Formal Test Purposes and The Validity of Test Cases

Peter H. Deussen
Fraunhofer FOKUS, Berlin, Germany
deussen@fokus.fhg.de

Stephan Tobies
Nokia Research Center, Bochum, Germany
Stephan.Tobies@nokia.com

February 1, 2008

Abstract

We give a formalization of the notion of test purpose based on (suitably restricted) Message Sequence Charts. We define the validity of test cases with respect to such a formal test purpose and provide a simple decision procedure for validity.

1 Introduction

The quality of a test system directly influences the quality of the tested implementation: high quality test systems are essential to obtain high quality implementations. Hence, a common problem in the testing area is the so-called “test the tester” problem [12]: how can the validity of a test system with respect to a given specification, and therefore the quality of the test system, be assured? To put it in conformance testing terminology: how can it be assured that a test case achieves its test purpose?

One approach used to obtain valid test systems is the derivation of test cases from formal specifications or test purpose definitions. Other approaches focus on the manual or automated simulation against a formal specification (see [14] for a description of tools that employ these two approaches). While many modern telecommunication protocols come with (semi-) formal specifications of test purposes, a formal protocol description is provided only in very few cases (see [5] for a notable exception). For example, Internet Protocols defined in RFCs use natural language to define the semantics of the specification. Due to this, a formal description of the specification would have to be elaborated to allow for an automatic generation of valid test cases. Additionally, even if formal descriptions are available, automated generation methods only

---

*This paper appeared in the proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002), number 2529 Lecture Notes in Computer Science.
generate test skeletons that need to be manually refined to obtain executable test cases for
the execution against a concrete system implementation. For all these reasons, the
implementation of test cases is still performed mainly in a manual manner.

In this paper, we give a new answer to the “test the tester problem”, namely, to
check the validity of a (possibly hand-written) test case against a formal test purpose
definition. It does not rely on the existence of a formal description of the system under
test (SUT) or the test system, but requires a formally defined test purpose. From this test
purpose, the allowed and required behavior of the test case is derived. This information
is then used in a guided simulation of the executable test system to determine whether
the test system is valid with respect to this test purpose. Since our approach is solely
based on test purposes, it is not necessary to develop a complete formal specification of
the system as test purposes are only a partial description of the system. We use Message
Sequence Charts (MSC) as the formal test purpose description language [6], which is
widely used in the system development process in the telecommunication area. This
allows for an easy re-use of the uses-cases developed during system design as a solid
basis for the test purpose definition. This further reduces the work necessary for the
test purpose specification.

Despite the fact that MSCs are widely used to capture test purpose, theoretical
studies of MSCs so far seem to have failed to address the following issues:

• What does it mean for a test case to implement a test purpose, i.e., when is a test
case valid w. r. t. a test purpose?

• When is an MSC a well-formed test purpose, i.e., when does an MSC characterize
behavior that is indeed (black-box) testable?

We address these issues using a semantics for MSC based on pomsets [11, 4] in the
spirit of [8]. We then describe a simple decision procedure for the validity of test cases
w. r. t. a test purpose and prove its correctness.

The paper is structured as follows: Section 2 of this paper introduces the partial or-
der semantics of MSCs and their usage as formal test purposes. In Section 3, we define
test case validity, describe the decision procedure and prove its correctness. Section 4
presents one possible implementation design for an MSC based test validator. Section 5
concludes. Proofs of key lemmata and theorems can be found in the appendix.

2 Formal Test Purposes

To check (or even define) validity of a test case wrt. a test purposes, we need a formal
definition of a test purpose together with suitable semantics. In this section, we sug-
gest a formalism to formally express test purposes and establish a set of criteria that
guarantee that a test purposes indeed describes (black-box) testable behavior.

We use Message Sequence Charts (MSCs) to express formal test purposes because
they are widely used to capture test purposes and semantics based on different ap-
proaches are available. We have chosen semantics based on pomsets [4, 11] in an
adaption of the definition of [8] to better suit our purposes. The particular choice of
Message Sequence Charts. The MSCs in Fig. 1 serve as an explanatory example for the basic MSC language as used throughout this paper. The most fundamental constructs of MSCs are instances and messages. Instances represent components or communication interfaces that exhibit a sequential behaviour. Our example MSC $m_1$ consists of three instances $p$, $q$, and $r$. A message exchange between a sending instance $p$ and a receiving instance $q$ comprises two events $!_{p,q}a$ and $?_{p,q}a$ for sending the message $a$ at $p$ and for receiving $a$ at $q$, respectively. Graphically, messages are depicted by arrows between instances labeled with messages.

Events are considered to be causally or temporally ordered only if they are located at the same instance (in this case the ordering is top-to-bottom), or if they are part of the same message exchange. In our example $m_1$, the event $!_{p,q}a$ precedes the events $?_{p,q}a$ and $!_{p,r}b$, but no assumption on an ordering of the events $!_{p,r}b$ and $!_{q,r}c$ is expressed, even if $!_{p,r}b$ is drawn above $!_{q,r}c$.

There is a way to express the concurrency of events of the same instance: the concurrent region (coregion, for short). Coregions are depicted by dashed sections on the corresponding instance line bordered by small horizontal bars: the events that occur on this dashed section are supposed to happen in parallel. In our example, the events $?_{p,r}b$ and $?_{q,r}c$ are temporally unrelated. On the other hand, it is possible to use general order arrows (dotted lines between events with an arrow head in their middle section) to express causal orderings of events on different instances. In $m_1$, the event $!_{q,r}c$ precedes $!_{p,q}d$. Finally, the MSC language allows to express message exchange with the environment of a MSC; e.g. in $m_1$ the message $e$ is send to the environment.
of this MSC.

