Softwaretechnik / Software-Engineering

Lecture 16: Program Verification

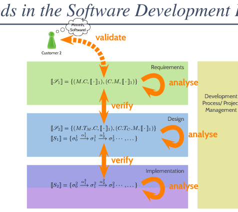
2019-07-18

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Code Quality Assurance: Content

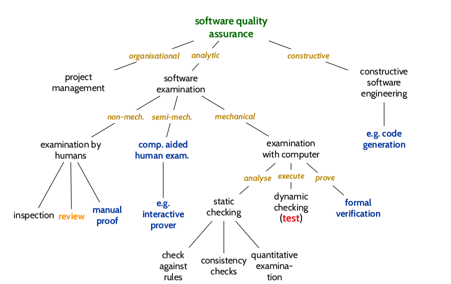
Formal Methods in the Software Development Process



validation—
The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.
Contrast with: **verification.**
IEEE 610.12 (1990)

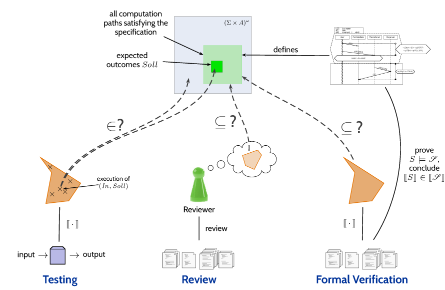
verification—
(1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
Contrast with: **validation.**
(2) Formal proof of program correctness.
IEEE 610.12 (1990)

Concepts of Software Quality Assurance



(Ludewig and Lichter, 2013)

Testing, Review, Verification Illustrated



Content

Sequential, Deterministic While-Programs

Deterministic Programs

Syntax:

$$S ::= skip \mid u := t \mid S_1; S_2 \mid \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid \textbf{while } B \textbf{ do } S_1 \textbf{ od}$$

where $u \in V$ is a **variable**, t is a type-compatible **expression**, B is a Boolean **expression**.

Semantics: (is induced by the following transition relation) — $\sigma : V \to \mathcal{D}(V)$

(i) $\langle skip, \sigma\rangle \to \langle E, \sigma\rangle$ empty program

(ii) $\langle u := t, \sigma\rangle \to \langle E, \sigma[u := \sigma(t)]\rangle$

(iii) $\dfrac{\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle}{\langle S_1; S, \sigma\rangle \to \langle S_2; S, \tau\rangle}$

(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$, if $\sigma \models B$,

(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$, if $\sigma \not\models B$,

(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle S; \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle$, if $\sigma \models B$,

(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle E, \sigma\rangle$, if $\sigma \not\models B$,

E denotes the **empty program**; define $E; S \equiv S; E \equiv S$.

Note: the first component of $\langle S, \sigma\rangle$ is a program (**structural operational semantics** (SOS)).

Example

(i) $\langle skip, \sigma\rangle \to \langle E, \sigma\rangle$ $\quad E; S \equiv S; E \equiv S$
(ii) $\langle u := t, \sigma\rangle \to \langle E, \sigma[u := \sigma(t)]\rangle$
(iii) $\dfrac{\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle}{\langle S_1; S, \sigma\rangle \to \langle S_2; S, \tau\rangle}$
(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle S; \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle E, \sigma\rangle$, if $\sigma \not\models B$,

Consider **program**

$$S \equiv \underbrace{a[0]}_{u} := \underbrace{1}_{t}; a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}$$

and a **state** σ with $\sigma \models x = 0$.

$$\langle S, \sigma\rangle \xrightarrow{(ii),(iii)} \langle a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}, \underbrace{\sigma[a[0] := 1]}_{(ii)}\rangle$$

Example

(i) $\langle skip, \sigma\rangle \to \langle E, \sigma\rangle$ $\quad E; S \equiv S; E \equiv S$
(ii) $\langle u := t, \sigma\rangle \to \langle E, \sigma[u := \sigma(t)]\rangle$
(iii) $\dfrac{\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle}{\langle S_1; S, \sigma\rangle \to \langle S_2; S, \tau\rangle}$
(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle S; \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle E, \sigma\rangle$, if $\sigma \not\models B$,

Consider **program**

$$S \equiv a[0] := 1; a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}$$

and a **state** σ with $\sigma \models x = 0$.

$$\begin{aligned}\langle S, \sigma\rangle &\xrightarrow{(ii),(iii)} \langle a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}, \sigma[a[0] := 1]\rangle\\ &\xrightarrow{(ii),(iii)} \langle \textbf{while } \underbrace{a[x] \neq 0}_{B} \textbf{ do } x := x + 1 \textbf{ od}, \sigma'\rangle\end{aligned}$$

where $\sigma' = (\sigma[a[0] := 1])[a[1] := 0]$.

Example

(i) $\langle skip, \sigma\rangle \to \langle E, \sigma\rangle$ $\quad E; S \equiv S; E \equiv S$
(ii) $\langle u := t, \sigma\rangle \to \langle E, \sigma[u := \sigma(t)]\rangle$
(iii) $\dfrac{\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle}{\langle S_1; S, \sigma\rangle \to \langle S_2; S, \tau\rangle}$
(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle S; \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle E, \sigma\rangle$, if $\sigma \not\models B$,

Consider **program**

$$S \equiv a[0] := 1; a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}$$

and a **state** σ with $\sigma \models x = 0$.

$$\begin{aligned}\langle S, \sigma\rangle &\xrightarrow{(ii),(iii)} \langle a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}, \sigma[a[0] := 1]\rangle\\ &\xrightarrow{(ii),(iii)} \langle \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}, \sigma'\rangle\\ &\xrightarrow{(vi)} \langle x := x + 1; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}, \sigma'\rangle\\ &\xrightarrow{(ii),(iii)} \langle \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}, \sigma'[x := 1]\rangle\\ &\xrightarrow{(vii)} \langle E, \sigma'[x := 1]\rangle\end{aligned}$$

where $\sigma' = \sigma[a[0] := 1][a[1] := 0]$.

