*L3 Mention Informatique*
*Parcours Informatique et MIAGE*

# Génie Logiciel Avancé - Advanced Software Engineering

## Deductive Verification III

Burkhart Wolff
wolff@lri.fr

# 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
- … and transit to a more automatic variant, Dijkstra's wp calculus.

# Hoare - Logic: A Proof System for Programs

- Revision Example (7):

  Proof (bottom up):

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

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

- Revision Example (7):

  Proof (bottom up):

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

Now we can apply the while rule.

# Hoare - Logic: A Proof System for Programs

- Revision 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

- Revision Example (7):

  Proof (bottom up):

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

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

- Revision Example (7):

  Proof (bottom up):

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

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

# Hoare - Logic: A Proof System for Programs

- Revision 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

- Revision 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: A Proof System for Programs

- 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: A Proof System for Programs

- Another Example (8) : The integer square-root

```
int i = 0;
int tm = 1;
int sum = 1;
WHILE sum ≤ a DO
  i  := i+1;
  tm := tm + 2;
  sum:= sum + tm;
```

$\Big\} \equiv$ prelude

$\Big\} \equiv$ body

- Program and Specification in a Hoare Triple

$\vdash \{a \geq 0\}$ prelude; WHILE sum $\leq$ a DO body $\{post\}$

where post $\equiv i^2 \leq a \wedge a < (i+1)^2$

# Hoare - Logic: A Proof System for Programs

- We cut it into 2 parts (sequence rule):
  - first: $\vdash \{a \geq 0\}$ prelude $\{a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1\}$

We know that already ....

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

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

- and:

$\vdash \{a \geq 0 \wedge A\}$ WHILE sum $\leq$ a DO body $\{i^2 \leq a \wedge a < (i+1)^2\}$

# Hoare - Logic: A Proof System for Programs

- so, for the body, we derive bottom-up:

$$\dfrac{I' \longrightarrow I''[i \mapsto i+1] \qquad \overline{\vdash \{I''[i \mapsto i+1]\}\, I := I+1\, \{I''\}} \qquad I'' \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2]}{\vdash \{I'\}\ i := i+1\, \{I[sum \mapsto sum+tm][tm \mapsto tm+2]\}}$$

$$\dfrac{\vdash \{I'\}\ i := i+1\, \{I[sum \mapsto sum+tm][tm \mapsto tm+2]\} \qquad \overline{\vdash \{I[sum \mapsto sum+tm][tm \mapsto tm+2]\}\, tm := tm+2\, \{I[sum \mapsto sum+tm]\}}}{\vdash \{I'\}\ i := i+1;\ tm := tm+2\, \{I[sum \mapsto sum+tm]\}}$$

$$\dfrac{\vdash \{I'\}\ i := i+1;\ tm := tm+2\, \{I[sum \mapsto sum+tm]\} \qquad \overline{\vdash \{I[sum \mapsto sum+tm]\}\, sum := sum+tm\ \{I\}}}{\vdash \{I'\}\ i := i+1;\ tm := tm+2;\ sum := sum+tm\ \{I\}}$$

$$\dfrac{I \wedge sum \leq a \longrightarrow I' \qquad \vdash \{I'\}\ i := i+1;\ tm := tm+2;\ sum := sum+tm\ \{I\} \qquad I \longrightarrow I}{\vdash \{I \wedge sum \leq a\}\ body\ \{I\}}$$

$$\dfrac{a \geq 0 \wedge A \longrightarrow I \qquad \dfrac{\vdash \{I \wedge sum \leq a\}\ body\ \{I\}}{\vdash \{I\}\ WHILE\ sum \leq a\ DO\ body\ \{a < sum \wedge I\}} \qquad a < sum \wedge I \longrightarrow post}{\vdash \{a \geq 0 \wedge A\}\ WHILE\ sum \leq a\ DO\ body\ \{post\}}$$

