Test Case Generation for Temporal Properties

(This report has also been submitted to the FORTE'93 conference.)

Robert Nahm Jens Grabowski Dieter Hogrefe

Institut für Informatik, Universität Bern
Länggassstr. 51, CH-3012 Bern

IAM-93-013

June 1993

Abstract[1]

The goal of testing is to make statements about the relation between the traces of an implementation and a temporal property. This is not possible for all temporal properties. Within this paper safety and guarantee properties are identified to be testable temporal properties and for these testable properties a test case definition is given. This is done by representing a safety property as a labeled transition system and by representing the guarantee property as a finite automaton. The test case definition is applied to practical testing by using SDL descriptions to specify safety properties and by using Message Sequence Charts (MSCs) to specify guarantee properties.

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

[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

The automatic generation of test cases for communication systems (e.g. OSI, ISDN) can be based on formal specifications written in a standardized formal description technique (FDT, i.e. LOTOS, Estelle, or SDL). For test case generation a formal specification can be simulated and the possible interactions between a system and its environment can be generated automatically. A communication system may have infinite traces since in general it should not terminate. Test cases are finite and therefore, we have to select finite parts of the potentially infinite traces. This can be done by defining reachability criteria, i.e. the simulation of the specification is aborted, if a reachability criterion is fulfilled. For the selected traces test verdicts have to be assigned. One way to do this is to define for every test verdict an own reachability criterion. In theoretical and practical approaches the definition of reachability criteria and the assignment of test verdicts is solved differently.

Theoretical approaches. Approaches coming from research like UIO [11] or the W-method concentrate on the test case generation for finite state machines. The reachability criteria are determined by the used conformance relation (e.g. behavioral equivalence), by fault coverage, or by assumptions about the implementation (e.g. the implementation behaves like a finite automaton with a certain number of states). The assignment of test verdicts is solved very simple. There is only a PASS and an implicit FAIL verdict, i.e. every system behavior which does not get a PASS gets a FAIL. Therefore, only reachability criteria for the PASS verdict have to be defined. One problem of the mentioned approaches is that they cannot be applied to systems with infinite state space. Unfortunately, FDTs force the description of systems with an infinite state space. Infinite signal queues of SDL processes or unlimited data descriptions are two examples for this. However, there can not exist test methods which guarantee behavioral equivalence for systems with an infinite state space. Even finite state machines which communicate by means of unbounded FIFO buffers (i.e. the base model of SDL) are as powerful as Turing Machines [3] for which the behavioral equivalence is undecidable [12]. As a consequence the behavioral equivalence can not be used as conformance relation to derive reachability criteria for generating test cases.

Practical testing. The goal of practical testing is to show certain properties and not to prove a behavioral equivalence between a specification and an implementation. The properties are defined by so-called test purposes. Typically, a test purpose is defined as a sequence of events which have to be performed or a sequence of states which have to be reached. Thus, a test purpose defines directly the reachability criterion for the selection of a test case. In practical testing there exist three test verdicts PASS, FAIL, and INCONCLUSIVE. PASS is given, if the implementation presents a behavior, which is allowed by the specification and which fulfills the test purpose. FAIL is given, if the implementation presents a behavior, which is not allowed by the specification. INCONCLUSIVE is given, if neither a PASS nor a FAIL can be assigned. The disadvantage of practical testing is, that it does not state clearly the relation between the specification and the implementation.

Our approach. Our approach formalizes practical testing. We concentrate on certain temporal properties. Especially safety properties which are described by formal specifi-

cations and guarantee properties which are specified by the test purposes. A guarantee property states that a sequence of events is performed or that a sequence of states is reached. Thus, guarantee properties can be used to estimate a reachability criteria for the selection of finite test cases from infinite traces. As in practical testing our approach also uses the test verdicts PASS, FAIL and INCONCLUSIVE. PASS is given, if the implementation presents a behavior which does not violate the safety property and fulfills the guarantee property. FAIL is given, if the implementation violates the safety property. INCONCLUSIVE is given, if the implementation does not violate the safety property, but does not fulfill the guarantee property. One advantage of our approach is, that it is not bounded to finite state specifications. Moreover, the notion of test purposes is formalized and the assignment of test verdicts can be done automatically.

The rest of the paper is organized in the following way. In section 2 we introduce several temporal properties and classify them by means of the Borel Hierarchy. Furthermore, we identify safety and guarantee properties to be testable temporal properties. In section 3 the representation of temporal properties by automata is explained. Afterwards (section 4) the representation of a test case is introduced and the assignment of test verdicts is described. In section 5 we formalize the test case generation by defining a test case for a given safety and a given guarantee property. In the last section (section 6) a research project is presented which uses the presented approach in order to generate TTCN test cases automatically from SDL specifications and MSCs. The used mathematical notation is explained within the appendix.

