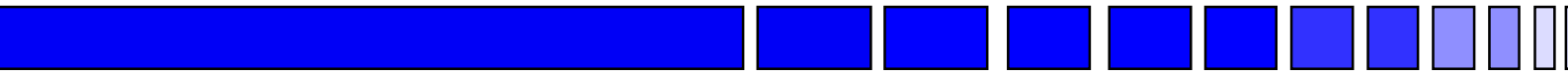
Some aspects of Test Data Selection from Formal Specifications

Agnès ARNOULD

Pascale LE GALL

Gilles BERNOT

Plan



- Main difficulties
- Contributions of formal methods

- Probabilistic approach
- Deterministic approach
- Focus on Lustre specifications

Introduction

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

or

- structural
  - ➡ domain → subdomains
- deterministic

or

- probabilistic
  - ➡ coverage criteria

Test execution

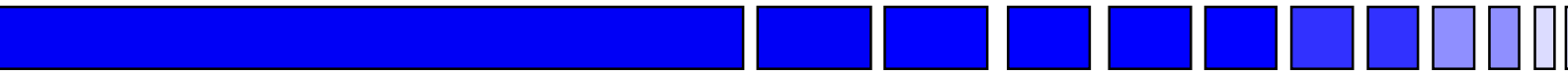
- good modularity
- adequate entry points
- adequate observation points

strong connection

- 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 case

Quantitative issues

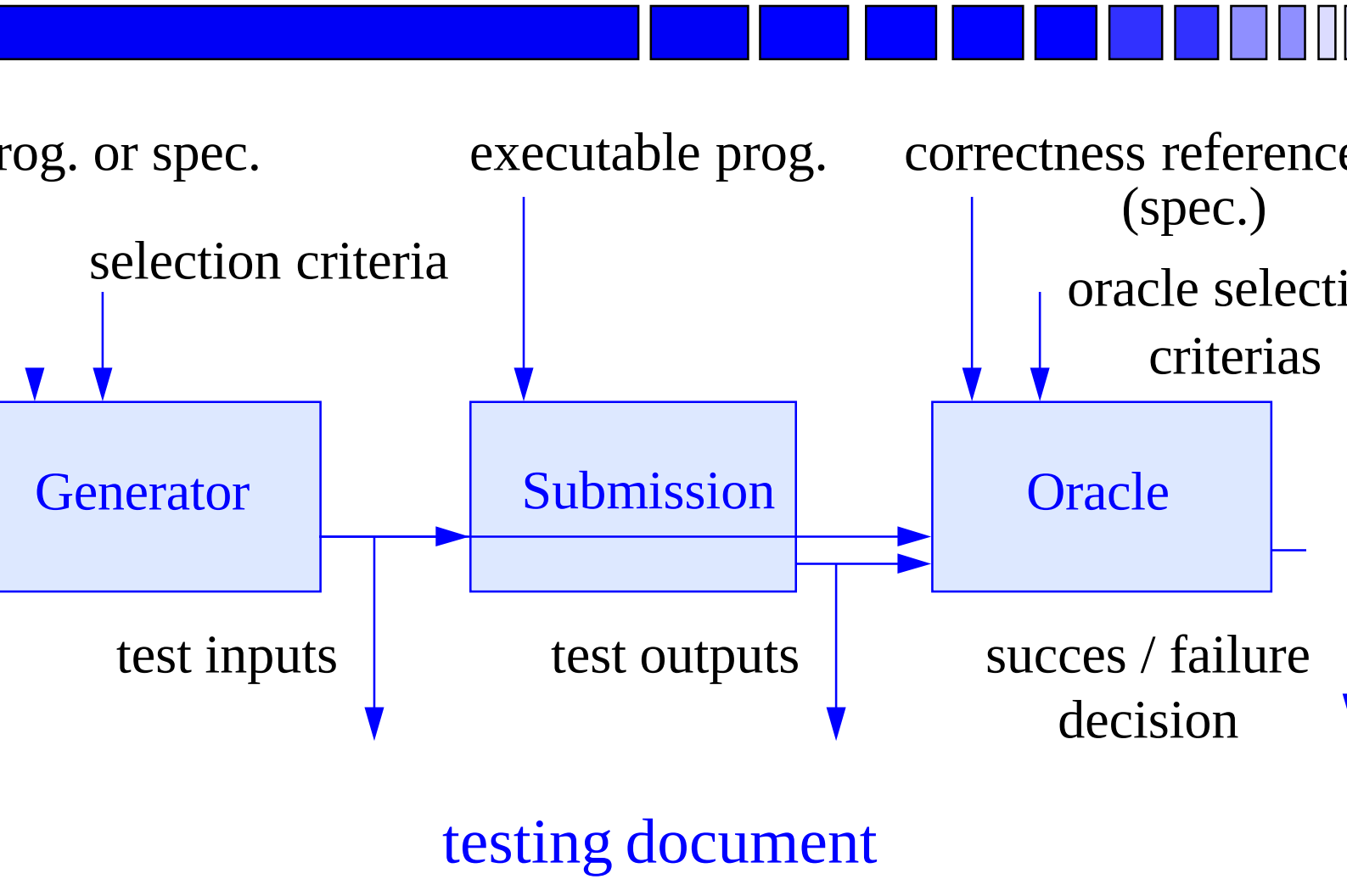
" I guarantee that the rate of failure will be less than ε "

