Model based testing of reactive systems. Intelligent approaches

Annamaria Szenkovits

Scientific Superviser: Prof. Dr. Pop Horia Florin
Department of Mathematics and Computer Science
Babeș-Bolyai University Cluj-Napoca

May 9, 2014

Siemens-BBU PhD program

Overview

1. Model-based testing
   - Testing
   - Process of model based testing
   - Model based testing vs. manual testing
2. Intelligent approaches
   - Statistical testing
   - Evolutionary testing
   - EDAs
3. Model-based testing of reactive systems
   - Reactive systems
   - Railway automation
4. Perspectives
   - Topics to be investigated
   - Modelling languages and tools
     - Lustre
     - Lurette

Testing

- crucial step in the software development life-cycle
- it is common to dedicate between 30 and 60 % of the project resources to this step [7]

Process of MBT

1. Model the system under test (SUT) and/or its environment.
2. Generate abstract tests from the model.
3. Concretize the abstract tests to make them executable.
4. Execute the tests on the SUT and assign verdicts.
5. Analyze the test results. [6]

Process of MBT

- **main idea**: derive a model based on the abstractions of the system under test (SUT) and/or its environment and then generate test cases based on this model

Process of MBT

- **main idea**: derive a model based on the abstractions of the system under test (SUT) and/or its environment and then generate test cases based on this model

- the automation of **black-box test design**
- generation of **executable test cases** that include **oracle information** (expected output values of the system under test)

- instead of defining individual test cases (like in the classical testing methods) $\rightarrow$ a **model of the software behavior** is created and specific test cases are derived from this model and are applied to the SUT
- **generation of the test cases**: automatically or semi-automatically $\rightarrow$ the costs related to testing can be significantly reduced [7]

- instead of defining individual test cases (like in the classical testing methods) $\rightarrow$ a **model of the software behavior** is created and specific test cases are derived from this model and are applied to the SUT
- **generation of the test cases**: automatically or semi-automatically $\rightarrow$ the costs related to testing can be significantly reduced [7]
- **manual testing**: low initial design costs, but their ongoing execution costs make them expensive as the number of releases increases
- **automated test scripts**: more expensive to design initially but can become cheaper than manual testing after several releases [6]

Statistical testing

- **test profiles** and **sizes** are derived based on structural and functional criteria [5]

Evolutionary testing

- produce effective solutions for complex and poorly understood search spaces with multiple dimensions
- the dimensions of the search spaces are directly related to the number of input parameters of the SUT

Evolutionary testing

- produce effective solutions for complex and poorly understood search spaces with multiple dimensions
- the dimensions of the search spaces are directly related to the number of input parameters of the SUT
- **white-box testing**: find test data with long or short execution times
- **control-flow graph**: find feasible distinct paths [1]

Evolutionary testing

- produce effective solutions for complex and poorly understood search spaces with multiple dimensions
- the dimensions of the search spaces are directly related to the number of input parameters of the SUT
- **white-box testing**: find test data with long or short execution times
- **control-flow graph**: find feasible distinct paths [1]
- methods used for test data generation: heuristic combinatorial optimization techniques. ex: **Simulated Annealing**, **Tabu Search**, **evolutionary algorithms**

Evolutionary testing

- produce effective solutions for complex and poorly understood search spaces with multiple dimensions
- the dimensions of the search spaces are directly related to the number of input parameters of the SUT
- **white-box testing**: find test data with long or short execution times
- **control-flow graph**: find feasible distinct paths [1]
- methods used for test data generation: heuristic combinatorial optimization techniques. ex: **Simulated Annealing**, **Tabu Search**, **evolutionary algorithms**
- in the class of evolutionary algorithms: **the Genetic Algorithms** (GAs) are one of the most popular and best known techniques for solving optimization problems ([2], [4])

Genetic Algorithms

The GAs are a population based search method and they involve the following **main steps**:

1. Set of individuals or candidate solutions to the optimization problem is created. This set is referred to as a **population**.
2. **Promising individuals** are selected from the population.
3. A new population is generated based on the selected individuals using **crossover** and **mutation** operators.

