Softwaretechnik / Software-Engineering

Lecture 16: Software Verification

2017-07-20

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

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Code Quality Assurance: Content

VL 14
- **Introduction and Vocabulary**
  - Test case, test suite, test execution.
  - Positive and negative outcomes.

⋮

VL 15
- **Limits of Software Testing**
- **Glass-Box Testing**
  - Statement-, branch-, term-**coverage**.

⋮

VL 16
- **Testing: Rest**
  - **When to stop testing?**
  - **Model-based testing**
  - **Testing in the development process**

⋮

- **Program Verification**
  - partial and total **correctness**,
  - **Proof System PD**.

VL 17
- **Other Approaches**
  - **Runtime verification**.
  - **Review**

⋮

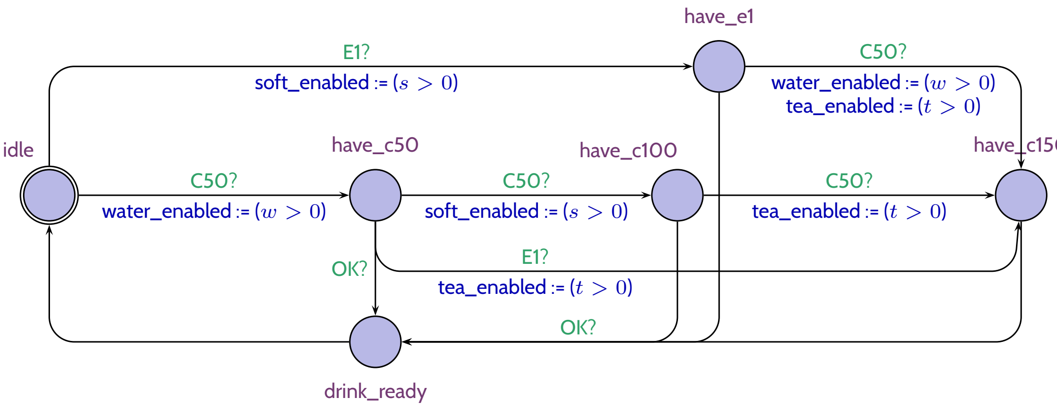
- Software quality assurance wrap-up

Content

- **Testing**: Rest
  - **Model-Based Testing**
  - **When To Stop Testing?**
  - **Testing in the Development Process**
- **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**

Model-Based Testing

Model-based Testing



- Does some software **implement** the given CFA model of the CoinValidator?

- **One approach**: **Location Coverage**.

  Check whether for **each location** of the model there is
  a **corresponding configuration** reachable in the software (needs to be observable somehow).

- Input sequences can **automatically be generated** from the model,
  e.g., using Uppaal's "drive-to" feature.
  - Check "can we reach 'idle', 'have_c50', 'have_c100', 'have_c150'?" by

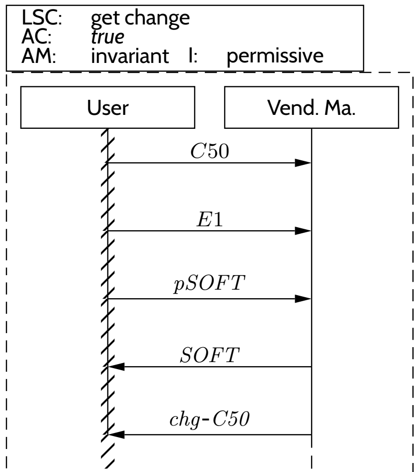
    $$T_1 = (\text{C50}, \text{C50}, \text{C50}; \{\pi \mid \exists\, i < j < k < \ell \bullet \pi^i \sim \text{idle}, \pi^j \sim \text{h\_c50}, \pi^k \sim \text{h\_c100}, \pi^\ell \sim \text{h\_c150}\})$$

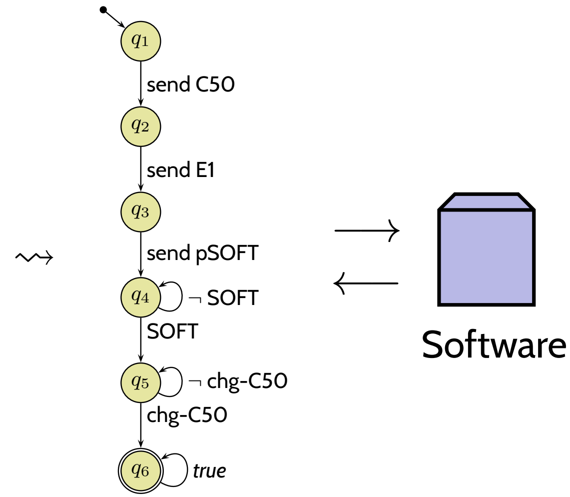
  - Check for 'have_e1' by $T_2 = (\text{C50}, \text{C50}, \text{C50}; \dots)$.
  - To check for 'drink_ready', more interaction is necessary.

