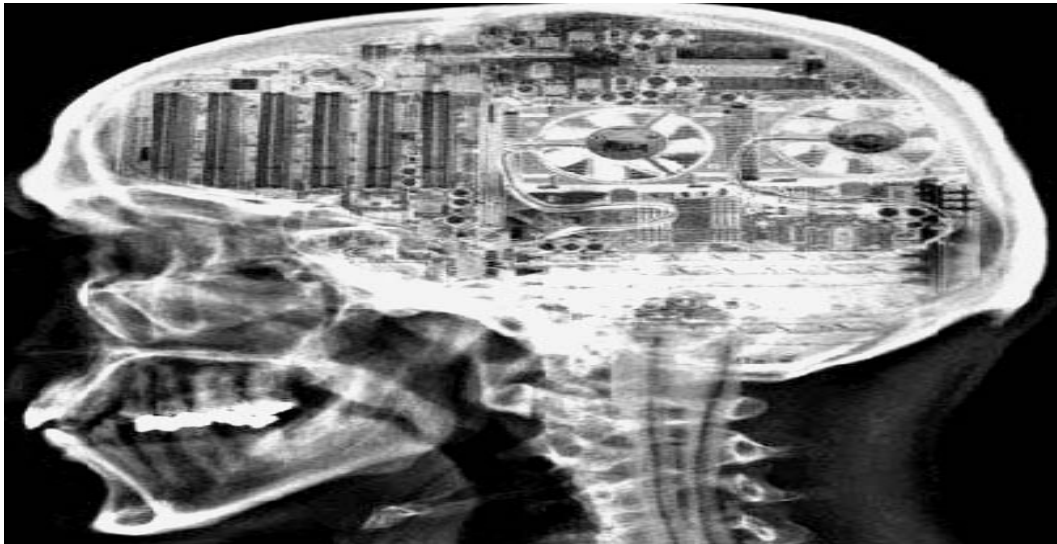


Licence Mention Informatique - L3/S6 - 2009/10
Introduction au Génie Logiciel
Partie3: Verification by Proof (I)



Burkhart Wolff
Département Informatique

Why Proof ?

- ❑ Test Procedures have theoretical limits in certifying Quality Assurance ...
- ❑ Test Procedures have practical limits in certifying Quality Assurance ...
(low coverage of feasible paths, complexity of DNF-computations, ...)
- ❑ Random-Testdata generators are usually **very ineffective**
- ❑ In some problem domains, usual Testing Hypotheses are not justifiable (stuck-at errors in Hardware)
- ❑ In some problem domains, usual Testability Hypothesis are not justifiable (SUT is *not* a function; non-determ. Sys.)

Difficulties with Testing so far

- ❑ So, for:

- non-deterministic operating system level systems
- large distributed code
- high-assurance, safety-critical systems (Common Criteria EAL 7 level)

... Testing techniques can not be applied (at least : alone).

- ❑ Anyway, systematic testing involves specification as well, so, modeling work has to be done in both cases

Who is using formal proofs in industry?

❑ Hardware Suppliers:

- INTEL: Proof of Floating Point Computation compliance to IEEE754
- INTEL: Correctness of Cash-Memory-Coherence Protocols
- AMD: Correctness of Floating-Point-Units against Design-Spec

❑ Software Suppliers:

- GemPlus: Verification of Smart-Card-Applications in Security (First EAL7 level certification ...)
- MicroSoft: Many Drivers running in „Kernel Mode“ were verified
- MicroSoft: Verification of the Hyper-V OS (60000 Lines of Concurrent, Low-Level C Code ...)
- . . .

Roles of Proof

- ❑ Validation possible ?
 - ... to a certain extent. Proofs — if done interactively — may help to deepen both the understanding of code & specification.
 - ... however, this tends to be costly.

- ❑ Verification possible ?
 - Yes — Both for internal properties, for the conformance of design to analysis specs as well as the conformance of design to code.
 - for low-level, concurrent code without alternative ...

