Softwaretechnik / Software-Engineering

Lecture 17: Wrapup & Questions

2019-07-22

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

$Proof\text{-}System\ PD\ (for\ sequential,\ deterministic\ programs)$

$p \to p_1, \{p_1\} S \{q_1\}, q_1 \to q$ $\{p\} S \{q\}$	$\{p\} S_1; S_2 \{q\}$
Rule 6: Consequence	Rule 3: Sequential Composition
$\{p \land B\} S \{p\}$ $\{p\}$ while B do S od $\{p \land \neg B\}$	$\left\{ p[u:=t]\right\} u:=t\left\{ p\right\}$
Rule 5: While-Loop	Axiom 2: Assignment
$\{p \land B\} S_1 \{q\}, \{p \land \neg B\} S_2 \{q\}, \{p\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \{q\}$	$\{p\}$ skip $\{p\}$
Rule 4: Conditional Statement	Axiom 1: Skip-Statement

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 $\models \{p\} S\{q\}$.

Topic Area Code Quality Assurance: Content

```
VL 184 • In roduction and Vocabulary

• Past case, test sale, test execution.

• Pastine and negative cotomes.

• Limits of Software Testing

• Class-Box Testing

• Sauements, bandh - term-coverage.

• Other Approaches

• Program Verification

• purish and total conventions.

• Program Verification

• purish we welfication

• purish we welfaction

• Program Verification

• Program Verifi
```

 $\begin{array}{lll} (A1 \mid p) * dep \; (p) & (B2 \mid \underline{G}) \; \underline{S} \; (f), \; (f) \; \underline{S} \; (g) & (B3 \mid \underline{D}) \; & (p \wedge B) \; \underline{S} \; (p) \\ & (p) \; \underline{S} \; (g) \; & (g, \overline{G}) \; & (B3 \mid \underline{D}) \; & (p) \; & (p) \; & (p) \; & (p) \; \\ & (p) \; \underline{S} \; (g), \; \underline{S} \; (g), \; \underline{S} \; (g) \; & (p) \; & \underline{S} \; (g) \; & (p) \; & \underline{S} \; (g) \; & \underline{S} \; (g) \; & \underline{S} \; \underline{S} \; \underline{S} \; (g) \; & \underline{S} \; \underline{S} \; \underline{S} \; (g) \; & \underline{S} \; \underline{S}$

Proof-System PD Cont'd

 $e Proof = s_0 = s_0 = s_0$

Example Proof $=\sup_{a\in S_0^D} \frac{a=B^D}{b:a:a+1} \text{ od}$ $DIV \equiv a:=0, b:=\overline{x}, \text{ while } b\geq y \text{ do } b:=b-y; \ a:=a+1 \text{ od}$ (The first termular) represented program that has been termular welfed (Proon, 1990). We can prove $[\pi(x\geq 0 \land y\geq 0)DIV(a:y+b=x \land b < y)]$ by dowling $b \vdash pD(x\geq 0 \land y\geq 0)DIV(a:y+b=x \land b < y)$. i.e., derivability in PD.

We an prove $=\{x\geq 0 \land y\geq 0\}$ $DIV\{(a\cdot y+b=x\land b< y\}$ by showing +np $(x\geq 0 \land y\geq 0)$ $DIV\{(a\cdot y+b=x\land b< y\}$, i.e., derivability in PD: $\sup_{x\in y^{(D)}} \frac{1}{x} \int_{x}^{x} \frac{1}$

J.

* (I) claims

* (I) dam's

* (I) A = 0, A =

Proof of (1) $\text{And } (p) \text{ disp } (p) \text{ and } \frac{(p \wedge P) \otimes (\alpha) \cdot (p \wedge -p) \otimes (\alpha)}{(p) \text{ if the task Senses } (1, \alpha)} \otimes \frac{(\alpha)}{(p)}$ $\text{And } (p) (p) \text{ and } (p) \text{ and$

Example Proof Cont'd

Proof of (1)

 $\begin{array}{ll} (AA) (p) \ skip (p) & \Re d \ [p,L] \ S \cdot \{p\} \ (P \ Lors) S \cdot \{p\} \ (P \ Lors) S \cdot \log S \cdot R \cdot \{p\} \ (P \ Lors) S \cdot \log S \cdot R \cdot \{p\} \ (P \ Lors) S \cdot \log S \cdot R \cdot \{p\} \ (P \ Lors) S \cdot \{p\} \ (P \ Lors$

Substitution

The rule 'Assignment' uses (syntacticall substitution: $\{p|u:=t\}\}$ $u:=t\{p\}$ (Informula μ , replace all free) occurences of (program or logicall whable u by term n.) Defined as usual, only indexed and bound variables need to be treated specially;

864

by (R6).

