Assignment 3: Static and Dynamic Semantics

15-411/611: Course Staff

Due Tuesday, October 20, 2020 (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.

Problem 1: Static Semantics (20 points)

In class, we've seen the way that typing judgments are structured. Take, for example, the typing judgment for if statements:

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash s_1 \ valid \quad \Gamma \vdash s_2 \ valid}{\Gamma \vdash \mathsf{if}(e, s_1, s_2) \ valid}$$

Essentially, the judgment says: for a statement $\mathsf{if}(e, s_1, s_2)$, if e is of type bool in context Γ, and s_1, s_2 are $valid$ in Γ, then the whole if statement is $valid$ in context Γ.

We also have the following rule for the ternary (?) operator:

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (e_1 \ ? \ e_2 \ : \ e_3) : \tau}$$

(a) if statements and the ? operator both branch based on a boolean value. Explain why the rule for the if statement judges the statement to be $valid$, while the rule for the ? operator judges the expression to have the type τ.

(b) Suppose we want to add support for integer comparisons to our language syntax. One way to do this is to introduce a new type cmp, which can take on the values lt, eq, and gt. We can also introduce the expression $\texttt{CMP}(e_1, e_2)$. The CMP operator will take in two integers and evaluate to lt, eq, or gt depending on how the arguments compare to each other.

Finally, we will introduce the statement $\mathsf{casecmp}(e, s_1, s_2, s_3)$. The execution of $\mathsf{casecmp}(e, s_1, s_2, s_3)$ evaluates e, and then executes s_1, s_2, or s_3 if e evaluates to lt, eq, and gt respectively. The following rules begin to describe the statics of our new constructs:

$$\frac{}{\Gamma \vdash \mathsf{lt} : \mathsf{cmp}} \qquad \frac{}{\Gamma \vdash \mathsf{eq} : \mathsf{cmp}} \qquad \frac{}{\Gamma \vdash \mathsf{gt} : \mathsf{cmp}}$$

Write down the typing judgments for `CMP` and casecmp.

Problem 2: Generalized Ifs (20 points)

In this problem, assume we're using a subset of the restricted abstract syntax used in lecture, and the corresponding statics and dynamics. For your convenience, these are reproduced below.

Language

$$\begin{array}{llll}
\text{Operators} & \oplus & ::= & + \mid < \\
\text{Expressions} & e & ::= & n \mid x \mid e_1 \oplus e_2 \mid e_1 \&\& e_2 \\
\text{Statements} & s & ::= & \mathsf{assign}(x, e) \mid \mathsf{if}(e, s_1, s_2) \mid \mathsf{while}(e, s) \\
 & & & \mid \mathsf{return}(e) \mid \mathsf{nop} \mid \mathsf{seq}(s_1, s_2) \mid \mathsf{decl}(x, \tau, s)
\end{array}$$

Statics

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{}{\Gamma \vdash n : \mathsf{int}} \qquad \frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 < e_2 : \mathsf{bool}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \mathsf{bool}}{\Gamma \vdash e_1 \&\& e_2 : \mathsf{bool}}$$

$$\frac{\Gamma(x) = \tau' \quad \Gamma \vdash e : \tau'}{\Gamma \vdash \mathsf{assign}(x, e) : [\tau]} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash s_1 : [\tau] \quad \Gamma \vdash s_2 : [\tau]}{\Gamma \vdash \mathsf{if}(e, s_1, s_2) : [\tau]}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash s : [\tau]}{\Gamma \vdash \mathsf{while}(e, s) : [\tau]} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{return}(e) : [\tau]}$$

$$\frac{}{\Gamma \vdash \mathsf{nop} : [\tau]} \qquad \frac{\Gamma \vdash s_1 : [\tau] \quad \Gamma \vdash s_2 : [\tau]}{\Gamma \vdash \mathsf{seq}(s_1, s_2) : [\tau]}$$

$$\frac{\Gamma, x{:}\tau' \vdash s : [\tau]}{\Gamma \vdash \mathsf{decl}(x, \tau', s) : [\tau]}$$

