Some aspects of Test Data Selection from Formal Specifications

Agnès ARNOULD
Pascale LE GALL
Gilles BERNOT
Plan

- Main difficulties
- Contributions of formal methods
- Probabilistic approach
- Deterministic approach
- Focus on Lustre specifications
Object: to check adequacy / inadequacy between:
  – the system under test
  – the specification reference object

Activities of testing:
  – selection of test cases
  – execution of tests
  – success / failure decision
Selection

- functional
- structural
  - domain → subdomains
- deterministic
- probabilistic
  - coverage criteria
Test execution

- good modularity
- adequate entry points
- adequate observation points
- instrumentation

→ impact on the early specifications
Success/failure decision (Oracle)

Predictions of the expected outputs?

- formal specifications can solve the problem

Other difficulties:

- the software gives not enough observations
- the specification says nothing
- the specification says nothing usable
- increase the number and the size of test cases
"I guarantee that the rate of failure will be less than $\varepsilon$"

Is a non-sense without a risk $\alpha$ to be wrong w.r.t. this affirmation.
Formal specifications can let you save money

- cost of 1 test $\approx 1/2$ engineer day
- computer aided selection and oracle $< 1$ min
- automatic manipulations

⇒ require formal specifications
Testing automation

Generator

Executable prog.

Submission

Test inputs

Test outputs

Oracle

Success / failure decision

Testing document

Prog. or spec.

Selection criteria

Correctness reference

Oracle selection criteria
What is a formal specification?

**Program interface description**

sorted : List \(\rightarrow\) Bool

- \(\text{sorted}([\ ]) = \text{true}\)
- \(\text{sorted}([x]) = \text{true}\)
- \(\text{sorted}([x, y | L]) = (x \leq y) \text{ and } \text{sorted}([y | L])\)

**Properties**
What is a formal test?

Test = formula without variable
operation(inputs) = output

sorted([1, 2, 3]) = true

Much better:

observable formula deduced from the specification

$\text{sorted}([1, 2, 3]) = (1 \leq 2) \text{ and } \text{sorted}([2, 3])$
Plan

- Main difficulties
- Contributions of formal methods
- Probabilistic approach
- Deterministic approach
- Focus on Lustre specifications
Probabilistic testing

ε = the vendor affirms to the client "at most Nε failures for N input values"

α = the risk that the vendor takes with this affirmation (over 100 test sets of N tests, almost surely less than 100α test sets may have more than Nε failures)

\[ N \geq \frac{\log(\alpha)}{\log(1-\varepsilon)} \]
Choice of the test cases

How to produce the $N$ relevant test cases:

$\mu = \text{a complete distribution on the domain of variables (has to be discussed with the client)}$

Problems:

– to formalize the discussion into $\mu$
– to generate test cases according to $\mu$
To automate the probabilistic test

A prototype of generator

- generates tests from a set description of domains of variables (cartesian product, union, recursive definition ...)
- hides probabilistic manipulations behind set descriptions: offers default distributions.
Advantages of probabilistic testing

- allows rough subdomain splitting
- quantitative estimate of the future system with an operational profile
- quantitative estimate of the exceptional behaviour with other criteria
- formal specification & domain description
  ➡ automatic test generation
Deterministic testing

cover the definitions case by case

ex:

\[
\text{sorted}(L) = \begin{cases} 
\text{true} & \text{sorted}([ ]) = \text{true} \\
\text{true} & \text{sorted}([x]) = \text{true} \\
(x \leq y) \text{ and } \text{sorted}([y | L']) = \text{true} & \text{sorted}([x, y | L]) = \\
\end{cases}
\]

\[
L = [ ] \\
L = [x] \\
L = [x, y | L'] \\
\text{where } (x \leq y) \text{ and } \text{sorted}([y | L']) = \text{true} \\
L = [x] \\
L = [x, y] \\
L = [x, y, z | L'] \\
\text{where } x \leq y = \text{true} \\
\text{where } ... \\
\]
To automate deterministic testing,

- solve constraints for each domain
- generate any one value in the domain
  - use constraint solving methods (logic programming techniques)
Advantages of deterministic testing

- automate current practice of functional testing
- allows thin subdomain splitting
  - automatic coverage of exceptional cases
- extracts the oracle from the specification
- opens the door to a standardization of functional coverage criterias
Lustre is a functional and dataflow language

A Lustre node as a cyclic behavior

node mem(On : bool ; Of : bool ; Init : bool)
returns (Out : bool) ;

tel ;

Out = if On then (true)
    else (if Of then (false)
            else ((Init) → (pre(Out))))) ;
Coverage criteria

- coverage on the last cycle
  ➔ one stream values per test case

- \( A = \text{if } B \text{ then } C \text{ else } D \)
  
  - 2 cases: \( B = (..., \text{true}) \)
    \( B = (..., \text{false}) \)

- \( A = B \rightarrow C \)
  
  - 2 cases: last cycle = first cycle
    last cycle = further cycle
Coverage criteria to cover all operators:

\[ \text{Out} = \begin{cases} \text{true} & \text{if On} \\ \text{false} & \text{if Of} \\ (\text{Init}) \rightarrow (\text{pre}(\text{Out})) & \text{else} \end{cases} \]

produces 4 test cases:

\[
\begin{align*}
\text{mem}((..., \text{true}), (..., _), (..., _)) &= (..., \text{true}) \\
\text{mem}((..., \text{false}), (..., \text{true}), (..., _)) &= (..., \text{false}) \\
\text{mem}((\text{false}), (\text{false}), (V)) &= (V) \\
\text{mem}((..., _, \text{false}), (..., _, \text{false}), (..., _, _)) &= (..., V, V)
\end{align*}
\]
LOFT, a test generator
(developed by B. MARRE)

- on one component:
  - 1386 lines of Lustre
  - 13 nodes
  - 101 inputs and 1 output

- 2 different selection criterias
  - 982 test cases generated in 20 s. per case
  - 33 test cases generated in 35 s. per case

- no limit to the test quality
Formals specifications allow to automate testing activities, including Oracle.

- functional probabilistic testing becomes reachable
- deterministic testing automate current empirical methods