

Mini-C: A Simple Typed Imperative Language

(Lecture Supplement for Foundations of Security and Privacy)

Deepak Garg

October 27, 2009

1 Syntax

We divide the syntax into types, declarations, expressions, and commands. There are no pointers or functions.

Types	T	$::=$	<code>int bool</code>
Variables	x, y		
Declarations	Δ	$::=$	<code>$x_1 : T_1; \dots; x_n : T_n$</code>
Integers	n	$::=$	<code>$\dots -1 0 1 \dots$</code>
Expressions	e	$::=$	<code>$x \text{true} \text{false} n e_1 + e_2 e_1 * e_2 e_1 <= e_2 e_1 == e_2$</code>
Commands	c	$::=$	<code><code>noop</code> <code>$x = e$</code> <code>$c_1; c_2$</code> <code>if e then c_1 else c_2</code> <code>while e do c</code></code>
Programs	P	$::=$	<code>decl Δ begin c end</code>
Values	v	$::=$	<code>true false n</code>
Stores	σ	$::=$	<code>$x_1 \mapsto v_1, \dots, x_n \mapsto v_n$</code>

As an example, a mini-C program that computes the factorial of 10 is shown in Figure 1.

2 Static semantics (Typing)

The static semantics of mini-C contains three judgments $\vdash P$ (P is a well-formed program), $\Delta \vdash c$ (c is a well-formed command, given the declarations Δ), and $\Delta \vdash e : T$ (e has type T , given the declarations Δ). Rules are shown in Figure 2. Further, we say that a store σ satisfies Δ , written $\Delta \vdash \sigma$ if:

```
decl
  f : int; c : int; m : int
begin
  n = 10;    // Computer 10!
  f = 1;     // Accumulator
  c = 1;     // Counter
  while c <= m do
    f = f * c;
    c = c + 1
  end

  // Output f here
end
```

Figure 1: Mini-C program to compute factorial of 10

1. $(x : \text{int}) \in \Delta$ implies $(x \mapsto n) \in \sigma$ for some n
2. $(x : \text{bool}) \in \Delta$ implies $(x \mapsto \text{true}) \in \sigma$ or $(x \mapsto \text{false}) \in \sigma$

As an illustration, the derivation which shows how to type the command `while (c <= m) do (f = f * c; c = c + 1)` from the factorial program of Figure 1 is shown in Figure 3. Each rule used in the derivation is an instance of a rule in Figure 2.

3 Dynamic Semantics

The reduction semantics are shown in Figure 4.

4 Preservation

Theorem 4.1 (Preservation for expressions). *Suppose the following hold:*

1. $\Delta \vdash e : T$
2. $\sigma \triangleright e \hookrightarrow e'$
3. $\Delta \vdash \sigma$

Then, $\Delta \vdash e' : T$ and $|e'| < |e|$.

Theorem 4.2 (Preservation for commands). *Suppose the following hold:*

1. $\Delta \vdash c$
2. $\sigma ; c \longrightarrow \sigma ; c'$
3. $\Delta \vdash \sigma$

Then, $\Delta \vdash c'$ and $\Delta \vdash \sigma'$.

5 Progress

Lemma 5.1 (Canonical forms). *If $\Delta \vdash v : \text{int}$, then $v = n$. If $\Delta \vdash v : \text{bool}$, then $v = \text{true}$ or $v = \text{false}$.*

Theorem 5.2 (Progress for expressions). *Suppose the following hold:*

1. $\Delta \vdash e : T$
2. $\Delta \vdash \sigma$

Then either $e = v$, or there is a e' such that $\sigma \triangleright e \hookrightarrow e'$.

Theorem 5.3 (Progress for commands). *Suppose the following hold:*

1. $\Delta \vdash c$
2. $\Delta \vdash \sigma$

Then either $c = \text{noop}$ or there are σ' and c' such that $\sigma ; c \longrightarrow \sigma' ; c'$.

