Sir Charles Antony Richard Hoare

There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

— Tony Hoare —

Picture taken in 2011
Hoare logic

- Hoare:
  - discovered Hoare Logic (in 1969), quicksort, CSP, occam, …

- Hoare Logic is an inference system
  - used to prove properties of programs.
  - based on proof rules, called Hoare triples:

\[
\{P\} Q \{R\}
\]

where
- P is a precondition
- Q is a program or a program statement
- R post-condition.
Change of state

What does $x:=1$ do?

- It is an action that assigns the value 1 to location $x$
- It is an action that results in a change of state (in the computer)
  - Before the statement, the pre-condition, we could be in ‘any state’
    - Any state satisfies ‘any state’, so we say this state satisfies the predicate $true$
  - After the statement, the post-condition, the predicate $x=1$ is true
- We express the change of state by a Hoare triple
  
  \[
  \{true\} \ x:=1 \ {x=1}\]

- Another example: what does $if \ x<0 \ then \ x:=-x$ do? The Hoare triple is:
  
  \[
  \{true\} \ if \ x<0 \ then \ x:=-x \ {x\geq0}\]

- The precondition and post-condition form a specification for the fragment of code
Specification and implementation

Any number of programs may implement the same specification.

Here are two identical specs:

- $\{true\} \quad x:=y \quad \{x=y\}$
  - Correct, but a poor specification. What is the intention?

- $\{true\} \quad x:=5; \quad y:=5 \quad \{x=y\}$
  - Same (poor) spec, but a different program. Again, what is the intention?

A better way to specify what is probably intended

- $\{x=a \land y=b\} \quad x:=y \quad \{x=b \land y=b\}$
  - $a$ and $b$ are dummy variables
Hoare Triple examples

- \{true\} \ x:=5 \quad \{x=5\}
- \{x=y\} \ x:=x+3 \quad \{x=y+3\}
- \{x>-1\} \ x:=x*2+3 \quad \{x>1\}
- \{x=a\} \quad \text{if} \ (x<0) \ \text{then} \ x:=-x \quad \{x=|a|\}
- \{false\} \ x:=3 \quad \{x=12345678\}
- \{x<0\} \quad \text{while} \ (x!=0) \ x:=x-1 \quad \text{does not exist}
Examples of valid Hoare Triples:
- \{x=5\} \ x:=x*2 \ \{true\}
- \{x=5\} \ x:=x*2 \ \{x>0\}
- \{x=5\} \ x:=x*2 \ \{x=10 \lor x=5\}
- \{x=5\} \ x:=x*2 \ \{x=10\} \ this \ has \ the \ strongest \ post-condition

All triples are true, but the last one is the most useful because it contains the strongest post-condition \(x=10\)

It is strongest because this post-condition implies all the post-conditions:
- \(x=10 \rightarrow true\)
- \(x=10 \rightarrow x>0\)
- \(x=10 \rightarrow (x=10 \lor x=5)\)
- \(x=10 \rightarrow x=10\)

Mathematically, if \(\{P\} S \ \{Q\}\) and for all \(R\) such that \(\{P\} S \ \{R\}\) it is the case that \(Q \rightarrow R\), then \(Q\) is the strongest post-condition
More examples of valid Hoare triples:

- \{x=5 \land y=10\} \ z:=x/y \ \{z<1\}
- \{x<y \land y>0\} \ z:=x/y \ \{z<1\}
- \{y\neq 0 \land x/y<1\} \ z:=x/y \ \{z<1\}

All are true, but the most useful triple is the last one because it contains the \text{weakest precondition}.

Using it allows the code to be used in the most general way.

It is weakest because it implies each of the (other) preconditions.

Mathematically, if \{P\} S \{Q\} and for all N such that \{N\} S \{Q\} it is the case that N \rightarrow P, then P is the weakest precondition \wp(S,Q) of S with respect to Q.
What makes a predicate strong/weak?

P → Q  

Remember, P is the antecedent and Q is the consequent

- P is stronger than Q. P is more restricted than Q
- Q is weaker than P. Q is more general than P
- There are objects in Q that do not come from P

Examples

- x can lift 100kg → x can lift 10kg
- x lives in Sydney → x lives in Australia
- x > 0 → x ≥ 0
- false → true  ... true is the weakest possible predicate, false the strongest

We can change the strength of the antecedent or consequent ...
Let’s revisit the weightlifter

\( P: \) x can lift 100kg, \( Q: \) x can lift 10kg

- \( P \rightarrow Q \) means
  - If \( x \) can lift 100kg then \( x \) can lift 10kg
  - If \( x \) cannot lift 100kg, nothing happens

