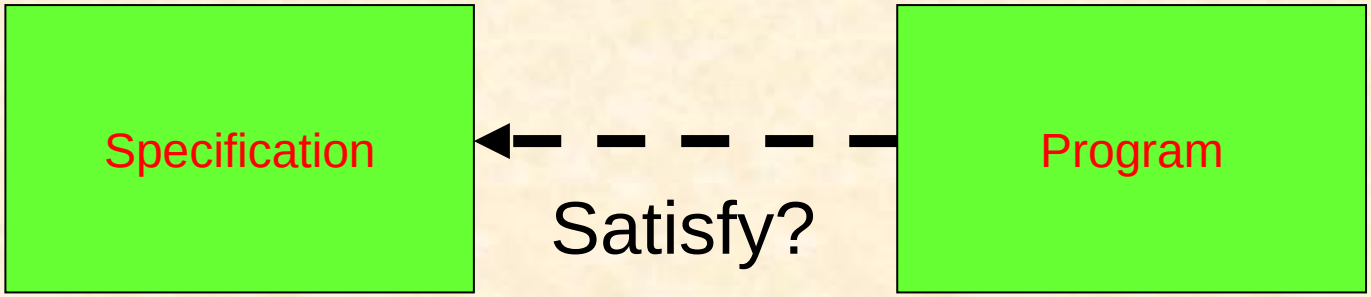
III. Hoare Logic for Formal Verification of Program Correctness



Formal verification or proof

- How to ensure that the program is correct with respect to the specification?
- What does the correctness of the program mean?

Partial and total correctness

(1) Partial correctness

Consider the Hoare triple:

{P} S {R}

It shows the partial correctness of program S with respect to P and R. **This means that If the precondition P is true before initiation of a program S, then the postcondition R will be true on its completion, we say that S is correct with respect to P and R.**

Note that the termination of S is assumed.

(2) Total correctness

Program S is totally correct with respect to predicates P and R iff S is partially correct and S terminates; that is,

Total correctness of S =

{P} S {R} +

Termination of S

Hoare logic is designed for the proof of partial correctness of a program. It is established on the basis of first order logic, but includes additional axioms for reasoning about programs.

The key of proving the termination of program S requires finding a loop variant, a mathematical function indicating that each repetition of a loop statement (e.g., while-loop) changes the loop-variable towards making the loop condition evaluate to *false*.

Program execution

One of the most important properties of a program is whether it carries out its intended function.

This property is known as

functional property of program.

Two ways to express the functional property of a program:

(1) List every initial state before the execution of the program and the corresponding final state after the execution terminates.

Examples:

{(x, 5), (y, 1)} initial state 1

```
int tem = 0;
tem := x;
x := y;
y := tem;
```

Swap program

{(x, 1), (y, 5)} final state 1

{(x, 9), (y, 3)} initial state 2

```
int tem = 0;
tem := x;
x := y;
y := tem;
```

Swap program

{(x, 3), (y, 9)} final state 2

{(x, 20), (y, 10)} initial state 3

```
int tem = 0;
tem := x;
x := y;
y := tem;
```

Swap program

{(x, 10), (y, 20)} final state 3

…

many others

(2) Use pre- and post-assertions (conditions) – Hoare triple:

$$\{P\}\ S\ \{R\}$$

If the assertion P is true before initiation of a program S, then the assertion R will be true on its completion.

Questions:

(1) Who is supposed to write P and R?

(2) What axioms and inference rules do we need for a formal proof of the validity of a specific Hoare triple?

Axioms for Program Proving

To prove the correctness of program S, we first need to establish necessary axioms to define the semantics of each program construct.

Program constructs:

(1) Assignment x := f
(2) Sequence S1; S2
(3) Alternation if B then S1
if B then S1 else S2
(4) Iteration while B do S
(5) Block begin…end

Concepts and notations used in expressing axioms and rules:

(1) $Q[e/x]$ **where Q is an expression or formula, x is a variable, and e is an expression. Explanation: the result of replacing all free occurrences of x in Q by e.**

Example:

$$(x + y * z > y * u)[t+1/y] =$$

$$x + (t + 1) * z > (t + 1) * u$$

(2)

$$\frac{A,\ B}{C}$$

where A, B, and C are predicate formulae. Explanation: a rule of inference which states that if A and B have been proved, then C can be deduced.

This expression is equivalent to:

A and B => C

(3)

$$\frac{A,\ B \vdash C}{D}$$

where A, B, C, and D are predicate formulae. Explanation: a rule of inference which states that if A has been proved and C is deduced from B, then D can be deduced.

