

Automated Requirements Testing with Abstract Oracles

Bernhard K. Aichernig
Institute for Software Technology
Technical University Graz, Austria
aichernig@ist.tu-graz.ac.at

During the last few years the interest in formal software development has been growing rapidly. One of the main reasons for this is the availability of tools to assist the developer in using these formal methods. The author, too, has contributed to the growing computer aid by an extension of a commercial tool [1]. However, formal methods are not often applied in industrial projects, despite the maturity of the theories and tools. Several reasons can be identified for the absence of formality in the software development process: Too many different notations have been invented, a lack of integration into informal approaches, and the strong emphasize on formal proofs.

Consequently, instead of promoting formal correctness proofs, we regard the automation of functional testing as the next step in a smooth integration of formal methods to raise the level of reliability, after formal specification techniques have been introduced. The Vienna Development Method (VDM) [4] serves to demonstrate how a well-established formal method supports the automation of testing. VDM is supported by CASE-tools and even allows an integration into informal methods like UML or Structured Analysis, and so does our testing approach.

Previous work has shown how test-cases may be derived out of formal specifications [2]. However, little attention had been given to the fact that formal models of software requirements are inherently abstract in the sense that detailed design decisions are not included. Consequently, the test-cases, derived (generated) from such an abstract model, are abstract, too, and thus inappropriate for a direct automatic test of a target sys-

tem. For that reason, a mapping between abstract and concrete test data is required.

The presented framework focuses on the usage of formal requirements specifications as test oracles for concrete implementations. The approach is based on the formal definition of abstraction as a homomorphism, called the retrieve function, which maps the concrete level into the abstract. If the retrieve function is implemented and the post-condition is executable, then the model may serve as a test oracle and specification as well. In Fig-

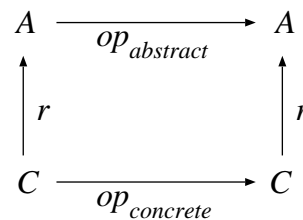


Figure 1. The morphism of abstraction.

ure 1 our notion of abstraction is represented by a commuting diagram, where operations (op) are viewed as functions from input to output states. The abstract operation $op_{abstract}$ manipulates an abstract state representation of type A . The concrete operation $op_{concrete}$ incorporates detailed design decisions and maps a refined input state to an output state. Examples for such data-refinement would be to implement a set through a balanced binary-tree, or the other way round, to view a data-base system as a set of data. The relationship between the abstract and the concrete is defined by a function r mapping a concrete state to its abstract counterpart. The diagram shows that the

retrieve function r is a homomorphism for which the following formula holds:

$$\forall in : C \cdot op_{abstract}(r(in)) = r(op_{concrete}(in))$$

In VDM the functionality of a system is specified by means of implicit operations defined by pre- and postconditions. Preconditions are constraints on the input (state) for which an operation is defined. Postconditions are predicates defining a relation between an input, the state before and after an operation's application, and its output. From a testers perspective the preconditions define the input values for 'good' tests, where valid outputs can be expected. An postcondition predicate may be viewed as a test oracle, evaluating to true, if its operation invoked with a test-input yields a valid (specified) result. Modern tools allow the interpretation or even code-generation of such postconditions which leads to an automated test evaluation through postcondition-oracles.

The combination of the notion of abstraction and the possibility to use specifications as test oracles leads to a testing framework with abstract oracles. In Figure 2 the data-flow of the new testing scenario is presented. An implementation is

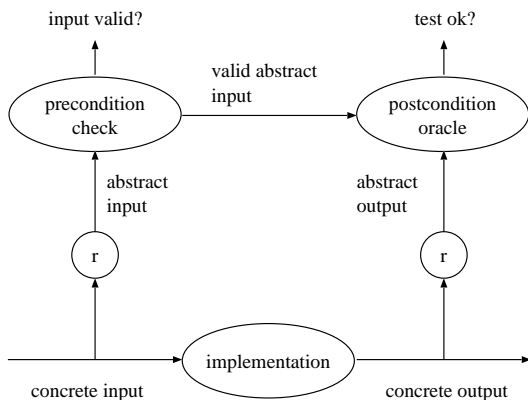


Figure 2. DFD of the testing scenario.

executed with a concrete test input (state). An implemented retrieve function maps the concrete input and output to its abstract representations. A precondition check validates the input and feeds it into the oracle which checks the relation to the produced output. If the postcondition evaluates to true, the test passed.

The advantage of the approach is the automation of black-box testing by the usage of structural test-data. Another possibility would be to code the reverse of r in order to test with abstract test-data derived from the specification. In an object-oriented setting the retrieve functions and code-generated pre- and post-conditions could be part of a testing class, inherited from the class to be tested. Currently code generators for C++ and Java [3] are available to support the testing approach.

References

- [1] B. K. Aichernig and P. G. Larsen. A proof obligation generator for VDM-SL. In J. Fitzgerald, C. Jones, and P. Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods*, volume 1313 of *Lecture Notes in Computer Science*, 1997.
- [2] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In J. Woodcock and P. Larsen, editors, *FME'93: Industrial-Strength Formal Methods*. Springer-Verlag, April 1993.
- [3] IFAD. IFAD's products. WWW at URL <http://www.ifad.dk/Products/products.htm>.
- [4] C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall International, Englewood Cliffs, New Jersey, second edition, 1990.