- **Analogously**: **Edge Coverage**.

  Check whether **each edge** of the model has **corresponding** behaviour in the software.

Existential LSCs as Test Driver & Monitor (Lettrari and Klose, 2001)

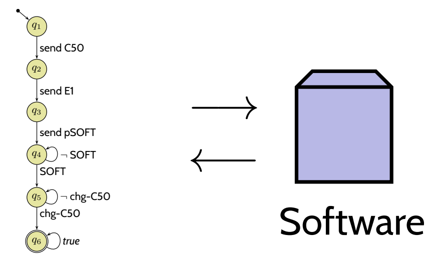




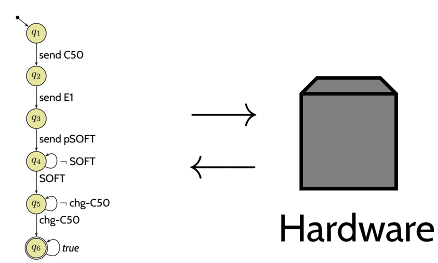
- If the LSC has designated **environment instance** lines, we can distinguish:
  - messages expected to **originate from** the environemnt (**driver role**),
  - messages expected **adressed to** the environemnt (**monitor role**).
- Adjust the TBA-construction algorithm to construct a **test driver & monitor** and let it (possibly with some **glue logic** in the middle) interact with the software.
- **Test passed** (i.e., test unsuccessful) if and only if TBA state q_6 is reached.

  Note: We may need to **refine** the LSC by adding an activation condition; or communication which drives the system under test into the desired start state.
- For example the **Rhapsody** tool directly supports this approach.

Vocabulary



- **Software-in-the-loop**:
  The final implementation is examined
  using a separate computer to simulate other system components.



- **Hardware-in-the-loop**:
  The final implementation is running on (prototype) hardware
  which is connected by its standard input/output interface (e.g. CAN-bus)
  to a separate computer which simulates other system components.

Content

- **Testing**: Rest
  - **Model-Based Testing**
  - **When To Stop Testing?**
  - **Testing in the Development Process**
- **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**

When To Stop Testing?

When To Stop Testing?

- There need to be defined **criteria** for when to stop testing;
  project planning should consider these criteria (and previous experience).

- Possible "**testing completed**" criteria:
  - all (previously) **specified test cases**
    have been executed with negative result,

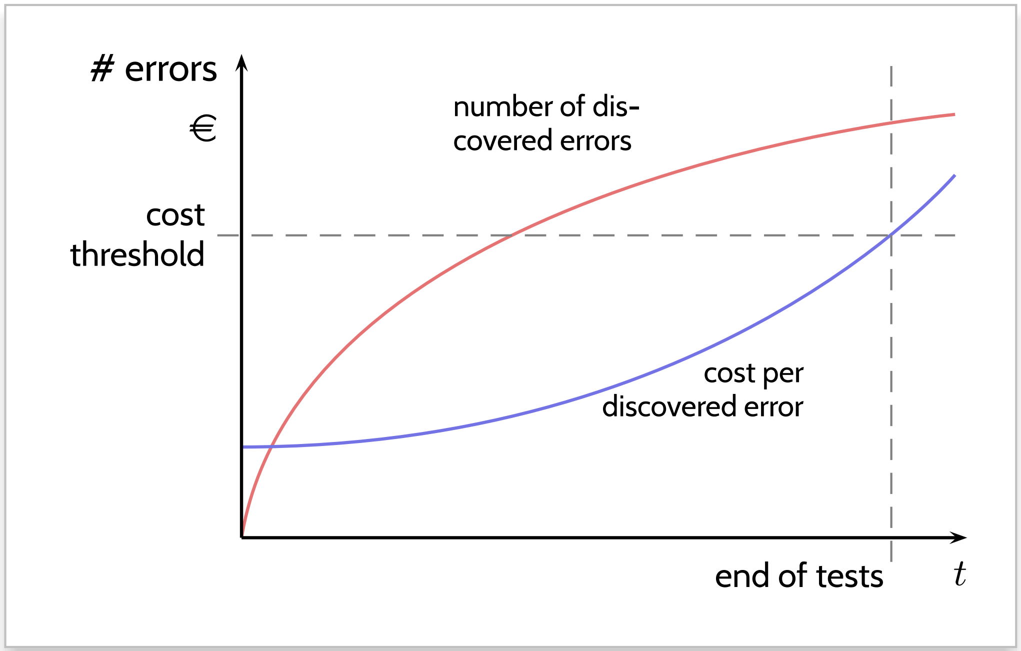
    (**Special case**: All test cases resulting from a certain strategy,
    like maximal statement coverage have been executed.)
  - **testing effort time** sums up to x (hours, days, weeks),
  - **testing effort** sums up to y (any other useful unit),
  - n **errors** have been discovered,
  - **no error** has been discovered **during**
    the last z hours (days, weeks) of testing,

  Values for x, y, n, z are fixed based on experience, estimation, budget, etc.

