

# Elements of Programming Languages

## Tutorial 7: Small-step semantics

### Solution notes

#### 1. Imperative programming

(a)  $y := x + x$

$$\frac{\overline{3 + 3 \Downarrow 6}}{\sigma, y := x + x \Downarrow \sigma[y := 6]}$$

(b)  $\text{if } x == y \text{ then } x := x + 1 \text{ else } y := y + 2$

$$\frac{\frac{3 \Downarrow 3 \quad 4 \Downarrow 4}{3 == 4 \Downarrow \text{true}} \quad \frac{4 + 2 \Downarrow 6}{\sigma, y := y + 2 \Downarrow \sigma[y := 6]}}{\sigma, \text{if } x == y \text{ then } x := x + 1 \text{ else } y := y + 2 \Downarrow \sigma[y := 6]}$$

(c)  $(\star) \text{ while } x < y \text{ do } x := x + 1$

$$\frac{3 < 4 \Downarrow \text{true} \quad \frac{3 + 1 \Downarrow 4}{\sigma, x := x + 1 \Downarrow \sigma[x := 4]} \quad \frac{4 < 4 \Downarrow \text{false}}{\sigma[x := 4], \text{while } x < y \text{ do } x := x + 1 \Downarrow \sigma[x := 4]}}{\sigma, \text{while } x < y \text{ do } x := x + 1 \Downarrow \sigma[x := 4]}$$

#### 2. Comparing large-step and small-step derivations

(a) For large-step:

$$\frac{\lambda x. x + 1 \Downarrow \lambda x. x + 1 \quad 42 \Downarrow 42 \quad \frac{42 \Downarrow 42 \quad 1 \Downarrow 1}{42 + 1 \Downarrow 43}}{(\lambda x. x + 1) 42 \Downarrow 43}$$

For small-step, the step derivations are:

$$\frac{}{(\lambda x. x + 1) 42 \mapsto 42 + 1}$$

$$\frac{}{42 + 1 \mapsto 43}$$

(b) For large-step:

$$\frac{f \Downarrow f \quad 42 \Downarrow 42 \quad \frac{42 \Downarrow 42 \quad 1 \Downarrow 1}{42 == 1 \Downarrow \text{false}} \quad \frac{42 \Downarrow 42 \quad 1 \Downarrow 1}{42 + 1 \Downarrow 43}}{(\lambda x. \text{if } x == 1 \text{ then } 2 \text{ else } x + 1) 42 \Downarrow 43}$$

For small-step, the step derivations are:

$$\frac{}{(\lambda x. \text{if } x == 1 \text{ then } 2 \text{ else } x + 1) 42 \mapsto \text{if } 42 == 1 \text{ then } 2 \text{ else } 42 + 1}$$

$$\frac{\overline{42 == 1 \mapsto \text{false}}}{\text{if } 42 == 1 \text{ then } 2 \text{ else } 42 + 1 \mapsto \text{if false then } 2 \text{ else } 42 + 1}$$

$$\frac{}{\text{if false then } 2 \text{ else } 42 + 1 \mapsto 42 + 1}$$

$$\frac{}{42 + 1 \mapsto 43}$$

### 3. Small-step derivations that go wrong

(a) The small-step derivation is:

$$\begin{aligned} ((\lambda x. \lambda y. \text{let } z = x + y \text{ in } z + 1) \ 42) \ \text{true} &\mapsto (\lambda y. \text{let } z = 42 + y \text{ in } z + 1) \ \text{true} \\ &\mapsto \text{let } z = 42 + \text{true} \text{ in } z + 1 \end{aligned}$$

which is stuck because the next subexpression to evaluate,  $42 + \text{true}$ , cannot take a step.

