Relating Test Purposes to Formal Specifications:
Towards a Theoretical Foundation of Practical Testing

Jens Grabowski, Dieter Hogrefe
Robert Nahm, Andreas Spichiger

IAM-93-014

June 1993

Abstract[1]

The problems of current theoretical foundations of testing are its constraint to Finite State Machines (FSMs) and its inability to be related to real black box testing. In this paper we give a theoretical foundation of practical testing. This foundation also implies a test methodology. A test generation tool which is based on this methodology will be presented at the end.

CR Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: General; C.2.2 [Computer-Communication Networks]: Network Protocols; D.2.5 [Software Engineering:] Testing and Debugging

General Terms: Verification, Theory, Standardization

Additional Key Words: Test Generation

[1]This work is performed within the F & E project, no. 233, *'Conformance Testing — A Tool for the Generation of Test Cases'*, funded by Swiss PTT.

1 Introduction

In the project *'Conformance Testing — A Tool for the Generation of Test Cases'* we are looking for methods to generate test cases for conformance tests. This part of the paper is the result of an analysis of the current theoretical approaches to testing, the approach taken in 'Conformance Testing Methodology and Framework' [ISO92a] and the testing methods applied in industry.

1.1 Drawbacks of current theoretical test methods

Quite a number of test methods (some are discussed in [Hol91]) have been introduced in the last twenty years. In general they have the aim to prove behavioural equivalence of an Implementation Under Test (IUT) and its specification. In conformance testing only black box testing is considered. Current methods make a lot of assumptions on the IUT and its specification. Some of these are:

- The IUT should behave like a deterministic and complete[2] Finite State Machine (FSM).
- The FSM represented by the IUT has to be strongly connected[3].
- The maximum number of states of the FSM has to be known and finite.
- The IUT has a known and finite alphabet of input signals.
- The IUT responds to inputs in a known finite time.

These assumptions on the IUT contradict the supposition of black box testing since they make statements concerning the internal structure of the IUT (e.g. number of states). Since a restricted IUT can not be behavioural equivalent to a more powerful specification, it is obvious that the restrictions must hold for the specification too. The implications of this are described in the following.

1.2 Reality and its model

A specification describes the allowed behaviour of a system. The behaviour can be regarded under the following aspects:

1. flow of control of processes,
2. flow of signals between processes,
3. data flow within processes,
4. time and
5. probability.

[2] complete means that in each state all signals may be received

[3] strongly connected means that each state has to be reachable from all other states

If the IUT has to behave like an FSM there are only limited possibilities to describe the above mentioned aspects:

1. The IUT can not have unrestricted procedure calls and the dynamic process creation has to be limited.
2. The buffering of signals has to be limited.
3. The range of variables has to be finite.
4. It is hard to describe time aspects with FSM.
5. It is not possible to describe probability aspects with FSM.

Todays formal description techniques (FDTs), i.e. SDL, Lotos and Estelle, are able to describe Turing Machines which are much more powerful than FSMs. It is possible to describe the control flow of processes, the flow of signals between processes and the data flow within processes without restrictions. The mentioned FDTs are not made to describe time and probability and, therefore, they have problems to describe such aspects.

2 Reasoning about the relation of test purposes and the formal specification

The goal of practical testing is to check certain properties of systems (which has been specified using an FDT) and not to prove behavioural equivalence between the specification and an IUT. The properties to check are defined by so-called test purposes. A test purpose may also describe issues of control flow, signal flow, data flow, time and probability. The selection of such test purposes is a intuitive process and is usually made by a human. Our approach is very much oriented towards this practical method. The only assumptions that are made on the IUT considered as a black box are:

- The IUT has to be representable as a Labelled Transition System (LTS).
- The IUT has a known alphabet of input signals.

This reduces the number of assumptions on an IUT very much. Furthermore, they allow to use a standardized FDT, since LOTOS, Estelle and SDL specifications are representable as LTS.