 $-p_D(\{x \ge 0 \land y \ge 0\}) a := 0; b := \{P\}$

7,64

Substitution

```
\begin{array}{ll} & \text{constant } op, \text{ terms } s_i; \\ & op(s_1,\ldots,s_n)[u=i] \\ & \equiv op(s_1[u:=i],\ldots,s_n[u:=i]), \\ & \text{conditional expression} \\ & (B^*_1 g_1:g_i)[u:=i] \\ & \equiv (B^*_1[u:=i],u:=i] : s_2[u:=i]) \end{array}
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        • constant c:

c[u := t] \equiv c
\bullet \text{ indexed vanishbe, } u \text{ plan or } u \equiv b(t_1,\ldots,t_m) \text{ and } o \neq k (o(s_1,\ldots,s_m)[u:=t] \equiv o(s_1[u:=t],\ldots,s_m[u:=t]) \bullet \text{ indexed vanishbe, } u \equiv d(s_1,\ldots,s_m) (o(s_1,\ldots,s_m)[u:=t] \equiv (\bigwedge_{i=1}^m s_i[u:=t] = t_i ? t: o(s_1[u:=t],\ldots,s_m[u:=t])
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    The rule 'Assignment' uses (syntactical) substitution: \{p|u:=t\}u:=t\{p\} (in formula p, replace all fined) eccurence of (program or logical) variable u by term t_i.) Defined as usual, only indexed and bound variables need to be reated specially.
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          \bullet \ \ \text{plain variable} \ x{:}x[u:=t] \equiv \begin{cases} t & \text{, if } x=u \\ x & \text{, otherwise} \end{cases}
                                                                                                                                                                                                                                         • bodson expression p\equiv s:
p(u:=t)\equiv \{u:=t\}
• regation:
(\neg q)[u:=t]\equiv \neg (q(u:=t))
• conjunction etc.
(r_i \land p)[u:=t]
• conjunction etc.
(r_i \land p)[u:=t]
• q(u:=t) \land \neg \{u:=t\}
\equiv q(u:=t) \land \neg \{u:=t\}
• constituen:
[\forall v: x_i \land p)[u:=t] \Rightarrow \forall v: x_i \nmid v:=t\}
y \text{ tesh} (\text{not in } q_i, t_i, t_i), \text{ anne type as } x_i
```

 $\begin{aligned} & \mathbf{A2} \left\{ p | \mathbf{u} = t \right\} \mathbf{u} = t \left\{ p \right\} & \mathbf{R5} \left\{ p \right\} \mathbf{white} B \ \text{do } S \ \text{od} \ \left\{ p \land B \right\} S \left\{ p \right\} \\ & \mathbf{R23} \left\{ p \right\} S_1 \left\{ r \right\}, \left\{ r \right\} S_2 \left\{ q \right\} & \mathbf{R6} \ p \xrightarrow{p \rightarrow p_1, \ p_2 \mid S} \left\{ q \right\}, \ q_1 \rightarrow q} \\ & \left\{ p \right\} S_1 \left\{ S_2 \left\{ q \right\} \right\} & \mathbf{R6} \ p \xrightarrow{p \rightarrow p_1, \ p_2 \mid S} \left\{ q \right\}, \ q_1 \rightarrow q} \end{aligned}$ (A) $\{p\}$ skip $\{p\}$ (R4) $\frac{\{p \land B\} S_1 \{q\}, \{p \land \neg B\} S_2 \{q\}\}}{\{p\}$ if B then S_1 else S_2 fl $\{q\}$

Proof of (2)

 $\begin{array}{ll} (A(1|j) \ s(p) \ f(p)) & (B(1|j) \ f(p)) \ f(p) \$

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

Proof of (2)

