Test Automation for Reactive Systems Theory and Practice

organized by

Prof. Dr. E. Brinksma (University of Twente, The Netherlands)
Prof. Dr. J. Peleska (University of Bremen, Germany)
Dr. M. Siegel (University of Kiel, Germany)

Preface

The design and implementation of correct reactive systems is one of the major challenges of information technology. More than ever before modern society has become critically dependent upon applications of such systems. Important application areas include telematic systems, such as communication protocols and services, process control systems, and embedded-software systems, ranging from aircraft control to consumer products like television sets. Most applications are critical in one or more aspects, e.g. in the context of safety, economy, ecology etc.

Testing is an indispensable technique for the validation of reactive systems. Although other tool supported methods, such as simulation, model checking and verification by theorem proving, are increasingly being applied to assess the correctness of system designs, testing remains the practically most widely used validation technique. Although testing is generally too weak to guarantee correctness, its advantages are that it is more readily applicable to large industrial systems, and that it is in fact the only validation technique that can link the functionality of a (physical) realization and its formal specification.

On the one hand, the great growth of reactive system applications and with that the growing need to assess their correctness, has brought about a considerable practical interest in the improvement of their testing procedures. On the other hand, academic researchers, who formerly regarded testing as an inferior and superfluous validation method, have begun to investigate the use of formal methods and tools to specify, generate and implement tests. This is of great practical interest, as most test suites for reactive systems are still being produced on the basis of ill-understood ad hoc procedures. As a result the production and maintenance of test suites have become very expensive activities, presenting a considerable practical bottleneck. Even minor improvements on existing working procedures are economically attractive. Major improvements are expected from algorithms for the (semi-)automated derivation and selection of test suites from formal specifications of the implementations under test, as well as the machine-assisted evaluation of test results.

In spite of these bright prospects for the application of testing theory and related tools, still much work is needed to turn the existing theories and tools into an effective testing workbench fit for industrial application.

The Dagstuhl workshop on test automation for reactive systems was the first of its kind, and was very well attended (more than 40 participants). It brought together people from different backgrounds with a strong interest in testing, such as protocol testing, real-time testing, performance testing, statistical testing, etc., to present and discuss the state-of-the-art in testing and test automation. There was a strong representation from the formal methods community, but here again a wide range of models and formalisms was represented: Mealy automata, labeled transition systems, abstract data types, I/O automata, timed automata, Z etc.

The presentations and discussions made clear that although there are quite a few research groups active in the area now, we are only at the beginning of things and major contributions to the (semi-)automation of practical software testing are still to come. A bottleneck is the limited availability of formal system specifications that can be used to derive tests from. In certain areas, however, such as e.g. safety-critical and embedded systems, the willingness of industry to go more formal is growing, which is among others motivated by the potential gains for testing implementations.

A discussion session that was held to talk about future research themes in test automation suggested at least the following topics: relating test methods to common design and implementation methodologies (e.g. objected oriented design), the issue of design for testability, which has been very successful in the area of hardware testing, and measures to quantify the coverage of tests. The general opinion was that the seminar was a success and it is planned to organize another Dagstuhl meeting on testing in the future.

The support of the TMR Program of the European Community is gratefully acknowledged.

The organizers

Ed Brinksma
Jan Peleska
Michael Siegel

Contents

Testing Real-Time Systems

Rachel Cardell-Oliver
http://cswww.essex.ac.uk/Research/FSS/projects/test.html

In this talk I discuss some solutions to problems in test generation for real-time systems: in formal terms, what can be claimed if an implementation passes all its specification tests? under what assumptions can these claims be made? and which types of specification yield manageable test sets? The correctness relation used is trace equivalence of specification and implementation graphs. The graphs are generated from timed transition system specifications E‖S where environment E produces inputs and system S responds with outputs. The implementation to be tested will be checked in this same environment. I distinguish between the assumptions necessary to prove the test correctness theorem and those needed in order to generate and run tests. For real-time systems the size of graph generated for a given S‖E is especially sensitive to minor changes in the specification. I propose transformation rules to reduce the graph size and thus the number of tests generated whilst maintaining real-world acceptable specifications. In particular, rules are proposed for "just sufficient" environments, for time grids over non-deterministic time bounds on actions and rules for the modular test of individual system components.

