Verification & Testing
Hoare Logic

Benedikt Maderbacher
IAIK

Today

- Undecidability
- Manual proofs with Hoare Logic
- Mechanizing Hoare Logic proofs

Motivation

Proving correctness of programs is undecidable

- You can do it only by hand
- Model checking does not (always) work: infinite state space

Hoare logic: notation plus set of rules that allows you to prove programs correct by hand.

- We use very simple version: no function calls, no mallocs, etc

We will use Hoare logic later to compute *abstractions*

The Halting Problem

Does this program halt?

```
int main(){
  BigInt i;
  i << cin; // cin > 0
  while(i != 1){
     if(i is even)
        i = i/2;
      else
        i = 3*i + 1;
  }
}
```

Halting Problem

Halting problem is **undecidable:**

There is no program H(G) that decides, given a program G, whether it halts

- This holds for programs without input, for programs with a fixed input, for the question whether the programs holds for all inputs, etc.

Proof sketch:

- Suppose there is an algorithm H with as input a program P that outputs true iff P halts (on all inputs)
- Take this program: `weird(){ if(H(weird)) while(1);}`
- Is H(weird) true or false?
- There is no correct implementation for H!

Reduction

Problem A *reduces to* problem B if you can use an algorithm for B to solve A

- If B is decidable, so is A
- If A is not decidable, neither is B

More undecidable problems:

- Can G reach location l?
- Can G reach location l with d=0?
- In G, can d ever be 0?

The halting problem *reduces* to these problems.

- For instance, R(G,l) = “can G reach location l” can be used to solve the halting problem
- H(G) = R(G,l) where l is the last line in the program

Ways Out

- Don't prove correctness
- Incomplete Verification
  - Closing the program by providing inputs (test, JPF)
  - Abstraction and refinement (SLAM, BLAST)
  - Verify only *some* programs
- Manual proof using Hoare Logic

Hoare Logic

A **Hoare triple:**

$$\{P\}\ S\ \{Q\},$$

P is the precondition
Q is the postcondition
S is a program

Meaning: if P holds before execution and S finishes, then Q holds afterwards.
Note: we prove **partial correctness**. If S runs forever, {P}S{Q} holds.

Example:

$x \geq 0$ y:= sqrt(x) $\{y * y = x\}$

x := x + 1 {x = 2}

{x > 9} x := x + 1

x := x + 1 {x > 10}

In the following we will assume that variables are integer.

Hoare Logic

Hoare triple **{P} S {Q},**

P: **precondition** S: program Q: **postcondition**

Meaning: if P holds before execution and S finishes, then Q holds afterwards.
Note: we prove **partial correctness.** If S runs forever, {P}S{Q} holds.

Examples:

1. $\{x \geq 0\}$ y:= sqrt(x) $\{y * y = x\}$
2. $\{\mathrm{TRUE}\}$ while(1) do skip od $\{\mathrm{x} + \mathrm{y} = 42\}$
3. {z = 1} z := y + 1 {y = 2}
4. {z > 9} z := y + 1 {y > 10}
5. {z > 100} z := y + 1 {y > 10}

Example 1 and 2 give the **weakest** precondition. We normally prefer that (it gives all circumstances under which the program is correct)

In the following we will assume that variables are integer.

Hoare Logic: Rules

Axioms to find the weakest precondition

- Assignment: x := e
- Consecution: S1; S2
- if-statement: if b then S1 else S2
- Loops: while b do S od

- Plus
  - extra “glue” rules to make things work
  - Function calls, mallocs, pointers, etc

Axiom of Assignment

Example:

$x := y \ \{x = 4\}$

$z := y + 1 \ \{y = 4\}$

$y := 2 * x \ \{x = 8\}$

$y := 2 * x \ \{x < 8\}$

This rule gives the *weakest precondition,* i.e., $\{P[x \rightarrow e]\}$ holds before S **if and only if** P holds afterwards

