Sample Solution for Homework 5

Problem 1 Operational Semantics (26 Points)

In this problem, we will practice how to use the inference rules of the operational semantics to evaluate expressions. Moreover, we will practice how to define inference rules for new language constructs.

(a) Show the derivation tree for the evaluation of the following expressions according to the big-step operational semantics given in Figure 4.1 of the course notes. Start your evaluation with the environment $env = \{x \mapsto 3, y \mapsto -2\}$. **(5 Points)**

(i) $3 * (x + 2)$

$$\dfrac{\dfrac{env(x) = 3}{env \vdash x \Downarrow 3}\ \textsc{EvalVar} \quad \dfrac{}{env \vdash 2 \Downarrow 2}\ \textsc{EvalVal} \quad 5 = toNum(3) + toNum(2)}{env \vdash x + 2 \Downarrow 5}\ \textsc{EvalPlus}$$

$$\dfrac{\dfrac{}{env \vdash 2 \Downarrow 2}\ \textsc{EvalVal} \quad 15 = toNum(3) \cdot toNum(5)}{env \vdash 3 * (x + 2) \Downarrow 15}\ \textsc{EvalTimes}$$

(ii) **const** $b = 2 + y; b\ ?\ x : y$

$$\dfrac{\dfrac{}{env \vdash 2 \Downarrow 2}\ \textsc{EvalVal} \quad \dfrac{env(y) = -2}{env \vdash y \Downarrow -2}\ \textsc{EvalVar} \quad 0 = toNum(2) + toNum(-2)}{env \vdash 2 + y \Downarrow 0}\ \textsc{EvalPlus}$$

$$env' = env[b \mapsto 0]$$

$$\dfrac{\dfrac{\dfrac{env'(b) = 0}{env' \vdash b \Downarrow 0}\ \textsc{EvalVar} \quad \dfrac{env'(y) = -2}{env' \vdash y \Downarrow -2}\ \textsc{EvalVar} \quad \texttt{false} = toBool(0)}{env' \vdash b\ ?\ x : y \Downarrow -2}\ \textsc{EvalIfElse}}{env \vdash \textbf{const}\ b = 2 + y; b\ ?\ x : y \Downarrow -2}\ \textsc{EvalConst}$$

(b) Show the evaluation steps of the following expressions according to the small-step operational semantics given in figures 4.2 and 4.3 of the course notes. Start your evaluation with the empty environment $env = \emptyset$ (i.e., the environment that does not bind any variables).

(i) $3 + (1 \;\&\&\; 5)$

The evaluation steps for this expression are as follows:

$$\begin{array}{lll} & 3 + (\underline{1 \;\&\&\; 5}) & \textsc{SearchBop}_2, \textsc{DoAndTrue} \\ \rightarrow & 3 + 5 & \textsc{DoPlus} \\ \rightarrow & 8 & \end{array}$$

(ii) **const** $x = 2 + 1; x * 0 \;?\; x : x + x$

The evaluation steps for this expression are as follows:

$$\begin{array}{lll} & \textbf{const}\ x = \underline{2 + 1}; x * 0 \;?\; x : x + x & \textsc{SearchConstDecl}_1, \textsc{DoPlus} \\ \rightarrow & \textbf{const}\ x = 3; \underline{x * 0} \;?\; x : x + x & \textsc{SearchConstDecl}_2, \textsc{DoTimes} \\ \rightarrow & \textbf{const}\ x = 3; \underline{0 \;?\; x : x + x} & \textsc{SearchConstDecl}_2, \textsc{DoIfElse} \\ \rightarrow & \textbf{const}\ x = 3; \underline{x} + x & \textsc{SearchConstDecl}_2, \textsc{DoVar} \\ \rightarrow & \textbf{const}\ x = 3; 3 + \underline{x} & \textsc{SearchConstDecl}_2, \textsc{DoVar} \\ \rightarrow & \textbf{const}\ x = 3; \underline{3 + 3} & \textsc{SearchConstDecl}_2, \textsc{DoPlus} \\ \rightarrow & \textbf{const}\ x = 3; 6 & \textsc{DoConstDecl} \\ \rightarrow & 6 & \end{array}$$

(c) Consider the small-step semantics of $e_1 + e_2$ given in figures 4.2 and 4.3 of the course notes (i.e., the rules DoPlus, SearchBop$_1$ and SearchBop$_2$). These rules realize a left-to-right evaluation order of such expressions (i.e., first e_1 is evaluated and then e_2). Provide alternative rules for the semantics of $e_1 + e_2$ that realize a right-to-left evaluation order of such expressions (i.e., first e_2 is evaluated and then e_1). **Hint:** You only need to change the search rules. **(5 Points)**

$$\frac{env \vdash e_1 \rightarrow e_1'}{env \vdash e_1 + v_2 \rightarrow e_1' + v_2}\ \textsc{SearchPlus1} \qquad \frac{env \vdash e_2 \rightarrow e_2'}{env \vdash e_1 + e_2 \rightarrow e_1 + e_2'}\ \textsc{SearchPlus2}$$

(d) The JavaScript sequencing operator , allows us to compose expressions sequentially. The intended semantics of an expression e_1 , e_2 is that we first evaluate e_1 and then we evaluate e_2. The entire expression then evaluates to the result of e_2. That is, the result of e_1 is discarded. We only evaluate e_1 to observe the side effects of its evaluation (e.g., printing). For example, the expression $(2 + 3)\ , (3 + 4)$, should evaluate to 7, which is the result of the subexpression after ,. The first subexpression $(2 + 3)$ should be evaluated, but its result is discarded.

