Search-based Functional Test Data Generation Using Data Metamodel

János Oláh and István Majzik

SSBSE 2011
10-12th September, Szeged

DOMAIN INFORMATION

We assume that at the beginning of the software development process, the domain experts and the software developers create the high-level description of the operating domain of the system under test (SUT) and define the high-level requirements.

INPUT

The input for the test data generation process is given in the form of a metamodel, well-formedness constraints and goal samples. The metamodel describes the possible objects in the context of the SUT and the constraints are applied to these objects. Goal samples define certain configurations, that should be included in the test data.

METAHEURISTICS

The aim of the test data generation process is to find test data (i.e., model instances conforming to this metamodel), which are adequate to test certain functions of the SUT. Their fitness is measured with respect to the given goal samples. Metaheuristics are used to find the acceptable test data.

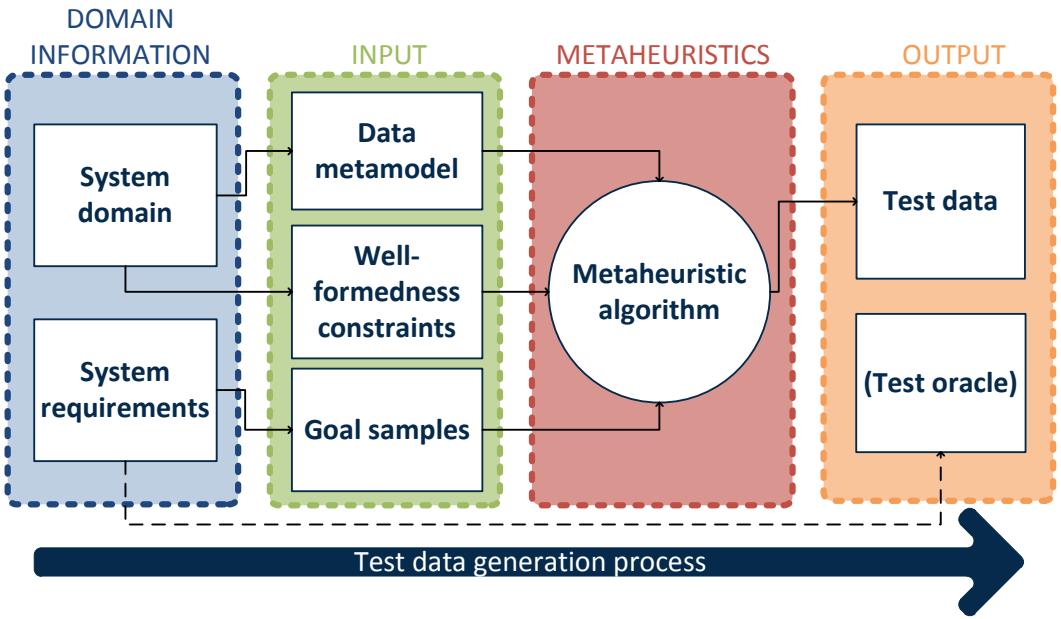
OUTPUT

The result of the search-based test data generation process is a candidate solution, whose fitness exceeds a given threshold, i.e., it contains the required goal sample and feasible according to the given well-formedness constraints.

Description of the **system's domain** and the **general requirements** may use arbitrary representation, e.g., plain text, diagrams, ontologies.

Data metamodel is created by domain and testing experts from the high-level description. Generated test data contains instantiated objects from this metamodel, thus **test data is a model conforming to this metamodel**.

Goal samples define a configuration of objects that have to be present in the generated test data in order to use that data to test a certain function of the SUT. **Well-formedness constraints** describe configurations that should be satisfied by the generated test data. Since test data is given in the form of a model instance, constraints and goal samples may be given in the form of **model patterns**, conforming to the same metamodel as the test data.



Although the introduced approach only focuses on the generation of input test data, we may utilize the model-based description of the requirements to exploit **test oracle** for the testing process. If we are able to create a metamodel for the output domain of the SUT, the expected output can be defined as a model pattern as well and the success of the test execution can be evaluated based on the appearance of that pattern in the output of the SUT.

The resulting **solution** (or set of solutions) is a model instance, that contains **instantiated objects** from the metamodel of the input data. Although it is a high-level representation of test data, it can be mapped to specific description of test data (e.g., with an XSLT transformation).

The **fitness** of a candidate solution (i.e., test data) is measured according to the goal samples. An adequate solution contains all required goal samples given in the input of the process, and comply with the defined constraints.

The iterative search for the test data requires the continuous manipulation of model instances, which manipulation is possible via **model transformations**. Moreover, since the goal samples and the constraints are represented as model instances conforming to the same metamodel, calculation of the fitness can be performed with **pattern matching**. This operation returns the number of occurrences of a pattern (i.e., the goal sample) within a model (i.e., candidate solution). Then the fitness value can be calculated using the number of occurrences.

Using this model-based representation, the continuous manipulation of model instances and pattern matching can be executed with popular **model transformation frameworks.** Such frameworks usually apply **graph transformation**, which technique provides a pattern and rule based manipulation of graph models. Eclipse Modeling Framework is a tool for metamodeling and manipulating the generated model instances. Moreover, there are development environments for graph transformation systems (such as AGG or VIATRA) that can be used for the construction of a metaheuristic search technique.

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group
{janos.olah,majzik}@mit.bme.hu

R3-COP
Robust & Safe Mobile Co-operative Autonomous Systems