IN5170/IN9170 – Models of concurrency

Program Analysis (Part 2), lecture 8

Fall 2022

14. 09. 2022



The Formal System for PL (Hoare logic)

Express and prove program properties

$$\{P\}\ S\ \{Q\}$$

- (P, Q) is the specification of the program S, called a contract
- Code analysis by proving the specification (in PL)
- Symbolic and static analysis:
  - No need to execute the code in order to do the analysis
- An interpretation maps triples to *true* or *false*
  - $\{\ x = 0\ \}\ x := x + 1;\ \{\ x = 1\ \}$ should be semantically *true*
  - $\{\ x = 0\ \}\ x := x + 1;\ \{\ x = 0\ \}$ should be semantically *false*

Reasoning about programs

- Basic idea: *Specify* what the program is supposed to do (pre- and post-conditions)
- Pre- and post-conditions are given as assertions over the program state
- use PL for a mathematical argument that the program satisfies its specification

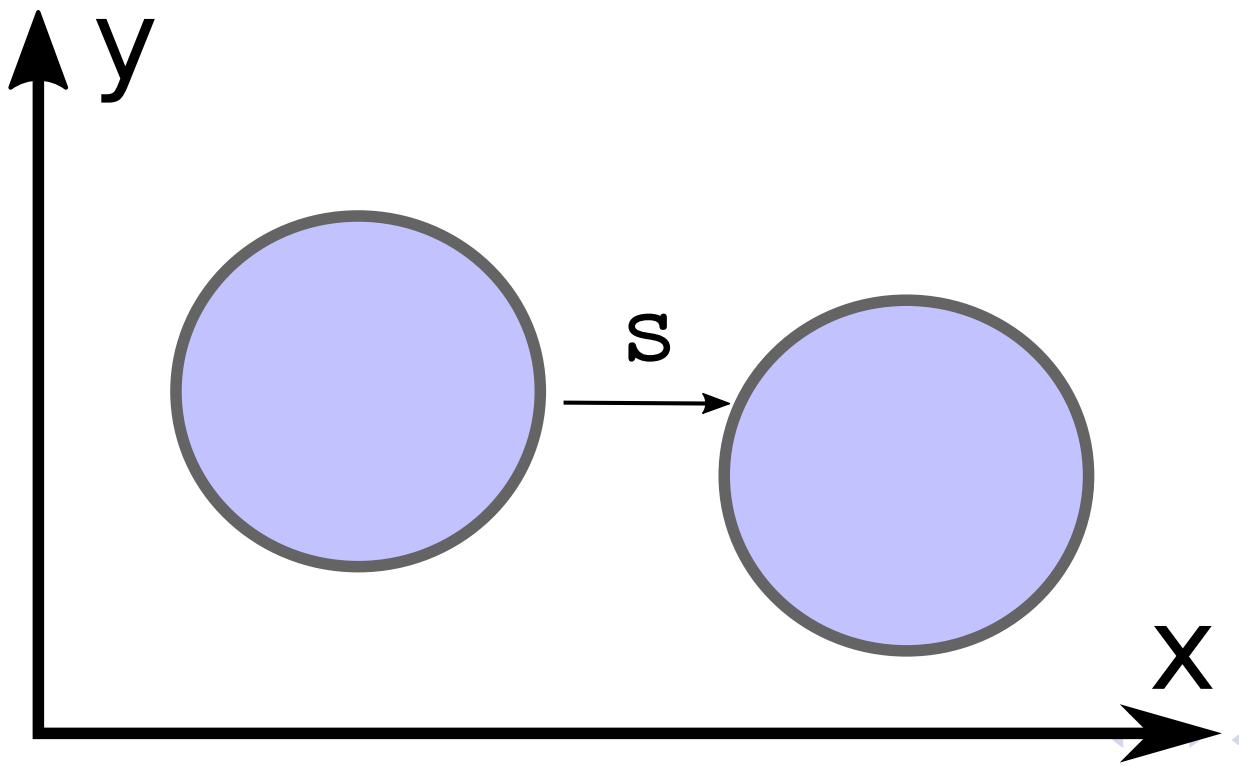
Predicate Transformation

Does the semantics match the specification?

Predicate Transformation

Does the semantics match the specification?