Estimation of distribution algorithms

- GAs: applied to many problems with good results, BUT:

Estimation of distribution algorithms

- GAs: applied to many problems with good results, BUT:
    - the determination of the **parameters** needed by the algorithm (crossover and mutation operators, probabilities of crossover and mutation, population size, number of generations, etc.) can be **difficult** and can lead itself to an optimization problem

Estimation of distribution algorithms

- GAs: applied to many problems with good results, BUT:
  - the determination of the **parameters** needed by the algorithm (crossover and mutation operators, probabilities of crossover and mutation, population size, number of generations, etc.) can be **difficult** and can lead itself to an optimization problem
- the Estimation of Distribution Algorithms ([3]) can eliminate these problems
  - instead of the classical crossover and mutation operators: the **probability distribution of the selected individuals is estimated** and this distribution is than sampled to create the **next population**

Estimation of distribution algorithms

- GAs: applied to many problems with good results, BUT:
  - the determination of the **parameters** needed by the algorithm (crossover and mutation operators, probabilities of crossover and mutation, population size, number of generations, etc.) can be **difficult** and can lead itself to an optimization problem
- the Estimation of Distribution Algorithms ([3]) can eliminate these problems
  - instead of the classical crossover and mutation operators: the **probability distribution of the selected individuals is estimated** and this distribution is than sampled to create the **next population**
  - based on the experiments: the EDAs obtain the same quality result with less generations as other evolutionary algorithms ([2])

Reactive systems

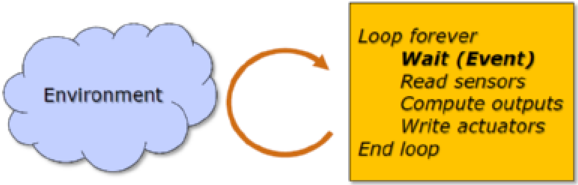
- permanently interact with their **environment**
- starting from some initial input, they will continue to interact with their environment during the course of their execution

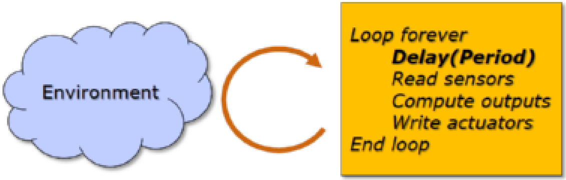
Reactive systems

- permanently interact with their **environment**
- starting from some initial input, they will continue to interact with their environment during the course of their execution

- usually modelled as **state machines**

Reactive systems





Application area: Railway automation domain

Safety-critical system

- system whose failure or malfunction may result in
  - death or serious injury to people
  - loss or severe damage to equipment
  - environmental harm

Application area: Railway automation domain

Safety-critical system

- system whose failure or malfunction may result in
  - death or serious injury to people
  - loss or severe damage to equipment
  - environmental harm

regulated by **standards**

- European Committee for Electrotechnical Standardization - **CENELEC**
- EN50126

Application area: Railway automation domain

Safety-critical system

- system whose failure or malfunction may result in
  - death or serious injury to people
  - loss or severe damage to equipment
  - environmental harm

regulated by **standards**

- European Committee for Electrotechnical Standardization - **CENELEC**
- EN50126
- model-based engineering is **strongly recommended**

Goal

- adapt existing intelligent methods for reactive systems
- improve existing methods

Goal

- adapt existing intelligent methods for reactive systems
- improve existing methods

- real-world, industrial application in the domain of **railway automation**

Test input generation

- each reaction: read inputs $\rightarrow$ compute outputs $\rightarrow$ update the internal state of the system
    - $\rightarrow$ the tester has to provide test sequences: sequences of input vectors

Test input generation

- each reaction: read inputs $\rightarrow$ compute outputs $\rightarrow$ update the internal state of the system
  - $\rightarrow$ the tester has to provide test sequences: sequences of input vectors
- a reactive system controls its environment $\rightarrow$ the input vector at a given reaction may depend on the previous outputs
  - $\rightarrow$ input sequences cannot be produced off-line, and their elaboration must be intertwined with the execution of the SUT

Control flow vs. data flow

- feasible test cases
- coverage measure

Control flow vs. data flow

- feasible test cases
- coverage measure

Fitness function for EAs

- maximize the number of **distinct feasible paths**
- maximize coverage

Lustre

- **synchronous language**: optimized for programming reactive systems
- based on the **data flow model**[1]
- designed for the description and verification of reactive systems
- can be used for both writing programs and expressing **program properties**.