2 Testable Properties

Section 2.1 provides a classification of temporal properties. For testing, the classification of temporal properties is of interest[2], since different properties may assume a different test principles, cause other test verdicts, or are not testable. In section 2.2 testing is explained as relating the sets of traces of the implementation and the of temporal property, by observing only one finite trace of the implementation.

2.1 Temporal properties

A Communication systems has its main interest in maintaining an ongoing interaction with its environment, rather then terminating. Theoretically, a communication system can be viewed as a generator of traces. Let E be the set of events which are executed by the communication system. We assume a trace to be a sequence of events and define a property to be a set of traces. We distinguish between finitary properties i.e. sets of finite traces $\Phi \subseteq E^*$ and infinitary properties i.e. sets of infinite traces $\Pi \subseteq E^\omega$. For simplicity we regard only infinitary properties. In the case that the system terminates a trace may always be extended by means of an infinite sequence of dummy events, e.g. clock events. An implementation I is said to have the property Π, if all traces of the implementation $Tr(I) \subseteq E^\omega$ belong to the property Π, i.e. $Tr(I) \subseteq \Pi$. If we are satisfied to restrict ourselves to expressing only safety properties, then the relatively simple finitary properties suffice.

[2]The theory of verification is interested in the classification of temporal properties, since the different properties are associated with different proof principles, e.g. safety properties are proven by computational induction and liveness properties are proven by using a well founded principle.

The only justification for using infinitary properties, which are considerable more complex is for expressing liveness properties. In general liveness properties are not testable, since one has to observe a complete trace to show, that a liveness property is violated. Therefore, we present a finer classification of the non-safety properties by the so-called *Borel Hierarchy* [16]. This classification enables us to distinguish between properties stating, that a certain *good thing*[3] occurs at least once or occurs infinitely many times in a trace. We characterize the different properties of the Borel Hierarchy by the way they are constructed from languages over finite sequences. In the sequel we construct infinitary properties from finitary ones and introduce four basic classes of properties.

- A *safety property* is a property, which states, that a good thing always occurs. We construct a safety property $\mathcal{A}(\Phi)$ from a finitary property Φ, such that an infinite trace belongs to the safety property, iff all its prefixes are in the finitary property.

$$\mathcal{A}(\Phi) = \{\sigma \in E^\omega \mid \forall w \sqsubseteq \sigma : w \in \Phi\}$$

- A *guarantee property* is a property, which states, that a good thing occurs at least once. We construct a guarantee property $\mathcal{E}(\Phi)$ from a finitary property Φ, such that an infinite trace belongs to the guarantee property, iff at least one of its prefixes is in the finitary property.

$$\mathcal{E}(\Phi) = \{\sigma \in E^\omega \mid \exists w \sqsubseteq \sigma : w \in \Phi\}$$

- A *recurrence property* is a property, which states, that a good thing occurs infinitely often. We construct a recurrence property $\mathcal{R}(\Phi)$ from a finitary property Φ, such that an infinite trace belongs to the recurrence property, iff infinitely many prefixes are in the finitary property.

$$\mathcal{R}(\Phi) = \{\sigma \in E^\omega \mid \forall w \in \sigma \exists w' (w \sqsubseteq w' \sqsubseteq \sigma) : w' \in \Phi\}$$

- A *persistence property* is a property, which states, that a good thing occurs continuously from a certain point on. We construct a persistence property $\mathcal{P}(\Phi)$ from a finitary property Φ, such that an infinite trace belongs to the persistence property, iff all but finitely many of its prefixes belong to the finitary property.

$$\mathcal{P}(\Phi) = \{\sigma \in E^\omega \mid \exists w \in \sigma \forall w' (w \sqsubseteq w' \sqsubseteq \sigma) : w' \in \Phi\}$$

The four classes of properties are closed under union and intersection, e.g. the union of two safety properties is again a safety property. Safety and guarantee properties form dual classes, i.e. the complement of a safety property is a guarantee property and vice versa. Also the classes of recurrence and persistence properties are dual to each other. By union and intersection of the four basic classes we get the obligation and reactivity properties.

- An *obligation property* can be obtained by an arbitrary union and intersection of safety and guarantee properties.

- A *reactivity property* can be obtained by an arbitrary union and intersection of recurrence and persistence properties.

The obligation and reactivity properties are closed under union, intersection and complement.