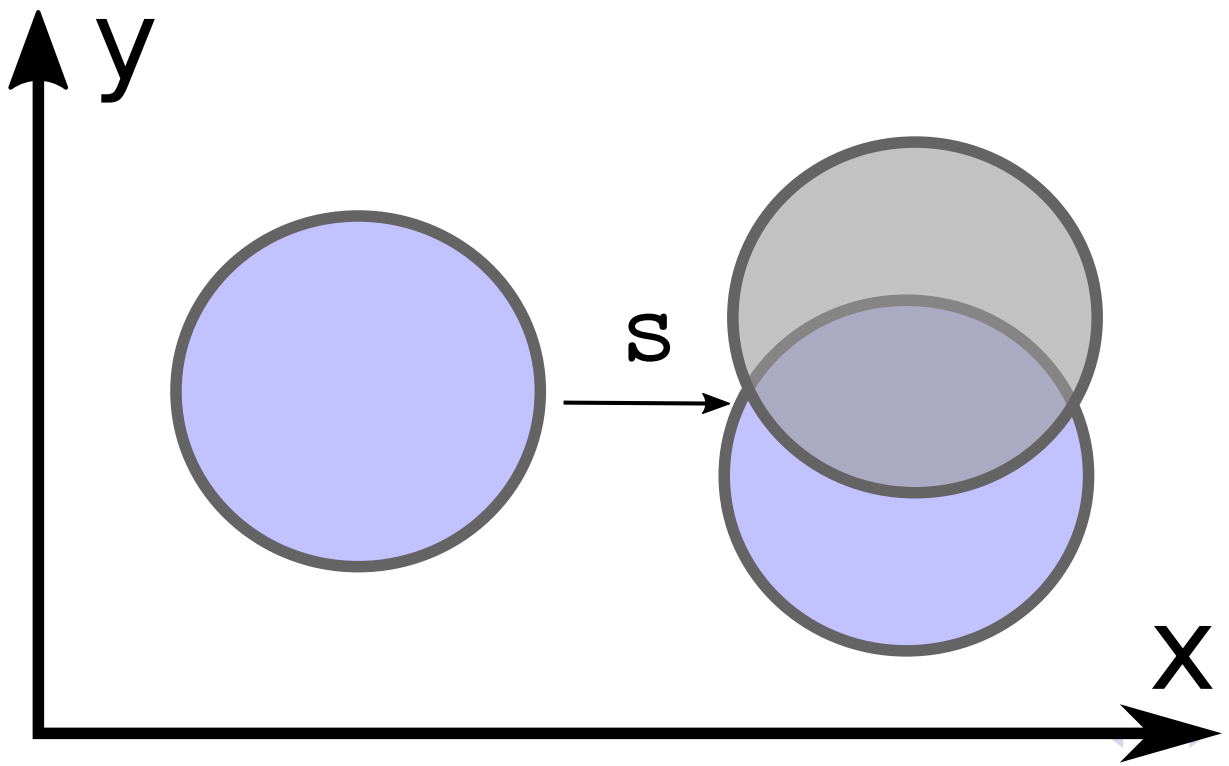
- The specification defines one transformation



Predicate Transformation

Does the semantics match the specification?

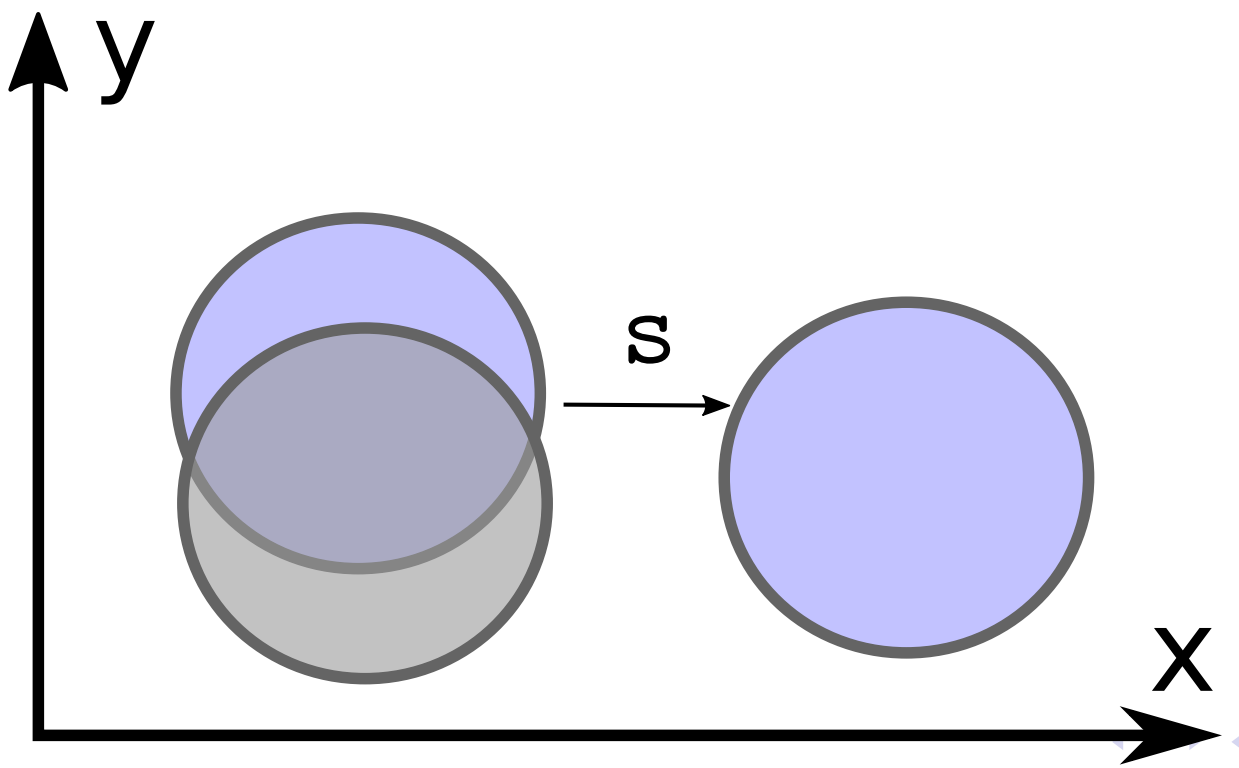
- The specification defines one transformation
- Post-condition may not match



Predicate Transformation

Does the semantics match the specification?

- The specification defines one transformation
- Post-condition may not match
- Pre-condition may not match



Interpretation:

Semantics of triples is related to program execution

Partial correctness

$\{ P \}\ S\ \{ Q \}$ is valid, denoted $\models \{ P \}\ S\ \{ Q \}$:

- If the initial state of S satisfies P and
- **if**[a] S *terminates,*
- *then* Q is *true* in the final state of S

[a] If S does not terminate, then the formula is considered true.

Interpretation:

Semantics of triples is related to program execution

Partial correctness

$\{ P \}\ S\ \{ Q \}$ is valid, denoted $\models \{ P \}\ S\ \{ Q \}$:

- If the initial state of S satisfies P and
- if[a] S *terminates,*
- *then* Q is *true* in the final state of S

[a] If S does not terminate, then the formula is considered true.

