Program analysis: from proving correctness to proving incorrectness

Roberto Bruni, Roberta Gori
(University of Pisa)
Lecture #02

BISS 2024
March 11-15, 2024
Before we start...
Please answer questions

There are the 3 possible answers to the verification problem
“does my program $c$ satisfy the specification $S$ ?”

- yes
- no
- don’t know

please pick one option whenever we ask questions in these classes
This is a course on (in)correctness

Inevitably, there will be errors in the slides, help us to correct them

Can you find the mistake?

1 2 3 4 5 6 7 8 9
Program correctness: a long standing problem
Origins? Turing’s assertions

“How can one check a routine in the sense of making sure that it is right?”

Alan Turing (1949)
Checking factorial

- a dashed letter indicates the value at the end of the process represented by the box
- an undashed letter represents the initial value of a quantity
- TEST is test for zero
- \[ \_ \] denotes factorial
- at the end (D) \( v = n! \)
PROOF OF ALGORITHMS BY GENERAL SNAPSHOTS

PETER NAUR

Abstract.
A constructive approach to the question of proofs of algorithms is to consider
collections of data and assume that the procedure has a certain type of
characteristics. It is shown by an elementary example how this possi-
bility may be used to prove the correctness of an algorithm written in ALGOL 60.

Introduction.
It is a deplorable consequence of the lack of influence of mathematical
textbooks on the way in which computer programming is currently being
practiced, that the regular use of systematic proof procedures, or even the
realization that such proof procedures exist, is unknown to the large
majority of programmers. Undoubtedly, this fact accounts for at least a
large share of the unreliability and the attendant lack of over-all effec-
tiveness of programs as they are used to-day.

Historically this state of affairs is easily explained. Large scale com-
puter programming started so recently that all of its practitioners are,
in fact, amateurs. At the same time the modern computers are so effec-
tive that they offer advantages in use even when their powers are largely
wasted. The stress has been on open sources, and, allegedly, more power-
ful systems, in spite of the fact that the available programmer com-
petence often is unable to cope with their complexities.

However, a reaction is bound to come. We cannot indefinitely con-
tinue to build on sand. When this is realized there will be an increased
interest in the less glamorous, but more solid, basic principles. This will
go in parallel with the introduction of these principles in the elementary
school curricula. One subject which will then come up for attention is
that of proving the correctness of algorithms. The purpose of the present
article is to show in an elementary way that this subject not only exists,
but is ripe to be used in practical. The illustrations are phrased in ALGOL
60, but the technique may be used with any programming language.
Floyd’s interpretations (1967)

"an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken"

ASSIGNING MEANINGS TO PROGRAMS

Introduction. This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion of an interpretation of a program; that is, an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken. To prevent an interpretation from being chosen arbitrarily, a condition is imposed on each command of the program. This condition guarantees that whenever a command is reached by way of a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. Then by induction on the number of commands executed, one sees that if a program is entered by a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. By this means, we may prove certain properties of programs, particularly properties of the form: “If the initial values of the program variables satisfy the relation $R_0$, the final values on completion will satisfy the relation $R_f$."

Proofs of termination are dealt with by showing that each step of a program decreases some entity which cannot decrease indefinitely. These modes of proof of correctness and termination are not original; they are based on ideas of Perlis and Gorn, and may have made their earliest appearance in an unpublished paper by Gorn. The establishment of formal standards for proofs about programs in languages which admit assignments, transfer of control, etc., and the proposal that the semantics of a programming language may be defined independently of all processors for that language, by establishing standards of rigor for proofs about

1This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SAD-146).
Floyd's examples

**Figure 1.** Flowchart of program to compute $S = \sum_{j=1}^{n} a_j$ with $n \geq 0$

**Figure 5.** Algorithm to compute quotient $Q$ and remainder $R$ of $X + Y$, for integers $X \geq 0$, $Y > 0$

We now demonstrate Floyd's examples and show their correctness by construction. We consider the following two statements:

1. $n \in J^+$ (where $J^+$ is the set of positive integers)
2. $i \in J^+ \land i = 1$

**Correctness:**

- $n \in J^+$
- $i \in J^+ \land i = 1$

*weakening*

**Statement 1:** $n \in J^+$

**Statement 2:** $i \in J^+ \land i = 1$

*weakening*

**Figure 1** shows the flowchart of a program to compute $S = \sum_{j=1}^{n} a_j$ with $n \geq 0$.

**Figure 5** illustrates the algorithm to compute the quotient $Q$ and remainder $R$ of $X + Y$, for integers $X \geq 0$, $Y > 0$. The flowchart demonstrates how the program works and ensures termination by proving the correctness of the algorithm.
Turing's proof in Floyd's notation
Hoare Logic

“The purpose of this study is to provide a logical basis for proofs of the properties of a program.”

find the quotient $q$ and the remainder $r$

obtained on dividing $x$ by $y$

$((r := x; \quad q := 0); \quad \text{while} \quad y \leq r \quad \text{do} \quad (r := r - y; \quad q := 1 + q))$