# Hoare - Logic: A Proof System for Programs

- so, for the body, we derive bottom-up:

$$
\dfrac{\dfrac{\dfrac{\dfrac{I' \longrightarrow I''[i\mapsto i+1] \quad \overline{\vdash \{I''[i\mapsto i+1]\}\, I := I+1\, \{I''\}} \quad I'' \longrightarrow I[sum\mapsto sum+tm][tm\mapsto tm+2]}{\vdash \{I'\}\ i := i+1\, \{I[sum\mapsto sum+tm][tm\mapsto tm+2]\}} \quad \overline{\vdash \{I[sum\mapsto sum+tm][tm\mapsto tm+2]\}\, tm := tm+2\, \{I[sum\mapsto sum+tm]\}}}{\vdash \{I'\}\ i := i+1;\ tm := tm+2\, \{I[sum\mapsto sum+tm]\}} \quad \overline{\vdash \{I[sum\mapsto sum+tm]\}\, sum:=sum+tm\ \{I\}}}{I \wedge sum \leq a \longrightarrow I' \quad \vdash \{I'\}\ i := i+1;\ tm := tm+2;\ sum:=sum+tm\ \{I\} \quad I \longrightarrow I}
$$

$$
\dfrac{a \geq 0 \wedge A \longrightarrow I \quad \dfrac{\vdash \{I \wedge sum \leq a\}\ body\ \{I\}}{\vdash \{I\}\ WHILE\ sum \leq a\ DO\ body\ \{a < sum \wedge I\}} \quad a < sum \wedge I \longrightarrow post}{\vdash \{a \geq 0 \wedge A\}\ WHILE\ sum \leq a\ DO\ body\ \{post\}}
$$

# Hoare - Logic: A Proof System for Programs

- Our proof boils down to the constraints:

$I' \longrightarrow I''[i \mapsto i+1]$

$I'' \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2]$

Solution $I'' \equiv I[sum \mapsto sum+tm][tm \mapsto tm+2]$

$I \wedge sum \leq a \longrightarrow I'$

$a \geq 0 \wedge A \longrightarrow I$

$a < sum \wedge I \longrightarrow post$

# Hoare - Logic: A Proof System for Programs

- Our proof boils down to the constraints:

$I' \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$

Solution $I' \equiv I[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$

$I \wedge sum \leq a \longrightarrow I'$

$a \geq 0 \wedge A \longrightarrow I$

$a < sum \wedge I \longrightarrow post$

# Hoare - Logic: A Proof System for Programs

- Our proof boils down to the constraints:

$I \wedge sum \leq a \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$

"Invariant is preserved in body"

$a \geq 0 \wedge A \longrightarrow I$

"Invariant initially holds at loop entry"

Recall: $\ldots \equiv a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1$

$a < sum \wedge I \longrightarrow post$

"Invariant at loop exit implies post"

# Hoare - Logic: A Proof System for Programs

- Our proof boils further down to finding the invariant I

$I \wedge sum \leq a \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$

$a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1 \longrightarrow I$

$a < sum \wedge I \longrightarrow i^2 \leq a \wedge a < (i+1)^2$

$i \geq 0$

$tm \geq 1$

$sum \geq 1$

$tm = 2*i + 1$

$sum = \sum_{k=0}^{i} (2*k + 1)$

$sum = (i+1)^2$

$a \geq i^2$

$$I \equiv sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1$$

# Hoare - Logic: A Proof System for Programs

- We check our invariant (1)

$$I \equiv \; sum = (i+1)^2 \;\wedge\; a \geq i^2 \;\wedge\; tm = 2*i + 1 \;\wedge\; tm \geq 1$$

$$I \wedge sum \leq a \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$$

$$a \geq 0 \wedge a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1 \longrightarrow I$$

$$a < sum \wedge I \longrightarrow \; i^2 \leq a \wedge a < (i+1)^2$$

# Hoare - Logic: A Proof System for Programs

- We check our invariant (constraint 1)

$$I \equiv sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1$$

$$I \wedge sum \leq a \longrightarrow I[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$$

$$\equiv \quad sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1 \wedge sum \leq a$$
$$\longrightarrow sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1[sum \mapsto sum+tm][tm \mapsto tm+2][i \mapsto i+1]$$

$$\equiv \quad sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1 \wedge sum \leq a$$
$$\longrightarrow sum+tm+2 = ((i+1)+1)^2 \wedge a \geq (i+1)^2 \wedge tm+2 = 2*(i+1) + 1 \wedge tm+2 \geq 1$$

$$\equiv \quad sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1 \wedge sum \leq a$$
$$\longrightarrow (i+1)^2 + 2*(i+1)+1 = ((i+1)+1)^2 \wedge a \geq (i+1)^2 \wedge (2*i + 1)+2 = 2*(i+1) + 1$$

$$\equiv \quad sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1 \wedge sum \leq a$$
$$\longrightarrow a \geq (i+1)^2$$

$$\equiv True$$

Invariant preserved

# Hoare - Logic: A Proof System for Programs

- We check our invariant (constraint 2)

