Chapter 3: Hoare Logic

Bohua Zhan

Institute of Software, Chinese Academy of Sciences

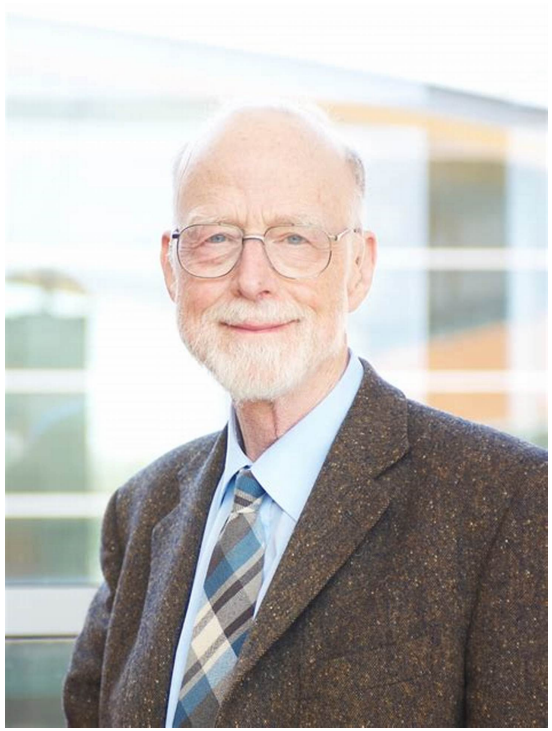
December 2023

Reading

- Concrete Semantics (Nipkow & Klein), Chapter 12
- Specification and Verification I (Mike Gordon), Chapter 1-4, 7
- Introduction to Formal Semantics (Zhou & Zhan), Chapter 3

Floyd-Hoare Logic

Robert W. Floyd
1978 Turing Award



Tony Hoare
1980 Turing Award

Motivation

Let us return to the example program shown earlier:

$$\begin{aligned} P \quad = \quad & \textbf{while } a > 0 \textbf{ do} \\ & \quad a := a - 1 \ ; \\ & \quad c := c + b \end{aligned}$$

We claim that if $a \geq 0$ and $c = 0$ initially, then the effect of the program is to reduce a to 0 and increase c to $a \cdot b$ (and leave b unchanged).

How do we prove this?

Motivation

First informal proof:

Each iteration of the loop reduces a by 1, hence the number of iterations is the initial value of a. Each iteration increases c by b, hence the total increase to c is $a \cdot b$.

OK, but difficult to make precise.

Proof based on invariants:

Let A be the initial value of a. The invariant $c = (A - a) \cdot b$ holds at the beginning of each loop: it holds initially and is preserved by each iteration through the loop. Hence it holds at the end of the loop. Since $a = 0$ at the end, we have $c = A \cdot b$ in the final state.

Easier to make precise and computer-checkable.

Motivation

Proof based on invariants, in more detail:

- **Why is $c = (A - a) \cdot b$ an invariant?**
  Because the loop changes a to $a - 1$ and c to $c + b$, so it suffices to check $c = (A - a) \cdot b$ implies $c + b = (A - (a - 1)) \cdot b$. This holds by arithmetic calculations.
- **Why does $c = (A - a) \cdot b$ hold at the beginning?**
  Because we defined A to be the initial value of a, so at the beginning $A - a = 0$. Also $c = 0$ at the beginning.
- **Why does $c = (A - a) \cdot b$ imply the conclusion?**
  Since we exited the loop, it must be the case that $a \leq 0$. We also need $a \geq 0$ (oops!) to get $a = 0$. So $c = A \cdot b$.

Also need $a \geq 0$ as an invariant.