The MSC formalism provides not only communication primitives but also control structures. For our purposes, only the \textit{alt operator}, modeling nondeterministic choice, is of importance. \textit{m}_2 in Fig. 1 shows an example: A choice between sending \textit{a} from \textit{p} to \textit{q} and sending \textit{b} from \textit{q} to \textit{p} is expressed. A final construct considered in this paper is that of \textit{conditions}. Conditions model global states or predicates related to more than one instance; \textit{m}_2 contains two conditions \textit{C}_1 and \textit{C}_2. It is not an easy task to assign a formal meaning to conditions. However, we use conditions only to express test verdicts and handle them formally in a special way. We will discuss this topic in detail in a later section.

Other important concepts of the basic MSC language not covered in this paper are: \textit{loop inline expressions} (since tests are finite, loops occurring in test purposes comprises always finite, fixed boundaries and therefore can be unfolded), and especially \textit{timers}, which require extra considerations and will be dealt with in forthcoming work.

\textbf{Expressing test purposes.} We will use the MSC formalism to capture test purposes in the following way: the set of instances is partitioned into a non-empty set of port instances and a non-empty set of SUT instance. Intuitively, the port instances represent the different ports (PCOs, interfaces) at which the SUT interacts with its environment. Conditions that span the port instances are used to assign the test verdicts.

The SUT instances are used as “syntactic sugar” and serve two purposes: (1) as communication partners for the port instances, and (2) to impose an ordering of the sequence of messages. The same could be achieved by using communication with the environment and generalized orderings, but our approach leads to a more concise and intuitive representation of the test purpose and matches the common usage. Fig. 2 shows the two alternative ways of depicting a simple test purpose: after having received the message \textit{a} on both its ports \textit{p} and \textit{q} (in arbitrary order), the SUT answers by sending the message \textit{b}, again both on port \textit{p} and \textit{q}. If the message is sent on port \textit{p} before it is sent on port \textit{q} then the SUT shall pass the test, otherwise it shall fail. We will come back to this example later in this paper.

2.1 Partial Orders

We quickly recapitulate how pomsets can be used to assign a semantics to MSCs. We start by introducing the basic notations used throughout this paper.

To avoid tedious notation, we fix the following convention: if a structure \textit{S} = \langle \textit{A}, \textit{B}, \ldots \rangle is introduced, the components of \textit{S} will be denoted by \textit{A}_S, \textit{B}_S, \ldots

For some set \textit{A}, \mathcal{P}(\textit{A}) is the set of all subsets of \textit{A}. For \textit{R} \subseteq \textit{A} \times \textit{B} and \textit{a} \in \textit{A}, we denote the \textit{image} of \textit{a} under \textit{R} by \textit{R}(\textit{a}) = _{\textit{df}} \{ \textit{b} \in \textit{B} : \textit{a} \textit{R} \textit{b} \}. For \textit{C} \subseteq \textit{A} we define \textit{R}(\textit{C}) = _{\textit{df}} \bigcup_{\textit{a} \in \textit{C}} \textit{R}(\textit{a}).

The \textit{inverse} \textit{R}^{-1} of a relation \textit{R}, the identity relation \textit{id}_A on \textit{A}, the relational composition \textit{R} \cdot \textit{S} or two relations \textit{R}, \textit{S}, the transitive closure \textit{R}^+ of \textit{R}, and the reflexive-transitive closure \textit{R}^* of \textit{R} are defined in the usual manner.
For the rest of this paper let us fix a finite alphabet $\Sigma$. A labeled partial order (lposet, for short) over $\Sigma$ is a structure $x = \langle E, <, \lambda \rangle$ where $E$ is a finite set of events, $< \subseteq E \times E$ is an (irreflexive) partial order, and $\lambda : E \rightarrow \Sigma$ is a labeling function.

Let $x$ be a lposet and let $e_1, e_2 \in E_x$. We use the following notions: The reflexive closure of $<_x$ is $\leq_x = \text{df} <_x \cup \text{id}_{E_x}$. Unrelated events are called concurrent, i.e., $e_1 \text{ co}_x e_2 \iff \text{df} e_1 \nleq_x e_2 \& e_2 \nleq_x e_1$, while related events are in line: $e_1 \text{ li}_x e_2 \iff \text{df} e_1 <_x e_2 \lor e_2 <_x e_1$.

The downward closure of a set $D \subseteq E_x$ is $\downarrow_x D = \text{df} \leq^{-1}_x(D)$. If $D = \downarrow_x D$ holds, then $D$ is called downward closed in $x$. By $\mathbf{C}(x)$ we denote the set of downward closed sets in $x$. If $D \subseteq E_x$, then $x[D] = \text{df} \langle D, <_x \cap (D \times D), \lambda \upharpoonright D \rangle$ is the lposet generated by $D$ in $x$ ($\lambda \upharpoonright D$ denotes the restriction of $\lambda$ to $D$).

Pomsets. Lposets $x$ and $y$ over $\Sigma$ are called isomorphic, written $x \equiv y$, if there is a bijection $f : E_x \rightarrow E_y$ such that $(e_1 <_x e_2 \iff f(e_1) <_y f(e_2)) \& \lambda_x = \lambda_y \circ f$ holds. A partially ordered multiset (a pomset for short) over $\Sigma$ is an isomorphism class of lposets, i.e., a set $[x] = \text{df} \{ y : x \equiv y \}$. We fix the convention, that pomsets are denoted by boldfaced small letters $x, y, z$. Moreover $x$ is assumed to be the equivalence class $[x]$ of $x$. By this convention, $E_x$ always denotes the set of events of a representative $x$ of $x$. The class of pomsets over $\Sigma$ is denoted by $\mathbf{P}(\Sigma)$.

Fig. 3 shows examples of pomsets. Graphically, we represent pomsets as directed acyclic (not necessarily connected) graphs. Nodes are labeled with elements from the underlying alphabet $\Sigma$. Transitive arcs are sometimes omitted.
Let \( x, y \in P(\Sigma) \) be pomsets. Then \( x \) is called a prefix of \( y \)—denoted \( x \preceq y \)—iff there are representatives \( x \in x \) and \( y \in y \) such that \( E_x \subseteq E_y \) & \( C(x) \subseteq C(y) \) holds. If there are representatives \( x \in x \) and \( y \in y \) such that \( E_x = E_y \) & \( <_x \subseteq <_y \) holds, then \( x \) is called less sequential than \( y \). This is denoted by \( x \prec y \). It is easy to see that both \( \preceq \) and \( \prec \) partially order \( P(\Sigma) \). In Fig. 3, \( x \preceq y, x \preceq z, \) and \( y \prec z \), holds.

