Subjects

We will give a detailed account to formal approaches of testing as well as relations to other validation techniques like model checking.

There are several schools of formal testing. We will give an account to some of them providing the basis for a comparison. We will show the similarities and differences in each approach. In general, we concentrate on so-called black-box testing. In this setting, the system under test is not given explicitly but may only be analyzed using interaction in terms of input and output actions.

T1: Testing of finite state machines (5 chapters)

Coordinator: Martin Leucker and Bengt Jonsson

During the last 50 years, a rich testing theory has been developed for testing finite state machines that produce outputs on state transitions after receiving inputs (also known as Mealy machines). Typical problems addressed in the domain of finite state machine testing are

- **Homing Sequence**: Determine the final state after the test: Rezeq Naim (38) & Sandberg (37)
- **State Identification**: Identify the unknown initial state: Molderez (27)
- **State Verification**: Verify that a machine is in a particular state Bjoerklund (36)
- **Machine Verification/Fault Detection/Conformance Testing**: Check whether two given machines are equivalent: Cooper (40) & Gargantini (32)
- testing nondeterministic systems: Krcal (30) & Roehl (16)

Further keywords in this area are transition tour, postman, unique input/output sequence etc.

An overview of testing finite state machines is given in [LY96]. It will serve as a starting point for the chapters dealing with testing finite state machines.

T2: Preorder-based testing of labeled transition systems (5 chapters)

Coordinator: Joost-Pieter Katoen

The theory of testing transition systems was initiated by the work of De Nicola and Hennessy in [MN83]. The basic goal is to test whether one transition system is an implementation of another one. Therefore, several notions of implementation relations were introduced and studied. A usually infinite number of test cases is identified that characterizes the implementation relation. The work was later
extended towards transition systems in which input and output actions are distinguished.

In [BT01], an annotated bibliography for this approach is given which points out the relevant work in this area.

Recently, extensions to cover real-time or probabilistic systems have been studied, for example in [SVD01] and [SV03].

To turn this approach into a practical method, a finite number of test cases has to be selected out of the infinite set describing the implementation relation. The goal is to choose test cases that cover the infinite set, as good as possible. This approach is studied in [FGMT02, Bri94].

This model also assumes that the tester communicates with the system under test in a synchronous manner. In practice, however, the two systems often interact via queues. [TV92] discusses this problem.

Chapters:

T2.1 Preorder relations: Bruda (44)
T2.2 Test generation algorithms based on preorders: Tschaen (13)
T2.3 I/O-automata based testing: vd Bijl (8) & Peureux (20)
T2.4 Testing theory for timed systems: Briones (34)
T2.5 Testing theory for probabilistic systems: Wolf (29) & Tepper (39)

T3,T4: Model-based testing—tools and case studies (5 chapters)

Coordinator: Alexander Pretschner

In the setting of model-based testing, a formal model of an implementation is given. The goal is to check whether the implementation conforms to the model. Usually, testing is restricted to certain interesting test cases for which conformance is checked. Here, a test case is a sequence of input and (expected) output values. Test cases are usually determined in terms of test specifications that specify which part of the system is to be tested. Often, test specifications are given in form of coverage criteria.

Typical models are extended finite state machines in which states are enriched with data or time variables. Thus, the model is usually not concrete in the sense that every state of the system is represented as a single state in the model. It is abstract so that abstract test cases require instantiation into concrete ones.

There is a huge number of papers on specifying and generating test cases. A selection of these methods describing basic ideas will be presented. The focus is on techniques rather than on which test cases to specify. One approach, for example, is to specify coverage criteria by means of temporal logic. Test cases are then witnesses or counter examples for the underlying formula and the model. Among model
checkers, test case generation technology includes symbolic execution or deductive theorem proving.

This yields the following chapters. By no means are the references intended to be comprehensive, and the authors are encouraged to add further literature.

T3.1 technology: model checking [ABM98, HLSU], symbolic execution [Kin76, Cla76, How77, How78, RHC76, LP01, PPS+03], LTS exploration [RWNH98, dZ99], theorem proving [HNS97, BCM01, Sad98]: Hong (6) & Samer (9) & Lucio (25)

T3.2 Real-Time and Hybrid Systems Testing [PS97, Nie00, HPPS03, Weg01]: Hessel (2) & Berkenkotter (22) & Kerin (33)

T3.3 Coverage based test case generation [HLSU, RH01, Pre03, WBP02], as compared to functional test case specifications and random testing [DN84, Nta98, HT90, Gut99] and user profiles: Gaston (18) & Seifert (17) & Mueck (15)

T4.1 Tools for test case generation. Some of the tools have been compared in [Gog01, Har02]: Belinfante (23) & Frantzen (28) & Schallhart (10)

– Synchronous languages: for Lustre [GaTeL [MA00], Lurette [RWNH98], Lutess [dZ99]), for AutoFocus [PPS+03]

– process algebras, state machines [Ura92, BDA96], SDL: TGV [FJJV96], STG [CJRZ02, RdJ00], TVEDA [CGPT96], TorX [VTB+00], AutoLink [KGHS98]

T4.2 Case studies: chip cards [PPS+03, LP01, CJRZ01], processors [DBG01, SA99, FKL99], parts of OS and exception handling [FHP02]: Prenninger (48) & El-Ramly (14)

T5: Test execution (2 chapters)

Coordinator: Martin Leucker

To allow different testing tools to be employed in practice, standards have to be developed for test execution, i.e., the interaction of a testing tool with the system under test. TTCN-3 [Wil01] is a language for describing the test setup as well as test cases. A compiler then produces a program that runs the given tests cases: Din(7). ISO 9646 as well as IEC 1131 and related standards should be taken into account: Zhen Ru Dai (24).

T6: Model checking, runtime verification, and testing (3 chapters)

Coordinator: Martin Leucker

T6.1 In the domain of run-time verification, one studies the question how to check properties of a system while executing it. In [HR02], for example, monitors
for temporal logic formulas are generated that check whether a property is satisfied or violated. Obviously, there is a strong relationship between model-checking, run-time verification, and testing which will be elaborated in this part¹: Colin (21) & Mariani (31) & Omitola (26)

T6.2 Model checking is one of the key techniques in verifying hard- but also software systems. The formal theory of model checking is well developed in contrast to the domain of testing. It is, however, likely that many achievements in the domain of model checking are valuable for formal testing as well. Recently, several attempts have been made. Adaptive model checking [GPY02] combines learning [Ang87] of automata and model checking to study an underlying system: Berg (1) & Raffelt (43)

References


¹Note that the issue is to compare the power of the different validation techniques, rather than studying the question whether one domain can be used for the other one. Such questions, like how model checkers can be used to generate test cases, are studied in the appropriate testing chapters.


8