$\neg y \leq r \land x = r + y \times q$

---

**TABLE III**

<table>
<thead>
<tr>
<th>Line number</th>
<th>Formal proof</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{true} \supset x = x + y \times 0$</td>
<td>Lemma 1</td>
</tr>
<tr>
<td>2</td>
<td>$x = x + y \times 0 [r := x] x = r + y \times 0$</td>
<td>D0</td>
</tr>
<tr>
<td>3</td>
<td>$x = r + y \times 0 [q := 0] x = r + y \times q$</td>
<td>D0</td>
</tr>
<tr>
<td>4</td>
<td>$\text{true} [r := x] x = r + y \times 0$</td>
<td>D1 (1, 2)</td>
</tr>
<tr>
<td>5</td>
<td>$\text{true} [r := x; \quad q := 0] x = r + y \times q$</td>
<td>D2 (4, 3)</td>
</tr>
<tr>
<td>6</td>
<td>$x = r + y \times q \land y \leq r \supset x = (r - y) + y \times (1 + q)$</td>
<td>Lemma 2</td>
</tr>
<tr>
<td>7</td>
<td>$x = (r - y) + y \times (1 + q) [r := r - y] x = r + y \times (1 + q)$</td>
<td>D0</td>
</tr>
<tr>
<td>8</td>
<td>$x = r + y \times (1 + q) [q := 1 + q] x = r + y \times q$</td>
<td>D0</td>
</tr>
<tr>
<td>9</td>
<td>$x = (r - y) + y \times (1 + q) [r := r - y; \quad q := 1 + q] x = r + y \times q$</td>
<td>D1 (7, 8)</td>
</tr>
<tr>
<td>10</td>
<td>$x = r + y \times q \land y \leq r [r := r - y; \quad q := 1 + q] x = r + y \times q$</td>
<td>D2 (6, 9)</td>
</tr>
<tr>
<td>11</td>
<td>$x = r + y \times q \quad \text{while} \quad y \leq r \quad \text{do} \quad (r := r - y; \quad q := 1 + q)) \neg y \leq r \land x = r + y \times q$</td>
<td>D3 (10)</td>
</tr>
<tr>
<td>12</td>
<td>$\text{true} [(r := x; \quad q := 0); \quad \text{while} \quad y \leq r \quad \text{do} \quad (r := r - y; \quad q := 1 + q)) \neg y \leq r \land x = \neg y \leq r \land x = r + y \times q$</td>
<td>D2 (5, 11)</td>
</tr>
</tbody>
</table>

**Notes**

1. The left hand column is used to number the lines, and the right hand column to justify each line, by appealing to an axiom, a lemma or a rule of inference applied to one or two previous lines, indicated in brackets. Neither of these columns is part of the formal proof. For example, line 2 is an instance of the axiom of assignment (D0); line 12 is obtained from lines 5 and 11 by application of the rule of composition (D2).
2. Lemma 1 may be proved from axioms A7 and A8.
3. Lemma 2 follows directly from the theorem proved in Sec. 2.
Preliminaries
A simple imperative language

command

\[ c ::= \]
\[ \begin{align*}
& x ::= a \\
& \text{skip} \\
& c_1 ; c_2 \\
& \text{if } b \text{ then } c_1 \text{ else } c_2 \\
& \text{while } b \text{ do } c
\end{align*} \]

integer variable

\[ \begin{align*}
& a ::= n | x | a_1 + a_2 | \ldots
\end{align*} \]

arithmetic expression

\[ \begin{align*}
& b ::= a_1 \leq a_2 | b_1 \land b_2 | \ldots
\end{align*} \]

Boolean expression

\[ \begin{align*}
& b ::= a_1 \leq a_2 | b_1 \land b_2 | \ldots
\end{align*} \]
Concrete domain

\[ \sigma : X \rightarrow \mathbb{Z} \]

\[ \Sigma \triangleq \{ \sigma : X \rightarrow \mathbb{Z} \} \]

\[ \mathcal{P}(\Sigma) \triangleq \{ P \mid P \subseteq \Sigma \} \]
Notation

\( [x \mapsto 1, y \mapsto 2] \)
the state where \( x \) holds 1, \( y \) holds 2 and any other variable holds 0

\( \sigma[x \mapsto n] \)
the state where \( x \) holds \( n \) and any other variable \( y \) holds \( \sigma(y) \)