Dynamics

$$
\begin{array}{lll}
\eta \vdash e_1 \oplus e_2 \rhd K & \to & \eta \vdash e_1 \rhd (\_ \oplus e_2, K) \\
\eta \vdash c_1 \rhd (\_ \oplus e_2, K) & \to & \eta \vdash e_2 \rhd (c_1 \oplus \_, K) \\
\eta \vdash c_2 \rhd (c_1 \oplus \_, K) & \to & \eta \vdash c \rhd K \qquad (c = c_1 \oplus c_2) \\
\\
\eta \vdash e_1 \&\& e_2 \rhd K & \to & \eta \vdash e_1 \rhd (\_ \&\& e_2, K) \\
\eta \vdash \mathsf{false} \rhd (\_ \&\& e_2, K) & \to & \eta \vdash \mathsf{false} \rhd K \\
\eta \vdash \mathsf{true} \rhd (\_ \&\& e_2, K) & \to & \eta \vdash e_2 \rhd K \\
\\
\eta \vdash x \rhd K & \to & \eta \vdash \eta(x) \rhd K \\
\\
\eta \vdash \mathsf{assign}(x, e) \blacktriangleright K & \to & \eta \vdash e \rhd (\mathsf{assign}(x, \_), K) \\
\eta \vdash c \rhd (\mathsf{assign}(x, \_), K) & \to & \eta[x \mapsto c] \vdash \mathsf{nop} \blacktriangleright K \\
\\
\eta \vdash \mathsf{decl}(x, \tau, s) \blacktriangleright K & \to & \eta[x \mapsto \mathsf{nothing}] \vdash s \blacktriangleright K \\
\\
\eta \vdash \mathsf{if}(e, s_1, s_2) \blacktriangleright K & \to & \eta \vdash e \rhd (\mathsf{if}(\_, s_1, s_2), K) \\
\eta \vdash \mathsf{true} \rhd (\mathsf{if}(\_, s_1, s_2), K) & \to & \eta \vdash s_1 \blacktriangleright K \\
\eta \vdash \mathsf{false} \rhd (\mathsf{if}(\_, s_1, s_2), K) & \to & \eta \vdash s_2 \blacktriangleright K \\
\\
\eta \vdash \mathsf{while}(e, s) \blacktriangleright K & \to & \eta \vdash \mathsf{if}(e, \mathsf{seq}(s, \mathsf{while}(e, s)), \mathsf{nop}) \blacktriangleright K \\
\\
\eta \vdash \mathsf{return}(e) \blacktriangleright K & \to & \eta \vdash e \rhd (\mathsf{return}(\_), K) \\
\eta \vdash v \rhd (\mathsf{return}(\_), K) & \to & \mathsf{value}(v)
\end{array}
$$

Thinking about C, Jan realizes how convenient it would be to have conditionals operate on any type by implicitly casting them to booleans. For example, we would expect the code fragment

```
if (7) { do_something_fun(); }
else { do_something_not_fun(); }
```

to call `do_something_fun()` in C, as 7 is non-zero. However, in C0 we only have a judgement for when the expression being compared upon is a boolean. To solve this problem, Jan adds a new typing rule

$$\frac{\Gamma \vdash e : \texttt{int} \quad \Gamma \vdash s_1 : [\tau] \quad \Gamma \vdash s_2 : [\tau]}{\Gamma \vdash \texttt{if}(e, s_1, s_2) : [\tau]}$$

However, when he runs a small program using the semantics, the program gets stuck.

```
if (7) {
    return 1;
} else {
    return 0;
}
```

1. What could be wrong?

2. Provide a trace in the format from lecture exposing the problem.
3. Help Jan out and provide a fix for this issue that will allow `if` statements to function as he desires. Ensure that your fix does not break any other features of this language.

Problem 3: Enums (20 Points)

Many programming languages contain enumerations or sets of named constants. These enum constructs appear in languages such as C, C++, and Java, among others.

In C, enumeration types u can be declared as

$$\mathsf{enum}\ u;$$

or defined as

$$\mathsf{enum}\ u\ \{v_1, \ldots, v_n\};$$

where $v_1, \ldots, v_n$ are distinct identifiers, and u is an identifier. Enum values are introduced by named constants v_i, which are now valid expressions. Enum values can be used in switch statements, which take the form

$$\mathsf{switch}(e)\{v_1 \mapsto s_1 \mid \ldots \mid v_n \mapsto s_n\}$$

Informally, a switch statement inspects the enum value that e evaluates to and branches accordingly. In the above example, if e steps to the constant v_1, then the statement s_1 will be executed. If e steps to v_2, then s_2 will be executed. The pattern continues.

Below are a couple of rules that begin to describe the static semantics of enumerations.

$$\frac{?}{\Sigma; \Gamma \vdash \mathsf{switch}(e)\{v_1 \mapsto s_1 \mid \ldots \mid v_n \mapsto s_n\} :?}\ (\mathrm{S1}) \qquad \frac{?}{\Sigma; \Gamma \vdash v :?}\ (\mathrm{S2})$$

The rules use an enumeration signature Σ that contains all defined enumerations. You can assume that every enumeration u and every element v appears at most once in the signature.

$$\Sigma ::= \cdot \mid \mathsf{enum}\ u\ \{v_1, \ldots, v_n\}, \Sigma$$

(a) Complete the type rules for enumerations to maintain the type safety of C0. Hint: one thing that the premises for the rule S1 should check is that the named constants $v_1, \ldots, v_n$ are distinct and exhaustive.

(b) Extend the dynamic semantics for expressions and statements to describe the evaluation of named constants and the execution of switch statements.