A Framework for Interoperability Testing of Network Protocols

J. Alilovic-Curgus and S.T. Vuong

Computer Science Department
University of British Columbia
Vancouver, B.C., Canada, V6T 1Z2

Abstract

In this paper, we extend the testing theory based on formal specifications by formalizing testing for interoperability with a new relation intoP. Intuitively, P intoP Q if, for every event offered by either P or Q, the concurrent execution of P and Q will be able to proceed with the traces in S, where S is their (common) specification. This theory is applicable to formal description methods that allow a semantic interpretation of specifications in terms of labelled transition systems. Some existing notions of implementation relations in protocol testing theory are placed in this framework and their discriminating power for identifying processes which will interoperate is examined. As an example, a subset of the ST-II protocol is formally specified and its possible implementations are shown to interoperate if each implementation satisfies the intoP relation with respect to S, the specification of the ST-II protocol (subset).

1 Introduction

In this paper a general mathematical framework is developed for reasoning about the interoperability of communicating systems. Interoperability is a pivotal notion within Open Distributed Processing (ODP) concept in general, and network protocols in particular. If we assume that a (formal) abstract specification of a communicating system is available, then the interoperability of its different implementations is dependent on the design decisions taken during the implementation process leading from the specification to an executable implementation. Viewed this way, the question of interoperability of communicating systems is closely related to the formalization of the notion of validity, i.e., the relation which should hold between the implementation of a system and its specification.

This relation has been a subject of extensive research, especially within process algebraic techniques. Examples are conf and red [2, 6] defined for conformance testing of communication protocols and distributed systems. We show that the proposed relations are too strict if the interoperability of implementations is desired. Moreover, the practical testers designed to test with these relations as target criteria may be difficult to design and run, leading to inconclusive test runs for tests where these relations impose too strict validity requirements. We show that the question of test selection turns out to be impossible to solve ade-

quately within the proposed theories themselves.

We then introduce our work by formally defining what is meant by interoperability of communicating systems. We propose a new intoP relation which is defined to hold between an implementation and a specification of a system. It insures that implementations which are valid in intoP sense can interoperate with other such implementations of the same specification. This relation improves the efficacy of the test selection process by distinguishing the traces which are possibly unexecutable and those that are essential for basic interoperability, without the need to step outside the theory for a solution. In particular, it is a tighter upper bound to the testing process than the previously defined relations, since the number of inconclusive test runs is reduced.

This result is significant for designing testers for systems which are capable of running independent concurrent processes (e.g., simultaneous network connections). When it comes to testing modern network protocols such as the ST-II (multimedia) protocol [8], some features emerge that were not critical in testing classical protocols. For instance, even some basic functionality of an ST-II agent cannot be tested without at least three simultaneous connections. This is a problem in protocol test design (which typically deals with one connection only), which we feel is best solved by redefining the basic test relations to better suit modern protocol design.

In the rest of the paper, we assume that protocol specifications are given in process algebraic form (e.g. its standardized derivative LOTOS) [7, 5]. The inference rules for basic LOTOS combinators used in this paper are given in Appendix 2. All results in this paper also hold for any specification technique that allows Labelled Transition Systems (LTS) as their underlying semantic model and use interleaving semantics of concurrency.

2 The interoperability of protocol implementations

In this section we first define the well-known implementation relations conf and red. We evaluate these relations informally with respect to the level of interoperability achievable between implementations which are valid in the sense of these relations. The notation needed for the trace-refusal formalism used in the rest
of the paper can be found in Appendix 1.

2.1 Implementation relations \( \text{conf} \) and \( \text{red} \)

Consider the following example.