• (2) claims: $|-r_D| \underbrace{\{ P \land b \geq y \}}_{\{-D \land b \geq y \}} b := b - y; \ a := a + 1 \ \{ P \}$ where $P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b = x \land b \geq 0, \\ \text{where } P \equiv \underbrace{\{ -v + b = x \land b = x$ $\bullet \vdash_{PD} \{(a+1) \cdot y + b = x \land b \geq 0\} \ a := a+1 \left\{\underbrace{a \cdot y + b = x \land b \geq 0}_{\equiv P} \right\} \quad \text{by (A2)}.$

10,64

• using $P \wedge b \geq y \to (a+1) \cdot y + (b-y) = x \wedge (b-y) \geq 0$ and $P \to P$ we obtain, $\bullet \vdash_{PD} \{(a+1) \cdot y + (b-y) = x \land (b-y) \geq 0\} \ b := b-y; \ a := a+1 \ \{P\} \quad \text{ by (R3)}.$

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

 $(\partial \Omega(p|u=t))|u=t(p) - \partial \Omega \frac{(p \wedge B)}{(p)} S(\frac{1}{2}, |p \wedge B) S_1(\frac{1}{2}, |p \wedge B) S_2(\frac{1}{2}, |p \wedge B)} \frac{(p)}{(p)} \frac{\partial \Omega(p|u=t)}{\partial \Omega(p|u=t)} \frac{\partial \Omega(p|u=t)}{\partial \Omega(p)} \frac{\partial \Omega(p)}{\partial \Omega(p)$

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

(A0 (p) skip (p)

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

Example Proof Cont'd

Proof of (2)

$$\begin{split} & \text{All}(p) \, \sin(p) & & \text{Red} \, \frac{|p_i(A)| \, S_i(q), |p_i(A-A)| \, S_i(q)}{\langle p_i(B)| \, \text{Red}, |p_i(A)| \, \text{Special Size} \, S_i(q)} \\ & \text{All}(p) \, (i = 1) \, \text{If} \, = i \, (p) \, \, & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}, |p_i(A)| \, \text{Red}} \, \text{Sec} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}, |p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, S_i(p_i(A)| \, \text{Red}, |p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}, |p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}, |p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}, |p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \\ & \text{Red} \, \frac{|p_i(A)| \, \text{Red}}{\langle p_i(A)| \, \text{Red}} \, \frac{|p$$

```
(3) \models P \land \neg (b \ge y) \rightarrow a \cdot y + b = x \land b < y.
                                                                                                                                                       (2) ⊢<sub>PD</sub> {P ∧ b ≥ y} b := b − y; a := a + 1 {P}.
                                                                                                                                                                                                 (1) \vdash_{PD} \{x \ge 0 \land y \ge 0\} \ a := 0; \ b := x \{P\}, \checkmark
                                                                                                                                                                                                                                                                In the following, we show
As loop invariant, we choose (creative actl):
```

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

 $\bullet \vdash_{PD} \{(a+1) \cdot y + \underline{(b-y)} = x \land \underline{(b-y)} \geq 0\}\underbrace{b}_{\mathbf{x}} := \underbrace{b-y}_{\mathbf{x}} \{(a+1) \cdot y + \underline{b} = x \land \underline{b} \geq 0\}$ by (A2).

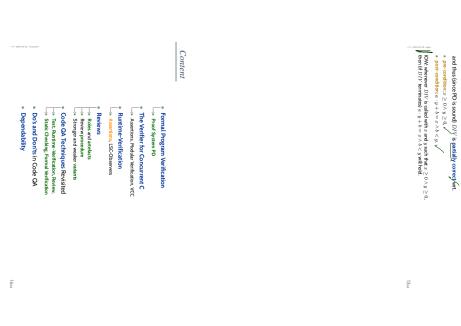
• (2) claims: $\vdash_{PD} \{P \wedge b \geq y\} \ b := b - y; \ a := a + 1 \ \{P\}$ where $P \equiv a \cdot y + b = x \wedge b \geq 0$.