$$I \equiv sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1$$

$$a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1 \longrightarrow I$$

$$\equiv \quad a \geq 0 \wedge a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1$$
$$\longrightarrow sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1$$

$$\equiv \quad a \geq 0 \wedge a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1$$
$$\longrightarrow 1 = (0+1)^2 \wedge a \geq 0^2 \wedge 1 = 2*0 + 1 \wedge 1 \geq 1$$

$$\equiv \quad a \geq 0 \wedge a \geq 0 \wedge i=0 \wedge tm=1 \wedge sum=1$$
$$\longrightarrow a \geq 0 \wedge 1 = 1$$

$$\equiv \text{True}$$

Invariant initially holds

# Hoare - Logic: A Proof System for Programs

- We check our invariant (constraint 3)

$$I \equiv sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1$$

$$a < sum \wedge I \longrightarrow i^2 \leq a \wedge a < (i+1)^2$$

$$\equiv \quad a < sum \wedge sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1 \longrightarrow i^2 \leq a \wedge a < (i+1)^2$$

$$\equiv \quad a < sum \wedge sum = (i+1)^2 \wedge a \geq i^2 \wedge tm = 2*i + 1 \wedge tm \geq 1 \longrightarrow a < sum$$

$$\equiv \quad \text{True}$$

Invariant implies post-condition

# Hoare - Logic: A Proof System for Programs

- We check termination:
  - We provide a function m that decreases for the program state (a, i , tm, sum) for any possible loop traversal (i.e. $sum \leq a \wedge I$ ), i.e.

$$sum \leq a \wedge I \longrightarrow m(a, i, tm, sum) > m(a, i+1, tm+2, sum+tm)$$

  - Iff such a function m (a measure) exists, the loop will terminate.
  - A candidate for m: $m(a, i, tm, sum) \equiv a - i$ which obviously decreases.

# Hoare - Logic: A Proof System for Programs

- Now, can we build a

Mechanised Logic for Programs ???

Well, yes ! Dijkstra's wp-calculus.

# Hoare - Logic: A Proof System for Programs

- How can we automate the tedious task ?
  - can we make the Hoare-calculus more deterministic ?
  - can we reduce the task of program-verification to ordinary, standard logic problems ? (like constraint-solving in test generation ?)

# Hoare - Logic: A Proof System for Programs

- Hoare revisited (i):

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

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

- ... this part is actually highly deterministic

# Hoare - Logic: A Proof System for Programs

- Hoare revisited (ii):

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

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

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

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

# Hoare - Logic: A Proof System for Programs

- Hoare revisited (ii):

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

  - Note: the sequence rule is deterministic for „basic programs":

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

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

- Hhm, do we actually really need pre- and postconditions?

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

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

For assignment sequences, if we have the post-condition, we can compute a pre-condition from it by proceeding from right to left

# The wp calculus

- Core Concept: The predicate transformer *wp*
- It captures our "strategy" to construct Hoare Proofs
- It is a recursive function going over elementary cmd's
- It calculates from the post-condition P the „weakest liberal precondition"

$$wp(SKIP, P) \equiv P$$

$$wp(x{:==}E, P) \equiv P[x \mapsto E]$$

$$wp(c;d\ , P) \equiv wp(c, wp(d, P))$$

$$wp(IF\ c\ THEN\ d\ ELSE\ e, P) \equiv$$
$$c \rightarrow wp(d, P) \land \neg c \rightarrow wp(e, P)$$

# The wp calculus

- Core Concept: The predicate transformer *wp*

  - Example for a basic program:

    wp(IF $a \geq 0$ THEN tm:=1; sum:=1; i:=0 ELSE SKIP, $tm=1 \wedge sum=1 \wedge i=0$)

$\equiv\ a\geq 0 \rightarrow wp(tm{:}{=}{=}1;\ sum{:}{=}{=}1;\ i{:}{=}{=}0,\ tm=1 \wedge sum=1 \wedge i=0 \wedge$

$\neg(a\geq 0) \rightarrow wp(SKIP,\ tm=1 \wedge sum=1 \wedge I=0$

$\equiv\ a\geq 0 \rightarrow (tm=1 \wedge sum=1 \wedge i=0)[tm \mapsto 1][sum \mapsto 1][i \mapsto 0] \wedge$

$\neg a\geq 0 \rightarrow (tm=1 \wedge sum=1 \wedge i=0)$

$\equiv a\geq 0 \rightarrow True \wedge \neg a\geq 0 \rightarrow (tm=1 \wedge sum=1 \wedge i=0)$

$\equiv a<0 \rightarrow (tm=1 \wedge sum=1 \wedge i=0)$

# The wp calculus

- Core Concept: The predicate transformer *wp*

  - Note:

$$a<0 \rightarrow (tm=1 \wedge sum=1 \wedge I=0)$$

is the weakest liberal precondition. If „a > 5“ the “usual” post-condition

$$tm=1 \wedge sum=1 \wedge i=0$$

just remains as a left-over ...

