Mutation Testing in UTP

UTP

Bernhard K. Aichernig

Institute for Software Technology
Graz University of Technology
Graz, Austria

UNU-IIST: United Nations University
International Institute for Software Technology
Macau S.A.R. China

PSSE 2007

Outline

- Introduction
- Unifying Theories of Programming (UTP)
- Formalization: Faults and Test Cases
- Mutation Testing for Pre-postcondition Contracts
- Mutation Testing for Programs
- Mutation Testing for Protocol Specifications

Literature

C.A.R. Hoare and He Jifeng. **Unifying Theories of Programming.** Prentice-Hall International, 1998.

Scientific Theories

- A scientific theory takes the form of a set of equations and inequations
- usually expressed in the language of mathematics: predicates
- Purpose of predicates: describe and predict all possible (direct & indirect) observations

Example (Einstein)

$$e = mc^2$$

e ... energy of the system
m ... its mass
c ... speed of light

Different levels of abstractions: e.g. interacting quarks, elementary particles, atoms, molecules or crystal structures.

Mathematical Theories in Engineering

- Engineering project begins with specification
  - describing the observable properties and behaviour of desired product.
- design documents at various stages are indirect descriptions of same behaviour.
  - restricted notation, level of abstraction to guide physical implementation
- Correctness: implementation correct if detailed description logically implies its specification

Components:

- concurrent behaviour as conjunction of individual predicate descriptions
- alternative modes of behaviour as disjunction

Observations and Alphabets

Scientific descriptions:

1. Selection of relevant properties, observable or controllable, to understand and predict system behaviour
2. Choosing a name for each property to denote a value
3. Instructions on
   - how and when that property is to be observed
   - in what unit it is to be measured, ...

Example

List of names comes usually with type declarations:
$x : \text{integer},\ y : \text{real}, \ldots,\ z : \text{Bool}$

Definition (Alphabet)

The collection of names is known as **alphabet**.

Alphabets and Predicates

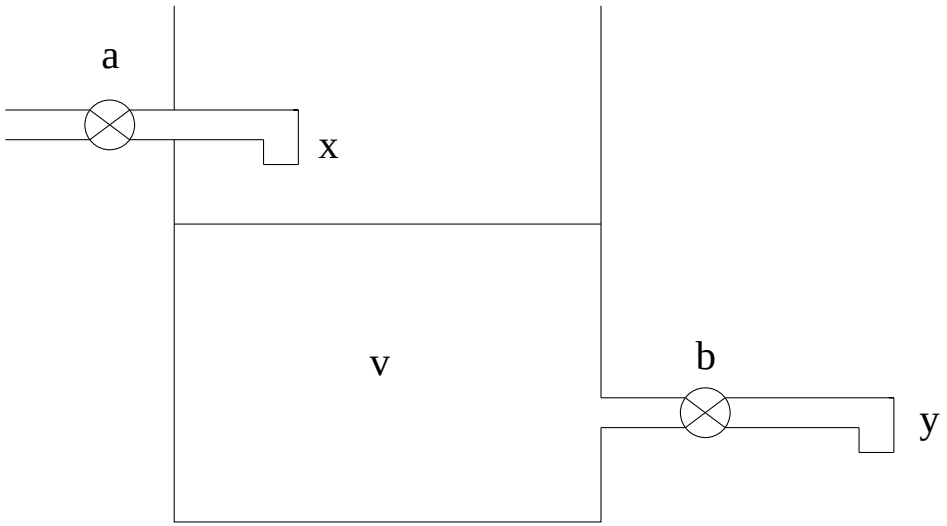
- Names in alphabet occur as free variables in predicates
- together with physical constants, mathematical symbols
- called variables, because values vary from experiment to experiment
- choice of alphabet determines and delimits a theory

- **Requirement:** Every predicate P has an associated alphabet αP
- P may contain all, some or none variables of αP.