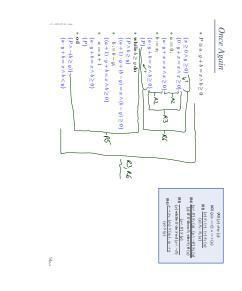
Example Proof Cont'd

```
(2) \vdash_{PD} \{P \land b \ge y\} \ b := b - y; \ a := a + 1\{P\}, \checkmark
                                                                      (3) \models P \land \neg(b \ge y) \rightarrow a \cdot y + b = x \land b < y.
                                                                                                                                                                    (1) \vdash_{PD} \{x \ge 0 \land y \ge 0\} \ a := 0; \ b := x \{P\}, \diagup
                                                                                                                                                                                         In the following, we show
As loop invariant, we choose (creative act!):
```



1264





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

thus

We have shown: (1) $\vdash_{PD} \{z \ge 0 \land y \ge 0\} a := 0; b := x \{P\},$ (2) $\vdash_{PD} \{P \land b \ge y\} b := b - y; a := a + 1 \{P\},$ (3) $\vdash_{P} P \land \neg (b \ge y) \rightarrow a \cdot y + b = x \land b < y.$

Back to the Example Proof

The Verifier for Concurrent C

0 - 200 et 31 - -

1564

17/64

VCC

- The Verifier for Concurrent C (VCC) basically implements Hoare-style reasoning.
- # #include < ycc.h>
- _(requires p) pre-condition, p is (basically) a C expression
- _(ensures q) post-condition, q is (basically) a C expression
- ullet _(invariant expr) loop invariant expr is (basically) a C expression
- \circ _(assert p) intermediate invariant, p is (basically) a C expression
- $_(writes~\&v) VCC~considers~concurrent~C~programs;~we~need~to~declare~for~each~procedure~which~global variables~it is allowed~to~write~to~(also~checked~by~VCC)$
- \[\tau \text{tread_local}(&v) = no other thread writes to variable v (in pre-conditions)
 \[\text{\capacital} \text{\capacital}(v) = the value of v when procedure was called (useful for post-conditions)
 \[\text{\capacital} \text{\capacital} = return value of procedure (useful for post-conditions)
 \]

VCC Syntax Example



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

VCC Features

Interpretation of Results

VCC result: "verification succeeded"

* We can only conclude that the tool — when is platform assumptions (32-bh), etc. — under its interpretation of the Catandard under its platform assumptions (32-bh), etc. — claims that there is a pool of ir $= [\mu]$ $D^{-1}(q)$. An analysis of the conclusion of the pool and check it musually received by the conclusion of the pool and check it musually received by the conclusion in practice) or with other tools like interactive theorem provers. Note: $\{f(abg)\}/(10-b)$ when the loss $\{h(abg)\}/(10-b)$ when the loss $\{h(abg)\}/(10-b)$ when the process in the program go undetected. That is, a mixture in writing down the pre-condition can make errors in the program go undetected.

- For the exercises, we use VCC only for sequential, single-thread programs.
- VCC checks a number of implicit <u>assentions</u>:
 no arithmetic overflow in expressions (according to G-standard).
 amp-out-of-bounds access.
 NULL-pointer der eference.

- 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:

- concurrent;

 different threads may write to shared global variables; VCC can check whether concurrent access to shared variables is properly managed;

 distanced me invaliants:

 we may declare invariants that have to hold for e.g., records (e.g. the length field is always equal to the length of the string field str); those invariants may temporarily be violated when updating the data structure.

Other case: "timeout" etc. — completely inconclusive outcome.

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 forly ly great have any provides satisfying, and causing a violation of q.
 — try to construct a (toue) counter-example from the hints.
 or make loop-invariants) (or pre-condition p) stronger, and try again.

VCC result: "verification failed"

VCC Web-Interface



 ${\bf Example \, program \, } DIV: \quad {\tt http://rise4fun.com/Vcc/4Kqe}$

2064

Modular Reasoning

```
Modular Reasoning
```

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

(87) \frac{\{p\}}{\{p\}}\frac{F(q)}{f(q)}

Any lower, lefting
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.
```

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

Example: if we have

we may be able to prove our pocket calculator correct.





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

Assertions

(over variables $V = \{v_1, \dots, v_n\}$; where result, $v_i^{old} \notin V$) into $Tf()\{...; return expr; \}$

```
Tf() \{
v_1^{old} := v_1; \dots; v_n^{old} := v_n;
result :=expr;
return result;
```

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

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

Content

Assertions

Extend the syntax of deterministic programs by

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

and the semantics by rule

(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).

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

```
The Verifier for Concurrent C

- Assertions, Moddar Verification VCC
- Runtime-Verification
- Assertions, LSC-Observes

