



POLYTECH[®]
PARIS-SUD

Cycle Ingénieur – 2^{ème} année

Département Informatique

Verification and Validation

Part IV : Proof-based Verification

Burkhart Wolff

Département Informatique

Université Paris-Sud / Orsay

Hoare – Logic: A Proof System for Programs

- Our question has been: Is there a

Logic for Programs ???

Hoare – Logic: A Proof System for Programs

- Now, can we build a

Logic for Programs ???

And a first answer:

Well, yes ! Hoare Logic !

Hoare – Logic: A Proof System for Programs

- Now, can we build a

Logic for Programs ???

And a first answer:

Well, yes ! Hoare Logic !

- But can we automate the tedious task ?

Hoare – Logic: A Proof System for Programs

- ❑ How can we automate the tedious task ?
 - can we make the Hoare-calculus more deterministic ?
(sometimes we had the feeling this is possible)

 - can we reduce the task of program-verification to ordinary, standard logic problems ?
(like constraint-solving)

Hoare – Logic: A Proof System for Programs

- Hoare revisited (i):

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

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

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

- ... this part is actually highly deterministic

Hoare – Logic: A Proof System for Programs

- Hoare revisited (ii):

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

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

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

- ... this part needs some work, and some new ideas.

Hoare – Logic: A Proof System for Programs

□ Hoare revisited (iii):

... this part needs some work, and some new ideas.

- the sequence rule is also deterministic, if combining „basic programs“
- the consequence rule is deterministic, if we apply it in the following strategy:
 - only at the beginning of an entire proof
 - before and immediately after a WHILE-rule application
- the WHILE-rule is then deterministic, if we have the invariant I

Hoare – Logic: A Proof System for Programs

- Hoare revisited (ii):
 - ... this part needs some work, and some new ideas.
 - the sequence rule is also deterministic, if combining „basic programs“:

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

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

□ *Hhm,*

do we actually really need pre- and postconditions?

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

if we have the post-condition, we can practically compute a pre-condition.

The wp calculus

- ❑ Core Concept: The predicate transformer wp
 - Dijkstra's „weakest precondition calculus“
 - $wp(\text{SKIP}, P) = P$
 - $wp(x := E, P) = P[x \mapsto E]$
 - $wp(c; d, P) = wp(c, wp(d, P))$
 - $wp(\text{IF } c \text{ THEN } d \text{ ELSE } e, P) =$
 $c \rightarrow wp(d, P) \wedge \neg c \rightarrow wp(e, P)$

The wp calculus

- Core Concept: The predicate transformer wp
 - Dijkstra's „weakest precondition calculus“

For the fragment of the language,
where the paths are fixed, we have
a solution.

Can we extend this to while ?

The wp calculus

- Core Concept: The predicate transformer wp
 - Invariants and Pre-Conditions must still be represented; why not represent this information

inside the program

instead of

inside the proof ?

The wp calculus

□ Basis cmd_A : IMP's cmd + assertions

- the empty command SKIP
- the assignment $x ::= E \quad (x \in V)$
- the sequential compos. $c_1 ; c_2$
- the conditional IF cond THEN c_1 ELSE c_2
- the loop WHILE cond DO c
- the „pre-conditions“ assume(P)
- the „post-conditions“ assert(P)

The wp calculus

- Intuitively:

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

means in Dijkstra's language (well, see details):

```
assume Pre;  
cmd;  
assert Post
```

- For an assume, the „programmer is off the hook“ (it is the environment that must assure this), while an assertion is a guarantee by the programmer ...

The wp calculus

- ❑ Core Concept: The predicate transformer wp
 - Dijkstra's „weakest precondition calculus“ (well, a modern variant thereof ...)
 - $wp(\text{assume } P, Q) = P \rightarrow Q$
 - $wp(\text{assert } P, Q) = Q \wedge P$

The wp calculus

- WHILE-loops annotated with invariants:
 - We introduce the notation:

WHILE c DO $\{I\}$ cmd

for:

```
assert I;  
WHILE  $c$  DO  
  assume  $c$ ;  
   $cmd$ ;  
  assert I
```

The wp calculus

- ❑ Still, that does not solve the problem, wp works only for programs with finite paths.
- ❑ However, we can chop all the paths out of a program, to paths to a loop, the paths inside, etc.

$$\text{wp}(\text{WHILE } c \text{ DO } \{I\} \text{ cmd}, Q) = Q$$

- ❑ ... and introduce a „verification condition generator“

$$\text{vcg}(\text{WHILE } c \text{ DO } \{I\} \text{ body}) P =$$

$$\begin{aligned} & ((I \wedge \neg c) \rightarrow Q) \wedge && \text{-- exit must establish } Q \\ & ((I \wedge c) \rightarrow \text{wp}(\text{cmd}, I)) \wedge && \text{-- } I \text{ must be invariant} \\ & (\text{vcg body } I) && \text{-- internal paths} \end{aligned}$$

$$\text{vcg}(_) P = \text{true}$$

collecting all paths !

The wp calculus

- Technically, Hoare-Logic and vcg and wp-calculus are connected by the following theorem:

Theorem: Correctness of vcg.

