Expressions and Statements

Kyle Dewey

1 Introduction

The previous handout considered a language which consisted only of expressions, no statements. In this handout, we use fundamentally the same language, but use statements instead of expressions for `let` and `assign`.

1.1 Language Used

We'll first define the abstract syntax for our language, shown below:

$$x \in \mathit{Variable} \qquad i \in \mathit{Integer}$$

$$\begin{aligned}
\tau \in \mathit{Type} &::= \textsf{int} \mid \textsf{bool} \\
e \in \mathit{Exp} &::= x \mid i \mid \textsf{true} \mid \textsf{false} \mid e_1 \mathbin{\&\&} e_2 \mid e_1 + e_2 \mid e_1 < e_2 \\
s \in \mathit{Stmt} &::= \textsf{let}\ x : \tau = e \mid x = e \\
p \in \mathit{Program} &::= s \mid s\ p
\end{aligned}$$

Compared to the last handout, there are two notable changes:

- `let` and assignment have been made into statements.
- A program is now explicitly defined as either a statement, or a statement followed by another program. Phrased more simply, a program consists of one or more statements.

2 Type System

Now that we have the syntax defined, we can define the language's type system. Formerly, this entailed the definition of a single set of rules which operated over the language's expressions. However, since we now have both statements and expressions, we need to define *two* sets of rules: one for expressions and one for statements. Both sets of rules will need type environments (expressions have variable access, and all statements currently involve variables), so we'll define that first:

$$\Gamma \in \mathit{TypeEnv} = \mathit{Variable} \rightarrow \mathit{Type}$$

Note this is the same type environment definition from last time; this is still a mapping of variables to types.

From here, we define the rules for expressions and statements in the subsections below.

2.1 Rules for Expressions

2.2 Rules for Statements