A Framework for Interoperability Testing of Network Protocols

by

Jadranka Alilovic-Curgus
Son T. Vuong

Technical Report No. 93-11
April 1993

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

A Framework for Interoperability Testing of Network Protocols

Abstract

In this report, we extend the testing theory based on formal specifications by formalizing testing for interoperability with a new relation *intop*. Intuitively, P $intop_S$ 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. Existing notions of implementation preorders and equivalences 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 work 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 nature of the relation which should hold between the implementation of a system and its (formal) specification.

This formal relation has been a subject of extensive research (especially within process algebraic techniques), and the results can broadly be summarized in two categories.

Equivalence Relations

Extensional equivalences play a central role in reasoning about the external behaviour of systems described by process algebraic languages (see, for example [Mil80, ?]). Two processes are considered equivalent in this context if they cannot be distinguished by external observation, i.e. short of taking them apart. Examples of these equivalences are observation equivalence [Mil80] or various testing equivalences [dN87].

Certain testing theories are naturally based on this notion of indistinguishability by observation. Testing based on equivalence relations of this kind ensures that the implementation will behave exactly as prescribed by specification. For testing communication protocols, equivalence relations *conf-eq* [Led92] and *te* [Bri88] have been defined and proposed as a criterion which establishes that a certain protocol implementation is a valid implementation of a given specification.

Implementation Relations

Implementation relations [Led92] allow the notion of validity between an implementation and its specification to be extended to those implementations which are not externally equivalent to their specification. These relations have been less studied within the process algebraic techniques than equivalences.

Some implementation relations based on the idea of reducing nondeterminism as a valid implementation choice have been proposed so far for the use in testing. Examples are *conf, conf*, red, ext* [Bri88, Led92] defined for communication protocols and distributed systems in the context of conformance testing.

We will briefly study both of these categories of relations on their representatives from the area of communication protocols testing. We show that all of 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, typically generating many inconclusive test runs for test cases where these relations impose too strict requirements (see also [Bri88]). In particular, we show that the question of test selection turns out to be impossible to solve adequatly within the proposed theories themselves.

We then introduce our work by formally defining what is meant by interoperability of (two or more) 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 unexecutable and those that are essential for basic interoperability, without the need to step outside the theory for an adequate solution. In particular, it is a more efficient upper bound to the testing process than the previously defined relations, since the number of inconclusive test runs is significantly reduced.

This result is particularly significant for designing testers for systems which are capable of running independent concurrent processes (i.e., for testing simultaneous network connections). When it comes to testing modern network protocols such as ST-II (multimedia) communication protocol [Top90], some features emerge (shown on the example ST-II specification throughout the report), that were not critical in testing classical protocols usually considered in the protocol test theory. For instance, even some basic functionality of an ST-II agent cannot be tested without testing at least three simultaneous connections. This then poses a problem in protocol test design (which typically deals with one connection or linear test sequences), which we feel is best solved by redefining the basic test relations to better suit the design of modern protocols. At the same time, the new testing framework greatly facilitates test development and tester design for multiple simultaneous network connections, paving the way to meeting stricter reliability requirements for tested communication systems.

We assume that formal specifications are given in process algebraic form, and we draw most of the examples and comparisons from the testing theory for process algebras and, wherever possible, LOTOS (a brief overview to be found in the APPENDIX 1.). Note, however, that this setting primarily serves to anchor our discussion, and that all of the results can be derived for formal specification techniques that allow Labelled Transition Systems (LTS) as their underlying semantic model and use interleaving semantics of concurrency. From now on we make no difference between a labelled transition system and a process. In addition, we will often use the technique detailed in [Mil80], to represent processes in terms of Synchronization Trees (STs).

2 Implementation relations and interoperability

In this section we will define and briefly study the relations *conf* and *red*. We then evaluate these relations informally with respect to the level of interoperability achievable between implementations which are valid in the sense of these relations. Our results apply to other relations mentioned in the introduction in much the same way. The notation needed for the trace-refusal formalism used in the rest of the report can be found in the APPENDIX 2.

2.1 Implementation Relations $conf$ and red

Let P_1 and P_2 be processes, and L the set of observable actions.

The most straight forward proposal for systems equivalence is to consider as equivalent two systems which can perform exactly the same sequences of *visible* actions.

Definition 2.1 $P_1 \sim_s P_2$ *iff for all* $s \in L^*$: $P_1 = s \Rightarrow$ *if and only if* $P_2 = s \Rightarrow$.

It is clear that $P_1 \sim_s P_2$ iff $Tr(P_1) = Tr(P_2)$ and it is obvious that it is an equivalence relation, which we call *strings equivalence* [dN87].

Consider the following example.