[3] The informal expression *good* and *bad thing* are taken from [1]

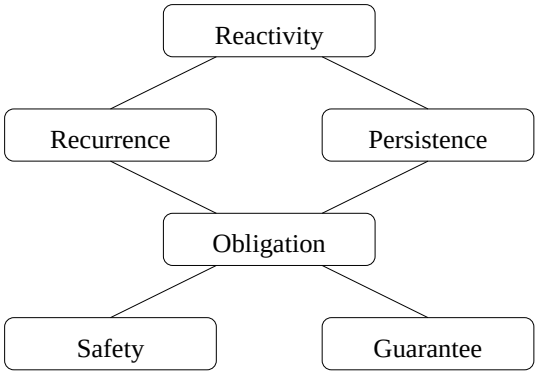


Figure 1: The Borel Hierarchy of temporal properties

Hierarchy. According to Figure 1 the Borel Hierarchy consists of six classes of properties. The class of obligation properties contain the class of safety and guarantee properties. The obligation properties are strictly contained in the recurrence and persistence properties. Per definition the class of reactivity properties strictly contains the class of recurrence and persistence properties. For the proofs we refer the reader to [16]. In the sequel we only regard properties of the four basic classes.

2.2 Testability

In testing we are interested in the relation between the sets of traces, which are executed by the implementation $Tr(I)$ and which belong to the temporal property Π. Thus, we have to make statements about the relation of sets of infinite traces. The best would be, if we can prove that an implementation fulfills a property, resp. $Tr(I) \subseteq \Pi$.

During a test it is only possible to observe a prefix w of an infinite trace of the implementation. Let Φ be the finitary property from which the infinitary property Π is constructed. We can decide, whether $w \in \Phi$ or not. From this we conclude, whether there is an infinite extension σ from w, which belongs to Π or not. Again from this we can make some conclusions about the relation of all traces of the implementation $Tr(I)$ and the temporal property Π. Between $Tr(I)$ and Π there are the following relations possible.

1. $\Pi = Tr(I)$ (the so-called *behavioral equivalence*)
2. $Tr(I) \subseteq \Pi$ (an implementation I fulfills the temporal property Π)
3. $\Pi \subseteq Tr(I)$
4. $\Pi \cap Tr(I) = \emptyset$ (the traces of Π and the implementation I are disjoint)

By checking a relation we can obtain the following results:

- *true*, if a relation is verified.
- *false*, if a relation is rejected.
- *unknown*, if we cannot say anything about a relation.

	Safety	Guarantee	Recurrence	Persistence
Π = Tr(I)	false unknown	unknown	unknown	unknown
Tr(I) $\subseteq$ Π	false unknown	unknown	unknown	unknown
Π $\subseteq$ Tr(I)	unknown	unknown	unknown	unknown
$\Pi \cap$ Tr(I) = { }	unknown	false unknown	unknown	unknown

Figure 2: Test results for temporal properties

We define a property to be a *testable property*, if we can reject or verify some of the above relations. Figure 2 lists the results which can be made for the four basic classes of temporal properties. In the sequel we explain, why only these results are possible.

Safety properties. Let Φ be the finitary property from which the safety property $\Pi = \mathcal{A}(\Phi)$ is constructed. If $w \notin \Phi$, then there cannot be an infinite extension of w which belongs to the safety property $\mathcal{A}(\Phi)$, since per definition all prefixes of a trace have to be in the finitary property Φ. Therefore, the implementation shows at least one trace, which does not belong to the safety property $\mathcal{A}(\Phi)$. Neither $Tr(I) = \mathcal{A}(\Phi)$ nor $Tr(I) \subseteq \mathcal{A}(\Phi)$ hold. The safety property is *violated*. From the fact that there is an infinite trace of the implementation which does not belong to the safety property, we cannot state whether the relations $Tr(I) \supseteq \mathcal{A}(\Phi)$ and $\mathcal{A}(\Phi) \cap Tr(I) = \emptyset$ hold or not.

If $w \in \Phi$ then it is still possible that there is a longer trace w', which belongs to Φ or not. Therefore, we cannot state, whether a possible infinite extension of w belongs to the safety property or not. Thus, it is impossible to say anything about a relation between $\mathcal{A}(\Phi)$ and $Tr(I)$.

Guarantee properties. Let Φ be the finitary property, from which the guarantee property [4] $\Pi = \mathcal{E}(\Phi)$ is constructed. If $w \in \Phi$, then every infinite extension of w belongs to the guarantee property $\mathcal{E}(\Phi)$, since there has only to be one prefix of the infinite trace, which belongs to Φ. Therefore, the implementation shows at least one trace, which belongs to the guarantee property $\mathcal{E}(\Phi)$. In this case we can reject, that the set of traces of the implementation and the guarantee property are disjoint $Tr(I) \cap \mathcal{E}(\Phi) = \emptyset$. The guarantee property is *validated*. We cannot guarantee, that all traces of the implementation belong to the guarantee property, but at least one time the implementation is able to show the required property. In this case we can neighter reject nor verify the other relations.

[4] The complement of a safety property is a guarantee property. Therefore, we may use a contra position to reason about test results for a guarantee property. But we prefer to give an own proof idea.

If $w \notin \Phi$, then it is still possible that there is an other longer trace w', which belongs to Φ or not. Therefore, we cannot state, whether an infinite extension of w belongs to the guarantee property property or not. Thus, it is impossible to say anything about a relation between $\mathcal{E}(\Phi)$ and $Tr(I)$.

