

Hoare Logic

KC Sivaramakrishnan
Spring 2020

IIT
MADRAS



MADRAS IIT

Context

- Previously: Lambda Calculus
 - ◆ This lecture: Prove deeper properties about programs correctness.
 - ◆ Not just absence of crashes (type soundness)
- Go back to an imperative language with valuation and heap (aliasing, pointers)
- Describe the semantics of the program using operational semantics
 - ◆ But directly proving properties on operational semantics becomes tedious
- Hoare Logic: Machinery for proving program correctness *automatically*
 - ◆ Invented by C.A.R.Hoare (the same person who invented quick sort).

Syntax

Numbers	n	\in	\mathbb{N}
Variables	x	\in	Strings
Expressions	e	$::=$	$n \mid x \mid e + e \mid e - e \mid e \times e \mid *[e]$
Boolean expressions	b	$::=$	$e = e \mid e < e$
Commands	c	$::=$	$\text{skip} \mid x \leftarrow e \mid *[e] \leftarrow e \mid c; c$ $\mid \text{if } b \text{ then } c \text{ else } c \mid \{a\}\text{while } b \text{ do } c \mid \text{assert}(a)$
Heap	h	$::=$	$\text{nat} \rightarrow \text{nat}$
Valuation	v	$::=$	$\text{var} \rightarrow \text{nat}$
Assertion	a	$::=$	$\text{heap} \rightarrow \text{valuation} \rightarrow \text{Prop}$

Semantics of Expressions

$$\begin{aligned} \llbracket n \rrbracket(h, v) &= n \\ \llbracket x \rrbracket(h, v) &= v(x) \\ \llbracket e_1 + e_2 \rrbracket(h, v) &= \llbracket e_1 \rrbracket(h, v) + \llbracket e_2 \rrbracket(h, v) \\ \llbracket e_1 - e_2 \rrbracket(h, v) &= \llbracket e_1 \rrbracket(h, v) - \llbracket e_2 \rrbracket(h, v) \\ \llbracket e_1 \times e_2 \rrbracket(h, v) &= \llbracket e_1 \rrbracket(h, v) \times \llbracket e_2 \rrbracket(h, v) \\ \llbracket *e \rrbracket(h, v) &= h(\llbracket e \rrbracket(h, v)) \\ \llbracket e_1 = e_2 \rrbracket(h, v) &= \llbracket e_1 \rrbracket(h, v) = \llbracket e_2 \rrbracket(h, v) \\ \llbracket e_1 < e_2 \rrbracket(h, v) &= \llbracket e_1 \rrbracket(h, v) < \llbracket e_2 \rrbracket(h, v) \end{aligned}$$

Semantics of Commands

$$\frac{}{(h, v, \text{skip}) \Downarrow (h, v)} \quad \frac{}{(h, v, x \leftarrow e) \Downarrow (h, v[x \mapsto \llbracket e \rrbracket(h, v)]])}$$

$$\frac{}{(h, v, *[e_1] \leftarrow e_2) \Downarrow (h[\llbracket e_1 \rrbracket(h, v) \mapsto \llbracket e_2 \rrbracket(h, v)], v)}$$

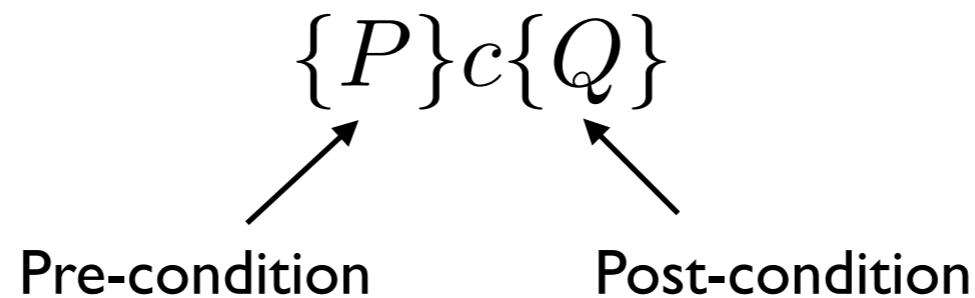
$$\frac{(h, v, c_1) \Downarrow (h_1, v_1) \quad (h_1, v_1, c_2) \Downarrow (h_2, v_2)}{(h, v, c_1; c_2) \Downarrow (h_2, v_2)}$$

$$\frac{\llbracket b \rrbracket(h, v) \quad (h, v, c_1) \Downarrow (h', v')}{(h, v, \text{if } b \text{ then } c_1 \text{ else } c_2) \Downarrow (h', v')} \quad \frac{\neg \llbracket b \rrbracket(h, v) \quad (h, v, c_2) \Downarrow (h', v')}{(h, v, \text{if } b \text{ then } c_1 \text{ else } c_2) \Downarrow (h', v')}$$

$$\frac{\llbracket b \rrbracket(h, v) \quad (h, v, c; \{I\}\text{while } b \text{ do } c) \Downarrow (h', v')}{(h, v, \{I\}\text{while } b \text{ do } c) \Downarrow (h', v')} \quad \frac{\neg \llbracket b \rrbracket(h, v)}{(h, v, \text{while } b \text{ do } c) \Downarrow (h, v)}$$

$$\frac{a(h, v)}{(h, v, \text{assert}(a)) \Downarrow (h, v)}$$

Hoare Triple



$$P, Q := \text{heap} \rightarrow \text{valuation} \rightarrow \text{Prop}$$

- Capture the *effect* of each command on the valuation and the heap.
 - ✦ Pre- and post-conditions are an abstraction of the program behaviour
- Use this to build up to the *effect* of the entire program