Example (Valuation)

Observation of particular example of chosen class of system:
$x = 4 \land y = 37.3 \land \cdots \land z = \mathit{false}$

Water Tank



Alphabet:

t	time since start
v_t	volume of liquid in tank
x_t	total amount poured into
y_t	total amount drained
a_t	setting of input valve
b_t	setting of output valve

Physical variables are functions over time.

Observations in Programs

- Aim: simple theory for sequential programs
- Observations: before start and after termination
- Two names for each variable:
  x ... initial value
  x' ... final value

Example

$$x = 5 \ \wedge \ x' = 7 \ \wedge \ y = 2$$

may be an observation of a run of program $x := x + y$.

- theory determines observables (Einstein)
- too many observables: theory too complicated
- too few observables: theory inaccurate

Behaviour and Predicates

- Consider an observation

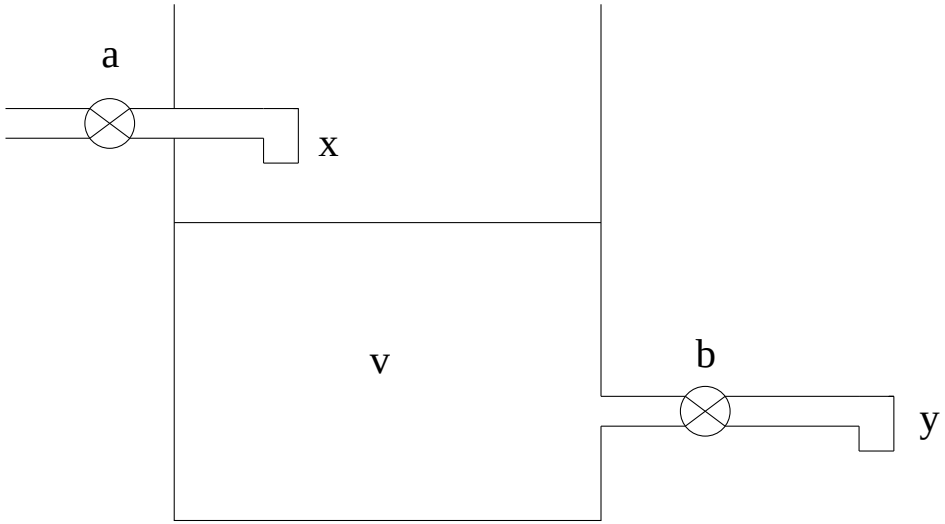
$$x = 4 \ \wedge \ y = 37.3 \ \wedge \cdots \wedge \ z = \mathit{false}$$

- Predicate $P(x, y, ..., z)$ correctly describes observation if substitution of each variable by its observed value makes the predicate true.

$$P(4, 37.3, ..., \mathit{false})$$

- Useful predicate: as strong as possible, subject to the constraints of correctness:
  - false exactly when its variables take combinations of values which in reality never occur together.
- Useless predicates: **true** (or $x = x \wedge y = y$) and **false**

Water Tank (cont.)



Laws (predicates):

$$x_t + v_0 = y_t + v_t \pm t \cdot \epsilon$$

for all $t \geq 0$

$$\dot{x} = k \cdot a$$
$$\dot{y} = k \cdot b + \delta \cdot v$$

Physical constraints:

$0 \leq a_t \leq a_{max}$, $0 \leq b_t \leq b_{max}$

Program Behaviour

- Assignment statement: $x := x + y$

$$x' = x + y$$

- Multiple assignment: $x, y := x + 3,\ y - x$

$$x' = x + 3 \ \land \ y' = y - x$$

Correctness

- Systems and specifications are (conjunctions of) predicates: easy concept of correctness

Definition (Correctness)

Let S be specification and P be a description of all possible behaviours of a program (implementation).

Assume P and S have the same alphabet standing for the same observations.

None of the possible observations of the implementation shall violate the specification:

$$\forall v, w, \ldots \bullet P \Rightarrow S$$

where $v, w, \ldots$ are all the variables in the alphabet. Short $[P \Rightarrow S]$

Correctness (cont.)

