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 11, 2020

Demo: `Hoare_ex`

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application
- Can we automate all but this?

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application
- Can we automate all but this?
- Yes! But how?

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application
- Can we automate all but this?
- Yes! But how?

1. Annotate all `while` loops with needed `invariants`

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application
- Can we automate all but this?
- Yes! But how?

1. Annotate all `while` loops with needed `invariants`
2. Use routine to "roll back" post-condition to weakest precondition, gathering side-conditions as we go

Algortihm for Proving Hoare Triples?

- Have seen in Isabelle that much of proving a Hoare triple is routine
- Will this always work?
- Why not automate the whole process?
  - Can't (always) calculate needed loop invariants
  - Can't (always) prove implications (side-conditions) in Rule of Consequence application
- Can we automate all but this?
- Yes! But how?

1. Annotate all `while` loops with needed `invariants`
2. Use routine to "roll back" post-condition to weakest precondition, gathering side-conditions as we go

- 2 called verification condition generation

Annotated Simple Imperative Language

- Give verification conditions for an annotated version of our simple imperative language
- Add a presumed invariant to each while loop

$$\begin{array}{ll}\langle command\rangle & ::= \langle variable\rangle := \langle term\rangle \\ & \mid \langle command\rangle;\ \ldots;\ \langle command\rangle \\ & \mid \mathit{if}\ \langle statement\rangle\ \mathit{then}\ \langle command\rangle\ \mathit{else}\ \langle command\rangle \\ & \mid \mathit{while}\ \langle statement\rangle\ \mathit{inv}\ \langle statement\rangle\ \mathit{do}\ \langle command\rangle\end{array}$$

Example:

```
while y < n inv x = y * y
do
 x := (2 * y) + 1;
 y := y + 1
od
```


HOL Type for Deep Part of Embedding

```
datatype 'data annotated_command =
   AnnAssignCom "var_name" "'data exp"
                (infix ":=" 110)
 | AnnSeqCom "'data annotated_command"
             "'data annotated_command"
             (infixl ";;" 109)
 | AnnCondCom "'data bool_exp"
               "'data annotated_command"
               "'data annotated_command"
        ("If _/ Then _/ Else _/ Fi" [70,70,70]70)
 | AnnWhileCom "'data bool_exp" "'data annotated_command"
        ("While _/ Inv _/  Do _/ Od" [70,70]70)
```


Hoare Logic for Annotated Programs

Assingment Rule

$$\frac{}{\{|P[e/x]|\}\ x\ :=\ e\ \{|P|\}}$$

Rule of Consequence

$$\frac{P \Rightarrow P' \quad \{|P'|\}\ C\ \{|Q'|\} \quad Q' \Rightarrow Q}{\{|P|\}\ C\ \{|Q|\}}$$

Sequencing Rule

$$\frac{\{|P|\}\ C_1\ \{|Q|\} \quad \{|Q|\}\ C_2\ \{|R|\}}{\{|P|\}\ C_1;\ C_2\ \{|R|\}}$$

If Then Else Rule

$$\frac{\{|P \wedge B|\}\ C_1\ \{|Q|\} \quad \{|P \wedge \neg B|\}\ C_2\ \{|Q|\}}{\{|P|\}\ \mathit{if}\ B\ \mathit{then}\ C_1\ \mathit{else}\ C-2\ \{|Q|\}}$$

While Rule

$$\frac{\{|P \wedge B|\}\ C\ \{|P|\}}{\{|P|\}\ \mathit{while}\ B\ \mathit{inv}\ P\ \mathit{do}\ C\ \{|P \wedge \neg B|\}}$$

Defining Hoare Logic Rules

```
inductive ann_valid :: "'data bool_exp ⇒
'data annotated_command ⇒'data bool_exp ⇒bool"
("⦃_⦄_⦃_⦄" [60,60,60]60)where
AnnAssignmentAxiom:"⦃(P[x⇐e])⦄(x:=e) ⦃P⦄" |
AnnSequenceRule:
"⟦⦃P⦄C ⦃Q⦄; ⦃Q⦄C' ⦃R⦄⟧⟹⦃P⦄(C;;C')⦃R⦄" |
AnnRuleOfConsequence:
"⟦|⊨(P [⟶] P') ; ⦃P'⦄C⦃Q'⦄; |⊨(Q' [⟶] Q) ⟧
⟹⦃P⦄C⦃Q⦄" |
AnnIfThenElseRule:
"⟦⦃(P [∧] B)⦄C⦃Q⦄; ⦃(P[∧]([¬]B))⦄C'⦃Q⦄⟧
⟹⦃P⦄(If B Then C Else C' Fi)⦃Q⦄" |
AnnWhileRule:
"⟦⦃(P [∧] B)⦄C⦃P⦄⟧
⟹⦃P⦄(While B Inv P Do C Od)⦃(P [∧] ([¬]B))⦄"
```


Relation Between Two Languages

- Hoare Logic for Simple Imperative Programs and Hoare Logic for Annotated Programs almost the same
- What it precise relationship?
- First need precise relation between the two languages

Definition

strip($v := e$) = $v := e$
strip(C_1 ; C_2) = strip(C_1) ; strip(C_2)
strip(if B then C_1 else C_2 fi) =
  if B then strip(C_1) else strip(C_2) fi
strip(while B inv P do C od) = while B do strip(C) od

- We recursively remove all invariant annotations from all `while` loops

Relation Between Two Hoare Logics

Theorem

For all pre- and post-conditions P and Q, and annotated programs C, if $⦃P⦄\ C\ ⦃Q⦄$, then $\{P\}\ strip(C)\ \{Q\}$.