Hoare Triple

$$\frac{}{\{P\}\text{skip}\{P\}} \quad \frac{\{P\}c_1\{Q\} \quad \{Q\}c_2\{R\}}{\{P\}c_1; c_2\{R\}} \quad \frac{\forall s. P(s) \Rightarrow I(s)}{\{P\}\text{assert}(I)\{P\}}$$

$$\{P\}x \leftarrow e \{ \lambda(h, v). \exists v'. P(h, v') \wedge v = v' [x \mapsto \llbracket e \rrbracket(h, v')] \}$$

$$\{P\}*[e_1] \leftarrow e_2 \{ \lambda(h, v). \exists h'. P(h', v) \wedge h = h' [\llbracket e_1 \rrbracket(h', v) \mapsto \llbracket e_2 \rrbracket(h', v)] \}$$

$$\frac{\{ \lambda s. P(s) \wedge \llbracket b \rrbracket(s) \} c_1 \{ Q_1 \} \quad \{ \lambda s. P(s) \wedge \neg \llbracket b \rrbracket(s) \} c_2 \{ Q_2 \}}{\{P\}\text{if } b \text{ then } c_1 \text{ else } c_2 \{ \lambda s. Q_1(s) \vee Q_2(s) \}}$$

[Consequence]
$$\frac{\{P\}c\{Q\} \quad (\forall s. P'(s) \Rightarrow P(s)) \quad (\forall s. Q(s) \Rightarrow Q'(s))}{\{P'\}c\{Q'\}}$$

Hoare Triple

$$\frac{(\forall s. P(s) \Rightarrow I(s)) \quad \{\lambda s. I(s) \wedge \llbracket b \rrbracket(s)\}c\{I\}}{\{P\}\{I\}\text{while } b \text{ do } c\{\lambda s. I(s) \wedge \neg \llbracket b \rrbracket(s)\}}$$

Soundness

- Connect Hoare Triple with Operational Semantics
 - ◆ Similar to types and operational semantics
 - ◆ What is the analogy of “well-typed programs do not crash” here?

THEOREM 12.2 (Soundness of Hoare logic). *If $\{P\}c\{Q\}$, $(h, v, c) \Downarrow (h', v')$, and $P(h, v)$, then $Q(h', v')$.*

Hoare Logic + Small-step

- As we know, big step operational semantics can only deal with terminating programs
- Hoare Logic naturally applies to small step semantics as well

Small-step Operational Semantics

$$\overline{(h, v, x \leftarrow e) \rightarrow (h, v[x \mapsto \llbracket e \rrbracket(h, v)], \text{skip})}$$

$$\overline{(h, v, *[e_1] \leftarrow e_2) \rightarrow (h[\llbracket e_1 \rrbracket(h, v) \mapsto \llbracket e_2 \rrbracket(h, v)], v, \text{skip})}$$

$$\overline{(h, v, \text{skip}; c_2) \rightarrow (h, v, c_2)} \quad \frac{(h, v, c_1) \rightarrow (h', v', c'_1)}{\overline{(h, v, c_1; c_2) \rightarrow (h', v', c'_1; c_2)}}$$

$$\frac{\llbracket b \rrbracket(h, v)}{\overline{(h, v, \text{if } b \text{ then } c_1 \text{ else } c_2) \rightarrow (h, v, c_1)}} \quad \frac{\neg \llbracket b \rrbracket(h, v)}{\overline{(h, v, \text{if } b \text{ then } c_1 \text{ else } c_2) \rightarrow (h, v, c_2)}}$$

$$\frac{\llbracket b \rrbracket(h, v)}{\overline{(h, v, \{I\}\text{while } b \text{ do } c) \rightarrow (h, v, c; \{I\}\text{while } b \text{ do } c)}} \quad \frac{\neg \llbracket b \rrbracket(h, v)}{\overline{(h, v, \{I\}\text{while } b \text{ do } c) \rightarrow (h, v, \text{skip})}}$$

$$\frac{a(h, v)}{\overline{(h, v, \text{assert}(a)) \rightarrow (h, v, \text{skip})}}$$

Invariant Safety

- Small-step semantics is said to be *stuck* when the command is not *Skip*, but no way to take a step.
 - ♦ In lambda calculus, $0 + (\lambda x.x)$. What is an example of stuck expression in our language?

$$\frac{a(h, v)}{(h, v, \text{assert}(a)) \rightarrow (h, v, \text{skip})}$$

THEOREM 12.6 (Invariant Safety). *If $\{P\}c\{Q\}$ and $P(h, v)$, then unstuckness is an invariant for the small-step transition system starting at (h, v, c) .*

LEMMA 12.3 (Progress). *If $\{P\}c\{Q\}$ and $P(h, v)$, then (h, v, c) is unstuck.*

LEMMA 12.5 (Preservation). *If $\{P\}c\{Q\}$, $(h, v, c) \rightarrow (h', v', c')$, and $P(h, v)$, then $\{\lambda s. s = (h', v')\}c'\{Q\}$.*