Hoare Logic and Turing Machines: Revision

COMP2600 — Formal Methods for Software Engineering

Jinbo Huang
Overview

All material from lectures, assignments, tutorials is examinable except

- Weakest precondition
- Generalized while-rule for total correctness
  - Standard while-rule for total correctness is examinable
- Stories about Nulka (the decoy missile), those about Alan Turing

While-rules (for both partial and total correctness) will be provided (as part of questions)

No other notes will be provided

All questions multiple-choice, worth 33% of total mark

- Your proof skills will still be tested
Preparing for the Exam

Review lectures

Work on / review questions from assignments, tutorials, midterm, past exams

If you’re stuck, seek help

- From fellow students
- On the forum
- By emailing me
- In person by setting up an appointment with me
Validity of Hoare Triples

(I) \( \{ x = 2 \} \ x := y + 1 \ \{ y = 1 \} \)

(II) \( \{ y = y + 1 \} \ x := y + 1 \ \{ y = x \} \)

(III) \( \{ true \} \ x := y + 1 \ \{ false \} \)

(IV) \( \{ true \} \ x := y + 1 \ \{ z = 1 \} \)

(V) \( \{ x = y \} \ if \ (x = 0) \ then \ x := y + 1 \ else \ z := y + 1 \)
\[ \{(x = y + 1) \lor (z = x + 1)\} \]

(VI) \( \{ x = y \} \ if \ (x = 0) \ then \ x := y + 1 \ else \ z := y + 1 \)
\[ \{(z = 1) \longrightarrow (x = 1)\} \]

Which of these triples are valid?
Validity of Hoare Triples: General Strategy

Find quick answer by reasoning with the code fragment

In case of negative answer, verify with a counterexample

In case of positive answer, verify with a proof (if you’re unsure about your answer)

\{x = 2\} x := y + 1 \{y = 1\}

- If \(x\) is 2, and we assign \(y + 1\) to \(x\), will \(y\) necessarily be 1 afterwards?
  - No

- Counterexample: If \(y\) is 0 initially, it will be still be 0 afterwards
Validity of Hoare Triples

\{ y = y + 1 \} \ x := y + 1 \ \{ y = x \}

- If $y = y + 1$, and we assign..., will...afterwards?
- Yes!
  - $y = y + 1$ is always false
  - Implication (conditional) is always true when premise is false

- Proof: Single application of assignment rule
Validity of Hoare Triples

\{true\} x := y + 1 \{false\}

- If true holds, and we assign..., will false necessarily hold afterwards?
- No
  - false will never hold
  - Such triple can only be valid if code doesn’t terminate
- This special case doesn’t really require a counterexample
Validity of Hoare Triples

\{true\} \ x := y + 1 \ {z = 1}\ 

- If true holds, and we assign y + 1 to x, will z necessarily be 1 afterwards?
- No
- Counterexample: If z is 0 initially, it will still be 0 afterwards
Validity of Hoare Triples

\{x = y\} \text{ if } (x = 0) \text{ then } x := y + 1 \text{ else } z := y + 1
\{(x = y + 1) \lor (z = x + 1)\}

- If \(x = y\), and we run the if-then-else, will \((x = y + 1) \lor (z = x + 1)\) necessarily hold afterwards?

- Yes
  - If \(x = 0\), then \(x\) will become \(y + 1\), and \(x = y + 1\) will hold
  - If \(x \neq 0\), then \(z\) will become \(y + 1\), which is the same as \(x + 1\) (as \(x = y\)), and hence \(z = x + 1\) will hold
  - Hence the disjunction will always hold

- Proof (exercise)
Validity of Hoare Triples

\{x = y\} \text{ if } (x = 0) \text{ then } x := y + 1 \text{ else } z := y + 1
\{(z = 1) \rightarrow (x = 1)\}

- If \(x = y\), and we run the if-then-else, will \((z = 1) \rightarrow (x = 1)\) necessarily hold afterwards?
  