- **Of course**: not all criteria are equally reasonable or compatible with each testing approach.

Another Criterion

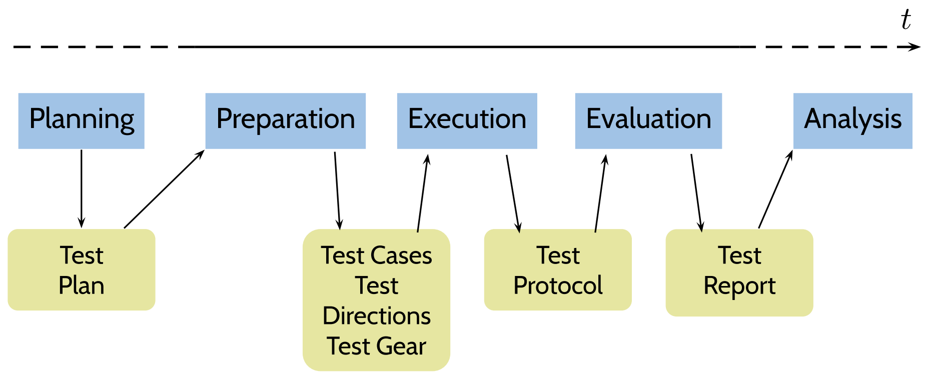
- Another possible "**testing completed**" criterion:
  - The **average cost per error discovery** exceeds a defined threshold c.



Value for c is again fixed based on experience, estimation, budget, etc..

Testing in The Software Development Process

Test Conduction: Activities & Artefacts



(Ludewig and Lichter, 2013)

- **Test Gear**: (may need to be developed in the project!)

> **test driver–** A software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results.
> Synonym: test harness. **IEEE 610.12 (1990)**

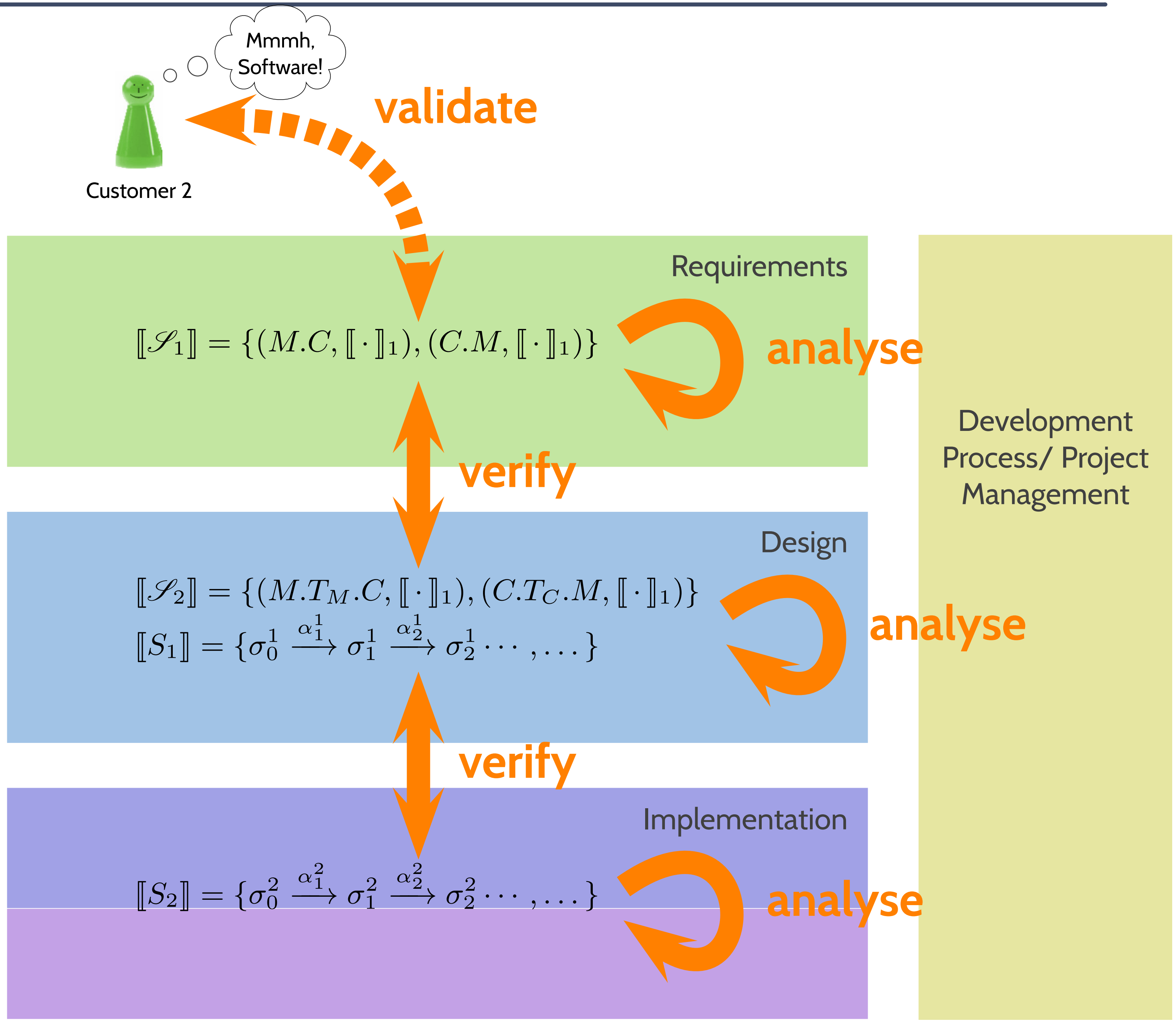
> **stub–**
> (1) A skeletal or special-purpose implementation of a software module, used to develop or test a module that calls or is otherwise dependent on it.
> (2) A computer program statement substituting for the body of a software module that is or will be defined elsewhere. **IEEE 610.12 (1990)**