**Example 2.1**

\[ S := \text{EstStreamReq}(T_1, T_2); \]
\[ (SConnect_{T_1}; SConnect_{T_2}; S') \]
\[ \] (\[ SRefuse_{T_2} ; SAccept_{T_1} ; S'' \] )
\[ \]
\[ S' := (SAccept_{T_1} ; SRefuse_{T_2} ; S'') \]
\[ \]
\[ (SRefuse_{T_2} ; SAccept_{T_1} ; S''') \]
\[ \]
\[ S'' := (\text{ConRejInd}_{T_1} ; \text{ConRejInd}_{T_2}; D) \]
\[ \]
\[ (\text{ConRejInd}_{T_2} ; \text{ConRejInd}_{T_1}; D) \]

The above process \( S \) is an excerpt from the connection establishment phase of the ST-II multimedia internet protocol \([8]\) (for the origin agent only). The ST-II protocol entity at an origin may receive a stimulus in the form of a stream establishment request from an application above, for setting up connections with a number of targets (e.g., \( T_1 \) and \( T_2 \)). We have instantiated one possible situation (where one target accepts and the other target rejects the connection), resolved the parallel composition of two simultaneous connections into a simpler choice (\( \square \)) structure, and shown some of the event interleavings only. The behaviour tree of this process is given in Figure 1. (Figure 2 represents an ST-II agent as a target agent, again showing a simple instantiated case).

![Figure 1: Connection Establishment Phase of ST-II protocol - ORIGIN](image1)

Let \( P_1 \) and \( P_2 \) be processes.

![Figure 2: Connection Establishment Phase of ST-II protocol - TARGET](image2)

*This branch evolves in a manner similar to the subtree starting at \( TConnect_{T_1} \)*

**Definition 2.1** \( P_1 \text{ conf } P_2 \) iff \( \forall \sigma \in Tr(P_2) \) we have \( Ref(P_1, \sigma) \subseteq Ref(P_2, \sigma) \).

Informally, \( P_1 \text{ conf } P_2 \) iff, placed in any environment whose traces are limited to those in \( P_2 \), \( P_1 \) cannot deadlock when \( P_2 \) cannot deadlock. This relation is defined in protocol testing theory for conformance testing of protocol implementations \([2, 6]\).

**Definition 2.2** \( P_1 \text{ red } P_2 \) iff

\((i) Tr(P_1) \subseteq Tr(P_2) \) and
\((ii) P_1 \text{ conf } P_2 \).

\( \text{red} \) is the reduction relation. It limits the traces of a "\( \text{red} \)"-valid process \( P_1 \) to those of \( P_2 \), but the deadlock property remains the same as in Def. 2.1.

Consider the relations \( \text{conf} \) and \( \text{red} \) on the example ST-II origin specification \( S \) and the sample implementations \( I_1 \) and \( I_2 \) of this specification, depicted in Figure 3 and Figure 4, resp.

**Example 2.2** The process \( I_1 \) (Figure 3)
\[ I_1 := EstStreamReq(T_1, T_2); \]
\[ (SConnect_{T_1}; SConnect_{T_2}; I'_1) \]
\[ \]
\[ (SConnect_{T_2}; SConnect_{T_1}; I'_2) \]
\[ I'_1 := (SAccept_{T_1}; SRefuse_{T_2}; S'') \]
\[ \text{is neither in relation \( \text{conf} \) nor \( \text{red} \) with the process (specification) \( S \). For instance, refusal set \( Ref(I_1, \sigma_1) \), where \( \sigma_1 = EstStreamReq(T_1, T_2) \). \( SConnect_{T_1}; SConnect_{T_2} \) includes also the set \( \{SRefuse_{T_2}\} \), which is not in the refusal set for the same trace in \( S \).} \]
Similarly, the process $I_2$ is neither in relation $\text{conf}$ nor $\text{red}$ with the process $S$. For instance, the refusal set $\text{Ref}(I_2, \sigma_2)$, where $\sigma_2 = \text{EstStreamReq}(T_1, T_2)$, $\text{SConnect}_1$, $\text{SConnect}_2$, $\text{SAccept}_1$, $\text{SRefuse}_2$, contains also the set $\{\text{ConConfInd}_T\}$, which is not in the refusal set for the same trace in the specification of the process $S$.