Expresses *partial correctness* (termination of S is assumed)

Example

$\{x = y\}\ \mathtt{x} := \mathtt{x} * 2;\ \mathtt{y} := \mathtt{y} * 2;\ \{x = y\}$ is *true*
if the initial state satisfies $x = y$ and if the execution terminates, then the final state satisfies $x = y$

Examples

Some valid triples (i.e., semantically true)

$$\{\, x = 0 \,\}\ x := x + 1;\ \{\, x = 1 \,\}$$
$$\{\, x = 4 \,\}\ x := 5;\ \{\, x = 5 \,\}$$
$$\{\, \mathsf{true} \,\}\ x := 5;\ \{\, x = 5 \,\}$$
$$\{\, y = 4 \,\}\ x := 5;\ \{\, y = 4 \,\}$$
$$\{\, x = 4 \,\}\ x := x + 1;\ \{\, x = 5 \,\}$$
$$\{\, x = 4 \wedge y = 7 \,\}\ x := x + 1;\ \{\, x = 5 \wedge y = 7 \,\}$$
$$\{\, x = y \,\}\ x := x + 1; y := y + 1;\ \{\, x = y \,\}$$

Examples

Some valid triples (i.e., semantically true)

$$\{\, x = 0 \,\}\ x := x + 1;\ \{\, x = 1 \,\}$$
$$\{\, x = 4 \,\}\ x := 5;\ \{\, x = 5 \,\}$$
$$\{\, \mathsf{true} \,\}\ x := 5;\ \{\, x = 5 \,\}$$
$$\{\, y = 4 \,\}\ x := 5;\ \{\, y = 4 \,\}$$
$$\{\, x = 4 \,\}\ x := x + 1;\ \{\, x = 5 \,\}$$
$$\{\, x = 4 \wedge y = 7 \,\}\ x := x + 1;\ \{\, x = 5 \wedge y = 7 \,\}$$
$$\{\, x = y \,\}\ x := x + 1; y := y + 1;\ \{\, x = y \,\}$$

Some unvalid triples (i.e., not semantically true)

$$\{\, x = 0 \,\}\ x := x + 1;\ \{\, x = 0 \,\}$$
$$\{\, x = 4 \,\}\ x := 5;\ \{\, x = 4 \,\}$$
$$\{\, x = 4 \,\}\ y := 5;\ \{\, x = 5 \,\}$$
$$\{\, x = y \,\}\ x := x + 1; y := y - 1;\ \{\, x = y \,\}$$

Partial correctness

- The interpretation of $\{ P \}\ S\ \{ Q \}$ assumes/ignores termination of S, since termination is not proven.
- The pre/post specification (P, Q) expresses *safety* properties

What does each of the following triple express?

$\{ P \}\ S\ \{ \mathit{true} \}$
$\{ P \}\ S\ \{ \mathit{false} \}$
$\{ \mathit{true} \}\ S\ \{ \mathit{false} \}$
$\{ \mathit{true} \}\ S\ \{ Q \}$

$\{ \mathit{false} \}\ S\ \{ Q \}$

Partial correctness

- The interpretation of $\{ P \}\ S\ \{ Q \}$ assumes/ignores termination of S, since termination is not proven.
- The pre/post specification (P, Q) expresses *safety* properties

What does each of the following triple express?

$\{ P \}\ S\ \{ \mathit{true} \}$ trivially true

$\{ P \}\ S\ \{ \mathit{false} \}$

$\{ \mathit{true} \}\ S\ \{ \mathit{false} \}$

$\{ \mathit{true} \}\ S\ \{ Q \}$

$\{ \mathit{false} \}\ S\ \{ Q \}$

Partial correctness

- The interpretation of $\{ P \}\ S\ \{ Q \}$ assumes/ignores termination of S, since termination is not proven.
- The pre/post specification (P, Q) expresses *safety* properties

What does each of the following triple express?

$\{ P \}\ S\ \{ \mathit{true} \}$ trivially true

$\{ P \}\ S\ \{ \mathit{false} \}$ S does not terminate for prestate P

$\{ \mathit{true} \}\ S\ \{ \mathit{false} \}$

$\{ \mathit{true} \}\ S\ \{ Q \}$

$\{ \mathit{false} \}\ S\ \{ Q \}$

Partial correctness

- The interpretation of $\{ P \}\ S\ \{ Q \}$ assumes/ignores termination of S, since termination is not proven.
- The pre/post specification (P, Q) expresses *safety* properties

What does each of the following triple express?

$\{ P \}\ S\ \{ \mathit{true} \}$	trivially true
$\{ P \}\ S\ \{ \mathit{false} \}$	S does not terminate for prestate P
$\{ \mathit{true} \}\ S\ \{ \mathit{false} \}$	S does not terminate
$\{ \mathit{true} \}\ S\ \{ Q \}$	
$\{ \mathit{false} \}\ S\ \{ Q \}$	

Partial correctness

- The interpretation of $\{ P \}\ S\ \{ Q \}$ assumes/ignores termination of S, since termination is not proven.
- The pre/post specification (P, Q) expresses *safety* properties

What does each of the following triple express?

$\{ P \}\ S\ \{ \mathit{true} \}$	trivially true
$\{ P \}\ S\ \{ \mathit{false} \}$	S does not terminate for prestate P
$\{ \mathit{true} \}\ S\ \{ \mathit{false} \}$	S does not terminate
$\{ \mathit{true} \}\ S\ \{ Q \}$	Q holds after S for all prestates (provided S terminates)
$\{ \mathit{false} \}\ S\ \{ Q \}$	

Partial correctness

- The interpretation of $\{ P \}\ S\ \{ Q \}$ assumes/ignores termination of S, since termination is not proven.
- The pre/post specification (P, Q) expresses *safety* properties

What does each of the following triple express?