- **Roles**: **tester** and **developer** should be different persons!

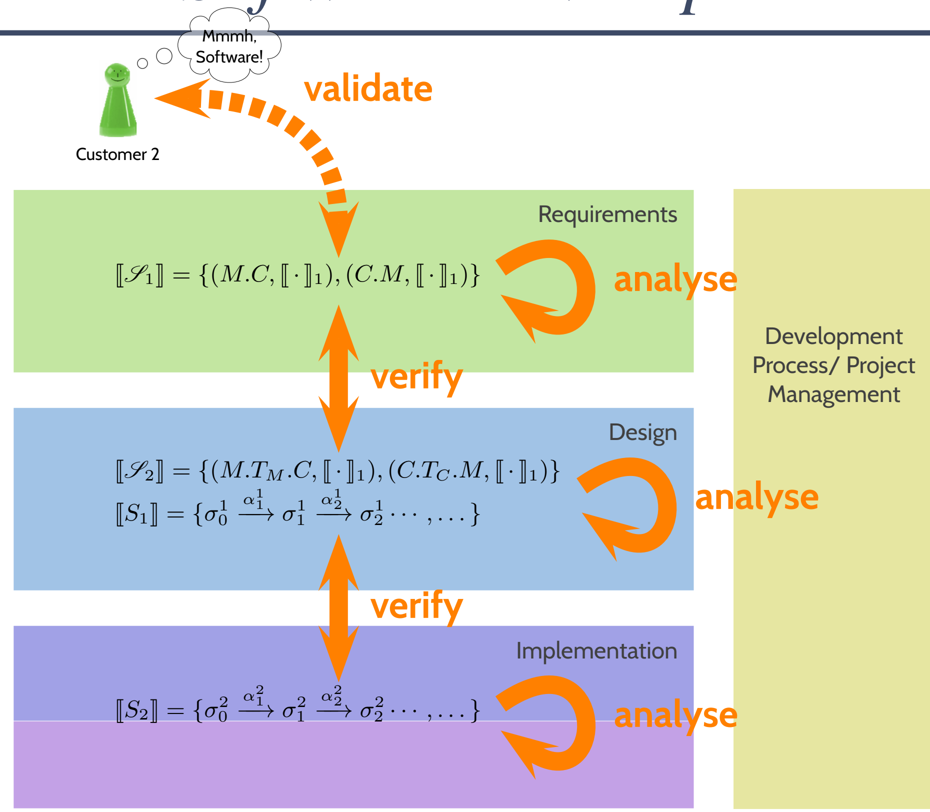
Content

- **Testing**: Rest
  - **Model-Based Testing**
  - **When To Stop Testing?**
  - **Testing in the Development Process**
- **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**

Formal Methods in the Software Development Process



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)