\( (x = 1, y = 2) \)
the set of all states where \( x \) holds 1 and \( y \) holds 2
Assertion language

\[ P ::= \text{true} \mid \text{false} \mid a_1 < a_2 \mid a_1 = a_2 \mid \ldots \]
\[ \mid \neg P \mid P_1 \land P_2 \mid \exists x. P \mid \ldots \]
Notation

\[ \sigma \models P \quad \text{or also} \quad \sigma \in P \]

the state \( \sigma \) satisfies the property \( P \)

\[ P \Rightarrow Q \quad \text{or also} \quad P \subseteq Q \quad \text{or also} \quad P \leq Q \]

any state that satisfies \( P \) satisfies \( Q \)
Collecting semantics

\[\llbracket c \rrbracket : \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)\]

\[\llbracket c \rrbracket P\]

is the set of all and only states reachable from some state in \(P\) after executing \(c\)

\[\llbracket c \rrbracket \sigma\] as a shorthand for \(\llbracket c \rrbracket \{\sigma\}\)

additive: \(\llbracket c \rrbracket (P_1 \cup P_2) = (\llbracket c \rrbracket P_1) \cup (\llbracket c \rrbracket P_2)\)
Collecting semantics

\[ \llbracket a \rrbracket : \Sigma \rightarrow \mathbb{Z} \]

\[ \llbracket a \rrbracket_\sigma \]
evaluates the arithmetic expression \( a \) in the current state \( \sigma \)

e.g.
\[ \llbracket x + 1 \rrbracket[x \mapsto 1, y \mapsto 2] = 2 \]
Collecting semantics

$$[[b]] : \wp(\Sigma) \rightarrow \wp(\Sigma)$$

$$[[b]]P \text{ (intuitively } b \land P\text{)}$$

is the set of all and only states in $$P$$ that satisfy the condition $$b$$

e.g.

$$[[x < y]] \{[[x \mapsto 1,y \mapsto 2],[x \mapsto 2,y \mapsto 1]]\} = \{[[x \mapsto 1,y \mapsto 2]]\}$$

$$[[x < y]] [[x \mapsto 2,y \mapsto 1]] = \emptyset$$
Collecting semantics: atomic commands

\[[\text{skip}]\]P \triangleq P

\[[x := a]P \triangleq \{ \sigma[x \mapsto [a]\sigma] \mid \sigma \in P \}\]

e.g.

\[[r := x][x \mapsto 5, y \mapsto 2] = \{ [x \mapsto 5, y \mapsto 2, r \mapsto 5] \} \]
Collecting semantics: sequence

\[[c_1; c_2]P \triangleq [c_2](\llbracket c_1 \rrbracket P)\]

e.g.
\[[r := x; \ q := 0][x \mapsto 5, \ y \mapsto 2] = \{[x \mapsto 5, \ y \mapsto 2, \ r \mapsto 5]\}\]
Collecting semantics: conditionals

\[
\llbracket \text{if } b \text{ then } c_1 \text{ else } c_2 \rrbracket P \triangleq \llbracket c_1 \rrbracket (\llbracket b \rrbracket P) \cup \llbracket c_2 \rrbracket (\llbracket \neg b \rrbracket P)
\]

e.g.
\[
\llbracket \text{if } x \geq 0 \text{ then skip else } x := -x \rrbracket \{\llbracket x \mapsto -1 \rrbracket, \llbracket x \mapsto 1 \rrbracket\} \triangleq \llbracket \text{skip} \rrbracket [x \mapsto 1] \cup \llbracket x := -x \rrbracket [x \mapsto -1] \triangleq \{\llbracket x \mapsto 1 \rrbracket\}
\]
Collecting semantics: loops

\[ \text{while } b \text{ do } c \] P \triangleq [\neg b] \text{ fix } (\lambda S. P \cup [c][b]S) \]
Kleene’s fixpoint theorem

Th.
Let \( f : \mathcal{C} \to \mathcal{C} \) be a continuous function on a CPO with bottom.
Then \( \text{fix } f = \bigcup_{k=0}^{\infty} f^k(\perp) \).

e.g.
\[
f \triangleq (\lambda S . P \cup \llbracket c \rrbracket \llbracket b \rrbracket S) : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)
\]
\[
f^0(\emptyset) = \emptyset, \quad f^1(\emptyset) = P, \quad f^2(\emptyset) = P \cup \llbracket c \rrbracket \llbracket b \rrbracket P, \ldots
\]
Collecting semantics: loops

\[[\text{while } b \text{ do } c]\]P ≜ [[\neg b]] \text{fix} (\lambda S. P \cup [[c]][[b]]S)

= [[\neg b]] \bigcup_{k=0}^{\infty} ( [[c]][[b]] )^k P

= \bigcup_{k=0}^{\infty} [[\neg b]]([[c]][[b]] )^k P
Collecting semantics: loops

\[
[[\text{while } b \text{ do } c]]P \triangleq \bigcup_{k=0}^{\infty} [[\neg b]]([[c][[b]])^kP
\]

\hspace{1em} e.g.

