CS477 Formal Software Dev Methods

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides based in part on previous lectures
by Mahesh Vishwanathan, and by Gul Agha

March 27, 2020

Model For Hoare Logic

- Seen proof system for Hoare Logic
- What about models?
- Informally, triple modeled by
  - pairs of assignments of program variables to values
  - where executing program starting with initial assignment results in a memory that gives the final assignment
- Calls for alternate definition of execution

Natural Semantics Models Hoare Logic (Soundness)

Definition

Say a pair of states (aka assignments) (m_1, m_2) satsifies, or models the Hoare triple $\{P\}\ C\ \{Q\}$ if whenever $m_1 \models P$ and $(C, m_1) \Downarrow m_2$ we have $m_2 \models Q$. Write $(m_1, m_2) \models \{P\}\ C\ \{Q\}$

Definition

A Hoare triple $\{P\}\ C\ \{Q\}$ is valid, written $\models \{P\}\ C\ \{Q\}$, if for all states m_1 and m_2 we have $(m_1, m_2) \models \{P\}\ C\ \{Q\}$.

Theorem

Let $\{P\}\ C\ \{Q\}$ be a provable Hoare triple. Then $\models \{P\}\ C\ \{Q\}$.

Natural Semantics Models Hoare Logic (Completeness)

Theorem

Let $\{P\}\ C\ \{Q\}$ be a valid Hoare triple. Then $\{P\}\ C\ \{Q\}$ is provable in Hoare logic.

Isabelle Theory:
`Hoare_sound_and_complete.thy`

Simple Imperative Programming Language #2

$$\begin{array}{lcl}
I & \in & \mathit{Identifiers} \\
N & \in & \mathit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \mid I ::= E \\
B & ::= & \mathsf{true} \mid \mathsf{false} \mid B \& B \mid B\ \mathsf{or}\ B \mid \mathsf{not}\ B \\
 & & \mid E < E \mid E = E \\
C & ::= & C; C \mid \{C\} \mid E \mid \mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ C\ \mathsf{fi} \\
 & & \mid \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}
\end{array}$$

Changes for Expressions

- Need new type of *result* for expressions

$$(E, m) \Downarrow (v, m')$$

- New rule for assignments as expressions:

$$\frac{(E, m) \Downarrow (V, m')}{(I ::= E, m) \Downarrow (V, m'[I \leftarrow V])}$$

- Modify old rules for expressions:

  Atomic Expressions:

  $(I, m) \Downarrow (m(I), m) \quad (N, m) \Downarrow (N, m)$

  Binary Operators:

  $$\frac{(E, m) \Downarrow (U, m') \quad (E', m') \Downarrow (V, m'') \quad U \oplus V = N}{(E \oplus E', m) \Downarrow (N, m'')}$$

Relations

- Must thread state through the relations:

$$\frac{(E, m) \Downarrow (U, m') \quad (E', m') \Downarrow (V, m'') \quad U \sim V = b}{(E \sim E', m) \Downarrow (b, m'')}$$

Changes for Boolean Expressions

- Arithmetic Expressions occur in Boolean Expresion; must change type of result for Boolens:

$$(B, m) \Downarrow (b, m')$$

- Modify old rules for Booleans to reflect new type:

  Atomic Booleans:

$$(\text{true}, m) \Downarrow (\text{true}, m)$$

$$(\text{false}, m) \Downarrow (\text{false}, m)$$

Changes for Boolean Expressions

$$\frac{(B, m) \Downarrow (\text{false}, m')}{(B\&B', m) \Downarrow (\text{false}, m')} \qquad \frac{(B, m) \Downarrow (\text{true}, m') \quad (B', m') \Downarrow (b, m'')}{(B\&B', m) \Downarrow (b, m'')}$$

$$\frac{(B, m) \Downarrow (\text{true}, m')}{(B \text{ or } B', m) \Downarrow (\text{true}, m')} \qquad \frac{(B, m) \Downarrow (\text{false}, m') \quad (B', m') \Downarrow (b, m'')}{(B \text{ or } B', m) \Downarrow (b, m'')}$$

$$\frac{(B, m) \Downarrow (\text{true}, m')}{(\text{not } B, m) \Downarrow (\text{false}, m')} \qquad \frac{(B, m) \Downarrow (\text{false}, m')}{(\text{not } B, m) \Downarrow (\text{true}, m')}$$

