On the Exploitation of Parallelism in a Test Generation Method for LOTOS-Specifications

Volkmar Pleßer

Technical University of Munich, Chair for Data Processing,
D-80290 München, Germany
vp@ldv.e-technik.tu-muenchen.de

Abstract
A major problem in the field of automatic test derivation from LOTOS and other formal description techniques is that resulting complete tests become too large to be useful in practice. This paper describes how a new automatic test derivation method exploits parallelism in the specification to overcome this problem. The method uses a fault model in order to define possible faults that the test has to investigate. Thus the method avoids the well-known state space explosion problem, and it, therefore, generates short test sequences with high fault coverage. The length of the test sequence grows only linear with the number of subbehaviours in a parallel expression.

1. Introduction

The use of formal description techniques in protocol specifications serves as a basis for deriving tests from the specification automatically. Much of work has been done on this in the recent years. Many methods have been proposed for deriving tests from LOTOS [1] specifications.

In [2] so-called canonical testers and a way to derive test suites from these are defined. Other methods transform the specification into labelled transition systems (LTSs) or finite state machines (FSMs) in order to derive tests from these (e. g. [3]). Often test derivation methods developed earlier for these models are applied after the transformation. A survey over these methods is given in [4]. [5] describes how LTSs can be transformed into FSMs in order to exploit the FSM-based test derivation methods.

Although LTSs serve as a semantic model for LOTOS, the transformation of LOTOS specifications into finite LTSs or FSMs is not possible in every case. The main difficulty is the well-known state space explosion problem. This problem can occur when parallel expressions are transformed into LTSs. LOTOS possesses the operators |||, |[...]| and || to express parallelism. The number of states of a behaviour B:=B1|||B2 is the product of the number of states of B1 and B2. This leads to very large LTSs. If there is any recursion in the parallelism the state space of the LTS can become infinite, even. Due to this problem, the methods, mentioned above, often lead to large or even infinite test. In order to get short test cases that are feasible in practice a test selection has to be done, which can be guided by test coverage metrics [6].

Approaches that exploit the structure of the specification can be found in [7] and [8]. In opposite to us, they aim at a test derivation that incorporates a human test designer. We aim at nearly complete automation.

T. Mizuno et al. (eds.), *Protocol Test Systems*

We have developed a test derivation method that exploits parallel expressions in the specification, in order to get short test sequences. The LOTOS specification is not entirely transformed into one LTS. Instead of this, it is transformed into a system of LTSs. This system consists of LTSs that are linked together with LOTOS parallel operators. Thus we avoid the state explosion problem and we can deal even with behaviours with an infinite state space.

We have developed a fault model for this system of LTSs, that considers parallel expressions especially. In this paper we combine the fault model for interleaving LTSs with the transition tour method (TTmethod) for deterministic FSMs which was introduced in [9]. A combination with other known test derivation methods for FSMs would be possible also.

The rest of this paper is structured as follows. Section 2 introduces a fault model for LOTOS parallel expression. In section 3, the accompanying test sequence is described. In section 4 a simple example demonstrates the benefit of this method. Section 5 considers non-determinism and, finally, section 6 gives a conclusion.

2. A fault model for LOTOS

Testing of infinite behaviours with full fault coverage is generally not possible. This follows from a proposition of [10] that says that a finite state machine is not identifiable unless the entire input alphabet and the maximum number of states in its minimal form are known in advance. In the practice of testing the number of states of an implementation that has to be tested (IUT = implementation under test) is never known. One way to overcome this problem is to restrict the faults that should be searched for to such that are likely to occur. This restriction is done by means of a fault model. Fault models were introduced in [11]. In the formal definition of the notion of a fault model we follow [5]:

Definition 1: Fault models

A fault model fm for the domain $\mathfrak{I}$ is a mapping fm: $\mathfrak{I} \rightarrow P(\mathfrak{I})$, $P(\mathfrak{I})$ is the powerdomain of $\mathfrak{I}$. It defines how the given specification can be transformed into an implementation to represent a fault that fits to the given fault model.

The set fm(S) contains conforming and non-conforming implementations. The tester should be able to decide for all implementations $I \in$ fm(S) whether they conform to the specification S or not. If the tester is able to do this, it has full fault coverage with respect to the fault model. If any implementation I is not an element of fm(S), the tester may detect that the implementation I does not conform to the specification, but it does not need to.