6 Type-Safety

Theorem 6.1 (Safety for expressions). *Suppose the following hold:*

1. $\Delta \vdash e : T$
2. $\Delta \vdash \sigma$

Then there is a value v such that $\sigma \triangleright e \hookrightarrow^ v$.*

Theorem 6.2 (Safety for commands). *Suppose the following hold:*

1. $\Delta \vdash c$
2. $\Delta \vdash \sigma$
3. $\sigma ; c \longrightarrow^* \sigma' ; c'$

Then either $c' = \text{noop}$ or there are σ'' and c'' such that $\sigma' ; c' \longrightarrow \sigma'' ; c''$.

Types for expressions

$$\begin{array}{c}
 \frac{(x : T) \in \Delta}{\Delta \vdash x : T} \qquad \frac{}{\Delta \vdash \text{true} : \text{bool}} \qquad \frac{}{\Delta \vdash \text{false} : \text{bool}} \qquad \frac{}{\Delta \vdash n : \text{int}} \\
 \\
 \frac{\Delta \vdash e_1 : \text{int} \quad \Delta \vdash e_2 : \text{int}}{\Delta \vdash e_1 + e_2 : \text{int}} \qquad \frac{\Delta \vdash e_1 : \text{int} \quad \Delta \vdash e_2 : \text{int}}{\Delta \vdash e_1 * e_2 : \text{int}} \qquad \frac{\Delta \vdash e_1 : \text{int} \quad \Delta \vdash e_2 : \text{int}}{\Delta \vdash e_1 <= e_2 : \text{bool}} \\
 \\
 \frac{\Delta \vdash e_1 : \text{int} \quad \Delta \vdash e_2 : \text{int}}{\Delta \vdash e_1 == e_2 : \text{bool}}
 \end{array}$$

Types for commands

$$\begin{array}{c}
 \frac{}{\Delta \vdash \text{noop}} \qquad \frac{(x : T) \in \Delta \quad \Delta \vdash e : T}{\Delta \vdash x = e} \qquad \frac{\Delta \vdash c_1 \quad \Delta \vdash c_2}{\Delta \vdash c_1 ; c_2} \\
 \\
 \frac{\Delta \vdash e : \text{bool} \quad \Delta \vdash c_1 \quad \Delta \vdash c_2}{\Delta \vdash \text{if } e \text{ then } c_1 \text{ else } c_2} \qquad \frac{\Delta \vdash e : \text{bool} \quad \Delta \vdash c}{\Delta \vdash \text{while } e \text{ do } c}
 \end{array}$$

Types for programs

$$\frac{\Delta \vdash c}{\vdash \text{decl } \Delta \text{ begin } c \text{ end}}$$

Figure 2: Type system of Mini-C

Note: $\Delta = f : \text{int}; c : \text{int}; m : \text{int}$

$$\begin{array}{c}
 \mathcal{D}_1 = \frac{\frac{\overline{(f : \text{int}) \in \Delta}}{\Delta \vdash f : \text{int}} \quad \frac{\overline{(c : \text{int}) \in \Delta}}{\Delta \vdash c : \text{int}}}{\Delta \vdash f * c : \text{int}}}{\Delta \vdash f = f * c} \\
 \\
 \mathcal{D}_2 = \frac{\frac{\overline{(c : \text{int}) \in \Delta}}{\Delta \vdash c : \text{int}} \quad \frac{\overline{\Delta \vdash 1 : \text{int}}}{\Delta \vdash 1 : \text{int}}}{\Delta \vdash c + 1 : \text{int}}}{\Delta \vdash c = c + 1} \\
 \\
 \mathcal{D} = \frac{\frac{\overline{(c : \text{int}) \in \Delta}}{\Delta \vdash c : \text{int}} \quad \frac{\overline{(m : \text{int}) \in \Delta}}{\Delta \vdash m : \text{int}} \quad \frac{\mathcal{D}_1 \quad \mathcal{D}_2}{\Delta \vdash f = f * c \quad \Delta \vdash c = c * 1}}{\Delta \vdash c \leq m : \text{bool}}}{\Delta \vdash \text{while } (c \leq m) \text{ do } (f = f * c; c = c + 1)}
 \end{array}$$