Another Example

(i) $\langle skip,\ \sigma\rangle \to \langle E,\ \sigma\rangle$ $\qquad E; S \equiv S; E \equiv S$
(ii) $\langle u := t,\ \sigma\rangle \to \langle E,\ \sigma[u := \sigma(t)]\rangle$
(iii) $\dfrac{\langle S_1,\ \sigma\rangle \to \langle S_2,\ \tau\rangle}{\langle S_1; S,\ \sigma\rangle \to \langle S_2; S,\ \tau\rangle}$
(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi},\ \sigma\rangle \to \langle S_1,\ \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi},\ \sigma\rangle \to \langle S_2,\ \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od},\ \sigma\rangle \to \langle S; \textbf{while } B \textbf{ do } S \textbf{ od},\ \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od},\ \sigma\rangle \to \langle E,\ \sigma\rangle$, if $\sigma \not\models B$,

Consider **program**

$$S_1 \equiv y := x; y := (x-1)\cdot x + y$$

and a **state** σ with $\sigma \models x = 3$.

$$\begin{aligned}\langle S_1,\ \sigma\rangle &\xrightarrow{(ii),(iii)} \langle y := (x-1)\cdot x + y,\ \{x \mapsto 3, y \mapsto 3\}\rangle\\ &\xrightarrow{(ii)} \langle E,\ \{x \mapsto 3, y \mapsto 9\}\rangle\end{aligned}$$

Consider **program** $S_3 \equiv y := x;\ y := (x-1)\cdot x + y;\ \textbf{while } 1 \textbf{ do } skip \textbf{ od}.$

$$\begin{aligned}\langle S_3,\ \sigma\rangle &\xrightarrow{(ii),(iii)} \langle y := (x-1)\cdot x + y; \textbf{while } 1 \textbf{ do } skip \textbf{ od},\ \{x \mapsto 3, y \mapsto 3\}\rangle\\ &\xrightarrow{(ii),(iii)} \langle \textbf{while } 1 \textbf{ do } skip \textbf{ od},\ \{x \mapsto 3, y \mapsto 9\}\rangle\\ &\xrightarrow{(vi)} \langle skip; \textbf{while } 1 \textbf{ do } skip \textbf{ od},\ \{x \mapsto 3, y \mapsto 9\}\rangle\\ &\xrightarrow{(i),(iii)} \langle \textbf{while } 1 \textbf{ do } skip \textbf{ od},\ \{x \mapsto 3, y \mapsto 9\}\rangle\\ &\xrightarrow{(vi)} \ldots\end{aligned}$$

Computations of Deterministic Programs

Definition. Let S be a deterministic program.

(i) A **transition sequence** of S (starting in σ) is a finite or infinite sequence

$$\langle S,\ \sigma\rangle = \langle S_0,\ \sigma_0\rangle \to \langle S_1,\ \sigma_1\rangle \to \ldots$$

(that is, $\langle S_i,\ \sigma_i\rangle$ and $\langle S_{i+1},\ \sigma_{i+1}\rangle$ are in transition relation for all i).

(ii) A **computation (path)** of S (starting in σ) is a **maximal** transition sequence of S (starting in σ), i.e. infinite or not extendible.

(iii) A computation of S is said to
   a) **terminate** in τ if and only if it is finite and ends with $\langle E,\ \tau\rangle$,
   b) **diverge** if and only if it is infinite.
   S **can diverge from** σ if and only if a diverging computation starts in σ.

(iv) We use $\to^*$ to denote the transitive, reflexive closure of $\to$.

Lemma. For each deterministic program S and each state σ, there is exactly one computation of S which starts in σ.

Input/Output Semantics of Deterministic Programs

Definition.
Let S be a deterministic program.

(i) The **semantics of partial correctness** is the function

states / valuation

$$\mathcal{M}[\![S]\!] : \Sigma \to 2^{\Sigma}$$

powerset of Σ

with $\mathcal{M}[\![S]\!](\sigma) = \{\tau \mid \langle S,\ \sigma\rangle \to^* \langle E,\ \tau\rangle\}$.

(ii) The **semantics of total correctness** is the function

$$\mathcal{M}_{tot}[\![S]\!] : \Sigma \to 2^{\Sigma} \,\dot\cup\, \{\infty\}$$

with $\mathcal{M}_{tot}[\![S]\!](\sigma) = \mathcal{M}[\![S]\!](\sigma) \cup \{\infty \mid S \text{ can diverge from } \sigma\}$.
∞ is an error state representing **divergence**.

Note: $\mathcal{M}_{tot}[\![S]\!](\sigma)$ has exactly one element, $\mathcal{M}[\![S]\!](\sigma)$ at most one.

Example: $\mathcal{M}[\![S_1]\!](\sigma) = \mathcal{M}_{tot}[\![S_1]\!](\sigma) = \{\tau \mid \tau(x) = \sigma(x) \wedge \tau(y) = \sigma(x)^2\}, \quad \sigma \in \Sigma.$
(Recall: $S_1 \equiv y := x; y := (x-1)\cdot x + y$)

Content

Correctness of While-Programs

Correctness of Deterministic Programs

Definition.
Let S be a program over variables V, and p and q Boolean expressions over V.

(i) The **correctness formula**

$$\{p\}\ S\ \{q\} \qquad \text{("Hoare triple")}$$

holds in the sense of partial correctness, denoted by $\models \{p\}\ S\ \{q\}$, if and only if

$\{\sigma \mid \sigma \models p\}$

$$\mathcal{M}[\![S]\!]([\![p]\!]) \subseteq [\![q]\!].$$

$\{\tau \mid \tau \models q\}$

We say S is **partially correct** wrt. p and q.

(ii) A **correctness formula**

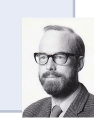
$$\{p\}\ S\ \{q\}$$

holds in the sense of total correctness, denoted by $\models_{tot} \{p\}\ S\ \{q\}$, if and only if

$\not\ni \infty$

$$\mathcal{M}_{tot}[\![S]\!]([\![p]\!]) \subseteq [\![q]\!].$$

We say S is **totally correct** wrt. p and q.



Example: Computing squares (of numbers $0, \dots, 27$)

- **Pre-condition**: $p \equiv 0 \le x \le 27$,
- **Post-condition**: $q \equiv y = x^2$.

Program S_1:

```
int y = x;
y = (x - 1) * x + y;
```

$\models^{?} \{p\}\ S_1\ \{q\}$ ✓ //

$\models^{?}_{tot} \{p\}\ S_1\ \{q\}$ ✓

Program S_2:

```
int y = x;
int z; // uninitialised
y = ((x - 1) * x + y) + z;
```

$\not\models^{?} \{p\}\ S_2\ \{q\}$ ✗

$\not\models^{?}_{tot} \{p\}\ S_2\ \{q\}$ ✗

S_5:

```
S2;
while(1);
```

$\models \{p\}\ S\ \{q\}$ ✓

$\not\models_{tot} \{p\}\ S\ \{q\}$ ✗

Program S_3:

```
int y = x;
y = (x - 1) * x + y;
while (1);
```

$\models^{?} \{p\}\ S_3\ \{q\}$ ✓

$\not\models^{?}_{tot} \{p\}\ S_3\ \{q\}$ ✗

Program S_4:

```
int x = read_input();
int y = x + (x-1) * x;
```

$\models^{?} \{p\}\ S_4\ \{q\}$

$\models^{?}_{tot} \{p\}\ S_4\ \{q\}$

math. $\mathbb{Z}$	64-bit Integer
✓	(✗)
✓	(✗)