An alternative definition of the prefix relation \( \preceq \) and the ordering by the degree of sequentiality \( \prec \) can be obtained by introducing the notion of weak homomorphisms [2].

Special pomsets that will be encountered in this paper are:

1. Letters \( a = \{\{a\}, \emptyset, a \mapsto a\} \) for \( a \in \Sigma \) (we abuse \( a, b, c, \ldots \) to denote both letters from \( P(\Sigma) \) and from \( \Sigma \)).
2. Strings \( \{\{0, \ldots , n-1\}, <, i \mapsto a_i\} \) for \( a_0a_1 \ldots a_{n-1} \in \Sigma^* \), where \( < \) denotes the standard order relation on integers.
3. The empty word \( \varepsilon = [\emptyset, \emptyset, \emptyset] \).

In this paper we do not distinguish between strings and pomset strings, i.e, if \( \Sigma \) is an alphabet then \( \Sigma^* \) is considered to be the set of pomsets \( \sigma \) over \( \Sigma \) such that \( <_\sigma \) is a total ordering.

If \( x \in P(\Sigma) \), then by \( \text{lin}(x) = \{\sigma \in \Sigma^* : x \preceq \sigma\} \) we denote the set of linearizations of \( x \).

**Dependencies and Weak Sequential Composition.** A reflexive and symmetric relation \( D \subseteq \Sigma \times \Sigma \) is called a dependence on \( \Sigma \); for the rest of this paper let \( D \) be a dependence on \( \Sigma \). If \( x \) and \( y \) are lposets over \( \Sigma \), such that \( E_x \cap E_y = \emptyset \) holds, then the weak sequential composition \( x \circ_D y \) is defined by

\[
x \circ_D y = \text{df} \quad [E_x \cup E_y, (<_x \cup <_y \cup R)^+, \lambda_x \cup \lambda_y],
\]

where \( R \subseteq E_x \times E_y \) is given by \( e_1 R e_2 \iff_{df} \lambda_x(e_1) D \lambda_y(e_2) \).

A pomset \( x \) is called \( D \)-consistent if we have, for all \( e_1, e_2 \in E_x, e_1 \circ e_2 \Rightarrow \neg\lambda_x(e_1) D \lambda_x(e_2) \). Let \( P(\Sigma, D) \) denote the class of \( D \)-consistent pomsets. Clearly, if \( x \) and \( y \) are \( D \)-consistent, then also \( x \circ_D y \) is.
Another operation on pomsets which is closely related to ◦_D is the unsequentialization via D: If \( x \in P(\Sigma) \), then by \( \langle x \rangle_D \) we denote the pomset \([E_x, R^+, \lambda_x]\), where \( R \subseteq E_x \times E_x \) is defined by \( e_1 R e_2 \iff e_1 \leq e_2 \) & \( \lambda_x(e_1) \leq \lambda_x(e_2) \).

The pomsets in Fig. 3 are all \( D \)-consistent for the dependence \( D \) shown in that figure. We have \( \langle z \rangle_D = y \).

The following lemma justifies the relation between the operations ◦_D and \( \langle \cdot \rangle_D \).

**Lemma 1** Let \( x, y \) in \( P(\Sigma) \). Then \( \langle x \circ_D y \rangle_D = \langle x \rangle_D \circ_D \langle y \rangle_D \).

Some more definitions: if \( A \subseteq \Sigma \) is a set of symbols and \( x \) is a pomset, then \( x \upharpoonright A =_{df} \{ x[\lambda_x^{-1}(A)] \} \). \( x \upharpoonright A \) is called the restriction of \( x \) to \( A \), i.e., \( x \) restricted to those events labeled with elements from \( A \). Finally, a set of pomsets \( X \subseteq P(\Sigma) \) is called pre-closed if \( x \in X \) & \( y \leq x \Rightarrow y \in X \) holds.

### 2.2 Partial Order Semantics for MSCs

To define the semantics of MSCs based on pomsets, we first need to fix an alphabet \( \Sigma_c \) and a dependence \( D_c \) on \( \Sigma_c \).

**Communication Alphabet and Dependence.** Let \( M \) be a set of messages and \( P \) a set of instances fixed throughout this paper. We assume that there is a non-empty set \( T \subseteq P \) of port instances; the instances in \( P - T \) will be called SUT instances.  

---

1The operations \( \circ_D \) and \( \langle \cdot \rangle_D \) impose an interesting and fruitful connection to the theory of Mazurkiewicz traces [10]. Although it is far beyond the scope of this paper it should be noted that pomsets of the form \( \langle x \rangle_D \in P(\Sigma, D) \) are just alternative representations of Mazurkiewicz traces: in fact we have that \( \text{lin}(\langle x \rangle_D) \) is a Mazurkiewicz trace over \( \Sigma \) and \( D \); moreover, the operation \( \circ_D \) coincides with trace concatenation.
we will have $|P - T| = 1$, but the theory presented in the following does not rely on this.

Let $\Sigma_1, \Sigma_2$ be the two alphabets:

1. $\Sigma_1 = \text{def} \{!\} \times P \times M \times P$ is the set of send actions. Its elements $(!, p, m, q)$ will be denoted by $!_{p,q}m$.
2. $\Sigma_2 = \text{def} \{?\} \times P \times M \times P$ is the set of receive actions. Its elements $(?, p, m, q) \in \Sigma_2$ will be denoted by $?_{p,q}m$.