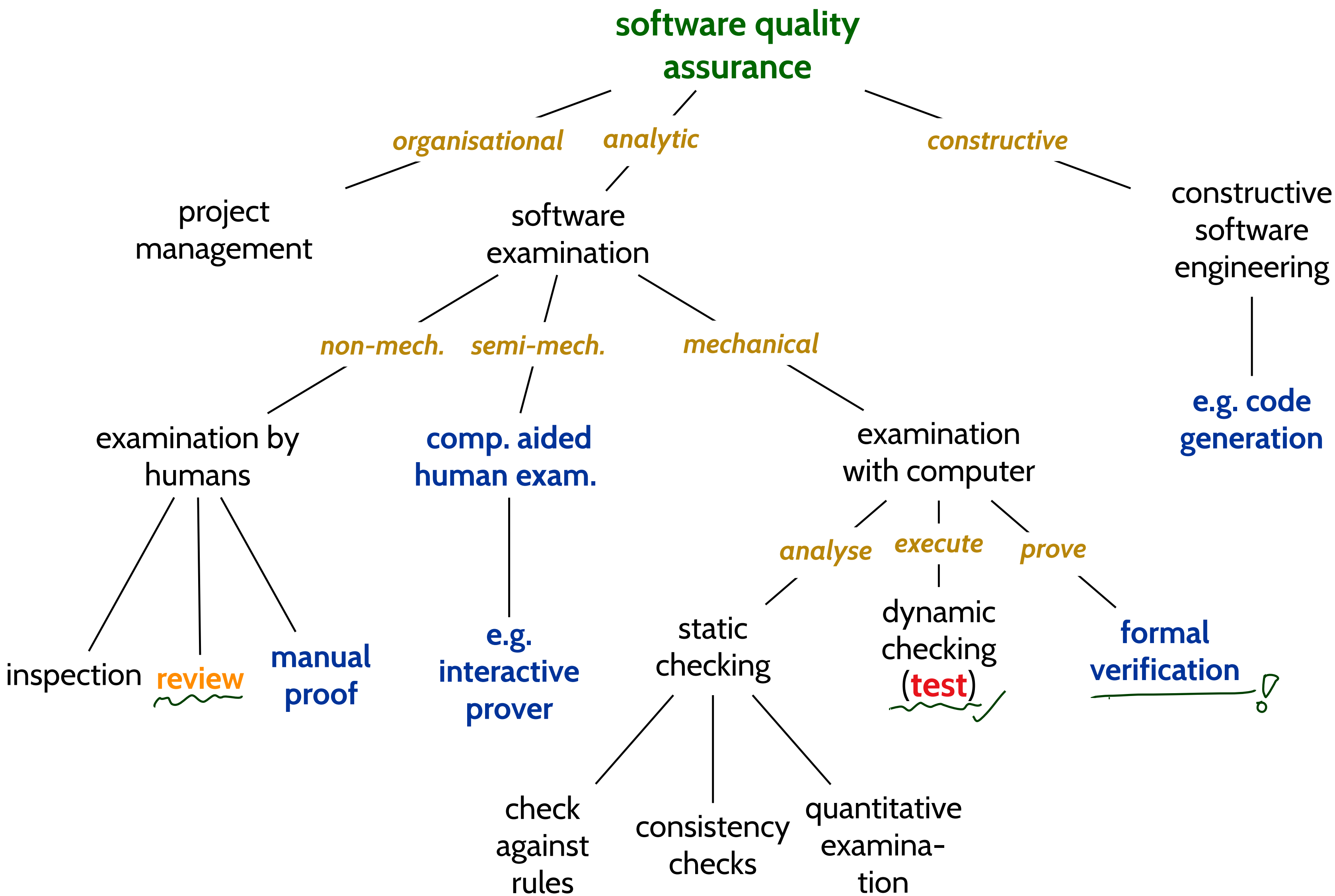
verification–

(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.

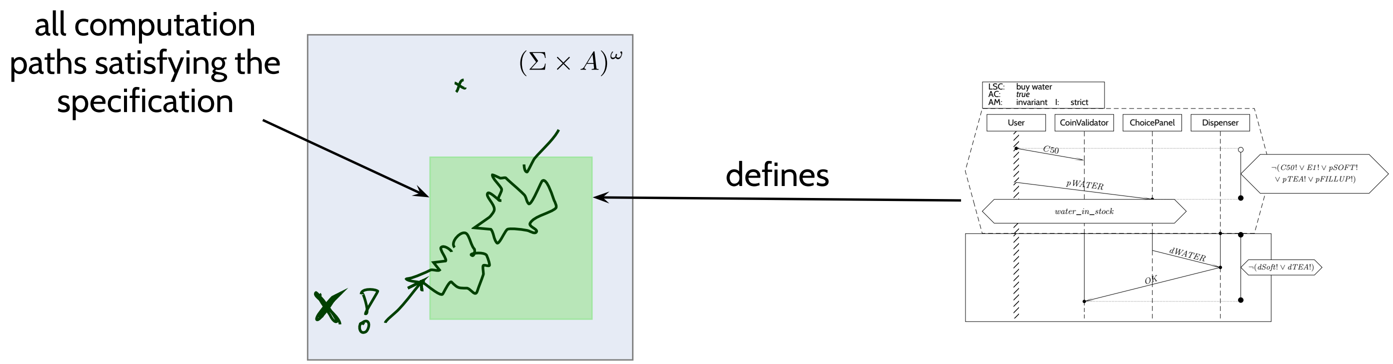
IEEE 610.12 (1990)