2.2 Implementation relations and interoperability

Are the implementations $I_1$ and $I_2$ equivalent with respect to their ability to interoperate, as origin agents, with other ST-II implementations? A brief informal investigation and some knowledge of the ST-II protocol show that the implementation $I_1$ may fail to interoperate with an implementation which implements all possible event exchanges specified by ST-II (such an implementation may be represented by the behaviour tree of Figure 1), if it is presented with the incoming event $\text{SRefuse}_2$ before the event $\text{SAccept}_1$, which are remote target responses to its connect messages. However, the implementation $I_2$ will always successfully interoperate with any other implementation which fully implements ST-II, the implementation choice in $I_2$ was to always inform the layer above about the rejection of connection first, which should be quite irrelevant to the upper layer so long as all connection establishment information gets delivered. Even more significantly, the implementation $I_3$ of the same specification $S$, depicted in Figure 5, also possesses the same ability to interoperate with full ST-II implementations under all circumstances, although it is not even trace equivalent to $S$ for very short traces. This implementation accepts all incoming ST-II events in all temporal orders specified by the specification $S$, but keeps only one of the alternative branches in its behaviour tree where it may make the choice what protocol primitive to evolve with first.

There are frequent situations in network protocols where the choice to drop some of the traces in an implementation of $S$ will affect the externally observable behaviour of the implementation but not its ability to successfully interoperate with other implementations of $S$. We quote some of those:

- one external stimuli (e.g. $\text{EstStreamReq}$) which generates multiple protocol events which may be arbitrarily interleaved (e.g., events $\text{SConnect}_1$, $\text{SConnect}_2$, $\ldots$, $\text{SConnect}_n$ in $I_3$)
- accumulation of external stimuli (e.g. $\text{SAccept}$ or $\text{SRefuse}$) which generates protocol events which may be arbitrarily temporally ordered, provided they individually satisfy timing correctness (e.g., events $\text{ConConfInd}$ and $\text{ConRejInd}$ in $I_2$)
- multiple independent network connections (represented by the arbitrary interleaving in the interleaving model of concurrency), where any one of many possible interleavings of certain events may be sufficient for interoperability.

Note: Not all events initiated by the implementation under observation will be in one of these categories. Consider, for example, certain priority control PDUs or expedited data PDUs.

The formal theory of protocol testing based on observation fails to distinguish between different aspects of the inability of an implementation to progress via specific events. The consequence is that the inability
3 The interoperability relations

In this section we define a new formal notion of validity between an implementation and a specification of a communication system. Although the definition appears more involved compared to other implementation relations, it turns out that the test design based on this relation differs only slightly than the design based on other implementation relations, whereas the testing based on this relation is more efficient. We defer these practical considerations to the next section, and concentrate on theoretical comparison in this section.

3.1 The intop relation

We presuppose the existence of two subsets \( L_{req} \) and \( L_{alt} \) of labelsets of visible actions in \( L \), such that:

\[
L_{req} \cap L_{alt} = \emptyset \quad \text{and} \quad L_{req} \cup L_{alt} = L
\]

Intuitively, we shall think of the elements of \( L_{req} \) as events which must be observable at the interface of an implementation whenever the specification allows the possibility of synchronizing on that event in the state in which the implementation is at the moment of observation (i.e., the "required" synchronization). On the contrary, the elements of \( L_{alt} \) are such events, which may not necessarily be observable at the interface of an implementation, although the specification allows the possibility of synchronizing one such event at that point (i.e., the "alternative" synchronization).

Similarly, let \( Ref_{req}(P, \sigma) = Ref(P, \sigma) \cap P(L_{req}) \) and \( Ref_{alt}(P, \sigma) = Ref(P, \sigma) \cap P(L_{alt}) \) (where \( P(A) \) denotes the powerset (the set of all subsets) of a set \( A \)) be the \( \subseteq \)-maximal subsets of the refusal set \( Ref(P, \sigma) \) which are completely contained in the powersets of \( L_{req} \) and \( L_{alt} \), respectively.