Axiom of Assignment

$$\frac{}{\{P[x \rightarrow e]\}\ x:=e\ \{P\}}$$

P[x → e] means that x is replaced by e in P

Example:

{y = 4} x := y {x = 4}
{x+1 = 4} x := x + 1 {x = 4}
{x = 4} x := 2 * x {x = 8}
{x < 4} x := 2 * x {x < 8}

This rule gives the *weakest precondition,* i.e., {P[x→e]} holds before S **if and only if** P holds afterwards

Axiom of Skip

$$\frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

(skip is an abbreviation for x:=x)

Axiom of Assert

$$\frac{}{\{P \wedge c\}\ \textbf{assert}\ c\ \{P\}}$$

An assertion holds iff the condition c holds whenever the assert is reached.

Sequencing Rule (Consecution)

Example:

(1) {x = 3} x := x + 1 {x = 4} (ass.)

(2) {x = 4} x := x * 2 {x = 8} (ass.)

(3) x := x + 1; y := x * 2

The horizontal line means: if everything above the line is true, then so is everything below the line.

Sequencing Rule (Consecution)

$$\frac{\{P\}\ S1\ \{Q\} \qquad \{Q\}\ S2\ \{R\}}{\{P\}\ S1;\ S2\ \{R\}}$$

Example:

(1) {x+1 = 4} x := x + 1 {x = 4} (ass.)

(2) {x = 4} x := x * 2 {x = 8} (ass.)

(3) {x = 3} x := x + 1; x := x * 2 {x = 8} (consecution, 1,2)

The horizontal line means: if everything above the line is true, then so is everything below the line.

Conditional Rule

if(x ≥ 0) then

x := x

else

x := -x

fi

{x ≥ 0}

Conditional Rule

$$\frac{S1\ \mathbf{\{Q\}} \qquad\qquad S2\ \mathbf{\{Q\}}}{\mathbf{\{P\}\ if}\ c\ \mathbf{then}\ S1\ \mathbf{else}\ S2\ \mathbf{fi\ \{Q\}}}$$

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\} \qquad \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{if } c \text{ then } S1 \text{ else } S2 \text{ fi}\ \{Q\}}$$

Example:

(1) $\{x \geq 0\}$ skip $\{x \geq 0\}$ (ass.)

(2) $\{x < 0\}$ x = -x $\{x \geq 0\}$ (ass.)

(3) $\{true\}$ if($x \geq 0$) then skip else x = -x fi $\{x \geq 0\}$ (condi. 1,2)

Conditional Rule (Alternative)

$$\frac{\{P1\}\ S1\ \{Q\} \qquad \{P2\}\ S2\ \{Q\}}{\{c \wedge P1 \vee \neg c \wedge P2\}\ \text{if } c \text{ then } S1 \text{ else } S2 \text{ fi}\ \{Q\}}$$

Example:

$\{x \geq 0\}$ skip $\{x \geq 0\}$

$\{x < 0\}$ x = -x $\{x \geq 0\}$

$\{x \geq 0 \vee x < 0\}$ if(x ≥ 0) then skip else x = -x fi $\{x \geq 0\}$

While Rule

Example

$\{x > 0\}$ x = x -1 $\{x \geq 0\}$

$\{x \geq 0\}$

while(x > 0) do

x = x - 1

od

$\{x = 0\}$

While Rule

$$\frac{\{I \wedge c\}\ S\ \{I\}}{\{I\}\ \text{while } c \text{ do } S \text{ od } \{I \wedge \neg c\}}$$

Example

(1) {x > 0} x = x -1 $\{x \geq 0\}$ (assignment)

(2) $\{x \geq 0\}$ while(x > 0) do x = x - 1 od {x = 0} (while, 1)

Notes:

I: $x \geq 0.$

c: x > 0

$x - 1 \geq 0$ = {x >0}

{I ∧ c} = $\{x \geq 0 \wedge x > 0\}$ = {x > 0}

{I ∧ ¬c} = $\{x \geq 0 \wedge x \leq 0\}$ = {x = 0}

This is the hardest rule: how do you find I?

