2017-2018

Cycle Ingénieur – 2ème année

Département Informatique

Verification and Validation

Part IV : Proof-based Verification (II)

Burkhart Wolff

Département Informatique

Université Paris-Sud / Orsay

Hoare – Logic: A Proof System for Programs

- Now, can we build a

Logic for Programs ???

Hoare – Logic: A Proof System for Programs

- Now, can we build a

  Logic for Programs ???

  Well, yes !

  There are actually lots of possibilities ...

- We consider the Hoare-Logic (Sir Anthony Hoare ...), technically an inference system PL + E + A + Hoare

Hoare – Logic: A Proof System for Programs

- Basis: 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, cond a boolean expr.

Hoare Logic vs. Symbolic Execution

- HL is also based notion of a *symbolic state.*

$$\text{state}_{\text{sym}} = V \rightarrow \text{Set}(D)$$

As usual, we denote sets by

$$\{\ x \mid E\ \}$$

where E is 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

  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 vs. Symbolic Execution

- However, instead of:

  $$\vdash \{\sigma::state_{sym} \mid Pre(\sigma(X_1), \ldots, \sigma\,(X_n)\}$$
  $$cmd$$
  $$\{\sigma::state_{sym} \mid Post(\sigma(X_1), \ldots, \sigma\,(X_n)\}$$

  where Pre and Post are sets of states.
  we just write:

  $$\vdash \{Pre\} \;\; cmd \; \{Post\}$$

  where Pre and Post are expressions over program variables.

Hoare Logic vs. Symbolic Execution

- Intuitively:

  |– {Pre} cmd {Post}

  means:

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

Hoare – Logic: A Proof System for Programs

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

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

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

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

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

Hoare – Logic: A Proof System for Programs

- The 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 rule for the assignment:

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

Example (1):

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

Hoare – Logic: A Proof System for Programs

- The rule for the assignment

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

Example (2):

|– {true} x:== 2 {x=2}

Hoare – Logic: A Proof System for Programs

- The rule for the conditional:

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

essentially case-split.

Hoare – Logic: A Proof System for Programs

- The rule for the conditional:

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

Example (3):

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

Hoare – Logic: A Proof System for Programs

- The rule for the conditional:

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

Example (3):

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

Hoare – Logic: A Proof System for Programs

- The rule for the sequence:

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

essentially relational composition on state sets.

Hoare – Logic: A Proof System for Programs

The rule for the sequence.

Example (4):

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

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 $B = tm = 1 \wedge sum = 1$.

Hoare – Logic: A Proof System for Programs

- The rule for the while-loop.

$$\frac{\vdash \{P \wedge cond\}\ c\ \{P\}}{\vdash \{P\}\ \mathrm{WHILE}\ cond\ \mathrm{DO}\ c\ \{P \wedge \neg cond\}}$$

Critical: The invention of an Invariant P.

If we have an invariant (a predicate that remains stable during loop taversal), then it must be true after the loop. And if states after the loop exist, the negation of the condition must be true.

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'.

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:

$$\frac{P \to P' \qquad \vdash \{P'\}\ cmd\ \{Q'\} \qquad Q' \to Q}{\vdash \{P\}\ cmd\ \{Q\}}$$

Example (5) (continuation of Example ()):

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

Hoare – Logic: A Proof System for Programs

- A handy derived rule (False):

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

Proof: by induction over *cmd* !

A very handy corollary of this and the consequence is rule (FalseE):

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

Hoare – Logic: A Proof System for Programs

- Another handy corollary of (False):

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

Proof:
by consequence, while-rule,
P and cond-contradiction,
False-rule.

Hoare – Logic: A Proof System for Programs

- Yet another handy corollary of (consequence):

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

Proof:
by consequence and the fact that $P = P'$ infers $P \rightarrow P'$

Note: We will apply this rule implicitly, allowing local massage of pre- and postconditions.

Hoare – Logic: A Proof System for Programs

- Example (6):

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

Hoare – Logic: A Proof System for Programs

- Example (6):

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

Proof:

$$\frac{true \to true \qquad \dfrac{\dfrac{}{\vdash \{true \wedge false\}\text{SKIP}\{false\}}}{\vdash \{true\}\ \text{WHILE}\ true\ \text{DO SKIP}\ \{false\}} \qquad false \to x = 42}{\vdash \{true\}\ \text{WHILE}\ true\ \text{DO SKIP}\ \{x = 42\}}$$

Hoare – Logic: A Proof System for Programs

- Example (6):

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

Note:

Hoare-Logic is a calculus for
partial correctness; on 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:

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

where $I'' = I'[x \mapsto x+1]$ and where we need solutions to:

$$A = true \to I$$

$$B = I \land \neg(x < 2) \to 2 \leq x$$

$$C = I \land x < 2 \to I'[x \mapsto x+1]$$

$$D = I' \to I$$

Hoare – Logic: A Proof System for Programs

- Example (7):
  Proof:

$$A = \ true \rightarrow I$$

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

$$C = I \wedge x < 2 \rightarrow I'[x \mapsto x{+}1]$$

$$D = I' \rightarrow I$$

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

Hoare – Logic: A Proof System for Programs

- Example (7):
  Remarks:
  - This proof rises the idea of particular construction method of Hoare-Proofs, which can be automated:

    - apply the consequence rule only at entry points of (the body of) loops (deterministic!)
    - extract the implications used in these consequence rule
    - try to find solutions for these implications (worst case: ask the user ...)
  - Essence of all: constraint solving of formulas ...

Hoare – Logic: Summary

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

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 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 a wp-calculus)

total correctness = partial correctness + termination …

Hoare – Logic: Summary

Theorem: Correctness of the Hoare-Calculus

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

Theorem: Relative Correctness of the Hoare-Calculus

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

where we define for a given semantic function C:

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

Hoare – Logic: Summary

Formal Proof

- Can be very hard – up to infeasible (no one will probably ever prove correctness of MS Word!)
- Proof Work typically exceeds Programming work by a factor 10!
- Tools and Tool-Chains necessary
- *Makes assumptions on language, method, tool-correctness, too !*

Hoare – Logic: Outlook

- Can we be sure, that the logical systems are consistent ?

  Well, yes, practically.
  (See Hales Article in AMS: "Formal Proof", 2008.
  http://www.ams.org/ams/press/hales-nots-dec08.html)

  - Can we ever be sure, that a specification "means" what we intend ?

    Well, no.
    But when can we ever be entirely sure that we know what we have in mind ?
    But at least, we can gain confidence validating specs, i.e. by animation and test, thus, by experimenting with them ...

Verification : Test or Proof

Test

- Requires Testability of Programs (initialitzable, reproducible behaviour, sufficient control over non-determinism)
- Can be also Work-Intensive !!!
- Requires Test-Tools
- Requires a Formal Specification
- Makes Test-Hypothesis, which can be hard to justify !

Validation : Test or Proof (end)

Test and Proof are Complementary ...

- ... and extreme ends of a continuum : from static analysis to formal proof of "deep system properties"

- In practice, a good "verification plan" will be necessary to get the best results with a (usually limited) budget !!!
  - detect parts which are easy to test
  - detect parts which are easy to prove
  - good start: maintained formal specification
    - ☞ this leaves room for changes in the conception
    - ☞ ... and for different implementation of sub-components