Motivation

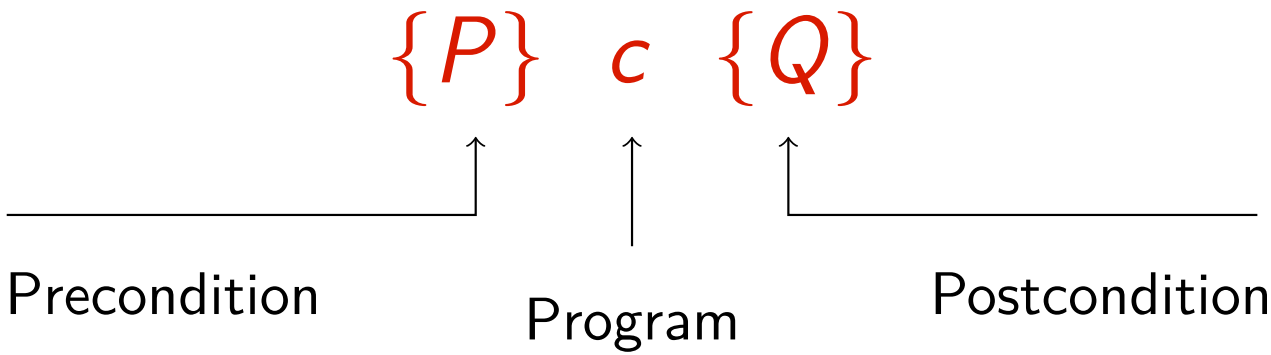
Let's try again:

- **Why is $c = (A - a) \cdot b \wedge a \geq 0$ an invariant?**
  Because the loop changes a to $a - 1$ and c to $c + b$, so it suffices to check $c = (A - a) \cdot b \wedge a \geq 0$ implies $c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0$. The first part holds by arithmetic calculations. Also, since we entered the loop, it must be the case that $a > 0$, so $a - 1 \geq 0$.
- **Why does $c = (A - a) \cdot b \wedge a \geq 0$ hold at the beginning?**
  Because we defined A to be the initial value of a, so at the beginning $A - a = 0$. Also $c = 0$ at the beginning. Also, we assumed $A \geq 0$ initially.
- **Why does $c = (A - a) \cdot b \wedge a \geq 0$ imply the conclusion?**
  Since we exited the loop, it must be the case that $a \leq 0$. Also $a \geq 0$ by the invariant. So $a = 0$. So $c = A \cdot b$.

Hoare Logic

- Proofs about programs based on invariants (or any other method) are error-prone.
- Hoare logic is a structured way of formulating such proofs, making them checkable by the computer.

Hoare Triple



- (Partial correctness) If P holds on the initial state, and if the execution of c terminates, then Q holds on the final state.
- (Total correctness) If P holds on the initial state, then c terminates and Q holds on the final state.

$$\text{Total correctness} = \text{partial correctness} + \text{termination}$$

Examples of Hoare triples

$$\{x = 5\}\ x := x + 5\ \{x = 10\}$$

$$\{\texttt{True}\}\ x := 10\ \{x = 10\}$$

$$\{x = y\}\ x := x + 1\ \{x \neq y\}$$

Hoare triples as specification

$$\{a \geq 0 \wedge c = 0\}$$
$$\textbf{while } a > 0 \textbf{ do}$$
$$a := a - 1 \ ;$$
$$c := c + b$$
$$\{c = a \cdot b\}$$

Completely wrong! Postcondition refers to values in the final state, where $a = 0$.

Hoare triples as specification

$$\{a = A \wedge c = 0 \wedge A \geq 0\}$$
$$\textbf{while } a > 0 \textbf{ do}$$
$$\quad a := a - 1 \ ;$$
$$\quad c := c + b$$
$$\{c = A \cdot b\}$$

If initially $a = A \geq 0$ and $c = 0$, then after running the program we have $c = A \cdot b$. Here A is a constant (not a program variable).

Specifying factorial

$$\{a = 1 \wedge n \geq 0 \wedge b = 1\}$$
$$\textbf{while } a \leq n \textbf{ do}$$
$$\quad b := b \cdot a \ ;$$
$$\quad a := a + 1$$
$$\{b = n!\}$$

The given **while** loop computes the factorial function.

OK, but $b := 1; n := 1$ would also satisfy specification.

Specifying factorial

$$\{a = 1 \wedge n = N \wedge N \geq 0 \wedge b = 1\}$$

while $a \leq n$ **do**

$b := b \cdot a$;

$a := a + 1$

$$\{b = N! \wedge n = N\}$$

The given **while** loop computes the factorial function, leaving n unchanged.