Concepts of Software Quality Assurance

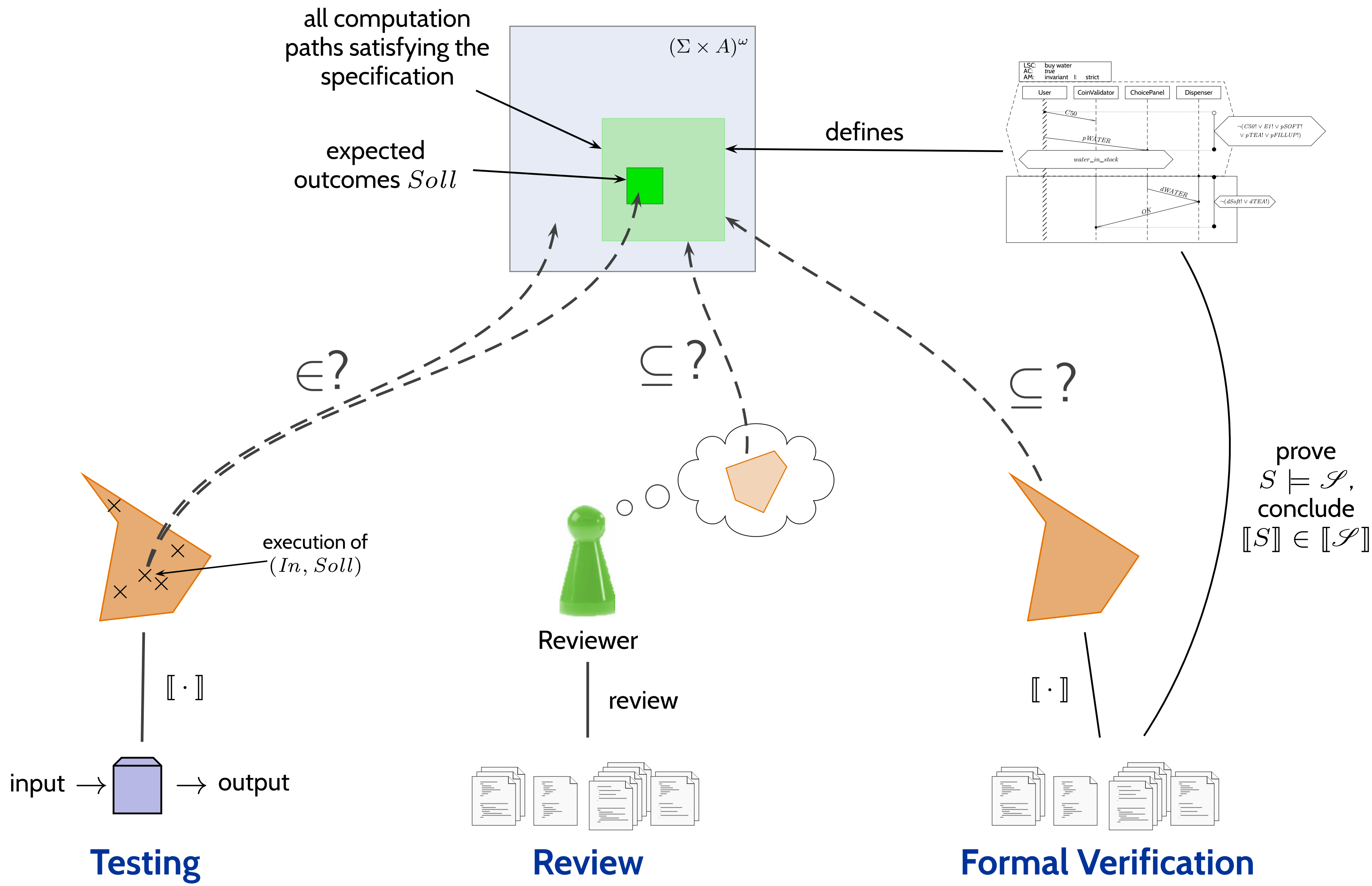


(Ludewig and Lichter, 2013)

Testing, Review, Verification Illustrated



Testing, Review, Verification Illustrated



… so, off to " 'technological paradise' where […] everything happens according to the blueprints".