# The wp calculus

- Core Concept: The predicate transformer *wp*

  So, for the "basic" fragment of the language, we have a solution.

  But can we extend this to while ?

  Solution: We annotate cmd's with the invariants I

# The wp calculus

- Basis $cmd_A$: IMP's cmd

  - the empty command SKIP
  - the assignment x:== E ($x \in V$)
  - the sequential composition $c_1 ; c_2$
  - the conditional IF cond THEN $c_1$ ELSE $c_2$
  - the annotated loop WHILE cond DO {I} c

  So, the only difference between the classic cmd and annotated $cmd_A$ and cmd is the invariant-annotation in the while-construct.

# The wp calculus

- Then we can complete the definition for wp by:

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

- ... and introduce a function vcg „verification condition generator“

  vcg(WHILE c DO {I} body, P) =

  $((I \wedge \neg c) \rightarrow P) \wedge$ -- exit must establish P

  $((I \wedge c) \rightarrow wp(body, I)) \wedge$ -- I must be preserved in body

  vcg(body, I) -- treating internal WHILE's

  $vcg(c; d, P) = vcg(c, wp(d,P)) \wedge vcg(d,P)$

  $vcg(IF\ b\ THEN\ c\ ELSE\ d, P) = vcg(c, P) \wedge vcg(d, P)$

  vcg(_ ,P) = true catchall other options !

# The wp calculus

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

Theorem: Correctness of vcg and wp.

Assume the constraints generated by vcg **and** wp **hold**:

$$\mathrm{vcg}(\mathrm{cmd}, Q) \quad \wedge \quad P \rightarrow \mathrm{wp}(\mathrm{cmd}, Q)$$

Then there exists a Hoare-Proof for:

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

Proof: By induction over the program structure cmd.

# The wp calculus

... in other words:

Instead of constructing a formal Hoare proof,
we can just run vcg and wp and prove
the resulting formula.

# The wp calculus

- Example:

$$\vdash \{True\}\ tm:=1;sum:=1;\ i:=0\ \{tm=1\ \wedge\ sum=1\ \wedge\ i=0\}$$

reduces to (by correctness theorem of vcg/wp)

$vcg\ (tm:=1;sum:=1;\ i:=0,\quad tm=1 \wedge sum=1 \wedge i=0) \wedge$

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

$\equiv tm=1 \wedge sum=1 \wedge i=0\ [I \mapsto 0, sum \mapsto 1, tm \mapsto 1]$

$\equiv 1=1 \wedge 1=1 \wedge 0=0 \equiv True$

# The wp calculus

- Example:

  $\vdash$ {True} IF x ≦ 0 THEN x:== -x ELSE SKIP {0 ≦ x}

- … reduces to (by correctness theorem of vcg/wp)

  vcg (IF x ≦ 0 … , 0 ≦ x) ∧

  true → wp(IF x ≦ 0 THEN x:== -x ELSE SKIP, 0 ≦ x)

  ≡ x ≦ 0 → wp(x:== -x, 0 ≦ x) ∧ ¬(x ≦ 0) → wp(SKIP, 0 ≦ x)

  ≡ x ≦ 0 → 0 ≦ -x ∧ ¬(x ≦ 0) → 0 ≦ x ≡ True

# The wp calculus

- Example:

  $\vdash$ {True} WHILE x<2 DO {x ≦ 2} x:== x+1  {2 ≦ x}

- is (by correctness theorem of vcg/wp)

  vcg(WHILE x<2 DO{x ≦ 2}..., 2 ≦ x) ∧

  true → wp(WHILE x<2 DO{x ≦ 2}..., 2 ≦ x)