We put $\Sigma_c = \text{def} \Sigma_1 \cup \Sigma_2$ to be the set of communications. The mapping $\iota_c(a)$ identifies the instance of an action $a \in \Sigma_c$, i.e., $\iota_c( (!_{p,q}m) = \text{def} p$ and $\iota_c( ?_{p,q}m) = \text{def} q$.

We put $\Sigma_o = \text{def} \{a \in \Sigma_c : \iota_c(a) \in T\}$ to be the set of tester observable actions. For convenience, we furthermore define $\Sigma'_o = \text{def} \Sigma_o \cap \Sigma_1$ and $\Sigma''_o = \text{def} \Sigma_o \cap \Sigma_2$.

The semantics of our example $M$ is given by a pre-closed set of pomsets $X_M \subseteq P(\Sigma_c, D_c)$. We illustrate the construction of $X_M$ only by informal means of an example $m_3$. The translation is done similar to $m_3$.

In the following, $\circ_{c}$ abbreviates $\circ_{D_c}$.

The semantics of our example $m_3$ is given by the set $X_{m_3}$:

$X_{m_3} = \{z \in P(\Sigma_c, D_c) : z \leq x \circ_c y_1 \lor z \leq x \circ_c y_2\}$.

where $x$, $y_1$, and $y_2$ are defined by:

$x = \text{def} \{!, p, a \circ_c !_{q,r}a \circ_c !_{?, p}a \circ_c ?_{q,r}a\}$

$y_1 = \text{def} \{!, p, b \circ_c ?_{r,p}b \circ_c !_{p,q}0 \circ_c ?_{p,q}0 \circ_c !_{r,q}b \circ_c ?_{r,q}b\}$

$y_2 = \text{def} \{!, r, q \circ_c ?_{r,q}b \circ_c !_{q,p}0 \circ_c ?_{q,p}0 \circ_c !_{r,q}b \circ_c ?_{r,q}b\}$

Without a proof (which would require a more formal treatment of the definition of $X_M$) we state:
Lemma 2 If $M$ is a MSC, then $(x)_{D_c} = x$ for all $x \in X_M$.

2.3 Message Sequence Charts as Test Purposes

Now that we have explained how to assign semantics to an MSC, we show how MSCs can be utilized as a formal language to express test purposes. We discuss how the notion of a test verdict can be integrated into an MSC and how it can be guaranteed that an MSC specifies behavior that is amenable to black-box testing.

Verdict assignments. Syntactically, a verdict assignment is expressed by a condition on the port instances on the very end of each terminal alternative of the MSC. Semantically, the condition-like constructs pass, fail, and inconc are not treated as an ordinary condition but as a convenient way to define a verdict assignment:

Let $V = \{\text{pass}, \text{fail}, \text{inconc}, \text{none}\}$ be a set of verdicts and let $V_f = V - \{\text{none}\}$ be the set of final verdicts. A mapping $\nu : X \rightarrow V_f$ for some finite, pre-closed set of pomsets $X$ is called a verdict assignment to $X$ if, for all $x \in X$, we have:

1. $\exists y \in X. x \leq y \land \nu(y) \neq \text{none}$, i.e., every pomset can be extended to a pomset that is assigned a final verdict, and

2. $\nu(x) = \text{none} \Rightarrow \forall y \in X. x < y$, i.e., pomsets that are assigned a final verdict are maximal in $X$.

The verdict conditions drawn in an MSC $M$ are used to define a verdict mapping $\nu_M$. Again, we introduce this informally by the example of $m_3$ from Fig. 2, where $\nu_{m_3}$ is defined by:

$$
\nu_{m_3}(z) = \begin{cases} 
\text{pass}, & \text{if } z = x \circ_c y_1; \\
\text{fail}, & \text{if } z = x \circ_c y_2; \\
\text{none}, & \text{otherwise.}
\end{cases}
$$

It is obvious that not every MSC that satisfies the syntactic restrictions that have been introduced above constitutes a test purpose, i.e., describes behavior of the SUT that can be tested in a black box testing approach. For example consider a modification of $m_3$ from Fig. 2 where the generalized ordering constraints have been eliminated. There the verdict does not depend on the order in which the messages $b$ can be observed at the ports of the SUT but rather on the (SUT-internal) events that cause these messages to be sent. Clearly, such an event is not visible to a black-box test system and hence no test case can distinguish between the behavior of the first and second alternative. In the following we present a number of criteria that an MSC must satisfy to be considered a well-formed test purpose. Later we will see that these criteria indeed guarantee the existence of a valid test case for a test purpose.

\footnote{Alternatively, one could allow verdict conditions to appear also at other places within the MSC and, e.g., use the verdict assignment rules of TTCN-3 \[3\] to resolve the case where different verdicts are encountered during a single run through the MSC.}
**Well-Formed Test Purposes.** First, we define a function that reduces the semantics of an MSC to the information that is available to the test case, i.e., the sequences of events that occur on port instances:

Given test purpose MSC $\mathcal{M}$ with semantics $X_{\mathcal{M}}$. For $x \in X_{\mathcal{M}}$, we define the tester observable traces of $x$ by $\text{obs}(x) =_{df} \text{lin}(x \mid \Sigma_o)$.

A MSC $\mathcal{M}$ is called a well-formed test purpose if it is possible to determine its state (and hence assigned verdict) based on this information in its tester observable traces, i.e., if

$WF_1$. for every $x, y \in X_{\mathcal{M}} \mid \Sigma_o$, $\text{lin}(x) \cap \text{lin}(y) \neq \emptyset$ implies $x = y$.

Unfortunately, this restriction does not yet suffice to guarantee that an MSC describes testable behavior. Another aspect that needs considerations is which party resolves essential choice in the sense of the following definition:

Let $X \subseteq \mathcal{P}(\Sigma_c, D_c)$ be a pre-closed set of pomsets. A pomset $x \in X$ is called a choice point for two actions $a, b \in \Sigma_c$ in $X$ if $x \circ_c a \in X$, $x \circ_c b \in X$, and

$$\{y \in \text{max}_{\leq}(X) : x \circ_c a \leq y\} \neq \{y \in \text{max}_{\leq}(X) : x \circ_c b \leq y\},$$

where $\text{max}_{\leq}(X)$ denotes the $\leq$-maximal pomsets in $X$.

