$**while true do** foo := foo + 1, $\sigma_0\rangle$ where store σ_0 maps all program variables to zero.
True or false: There exists a command c' and store σ such that

$$\langle \textbf{while true do } \mathsf{foo} := \mathsf{foo} + 1, \sigma_0\rangle \longrightarrow \langle c', \sigma\rangle.$$

a) True

b) False

7. Suppose we extended the IMP language with a **while** b **do** c_1 **else** c_2 construct. Intuitively, it behaves similarly to the loop **while** b **do** c_1, except that if the loop executes zero times, then command c_2 is executed. Here are the large-step inference rules for the new construct.

$$\frac{\langle b, \sigma\rangle \Downarrow \textbf{false} \quad \langle c_2, \sigma\rangle \Downarrow \sigma'}{\langle \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2, \sigma\rangle \Downarrow \sigma'} \qquad \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \quad \langle \textbf{while } b \textbf{ do } c_1, \sigma\rangle \Downarrow \sigma'}{\langle \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2, \sigma\rangle \Downarrow \sigma'}$$

Which *one* of the following small step inference rules will extend the small-step semantics to be equivalent to the large-step semantics?

a) $$\frac{}{\langle \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle \textbf{if } b \textbf{ then } (\textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2) \textbf{ else } c_2, \sigma\rangle}$$

b) $$\frac{}{\langle \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle \textbf{if } b \textbf{ then } (c_2; \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2) \textbf{ else } c_2, \sigma\rangle}$$

c) $$\frac{}{\langle \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle \textbf{while } b \textbf{ do } (\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2), \sigma\rangle}$$

d) $$\frac{}{\langle \textbf{while } b \textbf{ do } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle \textbf{if } b \textbf{ then } (\textbf{while } b \textbf{ do } c_1) \textbf{ else } c_2, \sigma\rangle}$$

8. Which *one* expression is alpha-equivalent to $(\lambda x.\, x\ y)\ (\lambda z.\, z)$?

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

b) $(\lambda x.\, x\ w)\ (\lambda z.\, z)$

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

d) $(\lambda z.\, z)\ y$

9. Consider the small-step operational semantics of the language of arithmetic expressions (Appendix A). Assuming an initial store that maps all variables to zero, how many steps does the following program take to reach a final configuration?

$$x := (\mathsf{bar} \times (\mathsf{foo} + 3)); y + 2 \times x$$

a) 3

b) 9

c) 6

d) 10

10. Let $e_1 = \lambda y.\ f\ 37$, and let $e_2 = \lambda y.\ y + 5$. What is the result of $e_1\{e_2/f\}$?

a) $(\lambda y.\ y + 5)\ ((\lambda y.\ ((\lambda y.\ y + 5)\ 35))\ 2)$

b) $\lambda y.\ (\lambda y.\ y + 5)\ 37$

c) $(\lambda y.\ y + 5)\ 37$

d) $\lambda y.\ 37 + 5$

11. Suppose we extend the arithmetic language with division operation. That is, extend the grammar shown in Appendix A with division:

$$e ::= \ldots | e_1 / e_2$$

The large-step rule for division is: $\boxed{\textsc{Div}\ \dfrac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2}{e_1/e_2 \Downarrow n}\ \text{where } n = n_1/n_2 \text{ and } n_2 \neq 0}$

Which **one** of the following is true?

a) $0/0$ is syntactically not correct.

b) $0/0$ is syntactically correct but the evaluation gets stuck for all stores.

c) $0/0$ is syntactically correct and yields an answer when evaluated under some stores.

d) $0/0$ is syntactically correct and yields NaN (not a number) when evaluated under any store.

