L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

Deductive Verification II

Burkhart Wolff
wolff@lri.fr

Recall: The role of formal proof

- formal proofs are another technique for program verification
  - based on a model of the underlying programming language, the conformance of a concrete program to its specification can be established

    FOR ALL INPUT DATA AND ALL INITIAL STATES !!!

- formal proofs as verification technique can:
  - verify that a more concrete design-model "fits" to a more abstract design model (construction by formal refinement)
  - verify that a program "fits" to a concrete design model.

Recall: Hoare - Logic

- A means to reason over all input and all states: Is there

A Logic for Programs ???

- We consider the Hoare-Logic, technically an inference system PL + E + A + Hoare

Hoare - Logic: A Proof System for Programs

- Basis: The mini-language „IMP“, (following Glenn Wynskell's Book)

- We have the following commands (*cmd*)
  - the empty command SKIP
  - the assignment x:== E $(x \in V)$
  - the sequential compos. c_1 ; c_2
  - the conditional IF cond THEN c_1 ELSE c_2
  - the loop WHILE cond DO c

  where c, c_1, c_2, are cmd's, V variables,

  E an arithmetic expression, and cond a boolean expression.

Hoare - Logic: A Proof System for Programs

- Core Concept: A Hoare Triple consisting ...
  - of a pre-condition P
  - a post-condition Q
  - and a piece of program cmd
  - the triple (P,cmd,Q) is written:

$$\vdash \{P\}\ cmd\ \{Q\}$$

  - *P* and *Q* are formulas over the variables *V*, so they can be seen as set of possible states.

Hoare - Logic: A Proof System for Programs

- Idea: We consider the specification (precond, postcond) and the program together
- The Hoare-Triple says : The program "is conform" to the specification
- More precisely:

$$\vdash \{P\}\ cmd\ \{Q\}$$

If a program *cmd* starts in a state admitted by *P* if it terminates, that the program must reach a state that satisfies *P*.

Hoare - Logic: A Proof System for Programs

- PL + E + A + Hoare (simplified binding) at a glance:

$$\frac{}{\vdash \{P\}\ \mathrm{SKIP}\ \{P\}} \qquad \frac{}{\vdash \{P[x \mapsto E]\}\ \mathrm{x} :== \mathrm{E}\{P\}}$$

$$\frac{\vdash \{P \wedge cond\}\ c\ \{Q\} \quad \vdash \{P \wedge \neg cond\}\ d\ \{Q\}}{\vdash \{P\}\ \mathrm{IF}\ cond\ \mathrm{THEN}\ c\ \mathrm{ELSE}\ d\{Q\}}$$

$$\frac{\vdash \{P\}\ c\ \{Q\} \quad \vdash \{Q\}\ d\ \{R\}}{\vdash \{P\}\ c;\ d\ \{R\}} \qquad \frac{\vdash \{P \wedge cond\}\ c\ \{P\}}{\vdash \{P\}\ \mathrm{WHILE}\ cond\ \mathrm{DO}\ c\ \{P \wedge \neg cond\}}$$

$$\frac{P \rightarrow P' \quad \vdash \{P'\}\ cmd\ \{Q'\} \quad Q' \rightarrow Q}{\vdash \{P\}\ cmd\ \{Q\}}$$

Hoare – Logic: A Proof System for Programs

Let's consider it one by one …

Hoare – Logic: A Proof System for Programs

- The SKIP-rule for the empty statement:

$$\frac{}{\vdash \{P\}\ \mathrm{SKIP}\ \{P\}}$$

well, states do not change ...

Therefore, valid states remain valid.

Hoare - Logic: A Proof System for Programs

- The assignment rule:

$$\frac{}{\vdash \{P[x \mapsto E]\}\ \mathrm{x} :== \mathrm{E}\{P\}}$$

- Example (1):

$$\vdash \{1 \leq x \wedge x \leq 10\}\ x :== x+2\ \{3 \leq x \wedge x \leq 12\}$$

- Is this really an *instance* of the assignment rule ? We calculate:

$(3 \leq x \wedge x \leq 12)\ [x \mapsto x+2]$

$\equiv 3 \leq (x+2) \wedge (x+2) \leq 12$

$\equiv 1 \leq x \wedge x \leq 10$

Hoare - Logic: A Proof System for Programs

- The assignment rule:

$$\overline{\vdash \{P[x \mapsto E]\}\ \mathrm{x} :== \mathrm{E}\{P\}}$$

- Example (2):

  $\vdash$ {true} x:== 2 {x=2}

- Is this really an *instance* of the assignment rule ? We calculate:

  (x=2) [x↦2]

  ≡ 2=2 ≡ true (reflexivity…)