\[
[[w]]_{\sigma} = \bigcup_{k=0}^{\infty} [[y > r]]([[\ldots][[y \leq r]])^k\sigma
\]

\[
= [[y > r]]_{\sigma} \cup_{k=1}^{\infty} [[y > r]]([[\ldots][[y \leq r]])^k\sigma
\]

\[
= [[y > r]](\ldots)_{\sigma} \cup_{k=2}^{\infty} [[y > r]](\ldots)^k\sigma
\]

\[
\sigma \triangleq [x \mapsto 5, y \mapsto 2, r \mapsto 5]
\]

\[
[[w]]_{\sigma} = [[y > r]]_{[x \mapsto 5, y \mapsto 2, r \mapsto 5, q \mapsto 1]} \cup_{k=2}^{\infty} [[y > r]](\ldots)^k\sigma
\]

\[
= [[y > r]]_{[x \mapsto 5, y \mapsto 2, r \mapsto 1, q \mapsto 2]} \cup_{k=3}^{\infty} [[y > r]](\ldots)^k\sigma
\]

\[
= \{ [[x \mapsto 5, y \mapsto 2, r \mapsto 1, q \mapsto 2]] \}
Inference rules

If all premises hold, then the conclusion holds.

Premises: $\phi_1, \phi_2, \ldots, \phi_n$

Conclusion: $\phi$

Axioms:

1. $\text{pos}(1)$
2. $\text{pos}(x) \text{ pos}(y) \Rightarrow \text{pos}(x + y)$
Proof systems

a set of inference rules

\[
\begin{align*}
&\text{[base]} \quad \frac{}{\text{pos}(1)} \\
&\frac{\text{pos}(x) \quad \text{pos}(y)}{\text{pos}(x + y)} \quad \text{[sum]}
\end{align*}
\]
Hoare Logic
(HL)
Hoare’s triples

\[ P \{ c \} Q \]

\[ \{ P \} c \{ Q \} \]

when the precondition is met, executing the command establishes the postcondition:

\[ \llbracket c \rrbracket P \subseteq Q \]

Since then, the original paper has shown that this can include non-reachable states over approximation.
An obvious axiom

\[
\{ P \} \text{ skip } \{ P \}
\]

\[
\{ x > 0 \} \text{ skip } \{ x > 0 \}
\]
Floyd’s axiom for assignment

\[\{P\} x := a \{ \exists x’. P[x’/x] \land x = a[x’/x]\}\]

\{\text{true}\} r := x \{ \exists r’. \text{true}, r = x\} \equiv \{r = x\}

\{x = r + qy\} r := r - y \{ \exists r’. x = r’ + qy, r = r’ - y\}
\equiv \{ \exists r’. x = r + y + qy, r’ = r + y\}
\equiv \{x = r + (q + 1)y\}
Hoare’s axiom for assignment

$$\{Q[a/x]\} \ x := a \ \{Q\}$$

$$\{\text{true}\} \equiv \{x = x + 0y\} \ r := x \ \{x = r + 0y\}$$

$$\{x = r\} \equiv \{x = r + 0y\} \ q := 0 \ \{x = r + qy\}$$

$$\{x = r + qy\} \equiv$$

$$\{x = r - y + (q + 1)y\} \ r := r - y\{x = r + (q + 1)y\}$$
An observation

\[ \{ P \} \ x := a \ \{ \ \exists \ x'. \ P[x'/x] \land x = a[x'/x] \} \]

[Forward oriented: Floyd's]

\[ \{ Q[a/x] \} \ x := a \ \{ Q \} \]

[Backward oriented: Hoare's]
Composition rule

\[
\begin{align*}
\{P\} c_1 \{R\} \{R\} c_2 \{Q\} \\
\setminus \\
\{P\} c_1; c_2 \{Q\}
\end{align*}
\]

\[
\begin{align*}
\{x = r + qy\} & \quad r := r - y \quad \{x = r + (q + 1)y\} \\
\{x = r + (q + 1)y\} & \quad q := q + 1 \quad \{x = r + qy\}
\end{align*}
\]

\[
\begin{align*}
\{x = r + qy\} & \quad r := r - y; q := q + 1 \quad \{x = r + qy\}
\end{align*}
\]
Inlining assertions

\[
\begin{align*}
\{x = r + qy\} & \quad r := r - y \quad \{x = r + (q + 1)y\} \\
\{x = r + (q + 1)y\} & \quad q := q + 1 \quad \{x = r + qy\}
\end{align*}
\]

\[
\begin{align*}
\{x = r + qy\} & \quad r := r - y; \; q := q + 1 \quad \{x = r + qy\}
\end{align*}
\]
While rule

\[
\begin{align*}
\{P \land b\} & \quad c \quad \{P\} \\
\{P\} \quad \text{while } b \quad \text{do} \quad c \quad \{P \land \neg b\}
\end{align*}
\]

\{x \geq 0\}
while $x > 0$ do
\quad \{x \geq 0 \land x > 0\} \equiv \{x > 0\} \equiv \{x \geq 1\} \equiv \{x - 1 \geq 0\}
x := x - 1;
\quad \{x \geq 0\}
\{x \geq 0 \land x \leq 0\} \equiv \{x = 0\}
Invariants as pre-fixed points

\([I \land b] \ c \ {I}\)  
\{I\} while b do c \({I \land \neg b}\)