12. Consider the following rules for the inductive set, **MySet**

$$\textsc{Rule1}\ \frac{}{3 \in \textbf{MySet}} \qquad \textsc{Rule2}\ \frac{}{7 \in \textbf{MySet}} \qquad \textsc{Rule3}\ \frac{a \in \textbf{MySet} \quad b \in \textbf{MySet}}{c \in \textbf{MySet}}\, c = a + b + 5$$

Student-A wants to prove that every number in **MySet** is an odd number. Formally, the property Student-A is proving for **MySet** is:

$$P(a) \triangleq \exists i \in \mathbb{Z}. a = 2i + 1$$

Student-A decides to use induction to prove this property and has written the following proof of the property:

- Case RULE1: This is an axiom. We know that $a \equiv 3$. Let $i = 1$. As $2 \cdot 1 + 1 = 3$, we have shown the existence of an i and shown the property holds.
- Case RULE2: This is an axiom. We know that $a \equiv 7$. Let $i = 3$. As $2 \cdot 3 + 1 = 7$, we have shown the existence of an i and shown the property holds.
- Case RULE3: This is an inductive case. The inductive hypothesis is that $P(a)$ and $P(b)$. We want to show $P(c)$ holds. By $P(a)$ we know that $a = 2i + 1$ for some i. By $P(b)$ we know that $b = 3$. We must show that there is a k such that $c = 2k + 1$. We also know by inversion of RULE3 that $c = a + b + 5$. Substituting in values of a and b gives us $c = 2i + 1 + 3 + 5$. Reducing yields $c = 2i + 9$. Rewriting gives us $c = 2(i + 4) + 1$. Thus, by the existence of $i + 4$, $P(c)$ holds which is what we aimed to show.

Student-A's TA points out a mistake in Student-A's proof. Which of the following is an error in the proof?

a) Case RULE2 is not an axiom

b) We cannot use inversion on RULE3

c) The inductive hypothesis is applied incorrectly to a in Case RULE3

d) The inductive hypothesis is applied incorrectly to b in Case RULE3.

13. Consider the following OCaml code that defines an arithemtic expression and evaluates it.

```
type exp =
  | EVar of string
  | EInt of int
  | EAdd of exp * exp
  | EMul of exp * exp
;;

  let rec eval e env =
  match e with
  | EVar x -> env x
  | EInt n -> n
  | EAdd (e1,e2) -> (eval e1 env) + (eval e2 env)
  | EMul (e1,e2) -> (eval e1 env) * (eval e2 env)
;;
```

If you enter the above program in **utop**, you will notice that the type of `eval` is `exp` → `(string` → `int)` → `int`. That is,

```
utop# eval;;
- :  exp → (string → int) → int = ⟨fun⟩
```

Now let us try to run the following comamnd:

```
utop# eval (EVar "x") (fun x → None);;
```

Which of the following is true about the last line?

a) The last statement issues a type checking error as it cannot infer the type of `env` used in function `eval`.

b) The last statement issues a type checking error due to the following:

   i. The type of `(fun x → None)` is `string → 'a option`.

   ii. The type of `env` in the function `eval` is `string → int`.

   iii. The types of caller and callee arguments mismatch.

c) The last statement issues a type checking error as it cannot find the key `x`.

d) None of the above.

Descriptive Questions

1 Small-step operational semantics

Consider the small-step operational semantics for the language of arithmetic expressions (Appendix A). Let σ_0 be a store that maps all program variables to zero.

(a) Show a derivation that $\langle 3 + (5 \times \mathsf{bar}), \sigma_0 \rangle \longrightarrow \langle 3 + (5 \times 0), \sigma_0 \rangle$.

(b) What is the sequence of configurations that $\langle \mathsf{foo} := 5;\ (\mathsf{foo} + 2) \times 7, \sigma_0 \rangle$ steps to? (You don't need to show the derivations for each step, just show what configuration $\langle \mathsf{foo} := 5;\ (\mathsf{foo} + 2) \times 7, \sigma_0 \rangle$ steps to in one step, then two steps, then three steps, and so on, until you reach a final configuration.)

(c) Find an integer n and store σ' such that $\langle ((6 + (\mathsf{foo} := (\mathsf{bar} := 3; 5); 1 + \mathsf{bar})) + \mathsf{bar}) \times \mathsf{foo}, \sigma_0 \rangle \longrightarrow^* \langle n, \sigma' \rangle$.

(d) Is the relation $\longrightarrow$ reflexive? Is it symmetric? Is it anti-symmetric? Is it transitive?

(For each of these questions, if the answer is "no", what is a suitable counterexample? If any of the answers are "yes", think about how you would prove it.)