Recurrence and persistence properties. For recurrence and persistence properties we cannot make any statement about the relation between the traces of the implementation and of the property. From a prefix w of an infinite trace σ we cannot conclude iff σ fulfills or violates the recurrence or the persistence property.

Testable properties. Within this section safety and guarantee properties were identified to be as testable properties, since we can at least observe a violation of a safety property and a validation of a guarantee property. Contrary to verification methods which verify for every property that an implementation fulfills the property, in testing, it is even impossible to verify that the implementation fulfills a testable property.

3 Representation of temporal properties

Contrary to other test methods which only generate test cases for safety properties, we define a test case for a safety and a guarantee property. For modeling the testable properties we use an automata theoretic approach. The safety property is modeled by a labeled transition system (LTS) and the guarantee property is modeled by a finite automaton (FA), which has structural constraints for its end state definition.

Labeled transition system. A *labeled transition system* is LTS $= (Q, E, R, q_0)$, where

- Q is a set of states,
- E is a set of labels resp. events,
- $R \subseteq Q \times E \times Q$ is a transition relation, and
- $q_0 \in Q$ is the initial state.

Observable events. Since the labeled transition system specifies all sequences of events of the system which should be tested, during the test not all events are observable. The events E are determined by $E = \{\tau\} \cup (\cup_{i=1}^{n}(I_i \cup O_i))$. τ represents internal events e.g. assignments or clock events and I_i and O_i are the inputs and outputs of the i-th process. The inputs and outputs are also called *communication events*. Let the processes 1 to m be the processes which control the test. In the sequel we call them *test processes*[5]. We define

- $OI = \cup_{i=1}^{m} I_i$ to be the *observable inputs*,
- $OO = \cup_{i=1}^{m} O_i$ to be the *observable outputs*, and
- $OE = OI \cup OO$ to be the *observable events*.

[5] According to ISO/IEC IS 9646 [13] a system under test is controlled by so-called upper and lower tester.

Traces and observables of a labeled transition system. A *trace* is a sequence of events E and an *observable* is a sequence of observable events OE. We define the traces of a labeled transition system LTS $= (Q, E, R, q_0)$ from a state in $M \subseteq Q$ to a state in $N \subseteq Q$ by:

$$Tr(\text{LTS}, M, N) = \{\langle e_0, \ldots, e_n\rangle \in E^* \mid$$
$$\exists \langle s_0, \ldots, s_{n+1}\rangle \in Q^* : (s_0 \in M \ \wedge \ s_{n+1} \in N \ \wedge \ \forall i \in 0 \ldots n : (s_i, e_i, s_{i+1}) \in R)\}$$

We define the observables of a labeled transition system LTS from a state in $M \subseteq Q$ to a state in $N \subseteq Q$ by:

$$Ob(\text{LTS}, M, N) = OE \copyright Tr(\text{LTS}, M, N)$$

The traces of a labeled transition system LTS with an observable o are:

$$Tr(\text{LTS}, o) = \{t \in Tr(\text{LTS}, \{q_0\}, Q) \mid o = OE \copyright t\}$$

All infinite traces, which are possible in the labeled transition system belong to the safety property or alternatively, the safety property is defined by:

$$\mathcal{A}(Tr(\text{LTS}, \{q_0\}, Q))$$

Finite automaton. A *finite automaton* is defined by a tuple FA $= (S, E, \delta, s_0, F)$, where

- S is a finite set of states,
- E is a set of events,
- $\delta \subseteq S \times E \times S$ is a transition relation,
- $s_0 \in S$ is the initial state, and
- $F \subseteq S$ is a set of end states.

Traces and observables of a finite automaton. A finite automaton FA $= (S, E, \delta, s_0, F)$ can be interpreted as a labeled transition system (S, E, δ, s_0). Thus, the traces and observables of a finite automaton are defined by the traces and observables of the corresponding labeled transition system. For specifing a guarantee property the automaton is not allowed to contain transitions from end states to non-end states. All infinite traces, where the automaton cycles infinitely often in an end state, belong to the guarantee property. Alternatively, the guarantee property is defined by:

$$\mathcal{E}(Tr(FA, \{s_0\}, F))$$

A complete treatment of the dependency between automata and temporal properties can be found in [16] or [14].

4 Representation of a test case

A test case [13] describes a tree where the nodes are observable events. We express a tree as a set of observables. The observables of a test case can be grouped into three disjoint sets - the observables which cause a PASS, a FAIL or an INCONCLUSIVE verdict. We call them *pass, fail* and *inconclusive observables*[6].

[6] The meaning of these observables is explained in section 5.