$\{ P \}\ S\ \{ \mathit{true} \}$	trivially true
$\{ P \}\ S\ \{ \mathit{false} \}$	S does not terminate for prestate P
$\{ \mathit{true} \}\ S\ \{ \mathit{false} \}$	S does not terminate
$\{ \mathit{true} \}\ S\ \{ Q \}$	Q holds after S for all prestates (provided S terminates)
$\{ \mathit{false} \}\ S\ \{ Q \}$	trivially true

Can express non-termination, but not termination!

The Proof System for PL

Proof system PL

A proof system consists of *axioms* and *rules* here: structural analysis of programs

- Axioms for basic statements:
  - $x := e$, skip,...
- Rules for composed statements:
  - $S_1;S_2$, if, while, await , co . . . oc, . . .

A proof system consists of *axioms* and *rules* here: structural analysis of programs

- Axioms for basic statements:
  - $x := e$, skip,...
- Rules for composed statements:
  - $S_1;S_2$, if, while, await, co ... oc, ...

Formulas in PL

- formulas = triples
- theorems = derivable formulas
- hopefully: all derivable formulas are also "really" (= semantically) true
- derivation: starting from axioms, using derivation rules
- $$\frac{H_1 \quad H_2 \quad \ldots \quad H_n}{C}$$
- axioms: can be seen as rules without premises

Soundness – $\vdash$ implies $\models$

If a triple $\{ P \} S \{ Q \}$ is derivable in PL, the the triple is valid:

- Example: we want

$$\{ x = 0 \} x := x + 1 \{ x = 1 \}$$

  to be a theorem (since it was interpreted as *true*),

- but

$$\{ x = 0 \} x := x + 1 \{ x = 0 \}$$

  should *not* be a theorem (since it was interpreted as *false*)

Soundness:

All theorems in PL hold

$$\vdash \{ P \} S \{ Q \} \quad \text{implies} \quad \models \{ P \} S \{ Q \} \qquad (1)$$

If we can use PL to prove some property of a program, then this property will hold for all executions of the program.

Substitution

$P[e/x]$ means, all free occurrences of x in P are replaced by expression e. (Notation of book: $P_{x \leftarrow e}$)

Example

$$
\begin{array}{lcl}
(x = 1)[(x+1)/x] & \Leftrightarrow & x + 1 = 1 \\
(x + y = a)[(y+x)/y] & \Leftrightarrow & x + (y + x) = a \\
(y = a)[(x+y)/x] & \Leftrightarrow & y = a
\end{array}
$$

Substitution propagates into formulas:

$$
\begin{array}{lcl}
(\neg A)[e/x] & \Leftrightarrow & \neg(A[e/x]) \\
(A \wedge B)[e/x] & \Leftrightarrow & A[e/x] \wedge B[e/x] \\
(A \vee B)[e/x] & \Leftrightarrow & A[e/x] \vee B[e/x]
\end{array}
$$

Free and "non-free" variable occurrences

$P[e/x]$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$
\begin{aligned}
(\exists y : x + y > 0)[1/x] &\Leftrightarrow \\
(\exists x : x + y > 0)[1/x] &\Leftrightarrow \\
(\exists x : x + y > 0)[x/y] &\Leftrightarrow
\end{aligned}
$$

Correspondingly for $\forall$

Free and "non-free" variable occurrences

$P[e/x]$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$(\exists y : x + y > 0)[1/x] \quad \Leftrightarrow \quad \exists y : 1 + y > 0$$
$$(\exists x : x + y > 0)[1/x] \quad \Leftrightarrow$$
$$(\exists x : x + y > 0)[x/y] \quad \Leftrightarrow$$

Correspondingly for $\forall$

Free and "non-free" variable occurrences

$P[e/x]$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$
\begin{array}{lcl}
(\exists y : x + y > 0)[1/x] & \Leftrightarrow & \exists y : 1 + y > 0 \\
(\exists x : x + y > 0)[1/x] & \Leftrightarrow & \exists x : x + y > 0 \\
(\exists x : x + y > 0)[x/y] & \Leftrightarrow &
\end{array}
$$

Correspondingly for $\forall$

Free and "non-free" variable occurrences

$P[e/x]$

- Only *free* occurrences of x are substituted
- Variable occurrences may be *bound* by quantifiers, then that occurrence of the variable is not free (but bound)

Example (Substitution)

$$
\begin{array}{lcl}
(\exists y : x + y > 0)[1/x] & \Leftrightarrow & \exists y : 1 + y > 0 \\
(\exists x : x + y > 0)[1/x] & \Leftrightarrow & \exists x : x + y > 0 \\
(\exists x : x + y > 0)[x/y] & \Leftrightarrow & \exists z : z + x > 0
\end{array}
$$

Correspondingly for $\forall$

Assignment

Backwards Reasoning for Assignments

Given the postcondition to the assignment, derive the precondition

What is the precondition?

$$\{\ ?\ \}\ x := e\ \{\ x = 5\ \}$$

If the assignment `x = e` should terminate in a state where `x` has the value 5, the expression e must have the value 5 before the assignment:

$$\begin{array}{rll} \{\ e = 5\ \} & \texttt{x := e} & \{\ x = 5\ \} \\ \{\ (x = 5)[e/x]\ \} & \texttt{x := e} & \{\ x = 5\ \} \end{array}$$

Axiom of assignment

"Backwards reasoning:" Given a postcondition, we may construct the precondition:

Axiom for the assignment statement

$$\{\ P[e/x]\ \}\ x := e\ \{\ P\ \} \quad \text{Assign}$$

If the assignment $x := e$ should lead to a state that satisfies P, the state before the assignment must satisfy P when x is replaced by e.
NB: e evaluated in the prestate = value of x in the poststate