→(* Roles and artefacts

→(* Review procedure

→(* Stronger and weaker variants

    Formal Program Verification
    Proof System PD

                                                                 Reviews
```

- Code QA Techniques Revisited
 Test Runtime-Verification, Review.
 Static Checking, Formal Verification

 Dependability Do's and Don'ts in Code QA

• So we cannot derive $\{tne\} x := 0$; $assert(x = 27) \{tne\} in PD$.

27/64

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

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

If p does not hold, we " $\operatorname{\mathsf{get}}$ stuck" (and cannot complete the derivation).

Extend PD by axiom:

Run-Time Verification

A Very Useful Special Case: Assertions

- Maybe the simplest instance of runtime verification: Assertions.
 Available in standard libraries of many programming languages (C, C++, Java,...).

For example, the C standard library manual reads:

- DESCRPTION

 Light monor asset() prints an error message to standard or or and terminate the program by calling abort(3) if gagges(a), is take (i.e., companes equal to zero). The purpose of this macro is to help the <u>programmer</u> find bugs in his program. The message "assertion talled in the food, function do_barQ, the 1187 is of no help at all to a <u>user</u>.
- In C code, assert can be disabled in production code (-D IDEBUG).
 Use java -ea ... to enable assertion checking (disabled by default).
 (distrips://docs.oracia.com/javas/8/docs/tedanotes/guides/language/assert.html)

30,64

Assertions At Work

 \bullet The abstract f-example from run-time verification: (specification: $\{p\} \ f \ \{q\})$







ist progress_bar_width(ist progress, ist window_left, ist window_right) {

asser(window_left-window_right); /* pre-condition */ " regi. precial com Dand 100-11

Search spreamant progress 1000; // extensi cores afready treated another state of the sta

Run-Time Verification: Example

Run-Time Verification: Idea

 $\bullet \;\;$ Computation paths of S may look like this:

 $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \xrightarrow{\alpha_{n-1}} \sigma_n \xrightarrow{\alpha \text{if } f} \sigma_{n+1} \cdots \sigma_m \xrightarrow{f \text{ returns}} \sigma_{m+1} \cdots$

pre-condition: p, post-condition: q.

Assume, there is a function f in software S with the following specification:







i (all x, y, sum;

a (all maint) (

sable (true) (

y = read_number();

y = read_number();

sum = add(x, y);

verity_sum(x, y, sum)

display**

3464

• If $check_p$ and $check_q$ notify us of violations of p or q, then we are notified of f violating its specification when running S' (= at run-time).

 $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \xrightarrow{\alpha_{n-1}} \sigma_n \xrightarrow{\operatorname{call} f} \sigma_{n+1} \xrightarrow{\operatorname{chark}_p} \sigma'_{n+1} \cdots \sigma_m \xrightarrow{\operatorname{chark}_p} \sigma'_m \xrightarrow{f} \xrightarrow{f \text{ returns}} \sigma_{m+1} \cdots$

For S', obtain computation paths like:

(i) extending S by implementations of check_p and check_q.

(ii) call check_p right after entering f.
 (iii) call check_q right before returning from f.

Idea: create software S' by

Assume there are functions $chark_p$, and $chack_q$, which check whether p and q hold at the current program state, and which do not modify the program state (except for program counter,



istency with the Proto-OCL constraint at runtime by using assertions:

public and set_key(int new_key) {
 assertioneries null prondign_bey consumery;
 assertioneries null row large before the color)
 assertioneries null row large verificialize(large) assertioneries null row large verificialize(large);
 key = new_large;
}

3264

More Complex Run-Time Verification: LSC Observers SILICATION BURG SINE CO

Run-Time Verification: Discussion

Experience. Assertions for pre/post conditions and intermediate invariants with a very attractive gain/effort ratio (low effort, high gain). are an extremely powerful tool

Assertions can serve as formal (support of) documentation: e.g. during later maintenance or efficiency improvement. assert(expr);

cted use of functions and

• "Dear reader, at this point in the program, I expect condition expr to hold" Be good to your readers: add a comment that explains the why.).

Content

 Formal Program Verification
 Proof System PD The Verifier for Concurrent C

Runtime-Verification Assertions, LSC-Observers

→ e Roles and artefacts
→ e Review procedure
→ stronger and weaker variants

Do's and Don'ts in Code QA

Code QA Techniques Revisited
 Test Runtime-Verification, Review,