This expression is equivalent to:

A and (B ⊢ C) => D

Axiom 1: Axiom of assignment

$$\frac{}{\{P[f/x]\}\ x := f\ \{P\}}\ \textbf{Ass-D0}$$

where

x is a variable identifier

f is an expression of a programming language without side effects, but possibly containing x.

P[f/x] is a predicate resulting from P by substituting f for all occurrences of x in P.

Examples:

(1)

{(x + y) * 5 > z}	{P}
x := x + y	S
{x * 5 > z}	{Q}

(2)

{add(x, z) + y * fact(z) > z}

y := add(x, z) + y * fact(z)

{y > z}

Class exercise

Derive the pre-assertion for each of the following two assignment statements based on their post-assertion:

(1)

x: = x + y;

{x > y}

(2)

y := x + y * (x + 5)

{x = 2 and y * x = 10}

Two rules of consequence

(1)

$$\frac{\{P\}\ S\ \{R\},\ R \Rightarrow Q}{\{P\}\ S\ \{Q\},}$$ **Con-D1**

If the execution of program S ensures the truth of the assertion R under the pre-assertion P, then it also ensures the truth of every assertion logically implied by R.

(2)

$$\frac{\{P\}\ S\ \{R\},\ Q \Rightarrow P}{\{Q\}\ S\ \{R\},}$$

Con-D2

If P is known to be a pre-assertion for a program S to produce result R, then so is any other assertion that logically implies P.

Axiom 2: Axiom of composition

$$\frac{\{P\}\ S1\ \{R1\},\ \{R1\}\ S2\ \{R\}}{\{P\}\ S1;\ S2\ \{R\}}\ \textbf{Com-D3}$$

If the proven result of the first part of a program is identical with the precondition under which the second part of the program produces its intended result, then the whole program will produce the intended result, provided that the precondition of the first part is satisfied.

Take the following two steps to use the rule of composition:

Step 1: Set the target Hoare triple for proof:

{P} S1; S2 {R}

Step 2: Try to find a predicate R1 such that the following two Hoare triples hold:

{P} S1 {R1} and {R1} S2 {R}

Challenge: how to find R1

Example:

The target Hoare triple for proof:

{x = 5 and y = 1} x: = x + 1; y := x + y {x = 6 and y = 7}

Proof:

Step 1: {x = 6 and x + y = 7} (**Ass-D0**)

y := x + y

{x = 6 and y = 7}

This is equivalent to

{x = 6 and y = 1}

y := x + y

{x = 6 and y = 7}

Step 2:

$\{x + 1 = 6 \text{ and } y = 1\}$ (**Ass-D0**)

$x := x + 1$

$\{x = 6 \text{ and } y = 1\}$

This is equivalent to:

$\{ x = 5 \text{ and } y = 1\}$

$x := x + 1$

$\{x = 6 \text{ and } y = 1\}$

Step 3:

{ x = 5 and y = 1} (**Com-D3**)

x: = x + 1;

y := x + y

{x = 6 and y = 7}

Class exercise

Prove the validity of the following Hoare triple:

{x = 5 and y = 2}

x: = x * y;

y := x + y * (x + 5)

{x = 10 and y = 40}

Answer for the exercise

{x = 5 and y = 2}

{x * y = 10 and 10 + 10 * y + y * 5 = 40}

{x * y = 10 and x * y + y * (x * y + 5) = 40}

x: = x * y;

{x = 10 and x + y * (x + 5) = 40}

y := x + y * (x + 5)

{x = 10 and y = 40}

Axiom 3: Axioms of alternation (selection, choice)

(1) if B then S1

$$\frac{\{P \wedge B\}\ S1\ \{R\},\ P \wedge \neg B \Rightarrow R}{\{P\}\ \text{if B then S1}\ \{R\}}\ \textbf{Alt-D4}$$

If the execution of S1 ensures the truth of R under the condition that both P and B are true, and the conjunction of P and the negation of B implies R, then the alternation statement will ensure the truth of R under the pre-assertion P.

(2) if B then S1 else S2

$$\frac{\{P \wedge B\}\ S1\ \{R\},\ \{P \wedge \neg B\}\ S2\ \{R\}}{\{P\}\ \text{if B then S1 else S2}\ \{R\}}\ \textbf{Alt-D5}$$

If the execution of S1 ensures the truth of R under the condition that both P and B are true, and the execution of S2 ensures the truth of R under the condition that the conjunction of P and the negation of B, then the alternation statement will ensure the truth of R under the pre-assertion P.

Important points on the rules of alternation:

(1) The same post-assertion R is used. The following case is incorrect:

$$\frac{\{P \wedge B\}\ S1\ \{R1\},\ \{P \wedge \neg B\}\ S2\ \{R2\}}{\{P\}\ \text{if } B \text{ then } S1 \text{ else } S2\ \{R\}}$$

where neither R1 nor R2 is equivalent to R.