PL structural inference rules

$$\frac{P' \Rightarrow P \qquad \{\, P \,\}\ S\ \{\, Q \,\} \qquad Q \Rightarrow Q'}{\{\, P' \,\}\ S\ \{\, Q' \,\}}\ \text{Consequence}$$

$$\frac{\{\, P \,\}\ S\ \{\, Q \,\} \qquad \{\, P' \,\}\ S\ \{\, Q' \,\}}{\{\, P \wedge P' \,\}\ S\ \{\, Q \wedge Q' \,\}}\ \text{Conjunction}$$

$$\frac{\{\, P \,\}\ S_1\ \{\, R \,\} \qquad \{\, R \,\}\ S_2\ \{\, Q \,\}}{\{\, P \,\}\ S_1 ; S_2\ \{\, Q \,\}}\ \text{Seq}$$

- Red: logical proof obligations

Axiom of skip

The skip statement does nothing

Axiom:

$$\{\ P\ \}\ \text{skip}\ \{\ P\ \} \quad \text{Skip-axiom}$$

Proving an assignment

We can derive a lemma inference rule that is less syntax driven:

$$\frac{P \Rightarrow Q[e/x]}{\{\ P\ \}\ x := e\ \{\ Q\ \}}$$

Derivation using the consequence rule:

$$\frac{P \Rightarrow Q[e/x] \qquad \dfrac{}{\{\ Q[e/x]\ \}\ x := e\ \{\ Q\ \}} \qquad \dfrac{}{Q \Rightarrow Q}}{\{\ P\ \}\ x := e\ \{\ Q\ \}}$$

- $Q[e/x]$ is the largest set of states such that the assignment is guaranteed to terminate with Q
- largest set corresponds to weakest condition $\Rightarrow$ weakest-precondition reasoning
- Opposite: strongest post-consition and forward reasoning (= symbolic execution)

Examples

$$\frac{\mathit{true} \Rightarrow 1 = 1}{\{\ \mathit{true}\ \}\ x := 1\ \{\ x = 1\ \}}$$

$$\frac{x = 0 \Rightarrow x + 1 = 1}{\{\ x = 0\ \}\ x := x + 1\ \{\ x = 1\ \}}$$

$$\frac{(x = a \wedge y = b) \Rightarrow x + y = a + b \wedge y = b}{\{\ x = a \wedge y = b\ \}\ x := x + y\ \{\ x = a + b \wedge y = b\ \}}$$

$$\frac{y > 0 \Rightarrow y \geq 0}{\{\ y > 0\ \} x := y \{\ x \geq 0\ \}}$$

Backward construction over assignments:

$$\dfrac{\dfrac{x = y \Rightarrow 2x = 2y}{\{\, x = y \,\}\ x := 2x\ \{\, x = 2y \,\}} \qquad \{\, (x = y)[2y/y] \,\}\ y := 2y\ \{\, x = y \,\}}{\{\, x = y \,\}\ x := 2x; y := 2y\ \{\, x = y \,\}}$$

Backwards reasoning keeps your substitutions simple.

Normally we just apply derived *lemmata*:

Backward construction over assignments:

$$\frac{x = y \Rightarrow 2x = 2y}{\{\, x = y \,\}\ x := 2x\ \{\, x = 2y \,\} \qquad \{\, (x = y)[2y/y] \,\}\ y := 2y\ \{\, x = y \,\}}$$
$$\{\, x = y \,\}\ x := 2x; y := 2y\ \{\, x = y \,\}$$

Backwards reasoning keeps your substitutions simple.

Normally we just apply derived *lemmata*:

$$\frac{\{\, P \,\}\ S\ \{\, Q[e/x] \,\}}{\{\, P \,\}\ S; x := e\ \{\, Q \,\}}$$

Backward construction over assignments:

$$\dfrac{\dfrac{x = y \Rightarrow 2x = 2y}{\{\, x = y \,\}\ x := 2x\ \{\, x = 2y \,\}} \qquad \{\, (x = y)[2y/y] \,\}\ y := 2y\ \{\, x = y \,\}}{\{\, x = y \,\}\ x := 2x; y := 2y\ \{\, x = y \,\}}$$

Backwards reasoning keeps your substitutions simple.

Normally we just apply derived *lemmata*:

$$\dfrac{\{\, P \,\}\ S\ \{\, Q[e/x] \,\} \qquad \dfrac{}{\{\, Q[e/x] \,\}\ x := e\ \{\, Q \,\}}}{\{\, P \,\}\ S; x := e\ \{\, Q \,\}}$$

Backward construction over assignments:

$$\dfrac{x = y \Rightarrow 2x = 2y}{\dfrac{\{\, x = y \,\}\ x := 2x\ \{\, x = 2y \,\} \qquad \{\, (x = y)[2y/y] \,\}\ y := 2y\ \{\, x = y \,\}}{\{\, x = y \,\}\ x := 2x; y := 2y\ \{\, x = y \,\}}}$$

Backwards reasoning keeps your substitutions simple.