Consequence Rule

the precondition

Example:

$\{true\}$ if($x \geq 0$) then skip else x = -x fi $\{x \geq 0\}$

$\{\quad\quad\}$ if($x \geq 0$) then skip else x = -x fi $\{x \geq 0\}$

Consequence Rule

the postcondition

Example:

$\{true\}$ if(x ≥ 0) then skip else x = -x fi $\{x \geq 0\}$

$\{true\}$ if(x ≥ 0) then skip else x = -x fi $\{ \qquad \}$

Consequence Rule

Strengthening the precondition

$$\frac{\{P\}\ S\ \{Q\} \qquad P' \rightarrow P}{\{P'\}\ S\ \{Q\}}$$

Weakening the postcondition

$$\frac{\{P\}\ S\ \{Q\} \qquad Q \rightarrow Q'}{\{P\}\ S\ \{Q'\}}$$

Mechanizing Hoare Logic

- Handwriting Hoare Logic proofs can be tedious and error prone.
- Many -- but not all – rules can be automated.
- Use an SMT solver to check proofs.
- Add annotations where necessary.

Annotations

$$\frac{\{I \land c\}\ S\ \{I\}}{\{I\}\ \text{while } c \text{ do } S \text{ od}\ \{I \land \neg c\}}$$

Finding I is hard, so add it as an annotation.

while c do {I} S od

Weakest Preconditions

Given an annotated program and a post condition compute its weakest precondition.

pre(x := e, P) =

pre(skip, P) =

pre(assert c, P) =

pre(S1; S2, P) =

pre(if c then S1 else S2, P) =

pre(while c do {I} S od, P) =

Weakest Preconditions

Given an annotated program and a post condition compute its weakest precondition.

pre(x := e, P) = P[$x \rightarrow e$]

pre(skip, P) = P

pre(assert c, P) = $P \wedge c$

pre(S1; S2, P) = pre(S1, pre(S2, P))

pre(if c then S1 else S2, P) =

$(c \wedge pre(S1, P)) \vee (\neg c \wedge pre(S1, P))$

pre(while c do {I} S od, P) = I

Verification Conditions

Additional conditions that need to be satisfied for the pre relation to hold.

vc(x := e, P) =

vc(skip, P) =

vc(assert c, P) =

vc(S1; S2, P) =

vc(if c then S1 else S2, P) =

vc(while c do {I} S od, P) =

Verification Conditions

Additional conditions that need to be satisfied for the pre relation to hold.

vc(x := e, P) = {}

vc(skip, P) = {}

vc(assert c, P) = {}

vc(S1; S2, P) = vc(S1, pre(S2, P)) ∪ vc(S2, P)

vc(if c then S1 else S2, P) = vc(S1,P) ∪ vc(S2, P)

vc(while c do {I} S od, P) =

$$\{(I \wedge \neg c) \Rightarrow P, (I \wedge c) \Rightarrow pre(S, I)\} \cup vc(S, P)$$

Mechanizing Hoare Logic

To check if the Hoare triple {Pre} S {Post}

is correct prove:

- Pre ⇒ pre(S, Post)
- ∧ vc(S, Post)

Proof Example I

```
{true}
if(a > b)

  t := a

  a := b

  b := t

else


  skip

fi
{b≥a}
```

Proof Example I

```
{true}
if(a > b)
   {a > b}
  {a ≥ b}
   t := a
  {t ≥ b}
   a := b
  {t ≥ a}
   b := t
  {b ≥ a}
else
  {b ≥ a}
   skip
  {b ≥ a}
fi
{b≥a}
```

Proof Example I

```
{true}
1 if(a > b)
   {a > b}
  {a ≥ b}
2  t := a
  {t ≥ b}
3  a := b
  {t ≥ a}
4  b := t
  {b ≥ a}
5 else
  {b ≥ a}
6  skip
  {b ≥ a}
7 fi
{b≥a}
```