Hoare - Logic: A Proof System for Programs

The conditional-rule:

$$\frac{\vdash \{P \land cond\}\ c\ \{Q\} \qquad \vdash \{P \land \neg cond\}\ d\ \{Q\}}{\vdash \{P\}\ \text{IF}\ cond\ \text{THEN}\ c\ \text{ELSE}\ d\{Q\}}$$

Example (3):

$$\vdash \{true\}\ \text{IF}\ 0 \leq x\ \text{THEN SKIP ELSE}\ x :== -x\ \{0 \leq x\}$$

This can be extended to the formal proof:

Hoare - Logic: A Proof System for Programs

- The conditional-rule:

$$\frac{\vdash \{P \wedge cond\}\ c\ \{Q\} \qquad \vdash \{P \wedge \neg cond\}\ d\ \{Q\}}{\vdash \{P\}\ \text{IF}\ cond\ \text{THEN}\ c\ \text{ELSE}\ d\{Q\}}$$

Example (3):

$$\frac{\dfrac{}{\vdash \{true \wedge 0 \leq x\}\ \text{SKIP}\ \{0 \leq x\}} \qquad \dfrac{\dots}{\vdash \{true \wedge \neg(0 \leq x)\}\ x :== -x\ \{0 \leq x\}}}{\vdash \{true\}\ \text{IF}\ 0 \leq x\ \text{THEN SKIP ELSE}\ x :== -x\ \{0 \leq x\}}$$

Hoare - Logic: A Proof System for Programs

- The *sequence* rule:

$$\frac{\vdash \{P\}\ c\ \{Q\} \qquad \vdash \{Q\}\ d\ \{R\}}{\vdash \{P\}\ c;\ d\ \{R\}}$$

- essentially a relational composition on state sets.

Hoare - Logic: A Proof System for Programs

The rule for the sequence.

Example (4):

$$\overline{\vdash \{true\}\ tm :== 1; (sum :== 1; i :== 0)\ \{tm = 1 \wedge sum = 1 \wedge i = 0\}}$$

This can be extended to the formal proof:

Hoare - Logic: A Proof System for Programs

The rule for the sequence.

Example (4):

$$\dfrac{\dfrac{}{\vdash \{true\}tm :== 1\{tm = 1\}} \qquad \dfrac{\dfrac{}{\vdash \{tm = 1\}sum :== 1\{B\}} \qquad \dfrac{}{\vdash \{B\}\ i :== 0\ \{A\}}}{\vdash \{tm = 1\}\ sum :== 1; i :== 0\ \{A\}}}{\vdash \{true\}\ tm :== 1; (sum :== 1; i :== 0)\ \{tm = 1 \wedge sum = 1 \wedge i = 0\}}$$

where $A = tm = 1 \wedge sum = 1 \wedge i = 0$ and where $B = tm = 1 \wedge sum = 1$.

It is often practical to introduce abbreviations.

Hoare - Logic: A Proof System for Programs

- The while-rule.

$$\frac{\vdash \{P \land cond\}\ c\ \{P\}}{\vdash \{P\}\ \text{WHILE}\ cond\ \text{DO}\ c\ \{P \land \neg cond\}}$$

- This works like an induction: if some P is true after n traversals of the loop and remain true for the n+1 traversal, it must be always true.
- When exiting the loop, the condition cond can on longer hold.
- The predicate P is called an invariant. Note that an invariant can be maintained even if the concrete state changes ! See:

$$\vdash \{1 \leq x \land x \leq 10\}\ \text{WHILE}\ x < 10\ \text{DO}\ x{:}{==}\ x+1\ \{\neg\ (x < 10) \land 1 \leq x \land x \leq 10\}$$

Hoare – Logic: A Proof System for Programs

- The consequence-rule:

$$\frac{P \rightarrow P' \quad \vdash \{P'\}\ cmd\ \{Q'\} \quad Q' \rightarrow Q}{\vdash \{P\}\ cmd\ \{Q\}}$$

Reflects the intuition that P' is a subset of legal states P and Q is a subset of legal states Q'.

This is the only rule that is not determined by the syntax of the program; it can be applied anywhere in the (Hoare-) proof.

Hoare - Logic: A Proof System for Programs

- The consequence-rule:

$$\frac{P \rightarrow P' \quad \vdash \{P'\}\ cmd\ \{Q'\} \quad Q' \rightarrow Q}{\vdash \{P\}\ cmd\ \{Q\}}$$

Example (5) (the continuation of Example (3)):

$$\frac{true \wedge \neg(0 \leq x) \rightarrow (0 \leq -x) \qquad \overline{\vdash \{(0 \leq x)[x \mapsto -x]\}\ x :== -x\ \{0 \leq x\}} \qquad 0 \leq x \rightarrow 0 \leq x}{\vdash \{true \wedge \neg(0 \leq x)\}\ x :== -x\ \{0 \leq x\}}$$

Hoare - Logic: A Proof System for Programs

- The Hoare calculus has a number of implicit consequences, i.e. rules that can be derived from the other ones.