Testing Timed Automata

Frits Vaandrager
http://www.cs.kun.nl/~fvaan/

We present a generalization of the classical theory of testing for Mealy machines to a setting of dense real-time systems. A model of timed I/O automata is introduced, inspired by the timed automaton model of Alur and Dill, together with a notion of test sequence for this model. Our main contribution is a test generation algorithm for black-box conformance testing of timed I/O automata. Although it is highly exponential and cannot be claimed to be of practical value, it is the first algorithm that yields a finite and complete set of tests for dense real-time systems. In a next step we identify a subclass of timed I/O automata, the class of bounded response automata, which is sufficiently expressive to model many interesting real-time systems, but allows for much smaller test sets.

Conformance Testing with Formal Methods

Jan Tretmans

Conformance testing concerns checking the functional correctness of an implementation with respect to its specification by means of executing experiments on the implementation. These experiments, referred to as test cases, should be derived systematically and algorithmically from the specification. Test cases should be such that correct implementations are not rejected, while maximizing the chance of rejecting erroneous implementations.

The first part of the talk presents the ingredients which are needed for conformance testing based on formal methods: what is conformance in a formal setting; what is testing; what is test execution; what is test generation; and what are the requirements they should satisfy.

The second part of the talk elaborates these formal testing concepts for specifications which can be expressed as labeled transition systems. Transition systems form a mathematical model for many formal specification languages, especially in the area of reactive systems. A brief, more or less historical overview of testing theories will be given, concentrating on the ingredients explained in the first part of the talk. Testing equivalence and preorder, refusal testing, canonical testers, asynchronous testing via queues, input/output testing, testing with inputs, outputs and repetitive quiescence, and testing with input and output channels are discussed. It is shown that testing with inputs, outputs and repetitive quiescence yields an interesting testing theory, which, on the one hand, has a firm basis in the theories of testing equivalence and refusal testing, and on the other hand, leads to practically applicable test generation algorithms, which provide the theoretical basis for test generation tools like TVEDA (developed at France Telecom CNET) and TGV (developed at IRISA, France).

Test Automation for Hybrid Reactive Systems

Peter Amthor
http://www.informatik.uni-bremen.de/~amthor/

This contribution aims to establish a development queue for hybrid systems, starting with specification and verification, proceeding with transformation, animation and implementation, ending with the test of the developed system.

A hybrid system usually includes continuous activities of analog variables as well as discrete events. They are also reactive, i.e. the system's components interact continuously with each other. Very often they are safety-critical, too, so certain safety requirements have to be observed to prevent the system from catastrophic behaviour.

Here, the specification of hybrid systems is done by the use of hybrid automata. They offer constructs to deal with the continuous evolution of variables' values as well as with discrete events. Additionally, the HyTech tool provides support for the verification of hybrid systems described as the parallel composition of hybrid automata. The whole specification process follows the physical/hazard/controller-model which has been developed to design safety-critical systems.

Further, one has to transform the specification into an executable model to support its animation, implementation and test. Doing this, usually the system has to be decomposed into several components to cope with its complexity or to support distributed architectures.

The received executable model, e.g. a transition graph which can be traversed, can then be integrated into a so-called safety software architecture (SSA), where the model is executed relating one or more concrete events (program functions) to its abstract events (transition annotations).

Finally, SSA can be used to animate and implement the hybrid system, but in particular to build up a test configuration to test the correct behaviour of the controller components of the system, e.g. a temperature controller, using the other system components as the test environment.

Structured Test Case Design on the Basis of Z Specifications

Harbhajan Singh

Software testing often consumes up to 50 percent of the overall software costs. A large amount of time and money within the test process is spent due to incomplete, inconsistent or ambiguous informal specifications of the test objects. A more formal approach to the early phases of software development can reduce the error rate drastically and, in addition, can significantly improve the central testing activities like test case design and test evaluation.

This paper presents an approach for generating test cases from formal specifications written in Z by combining the classification-tree method for partition testing with the disjunctive normal form approach. Firstly, a classification tree describing high level test cases is constructed from the formal specification of the test object. Then the high level test cases are further refined by generating a disjunctive normal form for them. The refined test cases obtained this way cover all specified aspects of the system explicitly and also contain all information necessary to evaluate the test results. The proposed combination of the classification-tree method with the disjunctive normal form approach preserves advantages of both methods, overcomes most of their limitations, and can be supported by tools.