Specifying search

Assume the language is extended with the concept of arrays.
Let $\mathit{array}(a, A)$ denote the value of variable a is the list A.

$$\{\mathit{array}(a, A) \wedge x = X \wedge \exists i.\ X = A[i]\}$$
$$\mathbf{binary\_search}(a, x) \qquad \text{(return value at } b\text{)}$$
$$\{\mathit{array}(a, A) \wedge x = X \wedge A[b] = X\}$$

Specifying sorting

Let $count(A, x)$ denote the number of times x appears in list A.

$\{array(a, A)\}$

$\mathbf{sort}(a)$ (sort a inplace)

$\{array(a, A') \wedge sorted(A') \wedge \forall x.\ count(A, x) = count(A', x)\}$

Hoare rules

- Hoare logic is defined by a finite set of rules, just like big-step or small-step operational semantics.
- A Hoare triple is provable if it can be derived from these rules. Provability is denoted by $\vdash \{P\}\ c\ \{Q\}$.

Hoare rules

- Rule for skip:

$$\frac{}{\vdash \{P\}\ \mathbf{skip}\ \{P\}}$$

- Rule for assignment:

$$\frac{}{\vdash \{P[a/x]\}\ x := a\ \{P\}}$$

  Here $P[a/x]$ is formed by replacing occurrences of x in P by a.

- Rule for sequence:

$$\frac{\vdash \{P_1\}\ c_1\ \{P_2\} \quad \vdash \{P_2\}\ c_2\ \{P_3\}}{\vdash \{P_1\}\ c_1; c_2\ \{P_3\}}$$

Hoare rules

- Rule for if:

$$\frac{\vdash \{P_1\}\ c_1\ \{Q\} \quad \vdash \{P_2\}\ c_2\ \{Q\}}{\vdash \{\text{if } b \text{ then } P_1 \text{ else } P_2\}\ \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2\ \{Q\}}$$

- Rule for while:

$$\frac{\vdash \{P \wedge b\}\ c\ \{P\}}{\vdash \{P\}\ \textbf{while } b \textbf{ do } c\ \{P \wedge \neg b\}}$$

- Consequence rule:

$$\frac{P' \longrightarrow P \quad \vdash \{P\}\ c\ \{Q\} \quad Q \longrightarrow Q'}{\vdash \{P'\}\ c\ \{Q'\}}$$

Invariant rule

- Rule for while (alternative form):

$$\frac{\vdash \{I \wedge b\}\ c\ \{I\} \quad I \wedge \neg b \longrightarrow Q}{\vdash \{I\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{Q\}}$$

I is known as invariant of the loop

Derivation:

$$\frac{\dfrac{\vdash \{P \wedge b\}\ c\ \{P\}}{\vdash \{P\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{P \wedge \neg b\}} \quad P \wedge \neg b \longrightarrow Q}{\vdash \{P\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{Q\}}$$

Example proof – outline

Assume $A \geq 0$ throughout.

$\{a = A \wedge c = 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0\}$

while $a > 0$ **do**

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0\}$

$\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}$

$a := a - 1$;

$\{c + b = (A - a) \cdot b \wedge a \geq 0\}$

$c := c + b$

$\{c = (A - a) \cdot b \wedge a \geq 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge \neg a > 0\}$

$\{c = A \cdot b\}$

Example proof

$$\frac{}{\{c + b = (A - a) \cdot b \wedge a \geq 0\}\ c := c + b\ \{c = (A - a) \cdot b \wedge a \geq 0\}}\ \text{assign}$$

$$\frac{}{\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}\ a := a - 1\ \{c + b = (A - a) \cdot b \wedge a \geq 0\}}\ \text{assign}$$

Let D denote the loop body $a := a - 1; c := c + b$

$$\frac{\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}\ a := a - 1\ \{c + b = (A - a) \cdot b \wedge a \geq 0\} \quad \{c + b = (A - a) \cdot b \wedge a \geq 0\}\ c := c + b\ \{c = (A - a) \cdot b \wedge a \geq 0\}}{\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}\ D\ \{c = (A - a) \cdot b \wedge a \geq 0\}}\ \text{seq}$$