 Static Checking, Formal Verification

Dependability

Review

3864

Reviews

Recall: Three Basic Directions

Input to Review Session:
 Review item: can be every closed,
 Imman-reducible part of software
 decommentation, models, est data,
 installation monaul, etc.)
 Social suppet; it is an artefact
 which is examined, not the human
 (who created it).

 $\begin{array}{c} \text{prove} \\ S \models \mathscr{S}, \\ \text{conclude} \\ [S] \in [\mathscr{S}] \end{array}$

Reference documents: need to enable an assessment

Moderator: lead a session responsible for properly conducted procedure. If any cause (i) of the excited unable in leptocembulator to find or the district unable in leptocembulator to find or the district unable or persons. Indeed the special procedure of the conducted Reviewards process with table to light he antificial under review. Trapke different reviewes for of different superts (or organization) colorage, etc. L. Intent or opticitized in ducted in given consistencies or incompleteness.

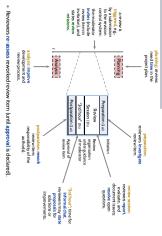
Transcript Writer: keeps minutes of review session, can be assumed by author.

The review team consists of everybody but the author(s).

 \rightarrow input \rightarrow \longrightarrow output \rightarrow Testing

Formal Verification

Review Procedure Over Time



Review Rules (Ludewig and Lichter, 2013)

(iv) The review Item is under review, not the author(s).

Reviewers choose their words accordingly.

Authors neither defend themselves nor the (vi) Style is sues (outside fixed conventions) are not discussed. (v) Roles are not mixed up, e. g., the moderator does not act as reviewer.
 (Exception: author may write transcript.) (iii) The review session is limited to 2 hours. If needed: organise more sessions. (ii) The moderator may terminate the review if conduction is not possible, e.g., due to in-puts, preparation, or people missing. The moderator organises the review, issues invitations, supervises the review session. (viii) The review team is not supposed to de-velop solutions. velop solutions are velop solutions of solutions of solutions of solutions of the author's for the author's for the author's fordings appropriately. (v) Issues are classified as:

ordical fevior wassable purposel,
major stassing seeds affected,
major stassing seeds affected,
major stassing seeds affected,
more stassing seeds affected,
pand to publish and
pand to publish affected,
accept within changes,
accept within changes,
accept within changes, (ix) Reviewers need to reach consensus on issues, consensus is noted down. (xii) The protocol is signed by all participants. 42,64

Stronger and Weaker Review Variants

Content

The Verifier for Concurrent C

Assertions, Modular Verification, VCC

The Verifier for Concurrent C

The Verifier for Concurrent C

The Verifier for Concurrent C Formal Program Verification
 Proof System PD

Runtime-Verification

Reviews -(* Assertions, LSC-Observers



Robies and artefacts
Renewey procedure
Stronger and weaker variants
Code QA Techniques Revisited
Renewey
Static Checking, Formal Verification

 Dependability Do's and Don'ts in Code QA

4464

done by developer,
 secommendation: "away from screen" (use print-out or different device and situation)

Techniques Revisited

Test Runtime- Verification Review	matic	can aun	toolchain considered	exhaus- tive	correct ×	partial results	
Test	()	V	<	×	×	~	
Runtime- Verification							
Review							
Static Checking							
Verification							

Code Quality Assurance Techniques Revisited

- Stereights

 can be fully assemble (per not easy for GLI programs).

 negative stat prover "program not complessly leader," non " (or positive scenarios);
 final postudis it sammels fully that probable and platform considered;
 final postudis it sammels fully that postular in pulliform considered;
 final propriet est cancer a usually party to destine
 for simple test cancer a usually party to destine
 provides approducible counter-samples (good stating point for repair)

45/64

(in most case) vastly incomplete, thus no poroch of correctness
 creating test cases for complex functions for complex conditional can be difficult:
 maintenance of many, complex test cases be challenging
 enecuting many tests may need substantial time (but can sometimes be ann in parallel):

Techniques Revisited

	auto- matic	prove "can run"	toolchain considered	exhaus- tive	prove	partial results	entry
Test	2	V	1	×	×	V	•
Runtime- Verification	•	Ŝ	•	(X)	×	V	Ŝ
Review							
Static Checking							
Verification							