Normally we just apply derived *lemmata*:

$$\dfrac{(q * y) + x = a \Rightarrow ((q + 1) * y) + x - y = a}{\dfrac{\{\, (q * y) + x = a \,\}\ x := x - y;\ \{\, ((q + 1) * y) + x = a \,\}}{\{\, (q * y) + x = a \,\}\ x := x - y; q := q + 1\ \{\, (q * y) + x = a \,\}}}$$

Logical variables

- Do *not* occur in program text
- Used only in *assertions*
- May be used to "freeze" initial values of variables
- May then talk about these values in the postcondition

Example

$$\{\ x = x_0\ \}\ \text{if}(x < 0)\ \text{then}\ x := -x\ \text{fi}\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}$$

where $(x = x_0 \vee x = -x_0)$ states that

- the final value of x equals the initial value, *or*
- the final value of x is the negation of the initial value

Branching

$$\frac{\{\, P \wedge B \,\}\ S\ \{\, Q \,\} \qquad P \wedge \neg B \Rightarrow Q}{\{\, P \,\}\ \text{if } B \text{ then } S \text{ fi}\ \{\, Q \,\}}\ \text{Cond1}$$

$$\frac{\{\, P \wedge B \,\}\ S\ \{\, Q \,\} \qquad \{\, P \wedge \neg B \,\}\ S'\ \{\, Q \,\}}{\{\, P \,\}\ \text{if } B \text{ then } S \text{ else } S'\ \{\, Q \,\}}\ \text{Cond2}$$

- Red: logical proof obligations

Example: if statement

Verification of:

$$\{\ x = x_0\ \}\ \text{if}\ (x < 0)\ \text{then}\ x := -x\ \text{fi}\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}$$

Example: if statement

Verification of:

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad * \qquad\qquad **}{\{\ x = x_0\ \}\ \text{if } (x < 0) \text{ then } x := -x \text{ fi } \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad (P \wedge \neg B) \Rightarrow Q}{\{\ P\ \}\ \text{if}\ \ B\ \text{then}\ \ S\ \text{fi}\ \{\ Q\ \}}\ \text{Cond1}$$

Example: if statement

Verification of:

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad * \qquad\qquad **}{\{\ x = x_0\ \}\ \text{if } (x < 0) \text{ then } x := -x \text{ fi } \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad (P \wedge \neg B) \Rightarrow Q}{\{\ P\ \}\ \text{if}\ \ B\ \text{then}\ \ S\ \text{fi}\ \{\ Q\ \}}\ \text{Cond1}$$

- Premisse $\{\ P \wedge B\ \}\ S\ \{\ Q\ \}$ $(*)$:

- Premisse $P \wedge \neg B \Rightarrow Q$ $(**)$:

Example: if statement

Verification of:

$$\frac{\qquad\qquad * \qquad\qquad **}{\{\, x = x_0 \,\}\ \text{if } (x < 0) \text{ then } x := -x \text{ fi } \{\, x \geq 0 \wedge (x = x_0 \vee x = -x_0) \,\}}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad (P \wedge \neg B) \Rightarrow Q}{\{\, P \,\}\ \text{if } B \text{ then } S \text{ fi } \{\, Q \,\}}\ \text{Cond1}$$

- Premisse $\{\, P \wedge B \,\}\ S\ \{\, Q \,\}$ $(*)$:

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{x = x_0 \wedge x < 0 \Rightarrow (-x \geq 0 \wedge (-x = x_0 \vee -x = -x_0))}$$
$$\{\, x = x_0 \wedge x < 0 \,\}\ x := -x\ \{\, x \geq 0 \wedge (x = x_0 \vee x = -x_0) \,\}$$

- Premisse $P \wedge \neg B \Rightarrow Q$ $(**)$:

Example: if statement

Verification of:

$$\frac{\qquad\qquad * \qquad\qquad **}{\{\ x = x_0\ \}\ \text{if } (x < 0) \text{ then } x := -x \text{ fi } \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}}$$

$$\frac{\{P \wedge B\}\ S\ \{Q\} \qquad (P \wedge \neg B) \Rightarrow Q}{\{\ P\ \}\ \text{if}\ \ B\ \text{then}\ \ S\ \text{fi}\ \{\ Q\ \}}\ \text{Cond1}$$

- Premisse $\{\ P \wedge B\ \}\ S\ \{\ Q\ \}$ $(*)$:

$$\frac{x = x_0 \wedge x < 0 \Rightarrow (-x \geq 0 \wedge (-x = x_0 \vee -x = -x_0))}{\{\ x = x_0 \wedge x < 0\ \}\ x := -x\ \{\ x \geq 0 \wedge (x = x_0 \vee x = -x_0)\ \}}$$

- Premisse $P \wedge \neg B \Rightarrow Q$ $(**)$:

$$\frac{}{x = x_0 \wedge x \geq 0 \Rightarrow (x \geq 0 \wedge (x = x_0 \vee x = -x_0))}$$

Loop Invariants

Proof Rule for Loops

$$\frac{\{\, I \wedge B \,\}\ S\ \{\, I \,\}}{\{\, I \,\}\ \text{while } B \text{ do } S\ \{\, I \wedge \neg B \,\}}\ \text{While}$$

Loop Invariant

- Red: logical proof obligations
- The rule for while needs a loop invariant!
- Usually combined with other rules

$$\frac{P \Rightarrow I \qquad \dfrac{\{\, I \wedge B \,\}\ S\ \{\, I \,\}}{\{\, I \,\}\ \text{while } B \text{ do } S\ \{\, I \wedge \neg B \,\}} \qquad I \wedge \neg B \Rightarrow Q}{\{\, P \,\}\ \text{while } B \text{ do } S\ \{\, Q \,\}}$$