Example: Correctness

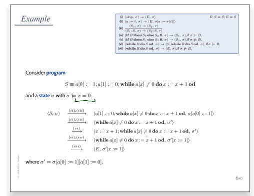
- By the example, **we have shown**

$$\models \{x = 0\}\ S\ \{x = 1\}$$

and

$$\models_{tot} \{x = 0\}\ S\ \{x = 1\}.$$

(because we only assumed $\sigma \models x = 0$ for the example, which is exactly the precondition.) ✓



Example

(i) $\langle skip, \sigma\rangle \to \langle E, \sigma\rangle$ $\quad E; S \equiv S; E \equiv S$
(ii) $\langle u := t, \sigma\rangle \to \langle E, \sigma[u := \sigma(t)]\rangle$
(iii) $\dfrac{\langle S_1, \sigma\rangle \to \langle S_2, \tau\rangle}{\langle S_1; S, \sigma\rangle \to \langle S_2; S, \tau\rangle}$
(iv) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle S; \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \textbf{while } B \textbf{ do } S \textbf{ od}, \sigma\rangle \to \langle E, \sigma\rangle$, if $\sigma \not\models B$,

Consider **program**

$$S \equiv a[0] := 1; a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x + 1 \textbf{ od}$$

and a **state** σ with $\sigma \models x = 0$.

$$\begin{aligned}\langle S, \sigma\rangle &\xrightarrow{(ii),(iii)} \langle a[1] := 0; \textbf{while } a[x] \neq 0 \textbf{ do } x := x+1 \textbf{ od}, \sigma[a[0] := 1]\rangle\\ &\xrightarrow{(ii),(iii)} \langle \textbf{while } a[x] \neq 0 \textbf{ do } x := x+1 \textbf{ od}, \sigma'\rangle\\ &\xrightarrow{(vi)} \langle x := x+1; \textbf{while } a[x] \neq 0 \textbf{ do } x := x+1 \textbf{ od}, \sigma'\rangle\\ &\xrightarrow{(ii),(iii)} \langle \textbf{while } a[x] \neq 0 \textbf{ do } x := x+1 \textbf{ od}, \sigma'[x := 1]\rangle\\ &\xrightarrow{(vii)} \langle E, \sigma'[x := 1]\rangle\end{aligned}$$

where $\sigma' = \sigma[a[0] := 1][a[1] := 0]$.

6/40

- **We have also shown** (= **proved** (!)):

$$\models \{x = 0\}\ S\ \{x = 1 \wedge a[x] = 0\}.$$

- The correctness formula $\{x = 2\}\ S\ \{true\}$ **does not hold** for S.
(For example, if $\sigma \models a[i] \neq 0$ for all $i > 2$.)

- In the sense of **partial correctness**, $\{x = 2 \wedge \forall\, i \ge 2 \bullet a[i] = 1\}\ S\ \{false\}$ also holds.

Content

- **Formal Program Verification**
  - **Deterministic Programs**
    - **Syntax**
    - **Semantics**
    - Termination, Divergence
  - **Correctness** of deterministic programs
    - **partial** correctness,
    - **total** correctness.
  - **Proof System PD**
- **The Verifier for Concurrent C**
  - **modular reasoning**
  - **return values / old values**
- **Assertions**

Proof-System PD

Proof-System PD *(for sequential, deterministic programs)*

Axiom 1: Skip-Statement

$$\{p\}\ skip\ \{p\}$$

Axiom 2: Assignment

$$\{p[u := t]\}\ u := t\ \{p\}$$

Rule 3: Sequential Composition

$$\frac{\{p\}\ S_1\ \{r\}, \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$$

Rule 4: Conditional Statement

$$\frac{\{p \wedge B\}\ S_1\ \{q\}, \{p \wedge \neg B\}\ S_2\ \{q\},}{\{p\}\ \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}\ \{q\}}$$

Rule 5: While-Loop

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while } B \textbf{ do } S \textbf{ od}\ \{p \wedge \neg B\}}$$

Rule 6: Consequence

$$\frac{p \to p_1, \{p_1\}\ S\ \{q_1\}, q_1 \to q}{\{p\}\ S\ \{q\}}$$

Proof-System PD *(for sequential, deterministic programs)*

Axiom 1: Skip-Statement

$$\{p\}\ skip\ \{p\}$$

Axiom 2: Assignment

$$\{p[u := t]\}\ u := t\ \{p\}$$

Rule 3: Sequential Composition

$$\frac{\{p\}\ S_1\ \{r\}, \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$$

Rule 4: Conditional Statement

$$\frac{\{p \wedge B\}\ S_1\ \{q\}, \{p \wedge \neg B\}\ S_2\ \{q\},}{\{p\}\ \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}\ \{q\}}$$

Rule 5: While-Loop

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while } B \textbf{ do } S \textbf{ od}\ \{p \wedge \neg B\}}$$

Rule 6: Consequence

$$\frac{p \to p_1, \{p_1\}\ S\ \{q_1\}, q_1 \to q}{\{p\}\ S\ \{q\}}$$

Theorem. PD is correct ("sound") and (relative) complete for partial correctness of deterministic programs, i.e. $\vdash_{PD} \{p\}\ S\ \{q\}$ if and only if $\models \{p\}\ S\ \{q\}$.

Example Proof

$$DIV \equiv \overbrace{a := 0;\ b := x}^{=:S_0^D};\ \mathbf{while}\ \overbrace{b \geq y}^{=:B^D}\ \mathbf{do}\ \overbrace{b := b - y;\ a := a + 1}^{=:S_1^D}\ \mathbf{od}$$

(2) $B \wedge P_1$ $P_1 \to q$ (1) (3)

(The first (textually represented) program that has been formally verified (Hoare, 1969).

We can prove $\models \{x \geq 0 \wedge y \geq 0\}\ DIV\ \{a \cdot y + b = x \wedge b < y\}$

by showing $\vdash_{PD} \{\underbrace{x \geq 0 \wedge y \geq 0}_{=:p^D}\}\ DIV\ \{\underbrace{a \cdot y + b = x \wedge b < y}_{=:q^D}\}$, i.e., derivability in PD:

$$\cfrac{\cfrac{(1)}{\{p^D\}\ S_0^D\ \{P\},} \quad \cfrac{\cfrac{}{P \to P,} \quad \cfrac{\cfrac{(2)}{\{P \wedge (B^D)\}\ S_1^D\ \{P\}}}{\{P\}\ \mathbf{while}\ B^D\ \mathbf{do}\ S_1^D\ \mathbf{od}\ \{P \wedge \neg(B^D)\},}\text{(R5)} \quad \cfrac{(3)}{P \wedge \neg(B^D) \to q^D}}{\{P\}\ \mathbf{while}\ B^D\ \mathbf{do}\ S_1^D\ \mathbf{od}\ \{q^D\}}\text{(R6)}}{\{p^D\}\ S_0^D;\ \mathbf{while}\ B^D\ \mathbf{do}\ S_1^D\ \mathbf{od}\ \{q^D\}}\text{(R3)}$$

(A1) $\{p\}\ skip\ \{p\}$ (R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

Example Proof

$$DIV \equiv \overbrace{a := 0;\ b := x}^{=:S_0^D};\ \mathbf{while}\ \overbrace{b \geq y}^{=:B^D}\ \mathbf{do}\ \overbrace{b := b - y;\ a := a + 1}^{=:S_1^D}\ \mathbf{od}$$

(The first (textually represented) program that has been formally verified (Hoare, 1969).