\{I \land b\} c \ {I}\) means that \([c](I \land b) \subseteq I\)
i.e. that \([c][b]I \subseteq I\)
i.e. that \(I\) is a pre-fixed point of \([c] \circ [b]\)

whenever \(P \subseteq I\), by definition of fix:
\([[\text{while } b \text{ do } c]]P \triangleq [[\neg b]] \text{fix}(\lambda S. P \cup [[c][b]]S) \subseteq [[\neg b]]I\)
Consequence rule

\[ P \Rightarrow P' \quad \{P'\} \ r \ \{Q'\} \quad Q' \Rightarrow Q \]
\[ \{P\} \ r \ \{Q\} \]

\[
\begin{align*}
\{x \geq 0 \land y > 0\} & \Rightarrow \\
\{-y < 0 \land x \geq 0 \land y \geq 0\} & \Rightarrow \\
\{x - y < x \land x + y \geq 0\} & \\
n & := x - y; \\
\{n < x \land x + y \geq 0\} &
\end{align*}
\]
Hoare's proof

\{\text{true}\} \equiv \{x = x + 0y\}

\begin{align*}
r &:= x \\
\{x = r + 0y\}
\end{align*}

\begin{align*}
q &:= 0; \\
\{x = r + qy\}
\end{align*}

while \( y \leq r \) do

\begin{align*}
\{x = r + qy \land y \leq r\} &\Rightarrow \{x = (r - y) + (q + 1)y\} \\
r &:= r - y; \\
\{x = r + (q + 1)y\}
\end{align*}

\begin{align*}
q &:= q + 1 \\
\{x = r + qy\}
\end{align*}

\{x = r + qy \land y > r\}

\begin{align*}
\{P \land b\} &c \{P\} \\
\frac{\{P\} \text{while } b \text{ do } c \{P \land \neg b\}}{}
\end{align*}
Wait a moment...

\{true\} \equiv \{x = x + 0y\}

\begin{align*}
r &:= x \\
\{x = r + 0y\}
\end{align*}

\begin{align*}
q &:= 0; \\
\{x = r + qy\}
\end{align*}

while \( y \leq r \) do

\begin{align*}
\{x = r + qy \land y \leq r\} &\Rightarrow \{x = (r - y) + (q + 1)y\} \\
r &:= r - y; \\
\{x = r + (q + 1)y\}
\end{align*}

\begin{align*}
q &:= q + 1 \\
\{x = r + qy\}
\end{align*}

\begin{align*}
\{x = r + qy \land y > r\}
\end{align*}

\[[c][x \mapsto 5, y \mapsto -2] = \ldots = \emptyset\]
Wait a moment…

\{ \text{true} \} \equiv \{ x = x + 0y \}

\begin{align*}
r &:= x \\
\{ x = r + 0y \}
\end{align*}

\begin{align*}
q &:= 0; \\
\{ x = r + qy \}
\end{align*}

While \( z = 0 \) do

\begin{align*}
\{ x = r + qy \land z = 0 \} &\implies \{ x = (r - y) + (q + 1)y \} \\
r &:= r - y; \\
\{ x = r + (q + 1)y \}
\end{align*}

\begin{align*}
q &:= q + 1 \\
\{ x = r + qy \}
\end{align*}

\begin{align*}
\{ x = r + qy \land z \neq 0 \}
\end{align*}

\[[ c ] [ x \mapsto 5, y \mapsto 2, z \mapsto 0 ] = \ldots = \emptyset\]
No guarantee of termination

\[
\{ x \geq 0 \} \quad \{ x \geq 0 \land x > 0 \} \equiv \{ x + 1 \geq 0 \}
\]

while \( x > 0 \) do

\[
x := x + 1; \\
\{ x \geq 0 \}
\]

\[
\{ x \geq 0 \land x \leq 0 \} \equiv \{ x = 0 \}
\]

\([c]] [x \mapsto 5] = \ldots = \emptyset\]
False positive

\{ x = 1 \} \text{ while } x > 0 \text{ do } x := x + 1 \{ x = 0 \}

complete the proof below

\{ x = 1 \} \Rightarrow \{ \ ? \ \}

while \ x > 0 \text{ do }
\{ \ ? \land x > 0 \} 

x := x + 1;
\{ \ ? \ \}
\{ \ ? \land x \leq 0 \} \Rightarrow \{ x = 0 \}

not a possible output!
Partial vs total correctness

when the precondition is met, executing the command and establishes the postcondition

\[ \{P\} c \{Q\} \]

when the precondition is met, executing the command terminates and establishes the postcondition

total correctness = partial correctness + termination
Rule for total correctness

\[
\begin{align*}
\{ P \land b \} & \quad c \quad \{ P \} \\
\{ P \land b \land t = z \} & \quad c \quad \{ t < z \} \quad P \Rightarrow t \geq 0 \\
\{ P \} & \quad \text{while } b \text{ do } c \quad \{ P \land \neg b \}
\end{align*}
\]
Total correctness proof

\[
\{x \geq 0\} \text{ take } t \equiv x
\]

while \(x > 0\) do

\[
\{x \geq 0 \land x > 0\} \equiv \{x - 1 \geq 0\}
\]