- What’s the precondition of \( P \rightarrow Q \)?
  - \( x \) can lift 100kg, i.e. \( P \)
  - or \( x \) cannot lift 100kg, i.e. \( \neg P \)
  - so we can write \( \{P \lor \neg P\} \ P \rightarrow Q \ \{ \ ? \} \)

- What’s the post-condition?
  - Well, if \( P \) is true then the state changes to \( Q \)
  - if \( P \) is not true then nothing changes so all we have is \( \neg P \)
  - so we can write \( \{\text{true}\} \ P \rightarrow Q \ \{Q \lor \neg P\} \)

In other words, \( P \rightarrow Q \ \equiv \ \{Q \lor \neg P\} \)  Material implication rule, see earlier
... adding conditions to change strength

If \( P \rightarrow Q \) we can:

- **strengthen the antecedent**: add conditions with \( \land \)
  - \( \ldots \) then \( P \land Q \rightarrow P \) is true
  - see for yourself

- **weaken the consequent**: add conditions with \( \lor \)
  - \( \ldots \) then \( P \rightarrow P \lor Q \) is true

**Examples**

- \((x \text{ likes reading} \land x \text{ likes running}) \rightarrow (x \text{ likes reading})\)
- \((x \text{ likes reading}) \rightarrow (x \text{ likes reading} \lor x \text{ likes running})\)

**Applying this to Hoare’s triples:**

- Example: the triple \( \{x=1\} \ x:=x+1 \ \{x=2\}\)
- We are allowed to strengthen the precondition:
  - \( \{x=1 \land y=1\} \ x:=x+1 \ \{x=2\}\)
  - \( \ldots \) but in this case it would seem pointless
Calculate the weakest precondition (wp)

**Assignment rule** \(x := E\)

- The weakest precondition \(wp(x := E; P) = [E/x] P\)
- ... so the triple is \(\{[E/x] P\} x := E \{P\}\)

This says substitute \(E\) for \(x\) in the post-condition \(P\)

- Example: fill in the \(wp\) in the triple \(\{wp\} x := 3 \{x+y>0\}\)
  - Apply the rule: \([E/x] P \equiv [3/x](x+y>0) \equiv 3+y>0 \equiv y>-3\)
  - So the solution is \(\{y>-3\} x := 3 \{x+y>0\}\)

- Example: fill in the \(wp\) in the triple \(\{wp\} x := 3*y+z \{x*y-z>0\}\)
  - \([3*y+z/x] (x*y-z>0) \equiv (3*y+z)*y-z>0 \equiv 3*y^2 + z*y-z>0\)
  - So the solution is \(\{3*y^2+z*y-z>0\} x := 3*y+z \{x*y-z>0\}\)
Calculating wp

**Sequence rule** $s_1; s_2$

- $\text{wp}((s_1; s_2); P) = \text{wp}(s_1; \text{wp}(s_2; P))$

**Example**

- What is the wp in the triple $\{wp\} x:=x+1; y:=x+y \{y>5\}$
- $\text{wp}((x:=x+1; y:=x+y); y>5)$
  - $\text{wp}(x:=x+1, \text{wp}(y:=x+y; y>5))$ ... applying the sequence rule
  - $\text{wp}(x:=x+1; x+y>5)$ ... applying assignment rule to $s_2$
  - $x+1+y>5$ ... applying assignment rule to $s_1$
  - $x+y>4$ ... simplifying
Calculating wp

Conditional rules  if $B$ then $s_1$ else $s_2$ and  if $B$ then $s$

- $wp$ (if $B$ then $s_1$ else $s_2$; $P) = B \rightarrow wp(s_1; P) \land \neg B \rightarrow wp(s_2; P)$
  
  $$= (B \land wp(s_1; P)) \lor (\neg B \land wp(s_2; P))$$

  *since: $(P \rightarrow Q) \land (\neg P \rightarrow R) \equiv (P \land Q) \lor (\neg P \land R)$*

- $wp$ (if $B$ then $s$; $P) = B \rightarrow wp(s; P) \land \neg B \rightarrow P$
  
  $$= (B \land wp(s; P)) \lor (\neg B \land P))$$

Example

- What is the $wp$ in  \{wp\} if $x > 0$ then $y := z$ else $y := -z$  \{y > 5\}

  $wp$ (if $x > 0$ then $y := z$ else $y := -z$; $y > 5$)

  $$(x > 0 \land wp(y := z; y > 5)) \lor (x \leq 0 \land wp(y := -z; y > 5))$$

  $wp$ is $(x > 0 \land z > 5) \lor (x \leq 0 \land z < -5)$

  *(which you could have worked out by looking at the code)*
Loops

- Want to prove the triple \{P\} while B do S \{Q\}
  - where B is the loop condition and S is a sequence of statements
- We used ‘weakest precondition’ rules to calculate
  - a single Hoare triple for a single statement
- A loop is different: a set of statements S repeatedly executed
  - we do not know how many times the loop will repeat
    - maybe zero times, or 1 time, or a zillion times
  - ... we try to find what never changes, called the invariant condition \(I\)
  - ... we then need to prove the following:
    - The precondition implies the invariant
    - The invariant is true inside the loop
    - The invariant \((I \land \neg B)\) implies the post-condition

\[ P \rightarrow I \]
\[
\{I \land B\} S \{I\}
\]
\[ I \land \neg B \rightarrow Q \]
What the invariant is doing