Testing Algebraic Data Types and Processes: A Unifying Theory

Marie-Claude Gaudel

This talk is based on a generic framework for test selection based on formal specifications which have been developed for several years at LRI. A notion of exhaustive test set is derived from the semantics of the formal notation and from the definition of correct implementation. Then a finite test set is selected via some selection hypotheses which are chosen depending on:

- some knowledge of the program,
- some characteristics of the specification,
- and ultimately cost considerations.

First, it is recalled how this framework is instantiated for algebraic data types. Then, it is generalized to full LOTOS specifications. This leads to a new integrated test selection strategy for full LOTOS, which improves the existing ones by exploiting at the same time the data type part and the behavior part of the specification.

TGV: Principles, Tool Architecture and Algorithms

Thierry Jeron
http://www.irisa.fr/pampa

The first part of the talk presents the main principles of TGV, a prototype tool developed by IRISA Rennes and Verimag Grenoble. Its aim is to automatically generate test cases from formal specifications in the context of conformance testing of protocols. TGV is based on testing theories which consider models of transitions systems with a distinction between inputs and outputs and a conformance relation relating implementations to specifications. In TGV, test selection is achieved by a test purpose specified by an abstract automaton. TGV is based on on-the-fly verification techniques which allow to produce a test case in a lazy way by the construction of parts of the state graph of the specification.

The on-the-fly generation induces a particular tool architecture composed of a stack of modules each of them providing an API for the construction of parts of intermediate state graphs. TGV can be connected to the API of the SDL simulator ObjectGeode and the API of the LOTOS simulator of the CADP tool-box. Using this API, the different modules perform a synchronous product of the specification test graph and the test purpose, hiding and renaming, τ^*-reduction and determinization, and finally test generation.

The second part of the talk presents this test generation algorithm (common work with Pierre Morel). The algorithm is adapted from the Tarjan's algorithm which computes maximal strongly connected components. The principle here is to build a subgraph of sequences leading to acceptor states of the test purpose. It is quite similar with the model-checking algorithm for reachability properties with adaptations for subgraph synthesis and elimination of controllability conflicts. This algorithm produces correct test cases having loops and minimal INCONCLUSIVE verdicts.

We conclude with a small example which additionally advocates for the use of static analysis on source specifications in order to avoid unnecessary unfoldings and improve test generation.

Specification-based Testing of Concurrent Systems

Andreas Ulrich

http://ivs.cs.uni-magdeburg.de/~ulrich/Papers/forte97e.ps.gz

The presentation addresses the problem of test suite derivation from a formal specification of a distributed concurrent software system by presenting a concurrency model, called behavior machine, and its construction algorithm from a collection of labeled transition systems. It then outlines how test derivation can now be based on the new concurrency model to derive test suites that still exhibit concurrency between test events. A toolset is presented to support the generation of concurrent test suites from specifications given in the formal description technique LOTOS. Finally, some comments on requirements for the design of a distributed test architecture are given.

The full paper was presented at the FORTE/PSTV 1997 Conference in Osaka, Japan.

Structured Reasoning About Z Specifications of Embedded Systems

Thomas Santen

I present an approach to work with Z specifications in a validation and verification context. The approach relies on the fact that specifiers structure specifications according to the importance of concepts and their relations. Exploiting the structure of a specification helps to identify theorems that are relevant to a practical verification task such as test case generation. It also drastically reduces the effort needed to derive theorems using an automated theorem prover.

The principal means to modularize Z specifications are schemas. They can be used in different roles, in particular to define states and operations of systems, and to define predicates on the system states. For schemas used as states, schema-views such as a disjunctive normal form of a schema's

predicate, or deriving the precondition of an operation are useful. Schema views are thus equations between schemas. For schemas used as predicates, relations between them are needed to validate a specification and to avoid referring to their definition in other proofs.

The tool HOL-Z based on the theorem prover Isabelle provides support for proof about Z. Special tactics allow one to derive schema views. Applications of these tactics to several specifications of non-trivial complexity showed that only exploiting the structure of specifications in the reasoning process makes automated deduction feasible.