\(x := x - 1;\)

\[
\{x \geq 0\}
\]

\[
\{x \geq 0 \land x \leq 0\} \equiv \{x = 0\}
\]

\[
P \Rightarrow t \geq 0
\]

\[
x \geq 0 \Rightarrow x \geq 0
\]

\[
\{P \land b \land t = z\} \land \{t < z\}
\]

\[
\{x \geq 0 \land x > 0 \land x = z\} \Rightarrow
\]

\[
\{x = z\} \Rightarrow
\]

\[
\{x < z + 1\} \equiv
\]

\[
\{x - 1 < z\} \Rightarrow
\]

\(x := x - 1\)

\[
\{x < z\}
\]
Total correctness proof

\[ \{ x \geq 0 \land y > 0 \} \equiv \{ x \geq 0 \land y > 0 \land x = x + 0y \} \]

\[ r := x \]

\[ \{ x \geq 0 \land y > 0 \land x = r + 0y \} \equiv \{ r \geq 0 \land y > 0 \land x = r + 0y \} \]

\[ q := 0; \]

\[ \{ r \geq 0 \land y > 0 \land x = r + qy \} \] take \( t \triangleq r \)

while \( y \leq r \) do

\[ \{ r \geq y > 0 \land x = r + qy \} \Rightarrow \{ r - y \geq 0 \land y > 0 \land x = r - y + (q + 1)y \} \]

\[ r := r - y; \]

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

\[ q := q + 1 \]

\[ \{ r \geq 0 \land y > 0 \land x = r + qy \} \]

\[ \{ y > r \geq 0 \land x = r + qy \} \]
Proof obligations

\[ P \Rightarrow t \geq 0 \]

\[(r \geq 0 \wedge y > 0 \wedge x = r + qy) \Rightarrow r \geq 0\]

\[
\{ P \wedge b \wedge t = z \} \ c \ \{ t < z \}
\]

\[
\{ r \geq y > 0 \wedge \cdots \wedge r = z \} \Rightarrow \{ r \geq 0 \wedge y > 0 \wedge \cdots \wedge r - y < z \}
\]

\[ r := r - y; \]

\[
\{ r \geq 0 \wedge y > 0 \wedge \cdots \wedge r < z \}
\]

\[ q := q + 1 \]

\[
\{ r \geq 0 \wedge y > 0 \wedge \cdots \wedge r < z \} \Rightarrow \{ r < z \}\]
If rule

\[
\begin{align*}
\{ P \land b \} & \ c_1 \ \{ Q \} \\
\{ P \land \neg b \} & \ c_2 \ \{ Q \}
\end{align*}
\]

\[
\{ P \} \text{ if } b \text{ then } c_1 \text{ else } c_2 \ \{ Q \}
\]

{true}
if \( x \geq 0 \) then
\[
\{ x \geq 0 \}
\]
skip
\[
\{ x \geq 0 \}
\]
else
\[
\{ \neg(x \geq 0) \} \equiv \{ -x > 0 \}
\]
\[
x := -x
\]
\[
\{ x > 0 \} \Rightarrow \{ x \geq 0 \}
\]
\[
\{ x \geq 0 \}
\]
Finding invariants

\[
\begin{align*}
\{ \text{true} \} \\
k &:= 1; \\
r &:= x; \\
\text{while } k > 0 \text{ do} \\
    \text{if } r > 100 \text{ then} \\
        r &:= r - 10; \\
        k &:= k - 1 \\
    \text{else} \\
        r &:= r + 11; \\
        k &:= k + 1 \\
\{ r = f(x) \}
\end{align*}
\]
McCarthy’s 91 function

\[
\begin{align*}
\{\text{true}\} \\
k &:= 1; \\
r &:= x; \\
\text{while } k > 0 \text{ do} \\
\quad \text{if } r > 100 \text{ then} \\
\qquad r &:= r - 10; \\
\qquad k &:= k - 1 \\
\quad \text{else} \\
\qquad r &:= r + 11; \\
\qquad k &:= k + 1 \\
\{r = f(x)\}
\end{align*}
\]

\[
f(x) \triangleq \begin{cases} 
91 & x \leq 100 \\
x - 10 & \text{otherwise}
\end{cases}
\]
Invariant for McCarthy’s 91 function?