The invariant $I$ placed in the code

$P \rightarrow I$ establish

while (B)

$\{I \land B\} S \{I\}$ maintenance

$\{ \begin{array}{l}
S_{\text{first statement of the loop}} \\
\ldots \\
S_{\text{last statement of loop}} 
\end{array}$

$I \land \neg B \rightarrow Q$ conclusion

It can be hard to find $I$. What we know for sure:

- $I$ must be weaker than $P$ because $P \rightarrow I$
- $I$ must be different to $B$ because $I$ must be true and $B$ must be false at the conclusion of the loop
Example: sum of odds is a square

\[ 1 = 1^2 \]
\[ 1+3 = 2^2 \]
\[ 1+3+5=3^2 \]
\[ 1+3+5+7=4^2 \]
\[ ... \]

We’ll write a program that sums odd numbers, and then verify its correctness.
A program that **sums odds**

input $n \geq 0$

\[ s:=0; \]
\[ i:=0; \]

the initialization statements must establish the invariant

while ($i \neq n$)

we need to find an invariant

\[
\begin{align*}
\{ \\
& i:=i+1; \\
& s:=s+(2i-1); \\
\}
\end{align*}
\]

the loop assignments must maintain the invariant

on conclusion, the invariant should produce the result $s=n^2$
**Sums odds**: maintain the invariant

- Observe: a loop iteration \( i \) generates a square number \( s = i^2 \)
  - also notice that \( s = i^2 \) links the result \( s \) and the loop index \( i \)
- So we guess the invariant \( I \) is \( s = i^2 \)
- We know \( B \) is \( i \neq n \)
- Compute \( \{ I \land B \} i := i+1; s := s+(2i-1); \{ I \} \) the Hoare triple to maintain the loop
  \[
  \begin{align*}
  ((s = i^2) \land (i \neq n)) & \implies i := i+1; s := s+(2i-1); \{ s = i^2 \} \quad \text{substitute for } I \text{ and } B \\
  ((s = i^2) \land (i \neq n)) & \implies i := i+1; \{ s + (2i - 1) = i^2 \} \quad \text{assign rule} \\
  ((s = i^2) \land (i \neq n)) & \implies \{ s + (2(i + 1) - 1) = (i + 1)^2 \} \quad \text{assign rule again} \\
  ((s = i^2) \land (i \neq n)) & \implies \{ s = i^2 \} \quad \text{simplify} \\
  ((s = i^2) \land (i \neq n)) & \implies s = i^2 \quad \text{antecedent strengthening} \\
  \end{align*}
  \]
- So \( s = i^2 \) is an invariant.
**Sums odds**: Establish the invariant, produce the result

- Does the initialization code establish the invariant?
  
  \[
  \begin{align*}
  \{wp\} & \ i:=0; \ s:=0; \ \{s=i^2\} \\
  \{wp\} & \ i:=0; \ \{0:=i^2\} \\
  wp & = 0:=0^2 \\
  \text{true}
  \end{align*}
  \]

- On conclusion, \( I \land \neg B \rightarrow Q \), where the postcondition \( Q \) is \( s=n^2 \)
  
  \[
  \begin{align*}
  (s=i^2) \land \neg (i\neq n) & \rightarrow (s=n^2) \\
  (s=i^2) \land (i=n) & \rightarrow (s=n^2) \\
  (s=n^2) & \rightarrow (s=n^2) \\
  \text{true}
  \end{align*}
  \]

We have successfully verified the “sums odds” program.
Correctness: partial or total

If we prove the invariant condition is always true

- code is **functionally** correct for any number of loop repetitions, even infinite (*what?*), but
  - What happens if the loop never terminates? Is that correct?
  - It depends how you define ‘correctness’.
    - ... it is often called **partial** correctness for this reason

A loop is **totally** correct if we prove

- the loop **terminates**

Need to show the loop index (**variant**)

1. makes progress towards some limit and
2. reaches the limit
Validity

If a conclusion is true for all assignments to its variables, then the predicate expression is valid. Examples:

- $\forall x (\text{isPerson}(x) \rightarrow \text{isMortal}(x))$ for the domain of objects
- $\forall x (x^2 \geq x)$ for the domain of integers

If there is an assignment that invalidates the expression, this is called a counter-example

- $\forall x (x^2 \geq x)$ for the domain of reals
  - a counter-example is $x = \frac{1}{2}$
  - this predicate is true for some values of $x$

A predicate that is not valid (everywhere) is called invalid.