The results of our test method are weaker than the results with current theoretical approaches, but the results of our method are in line with practical testing and [ISO92a]. When an IUT passes a test case (gets a pass verdict) then it has fulfilled the test purposes and did not show a behaviour contradicting the specification.

For the description of the test purposes MSCs [CCI92, GGR93] are used. In their simplest form they are able to describe the flow of signals very well. When

they are extended with states also test purposes concerning the flow of control can be described [GR89]. The introduction of parameters and variables in MSCs will make it possible to describe data flow. For the specification of the IUT SDL [CCI88] is chosen.

The next section will present the SAMSTAG (Sdl And Msc baSed Test cAse Generation) method. The SAMSTAG method interprets SDL as LTS and test purposes are described as FSMs, i.e. an MSC is transformed into an FSM. It is important to note that the SAMSTAG method is applicable to any test purpose which is representable as an FSM and for any system specification which can be represented by an LTS.

3 The SAMSTAG method

The goal of the SAMSTAG (Sdl And Msc baSed Test cAse Generation) method is to generate a TTCN test case from an SDL specification and an MSC.

The SDL specification describes the test architecture, i.e. the IUT, the test context and the tester processes. The tester processes, e.g. in the context of [ISO92a] upper and lower testers, are modelled as processes, which can send and receive every valid signal at any time. The MSC describes the test purpose, i.e. part of the signal exchange, which has to be performed in order to get a pass verdict.

The resulting test case can be seen as a tree, where the nodes are input and output events of the tester. Figure 1 shows the dynamic part of a test case for the Inres protocol [Hog91], in which it is tested if the Initiator can establish a connection after a third CR (Connection Request). Every path of the tree, from the root to a leaf node is associated with a test verdict. We call these paths *observables*. According to the three possible test verdicts of a test case we distinguish between *pass*, *fail* and *inconclusive observables*.

Pass observables. A *pass observable* is an observable of the SDL system from which we can conclude, that the the test purpose is fulfilled, i.e. the signal exchange of the MSC is performed. Additionally, a pass observable leads the system from its initial state to its initial state, such that the next test case can be applied.

Inconclusive observables. An *inconclusive observable* is an observable, from which we cannot conclude, that the test purpose is fulfilled, resp. the signal exchange of the MSC is performed, although it is a valid observable of the SDL system. Within the SAMSTAG method we do not require, that an inconclusive observable leads the SDL system back to the initial state. Therefore, we generate the shortest possible inconclusive observables.

Fail observables. A *fail observable* is an observable such that there exists no corresponding behaviour of the SDL system.

Test Case Dynamic Behaviour					
Test Case Name : Test_Case_2 Group : Inres_Protocol/Initiator_Test/Connection_Establishment Purpose : Connection Establishment after the third retransmission of a Connection Request Default : Unexpected Events Comments :					
Nr.	Label	Behaviour Desription	Constraint Ref.	Verdict	Comments
1		UT!ICONreq			
2		LT?MDATind(CR)			
3		LT?MDATind(CR)			
4		LT?MDATind(CR)			
5		LT!MDATreq(CC)			
6		UT?ICONconf			
7		UT!IDATreq			
8		LT?MDATind(DT)			
9		LT?MDATind(DT)			
10		LT?MDATind(DT)			
11		LT?MDATind(DT)			
12		UT?IDISind		PASS	
13		LT?MDATind(CR)		INCONC	
14		LT?MDATind(CR)		INCONC	
Detailed Comments :					

Default Dynamic Behaviour					
Test Step Name : Unexpected Events Group : Inres_Protocol/Initiator_Test/Connection_Establishment Objective : Handle unexpected Signals Comments :					
Nr.	Label	Behaviour Desription	Constraint Ref.	Verdict	Comments
1		UT?OTHERWISE		FAIL	
2		LT?OTHERWISE		FAIL	
Detailed Comments :					