(2) The following alternative rule can be used.

$$\frac{\{P \wedge B\}\ S1\ \{R1\},\ \{P \wedge \neg B\}\ S2\ \{R2\}}{\{P\}\ \text{if } B \text{ then } S1 \text{ else } S2\ \{R1 \vee R2\}}\ \textbf{Alt-D6}$$

where neither R1 nor R2 is equivalent to R.

Proof:

H: $\{P \wedge B\}\ S1\ \{R1\}$

Infer: $\{P \wedge B\}\ S1\ \{R1 \vee R2\}$ (**Con-D1**, R1 => R1 ∨ R2)

H: $\{P \wedge \neg B\}\ S2\ \{R2\}$

Infer: $\{P \wedge \neg B\}\ S2\ \{R1 \vee R2\}$ (**Con-D1**, R2 => R1 ∨ R2)

Infer: $\{P\}$ if B then S1 else S2 $\{R1 \vee R2\}$ (**Alt-D5**)

Example:

The target Hoare triple for proof:

{x <> y}

if x > y

then x := y – 1

else x := y;

{x = y ∨ x < y}

Proof:

Step 1: Prove

$$\{x <> y \land x > y\}\ x := y - 1\ \{x = y \lor x < y\}$$

Step 2: Prove

$$\{x <> y \land x <= y\}\ x := y\ \{x = y \lor x < y\}$$

Step 3: Prove the target Hoare triple.

Step 1: Prove

$$\{x <> y \wedge x > y\}\ x := y - 1\ \{x = y \vee x < y\}$$

Proof:

$\{y - 1 = y \vee y - 1 < y\}$ (**Ass-D0**)

$x := y - 1$

$\{x = y \vee x < y\}$

This is equivalent to

$\{y < y + 1\}$

$x := y - 1$

$\{x = y \vee x < y\}$

Now the main task is to prove:

$x <> y \wedge x > y \Rightarrow y < y + 1$ (L1)

This can be easily proved, because

$y < y + 1 <=> true$

Therefore, we have

$\{x <> y \wedge x > y\}$ (L1, **Con-D2**)

$x := y - 1$

$\{x = y \vee x < y\}$

Step 2: Prove

$$\{x <> y \wedge x <= y\}\ x := y\ \{x = y \vee x < y\}$$

Proof:

$\{y = y \vee y < y\}$ (**Ass-D0**)

$x := y$

$\{x = y \vee x < y\}$

This is equivalent to

$\{true\}$

$x := y$

$\{x = y \vee x < y\}$

Now the main task is to prove:

$x <> y \wedge x <= y \Rightarrow true$ (L2)

This is true according to the rule of =>.

Therefore, we have

$\{x <> y \wedge x <= y\}$ (L2, **Con-D2**)

$x := y$

$\{x = y \vee x < y\}$

Step 3: Prove the target Hoare triple:

{x <> y}
if x > y
then x := y – 1
else x := y;
{x = y ∨ x < y}

Proof:

{x <> y}
if x > y
then x := y – 1
else x := y;
{x = y ∨ x < y}

(step 1, step 2, Alt-D5)

Class exercise

Prove the validity of the following Hoare triple:

$\{x > 0\}$

if $x > y + 10$

then $x := y + 20$

else $y := x + 10$

$\{x = y + 20 \vee x < y\}$

Answer for the exercise

Step 1:

$\{x > 0 \land x > y + 10\}$

$x := y + 20$

$\{x = y + 20 \lor x < y\}$

Step 2:

$\{x > 0 \land x <= y + 10\}$

$y := x + 10$

$\{x = y + 20 \lor x < y\}$

Step 3:

{x > 0}

if x > y + 10

then x := y + 20

else y := x + 10

{x = y + 20 $\vee$ x < y}

Proof:

Step 1:

$\{y + 20 = y + 20 \vee y + 20 < y\}$ Ass-D0

$x := y + 20$

$\{x = y + 20 \vee x < y\}$

$x > 0 \wedge x > y + 10 \Rightarrow$ (L1)

$y + 20 = y + 20 \vee y + 20 < y$

$\{x > 0 \wedge x > y + 10\}$ (L1, Con-D2)

$x := y + 20$

$\{x = y + 20 \vee x < y\}$

Step 2:

$\{x = x + 30 \vee x < x + 10\}$ (Ass-D0)

$y := x + 10$

$\{x = y + 20 \vee x < y\}$

$x > 0 \wedge x <= y + 10 \Rightarrow$ (L2)

$x = x + 30 \vee x < x + 10$

{x > 0 ∧ x <= y + 10 } (L2, Con-D2)

y := x + 10

{x = y + 20 ∨ x < y}

Step 3:

{x > 0} (Step 1, Step 2, Alt-D5)

if x > y + 10

then x := y + 20

else y := x + 10

{x = y + 20 ∨ x < y}

Axiom 4: Axiom of iteration (while B do S)

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \text{while } B \text{ do } S\ \{P \wedge \neg B\}}\ \text{Ite-D7}$$

If P holds and B is true before the execution of statement S and its execution sustains the truth of P, then the execution of the iteration statement under the precondition P will sustain the truth of P and makes B false.