Notice that these two sets always exist and are unique. Let \( I \) be an implementation of a specification \( S \).

Definition 3.1 \( I \) intop \( S \) iff \( \forall \sigma \in Tr(S) \cap Tr(I) \) we have:

1. \( Ref_{req}(I, \sigma) \subseteq Ref_{req}(S, \sigma) \), and
2. \( L_{alt} \cap Out(S, \sigma) \cap Out(I, \sigma) \neq \emptyset \lor L_{alt} \in Ref_{alt}(S, \sigma) \)
3. For \( A = L_{alt} \cap (Out(S, \sigma) \cap Out(I, \sigma)) \) we have

\[
Ref(I, \sigma) \setminus \{R \mid R \cap A \neq \emptyset\} \subseteq Ref(S, \sigma)
\]

Informally, \( I \) intop \( S \) iff, when placed in an environment whose traces are limited to the traces common to \( S \) and \( I \), (i) the implementation \( I \) cannot deadlock on any events from \( L_{req} \) on which \( S \) cannot deadlock; (ii) the implementation \( I \) cannot deadlock on all events from \( L_{alt} \) on which \( S \) cannot deadlock. Notice that the property 2. of the definition ensures that the deadlock property (ii) will be satisfied: an intop valid implementation \( I \) will always be able to evolve via at least one event from the set \( Out(I, \sigma) \) that is also in \( L_{alt} \cap Out(S, \sigma) \) (one alternative synchronization event), unless it may nondeterministically evolve.
to the state where no further interactions are possible (i.e., \( \text{Lalt} \in \text{Refalt}(S, \sigma) \)).

As an example, consider the validity of the implementation \( I \) in the sense of \( \text{intop} \) (it is easy to observe that \( I \) is neither in \( \text{conf} \) nor \( \text{red} \) relation with \( S \)).

**Example 3.1** Let \( \text{Lreq} = \{ \text{EstStreamReq}(T_1, T_2), S\text{Accept}_T, S\text{Refuse}_T \} \) and \( \text{Lalt} = \{ \text{SConnect}_{T_1}, \text{SConnect}_{T_2}, \text{ConfInd}_{T_1}, \text{ConRejInd}_{T_2} \} \). Consider the implementation \( I \) of \( S \), after the trace \( \sigma = \text{EstStreamReq}(T_1, T_2), \text{Refalt}(I, \sigma) = \mathcal{P}(\{ \text{EstStreamReq}(T_1, T_2), S\text{Accept}_T, S\text{Refuse}_T \}) \), \( \text{Refalt}(I, \sigma) = \mathcal{P}(\{ \text{SConnect}_{T_2}, \text{ConfInd}_{T_1}, \text{ConRejInd}_{T_2} \}) \).

Then,

1. \( \text{Refalt}(I, \sigma) \subseteq \mathcal{P}(\{ \text{EstStreamReq}(T_1, T_2), S\text{Accept}_T, S\text{Refuse}_T \}) = \text{Refalt}(S, \sigma) \), and
2. \( \text{SConnect}_T \in \text{Lalt}, \{ \text{SConnect}_{T_1} \} \in \text{Out}(S, \sigma) \) and \( \{ \text{SConnect}_{T_2} \} \in \text{Out}(I, \sigma) \), and
3. \( \text{Ref}(I, \sigma) = \mathcal{P}(\bigcup \{ R \in \text{Refalt}(I, \sigma) \}) \cup (\bigcup \{ R \in \text{Refalt}(I, \sigma) \}) \) and \( A = \{ \text{SConnect}_{T_2} \} \).

Therefore, \( \text{Ref}(I, \sigma) \setminus \{ R \mid R \in \{ \text{SConnect}_{T_2} \} \} \subseteq \text{Ref}(S, \sigma) \).

Therefore, the conditions of the Definition 3.1 are satisfied in the case of trace \( \sigma \).

