Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 320 (2016) 3–19

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# Symbolic Input-Output Conformance Checking for Model-Based Mutation Testing

Bernhard K. Aichernig and Martin Tappler[1]

*Institute for Software Technology*
*Graz University of Technology, Austria*

**Abstract**

This paper presents an approach to use symbolic input output conformance checking for mutation-based test case generation. In this approach, a possibly non-deterministic action system model is used as basis for generating a number of mutants. Subsequently after the generation of mutants, the original model and the mutants are simultaneously symbolically executed and tested for conformance. Distinguishing test cases are generated, if non-conformance is detected during this process. Several optimisations of the conformance check are presented and their effectiveness is underpinned by listing experimental results.

*Keywords:* model-based testing, mutation testing, symbolic execution, action systems, sioco,ioco.

## 1 Introduction

In order to assure that a system under test (SUT) fulfils given requirements it is commonly executed and tested under conditions specified through a set of test cases. Traditional software testing however suffers from a number of drawbacks. It is for instance inherently incomplete and since it is a manual task, it is labour intensive and error-prone. Model-Based Testing aims at improving upon this situation, by utilising abstract models of the SUT [21]. Most importantly, models allow the automatic generation of test cases based on some criterion. Hence, the ad hoc nature of software testing is replaced by a well-defined process.

Model-Based Mutation Testing uses a fault-based approach to test case selection [6]. More concretely, mutation-based test case generation consists of two main steps: (1) mutated models are generated by injecting faults into the original model and (2) test cases are generated, which would reveal non-conforming behaviour of mutants. The rationale behind this approach is that a SUT implementing a non-conforming mutant would be detected to be faulty by the test case corresponding to the mutant.

[1] Authors are listed in alphabetical order.

http://dx.doi.org/10.1016/j.entcs.2016.01.002

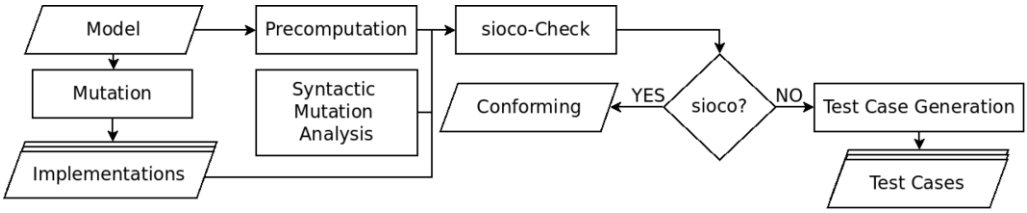


Fig. 1. The model-based mutation test case generation process.

There is a variety of conformance relations in use today, like refinement [5] or Input Output Conformance (**ioco**) [2]. In the following, we will use Symbolic Input Output Conformance (**sioco**) as defined by Frantzen et al. [14] to decide conformance between mutants and the original model. We have chosen **sioco** as it is well-suited for reactive systems and its symbolic nature provides a solution to the state space explosion faced when working concretely. However, we rather test for non-conformance than for conformance in the test case generation process depicted in Figure 1. More specifically, we generate sequences of actions and conditions leading to states where non-conformance of first-order mutants, i.e. mutants created by injecting a single fault, may be observed.

The contribution of our work is twofold. First, we give a formalisation of the subsequently introduced variant of the action system formalism. For this purpose, we will adapt the symbolic framework given in [14] and follow the same style. Second, we will present an efficient fully symbolic check for non-conformance between two action systems, which may behave non-deterministically. We demonstrate the efficiency through a comparison with a previously implemented concrete **ioco** checker.

## 2 Action Systems

Action systems were first defined by Back and Kurkio-Suonio [7] as a modelling formalism for distributed systems. We have chosen this formalism as it can effectively be used for modelling reactive systems [8] and because recently, it has also been used for model-based mutation testing [5]. There exist several variations of it like object-oriented action systems [9] and it also served as an inspiration for Event-B [1]. However, the action system formalism used here is more restricted than other variations. In some aspects it is similar to the Event-B language, but for instance it does not support set-theoretic constructs like Event-B.

Informally, the execution of an action system starts in an initial state which is manipulated by repeatedly executing actions. During this process one action is chosen at each step in a non-deterministic fashion from the set of enabled actions. An action is enabled iff its guard is satisfiable in the current state. The execution terminates when the set of enabled actions is empty.