Example proof (continued)

Now apply the consequence rule:

$$\frac{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0 \longrightarrow c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0 \quad \{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}\ D\ \{c = (A - a) \cdot b \wedge a \geq 0\}}{\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0\}\ D\ \{c = (A - a) \cdot b \wedge a \geq 0\}}\ \text{conseq}$$

Followed by the while rule:

$$\frac{\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0\}\ D\ \{c = (A - a) \cdot b \wedge a \geq 0\}}{\{c = (A - a) \cdot b \wedge a \geq 0\}\ \textbf{while}\ a > 0\ \textbf{do}\ D\ \{c = (A - a) \cdot b \wedge a \geq 0 \wedge \neg a > 0\}}\ \text{while}$$

where the invariant P is $c = (A - a) \cdot b \wedge a \geq 0$.

Example proof (continued)

The proof concludes with another consequence rule:

$$\frac{\begin{array}{c} a = A \wedge c = 0 \longrightarrow c = (A - a) \cdot b \wedge a \geq 0 \\ c = (A - a) \cdot b \wedge a \geq 0 \wedge \neg a > 0 \longrightarrow c = A \cdot b \\ \{c = (A - a) \cdot b \wedge a \geq 0\}\ \textbf{while}\ a > 0\ \textbf{do}\ D\ \{c = (A - a) \cdot b \wedge a \geq 0 \wedge \neg a > 0\} \end{array}}{\{a = A \wedge c = 0\}\ \textbf{while}\ a > 0\ \textbf{do}\ D\ \{c = A \cdot b\}}\ \text{conseq}$$

Exercise: carefully check each Hoare rule is correctly applied, and the proof corresponds to the outline.

Verification condition generation

- Performing the above for each program is painfully tedious. Fortunately, there is a better way.
- Verification condition generation: after specifying pre/postcondition and an invariant for each loop, all other assertions can be generated automatically.
- Key observation: in rules for skip, assign, seq, if, and the invariant rule, the postcondition is arbitrary.

Principle: Derive assertions from bottom to top.

VCG: atomic commands

For atomic commands **skip** and $x := a$:

$$\frac{}{\{P\}\ \mathbf{skip}\ \{P\}}$$

$$\frac{}{\{P[a/x]\}\ x := a\ \{P\}}$$

once the postcondition is known, the precondition can be derived.

VCG: sequence case

For a sequence $c_1; c_2; \ldots; c_n$:

$\{P\}$

$\{P_1\}$

$\{P\}$

$\{P_1\}$

$c_1;$

$\vdots$

$\{P_{n-1}\}$

$\{P_{n-1}\}$

$c_{n-1};$

$\{P_n\}$

$\{P_n\}$

c_n

VCG: if case

For the command $\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2$:

$\{P\}$

$\{\text{if } b \text{ then } P_1 \text{ else } P_2\}$

$\{P\}$

$\{\text{if } b \text{ then } P_1 \text{ else } P_2\}$

$\mathbf{if}\ b$

$\quad \{P_1\}$

$\quad \{P_1\}$

$\quad c_1$

$\quad \{Q\}$

$\quad \{Q\}$

$\mathbf{then}$

$\quad \{P_2\}$

VCG: while case

For the command $\mathbf{while}\ b\ \mathbf{do}\ c$:

$\{P\}$

$\{I\}$

$\{P\}$

$\{I\}$

$\{I\}$

$\mathbf{while}\ b$ (invariant I)

$\quad \{I \wedge b\}$

$\quad \{I \wedge b\}$

$\quad \{P'\}$

$\quad \{I \wedge b\}$

$\quad \{P'\}$

$\quad c$

Example revisited

$\{a = A \wedge c = 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0\}$

$\{a = A \wedge c = 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0\}$

while $a > 0$ **do** invariant $c = (A - a) \cdot b \wedge a \geq 0$

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0\}$

$\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0\}$

$\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0\}$

$a := a - 1$;

$\{c + b = (A - a) \cdot b \wedge a \geq 0\}$

VCG Method

- Start with an annotated program, with precondition, postcondition, and invariant for each loop.
- VCG reduces correctness of the program to a number of verification conditions, like:

$$c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0$$
$$\longrightarrow c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0$$

- The verification conditions can be solved using SMT solvers, which (tries to) decide the validity of logical formulas involving arithmetic, arrays, etc.