(Kopetz, 2011; Lovins and Lovins, 2001)

Content

- **Testing**: Rest
  - **Model-Based Testing**
  - **When To Stop Testing?**
  - **Testing in the Development Process**
- **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**

Sequential, Deterministic While-Programs

Deterministic Programs

Syntax:

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

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

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

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

(vii) $\langle \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, \sigma \rangle \to \langle E, \sigma \rangle$, 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) $\langle skip,\, \sigma\rangle \to \langle E,\, \sigma\rangle$ $\qquad E; S \equiv S; E \equiv S$

(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**

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

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

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

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) $\langle \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi},\ \sigma\rangle \to \langle S_1,\ \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi},\ \sigma\rangle \to \langle S_2,\ \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od},\ \sigma\rangle \to \langle S; \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od},\ \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od},\ \sigma\rangle \to \langle E,\ \sigma\rangle$, 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** $S_3 \equiv y := x;\ y := (x-1)\cdot x + y;\ \mathbf{while}\ 1\ \mathbf{do}\ skip\ \mathbf{od}.$

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

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

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

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

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

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

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

Correctness of While-Programs

Correctness of Deterministic Programs

pre-condition

post-condition

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

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

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

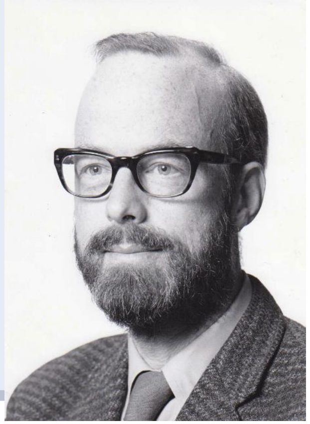
(ii) A **correctness formula**

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

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

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

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



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

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

Program S_1:

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

$\models^? \{p\}\ S_1\ \{q\}$ ✓

$\models^?_{tot} \{p\}\ S_1\ \{q\}$ ✓

Program S_2:

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

$\models^? \{p\}\ S_2\ \{q\}$ ✓

$\models^?_{tot} \{p\}\ S_2\ \{q\}$ ✗

$\models \{p\}$ while 1 do skip od $\{p\}$

Program S_3:

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

$\models^? \{p\}\ S_3\ \{q\}$ ✗

$\models^?_{tot} \{p\}\ S_3\ \{q\}$ ✗

$\sigma_d = \{x \mapsto d,\ y \mapsto 0,\ z \mapsto 1 \mid 0 \leq d \leq 27\}$

$\sigma_3 \rightsquigarrow y = 10 \neq 3^2$

Program S_4:

p

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

q

	math.	C 32-bit
$\models^? \{p\}\ S_4\ \{q\}$	✓	✗ (overflow)
$\models^?_{tot} \{p\}\ S_4\ \{q\}$	✓	✗

Example: Correctness

- By the 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) $\langle skip, \sigma\rangle \to \langle E, \sigma\rangle$ $\qquad E; S \equiv S; E \equiv S$
(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) $\langle \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, \sigma\rangle \to \langle S_1, \sigma\rangle$, if $\sigma \models B$,
(v) $\langle \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, \sigma\rangle \to \langle S_2, \sigma\rangle$, if $\sigma \not\models B$,
(vi) $\langle \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, \sigma\rangle \to \langle S; \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, \sigma\rangle$, if $\sigma \models B$,
(vii) $\langle \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}, \sigma\rangle \to \langle E, \sigma\rangle$, if $\sigma \not\models B$,

Consider **program**

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

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

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

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

- **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**, $\{x = 2 \wedge \forall\, i \geq 2 \bullet a[i] = 1\}\ S\ \{false\}$ also holds.

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**

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

Rule 5: **While-Loop**

$$\frac{\{p \land B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \land \neg B\}}$$

Rule 6: **Consequence**

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

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

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

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

Example Proof

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

(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 $\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}\ DIV\ \{a \cdot y + b = x \wedge b < y\}$, i.e., derivability in PD:

Example Proof

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

(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:

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:

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

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

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

Example Proof Cont'd

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

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!**):

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

Proof of (1)

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

- **(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). $\square$

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.