[1] Lustre home page: http://www-verimag.imag.fr/Lustre-V6.html; last visited: 17/02/2014

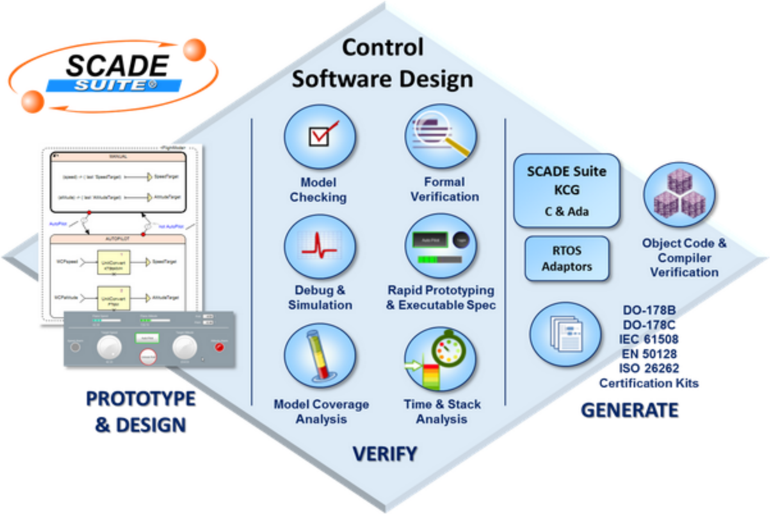
Lustre

- functional language
- structured on **nodes**:
  - represents a program or a subprogram
  - operates on **streams**: a finite or infinite sequence of values of a given type
- a program has a **cyclic behavior**: at the *n*th execution cycle of the program, all the involved streams take their *n*th value

Code example

SCADE

Safety Critical Application Development Environment



Lurette

- automatic test generator for reactive programs that focuses on **functional testing**
- based on the descriptions of the **environment** (constraints) and the expected properties of the SUT
- constraint: boolean or numerical
  - solve the constraints and randomly generate inputs that satisfy the assertions
- **description of the environment**:
  - an **automaton**: each transition is associated to a set of constraints that define the possible outputs
  - **weight**: defines the relative probability for each transition to be taken

Code example

- implementation: Lustre, Lurette
    - online learning for improving the weights

- implementation: Lustre, Lurette
  - online learning for improving the weights

- automatic test generation module $\rightarrow$ SCADE integration

References

[1] Baskiotis, N., Sebag, M., claude Gaudel, M., and Gouraud, R.
Exist: Exploitation/exploration inference for statistical software testing.
In *On-line Trading of Exploration and Exploitation, NIPS 2006 Workshop* (2006).

[2] Inza, I., Larraaga, P., Etxeberria, R., and Sierra, B.
Feature subset selection by bayesian network-based optimization.

[3] Larraaga, P., Etxeberria, R., Lozano, J. A., and Pea, J. M.
Combinatorial optimization by learning and simulation of bayesian networks.
In *in Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (2000), Morgan Kaufmann, pp. 343–352.

[4] Sagarna, R., Lozano, J. A., and Lardiazabal, P. M.
On the performance of estimation of distribution algorithms applied to software testing.

[5] Thvenod-Fosse, P., and Waeselynck, H.
On functional statistical testing designed from software behavior models.
In *Dependable Computing for Critical Applications 3*, C. Landwehr, B. Randell, and L. Simoncini, Eds., vol. 8 of *Dependable Computing and Fault-Tolerant Systems*. Springer Vienna, 1993, pp. 3–28.

[6] Utting, M., and Legeard, B.
Practical Model-Based Testing: A Tools Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[7] Utting, M., Pretschner, A., and Legeard, B.
A taxonomy of model-based testing approaches.
Softw. Test. Verif. Reliab. 22, 5 (Aug. 2012), 297–312.

Thank you for your attention!