Figure 3: Example of a typing derivation

$\sigma \triangleright e \hookrightarrow e'$

$$\frac{(x \mapsto v) \in \sigma}{\sigma \triangleright x \hookrightarrow v} \quad \frac{\sigma \triangleright e_1 \hookrightarrow e'_1}{\sigma \triangleright e_1 + e_2 \hookrightarrow e'_1 + e_2} \quad \frac{\sigma \triangleright e_2 \hookrightarrow e'_2}{\sigma \triangleright v_1 + e_2 \hookrightarrow v_1 + e'_2} \quad \frac{\text{add}(n_1, n_2) = n}{\sigma \triangleright n_1 + n_2 \hookrightarrow n}$$

$$\frac{\sigma \triangleright e_1 \hookrightarrow e'_1}{\sigma \triangleright e_1 * e_2 \hookrightarrow e'_1 * e_2} \quad \frac{\sigma \triangleright e_2 \hookrightarrow e'_2}{\sigma \triangleright v_1 * e_2 \hookrightarrow v_1 * e'_2} \quad \frac{\text{mult}(n_1, n_2) = n}{\sigma \triangleright n_1 * n_2 \hookrightarrow n}$$

$$\frac{\sigma \triangleright e_1 \hookrightarrow e'_1}{\sigma \triangleright e_1 \leq e_2 \hookrightarrow e'_1 \leq e_2} \quad \frac{\sigma \triangleright e_2 \hookrightarrow e'_2}{\sigma \triangleright v_1 \leq e_2 \hookrightarrow v_1 \leq e'_2} \quad \frac{\text{leq}(n_1, n_2) = b}{\sigma \triangleright n_1 \leq n_2 \hookrightarrow b}$$

$$\frac{\sigma \triangleright e_1 \hookrightarrow e'_1}{\sigma \triangleright e_1 == e_2 \hookrightarrow e'_1 == e_2} \quad \frac{\sigma \triangleright e_2 \hookrightarrow e'_2}{\sigma \triangleright v_1 == e_2 \hookrightarrow v_1 == e'_2} \quad \frac{\text{eq}(n_1, n_2) = b}{\sigma \triangleright n_1 == n_2 \hookrightarrow b}$$

$\sigma; c \longrightarrow \sigma'; c'$

$$\frac{\sigma \triangleright e \hookrightarrow^* v}{\sigma; (x = e) \longrightarrow \sigma[x \mapsto v]; \text{noop}} \quad \frac{\sigma; c_1 \longrightarrow \sigma'; c'_1}{\sigma; (c_1; c_2) \longrightarrow \sigma'; (c'_1; c_2)} \quad \frac{}{\sigma; (\text{noop}; c_2) \longrightarrow \sigma; c_2}$$

$$\frac{\sigma \triangleright e \hookrightarrow^* \text{true}}{\sigma; (\text{if } e \text{ then } c_1 \text{ else } c_2) \longrightarrow \sigma; c_1} \quad \frac{\sigma \triangleright e \hookrightarrow^* \text{false}}{\sigma; (\text{if } e \text{ then } c_1 \text{ else } c_2) \longrightarrow \sigma; c_2}$$

$$\frac{\sigma \triangleright e \hookrightarrow^* \text{true}}{\sigma; (\text{while } e \text{ do } c) \longrightarrow \sigma; (c; (\text{while } e \text{ do } c))} \quad \frac{\sigma \triangleright e \hookrightarrow^* \text{false}}{\sigma; (\text{while } e \text{ do } c) \longrightarrow \sigma; \text{noop}}$$

Figure 4: Reduction semantics of Mini-C