Figure 1: TTCN test case

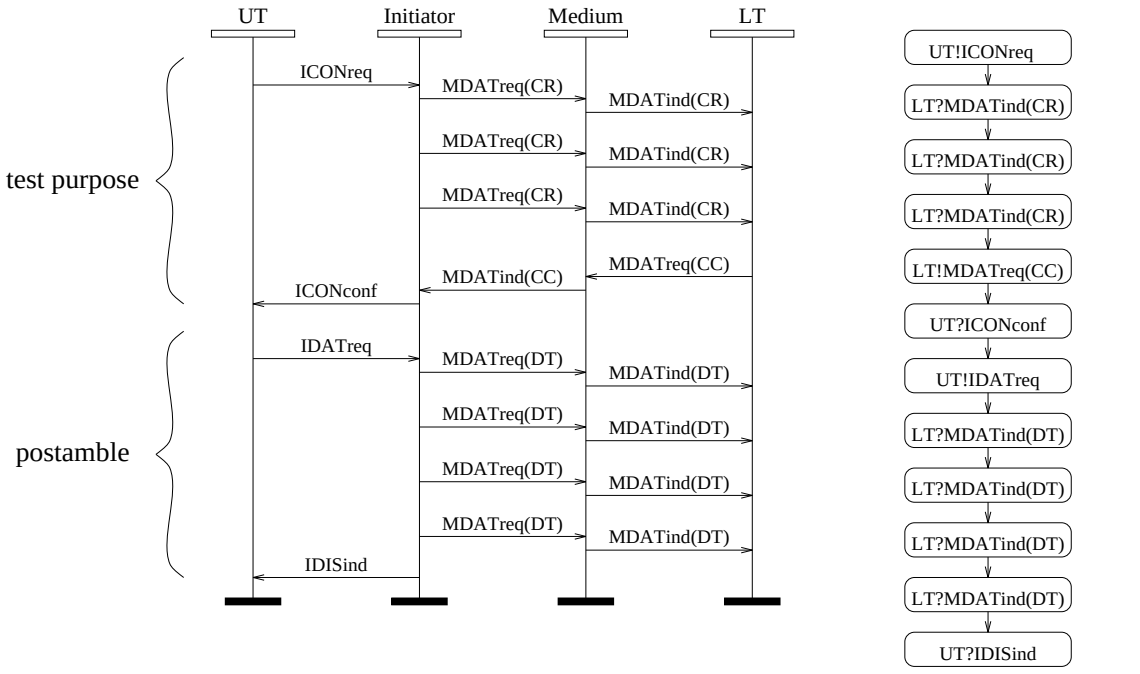
Unique Input Output approach

Unfortunately there is no unique relation between a trace (a sequence of events like inputs, outputs, tasks, ...) of an SDL system and its observable, i.e. two different traces may have the same observable. One problem of defining test purposes is, that they may describe an internal signal exchange or that an internal state is reached, e.g. the initial state at the end of a test case. Such requirements are not directly observable by the tester.

But they are indirectly testable by means of special observables, which can guarantee that every corresponding trace of the SDL system starts and ends in the initial

state and fulfills the test purpose. We call such observables *unique pass observables*. In order to minimize the costs of the test campaign we select the shortest unique pass observables as the pass observable for the test case.

For calculating the unique pass observables we generate first all observables of the traces of the SDL system, which start and end in the initial state and fulfill the test purpose. We call them *possible pass observables*. From the possible pass observables the unique pass observables are selected.



(a) Trace of the test architecture (b) unique pass observable

Figure 2: Trace of a system and its observable

Figure 2(a) presents a trace by means of an MSC and the corresponding observable by means of a sequence of inputs and outputs. The MSC presents the trace which has to be performed to get a pass verdict. Normally such a trace consists of a preamble, a test body, which performs the test purpose and a postamble. In our example the preamble is empty and the test body can be identified with the test purpose. The corresponding observable in Figure 2(b) is a unique pass observable. This means, that every corresponding trace of that observable leads a correct implemented IUT back to the initial state and performs the test purpose. The unique pass observable can be found in the test case in Figure 1 from line 1 to 12. It is left to the reader to prove, that the observable, which is obtained by taking a normal disconnection phase as postamble is not unique.