Hoare - Logic: A Proof System for Programs

- A handy derived rule, the False-rule:

$$\frac{}{\vdash \{false\}\ cmd\ \{false\}}$$

- **Proof**: by induction over cmd ! (At the Blackboard)

- A very handy corollary of the False-rule and the consequence-rule is the FalseE-rule:

$$\frac{}{\vdash \{false\}\ cmd\ \{P\}}$$

Hoare – Logic: A Proof System for Programs

- Another handy corollary of the False-rule:

$$\frac{}{\vdash \{P \land \neg cond\}\ \text{WHILE}\ cond\ \text{DO}\ c\ \{P \land \neg cond\}}$$

Proof:

by consequence-rule, while-rule,

P and cond-negation,

False-rule.

This means: If we can not enter into the WHILE-loop, it behaves like a SKIP.

Hoare - Logic: A Proof System for Programs

- Yet another handy corollary of the consequence rule:

$$\frac{P = P' \quad \vdash \{P'\}\ cmd\ \{Q'\} \quad Q' = Q}{\vdash \{P\}\ cmd\ \{Q\}}$$

Proof:

by consequence rule and the fact that $P = P'$ (ou $P \equiv P'$) infers $P \rightarrow P'$

- *Note: We will allow to apply this rule implicitly, thus leveraging local "logical massage" of pre- and post-conditions.*

Hoare - Logic: A Proof System for Programs

- Example (6):

$$\frac{}{\vdash \{true\}\ \text{WHILE}\ true\ \text{DO}\ SKIP\ \{x = 42\}}$$

Hoare - Logic: A Proof System for Programs

- Example (6):

$$\frac{}{\vdash \{true\}\ \text{WHILE}\ true\ \text{DO}\ SKIP\ \{x = 42\}}$$

Proof (bottom up):

$true \wedge \neg true \equiv false$

$$\frac{true \rightarrow true^{\surd} \quad \vdash \{true\}\ \text{WHILE}\ true\ \text{DO SKIP}\ \{false\} \quad false \rightarrow x = 42^{\surd}}{\vdash \{true\}\ \text{WHILE}\ true\ \text{DO SKIP}\ \{x = 42\}}$$

Hoare - Logic: A Proof System for Programs

- Example (6):

$$\frac{}{\vdash \{true\}\ \text{WHILE}\ true\ \text{DO}\ SKIP\ \{x = 42\}}$$

<u>Note</u>:

Hoare-Logic is a calculus for partial correctness; for non-terminating programs, it is possible to prove *anything*!

Hoare - Logic: A Proof System for Programs

- Example (7):

$$\frac{}{\vdash \{true\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{2 \leq x\}}$$

Hoare - Logic: A Proof System for Programs

- Example (7):

  Proof (bottom up):

$$\frac{true \to I \qquad \dfrac{}{\vdash \{I\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x+1\ \{I \wedge \neg(x<2)\}} \qquad I \wedge \neg(x<2) \to 2 \leq x}{\vdash \{true\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x+1\ \{2 \leq x\}}$$

We can't apply the WHILE-rule directly — the only other choice is the consequence rule. Instantiating the invariant variable P by a fresh variable I allows us to bring the triple into a shape that we can apply the WHILE rule later

Hoare - Logic: A Proof System for Programs

- Example (7):

  Proof (bottom up):

$$\frac{true \rightarrow I \qquad \overline{\vdash \{I\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{I \wedge \neg(x < 2)\}} \qquad I \wedge \neg(x < 2) \rightarrow 2 \leq x}{\vdash \{true\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{2 \leq x\}}$$

Now we can apply the while rule.

Hoare - Logic: A Proof System for Programs

- Example (7):

  Proof (bottom up):

$$\dfrac{true \rightarrow I \qquad \dfrac{\overline{\vdash \{I \wedge x < 2\}\ x :== x + 1\ \{I\}}}{\vdash \{I\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{I \wedge \neg(x < 2)\}} \qquad I \wedge \neg(x < 2) \rightarrow 2 \leq x}{\vdash \{true\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{2 \leq x\}}$$

To be sure (entering the while loop) we apply again the consequence rule. For the missing bit, we instantiate I".

Hoare - Logic: A Proof System for Programs

- Example (7):

  Proof (bottom up):

$$\dfrac{true \rightarrow I \qquad \dfrac{\dfrac{I \wedge x < 2 \rightarrow I'' \qquad \vdash \{I''\}\ x :== x+1\ \{I'\} \qquad I' \rightarrow I}{\vdash \{I \wedge x < 2\}\ x :== x+1\ \{I\}}}{\vdash \{I\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x+1\ \{I \wedge \neg(x < 2)\}} \qquad I \wedge \neg(x < 2) \rightarrow 2 \leq x}{\vdash \{true\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x+1\ \{2 \leq x\}}$$

Now, in order to make the assignment rule "fit", we must have $I'' \equiv I'[x \mapsto x+1]$.