Definition. A test case is defined by a triple $TC = (Pass, Fail, Inco)$, where

- $Pass \subseteq OE^*$ are *pass observables*,
- $Inco \subseteq OE^*$ are *inconclusive observables*, and
- $Fail \subseteq OE^*$ are *fail observables*.

Constraints. There are restrictions on the set of observables of a test case.

- A test verdict must be unique. There is no observable which cause two verdicts at once. Formally, this is expressed by:

$$Pass, Fail, Inco \text{ are pairwise disjoint}$$

- After deriving a test verdict, it is assumed that the test case is finished and could not be continued. This can be expressed by:

$$\forall v, w \in Pass \cup Fail \cup Inco : v \not\sqsubset w$$

- The tree of a test case cannot have arbitrary branching. The test processes perform a sequence of fixed outputs, and afterwards, they have to wait for the reaction of the system under test. Then the test processes again can perform a sequence of fixed outputs. This means for outputs a test case has a branching of size 1 and for inputs a test case can have arbitrary branching. We express that by the notion of *alternative* observables. Formally, this is expressed by

$$\forall v, w \in Pass \cup Fail \cup Inco : v \; altto \; w$$

We introduce two notations of alternative observables.

- An observable v is *alternative* to an observable w, if they have a prefix p in common and the first elements in which they differ is an input. Formally, this is denoted by $v \; altto \; w$ *iff*

$$\exists p, v', w' \in OE^*; a, b \in OI : v = p \cdot a \cdot v' \wedge w = p \cdot b \cdot w' \wedge a \neq b$$

- An observable v is *minimal alternative* to an observable w, if v is an alternative to w and v is only one element longer than the common prefix. Formally, this is denoted by $v \; \underline{altto} \; w$ *iff*

$$\exists p, w' \in OE^*; a, b \in OI : v = p \cdot a \wedge w = p \cdot b \cdot w' \wedge a \neq b$$

Test verdicts for a test case. An implementation I is driven according to the test case $TC = (Pass, Fail, Inco)$ and performs the observable w. According to the test case we give the following test verdict $verdict(w, TC)$. We give a *PASS*, if a prefix of w is a *pass observable*. We give a *FAIL*, if a prefix of w is a *fail observable*. We give an *INCONCLUSIVE*, if a prefix of w is an *inconclusive observable*, or the implementation does not respond during test time and w is a prefix of a *pass observable*. Formally, the test verdict is defined by $verdict(w, TC) =$

- *PASS* *iff* $\exists v \in Pass : v \sqsubseteq w$
- *FAIL* *iff* $\exists v \in Fail : v \sqsubseteq w$
- *INCONCLUSIVE* *iff* $(\exists v \in Inco : v \sqsubseteq w) \; \vee \; (\exists v \in Pass : w \sqsubset v)$

5 Formalizing the test case generation

Now we know that a test case consists of three disjoint sets of observables and each set corresponds to a test verdict. In this section we define the relation between observables representing a test case, a labeled transition system representing an safety property and a finite automaton representing a guarantee property.

An test case assigns a PASS verdict, if the safety property is not violated and the guarantee property is validated during the test. Additionally, we require for a PASS verdict that the system under test again reaches the initial state, such that the next test case can be applied. It gives an INCONCLUSIVE verdict, if the safety property is not violated, but also the guarantee property is not validated and it gives a FAIL verdict, if the safety property is violated during the test. In the sequel we define the *pass*, *fail* and *inconclusive observables*, such that the above requirements are fulfilled.

- A *pass observable* is an observable from which we can conclude, that the labeled transition system performs a cycle from its initial state back to the initial state and the finite automaton transits from the initial state to an end state.
- An *inconclusive observable* is an observable of the labeled transition system which has a prefix with a *pass observable* in common, but the first event in which they differ is an input.
- A *fail observable* is an arbitrary sequence of observable events which has a prefix with a *pass observable* in common and the first event in which they differ is an input.

5.1 Possible and unique pass observables

Since we only see a subset of events, which are performed by the implementation, and since we want to reason about observable events or about reached states, we adapt the UIO-method (unique input output sequences) to our situation. For defining the *pass observables* of a test case we introduce the notion of *possible* and *unique pass observables*. A *possible pass observable* is an observable, such that there exists a corresponding trace, where the labeled transition system LTS can perform a cycle from the initial state to the initial state and the finite automaton FA transits form the initial state to an end state. We define the set of *possible pass observables PPO* by :

$$PPO = Ob(\mathrm{LTS}, \{q_0\}, \{q_0\}) \cap Ob(\mathrm{FA}, \{s_0\}, F)$$

A *possible pass observable* does not ensure, that every corresponding trace leads the labeled transition system from its initial state back to its initial state and the finite automaton transits from its initial state to an end state. For this aim we define so called *unique pass observables UPO* by:

$$UPO = \{w \in PPO \mid \overline{Tr(\mathrm{LTS}, w)} \subseteq [Tr(\mathrm{LTS}, \{q_0\}, \{q_0\}) \cap Tr(\mathrm{FA}, \{s_0\}, F)]\}$$

Since we only consider the maximal corresponding traces of an observable w: $\overline{Tr(\mathrm{LTS}, w)}$, this definition works only if the initial state of the labeled transition system is a stable state, i.e. only observable events can cause progress in the initial state.