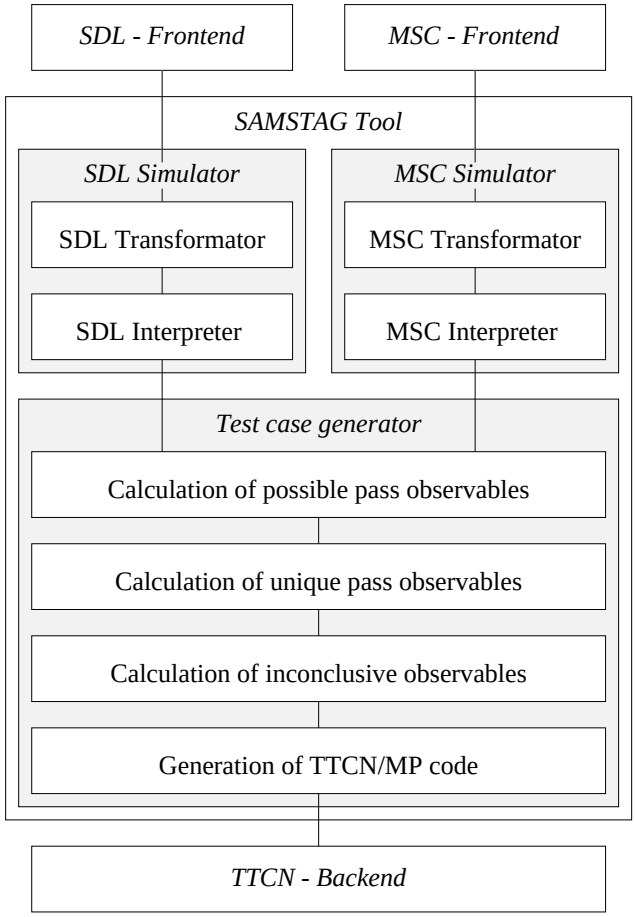


Figure 3: The architecture of the SAMSTAG tool

4 The implementation of the SAMSTAG method

Within the previous chapter our approach is presented on a more intuitive level. As a summary one can say that the approach is based on the calculation of four sets of observables: *possible pass*, *unique pass*, *inconclusive* and *fail observables*. In this chapter we explain how the observables can be calculated and how this is reflected in a tool architecture.

4.1 The computation of the observables of a test case

The observables of a test case are generated in four steps. In a first step the *possible pass observables* of the test case are computed. In a second step we check for each *possible pass observable* whether it is a *unique pass observable* or not. Since we only need one *unique pass observable* to ensure the MSC test purpose, we select one of the shortest. For the chosen *unique pass observable* the corresponding *inconclusive observables* are generated within a third and the *fail observables* are defined within a fourth step.

The computation of possible pass observables. The computation of *possible pass observables* is a typical search problem. We have to find SDL traces which include the events specified by the MSC and which lead the SDL system from its initial state back to its initial state. From such traces the *possible pass observables* are extracted. Unfortunately, we cannot ensure that we find *possible pass observables*, because this problem is equivalent to the reachability problem of turing machines [BZ83] which is not decidable.

We search the required observables by simulating the SDL description and the MSC in parallel. There exist several search methods like depth and breadth search. Breadth search can not applied because it is impossible to store all possible states of the SDL system[4]. Depth search also is not usable since we can not guarantee termination. As a consequence we use a k-bounded depth search which evaluates all possible traces of length k. If no trace with required properties is found, the search can be repeated with a higher bound or stopped without results.

The computation of unique pass observables. For each *possible pass observable* we have to check if it is a *unique pass observable*. In this analysis we simulate all traces with the possible pass observable. If all the simulated traces perform the test purpose and reach the initial state, then the possible pass observable is unique. In general there can exist none or a whole set of *unique pass observables* for an MSC. For proving a test purpose defined by an MSC we only need one. We choose one of the shortest *unique pass observables* to be the *unique pass observable* of the generated test case.