I concluded my talk with the question of testing componentware. By way of the example of a traffic light safety controller, I illustrated that nowadays software in embedded systems is increasingly based on generic components that are parameterized in the configuration of a concrete application. The safety controller is parameterized not only with the configuration of a traffic junction, but also with evaluation functions judging the safety of traffic situations. How to test such a component to work for any (admissible) instantiation of the parameters is not at all obvious.

Automated Test of a Power and Thermal Controller of a Satellite

Oliver Meyer

The test system VVT-RT has been employed for an automated hardware-in-the-loop test of the power and thermal controller (PTC) of the ABRIXAS satellite, which is built by OHB, Bremen. The most critical tasks of the PTC are the charge and discharge control of the satellite battery, the temperature control of all satellite components, and the generation of status reports for the tracking station. VVT-RT is developed by JP Software Consulting in cooperation with the Bremen Institute of Safe Systems (BISS) for the automated black-box conformance test of reactive systems. Test generation, test execution and test evaluation are based on (possibly combined) CSP specifications of the target system behaviour and its environment. The environment specification is used to generate input events for the system under test (SUT) while its outputs are checked on-the-fly against the behaviour specification with respect to the previous inputs: the behaviour specification is used as a test oracle which adapts its verdicts to the preceding inputs and

outputs. If it is possible to make reasonable assumptions about the after-state of the SUT after a "fail" verdict has been issued, the test execution has not necessarily to be stopped. This is especially essential for long term tests where the continuation might reveal errors which occur periodically. In order to perform the tests of the hybrid PTC, the test system has been extended to deal not only with discrete events but also with continuous values. The analogue readings of the PTC sensors are redirected to the test system and captured and influenced in real time. In this way it is possible to simulate component faults and sensor inaccuracies. The tests revealed more than 20 errors which can be grouped into the four categories "Incomplete or ambiguous specifications", "Configuration table inconsistencies", "Coding errors" and "Incompatibilities between Hardware and Software". The errors of the first category were detected during the formalization of the PTC specifications, while the others occurred during the functional and long term tests. Especially the errors of the fourth category demonstrate the importance of hardware-in-the-loop testing: even if the software works correctly, the communication between hardware and software might introduce new problems.

Refusal Testing for Multi I/O Transition Systems

Lex Heerink

Recent developments show that testing theories for transition systems are becoming more practical. One of these developments is the explicit distinction between input actions and output actions: input actions are initiated and controlled by the tester and can be consumed by the implementation under test (IUT), whereas output actions are initiated by the IUT. Under the assumption that implementations can never refuse input (i.e. input actions are continuously enabled) it turns out that applying the traditional testing theory of [1,2] gives a testing theory that lies closer to common testing practice (e.g. see the theory described in [3]). In this talk the testing theory of [3] is refined even further by not only distinguishing between inputs and outputs, but also by distinguishing between the locations (or: PCO, Points of Control and Observation) where these actions can occur. In that way the distribution of the interface of implementations in explicitly taken into account. By weakening the assumption on the enabling of input actions and

applying ideas from [2,3] a testing theory is obtained that is parameterized by the PCOs of implementations. For specific instantiations of the PCOs this theory collapses to the theory of [2] or [3]. A practically interesting class of correctness criteria has been defined that is also parameterized by the PCOs. Furthermore, a test generation algorithm that is parameterized by the PCOs is presented. This algorithm can generate sound tests from transition system specifications: any correct implementation will never fail any test that can be generated. The set of all test cases is complete: for any incorrect implementation a test can generated that will detect it.

[1] R.De Nicola and M. Hennesy. Testing equivalences for processes, TCS, 34:83-133,1987.

[2] I. Phillips, Refusal Testing. TCS 50(2):241-284,1987.

[3] J. Tretmans, Test generation with inputs, outputs, and repetitive quiescence. Software – Concepts and Tool, 17:103-120,1996