- they are satisfied for some variables only
Satisfiability

An expression is **satisfiable** if it is true for at least one assignment to its variables

\[(\exists x \ P(x) \ \land \ \exists x \ Q(x)) \rightarrow \ \exists x \ (P(x) \ \land \ Q(x))\]

- e.g. consider P is (wear a red shirt), Q is (wear green shorts)
  - is **satisfiable** because someone could be wearing both
  - but it is not valid as it is not always true

- An expression that is **satisfiable** everywhere is valid
- An expression that is not **satisfiable** everywhere is invalid

An expression that is never satisfied (always false) is **unsatisfiable**

\[\forall x \ P(x) \ \land \ \exists x \ \neg P(x)\] is unsatisfiable

- first term says P is *always true*
- second term says P is *sometimes not true*: a **contradiction**
Inference rules

In Hoare Logic, there are inference rules written as:

\[
\text{premise}_1 \quad \text{premise}_2 \quad \ldots
\]
\[
\text{conclusion}
\]

- can also be written \( \text{prem}_1, \text{prem}_2, \ldots \vdash \text{conclusion} \)
- if the premises are true then the conclusion is true

There are rules for inference on many program constructs:

- assignments \( x := a \)
- if-then-else \( \text{if } e \text{ then } S_1 \text{ else } S_2 \text{ fi} \)
- iteration \( \text{while } e \text{ do } S \text{ od} \)
- composition \( S_1; S_2 \)
D0: Assignments

Inference rule for assignments

\[
\{[E/x]P\} \ x:=E \ \{P\}
\]

Calculate precondition by replacing all x’s in P with E

Example:

\[
\{[x+1/x] \ (x<10) \ x:= x+1 \ \{x<10\}\}
\]

\[
\{x<9\} \ x:= x+1 \ \{x<10\}\]
D1: Consequence (strengthening & weakening)

- Inference rule for *post-condition weakening*
  \[
  \{P\} Q \{R\} \quad R \rightarrow S
  \]
  \[
  \{P\} Q \{S\}
  \]
  \[
  \{y+1<10\} y := y+1 \quad \{y<10\} \quad y<10 \rightarrow x=1
  \]
  \[
  \{y+1<10\} \quad y := y+1 \quad \{x=1\}
  \]

- Inference rule for *precondition strengthening*
  \[
  S \rightarrow P \quad \{P\} Q \{R\}
  \]
  \[
  \{S\} Q \{R\}
  \]
Inference rule for consequence

\[
\begin{array}{c}
M \rightarrow P \\
\{P\} Q \{R\} \\
R \rightarrow S \\
\{M\} Q \{S\}
\end{array}
\]

Combines *precondition strengthening* and *post-condition weakening* in one rule:

- If \( M = P \) we get the post-condition weakening rule
- If \( R = S \) we get the precondition weakening rule
**D2: Composition (also called sequencing)**

Inference rule for composition

\[
\begin{align*}
\{P\} & \quad Q_1 \quad \{R_1\} \quad \{R_1\} \quad Q_2 \quad \{R\} \\
\{P\} & \quad Q_1 \quad ; \quad Q_2 \quad \{R\} \\
\{x=1\} & \quad y := x \quad \{y=1\} \quad \{y=1\} \quad z := y \quad \{z=1\} \\
\{x=1\} & \quad y := x; \quad z := y \quad \{z=1\} \\
\{P\} & \quad Q_1 \quad ; \quad Q_2 \quad \{R\}
\end{align*}
\]
D3: Iteration

Inference rule for iteration: while B do S

\[
\{I \land B\} S \{I\}
\]

\[
\{I\} \text { while } B \text { do } S \{I \land \neg B\}
\]

Example: we want to infer:

\[\{x \leq 0\} \text { while } (x \leq 2) \text { do } x := x+1 \{x = 3\}\]

Let invariant \(I\) be \(x \leq 3\)

\[
\{x \leq 3 \land x \leq 2\} \quad x := x+1 \quad \{x \leq 3\}
\]

\[\{x \leq 3\} \text { while } (x \leq 2) \text { do } x := x+1 \quad \{x \leq 3 \land x > 2\}\]