Automation of Proof

- ❑ Can Program Verification be automated ?
 - ... to a large extent

(this is what the rest of the course is about)
 - ... still, some very nasty parts of verification, in particular finding loop-invariants or delicate framing conditions, require a lot of thought and human user interaction.

Automation of Proof

- ❑ Can we do Automated Verification for UML/OCL ?
 - ... no. There are tools (like HOL-OCL, see <http://www.brucker.ch/projects/hol-ocl/>) but they require too much knowledge on both proof and meta-theory of OCL for a lecture.
 - We will do most of it by hand ;-(

Requirements of Proof

- ❑ A Formal Specification (OCL or JML, but also Z, VDM, CSP, B, ..., ACDL or VCC)
 - know-how over the application domain
 - informal and formal requirements of the system (design-level, with framing conditions ...)

- ❑ a domain theory; a number of predicates describing foundational data-structures from the problem domain

- ❑ a semantics of the underlying programming language ...

Revision

STATE AND SYMBOLIC STATES ...

SEE SLIDES ON WHITE BOX TESTING ...

Foundation : What is a Formal Proof ?

▣ A Tree, where:

- the nodes are formulas
- the branches correspond to inference rules (so we need a set of these rules, the *inference system*)
- rules should be logically valid

(whatever this means ...)
- the root of the tree is called the *conclusion*, the leaves the *assumptions*

Foundation : Proof Systems

- An Inference System (or *Logical Calculus*) allows to infer formulas from a set of *elementary facts* (axioms) and inferred facts by rules:

$$\frac{A_1 \quad \dots \quad A_n}{A_{n+1}}$$

“from the *assumptions* A_1 to A_n , you can infer the conclusion A_{n+1} .” A rule with $n=0$ is an elementary fact. Variables occurring in the formulas A_n can be arbitrarily substituted.

Foundation : Proof Systems

- An Inference System for the equality operator (or “Equational Logic”) looks like this:

$$\frac{}{x = x} \qquad \frac{x = y}{y = x} \qquad \frac{x = y \quad y = z}{x = z}$$

$$\frac{x = y \quad P(x)}{P(y)}$$

(where the first rule is an elementary fact).

Foundation : Proof Systems

- A series of inference rule applications is usually displayed as *Proof Tree* (or : *Derivation*)

$$\frac{\frac{f(a,b) = a \quad \frac{f(a,b) = a \quad f(f(a,b), b) = c}{f(a,b) = c}}{a = c} \quad \frac{}{g(a) = g(a)}}{g(a) = g(c)}$$

- The non-elementary facts are the *global assumptions* (here $f(a,b) = a$ and $f(f(a,b), b) = c$).

Foundation : Proof Systems

- As a short-cut, we also write for a derivation:

$$\{f(a, b) = a, f(f(a, b), b) = c\} \vdash g(a) = g(c)$$

... or generally speaking: from global assumptions A to a **theorem** (in theory E) ϕ :

$$A \vdash_E \phi$$

This is what theorems are: derivable facts from assumptions in a certain logical system ...

Foundation : A Proof System for Propositional Logic

- Propositional Logic (PL) in so-called natural deduction:

$$\begin{array}{c}
 \frac{A}{A \vee B} \qquad \frac{B}{A \vee B} \qquad \frac{A \vee B \quad \begin{array}{c} [A] \\ \vdots \\ Q \end{array} \quad \begin{array}{c} [B] \\ \vdots \\ Q \end{array}}{Q} \\
 \\
 \frac{A \quad B}{A \wedge B} \qquad \frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B} \qquad \frac{A \wedge B \quad \begin{array}{c} [A, B] \\ \vdots \\ Q \end{array}}{Q}
 \end{array}$$