A fault model for LOTOS

Our fault model rests on the assumption, that the implementor gets some support on the implementation of processes in the specification. He does not need to make an error-prone transformation from parallel structures of LOTOS processes into one single state machine. Under this assumption the following fault model leads to short tests with high fault coverage. This is achieved by exploiting parallelism in the specification. The fault model does not deal with the LOTOS specification directly. Instead of this, it deals with a system of LTSs derived from the specification. This offers two advantages:

Firstly, the fault model does not need to consider every LOTOS operator separately. Instead of this, it can focus on operators that normally cause problems, namely parallelism.

Secondly, the fault model can be used for other formal description techniques that expresses parallelism as well. For this, only a transformation from the formal description technique to our system of LTS has to be derived.

Definition 2: Labelled transition systems (LTS)

A labelled transition system is a 4-tupel $<S,Act,T,s_0>$, where S is a non-empty set of states, Act is a set of observable actions, $T \subseteq S x (Act \cup \{i\}) x S$ is the transition relation, i is an internal, unobservable action, and $s_0 \in S$ is the initial state. A LTS is finite, if and only if (iff) the sets S and Act are finite.

Behaviour expressions that do not contain any parallelism are transformed into LTSs using known methods (see e. g. [1, 3, 12]). For these expressions the fault models of known test generation methods can be used. We will adapt the TT-method to LTS in our example in the next section, i. e. these parts are tested with respect to output errors. It would be possible also to use other methods in order to detect output errors and transition errors or to use methods that can deal with non-determinism.

Definition 3: The fault model for parallel expressions

The fault model for any parallel expression B1 |[...]| B2 |[...]| ... |[...]| Bn that is not expanded takes into consideration the following fault types:

E1: a fault in any of the subbehaviours B1, B2, ... Bn,

E2: a complete inactivity of at least one of the subbehaviours B1, B2, ... Bn in at least one state of another subbehaviour B1, B2, ..., or Bn,

E3: faulty synchronization of two subbehaviours.

This fault model can be used for every kind of parallel expression of LOTOS because interleaving (|||) and full synchronization (||) are special cases of the general parallel expression ([...]).

We want to consider the special case of only two subbehaviours B1 and B2 in the parallel expression in order to explain the details of this fault model. The fault model for a parallel expression B1|||B2 or B1|[...]|B2 or B1||B2 encompasses the following fault types:

E1 leads to faults in the behaviours B1 (fault type F1) and B2 (F2). E2 considers a complete inactivity of B1 in at least one state of B2 (F3) and vice versa (F4). E3 leads to a faulty (additional or missing) synchronization of B1 and B2 (F5).

In F1 and F2 the structures of B1 and B2, respectively, determine the faults to be considered. If a subbehaviour B1 or B2 is represented by a LTS without parallelism a known test generation method for FSMs adapted to LTSs would be used to test this subbehaviour. Otherwise the subbehaviour will include further parallelism. In this case the fault model for parallel expressions can be used for the parallel expressions in the subbehaviour, too.

F3 and F4 describe faults where one behaviour becomes completely inactive while the other behaviour is in a certain state. Faults like these can occur due to faulty scheduling of parallel processes. Completely inactive means that the behaviour totally blocks. It is not able to make any transition.

Fault type F5 deals with the synchronization of B1 and B2. For an action that is not in the synchronization list, a faulty synchronization means an additional synchronization in the implementation, that leads to an deadlock or an undesired state change of both behaviours. In the case of an synchronized action, a faulty absence of the synchronization has to be detected.

3. The derived test sequence

The test derivation method rests on the fault model discussed in section 2. Our goal is to short tests with high fault coverage (i. e. full coverage w.r.t. our fault model). As we want to get short tests we don't subdivide the tests into test cases as demanded in [13]. This would rob us of the opportunity to combine test events effectively in order to shorten the test.

Definition 4: Test sequences and test verdicts

A test sequence $\sigma_t \in Act^*$ is a sequence of actions. During the test the implementation under test (IUT) has to perform this sequence of actions. If it does, the tester has to come to an OK-verdict. If it does not, the tester has to pronounce a FAIL-verdict.