\[
\{ \text{true} \} \\
k := 1; \\
r := x; \\
\{ ? \} \\
\text{while } k > 0 \text{ do} \\
\text{if } r > 100 \text{ then} \\
\quad r := r - 10; \\
\quad k := k - 1 \\
\text{else} \\
\quad r := r + 11; \\
\quad k := k + 1 \\
\{ ? \} \Rightarrow \{ r = f(x) \}
\]

\[
f(x) \triangleq \begin{cases} 
91 & x \leq 100 \\
x - 10 & \text{otherwise}
\end{cases}
\]
Invariant for McCarthy’s 91 function

\[
\{ \text{true} \}
\]

\[
k := 1;
\]

\[
r := x;
\]

\[
\{ k \geq 0 \land f^k(r) = f(x) \}
\]

while \( k > 0 \) do

if \( r > 100 \) then

\[
r := r - 10;
\]

\[
k := k - 1
\]

else

\[
r := r + 11;
\]

\[
k := k + 1
\]

\[
\{ k = 0 \land f^k(r) = f(x) \} \Rightarrow \{ r = f(x) \}
\]

\[
f(x) \triangleq \begin{cases} 
91 & x \leq 100 \\
 x - 10 & \text{otherwise}
\end{cases}
\]
Variant for McCarthy’s 91 function?

\[
\begin{align*}
\{ \text{true} \} \\
k &:= 1; \\
r &:= x; \\
\{ k \geq 0 \land f^k(r) = f(x) \} & \quad t \triangleq ? \\
\text{while } k > 0 \text{ do} \\
\quad \text{if } r > 100 \text{ then} \\
\qquad r &:= r - 10; \\
\qquad k &:= k - 1 \\
\quad \text{else} \\
\qquad r &:= r + 11; \\
\qquad k &:= k + 1 \\
\{ k = 0 \land f^k(r) = f(x) \} & \Rightarrow \{ r = f(x) \}
\end{align*}
\]

\[ f(x) \triangleq \begin{cases} 
91 & x \leq 100 \\
x - 10 & \text{otherwise}
\end{cases} \]
Finding invariants (McCarthy91)

\[
\{ \text{true} \} \\
\begin{align*}
k &:= 1; \\
r &:= x; \\
\{ k \geq 0 \land f^k(r) = f(x) \} & \quad t = (|101 - r + 10k|, k)
\end{align*}
\]

while \( k > 0 \) do

if \( r > 100 \) then

\[
\begin{align*}
r &:= r - 10; \\
k &:= k - 1
\end{align*}
\]

else

\[
\begin{align*}
r &:= r + 11; \\
k &:= k + 1
\end{align*}
\]

\[
\{ k = 0 \land f^k(r) = f(x) \} \Rightarrow \{ r = f(x) \}
\]

\[
f(x) \triangleq \left\{ \begin{array}{ll}
91 & x \leq 100 \\
x - 10 & \text{otherwise}
\end{array} \right.
\]
Validity, soundness, completeness
Validity

A HL triple \( \{P\} \ c \ \{Q\} \) is valid if \([c]P \subseteq Q\)

Is \( \{x > 0\} x := 10x \ \{x > 10\} \) valid? \(\times\)

Is \( \{x > 0, y > 0\} x := yx \ \{x \geq 0\} \) valid? \(\checkmark\)

Is \( \{\text{false}\} \ c \ \{Q\} \) valid? \(\checkmark\)

Is \( \{P\} \ c \ \{\text{true}\} \) valid? \(\checkmark\)
Correctness

**Th.** Any derivable HL triple is valid

**Proof.** By induction on the derivation tree
Incompleteness I

**Conjecture** Any valid HL triple is derivable

**Counterexample:**

\{true\} \( c \) \{false\} is valid when \( c \) diverges but halting problem is not r.e. while the set of derivable HL triples is r.e.
Incompleteness II

Conjecture  Any valid HL triple is derivable

Counterexample:

\{\text{true}\} \text{ skip } \{Q\} \text{ is valid when } Q \text{ is a tautology}

but Godel’s Incompleteness Theorem (1939) tells us that there is no effective proof system such that its theorems coincide with all valid arithmetic assertions.
Relative completeness: suppose we can consult an oracle to check if an assertion $P \Rightarrow P'$ is valid or not, then HL is complete.

In other words, we separate concerns about programs and reasoning about them from concerns to do with arithmetic and the incompleteness of any proof system for it.
Dijkstra’s weakest precondition

Given a command $c$ and a postcondition $Q$ a **weakest liberal precondition** is a predicate $P$ such that for any precondition $R$

$$\{ R \} \ c \ {\{ Q \}} \ \text{iff} \ R \ \Rightarrow \ P$$

i.e., $P$ is the least restrictive requirement that guarantees that $Q$ holds after executing $c$ (if it terminates)

Typically, it is denoted by $wlp(c, Q) \triangleq \{ \sigma \in \Sigma \mid \llbracket c \rrbracket \{ \sigma \} \subseteq Q \}$
Adjoints

\[ P \Rightarrow \text{wlp}(c, Q) \]

iff

\[ \llbracket c \rrbracket P \subseteq Q \]

iff

\[ \{ P \} c \{ Q \} \]
(Relative) Completeness

Th. If the logic language is *expressive enough*, then any valid HL triple can be derived.