- fully automatic (once dozeners are in place); provides counter-example; (neaty) final product is examined, thus tookhan and platform one can stop at any time and take pantial results; assert-statements have a very good effort/effect ratio.

counter-examples not necessally approducible:
 may regulately after performance:
 cone is changed, pergoran may only not hecause of the observers:
 completeness depends on usage:
 completeness depends on usage:
 may also be usely tomorphists on or correctness pends:
 constructing observers for complex properties may be difficult,
 one models to learn how to construct observers, be

Techniques Revisited

Verification	Static Checking	Review	Verification	Runtime-	Test		
		×		<	9	matic	auto-
		×		Ŝ	V	"can run"	prove
		×		<	<	considered	todchain
		(v)		<u>×</u>	×	tive	exhaus-
		(V)		×	×	correct	prove
		V		,	V	results	partial
		<u>S</u>		Ŝ	•	cost	entry

- human readers can understand the code, may goot point errors:
 reported to be highly effective:
 one can stop at any time and take partial results:
 intermediate entry costs:
 good effort/effect ratio achievable.

- no tool support:
 no results on the all alexaculors, toolchain not reviewed;
 human readers may overlook eners; usually not aiming at poods.
 does (in general) and provide counter-examples, developes may deny existence of error.

Techniques Revisited

							Verification
П	V	(V)	V	×	(x)	V	Static Checking
Т	V	3	(2)	×	×	×	Review
Г							Verification
	ς.	×	<u>×</u>	<	Ŝ	<	Runtime-
	V	×	×	<	<	3	Test
cost	partial results	prove	exhaus- tive	toolchain considered	can run"	auto- matic	

- Strengths:
 these are (commercial), fully automatic tools (first, Coverity, Polyopace, etc.);
 some tools are complete (relative to assumptions on language semantics, platform, etc.);
 can be faster than testing;
 one can stop at any time and take partial results.

- no results on actual execution, tool chain not reviewed;
 can be very resource consuming (if level also positives wanted),
 ca, cook may need to be disagried for sucks analysis;
 many like positives can be very amorjing to developers (if hast checks wanted);
 distinguish false from the positives on the challerging;
 configuring the tools (to limit false positives) can be chall erging.

Techniques Revisited

(x)	~	3	7	×	(x)	V	Static Checking
(V)	V	((V)	×	×	×	Review
							Verification
Ŝ	ς.	×	×	ς.	Ŝ	<	Runtime-
•	~	×	×	V	<	٤	Test
cost	results	correct	tive	considered	"can run"	matic	
entry	partial	prove	ednaus-	toolchain	prove	auto-	

- some tool support an allabel fler commercial toold;
 complete feather to samplions on language semantics, platform, etc.);
 thus can provide correctness proofs;
 can provide correctness for midigle ilanguage semantics and platforms at a time;
 can be mose efficient from other techniques;

- no membro actual aceution, todoshain not melamed, not many intermediate master. Tade of a port may not distorate used in conclusions; entry cost high-ingelizent triangle (unded todosovato to desirate high-ingelizent triangle), and todosovato distorate and under a powing single (underlangle) falling to find a pod descreta distorate and useful conclusions.

 fisher regulatives (backen program "propert" current) hard to detect.

Do's and Don'ts in Code Quality Assurance

Techniques Revisited

can run



Avoid using special examination versions for examination.

(Test-harness, stubs, etc. may have errors which may cause false positives and (!) negatives.)

Some Final, General Guidelines

Avoid to stop examination when the first error is detected.

Clear: Examination should be aborted if the examined program is not executable at all.

- Do not modify the artefact under examination during examinatin.
- otherwise, it is unclear what exactly has been examined ("moving target"),
 (examination results need to be uniquely traceable to one artefact version.)
- fundamental flaws are sometimes easier to detect with a complete picture of unsuccessful/successful tests,

- changes are particularly error-prone, should not happen "en passant" in examination,
 fixing flaws during examination may cause them to go uncounted in the statistics (which we need for all kinds of estimation).

- roles developer and examinor are different anyway:
 an examinor fixing flaws would violate the role assignment.

47,64

Do not switch (fine grained) between examination and debugging.

Dependability Case

Contents of the Course

What Did We Do?

Construction Contract Control Constant Section of the latest of the l and the same of th

17 Lectures on Software Engineering

52/64

Looking Back:

Proposal: Dependability Cases (Jackson, 2009)

A dependable system is one you can depend on — that is, you can place your trust in it.