Strengthen the precondition (apply rule D1)

\[x \leq 0 \rightarrow x \leq 2 \quad \{x \leq 2\} \quad x := x+1 \quad \{x \leq 3\}\]

\[\{x \leq 0\} \quad \text { while } (x \leq 2) \text { do } x := x+1 \quad \{x = 3\}\]
D4: conditional (if-then-else)

if-then-else inference rule

\[ \{ P \land B \} \quad Q_1 \quad \{ R \} \quad \{ P \land \neg B \} \quad Q_2 \quad \{ R \} \]

\[ \{ P \} \text{ if } B \text{ then } Q_1 \text{ else } Q_2 \text{ fi } \quad \{ R \} \]

\[ \{ z = 0 \land x < 0 \} \quad y := z - x \quad \{ y > 0 \} \quad \{ z = 0 \land x \geq 0 \} \quad y := z + x \quad \{ y > 0 \} \]

\[ \{ z = 0 \} \text{ if } (x < 0) \quad y := z - x \text{ else } \quad y := z + x \quad \{ y > 0 \} \]
**D4: conditional (if-then)**

if-then inference rule replaces the else statement by `skip`

\[
\begin{align*}
\{ P \land B \} & \quad Q \quad \{ R \} & \quad \{ P \land \lnot B \} & \quad \text{skip} \quad \{ R \} \\
\{ P \} & \quad \text{if } B \text{ then } Q \text{ else skip fi} \quad \{ R \}
\end{align*}
\]

\[
\begin{align*}
\{ \text{true} \land x < 0 \} & \quad x := -x \quad \{ x \geq 0 \} & \quad \{ \text{true} \land x \geq 0 \} & \quad \text{skip} \quad \{ x \geq 0 \} \\
\{ \text{true} \} & \quad \text{if } (x < 0) \quad x := -x \quad \{ x \geq 0 \}
\end{align*}
\]
Revisit D1: let’s prove it

Here is the precondition strengthening inference rule D1 again

\[
\begin{align*}
S & \rightarrow P \quad \{P\} \quad Q \quad \{R\} \\
\{S\} & \quad Q \quad \{R\}
\end{align*}
\]

It is easy to prove this.

- Assume S is true. Then P must be true because S→P.
- If P is true, then run code Q, and R will be true
- Hence we can write \{S\} Q \{R\}

Similarly for post-condition weakening.
Predicate inference rules and equivalences

Disjunction intro. \( P \vdash P \lor Q \)
Conjunction elim. \( P \land Q \vdash P \)
Conjunction \( P, Q \vdash P \land Q \)
Modus ponens \( P, P \rightarrow Q \vdash Q \)
Modus tollens \( \neg Q, P \rightarrow Q \vdash \neg P \)
Hypothetical syllogism \( P \rightarrow Q, Q \rightarrow R \vdash P \rightarrow R \)
Disjunctive syllogism \( P \lor Q, \neg P \vdash Q \)
Resolution \( P \lor Q, \neg P \lor R \vdash Q \lor R \)

The most common equivalences are:

Material implication \( P \rightarrow Q \equiv \neg P \lor Q \)
De Morgan’s Law \( P \land Q \equiv \neg (\neg P \lor \neg Q) \)

Augustus de Morgan (1806-1871)
More predicate logic equivalences

- There are many more equivalences:
  - $\forall x \ P(x) \lor \neg \forall x \ P(x) \equiv \text{true}$
  - $\forall x \ (P(x) \land Q(x)) \equiv \forall x \ P(x) \land \forall x \ Q(x)$
  - $\neg \forall x \ P(x) \equiv \exists x \ \neg P(x)$

- If $\rightarrow Q \land Q \rightarrow P$ then $P$ and $Q$ are equivalent

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P→Q</th>
<th>Q→P</th>
<th>P→Q∧Q→P</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Premise: Innocent people have an alibi

Premise: Cas is innocent
- **Modus ponens**: Cas has an alibi
- A logically correct argument ...
  - and natural

Premise: Cas does not have an alibi
- **Modus tollens**: Cas is guilty
- A logical correct argument ...
  - but not used as often

∀x (I(x) → A(x)) ∧ I(c)

**M.P.** ⇒ A(c)

∀x (I(x) → A(x)) ∧ ¬A(c)

**M.T.** ⇒ ¬I(c)
More on maths and modus tollens

- We know \( x \geq 1 \rightarrow x \geq 0 \)
  - notice \( x = 0 \) is in \( x \geq 0 \) but not in \( x \geq 1 \)
  - remember: strong implies weak, small set implies large set

- Apply modus tollens
  - We get \( \neg(x \geq 0) \rightarrow \neg(x \geq 1) \)
    - \( x < 0 \rightarrow x < 1 \)
      - notice \( x = 0 \) is in \( x < 1 \) but not in \( x < 0 \)
      - so \( x < 1 \) is weaker than \( x < 0 \)

what does this say?
Quantification Rules (summary of earlier slides)