2 Large-step operational semantics

Consider the large-step operational semantics for the language of arithmetic expressions (Appendix A). Let σ_0 be a store that maps all program variables to zero.

(a) Show a derivation that $\langle 3 + (5 \times \mathsf{bar}), \sigma_0 \rangle \Downarrow \langle 3, \sigma_0 \rangle$.

(b) Find an integer n and store σ' such that $\langle \mathsf{foo} := 5;\ (\mathsf{foo} + 2) \times 7, \sigma_0 \rangle \Downarrow \langle n, \sigma' \rangle$.

If you have time and a big piece of paper, give the derivation of $\langle \mathsf{foo} := 5;\ (\mathsf{foo} + 2) \times 7, \sigma_0 \rangle \Downarrow \langle n, \sigma' \rangle$.

(c) Is the relation $\Downarrow$ reflexive? Is it symmetric? Is it anti-symmetric? Is it transitive?

(For each of these questions, if the answer is "no", what is a suitable counterexample? If any of the answers are "yes", think about how you would prove it.)

Appendix A Semantics for the language of arithmetic expressions

Small-step operational semantics

$$\textsc{Var}\ \frac{}{\langle x, \sigma\rangle \longrightarrow \langle n, \sigma\rangle}\ \text{where } n = \sigma(x)$$

$$\textsc{LAdd}\ \frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle e_1 + e_2, \sigma\rangle \longrightarrow \langle e_1' + e_2, \sigma'\rangle} \qquad \textsc{RAdd}\ \frac{\langle e_2, \sigma\rangle \longrightarrow \langle e_2', \sigma'\rangle}{\langle n + e_2, \sigma\rangle \longrightarrow \langle n + e_2', \sigma'\rangle}$$

$$\textsc{Add}\ \frac{}{\langle n + m, \sigma\rangle \longrightarrow \langle p, \sigma\rangle}\ \text{where } p \text{ is the sum of } n \text{ and } m$$

$$\textsc{LMul}\ \frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle e_1 \times e_2, \sigma\rangle \longrightarrow \langle e_1' \times e_2, \sigma'\rangle} \qquad \textsc{RMul}\ \frac{\langle e_2, \sigma\rangle \longrightarrow \langle e_2', \sigma'\rangle}{\langle n \times e_2, \sigma\rangle \longrightarrow \langle n \times e_2', \sigma'\rangle}$$

$$\textsc{Mul}\ \frac{}{\langle n \times m, \sigma\rangle \longrightarrow \langle p, \sigma\rangle}\ \text{where } p \text{ is the product of } n \text{ and } m$$

$$\textsc{Asg1}\ \frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle x := e_1; e_2, \sigma\rangle \longrightarrow \langle x := e_1'; e_2, \sigma'\rangle} \qquad \textsc{Asg}\ \frac{}{\langle x := n; e_2, \sigma\rangle \longrightarrow \langle e_2, \sigma[x \mapsto n]\rangle}$$

Large-step operational semantics

$$\textsc{Int}_{\textsc{Lrg}}\ \frac{}{\langle n, \sigma\rangle \Downarrow \langle n, \sigma\rangle} \qquad \textsc{Var}_{\textsc{Lrg}}\ \frac{}{\langle x, \sigma\rangle \Downarrow \langle n, \sigma\rangle}\ \text{where } \sigma(x) = n$$

$$\textsc{Add}_{\textsc{Lrg}}\ \frac{\langle e_1, \sigma\rangle \Downarrow \langle n_1, \sigma''\rangle \qquad \langle e_2, \sigma''\rangle \Downarrow \langle n_2, \sigma'\rangle}{\langle e_1 + e_2, \sigma\rangle \Downarrow \langle n, \sigma'\rangle}\ \text{where } n \text{ is the sum of } n_1 \text{ and } n_2$$

$$\textsc{Mul}_{\textsc{Lrg}}\ \frac{\langle e_1, \sigma\rangle \Downarrow \langle n_1, \sigma''\rangle \qquad \langle e_2, \sigma''\rangle \Downarrow \langle n_2, \sigma'\rangle}{\langle e_1 \times e_2, \sigma\rangle \Downarrow \langle n, \sigma'\rangle}\ \text{where } n \text{ is the product of } n_1 \text{ and } n_2$$

$$\textsc{Asg}_{\textsc{Lrg}}\ \frac{\langle e_1, \sigma\rangle \Downarrow \langle n_1, \sigma''\rangle \qquad \langle e_2, \sigma''[x \mapsto n_1]\rangle \Downarrow \langle n_2, \sigma'\rangle}{\langle x := e_1; e_2, \sigma\rangle \Downarrow \langle n_2, \sigma'\rangle}$$