5.2 Test case definition

Now we define a test case $TC = (Pass, Fail, Inco)$ for a labeled transition system LTS and a finite automaton FA .

- ***Pass.*** For the *pass observables* of the test case *Pass* we take a subset of the shortest *unique pass observables* $\underline{UPO}$ (see 1.). Each element of the *pass observables* must be alternative to each other element (see 2.) and there is no further shortest *unique pass observable*, which is alternative to all *pass observables* (see 3.).

  1. $Pass \subseteq \underline{UPO} \ \wedge$
  2. $\forall v, w \in Pass : (v \neq w \rightarrow v \ altto \ w) \ \wedge$
  3. $\forall v \in \underline{UPO} : (v \in Pass \vee \exists w \in Pass : \neg(v \ altto \ w))$

- ***Inco.*** For the *pass observables Pass* we define the shortest *inconclusive observables* of the test case *Inco*. *Inco* denotes the minimal alternative observables of *Pass*.

$$Inco = \{v \in Ob(\text{LTS}, \{q_0\}, Q) \mid \exists w \in Pass : v \ \underline{altto} \ w\} - pref(Pass)$$

- ***Fail.*** The *fail observables Fail* are the minimal alternatives of the *pass* and *inconclusive observables*.

$$Fail = \{v \in OE^* \mid \exists w \in Pass \cup Inco : v \ \underline{altto} \ w\} - pref(Pass \cup Inco)$$

6 The project

The work presented in this paper is performed at the University of Berne within the research project *'Conformance Testing - A Tool for the Generation of Test Cases'*. Within this project an SDL description [2, 4] is used to specify a safety property and a Message Sequence Chart (MSC) [5, 6] is used to specify a guarantee property[7]. The output is a test case in TTCN notation (cf. part 3 of [13]).

6.1 Specification and Description Language (SDL)

An SDL specification can be interpreted as a labeled transition system. The relation between an SDL description and a labeled transition system (Q, E, R, q_0) can be described in the following way. Intuitively Q denotes the global system states, which are determined by the control states of the processes, the contents of the signal queues and the values of the variables. The state q_0 is the initial state of the SDL specification. The transition relation R determines for every state $q \in Q$ and for every event $e \in E$ the corresponding next global state of the SDL system. The events E describe the actions of the SDL processes (e.g. inputs, outputs, tasks, decisions, etc.). Since there exist different approaches to derive a labeled transition system from an SDL specification [17, 10], we do not want to go into details here.

[7] According to ISO/IEC IS 9646 [13] the guarantee property (i.e. the MSC) can be interpreted as the *test purpose* of the test case.

6.2 Message Sequence Chart (MSC)

An MSC (e.g. Figure 3 (a)) describes a partially ordered set of events. It can be interpreted as a finite automaton. The automaton accepts traces, which contain the communication events of the MSC and which are compatible with its partial order. The relation between an MSC and a finite automaton is described in two steps by means of the example in Figure 3. The automaton *Automaton 1* explained in the first step accepts exactly the sequences of events which are defined by the partial order of the MSC *MSC 1*. In a second step *Automaton 1* is extended by additional events and *Automaton 2* is gained.

- **Step 1:** *Automaton 1* in Figure 3 accepts exactly the sequences of events, which are compatible with the partial order of the MSC *MSC 1*. One way for the translation of an MSC into a finite automaton is described in [7]. *MSC 1* consists of two instances `P1` and `P2`, which exchange the signal `CR` two times. It describes a partial ordered set of communication events, which allows the traces `<P1!CR,P1!CR,P2?CR,P2?CR>` and `<P1!CR,P2?CR,P1!CR,P2?CR>`. *Automaton 1* accepts these traces by transiting from the initial state s_0 to the final state f.