Example

$$[(x, y := x + 1, y) \Rightarrow x' > x \ \wedge \ y' = y]$$

- Note the mixing of programming notations with mathematical notations.
- Possible: identification of each program with a predicate
- Implication between two specifications: one is more general or abstract
- Given $[D \Rightarrow S]$, by transitivity of implication any correct implementation of D will also implement S.
- Hence, weaker specifications (S) are easier to implement.
- Easiest to implement: *true* can be implemented by anything.

Nondeterminism

- Let P and Q be product descriptions with the same alphabet.
- $P \vee Q$ may behave like P or Q.
- Both P and Q must be correct:

$$[P \vee Q \Rightarrow S] \quad \textbf{iff} \quad [P \Rightarrow S] \text{ and } [Q \Rightarrow S]$$

- Justified by disjunction being least upper bound of the implication ordering.

Stepwise Design

- Complex engineering project split into number of design stages.
- Transition by signing off a document D
- Before continuing towards a final implementation P, make sure that D is correct:

$$[D \Rightarrow S]$$

- Now, implementation reduces to a simpler task:

$$[I \Rightarrow D]$$

Separate development

- Even more effective when combining stepwise design and decomposition into subtasks.
- Let D and E be designs of components that will be assembled to meet specification S.
- Correctness of the designs can be checked before their implementation:

$$[D \wedge E \Rightarrow S]$$

- Then, separate development of D and E as products P and Q:

$$[P \Rightarrow D] \quad \text{and} \quad [Q \Rightarrow E]$$

- Their assembly will necessary satisfy the original specification:

$$[P \wedge Q \Rightarrow S]$$

Conditional

$$P \lhd b \rhd Q \quad =_{df} \quad (b \wedge P) \vee (\neg b \wedge Q), \quad \text{if } \alpha b \subseteq \alpha P = \alpha Q$$

$$\alpha(P \lhd b \rhd Q) \quad =_{df} \quad \alpha P$$

Laws of Conditional

L1 $P \lhd b \rhd P = P$ (cond idemp)

L2 $P \lhd b \rhd Q = Q \lhd \neg b \rhd P$ (cond symm)

L3 $(P \lhd b \rhd Q) \lhd c \rhd R = P \lhd b \wedge c \rhd (Q \lhd c \rhd R)$ (cond assoc)

L4 $P \lhd b \rhd (Q \lhd c \rhd R) = (P \lhd b \rhd Q) \lhd c \rhd (P \lhd b \rhd R)$ (cond distr)

L5 $P \lhd \mathit{true} \rhd Q = P = Q \lhd \mathit{false} \rhd P$ (cond unit)

Proofs in propositional calculus, via cases $b = \mathit{true}$ and $b = \mathit{false}$.

Laws of Conditional (cont.)

L6 $P \lhd b \rhd (Q \lhd b \rhd R) = P \lhd b \rhd R$

Proof.

$$
\begin{aligned}
& LHS && \{\textbf{L2}\} \\
= \; & (Q \lhd b \rhd R) \lhd \neg b \rhd P && \{\textbf{L3}\} \\
= \; & Q \lhd \mathit{false} \rhd (R \lhd \neg b \rhd P) && \{\textbf{L2 and L5}\} \\
= \; & RHS
\end{aligned}
$$

Laws of Conditional (cont.)

L7 $P \lhd b \rhd (P \lhd c \rhd Q) = P \lhd b \vee c \rhd Q$

Proof.

$$
\begin{array}{lll}
 & LHS & \{\textbf{L2}\} \\
= & (Q \lhd \neg c \rhd P) \lhd \neg b \rhd P & \{\textbf{L3} \text{ and } \textbf{L1}\} \\
= & Q \lhd \neg c \wedge \neg b \rhd P & \{\textbf{L2}\} \\
= & RHS &
\end{array}
$$

Sequential Composition

$$
\begin{aligned}
P(v'); Q(v) &=_{df} \exists v_0 \bullet P(v_0) \wedge Q(v_0)\,, \quad \text{if } out\alpha P = in\alpha' Q = \{v'\} \\
in\alpha(P(v'); Q(v)) &=_{df} in\alpha P \\
out\alpha(P(v'); Q(v)) &=_{df} out\alpha Q
\end{aligned}
$$

Laws of Composition

L1 $P; (Q; R) = (P; Q); R$ (; assoc)

L2 $(P \lhd b \rhd Q); R = (P; R) \lhd b \rhd (Q; R)$ (; left distr)

Proof Hint: Move the existential quantifier outward over predicates that do not contain the quantified variable.

