CS164 Programming Languages and Compilers **Spring 2024**

Discussion Worksheet 8

1 Operational Semantics

Recall that judgements for Type Checking are of the form:

$$O, M, C \vdash e : T$$

Judgements for Operational Semantics are (for now) of the form:

$$E, S \vdash e : v, S', R'$$

Exercise 1 Let's begin by going over the basics of what operational semantics mean.

1.1 What are the values v?

1.2 How are these values related to types?

1.3 What is the connection between the typing judgment and the operational semantics judgement? I.e., what does it mean for a program to be well-typed, and how does this relate to a program evaluating under operational semantics?

1.4 What is E?

1.5 What is S?

Exercise 2 Let's consider the integer addition expression $e_1 + e_2$. This is the operational rule, along with the rule for assignment statements:

$$\text{ADD}\ \frac{\begin{array}{c} E, S_1 \vdash e_1 : int(i_1), S_2, \_ \\ E, S_2 \vdash e_2 : int(i_2), S_3, \_ \end{array}}{E, S_1 \vdash e_1 + e_2 : int(v), S_3, \_}$$

$$\text{VAR-ASSIGNMENT-STMT}\ \frac{\begin{array}{c} E, S \vdash e : v, S_1, \_ \\ E(id) = l_{id} \\ S_2 = S_1[v/l_{id}]\_ \end{array}}{E, S \vdash id = e : \_, S_2, \_}$$

2.1 Why do we separate the store and the environment?

2.2 Why does the add judgement update the store but not the environment?

Exercise 3 Next, consider the if-else expression: b_1 `if` e `else` b_2.

3.1 Refresher: write the typing rule for the if-else expression.

$$\frac{}{O, M, C, R \vdash b_1\ \texttt{if}\ e\ \texttt{else}\ b_2 : T_1 \sqcup T_2}$$

3.2 Write the operational rules for the if-else expression.

$$\text{IF-ELSE-EXPR-TRUE}\ \frac{}{E, S \vdash b_1\ \texttt{if}\ e\ \texttt{else}\ b_2 : v, S_2, \_}$$

$$\text{IF-ELSE-EXPR-FALSE}\ \frac{}{E, S \vdash b_1\ \texttt{if}\ e\ \texttt{else}\ b_2 : v, S_2, \_}$$

3.3 Operational rules are more precise then typing rules. Why even bother with typing rules, then?

Exercise 4 Finally, we get to the purpose of the R in the conclusion of our operation semantics. The return statement has special, interruptive behavior, that is specially modeled in R .

Here are the operational rules for the `return` statement. Note how propagate the return value to R.

$$\frac{E, S \vdash e : v, S_1, \_}{E, S \vdash \texttt{return}\ e : \_, S_1, v}$$

$$\frac{}{E, S \vdash \texttt{return} : \_, S, \texttt{None}}$$

4.1 Recall the operational semantics of `while` shown in class, when the guard evaluates to `True`:

$$\text{WHILE-TRUE-LOOP}\ \frac{\begin{array}{c} E, S \vdash e_1 : bool(true), S_1, \_ \\ E, S_1 \vdash b_2 : \_, S_2, \_ \\ E, S_2 \vdash \texttt{while}\ e_1 : b_2 : \_, S_3, \_ \end{array}}{E, S \vdash \texttt{while}\ e_1 : b_2 : \_, S_3, \_}$$

Note that these do not propagate or consider return values. Complete the operational semantics for the execution `while` when the body of the loop returns (WHILE-TRUE-LOOP). Then, update WHILE-TRUE-LOOP to propagate return values properly. Hint: For WHILE-TRUE-LOOP, support the case where the current body does not return, but it may in a subsequent iteration.

$$\text{WHILE-TRUE-RETURN}\frac{}{E, S \vdash \texttt{while } e_1 : b_2 : \_, S_2, R}$$

$$\text{WHILE-TRUE-LOOP}\frac{}{E, S \vdash \texttt{while } e_1 : b_2 : \_, S_3, R}$$