Soundness of Hoare logic

Two approaches to Hoare logic:

- Assume the rules are correct. That is, the semantics of the language is *defined* by these rules (axiomatic semantics).
- Prove the soundness of the rules in terms of operational or denotational semantics.

Soundness of Hoare logic

定义 (Validity for partial correctness)

A Hoare triple $\{P\}\ c\ \{Q\}$ is valid for partial correctness in terms of big-step operational semantics if

$$\forall s\ t.\ P(s) \land (c, s) \Rightarrow t \longrightarrow Q(t)$$

We use $\models \{P\}\ c\ \{Q\}$ to denote validity of Hoare triples. Soundness of Hoare logic states that all Hoare triples that can be derived are valid:

定理 (Soundness)

$\vdash \{P\}\ c\ \{Q\} \longrightarrow \models \{P\}\ c\ \{Q\}$

Proof of soundness

- skip: We need to show $P(s)$ and $(\mathbf{skip}, s) \Rightarrow t$ implies $P(t)$. From $(\mathbf{skip}, s) \Rightarrow t$ we get $s = t$, hence the result follows.
- assign: We need to show $P[a/x](s)$ and $(x := a, s) \Rightarrow t$ implies $P(t)$. From $(x := a, s) \Rightarrow t$ we get $t = s(x := a)$. On the other hand, $P[a/x](s) = P(s(x := a))$, hence the result follows.
- seq: Given $\{P_1\}\ c_1\ \{P_2\}$ and $\{P_2\}\ c_2\ \{P_3\}$, we need to show $\{P_1\}\ c_1; c_2\ \{P_3\}$. This means showing $P_1(s)$ and $(c_1; c_2, s) \Rightarrow t$ implies $P_3(t)$. From $(c_1; c_2, s) \Rightarrow t$, there must exist s' such that $(c_1, s) \Rightarrow s'$ and $(c_2, s') \Rightarrow t$. Then $P_1(s)$ and $\{P_1\}\ c_1\ \{P_2\}$ implies $P_2(s')$, and $P_2(s')$ and $\{P_2\}\ c_2\ \{P_3\}$ implies $P_3(t)$. So the result follows.

Proof of soundness

- if: Given $\{P \wedge b\}\ c_1\ \{Q\}$ and $\{P \wedge \neg b\}\ c_2\ \{Q\}$, we need to show $\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}$. This means showing $P(s)$ and $(\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2, s) \Rightarrow t$ implies $Q(t)$. Divide into cases based on whether $[\![b]\!]_s$ holds. If $[\![b]\!]_s = \text{true}$ then it must be that $(c_1, s) \Rightarrow t$. This combined with $\{P \wedge b\}\ c_1\ \{Q\}$ shows $Q(t)$. The case where $[\![b]\!]_s = \text{false}$ is similar.

Proof of soundness

- while: We assume $\{P \wedge b\}\ c\ \{P\}$ and wish to show

$$\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}.$$

So assume $P(s)$ and $(\textbf{while}\ b\ \textbf{do}\ c, s) \Rightarrow t$, and the goal is to show $P(t)$ and $[\![b]\!]_t = \text{false}$. Divide into cases based on whether $[\![b]\!]_s$ holds. If $[\![b]\!]_s = \text{false}$, then $s = t$, and the result follows. If $[\![b]\!]_s = \text{true}$, then there must exist state s' such that $(c, s) \Rightarrow s'$ and $(\textbf{while}\ b\ \textbf{do}\ c, s') \Rightarrow t$. From $(c, s) \Rightarrow s'$ and $\{P \wedge b\}\ c\ \{P\}$ we get $P(s')$. By inductive hypothesis, the Hoare triple to be proved applies to $(\textbf{while}\ b\ \textbf{do}\ c, s') \Rightarrow t$, so that $P(s')$ implies $P(t)$ and $[\![b]\!]_t = \text{false}$. This finishes the proof.