We can prove $\models \{x \geq 0 \wedge y \geq 0\}\ DIV\ \{a \cdot y + b = x \wedge b < y\}$

by showing $\vdash_{PD} \{\underbrace{x \geq 0 \wedge y \geq 0}_{=:p^D}\}\ DIV\ \{\underbrace{a \cdot y + b = x \wedge b < y}_{=:q^D}\}$, i.e., derivability in PD:

$$\cfrac{\cfrac{(1)}{\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\},} \quad \cfrac{\cfrac{}{P \to P,} \quad \cfrac{\cfrac{(2)}{\{P \wedge (b \geq y)\}\ b := b - y;\ a := a + 1\ \{P\}}}{\{P\}\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}\ \{P \wedge \neg(b \geq y)\},}\text{(R5)} \quad \cfrac{(3)}{P \wedge \neg(b \geq y) \to a \cdot y + b = x \wedge b < y}}{\{P\}\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}\ \{a \cdot y + b = x \wedge b < y\}}\text{(R6)}}{\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x;\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}\ \{a \cdot y + b = x \wedge b < y\}}\text{(R3)}$$

(A1) $\{p\}\ skip\ \{p\}$ (R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

Example Proof Cont'd

$$\cfrac{\cfrac{(1)}{\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\},} \quad \cfrac{\cfrac{}{P \to P,} \quad \cfrac{\cfrac{(2)}{\{P \wedge (b \geq y)\}\ b := b - y;\ a := a + 1\ \{P\}}}{\{P\}\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}\ \{P \wedge \neg(b \geq y)\},}\text{(R5)} \quad \cfrac{(3)}{P \wedge \neg(b \geq y) \to a \cdot y + b = x \wedge b < y}}{\{P\}\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}\ \{a \cdot y + b = x \wedge b < y\}}\text{(R6)}}{\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x;\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}\ \{a \cdot y + b = x \wedge b < y\}}\text{(R3)}$$

In the following, we show

(1) $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$,

(2) $\vdash_{PD} \{P \wedge b \geq y\}\ b := b - y;\ a := a + 1\ \{P\}$,

(3) $\models P \wedge \neg(b \geq y) \to a \cdot y + b = x \wedge b < y$.

As **loop invariant**, we choose (**creative act!**):

$$P \equiv a \cdot y + b = x \wedge b \geq 0$$

Proof of (1)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(1)** claims:
  $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$
  where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

Proof of (1)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(1)** claims:
  $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$
  where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

- $\vdash_{PD} \underbrace{\{0 \cdot y + x = x \wedge x \geq 0\}}_{p[a:=0]}\ a := 0\ \underbrace{\{a \cdot y + x = x \wedge x \geq 0\}}_{p}$ by (A2),

  "a := 0"

Proof of (1)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(1)** claims:
  $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$
  where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

- $\vdash_{PD} \{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0\ \{a \cdot y + x = x \wedge x \geq 0\}$ by (A2),

- $\vdash_{PD} \{a \cdot y + x = x \wedge x \geq 0\}\ b := x\ \{\underbrace{a \cdot y + b = x \wedge b \geq 0}_{\equiv P}\}$ by (A2),

Proof of (1)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- (1) claims:
  $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$
  where $P \equiv a \cdot y + b = x \wedge b \geq 0$.
- $\vdash_{PD} \{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0\ \{a \cdot y + x = x \wedge x \geq 0\}$ by (A2),
- $\vdash_{PD} \{a \cdot y + x = x \wedge x \geq 0\}\ b := x\ \{\underbrace{a \cdot y + b = x \wedge b \geq 0}_{\equiv P}\}$ by (A2),
- thus, $\vdash_{PD} \{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0;\ b := x\ \{P\}$ by (R3),

Proof of (1)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- (1) claims:
  $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$
  where $P \equiv a \cdot y + b = x \wedge b \geq 0$.
- $\vdash_{PD} \{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0\ \{a \cdot y + x = x \wedge x \geq 0\}$ by (A2),
- $\vdash_{PD} \{a \cdot y + x = x \wedge x \geq 0\}\ b := x\ \{\underbrace{a \cdot y + b = x \wedge b \geq 0}_{\equiv P}\}$ by (A2),
- thus, $\vdash_{PD} \{0 \cdot y + x = x \wedge x \geq 0\}\ a := 0;\ b := x\ \{P\}$ by (R3),
- using $x \geq 0 \wedge y \geq 0 \to 0 \cdot y + x = x \wedge x \geq 0$ and $P \to P$, we obtain

$$\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$$

by (R6). □

Substitution

The rule '**Assignment**' uses (syntactical) **substitution**: $\{p[u := t]\}\ u := t\ \{p\}$

(In formula p, replace all (free) occurences of (program or logical) variable u by term t.)

Defined as usual, only **indexed** and **bound** variables need to be treated specially:

Substitution

The rule '**Assignment**' uses (syntactical) **substitution**: $\{p[u := t]\}\ u := t\ \{p\}$

(In formula p, replace all (free) occurences of (program or logical) variable u by term t.)

Defined as usual, only **indexed** and **bound** variables need to be treated specially:

Expressions:

- plain variable x: $x[u := t] \equiv \begin{cases} t & \text{, if } x = u \\ x & \text{, otherwise} \end{cases}$
- constant c:
  $c[u := t] \equiv c$.
- constant op, terms s_i:
  $op(s_1, \ldots, s_n)[u := t]$
  $\equiv op(s_1[u := t], \ldots, s_n[u := t])$.
- conditional expression:
  $(B\ ?\ s_1 : s_2)[u := t]$
  $\equiv (B[u := t]\ ?\ s_1[u := t] : s_2[u := t])$

Substitution

The rule '**Assignment**' uses (syntactical) **substitution**: $\{p[u := t]\}\ u := t\ \{p\}$

(In formula p, replace all (free) occurences of (program or logical) variable u by term t.)