Definition 5: Test sequences according to the Transition-Tour-Method [9]

The sequence σ_t=<a_1, a_2,..., a_n>, where a_1, a_2,..., $a_n \in$ Act, is a test sequence of the deterministic LTS <S,Act,T,s_0> according to the TT-method, iff the transition tour encompasses all transitions of the LTS:

$$\exists s_1, s_2, \ldots, s_n \in S: \bigcup_{1 \leq i \leq n} \{\langle s_{i-1}, a_i, s_i \rangle\} = T$$

In the case of deterministic LTS, there exists exactly one sequence of states <s_0, s_1,... s_n>$\in S^n$ that fulfils the condition. A non-deterministic LTS may refuse a transition tour. If we want to derive tests for non-deterministic LTS, we can use other methods that incorporate an appropriate conformance relation.

We focus now on the example of a system of two LTSs A_1 and A_2 that are combined with an interleaving operator. Such a behaviour will be given in section 4.

The first two points of the fault model F1 and F2 consider faults of the sub-behaviours, B1 and B2. Faults in these behaviours have to be detected by means of known test derivation methods for LTSs or FSMs. Here we want to fit the well-known transition tour method [9] to the LTSs in our system of LTS. That means, that a test sequence for a FSM must encompass all transitions of the LTSs in the system.

We want to derive the test sequence for the whole behaviour of the system of interleaving LTSs A_1 and A_2. Therefore we have to define it in terms of a LTS A of the whole system. The states of the LTS A are denoted as usual with pairs of the state denotations ($s_{i,1}$, $s_{i,2}$), where $s_{i,1}$ is a state denotation of A_1 and $s_{i,2}$ is a state denotation of A_2.

Definition 6: Test sequences according to the fault model

The sequence of actions σ_t=<a_1, a_2,..., a_n> is a test sequence of a LTS $A = \langle S, Act_1 \cup Act_2, T, (s_{0,1}, s_{0,2}) \rangle$ that represents the interleaving of the LTSs $A_1 = \langle S_1, Act_1, T_1, s_{0,1} \rangle$ and $A_2 = \langle S_2, Act_2, T_2, s_{0,2} \rangle$ according to the fault model defined in section 2 combined with the transition tour method, iff there exists a sequence of states $\langle (s_{1,1}, s_{1,2}), (s_{2,1}, s_{2,2}), \ldots, (s_{n,1}, s_{n,2}) \rangle \in S^n$ for that the following 5 conditions hold:

Condition C1: All transitions that have to be executed during the test must be transitions of A and the test has to start with the initial state $(s_{0,1}, s_{0,2})$:

$$\forall (i, 1 \le i \le n) : \langle (s_{i-1,1}, s_{i-1,2}), a_i, (s_{i,1}, s_{i,2}) \rangle \in T.$$

Condition C2: The test has to encompass all transitions of A_1:

$$\bigcup_{1 \le i \le n \wedge (s_{i-1,1} \ne s_{i,1} \vee \langle s_{i-1,2}, a_i, s_{i,2} \rangle \notin T_2)} \{\langle s_{i-1,1}, a_i, s_{i,1} \rangle\} = T_1$$

Condition C3: Same as C2 with all transitions of A_2:

Condition C4: In every state of A_2 at least one transition of A_1 has to be executed during the test:

$$\bigcup_{1 \le i \le n \wedge (s_{i-1,1} \ne s_{i,1} \vee \langle s_{i-1,2}, a_i, s_{i,2} \rangle \notin T_2)} \{s_{i-1,2}\} = S_2.$$

Condition C5: Same as C4 with reversed roles of A_1 and A_2.

Besides the condition C1 that ensure that the test is executable on all conforming implementations, there are the conditions C2 to C5. These conditions are related to the fault model defined in chapter 2. Figure 1 shows this relation.

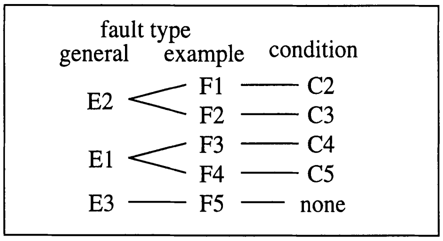

Figure 1: Relation between fault model and test sequence

F5 does not lead to a condition directly, because there is no synchronization between A_1 and A_2 (interleaving) and therefore no synchronization points have to be checked. Since all actions $a \in Act_1 \cup Act_2$ are checked such that only one LTS either A_1 or A_2 can participate, any additional synchronization would be detected surely, because this would cause a blocking of the test sequence.

Test sequence derivation

The conditions, the test sequence must fulfil, are stated above. The test derivation algorithm itself is a search algorithm that finds a sequence that fulfils the conditions. If the goal is to find only one solution for this problem, a depth first search needs less space and time than a breath first search, but the latter finds the shortest sequence fulfilling the conditions. The latter should be preferred, if computer resources are sufficient.