**Universal Instantiation**
For any $a \in D$:
$\forall x \in D, P(x)$
$P(a)$

**Existential Instantiation**
For an unspecified new (witness) $w \in D$:
$\exists x \in D, P(x)$
$P(w)$

**Universal Generalization**
For any arbitrary $x \in D$:
$P(x)$
$\forall x \in D, P(x)$

**Existential Generalization**
For any $a \in D$:
$P(a)$
$\exists x \in D, P(x)$
Problem: find the quotient and remainder of \( x/y \)
- input \( x \geq 0 \) and \( y > 0 \)
- output: \( q \) and \( r \) such that \( x = q \times y + r \), where \( y > r \)

Examples:
- \( 9 \div 2 = 4 \times 2 + 1 \)
- \( 11 \div 1 = 11 \times 1 + 0 \)
- \( 19 \div 5 = 3 \times 5 + 4 \)

Notice that the information above gives us a specification
- precondition \( P \) is \( (x \geq 0) \land (y > 0) \)
- post-condition \( Q \) is \( (x = q \times y + r) \land (r \geq 0) \land (r < y) \land (y > 0) \)
Quotient & remainder code

Code is:

```c
// {P}
r := x;
q := 0;
while (y <= r)
{
   r := r - y;
   q := 1 + q;
}
// {Q}
```

What we actually want is: prove the triple {P} code {Q}, where

- **P** is \((x \geq 0) \land (y > 0)\)
- **Q** is \((x = q \times y + r) \land (y > r) \land (r \geq 0) \land (y > 0)\)
- code is above

We let the invariant \(I\) be \(x = q \times y + r \land (r \geq 0) \land (y > 0)\)
Pre-, post-conditions, invariant for quotient

\[ r := x; \]
\[ q := 0; \]
\[ \text{while}(y \leq r) \]
\[ \{ r := r - y; \]
\[ q := 1 + q; \]
\[ \} \]

\{ (x = 0 * y + x) \land (x \geq 0) \land (y > 0) \} \quad \text{the precondition}

\{ (x = 0 * y + r) \land (r \geq 0) \land (y > 0) \}

\{ (x = q * y + r) \land (r \geq 0) \land (y > 0) \} \quad \text{the invariant}

\{ (x = (1 + q) * y + (r - y)) \land (r - y \geq 0) \land (y > 0) \land (y \leq r) \} \quad \text{same}

\{ (x = (1 + q) * y + r) \land (r \geq 0) \land (y > 0) \land (y \leq r) \}

\{ (x = q * y + r) \land (r \geq 0) \land (y > 0) \land (y \leq r) \}

\{ (x = q * y + r) \land (r \geq 0) \land (y > 0) \land (y > r) \} \quad \text{postcondition}
In the previous slide we showed that the invariant:
- was established by the 2 initialization statements from the precondition
- was maintained by the loop condition \( y \leq r \) and the 2 statements inside the loop
- finished with the required postcondition

So we have verified the program is partially correct.

To show the program is totally correct, we need to

- show that the loop index is bounded from below by 0 for as long as
  the loop has not terminated.

\[
I \land (y \leq r) \rightarrow r \geq 0
\]
\[
(x = q \times y + r) \land (r \geq 0) \land (y > 0) \land (y \leq r) \rightarrow r \geq 0
\]

This is trivially satisfied.
Repeat the verification

We’ll repeat the verification here, but show the inference rules this time

1. true $\rightarrow$ $x = x + y \cdot 0$
   
   Lemma 1, see later

2. $\{x = x + y \cdot 0\}$ $r := x$ $\{x = r + y \cdot 0\}$
   
   D0

3. $\{x = r + y \cdot 0\}$ $q := 0$ $\{x = r + y \cdot q\}$
   
   D0

4. $\{true\}$ $r := x$ $\{x = r + y \cdot 0\}$
   
   D1, 1,2

5. $\{true\}$ $r := x$ $; q := 0$ $\{x = r + y \cdot q\}$
   
   D2, 4,3

6. $(x = r + y \cdot q) \wedge (y \leq r) \rightarrow (x = (r - y) + y \cdot (1 + q))$
   
   Lemma 2, see later

7. $\{x = (r - y) + y \cdot (1 + q)\}$ $r := r - y$ $\{x = r + y \cdot (1 + q)\}$
   
   D0

8. $\{x = r + y \cdot (1 + q)\}$ $q := 1 + q$ $\{x = r + y \cdot q\}$
   
   D0

9. $\{x = (r - y) + y \cdot (1 + q)\}$ $r := r - y$ $; q := 1 + q$ $\{x = r + y \cdot q\}$
   
   D2, 7,8
Repeat the verification continued

10. \{x=r+y\cdot q \land y\leq r\} \ r := r-y; \ q := 1+q \ \{x=r+y\cdot q\} \hspace{2cm} D1,\ 6,\ 9