Defined as usual, only **indexed** and **bound** variables need to be treated specially:

Expressions:

- plain variable x: $x[u := t] \equiv \begin{cases} t & \text{, if } x = u \\ x & \text{, otherwise} \end{cases}$
- constant c:
  $c[u := t] \equiv c$.
- constant op, terms s_i:
  $op(s_1, \ldots, s_n)[u := t]$
  $\equiv op(s_1[u := t], \ldots, s_n[u := t])$.
- conditional expression:
  $(B\ ?\ s_1 : s_2)[u := t]$
  $\equiv (B[u := t]\ ?\ s_1[u := t] : s_2[u := t])$

Formulae:

- boolean expression $p \equiv s$:
  $p[u := t] \equiv s[u := t]$
- negation:
  $(\neg q)[u := t] \equiv \neg(q[u := t])$
- conjunction etc.:
  $(q \wedge r)[u := t]$
  $\equiv q[u := t] \wedge r[u := t]$

Substitution

The rule '**Assignment**' uses (syntactical) **substitution**: $\{p[u := t]\}\ u := t\ \{p\}$

(In formula p, replace all (free) occurences of (program or logical) variable u by term t.)

Defined as usual, only **indexed** and **bound** variables need to be treated specially:

Expressions:

- plain variable x: $x[u := t] \equiv \begin{cases} t & \text{, if } x = u \\ x & \text{, otherwise} \end{cases}$
- constant c:
  $c[u := t] \equiv c$.
- constant op, terms s_i:
  $op(s_1, \ldots, s_n)[u := t]$
  $\equiv op(s_1[u := t], \ldots, s_n[u := t])$.
- conditional expression:
  $(B\ ?\ s_1 : s_2)[u := t]$
  $\equiv (B[u := t]\ ?\ s_1[u := t] : s_2[u := t])$

Formulae:

- boolean expression $p \equiv s$:
  $p[u := t] \equiv s[u := t]$
- negation:
  $(\neg q)[u := t] \equiv \neg(q[u := t])$
- conjunction etc.:
  $(q \wedge r)[u := t]$
  $\equiv q[u := t] \wedge r[u := t]$
- **quantifier**:
  $(\forall x : q)[u := t] \equiv \forall y : q[x := y][u := t]$
  y fresh (not in q, t, u), same type as x.

Substitution

The rule '**Assignment**' uses (syntactical) **substitution**: $\{p[u := t]\}\ u := t\ \{p\}$

(In formula p, replace all (free) occurences of (program or logical) variable u by term t.)

Defined as usual, only **indexed** and **bound** variables need to be treated specially:

Expressions:

- plain variable x: $x[u := t] \equiv \begin{cases} t & \text{, if } x = u \\ x & \text{, otherwise} \end{cases}$
- constant c: $c[u := t] \equiv c$.
- constant op, terms s_i: $op(s_1, \dots, s_n)[u := t] \equiv op(s_1[u := t], \dots, s_n[u := t])$.
- conditional expression: $(B\ ?\ s_1 : s_2)[u := t] \equiv (B[u := t]\ ?\ s_1[u := t] : s_2[u := t])$

Formulae:

- boolean expression $p \equiv s$: $p[u := t] \equiv s[u := t]$
- negation: $(\neg q)[u := t] \equiv \neg(q[u := t])$
- conjunction etc.: $(q \wedge r)[u := t] \equiv q[u := t] \wedge r[u := t]$
- **quantifier**: $(\forall x : q)[u := t] \equiv \forall y : q[x := y][u := t]$ y fresh (not in q, t, u), same type as x.

- **indexed variable**, u plain or $u \equiv b[t_1, \dots, t_m]$ and $a \neq b$:
$$(a[s_1, \dots, s_n])[u := t] \equiv a[s_1[u := t], \dots, s_n[u := t]])$$
- **indexed variable**, $u \equiv a[t_1, \dots, t_m]$:
$$(a[s_1, \dots, s_n])[u := t] \equiv (\bigwedge_{i=1}^{n} s_i[u := t] = t_i\ ?\ t : a[s_1[u := t], \dots, s_n[u := t]])$$

Example Proof Cont'd

$$\dfrac{\dfrac{(1)}{\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\},} \quad \dfrac{\dfrac{}{P \to P,} \quad \dfrac{\dfrac{(2)}{\{P \wedge (b \geq y)\}\ b := b - y;\ a := a + 1\ \{P\}}}{\{P\}\ \textbf{while}\ b \geq y\ \textbf{do}\ b := b - y;\ a := a + 1\ \textbf{od}\ \{P \wedge \neg(b \geq y)\},}\text{(R5)} \quad \dfrac{(3)}{P \wedge \neg(b \geq y) \to a \cdot y + b = x \wedge b < y}}{\{P\}\ \textbf{while}\ b \geq y\ \textbf{do}\ b := b - y;\ a := a + 1\ \textbf{od}\ \{a \cdot y + b = x \wedge b < y\}}\text{(R6)}}{\{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x;\ \textbf{while}\ b \geq y\ \textbf{do}\ b := b - y;\ a := a + 1\ \textbf{od}\ \{a \cdot y + b = x \wedge b < y\}}\text{(R3)}$$

In the following, we show

(1) $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ a := 0;\ b := x\ \{P\}$,

(2) $\vdash_{PD} \{P \wedge b \geq y\}\ b := b - y;\ a := a + 1\ \{P\}$,

(3) $\models P \wedge \neg(b \geq y) \to a \cdot y + b = x \wedge b < y$.

As **loop invariant**, we choose (**creative act!**):

$$P \equiv a \cdot y + b = x \wedge b \geq 0$$

(A1) $\{p\}\ skip\ \{p\}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$

(R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$

(R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$

(R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

Proof of (2)

(A1) $\{p\}\ skip\ \{p\}$

(R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$

(R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$

(R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(2)** claims:
$\vdash_{PD} \{P \wedge b \geq y\}\ b := b - y;\ a := a + 1\ \{P\}$
where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

Proof of (2)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(2)** claims:
$\vdash_{PD} \{P \wedge b \geq y\}\ b := b - y;\ a := a + 1\ \{P\}$
where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

- $\vdash_{PD} \{(a+1) \cdot y + (b-y) = x \wedge (b-y) \geq 0\}\ b := b - y\ \{(a+1) \cdot y + b = x \wedge b \geq 0\}$
by (A2),

Proof of (2)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(2)** claims:
$\vdash_{PD} \{P \wedge b \geq y\}\ b := b - y;\ a := a + 1\ \{P\}$
where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

- $\vdash_{PD} \{(a+1) \cdot y + (b-y) = x \wedge (b-y) \geq 0\}\ b := b - y\ \{(a+1) \cdot y + b = x \wedge b \geq 0\}$
by (A2),

- $\vdash_{PD} \{(a+1) \cdot y + b = x \wedge b \geq 0\}\ a := a + 1\ \{\underbrace{a \cdot y + b = x \wedge b \geq 0}_{\equiv P}\}$ by (A2),

Proof of (2)

(A1) $\{p\}\ skip\ \{p\}$ (R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$

(A2) $\{p[u := t]\}\ u := t\ \{p\}$ (R5) $\dfrac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$

(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ (R6) $\dfrac{p \to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \to q}{\{p\}\ S\ \{q\}}$

