Proving programs correct

- **Primary characterization**
  - Given a specification (in a formal logic) and an implementation (in a programming language), prove that the implementation satisfies the specification.

- **Alternative characterization**
  - Given the specification, derive (construct) a program that satisfies the specification.

Examples

(Note: \(X\), \(Y\) are constants)

- True Hoare triples
  - \{true\} \(y := x \times x\) \{\(y \geq 0\)\}
  - \{\(x > 0\)\} \(x := x + 1\) \{\(x > 1\)\}
  - \{\(x > 1\)\} \(x := x + 1\) \{\(x > 0\)\}
  - \{\(x = X \land y = Y\)\}
    - \(t := x\) \(\land x := y\) \(\land y := t\);
    - \{\(x = Y \land y = X\)\}

- False Hoare triples
  - \{true\} \(y := x \times x\) \{\(y < 0\)\}
  - \{\(x = X \land y = Y\)\}
    - \(t := x\) \(\land x := y\) \(\land y := t\);
    - \{\(x = Y \land t = Y\)\}

Meaning of assignment

- We must precisely define the meaning of the assignment operator used in the programs.
- Back-substitution is the basic approach.

Consider the triple \(\{P?\} x : = \exp \{Q(x)\}\)
- \(P?\) is an unknown precondition.
- \(Q\) is the postcondition that may be parameterized in terms of the program variable \(x\).
- For \(Q(x)\) to be true requires that \(P?\) be equal to \(Q(exp)\) as a precondition.

Another example: true or false?

\{\(x < 0\)\}
  - if \(x > 0\) then
    - \(x := x + 1\)
  - else
    - \(x := -x\)
  fi

\{\(x > 0\)\}
Examples

\[ \{ P? \} x := x + 1 \{ x > 1 \} \]
- \( Q(x) = x > 1 \)
- So \( P? \land Q(x+1) \land x > 1 = x > 0 \)

\[ \{ P? \} y := x \times x \{ y \geq 0 \} \]
- \( Q(y) = y = 0 \)
- So \( P? \land Q(x \times x) = x \times x = 0 \land \text{true} \)

This is technically handled by the "proof rule"
- \( \{ B[a/x] \} x := a \{ B \} \)
- Where \( B[a/x] \) represents the predicate \( B \) with all free occurrences of \( x \) replaced by \( a \)

Meaning of conditionals

- There are also proof rules for \( \{ P \} \text{if } C \text{ then } S1 \text{ else } S2 \{ Q \} \)
- If we can prove
  - \( \{ P \land C \} S1 \{ Q \} \)
  - \( \{ P \land \neg C \} S2 \{ Q \} \)
- Then we have proven the triple

Example

\[ \{ x \neq 0 \} \]
if \( x > 0 \) then \( x := x + 1 \) else \( x := -x \)
\[ \{ x > 0 \} \]

Proving programs

- The basic approach to proving a program correct using Hoare triples is to
  - start with the precondition \( P \), the postcondition \( Q \), and the program \( S \)
  - \( S \) usually consists of a sequence of statements
  - One then introduces additional intermediate assertions between the statements
  - \( \{ P \} S1; S2; S3; S4 \{ Q \} \)
  - \( \{ P \} S1; S2; S3; S4 \{ Q \} \)
  - Then prove each triple (they are associative).

Some additional proof rules

- What is the semantics of the programming language construct \( ; \)
  - \( (\{ P1 \} S1; \{ P2 \} S2; \{ P3 \} \) implies \( \{ P1 \} S1; \{ P2 \} S2; \{ P3 \} \)
- Also, if \( P0 \) implies \( P1 \) in addition, then we also can prove \( \{ P0 \} S1; \{ P2 \} S2; \{ P3 \} \)

Loops

- Loops are the biggest challenge in proving correctness, since we can not write simple proof rules because the number of iterations through a loop is in general unbounded
- One issue is proving that the loop terminates; this is usually done separately from the proof about the program's computation
- We have to introduce an added assertion, called a \textit{loop invariant}; it is not generally possible to compute these, so they have to be chosen carefully to allow the proof to go through
Termination

- Weak (or partial) correctness: the proof of \( P \implies Q \) assumes that \( S \) terminates.
- Strong (or total) correctness: Termination of \( S \) is proven.
- Example: weakly correct but not known to be strongly correct.

\[
\begin{align*}
\{ x > 0 \} & \quad y := f(x); \\
& \quad \text{function } f(z : \text{int}): \text{int} \begin{cases}
0 & \text{if } z = 1 \\
otherwise & \text{if even}(z) \text{ return } f(z/2); \\
\text{else return } f(3*z+1); \\
end
\end{cases}
\end{align*}
\]

\( \{ y = 1 \} \)

Proving loops correct

- \( \{ P \} \text{ while } C \text{ do } S \{ Q \} \)
- We need to find a loop invariant \( I \) and prove the following proof obligations:
  - \( P \implies I \) \quad // I true when the loop starts
  - \( \{ I \text{ and } C \} \{ I \} \) \quad // I remains true each iteration
  - \( \{ I \text{ and not } C \} \implies Q \) \quad // if loop terminates, Q holds

Simple example (Ghezzi)

- \( \{ x >= 0 \} \text{ while } x>0 \text{ do } x := x + 1 \{ x = 0 \} \)
- \( I = (x >= 0) \)
- \( P \implies I \) [trivial]
- \( \{ x >= 0 \} x := x + 1 \{ x >= 0 \} \) [trivial]
- \( ((x >= 0) \text{ and } (x <= 0)) \implies (x = 0) \) [trivial, QED]
- Question: does this example make sense?

Another example: divide \( i \) by \( j \), quotient in \( \text{div} \), remainder in \( t \)

\[
\begin{align*}
\{ i > 0 \text{ and } j > 0 \} & \quad t := i; \\
& \quad \text{div} := 0; \\
& \quad \text{while } t >= j \text{ do } \\
& \quad \quad \text{div} := \text{div} + 1; \\
& \quad \quad t := t - j; \\
& \quad end \\
& \quad \{ i = \text{div} \times j + t \text{ and } 0 <= t < j \}
\end{align*}
\]

In small groups, prove this correct, including explicitly identifying the loop invariant.

Termination

- It is relatively rare for termination to be the central issue or problem with a program.
- Also, demonstrating non-termination is equally important for classes of programs, such as operating systems, avionics control systems, etc.

- Termination is generally proved by the use of well-founded sets:
  - A set is well-founded if it is partially ordered and every non-empty subset has a minimal element.
- In essence, one wants to show monotonic progress on every iteration towards a fixed bound.
- In the previous example, \( t \) becomes closer to \( j - 1 \) on every iteration.
- One can ignore the other computations in the loop.
Next lecture (1/14)

• Weakest precondition formulation
• Proof of correctness of abstract data types