- **indexed variable**, u plain or $u \equiv b[t_1, \ldots, t_m]$ and $a \neq b$:

$$(a[s_1, \ldots, s_n])[u := t] \equiv a[s_1[u := t], \ldots, s_n[u := t]])$$

- **indexed variable**, $u \equiv a[t_1, \ldots, t_m]$:

$$(a[s_1, \ldots, s_n])[u := t] \ \equiv (\textstyle\bigwedge_{i=1}^{n} s_i[u := t] = t_i\ ?\ t\ : a[s_1[u := t], \ldots, s_n[u := t]])$$

Proof of (2)

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

- (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),

- using $P \wedge b \geq y \rightarrow (a+1) \cdot y + (b-y) = x \wedge (b-y) \geq 0$ and $P \rightarrow P$ we obtain,

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

  by (R6). $\square$

Proof of (3)

(3) claims

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

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

Proof: easy.

Back to the Example Proof

We have shown:

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

and

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

thus

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

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

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

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

Once Again

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

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

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

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

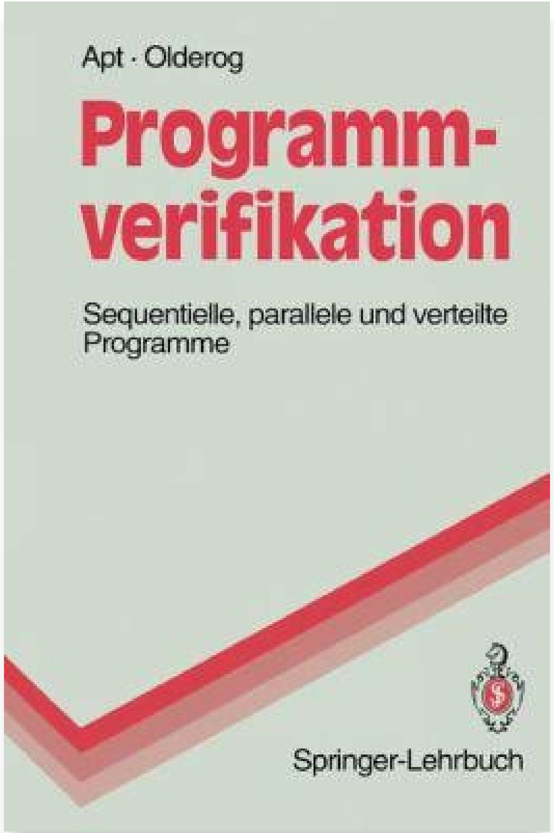
(R3) $\dfrac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$

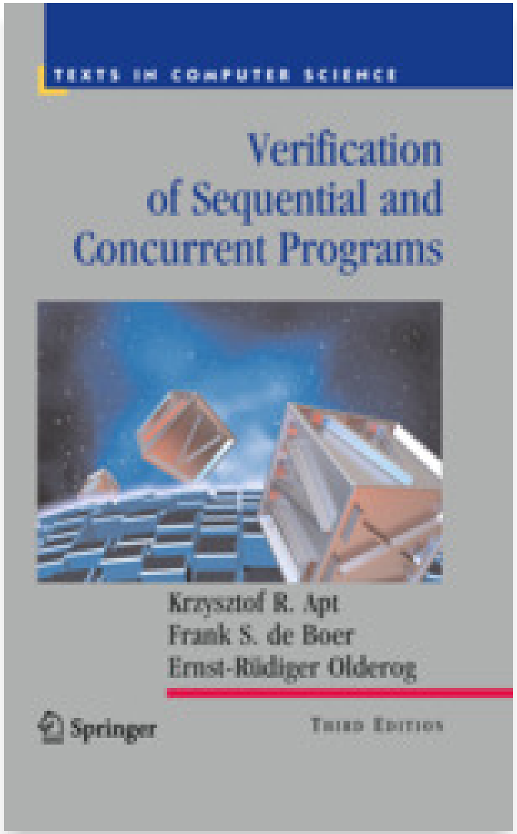
(R4) $\dfrac{\{p \wedge B\}\ S_1\ \{q\},\ \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$

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

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

Literature Recommendation





Tell Them What You've Told Them…

Testing:

- Define criteria for **"testing done"** (like coverage, or cost per error).
- **Process**: tester and developer should be different persons.

Formal Verification:

- There are **more approaches** to software quality assurance than just **testing**.
- For example, **program verification**.
- **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.

Kopetz, H. (2011). What I learned from Brian. In Jones, C. B. et al., editors, *Dependable and Historic Computing*, volume 6875 of *LNCS*. Springer.

Lettrari, M. and Klose, J. (2001). Scenario-based monitoring and testing of real-time UML models. In Gogolla, M. and Kobryn, C., editors, *UML*, number 2185 in Lecture Notes in Computer Science, pages 317–328. Springer-Verlag.

Lovins, A. B. and Lovins, L. H. (2001). *Brittle Power - Energy Strategy for National Security*. Rocky Mountain Institute.

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