Provide inference rules for both the big-step and small-step SOS of e_1 , e_2 that formalize the semantics of described above. Provide both the do and search rules for the small-step SOS of the new operator. Your search rules should enforce the correct evaluation order for the subexpressions e_1 and e_2. **(5 Points)**

Big-step rule:

$$\frac{env \vdash e_1 \Downarrow v_1 \qquad env \vdash e_2 \Downarrow v_2}{env \vdash e_1\ , e_2 \Downarrow v_2}\ \textsc{EvalSeq}$$

Small-step rules:

$$\frac{env \vdash e_1 \to e_1'}{env \vdash e_1 \text{ , } e_2 \to e_1' \text{ , } e_2}\ \textsc{SearchSeq} \qquad \frac{}{env \vdash v_1 \text{ , } e_2 \to e_2}\ \textsc{DoSeq}$$

Alternative small-step rules:

$$\frac{env \vdash e_1 \to e_1'}{env \vdash e_1 \text{ , } e_2 \to e_1' \text{ , } e_2}\ \textsc{SearchSeq}_1 \qquad \frac{env \vdash e_2 \to e_2'}{env \vdash v_1 \text{ , } e_2 \to v_1 \text{ , } e_2'}\ \textsc{SearchSeq}_2$$

$$\frac{}{env \vdash v_1 \text{ , } v_2 \to v_2}\ \textsc{DoSeq}$$

(e) For each of the following JavaScript programs, provide the result of evaluating the program according to the dynamic binding semantics given in figures 5.1 and 5.2 of the course notes. You do not need to show the intermediate results of the evaluation. However, for each application of the EvalVar rule during evaluation, describe (1) the using occurrence of the variable to which the rule is applied to, (2) the value that is retrieved for that variable from the current environment in the rule application, and (3) where in the program the respective variable was bound to this value during evaluation. **(6 Points)**

(i) Program:

```
1 const x = 2;
2 const g = y => x + y;
3 const f = y => g(y);
4 f(3)
```

This program evaluates to the value `5`. During evaluation, the EvalVar rule is applied five times as follows:

- the first application is for the using occurrence of `f` on line 4. In this case, `f` is bound to the value `y => g(y)` in the **`const`** declaration on line 3.
- the second application is for the using occurrence of `g` on line 3. In this case, `g` was bound to the value `y => x + y` in the **`const`** declaration on line 2.
- the third application is for the using occurrence of `y` in the definition of function `f` on line 3. In this case, `y` was bound to the value `3` in the call to `f` on line 4.
- the fourth application is for the using occurrence of `x` in the definition of function `g` on line 2. In this case, `x` was bound to the value `2` in the **`const`** declaration on line 1.
- the fifth application is for the using occurrence of `y` in `g` on line 2. This occurrence of `y` was bound to the value `3` in the call to `g` on line 3.

(ii) Program:

```
1 const x = 2;
2 const g = x => y => x + y;
3 const f = y => g(y)(y);
4 f(3)
```

This program evaluates to the value `5`. During evaluation, the EvalVar rule is applied six times as follows:

- the first application is for the using occurrence of `f` on line 4. In this case, `f` is bound to the value `y => g(y)(y)` in the **`const`** declaration on line 3.
- the second application is for the using occurrence of `g` on line 3. In this case, `g` was bound to the value `x => y => x + y` in the **`const`** declaration on line 2.
- the third application is for the first using occurrence of `y` in the definition of function `f` on line 3. In this case, `y` was bound to the value `3` in the call to `f` on line 4.
- the fourth application is for the second using occurrence of `y` in the definition of function `f` on line 3. In this case, `y` was bound to the value `3` in the call to `f` on line 4.
- the fifth application is for the using occurrence of `x` in the definition of function `g` on line 2. In this case, `x` was bound to the value `2` in the **`const`** declaration on line 1.
- the sixth application is for the using occurrence of `y` in the definition of function `g` on line 2. In this case, `y` was bound to the value `3` in the nested call to `g` on line 3.

Note that under static binding semantics the above program evaluates to `6`.

Problem 2 Substitutions (14 Points)

In the following, let $x, y, z \in \mathit{Var}$ be distinct variables. Given the expressions

$$
\begin{aligned}
e_1 &= (x * y) + 4 \\
e_2 &= \textbf{const}\ y = y; x + y \\
e_3 &= \textbf{const}\ x = y \Rightarrow x(y); x(y)
\end{aligned}
$$

compute the following independent substitutions:

(a) $e_1[3/x] = (3 * y) + 4$

(b) $e_1[3/z] = (x * y) + 4 = e_1$

(c) $e_2[3/x] = \textbf{const}\ y = y; 3 + y$

(d) $e_2[3/y] = \textbf{const}\ y = 3; x + y$

(e) $e_3[(y(2))/y] = \textbf{const}\ x = y \Rightarrow x(y); x(y(2))$

(f) $e_3[(y(x))/x] = \textbf{const}\ x = z \Rightarrow y(x)(z); x(y)$