Completeness

Completeness of Hoare logic states that all Hoare triples that are valid can be derived:

定理 (Completeness)

$\models \{P\}\ c\ \{Q\} \longrightarrow \vdash \{P\}\ c\ \{Q\}$

Proof sketch: define the concept of weakest precondition:

定义 (Weakest precondition)

Given a program c and an assertion Q, the weakest precondition $wp_{c,Q}$ is the weakest assertion P such that $\models \{P\}\ c\ \{Q\}$. It can be defined as

$$wp_{c,Q}(s) \longleftrightarrow (\forall t.\ (c,s) \Rightarrow t \longrightarrow Q(t)).$$

Proof of completeness (sketch)

It then follows that

$$\models \{P\}\ c\ \{Q\} \text{ iff } P \longrightarrow wp_{c,Q}$$

First prove (by induction on c)

$$\vdash \{wp_{c,Q}\}\ c\ \{Q\}$$

holds for any program c and postcondition Q. The only tricky case is when c is a **while** loop. In this case, we need to show $wp_{c,Q}$ is an invariant of the loop. This follows from the definition of weakest precondition.

The result is simple: if $\models \{P\}\ c\ \{Q\}$, then $P \longrightarrow wp_{c,Q}$. We also have $\vdash \{wp_{c,Q}\}\ c\ \{Q\}$ by the above. So by the consequence rule, we get $\vdash \{P\}\ c\ \{Q\}$.

Discussion

What does the completeness theorem mean?

- It shows that for any **while** loop and a valid Hoare triple for it, there exists some invariant (in particular, given by the weakest precondition) allowing the Hoare triple to be proved using Hoare logic.
- The invariant, however, may be extremely complicated, possibly not expressible using the usual operations.
- Moreover (in *really* wild cases), implications between assertions (e.g. from P to $wp_{c,Q}$) may be *valid* but not *provable* in the logic we choose.
- For programs that appear in practice, invariants exist and are expressible, but finding them is usually not automatic and can be very challenging. This is the main challenge when using Hoare logic.

Total correctness

$$\text{Total correctness} = \text{partial correctness} + \text{termination}$$

- To show total correctness, we also need to show all **while** loops terminate.
- This is usually done by proving a ranking function: a function from states to natural numbers (or more generally any well-ordered set) such that each iteration through the loop decreases the value of the ranking function.
- Hoare logic for total correctness is again defined by a set of rules. A Hoare triple is provable (denoted $\vdash_t \{P\}\ c\ \{Q\}$) if it can be derived from these rules.

Hoare rules

- Rules for skip, assign, seq, and if are the same as before (with $\vdash$ replaced by $\vdash_t$).
- Rule for while:

$$\frac{\forall n.\ \vdash_t \{P \wedge b \wedge f = n\}\ c\ \{P \wedge f < n\}}{\vdash_t \{P\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{P \wedge \neg b\}}$$

Here f is the ranking function (like P and b, it is a function of the state, but taking values in natural numbers rather than booleans).

Example

For the program

$$\begin{array}{l}\textbf{while } a > 0 \textbf{ do}\\ \quad a := a - 1\ ;\\ \quad c := c + b\end{array}$$

we can take a to be the ranking function. Since $a \geq 0$ throughout the execution, it is a natural number that decreases at every iteration.

In detail:

$$\frac{\forall n.\ \{c = (A-a)\cdot b \wedge a \geq 0 \wedge a > 0 \wedge a = n\}\ D\ \{c = (A-a)\cdot b \wedge a \geq 0 \wedge a < n\}}{\{c = (A-a)\cdot b \wedge a \geq 0\}\ \textbf{while } a > 0 \textbf{ do } D\ \{c = (A-a)\cdot b \wedge a \geq 0 \wedge \neg a > 0\}}$$

Example – outline

$\{a = A \wedge c = 0\}$

$\{c = (A - a) \cdot b \wedge a \geq 0\}$

while $a > 0$ **do** invariant $c = (A - a) \cdot b \wedge a \geq 0$

variant a

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a > 0 \wedge a = n\}$

$\{c + b = (A - (a - 1)) \cdot b \wedge a - 1 \geq 0 \wedge a - 1 < n\}$

$a := a - 1$;