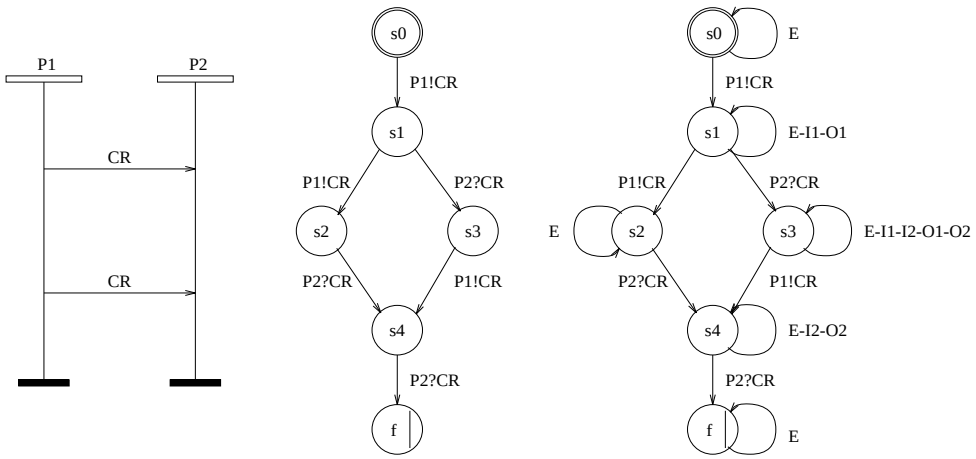
- **Step 2:** An MSC describes a part of the signal exchange of an SDL run by a partially ordered set of events. Our approach compares traces of a finite automaton representing an MSC and traces of a labeled transition system representing an SDL description. In order to do this, the finite automaton must be able to accept events of the labeled transition system, which are not explicitly mentioned by the MSC.

  For this aim the finite automaton is extended by *Null* transitions which consume arbitrary events of the labeled transition system without changing the state. For test case generation we require, that the signal exchange of the MSC is performed without interrupts, i.e. between two communication events on an instance axis the corresponding process is not allowed to perform further communication events[8]. To ensure this, for some states the Null transitions are restricted by certain events which should not cause a Null transition.

  The example in Figure 3 may clarify the extension. *Automaton 2* is gained from the *Automaton 1* by introducing Null transitions for every state. Since we do not allow further communication events of an instance i between two communication events on its instance axis, we disallow its outputs O_i and its inputs I_i for some states. E.g. in state s_1 the instance `P1` has already performed the communication event $P1!CR$ and should perform the communication event $P1!CR$. Therefore, in state s_1 we exclude the outputs O_1 and inputs I_1 of the instance `P1` from the Null transitions. This fact is stated by the arrow inscription $E - I_1 - O_1$. In same manner the Null transitions of state s_2, s_3 and s_4 are constructed. In the start state s_0 and in the final state f all possible communication events E are valid.[9]

[8] This restriction may be weakened to allow optional signals or abstractions in the MSC description.

[9] According to the test case definition in ISO/IEC IS 9646 [13] these events can be interpreted as the preamble and the postamble of the test case.



(a) MSC 1 (b) Automaton 1 (c) Automaton 2

Figure 3: MSC and corresponding finite automata

6.3 Calculation of a test case

Within section 5.2 a test case for a given labeled transition system (representing a safety property) and a given finite automaton (representing a guarantee property) is defined, but there is no algorithm to calculate it. By calculating a test case we have to solve a typical reachability problem, i.e. sometimes a certain event is executed or a certain state is reached.

For finite automata the reachability problem is solved and there exist efficient algorithms to calculate shortest traces, which lead to a certain state or contain a certain event [12]. But the decidability of the reachability problem of the labeled transition system depends heavily on its design. In [3] it is proved, that the reachability problem for communicating finite state machines which communicate by means of unbounded FIFO buffers is undecidable. Subsequently, the reachability problem for a labeled transition system which represents asynchronously communicating processes (i.e. the base model of SDL) is undecidable.

Our way to search for observables with the required properties is to simulate the labeled transition system and the finite automaton in parallel (i.e. a kind of *'on the fly validation'* [15]). There are different search methods, like depth search and breadth search. Breadth search is not usable, since it is impossible to store all states[10]. Also depth search is not applicable, since it is not possible to 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, then the search may be repeated with a higher bound or stopped without results.

The procedure of generating test cases. The procedure of generating a test case based on an SDL description and an MSC can be structured in four steps:

[10]Note, a state represents a global state of the SDL system, i.e. the control states of the processes, the contents of the queues and the values of the variables.

- **Step 1:** In a k-bounded depth search with increasing bound k *possible pass observables* are calculated.
- **Step 2:** Based on the *possible pass observables* we calculate the *unique pass observables*. If there are no *unique pass observables* we go back to step 1.
- **Step 3:** We choose a subset of the shortest *unique pass observables*, which are alternative to each other. These are the *pass observables* of our test case. Based on the *pass observables* we calculate the corresponding *inconclusive observables*.
- **Step 4:** The *pass observables* and the *inconclusive observables* have to be transformed into TTCN. Furthermore, the *fail observables* have to be added by means of a TTCN default behavior description.

Further information on the project and a complete example for our approach can be found in [8] and [9].

Summary and Outlook

The presented paper deals with the automatic generation of test cases for temporal properties. Temporal properties are classified by means of the Borel Hierarchy. The safety and the guarantee property are identified to be testable temporal properties. For the testable temporal properties a test case definition is given. The test case definition is applied to practical testing by relating the safety property to an SDL description and by relating an MSC to a guarantee property. Our approach is implemented at the University of Berne and its applicability for real systems will be proven within a following case study.