11. \{x=r+y\cdot q\} \ \textbf{while} \ y\leq r \ \textbf{do} \ r:=r-y; \ q:=1+q \ \{y>r \ \land \ x=r+y\cdot q\} \hspace{2cm} D3,\ 10

12. \{\text{true}\} \ r := x; \ q :=0; \ \textbf{while} \ y\leq r \ \textbf{do} \ r:=r-y; \ q:=1+q \ \{y>r \land x=r+y\cdot q\} \hspace{2cm} D2,\ 5,\ 11

We have verified that the remainder/quotient program satisfies the spec.
Lemma 1  (we needed this in the verification)

To prove $x = x + y \cdot 0$

we use the axioms from arithmetic:

- A0: $x = x$
- A1: $x \cdot 0 = 0$
- A2: $x + 0 = x$

Proof

1. $x = x$  by A0
2. $y \cdot 0 = 0$  by A1
3. $x = x + y \cdot 0$  by 1, 2 and A2
Lemma 2 (we needed this in the verification)

Informally we prove that:

\[(x = r + y \cdot q) \land (y \leq r) \rightarrow (x = (1+q) \cdot y + (r-y))\]

1. From the consequent: \(y + y \cdot q + r - y = y \cdot q + r\) \hspace{1cm} \text{simple arithmetic}
2. So \(y \cdot q + r = y \cdot (1+q) + (r-y)\) \hspace{1cm} \text{reverse 1.}
3. So \((x = r + y \cdot q) \rightarrow (x = (r-y) + y \cdot (1+q))\) \hspace{1cm} \text{if P=Q then P\Rightarrow Q}
4. Now add \(y \leq r\) to the antecedent \hspace{1cm} \text{antecedent strengthening of 3.}
5. QED

This is trivial, but formal is formal
method quotient(x : int, y : int) returns (q : int, r : int)
requires x >= 0 && y > 0; // this is the precondition
ensures q * y + r == x && r >= 0 && r < y; // postcondition
{
    q := 0;
    r := x;
    while (r >= y)
        invariant q * y + r == x && r >= 0 && y > 0;
        {
            r := r - y;
            q := q + 1;
        }
}

This program compiles and verifies okay in Dafny
Another example: $\sum 0..n$

The following program computes $0+1+2+\ldots+n = \sum 0..n$

\[
\begin{align*}
\{n \geq 0\} & \\
k & := 1; \quad \{P\} \text{ precondition} \\
s & := 0; \\
\{0 \leq k \leq n+1 \land s = \sum 0..k-1\} & \\
\text{while (} k \leq n \text{) } \{I\} \text{ the invariant} \\
\{s = \sum 0..n}\end{align*}
\]
Verifying the program

If the answers to all the questions:

- **Starting**: Is \{P\} S_1 \{I\} true?
- **Maintaining**: Is \{I \land B\} S_2 \{I\} true?
- **Finishing**: Does \(I \land \neg B \rightarrow Q\)?

is yes, then the code is functionally correct (with respect to the spec)

If we can prove termination: i.e. prove that \(B\) must eventually be false, then the program is totally correct
Does the program **start** properly?

\[
\text{Is } \{P\} \ k:=1; \ s:=0 \ \{I\} \text{ true?}
\]

\[
\{n \geq 0\} \ k:=1; \ s:=0 \ \{0 \leq k \leq n+1 \land s=\sum 0..k-1\}
\]

**Execute the assignment statements on** \{I\}

\[
\{0 \leq 1 \leq n+1 \land 0=\sum 0..1-1\}
\]

\[
\{0 \leq n \land 0=0\}
\]

**Hence true.**

I’ve been lazy here: I should have computed the state after EACH assignment statement using rule D0, and then used rule D2 to combine them.
Does the program maintain the invariant?

- Is \( \{I \land B\} S_2 \{I\} \) true?

\[
\{0 \leq k \leq n+1 \land s = \sum 0..k-1 \land k \leq n\} \quad s := s+k; \quad k := k+1 \quad \{0 \leq k \leq n+1 \land s = \sum 0..k-1\}
\]

- Simplifying the precondition: \( \{0 \leq k \leq n \land s = \sum 0..k-1\} \)
- Starting with the postcondition, we work right to left:
  - Substituting \( k := k+1 \) we get \( \{0 \leq k+1 \leq n+1 \land s = \sum 0..k+1-1\} \)
    - this simplifies to \( \{-1 \leq k \leq n \land s = \sum 0..k\} \)
  - Now substituting \( s := s+k \) we get \( \{-1 \leq k \leq n \land s+k = \sum 0..k\} \)
    - this simplifies to \( \{-1 \leq k \leq n \land s = \sum 0..k-1\} \)
    - which we can strengthen to the above precondition

- So, the triple is true, and the invariant is maintained

*Again, I've been lazy: I should have said when I used rule D0 and D2*
Does the program **finish** properly?