Assignment

Let $\mathsf{A} = \{x, y, \ldots, z, x', y', \ldots, z'\}$, and given expression e with $\alpha e \subseteq A$.

$$x :=_A e =_{df} (x' = e \land y' = y \land \cdots \land z' = z)$$
$$\alpha(x :=_A e) =_{df} A$$

Usually, we will not subscript an assignment.

Laws of Assignment

L1 $(x := e) = (x, y := e, y)$

L2 $(x, y, z := e, f, g) = (y, x, z := f, e, g)$

L3 $(x := e;\ x := f(x)) = (x := f(e))$

L4 $x := e;\ (P \lhd b(x) \rhd Q) = (x := e; P) \lhd b(e) \rhd (x := e; Q)$

Skip

Has no effect, mainly useful for program reasoning. Let A be its alphabet:

$$\begin{aligned} \mathbf{skip}_A &=_{df} (v = v') \\ \alpha(\mathbf{skip}_A) &=_{df} A \end{aligned}$$

v is the list of all (undashed) variables in A.

$$\mathbf{L1} \quad P ; \mathbf{skip}_{\alpha P} = P = \mathbf{skip}_{\alpha P} ; P \qquad (; \text{ unit})$$

Non-deterministic Choice

$$P \sqcap Q =_{df} P \vee Q, \qquad \text{provided} \alpha P = \alpha Q$$
$$\alpha(P \sqcap Q) =_{df} \alpha P$$

Laws of Non-deterministic Choice

L1	$P \sqcap Q = Q \sqcap P$	($\sqcap$ symm)
L2	$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R$	($\sqcap$ assoc)
L3	$P \sqcap P = P$	($\sqcap$ idemp)

Proof from symmetry, associativity and idempotency of disjunction.

Laws of Non-deterministic Choice (cont.)

L4	$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap (P \sqcap R)$	($\sqcap$ distr)
L5	$P \lhd b \rhd (Q \sqcap R) = (P \lhd b \rhd Q) \sqcap (P \lhd b \rhd R)$	(cond-$\sqcap$ distr)
L6	$(P \sqcap Q); R = (P; R) \sqcap (Q; R)$	(;-$\sqcap$ left distr)
L7	$P; (Q \sqcap R) = (P; Q) \sqcap (P; R)$	(;-$\sqcap$ right distr)
L8	$P \sqcap (Q \lhd b \rhd R) = (P \sqcap Q) \lhd b \rhd (P \sqcap R)$	($\sqcap$-cond distr)

Proofs in propositional and predicate calculus.

Abort

Increasing non-determinism by adding choices makes things worse:

$$[P \Rightarrow (P \sqcap Q)]$$

Taking this to an extreme is called Abort:

$$\begin{aligned} \bot_A &=_{df} \mathbf{true} \\ \alpha\bot_A &=_{df} A \end{aligned}$$

Bottom (weakest) element of the implication ordering

$$[\bot_A \Leftarrow P], \quad \text{for all } P \text{ with alphabet } A$$

Remember: Abort (**true**) stands for non-termination!