Testing Partially Defined, Nondeterministic and Embedded FSMs

Alexandre Petrenko

We consider the problem of test derivation from a specification modeled by an input/output finite state machine (FSM). The development of test generation methods based on this model has initially been driven by problems arising from functional testing of sequential circuits, see e.g. work of Hennie who pioneered the so-called state identification approach. This approach has yielded many test derivation methods, all of them require the FSM be minimal, deterministic, and completely defined. We demonstrate that these methods fail when applied to machines which do not fit to the above category and discuss our recent research that is concentrated on more complex models such as partially defined and nondeterministic machines. A specific feature of a partially specified FSM is that it has undefined or "don't care" transitions. There are a number of conventions that may be used to give a formal meaning to "undefined". For partial deterministic FSMs, the so-called quasi-equivalence relation replaces equivalence used a conformance relation for complete machines. Test derivation from a partial FSM is impeded by the fact that the FSM at hand may not have a unique minimal form, opposed to

any completely specified FSM, as a result, not all states are distinguishable. A new approach for deriving complete test suites from partial machines has been elaborated and later generalized to cover nondeterministic machines. A so-called "state-counting" approach includes the traditional state identification approach developed by Hennie as a special case and treats more realistic specifications. We also demonstrate that this approach can easily be adapted to solve the problem of test derivation for FSMs embedded into a given context machine.

A First Step for Integration Testing of Distributed Systems

Pascale Le Gall

Traditionally, we all learned that the integration testing of large systems must proceed in incremental steps. For modular systems with hierarchical structure, top-down vs. bottom-up are two well-known alternative strategies and in practice, mixed strategies are used according to the cost of stubs and drivers (that simulate resp. called and calling modules).

Classical approaches for integration testing of large systems no longer fit the needs for testing and analyzing modern complex systems, that are made of many components (each, likely, a complex system itself), variously distributed and interacting concurrently on networks which do not follow classical hierarchical structure. New strategies for arbitrary software architecture need to be identified.

Integration test strategies should allow testers to decompose a dynamic, heterogeneous system not only on the basis of the static structure, as earlier cited approaches did, but also considering the dynamic behaviour, as well as the specific test objectives. Indeed, test objectives vary as the system is looked at from different viewpoints for the analysis of the system architecture. Each view captures a specific problem domain, and requires a specific decomposition and specific test cases.

We propose an approach to distributed system decomposition for analysis and testing purposes. We introduce Information Space which consists of an architecture (the system topology) and of the flow of information between the components of the architecture. For each relevant information piece on the information space, we derive a sensible architectural decomposition into

three categories: the components from which the test inputs are launched; the components that process those inputs (the subsystem under test); and the components acting as test oracles (allowing interpretation of test outputs).

Such an approach allows to identify the sensible information pieces according to a given test objective and the notions of "launchers" and "oracle" components may advantageously replace the classical notions of stubs and drivers. It provides a great flexibility for the choice of test objectives according to different viewpoints. In practice, it is meant to be applied in the early stages of system development, over a high level description of the architecture, to identify a suitable test integration plan.

Test Data Selection for Reactive Synchronous Software

Bruno Marre

We present a test data selection tool, GATeL, from Lustre descriptions. Lustre is a declarative specification/programming language for reactive synchronous systems. It is used for control command systems in french nuclear power plants. Despite the fact that there exists verification tools for Lustre, there is a strong demand of testing techniques devoted to Lustre. The LRI has proposed a theory and a tool, LOFT, for test data selection from algebraic specifications. They have been applied to Lustre descriptions using a translation of Lustre into algebraic specifications. Experimentations on industrial case studies have shown the interest of the approach, but also some limitations on its scalability. These limitations came essentially from the selection tool which was not originally designed for Lustre descriptions. This led us to implement a selection tool taking into account the specificities of the Lustre lanmguage (data flow computations, temporal operators ...) This tool can be used for unit testing and integration testing. Future improvements will make it usable for the test of safety properties.

Testing of Real-Time and Performance Requirements

Ina Schieferdecker

This talk discusses the need of an integrated test method for functional, real-time and performance requirements from the perspective of current and emerging network technologies such as ATM (asynchronous transfer mode) networks and the integrated and differentiated service architecture in the next generation Internet.

Such a test method has to be based on an appropriate notation with a well-defined semantics in order to support the sound description, analysis and implementation of functional, real-time and performance (load/stress) tests with the benefit to make test results comparable.

It is argued that an unifying notation for the description of the three types of tests allows for an incremental test case development process and for the re-use and enhanced maintenance of test cases.

Such a notation can be defined as an extension of TTCN 04 04(the Tree and Tabular Combined Notation, ISO/IEC Standard 9646 Part 3), which is the standardized test notation for the description of OSI (Open Systems Interconnection) conformance tests.

