Assignment 2: L2 Statics as Inference Rules

15-411/611: Course Staff

Due Wednesday, February 15, 2023 (11:59PM)

Reminder: Assignments are individual assignments, not done in pairs. The work must be all your own. Hand in your solutions on Gradescope. Please read the late policy for written assignments on the course web page.

1 Inference Rules

The static semantics of imperative languages can become quite complicated to describe precisely. For C0 (and L1-L5), the statics ensure three desirable properties of the language:

- *type safety*: expressions are well-typed, variables are assigned values of the correct type, . . .
- *variable initialization*: variables are always initialized before use.
- *proper returns*: every control flow path in a function ends in a return.

Specifying the static semantics inductively with inference rules is useful because the rules lead to a recursive algorithm for the semantic analysis.

We provide inference rules for type safety in the L2 handout. In this question, we consider inference rules for the other two properties, which will help you when implementing the semantic analysis for L2.

It might be helpful to review the lecture notes on inductive definitions from Lecture 1.

2 Variable Initialization

In lecture we saw a formalization of variable initialization semantics, using the judgment:

$$\Gamma;\ \Delta \vdash s \Rightarrow \Delta'$$

The intended meaning of the judgment is:

> If the variables in Γ are *declared* and the variables in Δ are *initialized*, then the statement s does not use uninitialized variables, and, after executing s, leaves Δ' variables initalized.

The context Γ here is the same as in the typechecking judgment, but in this question we omit the types for simplicity.

The complete definition of the judgment is in Figure 1. We write $\Delta \vdash e$ to mean, "e uses only variables in Δ". Note the definition maintains the invariant that $\Delta \subseteq \Delta' \subseteq \Gamma$ always.

$$\frac{\Gamma;\ \Delta \vdash s_1 \Rightarrow \Delta' \qquad \Gamma;\ \Delta' \vdash s_2 \Rightarrow \Delta''}{\Gamma;\ \Delta \vdash \mathsf{seq}(s_1, s_2) \Rightarrow \Delta''}\ \textsc{Seq} \qquad\qquad \frac{}{\Gamma;\ \Delta \vdash \mathsf{nop} \Rightarrow \Delta}\ \textsc{Nop}$$

$$\frac{\Gamma \cup \{x\};\ \Delta \vdash s \Rightarrow \Delta'}{\Gamma;\ \Delta \vdash \mathsf{declare}(x, \tau, s) \Rightarrow \Delta' \setminus \{x\}}\ \textsc{Declare} \qquad\qquad \frac{\Delta \vdash e}{\Gamma;\ \Delta \vdash \mathsf{assign}(x, e) \Rightarrow \Delta \cup \{x\}}\ \textsc{Assign}$$

$$\frac{\Delta \vdash e \qquad \Gamma;\ \Delta \vdash s_1 \Rightarrow \Delta' \qquad \Gamma;\ \Delta \vdash s_2 \Rightarrow \Delta''}{\Gamma;\ \Delta \vdash \mathsf{if}(e, s_1, s_2) \Rightarrow \Delta' \cap \Delta''}\ \textsc{If} \qquad\qquad \frac{\Delta \vdash e \qquad \Gamma;\ \Delta \vdash s \Rightarrow \Delta'}{\Gamma;\ \Delta \vdash \mathsf{while}(e, s) \Rightarrow \Delta}\ \textsc{While}$$

$$\frac{\Delta \vdash e}{\Gamma;\ \Delta \vdash \mathsf{return}(e) \Rightarrow \Gamma}\ \textsc{Return}$$

Figure 1: Definition of the $\Gamma;\ \Delta \vdash s \Rightarrow \Delta'$ judgment.

We can use these rules to build derivation trees, where we prove the premises of each rule by applying more rules. For example, we can prove the program in Figure 2 does not use uninitialized variables with the derivation tree in Figure 3.

```
int main() {
    int x;
    x = 1;
    return x;
}
```

$$\begin{array}{l} \mathsf{declare}(x, \mathsf{int}, \\ \quad \mathsf{seq}(\\ \qquad \mathsf{assign}(x, 1), \\ \qquad \mathsf{return}(x))) \end{array}$$

Figure 2: Example program.

$$\dfrac{\dfrac{\dfrac{\{\} \vdash 1}{\{x\};\ \{\} \vdash \mathsf{assign}(x, 1) \Rightarrow \{x\}}\ \text{ASSN} \qquad \dfrac{\{x\} \vdash x}{\{x\};\ \{x\} \vdash \mathsf{return}(x) \Rightarrow \{x\}}\ \text{RET}}{\{x\};\ \{\} \vdash \mathsf{seq}(\mathsf{assign}(x, 1), \mathsf{return}(x)) \Rightarrow \{x\}}\ \text{ASSN}}{\{\};\ \{\} \vdash \mathsf{declare}(x, \mathsf{int}, \mathsf{seq}(\mathsf{assign}(x, 1), \mathsf{return}(x))) \Rightarrow \{\}}\ \text{DECL}$$

Figure 3: Derivation tree for the example program.

For simplicity we are taking the $\Delta \vdash e$ premises for granted (so long as they are true!). Here, $\{\} \vdash 1$ is true because 1 uses no variables, and $\{x\} \vdash x$ is true because the expression uses one variable x, which is in $\{x\}$.

(a) Define a variable set Δ' so that the following judgment is derivable and write a derivation tree for the judgment.

$$\{x, y\};\ \{\} \vdash \mathsf{if}(\mathsf{false}, \mathsf{assign}(x, 1), \mathsf{return}(2)) \Rightarrow \Delta'$$

You do not need to justify any expression initialization premises (the $\Delta \vdash e$ premises), and you do not need to label the rules you use.

(b) Try to derive the following conclusion with a derivation tree.

$$\{\};\ \{\} \vdash \mathsf{seq}(\mathsf{return}(0), \mathsf{declare}(x, \mathsf{int}, \mathsf{return}(x))) \Rightarrow \{\}$$

Explain informally why the derivation fails, by showing a partial derivation and indicating where you got stuck.

3 Proper Returns

For this problem, we define a judgment s **returns**, meaning that s is guaranteed to return during execution. Below is the inference rule for $\mathsf{declare}(x, \tau, s)$:

$$\frac{s\ \textbf{returns}}{\mathsf{declare}(x, \tau, s)\ \textbf{returns}}\ \textsc{Declare}$$

(a) Complete the definition with rules for $\mathsf{assign}(x, e)$, $\mathsf{if}(e, s_1, s_2)$, $\mathsf{while}(e, s)$, $\mathsf{return}(e)$, nop, and $\mathsf{seq}(s_1, s_2)$. Follow the informal definition provided in the L2 handout.

Hint: We express that a statement does *not* return by not including a rule for it. So, some of the constructs will not have an associated rule.

(b) Using your definition, explain why the following program does NOT satisfy the always-returns property:

$$s \triangleq \mathsf{if}(\mathsf{true}, \mathsf{return}(1), \mathsf{assign}(x, 1))$$

Specifically, try to derive s **returns** and show where the derivation fails to make progress.