Hoare - Logic: A Proof System for Programs

- Example (7):

  Proof (bottom up):

$$\dfrac{\text{true} \rightarrow I \quad \dfrac{\dfrac{I \wedge x < 2 \rightarrow I'' \quad \overline{\vdash \{I''\}\ x :== x + 1\ \{I'\}} \quad I' \rightarrow I}{\vdash \{I \wedge x < 2\}\ x :== x + 1\ \{I\}}}{\vdash \{I\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{I \wedge \neg(x < 2)\}} \quad I \wedge \neg(x < 2) \rightarrow 2 \leq x}{\vdash \{\text{true}\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{2 \leq x\}}$$

Additionally, in order that this constitutes a Hoare-Proof, we must have all the implications.

Hoare - Logic: A Proof System for Programs

- Example (7):

$$\frac{}{\vdash \{true\}\ \text{WHILE}\ x < 2\ \text{DO}\ x :== x + 1\ \{2 \leq x\}}$$

So, we have a Hoare Proof iff we have a solution to the following list of constraints:

$$I'' \equiv I'[x \mapsto x+1]$$

$$A \equiv true \rightarrow I$$

$$B \equiv I \wedge \neg(x < 2) \rightarrow 2 \leq x$$

$$C \equiv I \wedge x < 2 \rightarrow I'[x \mapsto x+1]$$

Hoare - Logic: A Proof System for Programs

- Example (7):

  Proof:

$$I'' \equiv I'[x \mapsto x+1]$$

$$A \equiv true \rightarrow I$$

$$B \equiv I \wedge \neg(x < 2) \rightarrow 2 \leq x$$

$$C \equiv I \wedge x < 2 \rightarrow I'[x \mapsto x+1]$$

$$D = I' \rightarrow I$$

  - I must be *true*, this solves A, B, D
  - we are fairly free for a solution for I'; e.g. $x \leq 2$ or $x \leq 5$ would do the trick !

Hoare - Logic: Some facts.

Assume that we have a reasonably well-defined "compiler function" that maps a program to a relation from input to output states:

$$C : \text{cmd} \rightarrow (\sigma \times \sigma)\text{Set}$$

(See Winskell's Book)

Then we can define the "validity" of a specification:

$$\models \{P\}\ cmd\ \{Q\} \equiv\ \forall \sigma, \sigma'. (\sigma, \sigma') \in C(cmd) \rightarrow P(\sigma) \rightarrow Q(\sigma')$$

Hoare - Logic: A Proof System for Programs

- Remarks:

  This proof rises the idea of particular construction method of Hoare-Proofs, which can be automated:

  - apply bottom-up all rules following the cmd-syntax; introduce fresh variables for the wholes where necessary
  - apply the consequence rule only at entry points of loops (this is deterministic!)
  - extract the implications used in these consequence rule
  - try to find solutions for these implications

    (worst case: ask the user ...)

  - Essence of all: again, we reduced a program verification problem to a constraint resolution problem of formulas ...
  - ... provided we have solutions for the invariants.

Hoare - Logic: Some facts.

Theorem: Correctness of the Hoare-Calculus:

$$\vdash \{P\}\ cmd\ \{Q\} \rightarrow\ \models \{P\}\ cmd\ \{Q\}$$

... so, whenever there is a proof, it is also valid wrt. C.

Hoare - Logic: Some facts.

Theorem: Relative Completeness of the Hoare-Calculus

$$\models \{P\}\ cmd\ \{Q\} \ \rightarrow\ \vdash \{P\}\ cmd\ \{Q\}$$

Amazingly, this holds also the other way round: whenever a specification is valid, (and we can solve all the implications on arithmetics), there is a Hoare-Proof.

Hoare - Logic: Summary

- … in the essence, the Hoare Calculus is an entirely syntactic game that constructs a labelling of the program with assertions …

Hoare-Logic : Summary

- Note: Validity is a « partial correctness notion »

  proof under condition that the program terminates.
  For non-terminating programs, the calculus allows
  to prove anything

- The Deductive Proof-Method is therefore two-staged:
  - verify termination (find mesures for loops and recursive calls that strictly decrease for each iteration)
  - prove partial correctness of the spec for the program via a Hoare-Calculus (or an alternative such as the wp-calculus)

total correctness = partial correctness + termination ...

Hoare - Logic: Summary

Formal Proof

- Can be very hard - up to infeasible (nobody will probably ever prove the correctness of MS Word!)

- But still, the proof-task can be automated to a large extent.