Syntax

$$e ::= x \mid n \mid e_1 + e_2 \mid e_1 \times e_2 \mid x := e_1; e_2$$

Appendix B Operational Semantics for IMP

Syntax

arithmetic expressions	$a \in$ **Aexp**	$a ::= x \mid n \mid a_1 + a_2 \mid a_1 \times a_2$
boolean expressions	$b \in$ **Bexp**	$b ::=$ **true** $\mid$ **false** $\mid a_1 < a_2$
commands	$c \in$ **Com**	$c ::=$ **skip** $\mid x := a \mid c_1; c_2$
		$\mid$ **if** b **then** c_1 **else** c_2
		$\mid$ **while** b **do** c

B.1 Small-step Operational Semantics

Arithmetic expressions.

$$\frac{}{\langle x, \sigma\rangle \longrightarrow \langle n, \sigma\rangle} \text{ where } n = \sigma(x)$$

$$\frac{\langle a_1, \sigma\rangle \longrightarrow \langle a_1', \sigma\rangle}{\langle a_1 + a_2, \sigma\rangle \longrightarrow \langle a_1' + a_2, \sigma\rangle} \qquad \frac{\langle a_2, \sigma\rangle \longrightarrow \langle a_2', \sigma\rangle}{\langle n + a_2, \sigma\rangle \longrightarrow \langle n + a_2', \sigma\rangle} \qquad \frac{}{\langle n + m, \sigma\rangle \longrightarrow \langle p, \sigma\rangle} \text{ where } p = n + m$$

$$\frac{\langle a_1, \sigma\rangle \longrightarrow \langle a_1', \sigma\rangle}{\langle a_1 \times a_2, \sigma\rangle \longrightarrow \langle a_1' \times a_2, \sigma\rangle} \qquad \frac{\langle a_2, \sigma\rangle \longrightarrow \langle a_2', \sigma\rangle}{\langle n \times a_2, \sigma\rangle \longrightarrow \langle n \times a_2', \sigma\rangle} \qquad \frac{}{\langle n \times m, \sigma\rangle \longrightarrow \langle p, \sigma\rangle} \text{ where } p = n \times m$$

Boolean expressions.

$$\frac{\langle a_1, \sigma\rangle \longrightarrow \langle a_1', \sigma\rangle}{\langle a_1 < a_2, \sigma\rangle \longrightarrow \langle a_1' < a_2, \sigma\rangle} \qquad \frac{\langle a_2, \sigma\rangle \longrightarrow \langle a_2', \sigma\rangle}{\langle n < a_2, \sigma\rangle \longrightarrow \langle n < a_2', \sigma\rangle}$$

$$\frac{}{\langle n < m, \sigma\rangle \longrightarrow \langle \textbf{true}, \sigma\rangle} \text{ where } n < m \qquad \frac{}{\langle n < m, \sigma\rangle \longrightarrow \langle \textbf{false}, \sigma\rangle} \text{ where } n \geq m$$

Commands.

$$\frac{\langle a, \sigma\rangle \longrightarrow \langle a', \sigma\rangle}{\langle x := a, \sigma\rangle \longrightarrow \langle x := a', \sigma\rangle} \qquad \frac{}{\langle x := n, \sigma\rangle \longrightarrow \langle \textbf{skip}, \sigma[x \mapsto n]\rangle} \qquad \frac{\langle c_1, \sigma\rangle \longrightarrow \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \longrightarrow \langle c_1'; c_2, \sigma'\rangle}$$

$$\frac{}{\langle \textbf{skip}; c_2, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle} \qquad \frac{\langle b, \sigma\rangle \longrightarrow \langle b', \sigma\rangle}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle \textbf{if } b' \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle}$$

$$\frac{}{\langle \textbf{if true then } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle} \qquad \frac{}{\langle \textbf{if false then } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle}$$

$$\frac{}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \longrightarrow \langle \textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else skip}, \sigma\rangle}$$