P is called invariant of the loop.

Important points on the rule of iteration:

(1) Deriving the invariant P is a process of abstracting the while-loop statement into a predicate expression that expresses its meaning (or function).

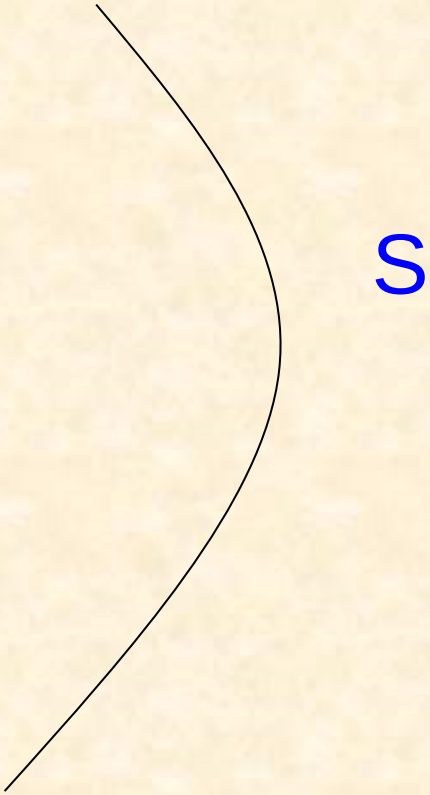
(2) For the same while-loop, there may be more than one invariant. For example, true can be an invariant for any while-loop.

(3) Deriving appropriate invariant P is the most challenging task in program proving, and there is no systematic method for this.

Example:

A program of finding the quotient q and remainder r obtained on dividing x by y, where both x and y are non-zero natural numbers.

```
 r := x;
q := 0;
while y <= r do
 begin
 r := r - y;
 q := 1 + q
 end
```



S

An important property of this program is when it terminates, we can recover the numerator x by adding to the remainder r the product of the divisor y and the quotient q (i.e., $x = r + y \times q$). Furthermore, the remainder is less than the divisor. These properties can be expressed formally:

$$\{true\}\ S\ \{x = r + y \times q \wedge \neg y <= r\}$$

where S represents the program displayed above.

The proof can be conducted in two steps:

Sept 1: {true} r := x; q := 0; {x = r + y × q}

Step 2:

{x = r + y × q}

while y <= r do

  begin

  r := r - y;

  q := 1 + q

  end

{x = r + y × q ∧ ¬y <=r}

A formal proof of the correctness of program S:

1 $true \Rightarrow x = x + y \times 0$ Lemma 1

2 $\{x = x + y \times 0\}\ r := x\ \{x = r + y \times 0\}$ Ass-D0

3 $\{x = r + y \times 0\}\ q := 0\ \{x = r + y \times q\}$ Ass-D0

4 $\{true\}\ r := x\ \{x = r + y \times 0\}$ Con-D2(1,2)

5 $\{true\}\ r := x;\ q := 0\ \{x = r + y \times q\}$ Com-D3(4,3)

6 $x = r + y \times q \wedge y <= r \Rightarrow$

$x = (r - y) + y \times (1 + q)$ Lemma 2

7 $\{x = (r - y) + y \times (1 + q)\}\ r := r - y$

$\{x = r + y \times (1 + q)\}$ Ass-D0

8 $\{x = r + y \times (1 + q)\}\ q := 1 + q$

$\{x = r + y \times q)\}$ Ass-D0

9 {x = (r – y) + y × (1 + q)} r := r – y; q := 1 + q

{x = r + y × q)} Com-D3(7, 8)

10 {x = r + y × q) ⋀ y <= r} r := r – y; q := 1 + q

{x = r + y × q)} Con-D2(6,9)

11 {x = r + y × q)}

while y <= r do

begin

r := r – y; q := 1 + q

end

{x = r + y × q ⋀ ¬y <= r} **Ite-D7(10)**

12 {true} r := x; q := 0;

while y <= r do

begin

r := r – y; q := 1 + q

end

{x = r + y × q ∧¬y <= r} **Com-D3(5, 11)**

Observation:

This program should have used x > 0 and y > 0 in the pre-assertion, but since it only ensures the termination of the while loop, its absence in the pre-assertion does not affect the proof of the partial correctness.

Software development based on formal verification