Coming back to example from Fig. 2 with semantics $X_{m_3}$ as defined in (1)–(3), $x \mid \Sigma_o$, is a choice point for $?_{r,p} b$ and $?_{r,q} b$. On the other hand, $e$ is not a choice point even though there are two “available” communications, namely $!_{p,r} a$ and $!_{q,r} a$, since this choice does not alter the reachable maximal configurations.

We require, for a well-formed test purpose, that each choice point is resolved by a message from the SUT:

$WF_2$. If $x$ is a choice point of $X_{\mathcal{M}} \mid \Sigma_o$ for actions $a, b \in \Sigma_o$, then both $a, b \in \Sigma_o^\circ$.

This restriction is necessary because both other possibilities for a choice point ($a, b \in \Sigma_o^\circ$, or $a \in \Sigma_o^\dagger$ and $b \in \Sigma_o^\ddagger$) are undesirable in a test purpose: a choice that has to be resolved by the test case indicates that the test purpose should indeed be (at least) two test purposes, one for each choice of the test case. Otherwise, a deterministic test case will only be able to test the part of the test purpose that corresponds to the (necessarily fixed) way the test case resolves the choice. On the other hand, a choice that can be resolved simultaneously by the test case and SUT leads to problems because it might lead to a race condition where both test case and SUT resolve the choice in an inconsistent manner. This situation bears strong resemblance to the presence of non-local choice in the MSC.

Figure 5 shows examples of malformed MSCs: in $m_5$ exist $x, y \in X_{m_5} \mid \Sigma_o$, with $x \neq y$ and $\sigma = !_{r,p} a \cdot ?_{p,r} b \in \text{lin}(x) \cap \text{lin}(y) \neq \emptyset$, and hence $WF_1$ is violated. Indeed there exist $x, y$ with that property such that $v_{m_5}(x) = \text{pass}$ and $v_{m_5}(y) = \text{fail}$. Taking into account the fact that a test system will only observe $\sigma$ it is clear that $m_5$ does not describe testable behaviour—which verdict should a test system assign after observing $\sigma$? The MSC $m_6$ is malformed because it violates $WF_2$: $e$ is a choice point for the actions $!_{r,p} a_1, !_{r,p} a_2$, and $!_{p,r} e$. In its initial configuration, the test system can either (deterministically) send $a_1$ or $a_2$, but will then not be able to test the behaviour.
of the SUT that corresponds to the respective other choice. Also, what happens if the test system decides to perform action $!_{r,p}a_1$ while the SUT, before it has received $a_1$, performs $!_{p,r}c$? This behaviour is not defined by the MSC. For an example of a well-formed MSC, the reader may verify that $m_3$ from Fig. 2 is indeed well-formed.

3 Test Case Validity

We now define the validity of a test case w.r.t. a well-formed test purpose. Our definition is different from the available conformance relations for labeled transition systems because it assigns different roles to test case and SUT. We show that the well-formedness conditions on MSCs from the previous section suffice to guarantee the existence of a valid test case. Moreover, we give a simple decision procedure that decides validity of a test case and prove its correctness.

First, we need to formalize the notion of a test case. Intuitively, a test case interacts with the SUT by means of exchanging messages and finally assigning a verdict. Formally, we model a test case as follows:

**Test Cases.** A test case is a partial function $\mathcal{T} : \Sigma_o^* \to \Sigma_o^{l} \cup \{\delta\} \cup V_f$, where $\delta$ is a symbol that denotes quiescence of the test case.

A run of a test case $\mathcal{T}$ is a sequence $\sigma_0, \sigma_1, \ldots, \sigma_n$, of words from $\Sigma_o^*$ such that
$\sigma_0 = \varepsilon$, and $\sigma_i \xrightarrow{T} \sigma_{i+1}$ for $0 \leq i < n$, where the relation $\xrightarrow{T}$ is defined by

$$\sigma \xrightarrow{T} \sigma a \iff (T(\sigma) = a \in \Sigma_0^1 \lor T(\sigma) = \delta \land a \in \Sigma_0^0)$$

A run $\sigma_0, \sigma_1, \ldots, \sigma_n$ is called complete if $T(\sigma_n) \in V_f$. Note that it is indeed impossible to extend a complete run due to the definition of $\xrightarrow{T}$.

In the following we will show how to model test case validity as a certain language inclusion problem.

**Test Languages.** Both the runs of a test case and the tester observable traces of a well-formed test purpose naturally induce test languages, i.e., languages $L \subseteq \Sigma_0^*$ together with a verdict assignments $\upsilon_L$.

For a test case $T$, the test language $\langle L_T, \upsilon_T \rangle$ is defined by $L_T = \{ \sigma \in \Sigma_0^*: \varepsilon \xrightarrow{T} \sigma \}$ with verdict assignment $\upsilon_T$ defined by

$$\upsilon_T(\sigma) = \begin{cases} T(\sigma), & \text{if } T(\sigma) \in V_f; \\ \text{none}, & \text{otherwise.} \end{cases}$$

For a well-formed test purpose $M$, the induced test language $\langle L_M, \upsilon_M \rangle$ is defined by setting $L_M = \text{obs}(X_M)$ and, for $\sigma \in L_M$, $\upsilon_M(\sigma) = \upsilon_M(x)$ for the (due to WF uniquely defined) $x \in X_M$ with $\sigma \in \text{obs}(x)$.

It can easily be shown that $\upsilon_T$ and $\upsilon_M$ are well-defined and satisfy the requirements imposed on verdict assignments.