Proof. Suppose \( \{P\} \, c \, \{Q\} \) is valid (with \( P \) and \( Q \) expressible). By structural induction on \( c \) we can build an assertion \( R \) that is equivalent to \( wlp(c, Q) \) and such that \( \{R\} \, c \, \{Q\} \) is derivable. By applying the consequence rule we derive \( \{P\} \, c \, \{Q\} \).
Weakest liberal preconditions

\( wlp(\text{skip}, Q) \triangleq Q \)

\( wlp(x := a, Q) \triangleq Q[x \mapsto a] \)

\( wlp(c_1; c_2, Q) \triangleq wlp(c_1, wlp(c_2, Q)) \)

\( wlp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) \triangleq (b \Rightarrow wlp(c_1, Q)) \land (\neg b \Rightarrow wlp(c_2, Q)) \)

\( wlp(\text{while } b \text{ do } c, Q) \triangleq \text{more complicated… but possible} \)
Adding nondeterminism
Regular commands

\[ r ::= e \]

\[ \mid r_1 ; r_2 \]

\[ \mid r_1 + r_2 \]

\[ \mid r^* \]

\[ e ::= \text{skip} \mid x ::= a \mid b? \mid \ldots \]

\[ [[b?]]P \triangleq [[b]]P \]

\[ [[r_1 + r_2]]P \triangleq [[r_1]]P \cup [[r_2]]P \]

\[ [[r^*]]P \triangleq \bigcup_{k=0}^{\infty} [[r]]^kP \]
Encoding while commands

if $b$ then $c_1$ else $c_2$ $\triangleq (b?; c_1) + (\neg b?; c_2)$

while $b$ do $c$ $\triangleq (b?; c)^*; \neg b?$
Minimal set of rules

\[ \{P\} e \{[[e]]P\} \quad \text{\{atom\}} \]

\[ \{P\} r_1 \{R\} \quad \{R\} r_2 \{Q\} \quad \text{\{seq\}} \]

\[ \forall i \in \{1,2\} \quad \{P\} r_i \{Q\} \quad \text{\{choice\}} \]

\[ \{P\} r_1 + r_2 \{Q\} \quad \text{\{choice\}} \]

\[ \{P\} r \{P\} \quad \text{\{iter\}} \]

\[ \{P\} r^* \{P\} \quad \text{\{iter\}} \]

\[ P \Rightarrow P' \quad \{P'\} r \{Q'\} \quad Q' \Rightarrow Q \quad \text{\{cons\}} \]

\[ \{P\} r \{Q\} \quad \text{\{cons\}} \]
Auxiliary rules

\[
\begin{align*}
\{P_1\} \quad & r \quad \{Q_1\} \quad \{P_2\} \quad & r \quad \{Q_2\} \\
\{P_1 \lor P_2\} \quad & r \quad \{Q_1 \lor Q_2\} \\
\{P_1 \land P_2\} \quad & r \quad \{Q_1 \land Q_2\}
\end{align*}
\]
{disj}

\[
\begin{align*}
\{P_1\} \quad & r \quad \{Q_1\} \quad \{P_2\} \quad & r \quad \{Q_2\} \\
\{P_1 \land P_2\} \quad & r \quad \{Q_1 \land Q_2\}
\end{align*}
\]
{conj}

\[
\begin{align*}
\{P\} \quad & r \quad \{Q\} \\
\{P \land R\} \quad & r \quad \{Q \land R\}
\end{align*}
\]
{frame}

\[
\begin{align*}
P \Rightarrow P' \quad & \{P'\} \quad & r \quad \{Q\} \\
\{P\} \quad & r \quad \{Q\}
\end{align*}
\]
{stren}

\[
\begin{align*}
\{P\} \quad & r \quad \{Q'\} \\
\{P\} \quad & r \quad \{Q\}
\end{align*}
\]
{weak}
Questions
Question 1

Is $\neg b$ an obvious invariant?

\[
\begin{aligned}
\{P \land b\} & \implies \{P\} \\
\{P\} & \text{ while } b \text{ do } c \{P \land \neg b\}
\end{aligned}
\]

\[
\begin{aligned}
\{\text{false}\} & \implies \{\neg b\} \\
\{\neg b\} & \text{ while } b \text{ do } c \{\neg b\}
\end{aligned}
\]
Find a derivation for the HL triple

\{ true \} if $x \geq y$ then $z := x$ else $z := y$ \{ $z = \max(x, y)$ \}

\{ true \}
if $x \geq y$ then

\{ $x \geq y$ \}
$z := x$
\{ $z = x \geq y$ \} \Rightarrow \{ $z = \max(x, y)$ \}
else

\{ $x < y$ \}
$z := y$
\{ $z = y > x$ \} \Rightarrow \{ $z = \max(x, y)$ \}
\{ $z = \max(x, y)$ \}
Prove that rule \{\text{conj}\} is sound

\[
\begin{align*}
\{P_1\} & \rightharpoonup \{Q_1\} & \{P_2\} & \rightharpoonup \{Q_2\} \\
\{P_1 \land P_2\} & \rightharpoonup \{Q_1 \land Q_2\} & \text{\{} \text{conj} \text{\}}
\end{align*}
\]
Show that the following rule for assignment is not sound

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