References

[1] B. Alpern and F.B. Schneider. Recognizing Safety and Liveness. Distributed Computing, 2:117–126, 1987.

[2] Ferenc Belina, Dieter Hogrefe, and Sarma Amardeo. SDL with Applications from Protocol Specification. Prentice Hall International, 1991.

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

[4] SG X CCITT. Recommendation Z.100: Specification and Description Language (SDL), 1992. Geneva.

[5] SG X CCITT. Recommendation Z.120: Message Sequence Chart (MSC), 1992. Geneva.

[6] Jens Grabowski, Peter Graubmann, and Ekkart Rudolph. The Standardisation of Message Sequence Charts. Proceedings of the IEEE Software Engineering Standards Symposium 1993.

[7] Jens Grabowski, Dieter Hogrefe, Peter Ladkin, Stefan Leue, and Robert Nahm. Conformance Testing - A Tool for the Generation of Test Cases. Interim Report of the F & E project contract no. 233, funded by Swiss PTT, 1992.

[8] Jens Grabowski, Dieter Hogrefe, and Robert Nahm. A Method for the Generation of Test Cases Based on SDL and MSCs. Technical Report IAM 93-010, University of Berne, Switzerland,1993.

[9] Jens Grabowski, Dieter Hogrefe, and Robert Nahm. Test Case Generation with Test Purpose Specification by MSCs. Proceedings of the 6th SDL Forum, North-Holland, 1993.

[10] Dieter Hogrefe. Automatic Generation of Test Cases from SDL-Specifications. SDL-Newsletters, 12, 1988.

[11] Gerard J. Holzman. Design and Validation of Computer Protocols. Prentice-Hall International, Inc., 1991.

[12] John E. Hopcroft and Jeffrey D. Ullman. Introduction to Automata Theory, Languages and Computation. Addison Wesley, 1979.

[13] ISO/IEC JTC 1/SC 21 N. Information Technology - Open System Interconnection - Conformance Testing Methodology and Framework - Part1-5. International Standard 9646, ISO/IEC, 1991.

[14] M. Kaminski. A Classification of ω-regular Languages. Theoretical Computer Science, 36:217–229, 1985.

[15] Günther Karjoth. Generating Transition Graphs from LOTOS Specifications. In Michel Diaz and Roland Groz, editors, FORTE'92, pages 275–287, October 1992.

[16] Zohar Manna and Amir Pnueli. A Hierarchy of Temporal Properties. Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, pages 377–408. ACM Press, 1990. 1990 ACM-0-89791-404-X/90/0008/3777.

[17] Robert Nahm. Semantics of Simple SDL. Proceedings of the GI/ITG workschop on formal description techniques for distributed systems in Magdeburg (Germany), 1993.

A Mathematical notation

Set of sequences. Let A be an arbitrary set, then we define the following three sets

- A^* are the finite sequences over A,
- A^ω are the infinite sequences over A and
- $A^\infty = A^* \cup A^\omega$ are the finite and infinite sequences.

Operations on sequences. Let $S \subseteq A^\infty$, $t, u, v \in A^\infty$ and $a, b, c, d, a_0, \ldots, a_n \in A$

- $\langle\rangle$ is the empty sequence,
- $\langle a_0, \ldots, a_n \rangle$ is the finite sequence consisting of the elements $a_0, \ldots, a_n$,
- $t \cdot u$ denotes the concatenation of t and u (Note, if t is infinite the $t \cdot u = t$),
- $t \sqsubset u$ "t is a strict prefix of u" holds, iff $\exists v \neq \langle\rangle : t \cdot v = u$,
- $t \sqsubseteq u$ "t is a prefix of u" holds, iff $\exists v : t \cdot v = u$,
- $\#t$ denotes the length of t (Note, if t is infinite then $\#t = \infty$),
- a©t denotes the filtered trace of t, which contains only the element a, e.g. a© $< a, b, a, c > = < a, a >$. As a generalisation of this *filter* operation, the first operand may also be a set,
- $f : A \to A'$ can be canonically extended to sequences, by $f(< a_0, \ldots, a_n, \ldots >) = < f(a_0), \ldots, f(a_n), \ldots >$,
- $f : A^\infty \to A'^\infty$ can be canonically extended to sets of sequences by $f(S) = \{f(t) \mid t \in S\}$,
- $\mathit{pref}(S) = \{t \mid \exists u \in S : t \sqsubseteq u\}$, is the set of prefixes of S,
- $\overline{S} = \{t \in S \mid \neg\exists u \in S : t \sqsubset u\}$ are the maximal sequences and
- $\underline{S} = \{t \in S \mid \forall u \in S : \#t \leq \#u\}$ are the shortest sequences.