1 Syntax

Consider the following Mini-C language.

Program  $P ::= (D, s)$
Declarations  $D ::= x : \tau_1; D$
Types  $\tau ::= \text{bool} \mid \text{int}$
Variables  $x, y, \ldots$
Statements  $s ::= x = e \mid s_1; s_2 \mid \text{if } e \text{ then } s_1 \text{ else } s_2 \mid \text{while } e \text{ do } s$
Expressions  $e ::= x \mid v \mid \odot e \mid e_1 \oplus e_2$
Values  $v ::= n \mid \text{true} \mid \text{false}$
Integers  $n$
Unary op  $\odot ::= - \mid !$
Binary op  $\oplus ::= + \mid - \mid * \mid / \mid < \mid > \mid = \mid \land \mid \lor$

Exercise.  Write a program in Mini-C to compute factorial.

2 Static Semantics

\[
\frac{D \triangleright \Gamma \quad \Gamma \vdash s}{\Gamma \vdash (D, s)} \quad \text{TyProg}
\]
\[
\frac{D \triangleright \Gamma}{x : \tau \triangleright \Gamma, (x, \tau)} \quad \text{TyDec}
\]
\[
\frac{\emptyset \triangleright \emptyset}{\quad \text{TyDecEmpty}}
\]
\[
\frac{(x, \tau) \in \Gamma \quad \text{Gamma} \vdash e : \tau}{\Gamma \vdash x = e} \quad \text{TyStmAssign}
\]
\[
\frac{\Gamma \vdash s_1 \quad \text{Gamma} \vdash s_2}{\Gamma \vdash s_1; s_2} \quad \text{TyStmSeq}
\]
Exercise. Give a few examples of typing derivation.

3 Dynamic Semantics

3.1 Small Step Operational Semantics

An abstract program state \((\Delta, C)\) consists of the current computation store \(\Delta\) (maps variables to values) and the code \(C\) needed for the rest of the computation. This can be viewed as an abstract machine. The machine starts with the original program \(P\) and an empty store (memory) \([\_]\). The machine halts when there is no code left to be executed. To make it more readable, a “$” is put there in the abstract machine to indicate it’s done.

\[
\langle \Delta, (x : \text{bool}; D, s) \rangle \mapsto \langle \Delta[(x, \text{false})], (D, s) \rangle \quad \text{EvalProgDeclBool}
\]
\[
\begin{align*}
(\Delta, (x : \text{int}; D, s)) & \mapsto (\Delta[(x, 0)], (D, s)) & \text{EvalProgDeclInt} \\
(\Delta, (\text{nil}, s)) & \mapsto (\Delta, s) & \text{EvalProgDeclNil} \\
(\Delta, x = v) & \mapsto (\Delta[(x, v)], $) & \text{EvalStmAssignVal} \\
\Delta \triangleright e & \mapsto e' & \text{EvalStmAssign} \\
(\Delta, x = e) & \mapsto (\Delta, x = e') & \text{EvalStmSeq} \\
(\Delta, s_1) & \mapsto (\Delta', s'_1) & \text{EvalStmIfTrue} \\
(\Delta, (s_1; s_2)) & \mapsto (\Delta', (s'_1; s_2)) & \text{EvalStmIfFalse} \\
(\Delta, \text{if true then } s_1 \text{ else } s_2) & \mapsto (\Delta, s_1) & \text{EvalStmIf} \\
(\Delta, \text{if false then } s_1 \text{ else } s_2) & \mapsto (\Delta, s_2) & \\
\Delta \triangleright e & \mapsto e' & \text{EvalStmWhileTrue} \\
(\Delta, \text{while } e \text{ do } s) & \mapsto (\Delta, s ; \text{ while } e' \text{ do } s) & \\
\Delta \triangleright e & \mapsto \text{true} & \text{EvalStmWhileFalse} \\
(\Delta, \text{while } e \text{ do } s) & \mapsto (\Delta, $) & \\
\Delta \triangleright e & \mapsto \text{false} &
\end{align*}
\]

Exercise. Work out the small-step typing rules for expressions.

3.2 Big-Step Operational Semantics

See Harper [3, Ch. 7].

4 Type Safety \(=\) Preservation \(+\) Progress

See Chapter 6 of Harper [3] and Chapter 8 of Pierce [5].

5 Homework 3

Problem 1. Add a “halt” statement in the language. Work out typing and evaluation rules and augment the safety proof.

Problem 2. Add a “ref \(\tau\)” type for pointers and a pointer dereference operator “\(*\)” as in C, but with no pointer arithmetic. Work out their typing rules and prove the type safety theorem.
Problem 3. Add a security type qualifier “red $\tau$” which stands for sensitive datatypes. You can use red color instead of text “red.” Work out their typing rules and prove the type safety theorem.

6 Bibliography Notes and Further Reading

Informally speaking, the type safety states that well-typed programs do not “go wrong.” The slogan “safety is preservation plus progress” is due to Harper.

The small-step style operational semantics is sometimes called structural operational semantics (as it operates on the structures of terms) introduced by Plotkin [6]. The big-step style is due to Kahn [4].


References