Generally a test sequence becomes short, if the single transitions contribute to the fulfilling of more than one condition. For instance, a single transition can add an element to the sets in the conditions C2 and C4, as well. As a result, it can contribute to the fulfilling of 2 conditions at the same time, in this case the conditions C2 and C4.

4. Example

In this chapter, the test sequence for an example process B is discussed. Figure 2 shows the specification of the process B. Figure 3 illustrates the graphs of the LTSs for B and the locally defined processes B1 and B2. The bold circles denote the initial states.

```
process B[a,b,c,d,e,f,g] :=
    B1 ||| B2
where
    process B1[a,b,e,f] :=
        a;e;b;f;B1
    endproc
    process B2[c,d,e,g] :=
        c;g;d;e;B2
    endproc
endproc
```

Figure 2: Specification of the example

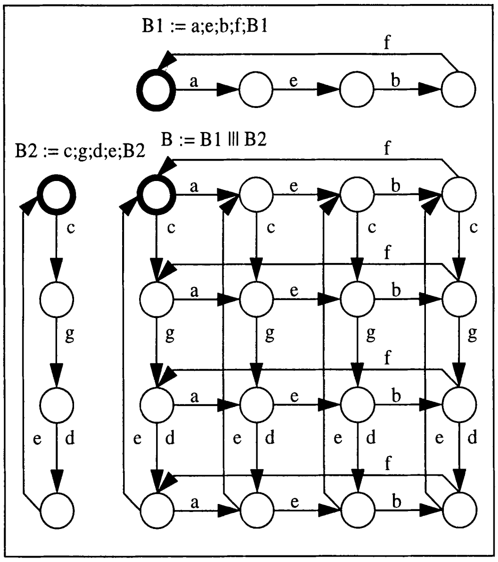

Figure 3: The LTSs of the example

The sequence σ_t=<a, c, e, g, b, d, f, e> is a test sequence for B according to the fault model defined in section 2 and the transition tour method. Exactly one sequence of states exist for σ_t for that the conditions from C1 to C5 hold. The bold arrows in figure 4 show this sequence.

The test sequence contains only 8 steps. The shortest transition tour for B has 32 steps, because the LTS of B has 32 transitions and the transition tour has to visit all. The effort for executing the test, therefore, is reduced by the factor 4 in this example. The test, nevertheless, will discover any fault described in the fault model.

5. Non-Determinism

We want to consider three types of non-determinism, here. Non-determinism in the subbehaviours, non-determinism in the whole behaviour and non-determinism due to the mapping of LOTOS-actions to inputs and outputs of an implementation.

The method in the line-up described here is not able to deal with non-determinism in the subbehaviours of the specification. This follows from the use of the transition tour method for deriv-

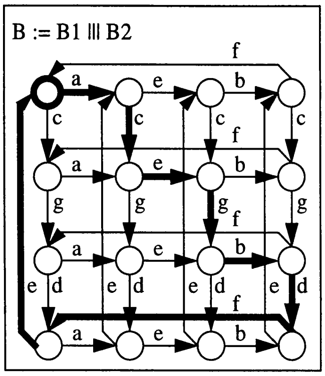

Figure 4: The test sequence

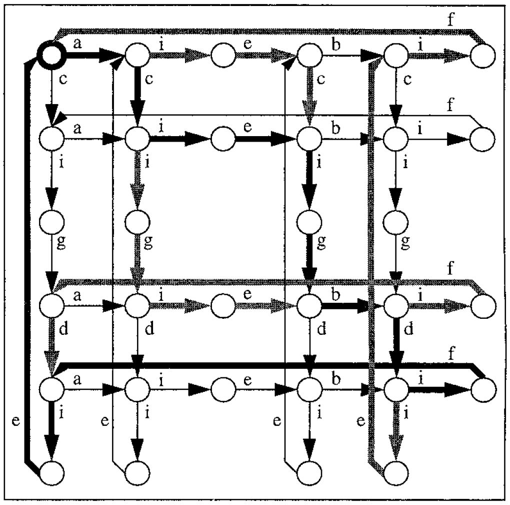

Figure 5: Process B with non-determinism

ing tests for the sub-behaviours, because the transition tour method has not been designed for non-deterministic state machines. If the test derivation has to deal with non-determinism the method presented here has to be combined with a conformance relation, which defines how the specified non-determinism may be mapped into the implementation, and a test derivation method for non-deterministic LTS.