The talk analysis related work on time-extended versions of TTCN with respect to their language features and semantics definition. In particular, Real Time TTCN with means to express hard timing requirements and Performance TTCN with means to express performance test configurations, traffic models, measurements, performance constraints and performance verdicts are considered.

Finally, on the basis of both approaches a combined and extended notation is proposed for the description of functional, real-time and performance tests of telecommunication protocols, services, and applications.

Test Re-use in an OO Setting

Ruurd Kuiper

Short presentation of preliminary ideas on test re-use in an OO setting, exemplified on a Java class.

In CapGemini's ComBAD (Component-based Application Technologies Development) approach, from Generic Frameworks applications are instantiated for clients. The idea is, to complement a framework with a, similarly

generic, set of tests. Such a set parallels the framework's class hierarchy: for each class there be a tester-class. This tester-class is build from the methods that are present in the class under test, together with extra tester-methods that may target the behaviour of a single method, but also joint behaviour of several methods (say, a feature) or interactions of these (say, feature interaction). The latter two methods should be build using the single method tester-methods. All these tester-methods are collected together in a method testscript, the single entrypoint for all testing.

Instantiations of a class should now be paralleled by instantiations of the tester-class: in part this will be automatic, like overridden methods in the class also being used in the tester-class, in part adaptations need to be made, for example to tester-methods that apply to overridden methods. The idea is, that over all reasonable instantiations, the structure of the test set ensures that adaptation can be limited to tester-methods that target single methods, thus providing the feature and feature interaction tests for free.

In the case of Java, judicious choice of the organization of the testscript ensures that instantiation decisions are reflected in the API, indicating which tests should be adapted.

Systematic Derivation of Fault-Sensitive Test Cases

Monika Muellerburg

The method of *systematic testing* is introduced and its application in the validation of reactive systems is illustrated. In systematic testing, test (data selection) criteria (and respective coverage and testability measures) are used for assessing the effectiveness of the selected input set. A classification scheme – relating entities, test criteria, and fault types – supports the choice of a set of criteria that is appropriate for the considered entity: the specification for black box test or the program source code for glass box test.

The second part briefly explains the language (and respective tool environment), SYNCHRONOUSEIFEL (SE). Developing a reactive system in SE means defining *reactive objects* that communicate via signals on a bus with a special signal, a clock tick. Reactive objects read their input signals, compute their reaction, and finally emit their output signals between two clock ticks. A *reactive class* contains the description of reactive behaviour in addition to the description of operations and attributes as known from object-oriented

construction. It is shown how reactive objects may be used for testing: for checking system properties (*oracle by observers*), for checking assumptions about signals from the environment, and for generating inputs simulating the environment.

The support for using observers, comes with the synchronous model. From the viewpoint of systematic testing, however, this is not sufficient. We therefore experiment with an *additional behavioural test specification*, an extended finite-state-machine (EFSM), using it for deriving fault-sensitive test cases and as an oracle. The EFSM consists of an FSM part (representing control) and a data part. Events may be grouped into data structures; labels attached to transitions may be refined in *activation* and *response refinements*. Thus, the control part is made easier by transferring complexity from control into some data part.

Testing Techniques for Synchronous Software

Farid Ouabdesselam

Several studies have shown that automated testing is a promising approach to save significant amounts of time and money in the industry of reactive software. But automated testing requires a formal framework and adequate means to generate test data.

In the context of synchronous reactive software, we have built such a framework and its associated tool -Lutess- to integrate various well-founded testing techniques. This tool automatically constructs test harnesses for fully automated test data generation and verdict return. The generation conforms to different formal descriptions: software environment constraints, functional and safety-oriented properties to be satisfied by the software, software operational profiles and software behavior patterns. These descriptions are expressed in an extended executable temporal logic. They correspond to more and more complex test objectives raised by the first pre-industrial applications of Lutess.

This talk concentrates on the latest development of the tool and its use in the validation of standard feature specifications in telephone systems. The four testing techniques which are coordinated in Lutess uniform framework, are shown to be well-suited to efficient software testing and specification debugging. The lessons learnt from the use of Lutess in the context of industrial partnerships are discussed.