If:

$\text{vcg cmd } Q$ and $P \rightarrow \text{wp}(\text{cmd}, Q)$

then

$\vdash \{P\} (\text{strip_assertions cmd}) \{Q\}$

Hoare – Logic: Revisit Semantic Foundation

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 we define for a given semantic function C :

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

The wp calculus

□ Example:

$\vdash \{\text{true}\}$

$\text{tm} ::= 1; \text{sum} ::= 1; i ::= 0$

$\{\text{tm} = 1 \wedge \text{sum} = 1 \wedge i = 0\}$

is (by correctness theorem of vcg/wp)

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

$\text{true} \rightarrow \text{wp}(\text{tm} ::= 1; \text{sum} ::= 1; i ::= 0,$
 $\quad \text{tm} = 1 \wedge \text{sum} = 1 \wedge i = 0)$

$= \text{tm} = 1 \wedge \text{sum} = 1 \wedge i = 0 [i \mapsto 0, \text{sum} \mapsto 1, \text{tm} \mapsto 1]$

$= 1 = 1 \wedge 1 = 1 \wedge 0 = 0 = \text{true}$

The wp calculus

□ Example:

$\vdash \{\text{true}\}$

IF $x \leq 0$ THEN $x := -x$ ELSE SKIP

$\{0 \leq x\}$

is (by correctness theorem of vcg/wp)

$\text{vcg (IF } x \leq 0 \dots)(0 \leq x) \wedge$

$\text{true} \rightarrow \text{wp}(\text{IF } x \leq 0 \text{ THEN } x := -x \text{ ELSE SKIP,}$
 $0 \leq x)$

$= x \leq 0 \rightarrow \text{wp}(x := -x, 0 \leq x) \wedge$

$\neg(x \leq 0) \rightarrow \text{wp}(\text{SKIP}, 0 \leq x)$

$= x \leq 0 \rightarrow 0 \leq -x \wedge \neg(x \leq 0) \rightarrow 0 \leq x = \text{true}$

The wp calculus

□ Example:

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

is (by correctness theorem of vcg/wp)

$$\begin{aligned} & \text{vcg}(\text{WHILE } x < 2 \text{ DO } \{ x \leq 2 \} \dots, 2 \leq x) \wedge \\ & \text{true} \rightarrow \text{wp}(\text{WHILE } x < 2 \text{ DO } \{ x \leq 2 \} \dots, 2 \leq x) \\ = & (x \leq 2 \wedge \neg x < 2) \rightarrow 2 \leq x \wedge \\ & (x \leq 2 \wedge x < 2) \rightarrow \text{wp}(x ::= x + 1, x \leq 2) \wedge \\ & \text{vcg}(x ::= x + 1, x \leq 2) \wedge \\ & \text{true} \rightarrow 2 \leq x \end{aligned}$$

Tools following the vcg-approach

- Microsoft Visual-Studio + Spec# + Boogie + Z3
(for a C# like language)
- Microsoft Visual-Studio + VCC + Boogie + Z3
(for a realistic subset of C / X86)
- gwhy + Why + AltErgo
- Eclipse + Jessy + Why + Z3 / AltErgo
(Vanilla C)
- Isabelle/HOL + Simpl + ...
(Has a Vanilla C frontend)

Tools: gwhy and Squareroot

The screenshot shows the gWhy verification conditions viewer interface. The window title is "gWhy: a verification conditions viewer". The menu bar includes "File", "Configuration", and "Proof".

The left pane displays proof obligations for the "C function sqrt Correctness". The overall status is "Alt-Ergo 0.9" with a green checkmark and "Statistics: 12/12". The obligations are:

Proof obligations	Alt-Ergo 0.9	Statistics
1. loop invariant initially holds	✓	12/12
2. loop invariant initially holds	✓	
3. loop invariant initially holds	✓	
4. loop invariant initially holds	✓	
5. assertion	✓	
6. loop invariant preserved	✓	
7. loop invariant preserved	✓	
8. loop invariant preserved	✓	
9. variant decreases	✓	
10. variant decreases	✓	
11. postcondition	✓	
12. postcondition	✓	

The right pane shows the verification conditions and the C code for the sqrt function. The code is as follows:

```
sqrt_impl_po_1
a: int
H1: 0 <= a

0 * 0 <= a

*****

/*@ axiom square_sum :
  @ \forall int i; i * i + ((2 * i) + 1) == (i + 1) * (i + 1)
  @*/

/*@ requires 0<=a
  @ ensures \result * \result <= a < (\result+1) * (\result+1)
  @*/

int sqrt(int a) {
  int i = 0;
  int tm = 1;
  int sum = 1;
  /*@ invariant
  @ (i * i <= a) && (tm == 2 * i + 1) && (tm > 0)
  @&& (sum == (i+1) * (i+1))
  @ variant (a - sum)
  @*/
  while (sum <= a) {
    i++;
    tm=tm+2;
    /*@ assert tm == 2 * i + 1
    sum=sum+tm;
  };
  return(i);
}
```

The bottom status bar shows "Timeout 10", "Pretty Printer" (unchecked), and "file: Sqrt.c Correctness of C function sqrt".

Dijkstra's – Calculus: Summary

Verification by Formal Proof

- Substantially improved degree of automation !
Both on methodology and by automated theorem provers ...
- Still, you have to provide the invariants, which is the key work !
- Tools and Tool-Chains necessary
(but, meanwhile, there are a few ...)

Dijkstra's – Calculus: Summary

Verification by Formal Proof

- Proof Work typically exceeds conventional programming work by a factor 10!
(However, the factor decreases for really critical code; and thorough testing costs already a factor 2 !)
- Advantage of Testing: Usually no Invariants necessary.
- Advantage of Testing: offers Validation !

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 !