Miracle

Non-determinism over an empty set is impossible:

$$\bigsqcap \{\} = \textbf{false}$$

If it did exist, it would satisfy every specification

$$[\textbf{false} \Rightarrow P], \quad \text{for all } P$$

Important as a reasoning mechanism:

Definition (Miracle)

$$\begin{aligned} \top_A &=_{df} \textbf{false} \\ \alpha\top_A &=_{df} A \end{aligned}$$

A Problem with Our Theory

Example (Counter-Example)

$\mathbf{true}; (x := 3)$

- In any normal implementation, this would fail to terminate and so be equal to **true**
- Unfortunately, our theory gives the unexpected result $x' = 3$
- Designs are specialised relations that repair this flaw.
- Requirement: the following should hold

$$\mathbf{true}; P = \mathbf{true}$$
$$P; \mathbf{true} = \mathbf{true}$$

Designs

- Goal: Solving the paradox of non-termination!
- Solution: adding two additional Boolean observations.

Definition (ok and ok')

ok records the observation that the program has been started.

ok' records the observation that the program has terminated.

Definition (Design)

Let P and Q be predicates not containing ok or ok'.

$$p \vdash Q \quad =_{df} \quad (ok \wedge p) \Rightarrow (ok' \wedge Q)$$

A design is a relation whose predicate is (or could be) expressed in this form (+ Healthiness Conditions).

Design Example

Designs represent pre-postcondition specifications, like in the specification languages VDM, B, OCL and JML.

Example (Square Root)

The following contract is a design of a square root algorithm using a program variable x for input and output. A constant e specifies the precision of the computation.

$$(x \geq 0 \land e > 0) \vdash (-e \leq x - x'^2 \leq e)$$

Refinement

Theorem (Refinement of Designs)

$$[(p_1 \vdash Q_1) \Rightarrow (p_2 \vdash Q_2)] \quad \textbf{iff} \quad [p_2 \Rightarrow p_1] \ \textit{and} \ [(p_2 \wedge Q_1) \Rightarrow Q_2]$$

Non-termination

$$\textbf{L1 true}; (P \vdash Q) = \textbf{true}$$

Proof: $\textbf{true}; (P \vdash Q)$

$$= \exists ok^0, \cdots \bullet \textbf{true} \wedge (ok^0 \wedge P \Rightarrow ok' \wedge Q) \qquad \{\text{let } ok^0 = \mathit{false}\}$$

$$= \textbf{true}$$

Program Statements as Designs

Definition (Assignment)

Given a program variable x and an expression e

$$x := e \quad =_{df} \quad (wf(e) \vdash x' = e \land y' = y \land \cdots \land z' = z)$$

with *wf* being the predicate defining the well-formedness (e can be evaluated) of expression e.

Definition (Conditional)

$$P \lhd b \rhd Q \quad =_{df} \quad (wf(b) \vdash (b \land P \lor \neg b \land Q))$$

with *wf* being the predicate defining the well-formedness of the Boolean expression b.

Bottom and Top of the Lattice of Designs

Definition (Abort)

$$\bot \quad =_{df} \quad \mathbf{true} \qquad = \quad \mathbf{false} \vdash \mathbf{true} \quad = \quad \mathbf{false} \vdash \mathbf{false} \quad = \quad \mathbf{false} \vdash Q$$

Definition (Magic)

$$\top \quad =_{df} \quad (\mathbf{true} \vdash \mathbf{false}) = \neg ok$$

Laws of Designs

Theorem

$$(p_1 \vdash Q_1) \sqcap (p_2 \vdash Q_2) \quad = \quad (p_1 \wedge p_2 \vdash Q_1 \vee Q_2)$$

$$(p_1 \vdash Q_1) \sqcup (p_2 \vdash Q_2) \quad = \quad (p_1 \vee p_2 \vdash ((p_1 \Rightarrow Q_1) \wedge (p_2 \Rightarrow Q_2)))$$

$$(p_1 \vdash Q_1) \lhd b \rhd (p_2 \vdash Q_2) \quad = \quad (p_1 \lhd b \rhd p_2 \vdash Q_1 \lhd b \rhd Q_2)$$

$$(p_1 \vdash Q_1); (p_2 \vdash Q_2) \quad = \quad (p_1 \wedge \neg(Q_1; \neg p_2) \vdash Q_1; Q_2)$$

$$\neg(Q_1; \neg p_2) = \neg \exists v_0 \bullet Q_1(v_0) \wedge \neg p_2(v_0)$$