What is the correct relation between $\langle L_M, \upsilon_M \rangle$ and $\langle L_T, \upsilon_T \rangle$ to define validity of $T$ w.r.t. $M$? Clearly, $\upsilon_M$ and $\upsilon_T$ should agree on $L_M \cap L_T$. But what is the right relations between $L_M$ and $L_T$? None of the “obvious” choices leads to a satisfactory notion of validity:

- if we would require $L_M \subseteq L_T$ then there would be no valid test cases for any test purpose that allows (inessential) choice between two actions $a, b \in \Sigma_0^1$ because $L_M$ contains traces for both choices while a deterministic test case would be limited to only a single choice.
- requiring $L_T \subseteq L_M$ would allow the test case to send arbitrary messages to the SUT even though these would not be specified in the test purpose
- if we require $L_T \cap L_M \neq \emptyset$ then the test case would only be required to react to one of the possible many (essential) choices that the SUT might have.

While the first option matches the intuitive meaning of test case validity best, it needs to be modified to eliminate the influence of inessential choice. This is done by means of the following equivalence relation on strings:

Let $L \subseteq \Sigma_0^*$ be a language. We define an equivalence relation $\simeq_L \subseteq L \times L$ by setting $\sigma \simeq_L \rho \iff \rho$ is a permutation of $\sigma$ such that $\sigma \upharpoonright \Sigma_0^1 = \rho \upharpoonright \Sigma_0^1$. The equivalence class w.r.t. $\simeq_L$ of $\sigma \in L$ is denoted by $[\sigma]_L = \{ \rho \in L : \sigma \simeq_L \rho \}$. 

12
valid(test\_purpose M; test\_case T; string ρ) {
    if T(ρ) is undefined then fail;
    if T(ρ) ∈ V_t & T(ρ) ≠ v_M(ρ) then fail;
    else if T(ρ) = ϵ then
        if en(M, ρ) ∩ Σ_o = ∅ then fail;
        else foreach a ∈ en(M, ρ) ∩ Σ_o do valid(M, T, ρ · a);
    else if ρ · T(ρ) ∈ |L_M then fail;
    else valid(M, T, ρ · T(ρ));
    success;
}
where en(M, ρ) = df {a ∈ Σ_o : ρ · a ∈ obs(X_M)}

Algorithm 1: Validation algorithm.

Test Case Validity. Let M be a well-formed test purpose and T be a test case for M. Then T is called a valid test case w. r. t. M if

- for every σ ∈ L_T ∩ L_M, ν_T(v) = ν_M(v), and
- for every σ ∈ L_M with ν_M(σ) ∈ V_t, [σ]_{L_M} ∩ L_T ≠ ∅.

Since we have given the definition both for well-formed test purposes and test case validity, it would be futile to use one to justify the other. What can be shown formally though, is that these notions are compatible in the following sense:

Theorem 1 Let M be a well-formed test purpose. Then there exists a test case T that is valid w. r. t. M.

Also, it is easy to see that there are MSCs that violate one of the well-formedness conditions, for which no valid test case exists.

Deciding Validity. In the following we present an algorithm that decides validity of a test case w. r. t. a well-formed test purpose and establish the algorithm’s correctness. Interestingly, the algorithm does not require the calculation of the \(\simeq_{L_M}\)-classes but only refers to \(obs(X_M)\), \(v_M\), and \(L_M\), which can easily be derived from M.

Theorem 2 Let M a well-formed test purpose and T a test case for M. Then T is valid w. r. t. M iff valid(M, T, ε) does not fail.

4 Practical Considerations

The previous sections have discussed formally the relationship between a test purpose defined using MSC and a test system that implements the test purpose. No assumptions have been made on the test system besides that it is deterministic and that it has observable test events and a final verdict status. An MSC based validator tool has been designed and developed within a joint project between Nokia Research Center and Fraunhofer FOKUS.
The validator is designed to run against any test system that provides some basic functionality, like starting of a test case, retrieving the status of the final verdict, sending and receiving messages, etc. The basic idea was to create a validator that is not only able to validate the abstract test suite but also a real test system (tester), i.e., an abstract test suite plus its execution environment plus the glue that is necessary to tie the test suite to the actual system under test. Since this glue can be of considerable complexity, e.g., consisting of implementations of various protocol stacks, message en- and decoders, possibly tailored hardware, etc., testing of the whole test system is indeed an important aspect.

This is also one of the advantages of our approach as compared to other approaches like an isolated verification of the abstract test suite or an automatic generation of test cases from test purposes.

Given a sufficiently detailed specification of the test purpose, a combination of automatic generation of test cases [14] from the test purpose together with a validation following our approach seems optimal. The validation guarantees both the correctness of the implementation of the generation algorithm and of the additional components that make up the test system.

The design of the validator aims to make it as independent of the test system as possible by defining a small, well-defined interface to connect the validator to the test system. In our case study we have used a TTCN-3 test system with the MSC validator. The validator accesses the test system at its (proprietary) control interface to trigger the execution of test cases and retrieve the final verdict. It uses TTCN-3’s standardized communication interface toward the SUT [13] to exchange messages with the test system. The MSC validator has been implemented using JAVA and the test system runs independently of the validator. Although not all work within this project has been completely finished, results so far show that using MSCs as test purpose definition language and as basis for the test case validation can improve the quality of test cases and thus the quality of system implementations.

5 Future Work

This paper defines a novel approach to test case validation and provides the necessary theoretical background. Yet, it is only a first step toward a working test case validation system. In particular, the following issues need to be addressed in the future:

Algorithms and Complexity. Deciding well-formedness of an MSC $M$ so far requires the calculation of the semantics $X_M$, which is a costly operation. A syntactic characterization of well-formedness would be desirable because it would probably allow for faster tests for well-formedness that could, e.g., also be built into an MSC editor to support test purpose development by pointing out problematic constructs. Additionally, a detailed analysis of the complexity of well-formedness and test case validity would be desirable.

Data. Since its last revision, data is an integral part of the MSC language. An extension of our approach that also takes into account data passed in messages is essential.
for the practical applicability of our approach to a wider class of test cases. While this should not impose any theoretical problems, it will be a challenge to integrate data into the implementation in a user-friendly manner.