  ≡ (x ≦ 2 ∧ ¬x<2) → 2 ≦ x ∧

  (x ≦ 2 ∧ x<2)) → wp(x:== x+1, x ≦ 2) ∧

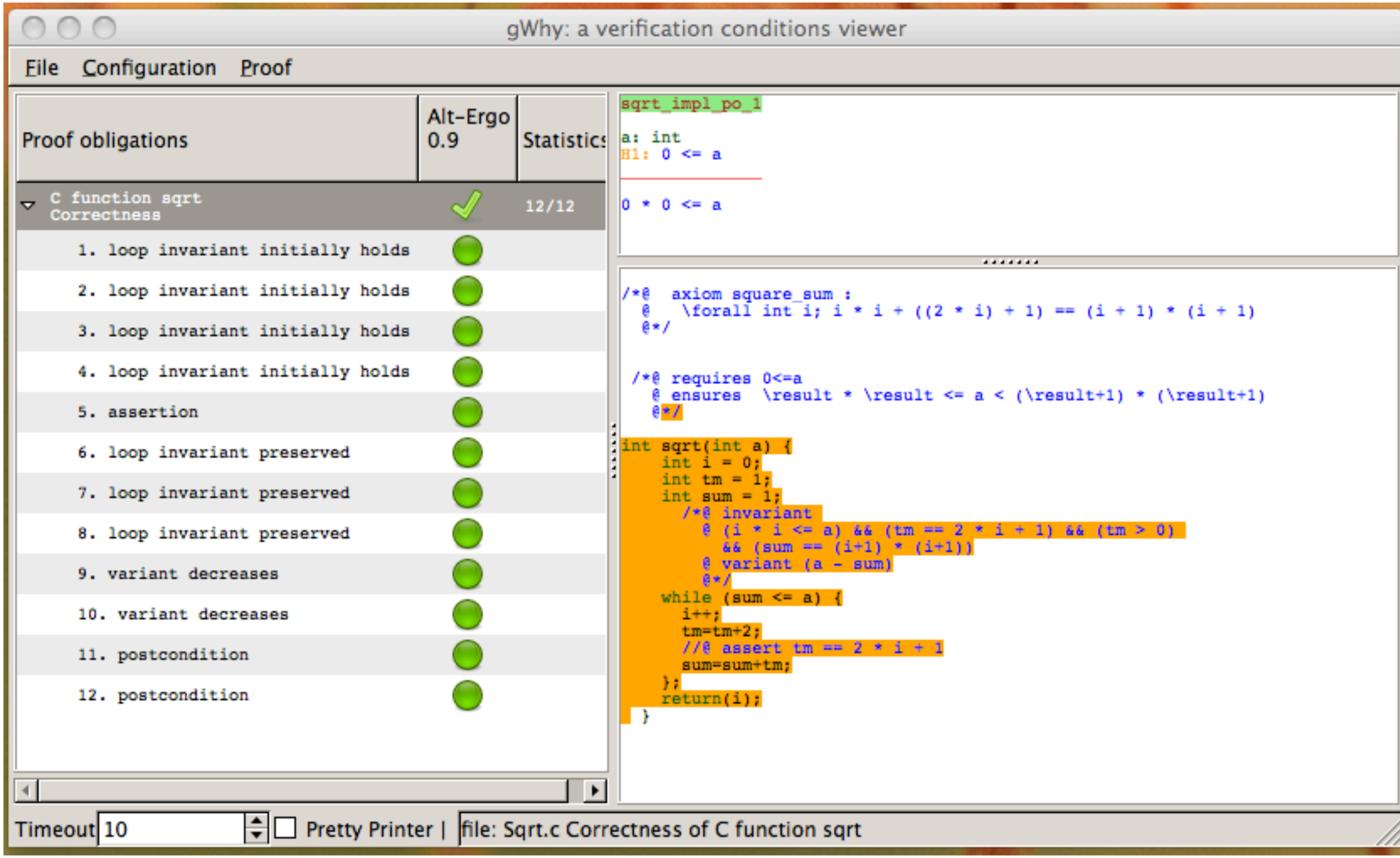
  vcg(x:== x+1, x ≦ 2)

  ≡ True

# 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
- Frama-C + Why + Z3 / AltErgo (Vanilla C frontend)
- Isabelle/HOL + AutoCorres (Vanilla C frontend)

# Tools: gwhy and Squareroot



# Dijkstra's - Calculus: Summary

Verification by Formal Proof

- Substantially improved degree of automation !
  Both by methodology and by automated
  theorem provers ...

- Still, you have to provide the invariants,
  which is the key work ! A particular nasty part
  are framing conditions

- Tools and Tool-Chains necessary
  (but, meanwhile, there are quite a few ...)