### 3.2 Properties of the \( \text{intop} \) relation

We collect some easy facts about the \( \text{intop} \) relation, and its relationship to \( \text{conf} \).

**Proposition 3.1** Let \( I \) \( \text{intop} \) \( S \) and \( I = \sigma \Rightarrow \).

1. \( \text{Refalt}(I, \sigma) \subseteq \text{Refalt}(S, \sigma) \Rightarrow \text{Ref}(I, \sigma) \subseteq \text{Ref}(S, \sigma) \)
2. \( ( \forall \sigma \in \text{Tr}(S) \cap \text{Tr}(I), \text{Refalt}(I, \sigma) \subseteq \text{Refalt}(S, \sigma) \Rightarrow I \text{conf} S \).

3. \( I \text{conf} S \Rightarrow I \text{intop} S \) (i.e. \( \text{intop} \supseteq \text{conf} \))

4. \( \text{intop} \) is reflexive
5. \( \text{intop} \) is not transitive

**Proof:** The proof of this proposition can be found in [3].

### 3.3 The \( \text{intop}_\text{red} \) relation

The results of Proposition 3.1, 4, and 5, are in keeping with the theory developed in [6], that a valid implementation relation must be reflexive (because specification is a valid implementation of itself), but not necessarily symmetric or transitive (because the implementation and specification are not in general interchangeable). Notice that, similar to \( \text{conf} \), the relation \( \text{intop} \) allows additional traces to exist in the implementation, that are not part of the specification. This feature becomes even more critical in interoperability testing, since such implementations could synchronize on traces that are not in their common specification. For such traces, the concept of interoperability is really hard to define both formally and informally. We therefore extend the formal notion of interoperability by defining a relation, \( \text{intop}_\text{red} \), which restricts the traces in an implementation to those of the specification. Unlike \( \text{intop} \), this relation is also transitive.

**Definition 3.2** \( I \text{intop}_\text{red} S \) if and only if

1. \( \text{Tr}(I) \subseteq \text{Tr}(S) \)
2. \( I \text{intop} S \)

**Proposition 3.2** The relation \( \text{intop}_\text{red} \) has the following properties:

1. \( \text{intop} \supseteq \text{intop}_\text{red} \)
2. \( \text{intop}_\text{red} \supseteq \text{red} \)
3. \( \text{intop}_\text{red} \) is a preorder

**Proof:** The proofs for 1. and 2. are easy and follow directly from the definitions of the corresponding implementation relations. The proof for 3. is quite involved and can be found in [3].

The above theoretical considerations are sufficient to form a basis for specifying the architectural and design requirements in interoperability testing.

### 4 The interoperability tester design

After establishing the necessary theoretical basis in the previous section, we turn our attention towards some practical considerations in interoperability testing of network protocols. Based on these considerations, we outline the design of a network protocol interoperability tester whose theoretical upper bound is the satisfaction of the interoperability relation \( \text{intop} \) between an Implementation Under Test (IUT) and its specification \( S \).

#### 4.1 Architectural considerations

The general methodology of protocol conformance testing allows for different test architectures and different test interfaces. Consider the test architecture given in Figure 6. Generally, the Points of Control and Observation (PCOs) may be positioned at the upper IUT interface (PCO1, PCO4) as in the system SUT1 in Fig. 6, or at the lower IUT interface (PCO2, PCO3). For interoperability testing of protocol implementations it is necessary to observe both upper interface (service) PCOs and lower interface PCOs (as in SUT2 of Fig. 6.), in order to ensure the proper internetworking of different implementations in all environments. To take advantage of our theory, we model the interoperability test architecture in the following manner:

1. \( I \) is a protocol implementation
2. \( IT \) is the interoperability tester
3. NET is the underlying (network) connection between the I and IT. NET behaves as a reliable FIFO channel in either direction (FIFO without loss)\(^1\)

4. A tester IT is capable of observing at least one set of PCOs at the upper interface of I and one set of PCOs at the lower interface of I or IT.