- Yes (recall that \(A \rightarrow B\) is true when \(A\) is false or \(B\) is true)
  
  - If \(x = 0\), then \(x\) will become \(y + 1\), which is 1 because \(y = x = 0\), meaning that \(x = 1\) is true
  
  - If \(x \neq 0\), then \(z\) will become \(y + 1\), which \(\neq 1\) because \(y = x \neq 0\); hence \(z = 1\) is false

  - Hence the implication will always hold

- Proof (exercise)
Applying While-Rule for Partial Correctness

\[
\{P \land b\} \ S \ \{P\}
\]

\[
\{P\} \text{ while } b \text{ do } S \ \{P \land \neg b\}
\]

Find a condition \( P \) such that

- \( \{P \land b\} \ S \ \{P\} \) can be proved (i.e., is valid)
- The triple in denominator (of the rule) is useful for the rest of the proof

This often involves trial and error
Applying While-Rule for Total Correctness

\[ P \land b \Rightarrow E \geq 0 \quad [P \land b \land (E = n)] \ S [P \land (E < n)] \]

\[ [P] \text{ while } b \text{ do } S [P \land \neg b] \]

Find a condition \( P \) and expression \( E \) such that

- \( P \land b \Rightarrow E \geq 0 \) is true (this part is just math, no Hoare logic involved)
- \([P \land b \land (E = n)] S [P \land (E < n)]\) can be proved (i.e., is valid)
- The triple in denominator (of the rule) is useful for the rest of the proof

Likewise, this often involves trial and error
Partial vs. Total Correctness

\{P\} S \{Q\}

If $P$ holds initially, and $S$ terminates, then $Q$ will necessarily hold afterwards

- Always valid if $S$ does not terminate under precondition $P$
- Always valid if $Q$ is equivalent to $true$

\[P\] S \[Q\]

If $P$ holds initially, then $S$ will terminate, and $Q$ will necessarily hold afterwards

- Always not valid if $S$ does not terminate under precondition $P$
- Not necessarily valid even if $Q$ is equivalent to $true$

Both always valid if $P$ is equivalent to $false$
Turing Machines

Only way to accept a string: Enter a final state (and halt)

Two ways to reject a string

- Halt in a non-final state (because no move is available)
- Run forever

Language of TM: Set of strings accepted

- Notation: $L(M)$ where $M$ is the TM

Every TM can be encoded, and regarded, as a string, and given to another TM as input

Can also give a pair $(M, w)$ to a TM as input, where $M$ is a TM and $w$ is any string
Computability

Language is *recursively enumerable* if accepted by some TM

- That TM may run forever given an input string not in the language

Language is *decidable* if accepted by some TM that halts on all inputs

- Given any string, that TM will always accept/reject the string in finite time

*Diagonalization* language $L_d = \{w_i \mid w_i \notin L(M_i)\}$ is not recursively enumerable (here $w_i$ is the string encoding TM $M_i$)

*Universal* language $L_u = \{(M, w) \mid w \in L(M)\}$ is recursively enumerable, but not decidable
Proving Undecidability by Reduction

Knowing that $A$ is undecidable, if we can show that a way to decide $B$ would give rise to a way to decide $A$, then we have proved that $B$ is also undecidable.

Works also in relation to recursive enumerability.

Knowing that $A$ is not recursively enumerable, if we can show that a TM to accept $B$ would give rise to a TM to accept $A$, then we have proved that $B$ is also not recursively enumerable.

In both cases, this is called reducing problem $A$ to problem $B$. 
Rice’s Theorem

All nontrivial properties about the language of a TM are undecidable.

Property is nontrivial if some TMs have it, and some don’t.

Example questions about trivial properties:

- Does a given TM accept a recursively enumerable language?
  - Trivial, as all TMs do; the answer is always yes.

- Does a given TM accept a string of infinite length?
  - Trivial, as no TM does; the answer is always no.

Theorem applies to the language only; other aspects of TM can be decidable.

- Does a given TM ever make more than five moves? Formally, is the set of (codes for) TMs that sometimes make more than five moves decidable?
Important Note

You are responsible for all examinable material, not just the bits that have been reviewed today.