- **(2)** claims:
$\vdash_{PD} \{P \wedge b \geq y\}\ b := b - y;\ a := a + 1\ \{P\}$
where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

- $\vdash_{PD} \{(a+1) \cdot y + (b-y) = x \wedge (b-y) \geq 0\}\ b := b - y\ \{(a+1) \cdot y + b = x \wedge b \geq 0\}$
by (A2),

- $\vdash_{PD} \{(a+1) \cdot y + b = x \wedge b \geq 0\}\ a := a + 1\ \{\underbrace{a \cdot y + b = x \wedge b \geq 0}_{\equiv P}\}$ by (A2),

- $\vdash_{PD} \{(a+1) \cdot y + (b-y) = x \wedge (b-y) \geq 0\}\ b := b - y;\ a := a + 1\ \{P\}$ by (R3),

Proof of (2)

> **(A1)** $\{p\}\ skip\ \{p\}$ **(R4)** $\dfrac{\{p\wedge B\}\ S_1\ \{q\},\ \{p\wedge\neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$
>
> **(A2)** $\{p[u:=t]\}\ u:=t\ \{p\}$ **(R5)** $\dfrac{\{p\wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p\wedge\neg B\}}$
>
> **(R3)** $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ **(R6)** $\dfrac{p\to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1\to q}{\{p\}\ S\ \{q\}}$

- **(2)** claims:
  $\vdash_{PD}\ \{P\wedge b\ge y\}\ b:=b-y;\ a:=a+1\ \{P\}$
  where $P\equiv a\cdot y+b=x\wedge b\ge 0$.

- $\vdash_{PD}\ \{(a+1)\cdot y+(b-y)=x\wedge(b-y)\ge 0\}\ b:=b-y\ \{(a+1)\cdot y+b=x\wedge b\ge 0\}$
  by (A2),
- $\vdash_{PD}\ \{(a+1)\cdot y+b=x\wedge b\ge 0\}\ a:=a+1\ \{\underbrace{a\cdot y+b=x\wedge b\ge 0}_{\equiv P}\}$ by (A2),
- $\vdash_{PD}\ \{(a+1)\cdot y+(b-y)=x\wedge(b-y)\ge 0\}\ b:=b-y;\ a:=a+1\ \{P\}$ by (R3),
- using $P\wedge b\ge y\to(a+1)\cdot y+(b-y)=x\wedge(b-y)\ge 0$ and $P\to P$ we obtain,

$$\vdash_{PD}\ \{P\wedge b\ge y\}\ b:=b-y;\ a:=a+1\ \{P\}$$

by (R6). □

Example Proof Cont'd

$$\dfrac{\dfrac{(1)}{\{x\ge 0\wedge y\ge 0\}\ a:=0;\ b:=x\ \{P\},}\quad \dfrac{\dfrac{}{P\to P,}\quad \dfrac{\dfrac{(2)}{\{P\wedge(b\ge y)\}\ b:=b-y;\ a:=a+1\ \{P\}}}{\{P\}\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}\ \{P\wedge\neg(b\ge y)\},}\ (R5)\quad \dfrac{(3)}{P\wedge\neg(b\ge y)\to a\cdot y+b=x\wedge b<y}}{\{P\}\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}\ \{a\cdot y+b=x\wedge b<y\}}\ (R6)}{\{x\ge 0\wedge y\ge 0\}\ a:=0;\ b:=x;\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}\ \{a\cdot y+b=x\wedge b<y\}}\ (R3)$$

In the following, we show

(1) $\vdash_{PD}\ \{x\ge 0\wedge y\ge 0\}\ a:=0;\ b:=x\ \{P\}$,

(2) $\vdash_{PD}\ \{P\wedge b\ge y\}\ b:=b-y;\ a:=a+1\ \{P\}$,

(3) $\models P\wedge\neg(b\ge y)\to a\cdot y+b=x\wedge b<y$.

As **loop invariant**, we choose (**creative act!**):

$$P\equiv a\cdot y+b=x\wedge b\ge 0$$

> **(A1)** $\{p\}\ skip\ \{p\}$ **(R3)** $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$ **(R5)** $\dfrac{\{p\wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p\wedge\neg B\}}$
>
> **(A2)** $\{p[u:=t]\}\ u:=t\ \{p\}$ **(R4)** $\dfrac{\{p\wedge B\}\ S_1\ \{q\},\ \{p\wedge\neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$ **(R6)** $\dfrac{p\to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1\to q}{\{p\}\ S\ \{q\}}$

Proof of (3)

(3) claims

$$\models P\wedge\neg(b\ge y)\to a\cdot y+b=x\wedge b<y.$$

where $P\equiv a\cdot y+b=x\wedge b\ge 0$.

Proof: easy.

Back to the Example Proof

We have shown:

(1) $\vdash_{PD}\ \{x\ge 0\wedge y\ge 0\}\ a:=0;\ b:=x\ \{P\}$,

(2) $\vdash_{PD}\ \{P\wedge b\ge y\}\ b:=b-y;\ a:=a+1\ \{P\}$,

(3) $\models P\wedge\neg(b\ge y)\to a\cdot y+b=x\wedge b<y$.

and

$$\dfrac{\dfrac{(1)}{\{x\ge 0\wedge y\ge 0\}\ a:=0;\ b:=x\ \{P\},}\quad \dfrac{\dfrac{}{P\to P,}\quad \dfrac{\dfrac{(2)}{\{P\wedge(b\ge y)\}\ b:=b-y;\ a:=a+1\ \{P\}}}{\{P\}\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}\ \{P\wedge\neg(b\ge y)\},}\ (R5)\quad \dfrac{(3)}{P\wedge\neg(b\ge y)\to a\cdot y+b=x\wedge b<y}}{\{P\}\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}\ \{a\cdot y+b=x\wedge b<y\}}\ (R6)}{\{x\ge 0\wedge y\ge 0\}\ a:=0;\ b:=x;\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}\ \{a\cdot y+b=x\wedge b<y\}}\ (R3)$$

thus

$$\vdash_{PD}\ \{x\ge 0\wedge y\ge 0\}\ \underbrace{a:=0;\ b:=x;\ \mathbf{while}\ b\ge y\ \mathbf{do}\ b:=b-y;\ a:=a+1\ \mathbf{od}}_{\equiv DIV}\ \{a\cdot y+b=x\wedge b<y\}$$

and thus (since PD is sound) DIV is **partially correct** wrt.

- **pre-condition**: $x\ge 0\wedge y\ge 0$,
- **post-condition**: $a\cdot y+b=x\wedge b<y$.

IOW: whenever DIV is called with x and y such that $x\ge 0\wedge y\ge 0$,
then (if DIV terminates) $a\cdot y+b=x\wedge b<y$ will hold.