Proof.

(Sketch) Use rule induction on proof of $⦃P⦄\ C\ ⦃Q⦄$; in case of While Rule, erase invariant □

Relation Between Two Hoare Logics

Theorem

For all pre- and post-conditions P and Q, and unannotated programs C, if $\{P\}\ C\ \{Q\}$, then there exists an annotated program S such that $C = strip(S)$ and $⦃P⦄\ S\ ⦃Q⦄$.

Proof.

(Sketch) Use rule induction on proof of $\{P\}\ C\ \{Q\}$; in case of While Rule, add invariant from precondition as invariant to command. □

Weakest Precondition

Question: Given post-condition Q, and annotated program C, what is the most general pre-condition P such that $⦃P⦄\ C\ ⦃Q⦄$?

Answer: Weakest Precondition

Definition

wp $(x := e)$ $Q = Q[x \Leftarrow e]$
wp $(C_1; C_2)$ Q = wp C_1 (wp C_2 Q)
wp (if B then C_1 else C_2 fi) Q =
  $(B \wedge (\text{wp } C_1\ Q)) \vee ((\neg B) \wedge (\text{wp } C_2\ Q))$
wp (while B inv P do C od) $Q = P$

Assumes, without verifying, that P is the correct invariant

Weakest Justification

Weakest in weakest precondition means any other valid precondition implies it:

Theorem

For all annotated programs C, and pre- and post-conditions P and Q, if $⦃P⦄\ C\ ⦃Q⦄$ then $P \Rightarrow wp\ C\ Q$.

- Proof somewhat complicated
- Uses induction on the structure of C
- In each case, want to assert triple proof must have used rule for that construct (e.g. Sequence Rule for sequences)
- Can't because of Rule Of Consequence
- Must induct on proof (rule induction) - in each case
- Uses:

Lemma

$\forall C\ P\ Q.\ (P \Rightarrow Q) \Rightarrow (wp\ C\ P \Rightarrow wp\ C\ Q)$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \text{true}$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \text{true}$

$\text{vcg } (C_1; C_2)\ Q = (\text{vcg } C_1\ (\text{wp } C_2\ Q)) \wedge (\text{vcg } C_2\ Q)$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \text{true}$

$\text{vcg } (C_1; C_2)\ Q = (\text{vcg } C_1\ (\text{wp } C_2\ Q)) \wedge (\text{vcg } C_2\ Q)$

$\text{vcg } (\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \texttt{ fi})\ Q = (\text{vcg } C_1\ Q) \wedge (\text{vcg } C_2\ Q)$

What About Precondition?

Question: Do we have $\{\!|\text{wp } C\ Q|\!\}\ C\ \{\!|Q|\!\}$?

Answer: Not always - need to check while-loop side-conditions – verification conditions

Question: How to calculate verification conditions?

Definition

$\text{vcg } (x := e)\ Q = \text{true}$
$\text{vcg } (C_1; C_2)\ Q = (\text{vcg } C_1\ (\text{wp } C_2\ Q)) \wedge (\text{vcg } C_2\ Q)$
$\text{vcg } (\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \texttt{ fi})\ Q = (\text{vcg } C_1\ Q) \wedge (\text{vcg } C_2\ Q)$
$\text{vcg } (\texttt{while } B \texttt{ inv } P \texttt{ do } C \texttt{ od})\ Q =$
$((P \wedge B) \Rightarrow (\text{wp } C\ P)) \wedge (\text{vcg } C\ P) \wedge ((P \wedge (\neg B)) \Rightarrow Q)$

Verification Condition Guarantees wp Precondition

Theorem

$vcg\ C\ Q \Rightarrow \{\!|wp\ C\ Q|\!\}\ C\ \{\!|Q|\!\}$

Proof.

(Sketch)

- Induct on structure of C
- For each case, wind back as we did in specific examples:
  - Assignment: wp C Q exactly what is needed for Assignment Axiom
  - Sequence: Follows from inductive hypotheses, all elim, and modus ponens
  - If_Then_Else: Need to use Precondition Strengthening with each branch of conditional; wp and inductive hypotheses give the needed side conditions
  - While: Need to use Postcondition Weakening, While Rule and Precondition Strengthening

□

Verification Condition Guarantees wp Precondition

Corollary

$((P \Rightarrow wp\ C\ Q) \wedge (vcg\ C\ Q)) \Rightarrow \{\!|P|\!\}\ C\ \{\!|Q|\!\}$

This amounts to a method for proving Hoare triple $\{P\}\ C\ \{Q\}$:

1. Annotate program with loop invariants
2. Calculate wp C Q and vcg C Q (automated)
3. Prove $P \Rightarrow$ wp C Q and vcg C Q

Basic outline of interaction with Boogie: Human does 1, Boogie does 2, Z3 / Simplify / Isabelle + human / ... does 3

For more infomation

- http://research.microsoft.com/en-us/projects/boogie/
- http://research.microsoft.com/en-us/um/people/moskal/pdf/hol-boogie.pdf
- http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/library/HOL/HOL-Hoare/index.html