Changes for Commands

- Replace rule for Assignment by one for Expressions as Commands:

$$\frac{(E, m) \Downarrow (v, m')}{(E, m) \Downarrow m'}$$

- Unfortunately, can't stop there
  - `if_then_else` and `while` use Booleans; must be changed

Revised if_then_else Rule

$$\frac{(B, m) \Downarrow (\text{true}, m') \quad (C, m') \Downarrow m''}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m''}$$

$$\frac{(B, m) \Downarrow (\text{false}, m' \quad (C', m') \Downarrow m''}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m''}$$

Revised while Rule

$$\frac{(B, m) \Downarrow (\text{false}, m')}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m'}$$

$$\frac{(B, m) \Downarrow (\text{true}, m') \quad (C, m') \Downarrow m'' \quad (\text{while } B \text{ do } C \text{ od}, m'') \Downarrow m'''}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m'''}$$

Termination and Errors in Natural Semantics

- (C,m), (E,m), (B,m) called configurations
- A configuration c evaluates to a result r if $c \Downarrow r$.
- If a configuration c evaluates to a result r, then c terminates without error
- Problem: Can not distinguish between nontermination (*e.g.* a while loop that runs forever), versus and error (*e.g.* referencing an unassigned value
- Can be (partially) remedied by adding error result
  - Roughly doubles number of rules

Transition Semantics

- Aka "small step structured operational semantics"
- Defines a relation of "one step" of computation, instead of complete evaluation
  - Determines granularity of atomic computaions
- Typically have two kinds of "result": configurations and final values
- Written $(C, m) \to (C', m')$ or $(C, m) \to m'$

Simple Imperative Programming Language #1 (SIMPL1)

$$\begin{array}{lcl}
I & \in & \textit{Identifiers} \\
N & \in & \textit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \\
B & ::= & \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B \\
 & & \mid E < E \mid E = E \\
C & ::= & \text{skip} \mid C; C \mid \{C\} \mid I ::= E \\
 & & \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \\
 & & \mid \text{while } B \text{ do } C \text{ od}
\end{array}$$

Transitions for Atomic Expressions

Identifiers:	$(I, m) \longrightarrow m(I)$
Numerals are values:	$(N, m) \longrightarrow N$
Booleans:	$(\text{true}, m) \longrightarrow \text{true}$
	$(\text{false}, m) \longrightarrow \text{false}$

Booleans:

- Values = {true, false}
- Operators: (short-circuit)

$$\begin{array}{c} (\text{false}\& B, m) \longrightarrow \text{false} \\ (\text{true}\& B, m) \longrightarrow (B, m) \end{array} \qquad \frac{(B, m) \longrightarrow (B'', m)}{(B \& B', m) \longrightarrow (B'' \& B', m)}$$

$$\begin{array}{c} (\text{true or } B, m) \longrightarrow \text{true} \\ (\text{false or } B, m) \longrightarrow (B, m) \end{array} \qquad \frac{(B, m) \longrightarrow (B'', m)}{(B \text{ or } B', m) \longrightarrow (B'' \text{ or } B', m)}$$

$$\begin{array}{c} (\text{not true}, m) \longrightarrow \text{false} \\ (\text{not false}, m) \longrightarrow \text{true} \end{array} \qquad \frac{(B, m) \longrightarrow (B', m)}{(\text{not } B, m) \longrightarrow (\text{not } B', m)}$$

Relations

- Let U, V be arithmetic values

$$\frac{(E, m) \longrightarrow (E'', m)}{(E \sim E', m) \longrightarrow (E'' \sim E', m)}$$

$$\frac{(E, m) \longrightarrow (E', m)}{(V \sim E, m) \longrightarrow (V \sim E', m)}$$

$$(U \sim V, m) \longrightarrow b$$

where $U \sim V = b$

Arithmetic Expressions

$$\frac{(E, m) \longrightarrow (E'', m)}{(E \oplus E', m) \longrightarrow (E'' \oplus E', m)}$$

$$\frac{(E, m) \longrightarrow (E', m)}{(V \oplus E, m) \longrightarrow (V \oplus E', m)}$$

$$(U \oplus V, m) \longrightarrow N$$

where N is the specified value for $U \oplus V$

Commands - in English

- `skip` means done evaluating
- When evaluating an assignment, evaluate expression first
- If the expression being assigned is a value, update the memory with the new value for the identifier
- When evaluating a sequence, work on the first command in the sequence first
- If the first command evaluates to a new memory (ie completes), evaluate remainder with new memory