On Coverage Measures for Partial Validation

Ed Brinksma
http://wwwtios.cs.utwente.nl/~brinksma/

The validation of implementations is an essential part of the design of both hardware and software systems in order to establish the correctness of such systems. As such it has been an important application area for all kinds of formal methods to support this activity. Many of such methods, however, aim at a complete proof of correctness, which become unmanageable in the case of larger, realistic designs. In practice, therefore, attention is limited to such methods that can be applied partially or in an approximative manner, such as by testing. Albeit more pragmatic, these approaches usually lack a good measure for the extent to which correctness is established. Such *coverage* measures are needed to compare and assess different strategies for partial validation in the context of a given specification. In this presentation we propose to follow a measure-theoretic approach in which an exogenous *cost* function (quantifying the effect of certain properties in an implementation) is integrated over a measure that is induced by the probability of error occurrences in implementations. In this way, in fact, we do not only obtain a notion of coverage, but a general way of assigning measures to specification theories in the context of a given class of implementation structures.

Towards Automatic Distribution of Testers for Conformance Testing

Claude Jard

This talk presented a first step towards automatic generation of distributed tests for testing distributed systems. This question poses two main problems. The first problem is now well-known and concerns the taking into account of the asynchronous nature of the interaction between the testers and the implementation under testing. The second one is the automatic synthesis of the coordination protocol between the different local testers. We first define a characterization of the tests for which the property of unbias (conformant implementations cannot be rejected) is preserved by the existence of an asynchronous environment. Then, starting from a centralized test case, we propose a method to derive automatically a set of local communicating testers using fifo queues. The method inserts synchronization messages to

implement sequentiality and proposes to use a distributed consensus service to solve the non-local choices. We prove that the generated distributed test case is not biased, it tests the same behaviors of the implementation and has the same testing power as the centralized test case. Open questions remain still open to optimize the synchronization between the testers by using some additional causal information deduced from a formal specification or observed during the testing.

Statistical Testing Based on Structural and Functional Criteria

Pascale Thevenod-Fosse

Statistical testing is based on a probabilistic generation of test patterns: structural or functional criteria serve as guides for defining an input profile and a test size. The method is intended to compensate for the imperfect connection of criteria with software faults, and should not be confused with random testing, a blind approach that uses a uniform profile over the input domain. After a brief description of the approach, the talk focused on experimental results involving safety-critical software from various application domains: avionics, civil and military nuclear field. In most of the experiments, mutation analysis was used to assess the error detection power of various test patterns. First, we gave results related to procedural programs: they show the best effectiveness of statistical testing in comparison to deterministic and random testing. Yet, a limitation of the statistical patterns experimented on was their lack of adequacy with respect to faults related to extremal/special cases, thus justifying the use of a mixed test strategy involving both statistical and deterministic test sets. Such a mixed strategy has been defined for synchronous data flow programs. It may be applied at either the unit or integration testing levels and has been experimented with Lustre programs.

Exploiting Symmetry in Protocol Testing

Judi Romijn

Test generation and execution are often hampered by the large state spaces of the systems involved. In automata (or transition system) based

test algorithms, taking advantage of symmetry in the behavior of specification and implementation may substantially reduce the amount of tests. We present a framework for describing and exploiting symmetries in black box test derivation methods based on finite state machines (FSMs). An algorithm is presented that, for a given symmetry relation on the traces of an FSM, computes a sub-automaton that characterizes the FSM up to symmetry. This machinery is applied to Chow's classical W-method for test derivation. Finally, we focus on symmetries defined in terms of repeating patterns.

Using static analysis to improve test generation

Jean-Claude Fernandez

Distributed systems and more particularly telecommunication systems are more and more complex. Thus formal specifications of such systems become very hard to manage with automatic test generation tools and model checkers. The model based approach for test generation suffers from the state explosion problem. To cope with it, a promising way is to exploit statically available informations about the specifications before to reason on a low level model. This is justified by the fact that the control size part of a protocol is generally very smaller than the data size part.

In this talk, I show how static analysis improves the automatic generation process. Compiler for Formal Description Techniques (such a Lotos, Sdl) built a simulator from a formal specification. We introduce in the compiler chain an intermediate form (extended communicating automate) on which are performed static analysis. Some of these analysis are implemented in a tool connected with the verification tool Aldebaran and the test generator Tgv. We have applied these techniques on a Sdl description of the Sscop protocol:

- when we tried to generate the low level model, the generation process was drastically limited to about 50 000 states.
- after automatic analysis of live variables, the model of the Sdl specification was divided by more than 200.