(1)	$\{b \geq a\}$	skip	$\{b \geq a\}$	(skip)
(3)	$\{b \geq a\}$	b:=t	$\{t \geq a\}$	(ass)
(4)	$\{t \geq b\}$	a:=b	$\{t \geq b\}$	(ass)
(5)	$\{t \geq b\}$	3-4	$\{b \geq a\}$	(consec 3,4)
(6)	$\{a \geq b\}$	t:=a	$\{t \geq b\}$	(ass)
(7)	$\{a \geq b\}$	2-4	$\{b \geq a\}$	(consec 5,6)
(8)	$\{a > b\}$	2-4	$\{b \geq a\}$	(str. pre. 7)
(9)	$\{true\}$	1-7	$\{b \geq a\}$	(if 8,1)

Proof Example II

```
y = 0

x0 = x

while(x != 0) do

  x := x - 1

  y := y + 1

od
```

Hoare Logic, Part 2

Things We Cannot Prove

Suppose `correct(P)` returns true iff `P` never throws assertion violation

```
void strange(){
  assert( !correct(strange) );
}
```

Things We Cannot Prove

```
void f(BigInteger a, b, c, n){
   if(n <= 3) return;
   assert( pow(a, n) + pow(b,n) != pow(c,n) );
}
```

Things We Cannot Prove

Goldbach's conjecture is one of the oldest and best-known unsolved problems in number theory and all of mathematics. It states:

Every even integer greater than 2 is the sum of two primes.

[wikipedia]

Proof Example III

```
{x ≥ 0}
r := x; q := 0;

while(r ≥ y) do


  r := r - y;

  q := q + 1;

od
```

Proof Example III

```
{x ≥ 0}
r := x; q := 0;
{x = (y·q + r) ∧ 0 ≤ r}
while(r ≥ y) do
   {x = (y·q + r) ∧ r ≥ y}
   {x = (y·(q+1) + r - y) ∧ 0 ≤ r - y}
   r := r - y;
   {x = (y·(q+1) + r) ∧ 0 ≤ r}
   q := q + 1;
   {x = (y·q + r) ∧ 0 ≤ r}
od
{x = (y·q + r) ∧ 0 ≤ r ∧ r ≤ y}
```

Proof Example III

```
{x ≥ 0}
1 r := x; q := 0;
{x = (y·q + r) ∧ 0 ≤ r}
2 while(r ≥ y) do
   {x = yq+r ∧ r ≥ y}
   {x = (y·(q+1)+r-y) ∧ 0≤r-y}
3   r := r - y;
   {x = (y·(q+1)+r) ∧ 0≤r}
4   q := q + 1;
   {x = (y·q+r) ∧ 0≤r}
5 od
{x = (y·q+r) ∧ 0≤r ∧ r≤y}
```

(1) {x=(y(q+1)+r) ∧ 0≤r} q:=q+1 {x=(yq+r) ∧ 0≤r} (ass)

(2) {x=(y(q+1)+r-y) ∧ 0≤r-y} r := r - y {x = (y(q+1)+r) ∧ 0≤r} (ass)

(3) {x=yq+r ∧ 0≤r-y} 3-4 {x=(yq+r) ∧ 0≤r} (cons 1,2)

(4) {x=yq+r ∧ 0≤r} 2-4 {x=y·q+r ∧ 0≤r ∧ r≤y} (while, 3)

(5) {x=r ∧ 0≤r} q := 0; {x=yq+r ∧ 0≤r} (ass)

(6) {x≥0} r:=x {x=r ∧ 0≤r}

(7) {x≥0} r:=x;q := 0; {x=yq+r ∧ 0≤r} (cons 5,6)

(8) {x≥0} r:=x;q := 0; {x=y·q+r ∧ 0≤r ∧ r≤y} (cons 7,4)