Is a non-sense without a risk α to be wrong w.r.t. this affirmation.

Formal specifications can let you save money

- cost of 1 test ≈ 1/2 engineer day
- computer aided selection and oracle < 1 min
- automatic manipulations
  - require formal specifications

Testing automation



testing document

What is a formal specification ?

- program interface description

  sorted : List → Bool

- properties

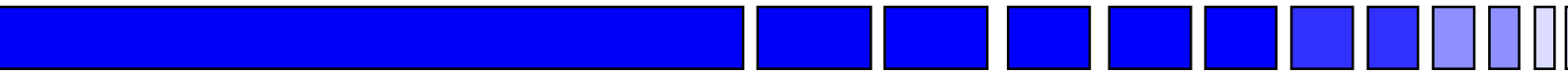
  sorted([ ]) = true

  sorted([x]) = true

  sorted([x, y | L]) =

  (x ≤ y) and sorted([y | L])

What is a formal test ?



est = formula without variable

operation(inputs) = output

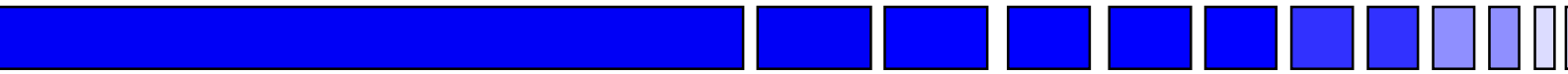
sorted([1, 2, 3]) = true

uch better:

observable formula deduced from the specification

sorted([1, 2, 3]) = (1 ≤ 2) and sorted([2, 3])

Plan



- Main difficulties
- Contributions of formal methods

- Probabilistic approach
- Deterministic approach
- Focus on Lustre specifications

Probabilistic testing

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

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

$$N \geq \log(\alpha) / \log(1- \varepsilon)$$

Choice of the test cases

ow to produce the N relevant test cases:

= a complete distribution on the domain of variables (has to be discussed with the client)

roblems:

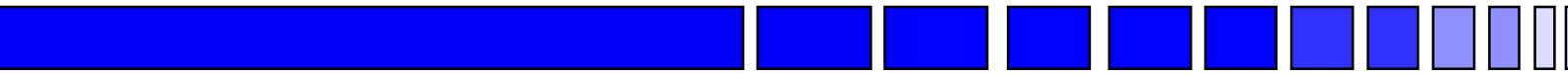
- to formalize the discussion into μ
- to generate test cases according to μ

To automate the probabilistic test

A prototype of generator

- generates tests from a set description of domains of variables (cartesien 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