The computation of inconclusive observables. For the chosen *unique pass observable* the corresponding *inconclusive observables* have to be generated. Therefore, the SDL description is simulated according to the *pass observable*. The *inconclusive observables* are ending in a response of the SUT from which one can conclude that the required *unique pass observable* is not performed.

Fail observables. *Fail observables* are added by means of the TTCN constructs OTHERWISE and default behaviour. Therefore, they need not to be calculated.

4.2 The test case generation tool

Figure 3 presents the architecture of a tool which is developed at the University of Berne and which implements the presented approach. The tool is structured in the three parts *SDL simulator*, *MSC simulator* and *test case generator*. Both simulators consist of a *transformator* and an *interpreter*. The transformators read descriptions in phrase representation of SDL (SDL/PR) and MSC (MSC/PR) and transform them into internal representations. Afterwards the internal representations are simulated by the interpreters. The test case generator is structured in four modules:

[4] A state of an SDL system includes the control states of the processes, the contents of the queues and the values of the variables.

- Calculation of *possible Pass observables.*
- Calculation of *unique Pass observables.*
- Calculation of *Inconclusive observables.*
- Generation of the corresponding TTCN/MP[5] code.

The tool is implemented on Sun workstations. Its inputs are MSC/PR [CCI92] and SDL/PR [CCI88] descriptions, and its output is a TTCN/MP description [ISO92b]. Front- and backends of the tool are commercial SDL, MSC and TTCN editors.

5 Conclusions

A test generation methodology has been presented which is totally in line with current practical testing approaches and [ISO92a]. The implemented tool SAMSTAG is able to generate a complete test case in TTCN from an SDL specification of the IUT and an MSC description of the test purpose. The presented method can easily be adopted to other description techniques and is therefore not restricted to SDL and MSC at all. For further details we refer the reader to [GHN93] or [NGH93].

References

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite State Machines. *Journal of the Association for Computing Machinery*, 30(2):323–342, April 1983.

[CCI88] CCITT SG X. Specification and Description Language. Recommendation Z.100, CCITT, 1988.

[CCI92] CCITT. Message Sequence Charts. Recommendation Z.120, CCITT, May 1992.

[GGR93] Jens Grabowski, Peter Graubmann, and Ekkart Rudolph. The standardisation of Message Sequence Charts. In *Software Engineering Standards Symposium*, 1993.

[GHN93] Jens Grabowski, Dieter Hogrefe, and Robert Nahm. Test Case Generation with Test Purpose Specification by MSCs. In *6th SDL Forum*. North-Holland, October 1993.

[5] TTCN/MP denotes the machine processable form of TTCN. TTCN/MP has a standardized ASCII syntax. During test case generation it is not very efficient to use ASCII files as internal computer representation. As a consequence we have to translate our internal representation into TTCN/MP.

[GR89] Jens Grabowski and Ekkart Rudolph. Putting extended sequence charts into practice. In Ove Faergemand and M.M. Marques, editors, *SDL '89 The Language at Work*. North-Holland, 1989.

[Hog91] Dieter Hogrefe. OSI Formal Specification Case Study: The INRES Protocol and Service. Technical Report IAM-91-012, University of Berne, 1991.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International, Inc., 1991.

[ISO92a] ISO/IEC JTC 1/SC 21 N. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework. International Multipart Standard ISO/IEC 9646, ISO, 1992.

[ISO92b] ISO/IEC JTC 1/SC21 N. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation. International Standard ISO/IEC 9646-3, ISO, 1992.

[NGH93] Robert Nahm, Jens Grabowski, and Dieter Hogrefe. Test Case Generation for Temporal Properties. Technical Report IAM-93-013, University of Berne, 1993. submitted to FORTE'93 Conference.