The example of figure 2 contains non-determinism, because B1 and B2 have one action in common. This is no problem, because we are able to avoid the critical state in the test sequence here, but this is not possible in every case.

An example for this is the non-determinism due to the mapping of LOTOS-actions to inputs and outputs. If we assume that a, b, c, d are inputs and e, f, g are outputs of an implementations, we get the behaviour shown in figure 5, because the implementation may make any output instantly after the guarding input has occurred. Therefore the tester has to tolerate the deviations (▨➤) from the planned test sequence (➡).

6. Conclusion

This paper has presented a new method for deriving test sequences from LOTOS specifications. The goal was to get short test sequences that are feasible in practice. The method uses a fault model, which deals with the parallelism in the specification especially. As a result the state-space-explosion due to parallelism is avoided and the goal to get short test sequences is achieved. It can be shown that the test sequence length grows only linear with the number of subbehaviours in a parallel expression. Without the use of a fault model that exploits parallelism the test sequence length would grow exponentially. A simple example was shown, where the length of the test sequence was a quarter of the length of a test sequence generated by the transition tour method.

It has also been shown that a specification that is deterministic in the LOTOS world can contain concealed non-determinism. This non-determinism is revealed when LOTOS-actions are mapped to input and output events in a real world implementation and reasonable assumptions about the scheduling of these events are made. The paper has shown how to deal with this non-determinism.

Data structures have been ignored. One question is how to check the implementation of the abstract data types. For this a fault model has to be developed and incorporated into our test derivation method. The other question is how to consider the influence of the data on the behaviour. Predicates in the specification may inhibit a behaviour part. Sometimes such a predicate can never be satisfied. As a result a test sequence may be infeasible. Although the detection of predicates that can never be satisfied is known as an incomputable problem, in practice, it should be possible to detect these predicates in most cases. In the opinion of the author, it should be possible to find variable settings that fulfil certain predicates in most cases of practical interest automatically.

In our complete test derivation method are some additional conditions concerning non-determinism and resources. These conditions have been left out here, for reasons of conciseness and clarity. The notion of resources does not exist in the LOTOS-world, but we have some heuristics to derive possible resources needed by an implementation. Thus the testsequence examines some critical situations more thorough.

References

1. ISO, "LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", International Standard ISO 8807, Geneve 1989.
2. E. Brinksma, "A theory for the derivation of tests" in PSTV VIII, North Holland, Amsterdam 1988, pp. 63-74.
3. P. Tripathy, B. Sarikaya, "Test Generation from LOTOS Specifications", IEEE Transactions on Computers, Vol. 40, No. 4, April 1991, pp. 543-552.
4. P. Sidhu, T./K. Leung, "Formal Methods for Protocol Testing: A Detailed Study", IEEE Transactions on Software Engineering, Vol. 15, No. 4, April 1989 pp. 413-426.
5. A. Petrenko, G. v. Bochmann, R. Dssouli, "Conformance relations and test derivation" in IWPTS VI, IFIP, 1993, pp. 161-182.
6. S. T. Vuong, J. Alilovic-Curgus, "On Test Coverage Metrics for Communication Protocols" in IWPTS IV, North-Holland, Amsterdam 1992, pp. 31-45.
7. A. Ulrich, H. König, "Test Derivation from LOTOS using Structure Information" in IWPTS VI, 1993, pp. 283-297.
8. R. J. Velthuys, J. M. Schneider, G. Zörntlein, "A test derivation method based on exploiting structure information" in PSTV XII, Florida, June 1992.
9. S. Naito, M. Tsunoyama, "Fault-Detection for Sequential-Machines by Transition-Tours" in: Proceedings of the Eleventh Annual International Symposium on Fault-Tolerant Computing, IEEE, Portland 1981.
10. A. Gill, "Introduction to the Theory of Finite State Machines", McGraw Hill, 1962.
11. G. v. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, G. Luo, "Fault Models in Testing" in IWPTS IV, North Holland, Amsterdam 1992, pp. 17-30.
12. J.-P. Wu, S. T. Chanson, "Translation from LOTOS and Estelle Specifications to Extended Transition System and its Verification" in FORTE '89, Elsevier Science Publishers B. V., North-Holland, Amsterdam 1990, pp. 533-549.
13. ISO, "OSI Conformance Testing Methodology and Framework", International Standard ISO DP 9646, Geneve 1991.