Proof Rule for Loops

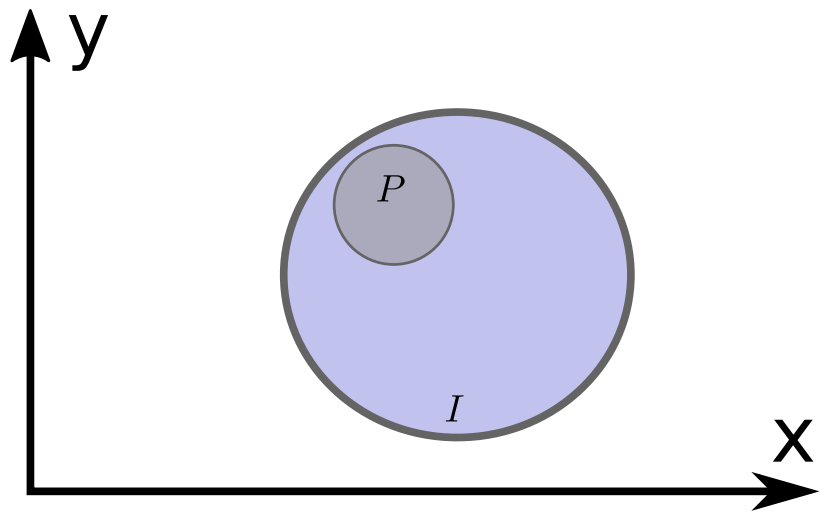
$$\frac{P \Rightarrow I \qquad \dfrac{\{\, I \wedge B \,\}\ S\ \{\, I \,\}}{\{\, I \,\}\ \text{while } B \text{ do } S\ \{\, I \wedge \neg B \,\}} \qquad I \wedge \neg B \Rightarrow Q}{\{\, P \,\}\ \text{while } B \text{ do } S\ \{\, Q \,\}}$$

Three Hypotheses

- $P \Rightarrow I$ (*Initial Case*)
- $\{\, I \wedge B \,\}\ S\ \{\, I \,\}$ (*Preserves Case*)
- $I \wedge \neg B \Rightarrow Q$ (*Use Case*)

Three Hypotheses

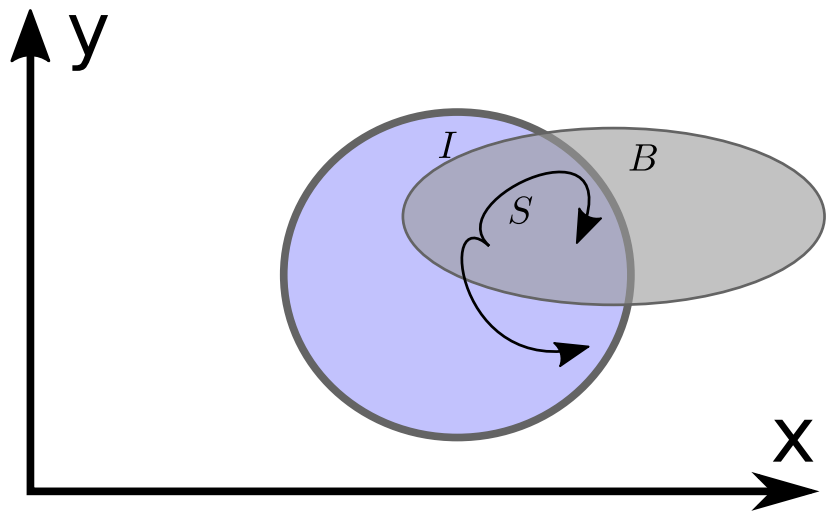
- $P \Rightarrow I$ (*Initial Case*): Invariant holds before first loop iteration



Proof Rule for Loops

Three Hypotheses

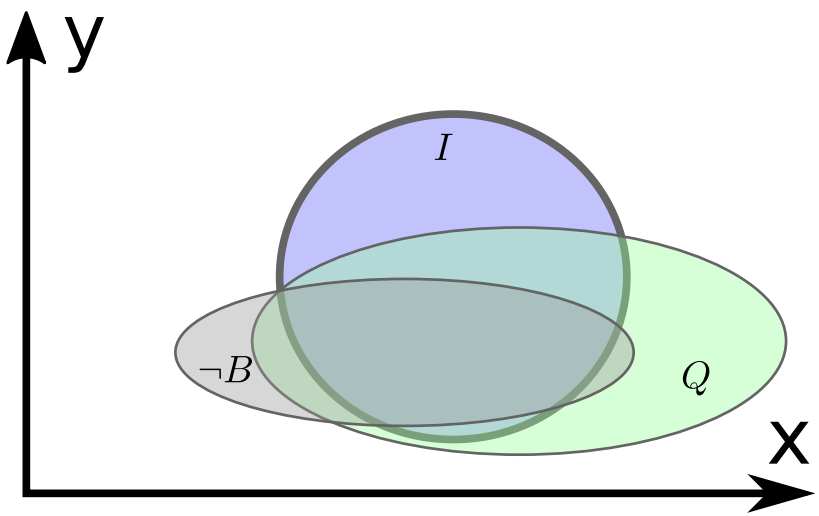
- $P \Rightarrow I$ (*Initial Case*): Invariant holds before first loop iteration
- $\{ I \wedge B \}\ S\ \{ I \}$ (*Preserves Case*): Invariant is preserved by loop body