Foundation : A Proof System for Propositional Logic

- Propositional Logic (PL) in so-called natural deduction:

$$\begin{array}{c}
 \frac{\textit{False}}{A} \\
 \\
 \frac{\begin{array}{c} [A] \\ \vdots \\ \neg A \end{array}}{B} \\
 \\
 \frac{\neg\neg A}{A} \\
 \\
 \frac{A}{\neg\neg A} \\
 \\
 \frac{P \rightarrow Q \quad P}{Q} \\
 \\
 \frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{A \rightarrow B}
 \end{array}$$

Foundation : A Proof System for Propositional Logic

- PL + E + Arithmetics (A) in so-called natural deduction:

$$\overline{1 + x \neq x}$$

$$\overline{(1 + x = 1 + y) \rightarrow x = y}$$

$$\frac{P(0) \quad \forall x. P(x) \rightarrow P(1 + x)}{\forall x. P(x)}$$

$$\overline{(1 + x) + y = 1 + (x + y)}$$

$$\overline{x + y = y + x}$$

$$\overline{x + (y + z) = (x + y) + z}$$

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: 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

- 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 vs. Symbolic Execution

- However, instead of:

$$\begin{array}{l} \vdash \{ \sigma :: \text{state}_{\text{sym}} \mid \text{Pre}(\sigma(X_1), \dots, \sigma(X_n)) \} \\ \text{cmd} \\ \{ \sigma :: \text{state}_{\text{sym}} \mid \text{Post}(\sigma(X_1), \dots, \sigma(X_n)) \} \end{array}$$

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

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

where Pre and Post are expressions over program variables.

Hoare Logic vs. Symbolic Execution

- Intuitively:

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

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:

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

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

$$\frac{}{\vdash \{P \wedge \text{cond}\} c \{P\}}$$

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

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

Hoare – Logic: A Proof System for Programs

- The rule for the empty statement:

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

well, states do not change ...

Therefore, valid states remain valid.

Hoare – Logic: A Proof System for Programs

- The rule for the assignment:

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

Example (1):

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

Hoare – Logic: A Proof System for Programs

- The rule for the assignment

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

Example (2):

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

Hoare – Logic: A Proof System for Programs

- The rule for the conditional:

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

essentially case-split.

Hoare – Logic: A Proof System for Programs

- The rule for the conditional:

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

Example (3):

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

Hoare – Logic: A Proof System for Programs

- The rule for the conditional:

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

Example (3):

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

Hoare – Logic: A Proof System for Programs

- The rule for the sequence:

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

essentially relational composition on state sets.

Hoare – Logic: A Proof System for Programs

The rule for the sequence.

Example (4):

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

Hoare – Logic: A Proof System for Programs

The rule for the sequence.

Example (4):

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

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.

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

Critical: The invention of an Invariant P .

If we have an invariant (a predicate that remains stable during loop traversal), 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:

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

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:

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

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

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

Hoare – Logic: A Proof System for Programs

- A handy derived rule (False):

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

Proof: by induction over *cmd* !

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

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

Hoare – Logic: A Proof System for Programs

- Another handy corollary of (False):

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

Proof:

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

Hoare – Logic: A Proof System for Programs

- Yet another handy corollary of (consequence):

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

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):

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

Hoare – Logic: A Proof System for Programs

- Example (6):

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

Proof:

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

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):

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

Hoare – Logic: A Proof System for Programs

□ Example (7):

Proof:

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

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

$$A = true \rightarrow I$$

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

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

$$D = I' \rightarrow I$$

Hoare – Logic: A Proof System for Programs

- Example (7):
Proof:

$$A = \text{true} \rightarrow I$$

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

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

$$D = I' \rightarrow I$$

- 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 measures 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 ...

Verification : Test or Proof

Test

- Requires Testability of Programs (initializable, 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 !

Hoare – Logic: Summary

Theorem: Correctness of the Hoare-Calculus

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

Theorem: Relative Correctness of the Hoare-Calculus

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

where for a semantic function C we define:

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

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 ...

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