5. A tester IT is capable of controlling PCO1 and either PCO2 or PCO3.

We will also assume that an executable tester (an implementation of the interoperability tester IT) is capable of executing strong control over the PCOs it controls in the following manner: it will always be able to synchronize on any events that are its output events, before synchronizing on any events that are its input. In particular, we expect that an executable tester is able to send a PDU into network or request a service from an IUT at any time. If this assumption holds, then the possibility of inconclusive test runs linked to the tester trying to observe a particular event in L_{reg} as the next event is eliminated.

4.2 Formal network protocol specification issues

The requirements of interoperability testing regarding the observability and controllability of PCOs dictate the formal specification style of a protocol process to be tested. In LOTOS, this style requires that gates modelling the PCO1 and one of PCO2 or PCO3 not be hidden (i.e., event synchronization at these gates is visible). The observability of PCO4 is entirely optional and depends on the executable tester design. We require that all the protocol processes be fully synchronized with the underlying process NET representing the network.

For the purpose of illustration, we complete our example ST-II specification of the stream establishment phase with the specification of the target process. We simplify the specification in the manner similar to Example 2.1. (We omit the details of the negotiation and simply let the ST-II target agent decide on acceptance or refusal of the connection by itself.)

**Example 4.1** The target ST-II agent T is the process

\[
T := (TConnect_{T1}; TAcept_{T1}; D) \\
(TConnect_{T2}; TRefuse_{T2})
\]

The full specification of our example ST-II process is the independent parallel composition of the processes S and T:

\[
ST := S ||| T
\]

and is represented as a parallel composition of the behaviour trees given in Figure 1 and Figure 2.

The specification of the NET FIFO process can be given as in [4]. For the purposes of our brief example we will informally observe that for every event e prefixed by T, i.e. event Te on which the NET process synchronizes at interaction points PCO2 (PCO3), it will subsequently synchronize on an event Se at PCO3 (PCO2 respectively) distinguishable from the event Te by its prefix S only, after which it is ready for a new interaction at PCO3 or PCO2. Similarly, if the NET process synchronizes on an Se event at the interaction points PCO2 (PCO3) first, then this is followed by a synchronization on a Te event at PCO3 (PCO2) and the process NET is ready for a new interaction. Notice that the parallel composition of the processes ST and NET with all gates observable, will yield exactly the traces of our full example ST-II specification. The following example illustrates this behaviour.

**Example 4.2** Let EstStreamReq(T1, T2) be applied at the PCO1, and assume that the additional revealed PCOs are PCO2, PCO3 and PCO4 (the system specified is exactly SUT2 in Fig. 6). Then the following trace may be observable at these PCOs:

\[
\sigma = \text{EstStreamReq}(T1, T2), \text{SCconnec}_{T1}, \text{SCconnec}_{T2}, TConnect_{T1}, TConnect_{T2}, TAcept_{T1}, TRefuse_{T2}, SAcept_{T1}, SRefuse_{T2}, ConRejInd_{T2}, ConConfInd_{T1}
\]

4.3 Interoperability tester design

In this section we introduce the design of the interoperability tester IT(S) for protocol implementations. The purpose of the interoperability tester is to properly distinguish between implementations that do or do not satisfy the intop relation with respect to their specification S.

The construction of the interoperability tester IT(S) is based on the canonical tester [2, 6]. The canonical tester T(S) is constructed systematically from the specification of the system S and is defined in the following manner.

**Definition 4.1** Let S be a specification. The canonical tester of S, T(S), is defined implicitly as a solution X satisfying the following two equations:
1. $Tr(X) = Tr(S)$
2. $\forall i \mapsto conf\ S \iff (\forall \sigma \in L^* \text{ we have } (L \in Ref(f \mid X, \sigma) \Rightarrow L \in Ref(X, \sigma)))$

We first observe that by the Proposition 3.1, 3, $intop \supset conf$. Using this observation, we can transfer the problem of finding the interoperability tester $IT(S)$ to “relaxing” the structure of the canonical tester $T(S)$. We here do not repeat the theoretical work of [2, 6], but assume that the canonical tester $T(S)$ of a specification $S$ is given. The crucial observation in the construction of the $IT(S)$ is that the only behaviours that are treated differently in the relations $conf$ and $intop$ are precisely the ones that allow the choice over actions that are in $L_{alt}$.