**Time.** MSCs allow to express various timing constraints and timing aspects are important in many testing efforts. Therefore, we plan to extend our approach to MSCs with timing constraints. From a theoretical point of view, this is probably the most interesting way to continue the work presented in this paper.

## 6 Appendix

This appendix contains the proofs of Theorems 1 and 2 (for technical reason in reverse order). In the following, let $\mathcal{M}$ denote a well-formed test purpose.

From property [WF] we get that the function $\langle \cdot \rangle : \text{obs}(X_M) \to X_M$ that maps every $\sigma \in \text{obs}(X_M)$ to a $\overline{x_\sigma} \in X_M$ such that $\sigma \in \text{lin}(\overline{x_\sigma})$ is in fact a well-defined and total. It is easy to show the following property:

**Lemma 3** Let $\mathcal{M}$ be a well-formed test purpose and $\rho, \sigma \in \text{obs}(X_M)$ with $\rho \leq \sigma$. Then $\langle \rho \rangle_M \leq \langle \sigma \rangle_M$.

Let $\mathcal{M}$ be a well-formed test purpose, $T$ a test case for $\mathcal{M}$ and $\sigma \in L_M$ with $\nu_M(\sigma) \neq \text{none}$. A validation for $\sigma$ is a complete run $\rho_0 \overset{T}{\rightarrow} \rho_1 \overset{T}{\rightarrow} \cdots \overset{T}{\rightarrow} \rho_n$ such that $\rho_n \simeq \rho_0 \mathcal{M} \sigma$ and $\nu_T(\rho_n) = \nu_M(\sigma)$.

It is easy to see that validity of a test case w. r. t. a test purpose can equivalently be formulated as follows.

**Lemma 4** Let $\mathcal{M}$ be a well-formed test purpose and $T$ a test case for $\mathcal{M}$. Then $T$ is valid w. r. t. $\mathcal{M}$ iff every $\sigma \in L_M$ has a validation.

We will need the following technical lemma:

**Lemma 5** Let $\mathcal{M}$ be a well-formed test purpose, $a, b \in \Sigma_o$ be actions, and $\rho, \sigma \in \Sigma_o^*$. Moreover, assume $\rho a, \rho b \in L_M$, $\nu_M(\sigma) \neq \text{none}$, and $\rho b \leq \sigma$. If $a \notin \Sigma_o$ or $b \notin \Sigma_o$ (or both), then there exists a $\sigma' \in L_M$ with $\sigma \simeq L_M \sigma'$, $\nu_M(\sigma) = \nu_M(\sigma')$, $\rho a \leq \sigma'$, and $\nu_M(\sigma) = \nu_M(\sigma')$.

**Proof 1** Let $a, b, \rho, \sigma$ as required by the lemma and let $x = \langle \rho \rangle_M$, and $z = \langle \sigma \rangle_M$. Let $y_a, y_b \in X_M \upharpoonright \Sigma_o$ such that $y_a = x \circ_c a$ and $y_b = x \circ_c b$. From [WF] we get $x \leq y_b \leq z$. From [WF] also $x \leq y_a \leq z$ holds. Hence, there exists $u \in X_M \upharpoonright \Sigma_o$ with $u = x \circ_a \circ \rho \circ b$ and $u \leq z$ and we obtain $\sigma'$ setting $\sigma' = \rho a n$, where $n$ is the string that can be appended to $\rho b$ to obtain $\sigma$ with the first occurrence of $a$ deleted. From what have said before, $\sigma'$ is a linearization of $z$, hence $\sigma \simeq L_M \sigma'$ and $\nu_M(\sigma) = \nu_M(\sigma')$.

**Proof 2 (Proof of Theorem 2)** Assume that valid$(\mathcal{M}, T, \varepsilon)$ does not fail and let $\sigma \in L_M$ with $\nu_M(\sigma) \neq \text{none}$ and $|\sigma| = n$. By Lemma 4 it suffices to show that there exists a validation of $\sigma$. To this purpose, we will construct sequences $\rho_0, \ldots, \rho_n$ and

15
σ₀, . . . , σₙ such that |ρᵢ| = i, valid(M, T, ρᵢ) is called during the execution of the algorithm, ρᵢ ≤ σᵢ, σᵢ ∼ₗₘ σ, and vₘ(σ) = vₘ(σᵢ), for each 0 ≤ i ≤ n.

We start with ρ₀ = ε and σᵢ = σ, which satisfies all the required properties. Assume that the sequences have been constructed up to i. Since valid(M, T, ρᵢ) does not fail, ρᵢ ∈ Lₘ holds and there are the following possibilities:

- T(ρ) ∈ Vᵢ and T(ρ) = vₘ(ρ). In this case, |ρ| = n must hold because otherwise vₘ(σᵢ) = none and vₘ(ρᵢ) = none, together with ρᵢ < σᵢ, would be (by Lemma 3) a contradiction to the fact that vₘ is a verdict assignment on Lₘ.
- T(ρ) = δ ∨ eₙ(M, ρ) ∩ Σ₀ⁿ = ∅. Then i < n must hold and since ρᵢ < σᵢ, there exists b ∈ Eₗ(T) such that ρ₁b ≤ σᵢ. If b ∈ Σ₀ⁿ then there will be a call valid(M, T, ρ₁b) and we set ρᵢ₊₁ = df ρ₁b and σᵢ₊₁ = df σᵢ to continue the sequences. Clearly, this satisfies all necessary properties. If b ∈ Σ₀ⁿ then let a ∈ eₙ(M, ρᵢ) ∩ Σ₀ⁿ. Then a, b, ρᵢ, σᵢ satisfy the prerequisites of Lemma 3 which yields the existence of σᵢ₊₁ ∈ Lₘ with σᵢ ∼ₗₘ σᵢ₊₁, vₘ(σᵢ) = vₘ(σᵢ₊₁), and ρᵢ₊₁ = df σᵢ₊₁ then we have extended the sequence as required.