Commands

Skip: $(\text{skip}, m) \longrightarrow m$

Assignment: $$\frac{(E, m) \longrightarrow (E', m)}{(I ::= E, m) \longrightarrow (I ::= E', m)}$$

$$(I ::= V, m) \longrightarrow m[I \leftarrow V]$$

Sequencing:
$$\frac{(C, m) \longrightarrow (C'', m')}{(C; C', m) \longrightarrow (C''; C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(C; C', m) \longrightarrow (C', m')}$$

Block Command

- Choice of level of granularity:
  - Choice 1: Open a block is a unit of work

$$(\{C\}, m) \longrightarrow (C, m)$$

  - Choice 2: Blocks are syntactic sugar

$$\frac{(C, m) \longrightarrow (C', m')}{(\{C\}, m) \longrightarrow (C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(\{C\}, m) \longrightarrow m'}$$

If Then Else Command - in English

- If the boolean guard in an `if_then_else` is true, then evaluate the first branch
- If it is false, evaluate the second branch
- If the boolean guard is not a value, then start by evaluating it first.

If Then Else Command

$$(\text{if true then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C, m)$$

$$(\text{if false then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C', m)$$

$$\frac{(B, m) \longrightarrow (B', m)}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (\text{if } B' \text{ then } C \text{ else } C' \text{ fi}, m)}$$

While Command

$$(\text{while } B \text{ do } C \text{ od}, m)$$
$$\longrightarrow$$
$$(\text{if } B \text{ then } C; \text{while } B \text{ do } C \text{ od else skip fi}, m)$$

- In English: Expand a `while` into a test of the boolean guard, with the true case being to do the body and then try the while loop again, and the false case being to stop.

Example

$$(\text{y := i; while i > 0 do \{i := i - 1; y := y * i\} od}, \langle \text{i} \mapsto 3 \rangle)$$

$$\longrightarrow \underline{\ ?\ }$$

Alternate Semantics for SIMPL1

- Can mix Natural Semantics with Transition Semantics to get larger atomic computations
- Use $(E, m) \Downarrow v$ and $(B, m) \Downarrow b$ for arithmetics and boolean expressions
- Revise rules for commmands

Revised Rules for SIMPL1

Skip: $(\text{skip}, m) \longrightarrow m$

Assignment: $\dfrac{(E, m) \Downarrow v}{(I ::= E, m)} \longrightarrow m[I \leftarrow V]$

Sequencing:
$$\frac{(C, m) \longrightarrow (C'', m')}{(C; C', m) \longrightarrow (C''; C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(C; C', m) \longrightarrow (C', m')}$$

Blocks:
$$\frac{(C, m) \longrightarrow (C', m')}{(\{C\}, m) \longrightarrow (C', m')} \qquad \frac{(C, m) \longrightarrow m'}{(\{C\}, m) \longrightarrow m'}$$

If Then Else Command

$$\frac{(B, m) \Downarrow \text{true}}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C, m)}$$

$$\frac{(B, m) \Downarrow \text{false}}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \longrightarrow (C', m)}$$

Transition Semantics for SIMPL2?

- What are the choices and consequences for giving a transition semantics for the Simple Imperative Programming Language #2, SIMP2?

Simple Concurrent Imperative Programming Language

$$
\begin{array}{lcl}
I & \in & \textit{Identifiers} \\
N & \in & \textit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \\
B & ::= & \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B \\
 & & \mid E < E \mid E = E \\
C & ::= & \text{skip} \mid C; C \mid \{C\} \mid I ::= E \mid C \| C' \\
 & & \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \\
 & & \mid \text{while } B \text{ do } C \text{ od}
\end{array}
$$