$\{c + b = (A - a) \cdot b \wedge a \geq 0 \wedge a < n\}$

$c := c + b$

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge a < n\}$

$\{c = (A - a) \cdot b \wedge a \geq 0 \wedge \neg a > 0\}$

$\{c = A \cdot b\}$

Soundness (for total correctness)

定义 (Validity for total correctness)

A Hoare triple $\{P\}\ c\ \{Q\}$ is valid for total correctness in terms of big-step operational semantics if

$$\forall s.\ P(s) \longrightarrow (\exists t.\ (c, s) \Rightarrow t \wedge Q(t))$$

We use $\models_t \{P\}\ c\ \{Q\}$ to denote validity of Hoare triples for total correctness. Note that here we assume c is deterministic.

定理 (Soundness)

$\vdash_t \{P\}\ c\ \{Q\} \longrightarrow \models_t \{P\}\ c\ \{Q\}$

Proof of soundness

- Proof for skip, assign, seq, and if are the same as before.
- while: Given $\forall n.\ \{P \wedge b \wedge f = n\}\ c\ \{P \wedge f < n\}$, we need to show $\{P\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{P \wedge \neg b\}$. This means we have some state s satisfying $P(s)$, and need to show there exists state t such that $(\mathbf{while}\ b\ \mathbf{do}\ c, s) \Rightarrow t$ and $Q(t)$. If $[\![b]\!]_s = \text{false}$ the claim follows as before. Now suppose $[\![b]\!]_s = \text{true}$. Perform induction on $f(s)$. Let $n = f(s)$. By the assumption, there exists state s' such that $(c, s) \Rightarrow s'$ and $P(s') \wedge f(s') < n$. By inductive hypothesis, there exists state t such that $(\mathbf{while}\ b\ \mathbf{do}\ c, s') \Rightarrow t$ and $Q(t)$. By while rule of big-step semantics, it follows that $(\mathbf{while}\ b\ \mathbf{do}\ c, s) \Rightarrow t$, as desired.

Completeness (for total correctness)

定理 (Completeness)

$\models_t \{P\}\ c\ \{Q\} \longrightarrow \vdash_t \{P\}\ c\ \{Q\}$

Proof sketch: modify the concept of weakest precondition for total correctness:

定义 (Weakest precondition for total correctness)

The weakest precondition for total correctness $wpt_{c,Q}$ is the weakest assertion P such that $\models_t \{P\}\ c\ \{Q\}$. It can be defined as

$$wpt_{c,Q}(s) \longleftrightarrow (\exists t.\ (c,s) \Rightarrow t \wedge Q(t)).$$

As before, show $\vdash_t \{wpt_{c,Q}\}\ c\ \{Q\}$ for every c and Q.

Termination is undecidable

Not just theoretically (proved by Alan Turing), but in practice:

Whether the program

$$
\begin{array}{l}
\textbf{while } x > 1 \textbf{ do} \\
\quad \textbf{if } x \bmod 2 = 1 \textbf{ then} \\
\quad\quad x := 3 \cdot x + 1 \\
\quad \textbf{else} \\
\quad\quad x := x \operatorname{div} 2
\end{array}
$$

terminates for all starting value of x is an open conjecture (Collatz conjecture).

Discussion

- Again, the completeness theorem only shows there exists some ranking function (for example: given state s, let $f(s)$ be the number of iterations of the loop required).
- It does not mean the ranking function can be expressed using the usual operations, or that the resulting assertions can be proved.
- For most cases occurring in practice, the ranking function is easy to find. But in some special cases (like the one on the previous slide), finding the ranking function may be extremely difficult.

Conclusion

The techniques described so far form the backbone of both semi-automatic and interactive program verification, and have been implemented in many practical tools.

Extensions:

- Arrays (relatively simple).
- Recursion.
- Pointers and aliasing (separation logic).
- Concurrency (Owicki/Gries and rely-guarantee).

Stay-tuned for more!