- T(ρ) = a ∈ Σ₀ⁿ & ρ₀ ∈ Lₘ. In this case we necessarily have to set ρᵢ₊₁ = df ρᵢa and we need to show the existence of a suitable σᵢ₊₁. This can be done similar to the previous case using Lemma 3.

It is easy to see that T(ρₙ) ∈ Vᵢ and by construction it holds that ρₙ ∼ₗₘ σ as well as vₗ(ρₙ) = vₘ(σ). Moreover, obviously ρ₀ →ₗ ρ₁ →ₗ . . . →ₗ ρₙ is complete run and thus we have found the desired validation for σ.

For the converse direction, let T be a test case that is valid w. r. t. M. We need to show that the call valid(M, T, ε) does not fail. Hence assume that is does fail and let ρ ∈ Lₘ a prefix-maximal word such that valid(M, T, ρ) is evaluated. By definition valid, ρ ∈ Lₘ must holds. One of the following choices for ρ is the one that leads to failure.

- T(ρ) is undefined, then obviously, for every σ ∈ Lₘ with ρ ≤ σ, [σ]ₗₘ ∩ Lₜ = ∅.
- T(ρ) ∈ Vᵢ and T(ρ) = vₘ(ρ), which violates the first condition in the definition of test case validity.
- T(ρ) = δ and eₙ(M, ρ) ∩ Σ₀ⁿ = ∅. If vₘ(ρ) = none then Lₜ and Lₘ, then again the first condition of the definition of test case validity is violated. If vₘ(ρ) = none then there exists σ ∈ Lₘ with vₘ(σ) = none and ρ < σ. It is easy to see that, if there exists η ∈ [σ]ₗₘ ∩ Lₜ, then ρ < η, but since T(ρ) = δ and eₙ(M, ρ) ∩ Σ₀ⁿ = ∅, for every χ ∈ Lₜ with ρ < χ, χ ∈ Lₘ holds and hence the does not exists such a η.

The case that T(ρ) = a ∈ Σ₀ⁿ but ρ · a ∈ Lₘ is analog to the previous case.

Hence, valid(M, T, ε) cannot fail.
Proof 3 (Proof of Theorem) We define a test case $T$ as follows: for $\sigma \in \text{obs}(X_M)$:

$$T(\sigma) \overset{\text{def}}{=} \begin{cases} 
\nu_M(\langle x \rangle_M), & \text{if } \text{en}(M, \sigma) = \emptyset; \\
\delta, & \text{if } \emptyset \neq \text{en}(M, \sigma) \subseteq \Sigma_\omega^o \\
a, & \text{for an arbitrary } a \in \text{en}(M, \sigma) \cap \Sigma_\omega^i \text{ otherwise}
\end{cases}$$

It is easy to see that for this test case $T$, $\text{valid}(M, T, \varepsilon)$ does indeed not fail and hence, by Theorem 2, $T$ is a valid test case w. r. t. $M$.

References

[1] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In E. Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of Lecture Notes in Computer Science, pages 259–274. Springer Verlag, 1997.

[2] P. Deussen. Concurrent automata. Technical Report 1-05/1998, Brandenburg Tech. Univ. Cottbus, 1998.

[3] ETSI. The tree and tabular combined notation version 3; part 1: TTCN-3 core language. Technical Report ES 201 873-1, ETSI, 2001.

[4] J. Grabowski. On partial languages. *Fundamenta Informaticae*, 4(2):427–498, 1981.

[5] J. Grabowski and D. Hogrefe. TTCN SDL- and MSC-based specification and automated test case generation for inap. In *Proceedings of the 8th International Conference on Telecommunication Systems (ICTS’2000) - Modeling and Analysis*, Nashville, March 2000.

[6] ITU-T. ITU-T recommendation z.120 message sequence chart (MSC). Technical report, ITU-T, 1999.

[7] B. Jonsson and G. Padilla. An execution semantics of MSC-2000. In R. Reed and J. Reed, editors, *Proceedings of the 10th Internation SDL-Forum*, number 2078 in Lecture Notes in Computer Science. Springer Verlag, 2001.

[8] J.-P. Katoen and L. Lambert. Pomsets for message sequence charts. In *Proc. of 1st Workshop of the SDL Forum Society on SDL and MSC*, SAM98, Berlin, 1998.

[9] S. Mauw and M. A. Reniers. Operational semantics for MSC’96. In A. Cavalli and D. Vincent, editors, *Tutorials of the Eighth SDL Forum SDL’97: Time for Testing - SDL, MSC and Trends*, pages 135–152, Evry, France, 1997. Institut national des télcommunications.

[10] A. Mazurkiewicz. Introduction to trace theory. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 1, pages 3 – 42. World Scientific, Singapore — New Jersey — London — Hong Kong, 1995.
[11] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[12] R. Scheurer, J. Grabowski, and D. Hogrefe. Revised comparison of an automatically generated and a manually specified test suite for the B-ISDN protocol SS-COP. In H. König and P. Langendörfer, editors, *FBT’98 - Formale Beschreibungstechniken für verteilte Systeme*. Shaker Verlag, Aachen, 1998. Available online from [http://www.itm.mw-luebeck.de/](http://www.itm.mw-luebeck.de/).

[13] S. Schhulz and T. Vassiliou-Gioles. Implementation of TTCN-3 test systems using the TRI. In I. Schieferdecker, K. H, and A. Wolisz, editors, *Testing of Communication Systems XIV, Proc. of TestCom-2002*, pages 425–442, Berlin, Germany, 2002. Kluwer Academic Publishers.

[14] M. Schmitt, M. Ebner, and J. Grabowski. Test generation with autolink and testcomposer. In *Proceedings of the 2nd Workshop of the SDL Forum Society on SDL and MSC (SAM’2000)*. SDL Forum Society, 2000. Proceedings available online from [http://www.lrisa.fr/manifestations/2000/sam2000/](http://www.lrisa.fr/manifestations/2000/sam2000/).