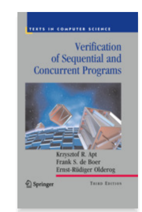
Once Again

> **(A1)** $\{p\}\ skip\ \{p\}$
>
> **(A2)** $\{p[u:=t]\}\ u:=t\ \{p\}$
>
> **(R3)** $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$
>
> **(R4)** $\dfrac{\{p\wedge B\}\ S_1\ \{q\},\ \{p\wedge\neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$
>
> **(R5)** $\dfrac{\{p\wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p\wedge\neg B\}}$
>
> **(R6)** $\dfrac{p\to p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1\to q}{\{p\}\ S\ \{q\}}$

- $P\equiv a\cdot y+b=x\wedge b\ge 0$

  $\{x\ge 0\wedge y\ge 0\}$
  $\{0\cdot y+x=x\wedge x\ge 0\}$
- $a:=0;$
  $\{a\cdot y+x=x\wedge x\ge 0\}$
- $b:=x;$
  $\{a\cdot y+b=x\wedge b\ge 0\}$
  $\{P\}$
- **while** $b\ge y$ **do**
  $\{P\wedge b\ge y\}$
  $\{(a+1)\cdot y+(b-y)=x\wedge(b-y)\ge 0\}$
- $b:=b-y;$
  $\{(a+1)\cdot y+b=x\wedge b\ge 0\}$
- $a:=a+1$
  $\{a\cdot y+b=x\wedge b\ge 0\}$
  $\{P\}$
- **od**
  $\{P\wedge\neg(b\ge y)\}$
  $\{a\cdot y+b=x\wedge b<y\}$

Literature Recommendation



Content

- **Formal Program Verification**
  - **Deterministic Programs**
    - **Syntax**
    - **Semantics**
    - Termination, Divergence
  - **Correctness** of deterministic programs
    - **partial** correctness,
    - **total** correctness.
  - **Proof System PD**
- **The Verifier for Concurrent C**
  - **modular reasoning**
  - **return values / old values**
- **Assertions**

The Verifier for Concurrent C

VCC

- The **Verifier for Concurrent C** (VCC) basically implements Hoare-style reasoning.
- **Special syntax:**
  - `#include <vcc.h>`
  - `_(requires` p`)` — **pre-condition**, p is (basically) a C expression
  - `_(ensures` q`)` — **post-condition**, q is (basically) a C expression
  - `_(invariant` $expr$`)` — **loop invariant**, $expr$ is (basically) a C expression
  - `_(assert` p`)` — **intermediate invariant**, p is (basically) a C expression
  - `_(writes &v)` — VCC considers **concurrent** C programs; we need to declare for each procedure which global variables it is allowed to write to (also checked by VCC)
- **Special expressions:**
  - `\thread_local(&v)` — no other thread writes to variable v (in pre-conditions)
  - `\old(v)` — the value of v when procedure was called (useful for post-conditions)
  - `\result` — return value of procedure (useful for post-conditions)

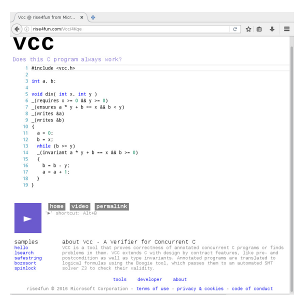
VCC Syntax Example

```c
#include <vcc.h>

int a, b;

void div( int x, int y )
  _(requires x >= 0 && y >= 0)
  _(ensures a * y + b == x && b < y)
  _(writes &a)
  _(writes &b)
{
  a = 0;
  b = x;
  while (b >= y)
  _(invariant a * y + b == x && b >= 0)
  {
    b = b - y;
    a = a + 1;
  }
}
```

$$DIV \equiv a := 0;\ b := x;\ \mathbf{while}\ b \geq y\ \mathbf{do}\ b := b - y;\ a := a + 1\ \mathbf{od}$$

$$\{x \geq 0 \wedge y \geq 0\}\ DIV\ \{x \geq 0 \wedge y \geq 0\}$$

VCC Web-Interface



Example program DIV: `http://rise4fun.com/Vcc/4Kqe`

Interpretation of Results

- VCC result: "**verification succeeded**"

- VCC result: "**verification failed**"

- Other case: "**timeout**" etc.

Interpretation of Results

- VCC result: "**verification succeeded**"
  - We can **only** conclude that the tool
    — under its interpretation of the C-standard, under its platform assumptions (32-bit), etc. —
    claims that there is a proof for $\models \{p\}\ DIV\ \{q\}$.
- VCC result: "**verification failed**"
- Other case: "**timeout**" etc.

Interpretation of Results

- VCC result: "**verification succeeded**"
  - We can **only** conclude that the tool
    — under its interpretation of the C-standard, under its platform assumptions (32-bit), etc. —
    claims that there is a proof for $\models \{p\}\ DIV\ \{q\}$.
  - May be due to an error in the tool! (That's a **false negative** then.)
    Yet we can ask **for a printout of the proof** and check it manually
    (hardly possible in practice) or with other tools like interactive theorem provers.
- VCC result: "**verification failed**"
- Other case: "**timeout**" etc.

Interpretation of Results

- VCC result: "**verification succeeded**"
  - We can **only** conclude that the tool
    — under its interpretation of the C-standard, under its platform assumptions (32-bit), etc. —
    claims that there is a proof for $\models \{p\}\ DIV\ \{q\}$.
  - May be due to an error in the tool! (That's a **false negative** then.)
    Yet we can ask **for a printout of the proof** and check it manually
    (hardly possible in practice) or with other tools like interactive theorem provers.
  - **Note:** $\models \{\mathit{false}\}\ f\ \{q\}$ **always** holds.
    That is, a **mistake** in writing down the pre-condition can make errors in the program go undetected!
- VCC result: "**verification failed**"
- Other case: "**timeout**" etc.

Interpretation of Results

- VCC result: "**verification succeeded**"
  - We can **only** conclude that the tool
    — under its interpretation of the C-standard, under its platform assumptions (32-bit), etc. —
    claims that there is a proof for $\models \{p\}\ DIV\ \{q\}$.
  - May be due to an error in the tool! (That's a **false negative** then.)
    Yet we can ask **for a printout of the proof** and check it manually
    (hardly possible in practice) or with other tools like interactive theorem provers.
  - **Note:** $\models \{\mathit{false}\}\ f\ \{q\}$ **always** holds.
    That is, a **mistake** in writing down the pre-condition can make errors in the program go undetected!
- VCC result: "**verification failed**"
  - May be a **false positive** (wrt. the goal of finding errors).
    The tool **does not provide counter-examples** in the form of a computation path,
    it (only) gives **hints on input values** satisfying p and causing a violation of q.
- Other case: "**timeout**" etc.