(b) The small-step derivation is:

$$\begin{aligned} (\lambda x. \text{if } x \text{ then } x + 1 \text{ else } x + 2) \ \text{true} &\mapsto \text{if } \text{true} \text{ then } \text{true} + 1 \text{ else } \text{true} + 2 \\ &\mapsto \text{true} + 1 \end{aligned}$$

### 4. Small-step rules for $L_{\text{Data}}$

(a) Here are some possible rules:

$$\boxed{e \mapsto e'}$$

$$\begin{array}{c} \frac{e_1 \mapsto e'_1}{(e_1, e_2) \mapsto (e'_1, e_2)} \quad \frac{e_2 \mapsto e'_2}{(v_1, e_2) \mapsto (v_1, e'_2)} \\ \frac{}{\text{fst } (v_1, v_2) \mapsto v_1} \quad \frac{}{\text{snd } (v_1, v_2) \mapsto v_2} \\ \frac{e \mapsto e'}{\text{fst } e \mapsto \text{fst } e'} \quad \frac{e \mapsto e'}{\text{snd } e \mapsto \text{snd } e'} \\ \frac{e \mapsto e'}{\text{left}(e) \mapsto \text{left}(e')} \quad \frac{e \mapsto e'}{\text{right}(e) \mapsto \text{right}(e')} \\ \frac{e \mapsto e'}{\text{case } e \text{ of } \{\text{left}(x) \Rightarrow e_1 ; \text{right}(y) \Rightarrow e_2\} \mapsto \text{case } e' \text{ of } \{\text{left}(x) \Rightarrow e_1 ; \text{right}(y) \Rightarrow e_2\}} \\ \frac{}{\text{case left}(v) \text{ of } \{\text{left}(x) \Rightarrow e_1 ; \text{right}(y) \Rightarrow e_2\} \mapsto e_1[v/x]} \\ \frac{}{\text{case right}(v) \text{ of } \{\text{left}(x) \Rightarrow e_1 ; \text{right}(y) \Rightarrow e_2\} \mapsto e_2[v/y]} \end{array}$$

One example of a design choice is whether to evaluate the subexpressions of a pair left-to-right, or right-to-left (or not specify the order, or not evaluate them eagerly at all...)

(b) i. The step derivations are:

$$\begin{aligned} &\overline{(\lambda p. (\text{snd } p, \text{fst } p + 2)) \ (17, 42) \mapsto (\text{snd } (17, 42), \text{fst } (17, 42) + 2)} \\ &\frac{}{\text{snd } (17, 42) \mapsto 42} \\ &\frac{}{(\text{snd } (17, 42), \text{fst } (17, 42) + 2) \mapsto (42, \text{fst } (17, 42) + 2)} \\ &\frac{}{\text{fst } (17, 42) \mapsto 17} \\ &\frac{}{\text{fst } (17, 42) + 2 \mapsto 17 + 2} \\ &\frac{}{(42, \text{fst } (17, 42) + 2) \mapsto (42, 17 + 2)} \\ &\frac{}{(42, 17 + 2) \mapsto (42, 19)} \end{aligned}$$

ii. The step derivations are:

$$\begin{aligned} &\overline{(\lambda x. \text{case } x \text{ of } \{\text{left}(y).y + 1 ; \text{right}(z).z\}) \ (\text{left}(42)) \mapsto \text{case left}(42) \text{ of } \{\text{left}(y).y + 1 ; \text{right}(z).z\}} \\ &\frac{}{\text{case left}(42) \text{ of } \{\text{left}(y).y + 1 ; \text{right}(z).z\} \mapsto 42 + 1} \\ &\frac{}{42 + 1 \mapsto 43} \end{aligned}$$

5. (★) **Type soundness for nondeterminism**

- (a) Preservation means that if  $\vdash e : \tau$  holds and  $e \mapsto e'$  then  $\vdash e' : \tau$  holds.

To prove preservation, suppose we have a well-formed nondeterministic choice expression:

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \square e_2 : \tau}$$

such that  $e_1 \square e_2 \mapsto e'$ .

There are two cases. If  $e_1 \square e_2 \mapsto e_1$  then we know  $\vdash e_1 : \tau$  by assumption. Similarly, if  $e_1 \square e_2 \mapsto e_2$  then we know  $\vdash e_2 : \tau$ .

- (b) Progress means that if  $\vdash e : \tau$  holds then either  $e$  is a value, or  $e$  can take a step ( $e \mapsto e'$ ).

To prove progress for a nondeterministic expression  $\vdash e_1 \square e_2 : \tau$ , this is immediate since such an expression can definitely step to  $e_1$  (or  $e_2$ ). Note that progress does not require that the step taken is unique; indeed, type soundness can hold for a nondeterministic language as long as both alternatives in a choice expression have the same type.