In the canonical tester, the choice over the different actions in $L$ is replaced by the internal choice, i.e. these actions are prefixed by the internal action $i$. Consider the example given in Figure 7, where $S$, $T$ and $IT$ denote the specification, its canonical tester and its interoperability tester. In the derivation of the

choices of $T(S)$, each followed by one of the $n$ different actions in $A \subseteq L_{alt}$ by one internal choice in $IT(S)$ followed by the external choice over the $n$ different actions in $A \subseteq L_{alt}$. Therefore, $T(S) := i; a[i]; b[i]; c$ in the example becomes $IT(S) := i; a[i]; b[i]; c$. The interoperability tester will therefore resolve such a choice in the course of the interaction with the environment (for example, driven by the choice of the peer protocol implementation), rather than by attempting to synchronize on each one of the actions. More specifically, if a node in the synchronization tree of the canonical tester $T(S)$ has the form

$$\exists \{a_i; \ldots \mid i \in P\} \{i; c_i; \ldots \mid q \in Q\}$$

then it is transformed into a node of the synchronization tree of the interoperability tester $IT(S)$ of the form

$$\exists \{a_i; \ldots \mid i \in P\} \{i; c_i; \ldots \mid b \in R \subseteq Q\} \{i; b_a; \ldots \mid b \in A \subseteq Q\}$$

where $R$ is the set of all $q \in Q$ such that $b \in L_{req}$ and $A$ is the set of all $q \in Q$ such that $b \in L_{alt}$.

All other nodes and branches of $T(S)$ are left intact.

We observe that the execution of $IT(S)$ against an implementation $I$ of a protocol will have the following impact on the testing process:

- Fewer traces need to be examined or observed (even if they happen to be implemented), resulting in a more efficient upper bound of the testing process.

From the construction of the tester and the definition of the relation $intop$ it follows that, if a branch labelled $i$ leads to a node whose all emanating branches are labelled with the events in $A \subseteq L_{alt}$, then the number of tests is reduced from the number of events in $A$ to a subtree which is to be considered as one test case only.

- Elimination of inconclusive runs for test cases where one of many possible temporal orderings of events is sufficient to guarantee the interoperability of implementations.

This observation follows directly from the fact that, for such events, the multiple internal choices of the canonical tester are substituted with one choice which is always possible to resolve on interaction of the interoperability tester with the environment.

- A selection process can be designed within the theory, which will guarantee not to sacrifice traces needed for interoperability in favour of possibly unobservable traces.

All the events that can happen are collected under the nodes which have all emanating branches labelled with the events in $L_{alt}$ and the branches leading to such nodes are labelled $i$. Such nodes are uniquely distinguishable and should participate in the test selection with the weight representative of one test case only.
5 Conclusion

In this paper we have extended the formal theory of testing protocol implementations by a new relation, and proposed a corresponding test architecture, specification style and tester design. The new framework is aimed at simplifying the practical testing and consequently our considerations are often targeted more towards applicability than rigorous theory. Further work includes lifting the restriction that the sets of required and alternative synchronization events must be disjoint, and extending the approach to the interoperability of protocol implementations that do not necessarily share the same specification. The framework could benefit both from including more strict theoretical results (especially along the results in [6]) as well as more efficient algorithmic solutions of the interoperability tester derivation. For truly rigorous testing of modern network protocols in their full multi-connection capacity, both ingredients are essential.