B.2 Large-step Operational Semantics

Arithmetic expressions.

$$\frac{}{\langle n, \sigma\rangle \Downarrow n} \qquad \frac{}{\langle x, \sigma\rangle \Downarrow n} \text{ where } \sigma(x) = n$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 + e_2, \sigma\rangle \Downarrow n} \text{ where } n = n_1 + n_2 \qquad \frac{\langle e_1, \sigma\rangle \Downarrow n_1 \qquad \langle e_2, \sigma\rangle \Downarrow n_2}{\langle e_1 \times e_2, \sigma\rangle \Downarrow n} \text{ where } n = n_1 \times n_2$$

Boolean expressions.

$$\frac{}{\langle \textbf{true}, \sigma\rangle \Downarrow \textbf{true}} \qquad \frac{}{\langle \textbf{false}, \sigma\rangle \Downarrow \textbf{false}}$$

$$\frac{\langle a_1, \sigma\rangle \Downarrow n_1 \qquad \langle a_2, \sigma\rangle \Downarrow n_2}{\langle a_1 < a_2, \sigma\rangle \Downarrow \textbf{true}} \text{ where } n_1 < n_2 \qquad \frac{\langle a_1, \sigma\rangle \Downarrow n_1 \qquad \langle a_2, \sigma\rangle \Downarrow n_2}{\langle a_1 < a_2, \sigma\rangle \Downarrow \textbf{false}} \text{ where } n_1 \geq n_2$$

Commands.

$$\textsc{Skip}\ \frac{}{\langle \textbf{skip}, \sigma\rangle \Downarrow \sigma} \qquad \textsc{Asg}\ \frac{\langle e, \sigma\rangle \Downarrow n}{\langle x := e, \sigma\rangle \Downarrow \sigma[x \mapsto n]} \qquad \textsc{Seq}\ \frac{\langle c_1, \sigma\rangle \Downarrow \sigma' \qquad \langle c_2, \sigma'\rangle \Downarrow \sigma''}{\langle c_1; c_2, \sigma\rangle \Downarrow \sigma''}$$

$$\textsc{If-T}\ \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \qquad \langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \Downarrow \sigma'} \qquad \textsc{If-F}\ \frac{\langle b, \sigma\rangle \Downarrow \textbf{false} \qquad \langle c_2, \sigma\rangle \Downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \Downarrow \sigma'}$$

$$\textsc{While-F}\ \frac{\langle b, \sigma\rangle \Downarrow \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \Downarrow \sigma} \qquad \textsc{While-T}\ \frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \qquad \langle c, \sigma\rangle \Downarrow \sigma' \qquad \langle \textbf{while } b \textbf{ do } c, \sigma'\rangle \Downarrow \sigma''}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \Downarrow \sigma''}$$

Appendix C Pure lambda calculus semantics

Syntax:

$$e ::= x \mid \lambda x.\, e \mid e_1\ e_2$$
$$v ::= \lambda x.\, e$$

Call-by-value Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{e_2 \longrightarrow e_2'}{v\ e_2 \longrightarrow v\ e_2'} \qquad \frac{}{(\lambda x.\, e)\ v \longrightarrow e\{v/x\}}$$

Call-by-name Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{}{(\lambda x.\, e)\ e_2 \longrightarrow e\{e_2/x\}}$$

Full beta-reduction Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{e_2 \longrightarrow e_2'}{e_1\ e_2 \longrightarrow e_1\ e_2'} \qquad \frac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'} \qquad \frac{}{(\lambda x.\, e)\ e_2 \longrightarrow e\{e_2/x\}}$$

Normal-order Evaluation Semantics:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \text{ where } e_1 \text{ is not of the form } \lambda x.\, e \qquad \frac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'} \qquad \frac{}{(\lambda x.\, e)\ e_2 \longrightarrow e\{e_2/x\}}$$