and make an explicit argument that the system satisfies them." "Developers [should] express the critical properties

- Identify the critical requirements, and determine what level of confidence is needed. (Most systems do also have non-critical requirements, Continued abendability case, i.e. an argument, that the software, in concert with other components,

- es tablishes the critical properties.

 The dependability case should be
- auditable can (assiy) be evaluated by thed-party certifier.
 any assumption to have in the sugarment
 any assumption to have not plurified should be noted
 (e.g. assumptions on complier, on protocol obspect by users, etc.)
 sound e.g. should not default full conventions of landed on note shaustive testing should not make unwarranted assumptions on independence of component failures;
 etc.

50,64

Tell Them What You've Told Them...

- Runtime Verification
- dus the name surgessign decks properties at program nun-time, experiences of energy control to a valuable safe-goard against engression, usage contains particular and server as formul documentation of intermediated assumptions. Very attractive effort, effect ratio.
- Review (structured examination of artefacts by humans)
- (mild variant) advocated in the XP approach.
- lead programmer reviews all commits from team members,
 literature reports good effort/effect ratio achievable.
- All approaches to code quality assurance have their
 advantages and drawbacks.
 Which to use? It depends!
- Overall: Consider Dependability Cases
- an (auditable, complete, sound) argument, that a software has the critical properties.

) , þ ∳. **!0**

5464



The Software-Engineering Course on One Slide Topic Area: Software Quality Assurance Topic Area: Project Management

55%4

The Software-Engineering Course on One Slide

Topic Area: Project Management

measure, know what you measure (scales, pseudo-metrics)
 e estimate, measure, improve estimation – it's about experience
 describe processes in terms of artefact, activity, role, etc. – and risk

Topic Area: Requirements Engineering requirements data charies exceptable and unacceptable softwares (there may be a gay zozel) (there may be a gay zozel) formal requirements unshribous, zozet analysis methods requirements engineers see the absence of meaning

Topic Area: Architecture & Design

Model: Nobody builds shouse without a plan" (IL Lampod)

software has instituted and behavioural aspects

shows even methods and book to analyse shouse models
(brown how to interpret analysis outcomes)

Topic Area: Software Quality Assurance

• testing is almost always incomplete; testing is almost always incomplete; testing is necessary
(foron/tow) to interpret the outcomes: treaf false positive/negative)
• there are methods and roots to prove contentes as ode
(correctness is relative; correct writ specification (and assumptions))

55,64

Questions?

58,64

Advertisements

59/64

56,64

That's Today's Software Engineering — More or Less...

 Program Verification
 (the theory behind tools like VCC)
 Formal Methods for Java
 (JML and "VCC for Java") Further studies:
 Real-Time Systems (not in 2019/20)
 Real-Grand verification of real-time systems) $\rightarrow \mathtt{https://swt.informatik.uni-freiburg.de/teaching}$ Decision Procedures
 (the basis for program verification) Software Design, Modelling, and Analysis in UML (not in 2019/20) (a formal in-depth view on structural and behavioural modelling) Cyber-Physical Systems! - Discrete Models more on wains of Chand agazins (III., CII., CII.') cyber-Physical Systems - Hybrid Models (Modelsing and analysis of cyber-physical systems with hybrid automata) (Modelling and analysis of cyber-physical systems with hybrid automata)

> Want to be a scientific student assistant? Want to be a tutor, e.g. Software Engineering 2020,
>
> ontact us (around early September / early March).

→ contact us.

Advertisement

Advertisement

• Individual Projects
(BSc/MSc project, Lab Project, BSc/MSc thesis)

→ contact us (3–6 months before planned start).

 own topics improving analysis techniques formal modelling of industrial case studies

References

6364

References

Fagan, M. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211.

Hoare, C. A.R. (1969). An axiomatic basis for computer programming. Commun. ACM, 12(10):576–580. Fagan, M. (1986). Advances in software inspections. IEEE Transactions On Software Engineering, 12 (7):744–751.

Jackson, D. (2009). A direct path to dependable software. Comm. ACM, 52(4).

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

6464

Thanks For Your Participation...