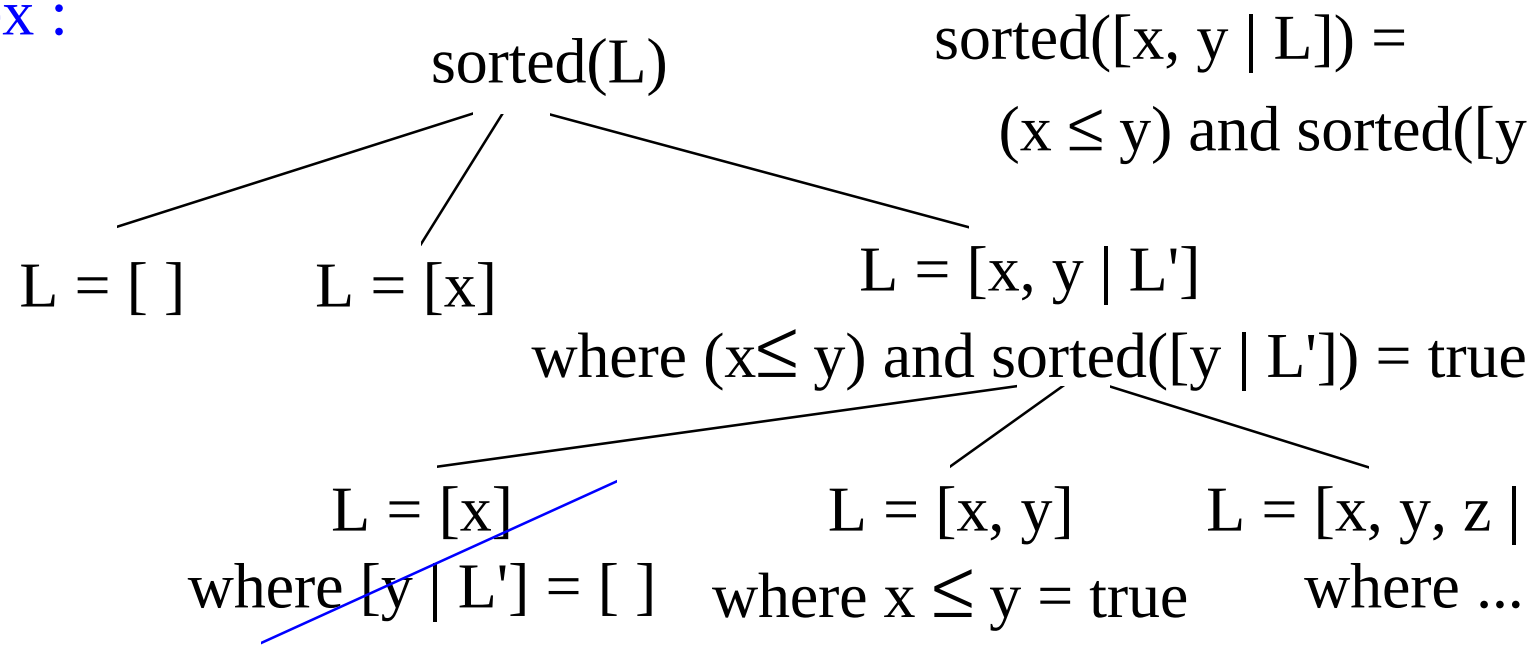
sorted([ ]) = true

sorted([x]) = true

sorted([x, y | L]) =

(x ≤ y) and sorted([y |

ex :



To automate deterministic testing

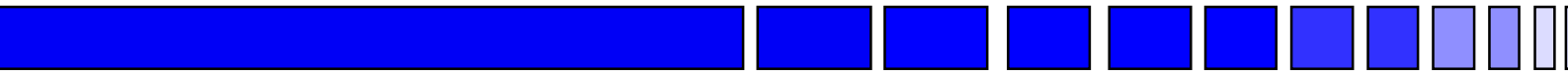
- solve constraints for each domain

- generate any one value in the domain
  - ➡ use constraint solving methods
    (logic programing techniques)

dvantages 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

Application to the Lustre langage



Lustre is a functional and dataflow language

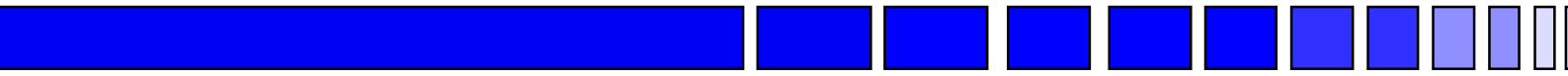
a Lustre node as a cyclic behavior

```
ode mem(On : bool ; Of : bool ; Init : bool)
turns (Out : bool) ;
t
ut = 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 = if B then C else D
  - – 2 cases: B = (..., true)
    B = (..., false)
- A = B $\rightarrow$ C
  - – 2 cases: last cycle = first cycle
    last cycle = further cycle

Coverage criteria



to cover all operators :

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

roduces 4 test cases:

mem((..., true), (..., _), (..., _)) = (..., true)

mem((..., false), (..., true), (..., _)) = (..., false)

mem((false), (false), (V)) = (V)

mem((..., _, false), (..., _, false), (..., _, _)) = (..., V, V

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 genered in 20 s. per case
  - 33 test cases genered in 35 s. per case
- no limit to the test quality

Conclusion

Formals specifications allow to automate testing activities, including Oracle.

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