References


Appendix 1: Notation

Processes (denoted by T, and ranged over T1, T2,...) will be sets of labelled transition systems over an alphabet L U {i} (i is the unobservable action) of elementary actions.

\[ P = a \rightarrow P' \] means that process P may engage in an action \( a \in L \) and, after doing so, behaves like process \( P' \).

\[ P - i^k \rightarrow P' \] means that process P may engage in the sequence of \( k \) internal actions and, after doing so, behaves like process \( P' \).

\[ P - a \rightarrow P' \] means \( P - a \rightarrow P'' \) and \( P'' - b \rightarrow P' \).

\[ P = a \Rightarrow P' \] means \( \exists k, k_1 \in N \) such that

\[ P - i^k, a, i^{k_1} \rightarrow P' \]

\[ P = a \Rightarrow \text{means } \exists P' \text{ such that } P = a \Rightarrow P', \text{ i.e. } P \text{ may engage in an action } a \]

\[ P \neq a \Rightarrow \text{ means } \neg(P = a \Rightarrow) \text{ i.e. } P \text{ cannot engage in an action } a \]

\[ P = \sigma \Rightarrow P' \] means that process P may engage in a sequence of observable actions \( \sigma \) and, after doing so, behaves like process \( P' \).

\[ P = \sigma \Rightarrow \text{ means that } \exists P' \text{ such that } P = \sigma \Rightarrow P', \text{ i.e. } \{\sigma \mid P = \sigma \Rightarrow \}; \text{Tr}(P) \]

\[ P \text{ after } \sigma = \{P' \mid P = \sigma \Rightarrow P'\}, \text{ i.e. the set of all behaviour expressions (or states) reachable by } \sigma \]

\[ \text{Out}(P, \sigma) \text{ is the set of possible observable actions after the trace } \sigma, \text{Out}(P, \sigma) = \{a \in L \mid \sigma.a \in \text{Tr}(P)\}\]

\[ \text{Ref}(P, \sigma) \text{ is the refusal set of } P \text{ after trace } \sigma, \text{ i.e. } \text{Ref}(P, \sigma) = \{X \in L \mid \exists P' \in P \text{ after } \sigma, \text{ such that } P' \neq a \Rightarrow \forall a \in X \}\]

Appendix 2: LOTOS

<table>
<thead>
<tr>
<th>Comb.</th>
<th>Axioms or Inference Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>none</td>
</tr>
<tr>
<td>( m; B )</td>
<td>( m; B \rightarrow m \rightarrow B )</td>
</tr>
<tr>
<td>( i; B )</td>
<td>( i; B \rightarrow i \rightarrow B )</td>
</tr>
<tr>
<td>( B_1 \parallel B_2 )</td>
<td>( B_1 \rightarrow B_1' \lor \lnot B_1 \parallel B_2 \rightarrow m \rightarrow B_1 )</td>
</tr>
<tr>
<td>or ( B_2 \rightarrow m \rightarrow B_2' \lor \lnot B_1 \parallel B_2 \rightarrow m \rightarrow B_2 )</td>
<td></td>
</tr>
</tbody>
</table>
| \( B_1 \parallel [g] B_2 \) | \( B_1 \rightarrow B_1', \text{name}(m) \notin \{g\} \rightarrow \)
| \( \lnot B_1 \parallel \lnot B_2 \rightarrow m \rightarrow B_1' \parallel [g] B_2 \) |
| or \( B_2 \rightarrow m \rightarrow B_2', \text{name}(m) \notin \{g\} \rightarrow \)
| \( \lnot B_1 \parallel \lnot B_2 \rightarrow m \rightarrow B_1' \parallel [g] B_2 \) |
| \( B_1 \parallel B_2 \) | \( B_1 \rightarrow B_1', B_2 \rightarrow m \rightarrow B_2' \rightarrow \)
| \( B_1 \parallel B_2 \rightarrow B_1' \parallel [g] B_2 \) |
| \( B_1 \parallel B_2 \) | \( B_1 \rightarrow B_1', B_2 \rightarrow m \rightarrow B_2' \rightarrow \)
| \( B_1 \parallel B_2 \rightarrow B_1' \parallel [g] B_2 \) |

Table 1: Axioms and Inference Rules in LOTOS