Does \( I \land \neg B \rightarrow Q \)?

\[(0 \leq k \leq n+1 \land s=\sum 0..k-1) \land \neg k \leq n \rightarrow s=\sum 0..n\]

Antecedent (ie. the LHS) says \( k = n+1 \). Substituting into \( s=\sum 0..k-1 \) generates

\[s = \sum 0..n+1-1 \rightarrow s = \sum 0..n\]

Hence true.

Start, maintenance and finish are true, hence program is verified.
1. Q&A on inferences

Here is D1 (precondition strengthening) again

\[ S \rightarrow P \quad \{P\} \quad Q \quad \{R\} \]
\[ \{S\} \quad Q \quad \{R\} \]

**Q:** Is inference \( \{0 < x \land x < 10\} \quad x := x \times x \quad \{x < 100\} \) correct?

**A:** Yes, because \( (x = 9) \rightarrow ((0 < x) \land (x < 10)) \) and use D1

- In other words, \( x = 9 \) strengthens \( 0 < x < 10 \) so can be used as a precondition
2. Q&A

Here is D1 (precondition strengthening) again

\[ S \rightarrow P \quad \{P\} \ Q \ \{R\} \]

\[ \{S\} \ Q \ \{R\} \]

- **Q:** Is the inference \( \{x^2x<100\} \ x := x^2 \ \{x<100\} \) correct?

  \[ \{0<x \land x<10 \} \ x := x^2 \ \{x<100\} \]

- **A:** Yes, because \( 0<x \land x<10 \rightarrow x^2x<100 \) and applying D1
Q: Is inference \[
\{x=6\} \quad x := x^x \quad \{x<100\}\]
correct?
\[
\{0<x \land x<10\} \quad x := x^x \quad \{x<100\}\]

A: No, because \(0<x \land x<10 \Rightarrow x=6\), obviously
Here is D1 (postcondition weakening) again

\{P\} Q \{R\} \quad R \rightarrow S

\{P\} Q \{S\}

- **Q:** Is the inference \{x+y=5\} \quad x:=x+y \quad \{x=5\} \quad correct? \{x+y=5\} \quad x:=x+y \quad \{x<10\}

- **A:** Yes, because \( x=5 \rightarrow x<10 \) and applying D1
Everything is physical. Everything has a soul. Therefore some physical things have souls.

Q: Express this as a predicate formula and prove its correctness.

∀xP(x), ∀xS(x) ⊢ ∃x(P(x) ∧ S(x)) where P and S mean …

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>∀xP(x)</td>
<td>Premise</td>
</tr>
<tr>
<td>2</td>
<td>∀xS(x)</td>
<td>Premise</td>
</tr>
<tr>
<td>3</td>
<td>P(a)</td>
<td>UI (1)</td>
</tr>
<tr>
<td>4</td>
<td>S(a)</td>
<td>UI (2)</td>
</tr>
<tr>
<td>5</td>
<td>P(a) ∧ S(a)</td>
<td>Conjunction (3,4)</td>
</tr>
<tr>
<td>6</td>
<td>∃x(P(x) ∧ S(x))</td>
<td>EG (5)</td>
</tr>
</tbody>
</table>
6. Q&A inference proofs (using contradiction)

Albert is an Australian and a lecturer. Albert doesn’t play rugby. Therefore not all Australians play rugby.

Q: Express this as a predicate formula and prove its correctness.

\[ A(a) \land L(a), \neg R(a) \vdash \neg \forall x(A(x) \rightarrow R(x)) \]

where A, L and R mean …

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( A(a) \land L(a) )</td>
<td>Premise, Albert is Aust, Albert is lecturer</td>
</tr>
<tr>
<td>2</td>
<td>( \neg R(a) )</td>
<td>Premise, Albert doesn’t play rugby</td>
</tr>
<tr>
<td>3</td>
<td>( \forall x(A(x) \rightarrow R(x)) )</td>
<td>Assume conclusion incorrect</td>
</tr>
<tr>
<td>4</td>
<td>( A(a) \rightarrow R(a) )</td>
<td>UI(3) with constant ( a )</td>
</tr>
<tr>
<td>5</td>
<td>( A(a) )</td>
<td>Conjunction Elim. (1)</td>
</tr>
<tr>
<td>6</td>
<td>( R(a) )</td>
<td>Modus Ponens (4,5)</td>
</tr>
<tr>
<td>7</td>
<td>( \neg R(a) \land R(a) )</td>
<td>Conjunction (2,6)</td>
</tr>
<tr>
<td>8</td>
<td>( \neg \forall x(A(x) \rightarrow R(x)) )</td>
<td>Contradiction (7), assump(3) incorrect</td>
</tr>
</tbody>
</table>