Interpretation of Results

- VCC result: "**verification succeeded**"
  - We can **only** conclude that the tool
    — under its interpretation of the C-standard, under its platform assumptions (32-bit), etc. —
    claims that there is a proof for $\models \{p\}\ DIV\ \{q\}$.
  - May be due to an error in the tool! (That's a **false negative** then.)
    Yet we can ask **for a printout of the proof** and check it manually
    (hardly possible in practice) or with other tools like interactive theorem provers.
  - **Note:** $\models \{\mathit{false}\}\ f\ \{q\}$ **always** holds.
    That is, a **mistake** in writing down the pre-condition can make errors in the program go undetected!
- VCC result: "**verification failed**"
  - May be a **false positive** (wrt. the goal of finding errors).
    The tool **does not provide counter-examples** in the form of a computation path,
    it (only) gives **hints on input values** satisfying p and causing a violation of q.
  - $\rightarrow$ try to construct a (true) counter-example from the hints.
    or: make loop-invariant(s) (or pre-condition p) stronger, and try again.
- Other case: "**timeout**" etc.

Interpretation of Results

- VCC result: "**verification succeeded**"
  - We can **only** conclude that the tool
    — under its interpretation of the C-standard, under its platform assumptions (32-bit), etc. —
    claims that there is a proof for $\models \{p\}\ DIV\ \{q\}$.
  - May be due to an error in the tool! (That's a **false negative** then.)
    Yet we can ask **for a printout of the proof** and check it manually
    (hardly possible in practice) or with other tools like interactive theorem provers.
  - **Note:** $\models \{\mathit{false}\}\ f\ \{q\}$ **always** holds.
    That is, a **mistake** in writing down the pre-condition can make errors in the program go undetected!
- VCC result: "**verification failed**"
  - May be a **false positive** (wrt. the goal of finding errors).
    The tool **does not provide counter-examples** in the form of a computation path,
    it (only) gives **hints on input values** satisfying p and causing a violation of q.
  - $\rightarrow$ try to construct a (true) counter-example from the hints.
    or: make loop-invariant(s) (or pre-condition p) stronger, and try again.
- Other case: "**timeout**" etc. — completely **inconclusive** outcome.

VCC Features

- For the exercises, we use VCC only for **sequential, single-thread programs**.
- VCC checks a number of **implicit assertions**:
  - **no arithmetic overflow** in expressions (according to C-standard),
  - **array-out-of-bounds access**,
  - **NULL-pointer dereference**,
  - and many more.

VCC Features

- For the exercises, we use VCC only for **sequential, single-thread programs**.
- VCC checks a number of **implicit assertions**:
  - **no arithmetic overflow** in expressions (according to C-standard),
  - **array-out-of-bounds access**,
  - **NULL-pointer dereference**,
  - and many more.

- Verification **does not always succeed**:
  - The backend SMT-solver may not be able to discharge proof-obligations (in particular non-linear multiplication and division are challenging);
  - In many cases, we need to provide **loop invariants** manually.

VCC Features

- For the exercises, we use VCC only for **sequential, single-thread programs**.
- VCC checks a number of **implicit assertions**:
  - **no arithmetic overflow** in expressions (according to C-standard),
  - **array-out-of-bounds access**,
  - **NULL-pointer dereference**,
  - and many more.
- Verification **does not always succeed**:
  - The backend SMT-solver may not be able to discharge proof-obligations (in particular non-linear multiplication and division are challenging);
  - In many cases, we need to provide **loop invariants** manually.

- VCC also supports:
  - **concurrency**: different threads may write to shared global variables; VCC can check whether concurrent access to shared variables is properly managed;
  - **data structure invariants**: we may declare invariants that have to hold for, e.g., records (e.g. the length field *l* is always equal to the length of the string field str); those invariants may **temporarily** be violated when updating the data structure.
  - and much more.

Modular Reasoning

Modular Reasoning

We can add another rule for calls of functions $f : F$ (simplest case: only global variables):

$$\text{(R7)}\ \frac{\{p\}\ F\ \{q\}}{\{p\}\ f()\ \{q\}}$$

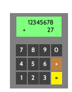
"If we have $\vdash \{p\}\ F\ \{q\}$ for the **implementation** of function f, then if f is **called** in a state satisfying p, the state after return of f will satisfy q."

p is called **pre-condition** and q is called **post-condition** of f.

Example: if we have

- $\{true\}$ `read_number` $\{0 \leq result < 10^8\}$
- $\{0 \leq x \wedge 0 \leq y\}$ `add` $\{(old(x) + old(y) < 10^8 \wedge result = old(x) + old(y)) \vee result < 0\}$
- $\{true\}$ `display` $\{(0 \leq old(sum) < 10^8 \implies \text{"}old(sum)\text{"}) \wedge (old(sum) < 0 \implies \text{"-E-"})\}$

we may be able to prove our pocket calculator correct.




```
int main() {

  while (true) {
    int x = read_number();
    int y = read_number();

    int sum = add( x, y );

    display(sum);
  }
}
```


Return Values and Old Values

- For **modular reasoning**, it's often useful to refer in the post-condition to
  - the **return value** as $result$,
  - the **values** of variable x **at calling time** as $old(x)$.
- Can be defined using **auxiliary variables**:
  - Transform function

$$T\ f()\ \{\ldots; \mathbf{return}\ expr;\}$$

(over variables $V = \{v_1, \ldots, v_n\}$; where $result, v_i^{old} \notin V$) into

$$\begin{array}{l} T\ f()\ \{ \\ \quad v_1^{old} := v_1; \ldots; v_n^{old} := v_n; \\ \quad \ldots; \\ \quad result := expr; \\ \quad \mathbf{return}\ result; \\ \} \end{array}$$

over $V' = V \cup \{v^{old} \mid v \in V\} \cup \{result\}$.

- Then $old(x)$ is just an abbreviation for x^{old}.

Assertions

Assertions

- Extend the **syntax** of **deterministic programs** by

$$S := \cdots \mid \mathbf{assert}(B)$$

- and the **semantics** by rule

$$\langle \mathbf{assert}(B), \sigma \rangle \to \langle E, \sigma \rangle \text{ if } \sigma \models B.$$

(If the asserted boolean expression B does not hold in state σ, the empty program is not reached; otherwise the assertion remains in the first component: **abnormal** program termination).

Extend PD by axiom:

$$\text{(A7)}\ \{p\}\ \mathbf{assert}(p)\ \{p\}$$

- That is, if p holds **before** the assertion, then we can **continue** with the derivation in PD.

  If p does not hold, we **"get stuck"** (and cannot complete the derivation).

- So we **cannot** derive $\{\mathit{true}\}\ x := 0;\ \mathtt{assert}(x = 27)\ \{\mathit{true}\}$ in PD.

Tell Them What You've Told Them. . .

Formal Verification:

- **Program verification** is **another approach** to software quality assurance.
- **Proof System PD** can be used
  - to **prove**
  - that a given program is
  - **correct** wrt. its specification.

  This approach considers **all inputs** inside the specification!
- Tools like **VCC** implement this approach.

References

References

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.