More Examples

```
x = a;
y = 0;
while(x != 0) {

  x = x - 1;
  y = y + 2;
}
assert(y == 2*a);
```

```
{0 == 2*(a - a)} ↔ {true}
x = a;
{0 == 2*(a - x)}
y = 0;
{y == 2*(a - x)}
while(x != 0) {
  {y == 2*(a - x) ∧ x != 0}
  {y+2 == 2*(a - (x - 1))} ↔ {y+2 == 2*(a - x)+2}
  x = x - 1;
  {y + 2 == 2*(a - x)}
  y = y + 2;
  {y == 2*(a - x)}
}
{y == 2*a ∧ x == 0} ↔ {y == 2*(a - x) ∧ x == 0}
{y == 2*a}
```

```
Input:
a … array of
    integers
n … length of a
```

```
s = 0;

i = 0;

while(i != n) {

  s = s + a[i];

  i = i + 1;

}

assert(s == Σ_{j=0}^{n-1} a[j]);
```

$$\text{assert}(s == \sum_{j=0}^{n-1} a[j]);$$

```
{0 == 0} ↔ {true}
s = 0;
{s == Σ_{j=0}^{-1} a[j]} ↔ {s == 0}
i = 0;
{s == Σ_{j=0}^{i-1} a[j]}
while(i != n) {
   {s == Σ_{j=0}^{i-1} a[j] ∧ i != n}
   {s + a[i] == Σ_{j=0}^{i} a[j]} ↔ {s == Σ_{j=0}^{i-1} a[j]}
   s = s + a[i];
   {s == Σ_{j=0}^{i} a[j]}
   i = i + 1;
   {s == Σ_{j=0}^{i-1} a[j]}
}
{s == Σ_{j=0}^{n-1} a[j] ∧ i == n} ↔ {s == Σ_{j=0}^{i-1} a[j] ∧ i == n}
{s == Σ_{j=0}^{n-1} a[j]}
```

```
r = false;
i = 0;
while(i != n) {

  if(a[i] == x) {
    r = true;
  }
  i = i + 1;
}
assert(r == (⋁_{j=0}^{n-1} a[j] == x));
```

```
r = false;
i = 0;
while(i != n) {
  if(a[i] == x) {
    r = true;
  }
  i = i + 1;
}
```

Input:

a … array

n … length of a

x … value to look for in a

Hint:

$(\bigvee_{j=0}^{-1} \Phi)$ == false

assert(r == $(\bigvee_{j=0}^{n-1}$ a[j] == x));

{false == false} ↔ {true}

r = false;

{r == $(\bigvee_{j=0}^{-1}$ a[j] == x)} ↔ {r == false}

i = 0;

{r == $(\bigvee_{j=0}^{i-1}$ a[j] == x)}

while(i != n) {

  {(r == $(\bigvee_{j=0}^{i-1}$ a[j] == x)) ∧ i != n}

  {r == $(\bigvee_{j=0}^{i-1}$ a[j] == x)}

  if(a[i] == x) {

    {(r == $(\bigvee_{j=0}^{i-1}$ a[j] == x)) ∧ a[i] == x}

    {(true == $(\bigvee_{j=0}^{i}$ a[j] == x)) ∧ a[i] == x} ↔ {true ∧ a[i] == x} ↔ {a[i] == x}

    r = true;

    {r == $(\bigvee_{j=0}^{i}$ a[j] == x)}

  } else {

    {(r == $(\bigvee_{j=0}^{i}$ a[j] == x)) ∧ a[i] != x} ↔ {(r == $(\bigvee_{j=0}^{i-1}$ a[j] == x)) ∧ a[i] != x}

  }

  {r == $(\bigvee_{j=0}^{i}$ a[j] == x)}

  i = i + 1;

  {r == $(\bigvee_{j=0}^{i-1}$ a[j] == x)}

}

{r == $(\bigvee_{j=0}^{n-1}$ a[j] == x) ∧ i == n} ↔ {r == $(\bigvee_{j=0}^{i-1}$ a[j] == x) ∧ i == n}

{r == $(\bigvee_{j=0}^{n-1}$ a[j] == x)}

Hint:

$(\bigvee_{j=0}^{-1} \Phi)$ == false