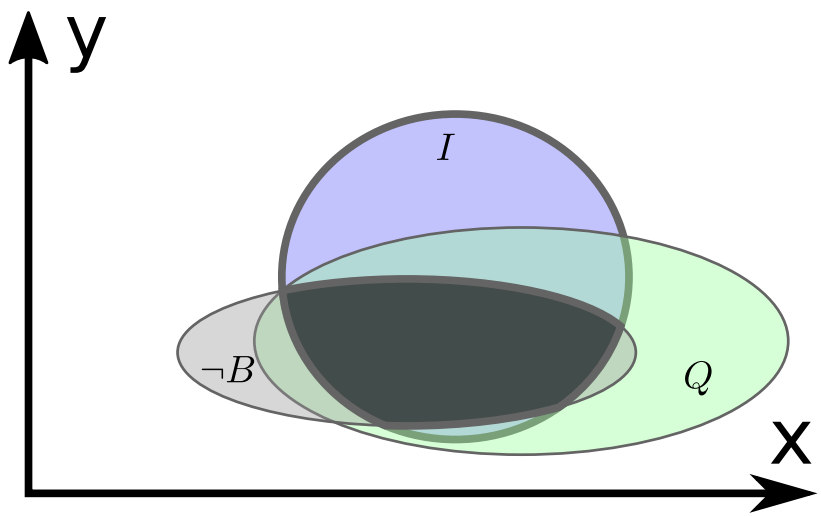


Proof Rule for Loops

Three Hypotheses

- $P \Rightarrow I$ (*Initial Case*): Invariant holds before first loop iteration
- $\{ I \wedge B \}\ S\ \{ I \}$ (*Preserves Case*): Invariant is preserved by loop body
- $I \wedge \neg B \Rightarrow Q$ (*Use Case*): Invariant is strong enough to describe post-state





Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

$\{ x \geq 0 \}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{ x = y \}$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

$$\frac{}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}}$$
$$\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \dfrac{}{\{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

$$\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

Example

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \dfrac{\dfrac{}{x \geq 0 \wedge y = 0 \Rightarrow y \leq x}}{\{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

$$\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \dfrac{\dfrac{}{x \geq 0 \wedge y = 0 \Rightarrow y \leq x} \qquad \dfrac{}{y \leq x \wedge y \geq x \Rightarrow x = y}}{\{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

$$\dfrac{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\} \qquad \{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}{\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \dfrac{\dfrac{}{x \geq 0 \wedge y = 0 \Rightarrow y \leq x} \quad * \quad \dfrac{}{y \leq x \wedge y \geq x \Rightarrow x = y}}{\{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

$$\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates

Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{ x \geq 0 \}\ y := 0\ \{ y = 0 \wedge x \geq 0 \}} \qquad \dfrac{\dfrac{}{x \geq 0 \wedge y = 0 \Rightarrow y \leq x} \quad * \quad \dfrac{}{y \leq x \wedge y \geq x \Rightarrow x = y}}{\{ x \geq 0 \wedge y = 0 \}\ \text{while } y < x \text{ do } y := y + 1\ \{ x = y \}}$$

$$\{ x \geq 0 \}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{ x = y \}$$

*

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates
Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \dfrac{\dfrac{}{x \geq 0 \wedge y = 0 \Rightarrow y \leq x} \quad * \quad \dfrac{}{y \leq x \wedge y \geq x \Rightarrow x = y}}{\{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

$$\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

$$\dfrac{}{\{\, y \leq x \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, y \leq x \wedge y \geq x \,\}}$$

$*$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $y = x$ if it terminates
Invariant: $y \leq x$.

$$\dfrac{\dfrac{}{x \geq 0 \Rightarrow 0 = 0 \wedge x \geq 0}}{\{\, x \geq 0 \,\}\ y := 0\ \{\, y = 0 \wedge x \geq 0 \,\}} \qquad \dfrac{\dfrac{}{x \geq 0 \wedge y = 0 \Rightarrow y \leq x} \quad * \quad \dfrac{}{y \leq x \wedge y \geq x \Rightarrow x = y}}{\{\, x \geq 0 \wedge y = 0 \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}}$$

$$\{\, x \geq 0 \,\}\ y := 0; \text{while } y < x \text{ do } y := y + 1\ \{\, x = y \,\}$$

$$\dfrac{\dfrac{\dfrac{}{y \leq x \wedge y < x \Rightarrow y + 1 \leq x}}{\{\, y \leq x \wedge y < x \,\}\ y := y + 1\ \{\, y \leq x \,\}}}{\{\, y \leq x \,\}\ \text{while } y < x \text{ do } y := y + 1\ \{\, y \leq x \wedge y \geq x \,\}}$$

$*$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

terminates.

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

terminates. What is the termination witness?
(Rem: Monotonously descreasing and positive.)

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

terminates. What is the termination witness?
(Rem: Monotonously descreasing and positive.)

$\{ old = x - y \wedge x - y \geq 0 \wedge y < x \}\ y := y{+}1\ \{ old > x - y \wedge x - y \geq$

Counter

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

terminates. What is the termination witness?
(Rem: Monotonously descreasing and positive.)

$\{ old = x - y \wedge x - y \geq 0 \wedge y < x \}\ y := y+1\ \{ old > x - y \wedge x - y \geq$

Note that the termination argument is outside the logic.

Example

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $0 \leq y = x$

Example

Prove that for every $x \geq 0$ the program

$$y := 0; \text{while } y < x \text{ do } y := y + 1$$

results in a state where $0 \leq y = x$

Framing

Seems obvious, x does not change....

- The loop does not change x, but we still need to specify it!
- New invariant $0 \leq y \leq x$ (keep $x \geq 0$)
- Typical problem: need to *frame* which variables are relevant
- Often in two different sets: (1) read and (2) written variables
  - R: $\{x, y\}$
  - W: $\{y\}$
- Can be automatically derived in simple scenarios, becomes more involved in OO and with data structures

References I

[Andrews, 2000] Andrews, G. R. (2000).
Foundations of Multithreaded, Parallel, and Distributed Programming.
Addison-Wesley.

[Dahl, 1992] Dahl, O.-J. (1992).
Verifiable Programming.
International Series in Computer Science. Prentice Hall, New York, N.Y.

[Gordon, 2016] Gordon, M. (2016).
Background reading on Hoare Logic.
https://www.cl.cam.ac.uk/archive/mjcg/HL/Notes/Notes.pdf.

[Hoare, 1969] Hoare, C. A. R. (1969).
An Axiomatic Basis of Computer Programming.
Communications of the ACM, 12:576–580.