Example 2.1 *Let S be a process*

$S := EstStreamReq_{\{T1,T2\}};$
$((SConnect_{T1}; SConnect_{T2}; S')$

$[]$
$(SConnect_{T2}; SConnect_{T1}; S'))$
$S' := (SAccept_{T1}; SRefuse_{T2}; S'')$
$[]$
$(SRefuse_{T2}; SAccept_{T1}; S'')$
$S'' := (ConConfInd_{T1}; ConRejInd_{T2}; D)$
$[]$
$(ConRejInd_{T2}; ConConfInd_{T1}; D)$

The above is an example of a process (an excerpt from a connection establishment phase of ST-II communication protocol [Top90], origin agent only). ST-II protocol at origin may receive a stimuli in the form of a (multimedia) stream establishment request from an application above, towards a number of targets ($T1$ and $T2$ in the example). For simplicity, 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 ([]) structure, and shown some of the events and interleavings only. The synchronization tree of this process is given in Figure 1 (a). (Figure 1 (b) represents an ST-II agent as a target agent, again showing a simple instantiated case). An implementation I_S which is strings equivalent to the specification S will have the following trace set, for traces of length ≤ 3, (ε is the empty trace)

$Tr(I_S) = \{\varepsilon,$
$EstStreamReq_{\{T1,T2\}},$
$EstStreamReq_{\{T1,T2\}}.SConnect_{T1},$
$EstStreamReq_{\{T1,T2\}}.SConnect_{T2},$
$EstStreamReq_{\{T1,T2\}}.SConnect_{T1}.SConnect_{T2},$
$EstStreamReq_{\{T1,T2\}}.SConnect_{T2}.SConnect_{T1}\}$

More involved concepts of equivalence between a specification and an implementation include the consideration of the traces that can be observed at the interface of a system with its environment as well as the sets of actions that may be refused after a certain trace. Such relations are *conf* and *red.*

Definition 2.2 P_1 *conf* P_2 iff $\forall \sigma \in Tr(P_2)$ we have $Ref(P_1, \sigma) \subseteq Ref(P_2, \sigma)$ or equivalently

P_1 *conf* P_2 iff $\forall \sigma \in Tr(P_2) \cap Tr(P_1)$ we have $Ref(P_1, \sigma) \subseteq Ref(P_2, \sigma)$.

Informally, P_1 *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 known as *conformance* relation in protocol testing theory and is used for conformance testing of protocol implementations [Bri88, Led92].

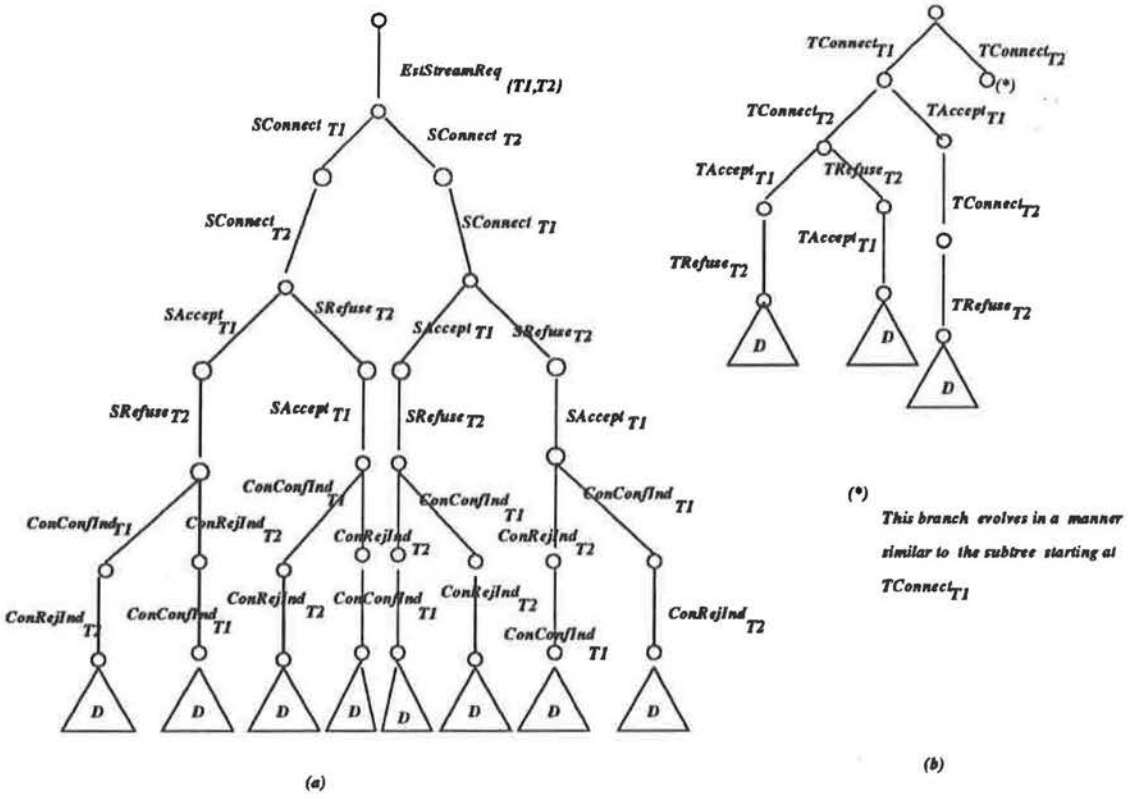

Figure 1: Connection Establishment Phase of ST-II protocol

Definition 2.3 P_1 *red* P_2 iff
$(i) Tr(P_1) \subseteq Tr(P_2)$ and
$(ii) P_1$ *conf* P_2.

Red is the reduction relation. It limits the traces of a "*red*"-valid process $P1$ to those of $P2$, but the essential deadlock property remains the same as in Def. 2.2.

These relations are the basis for the equivalence relations *conf-eq* and *te* [Led92, Bri88] proposed in protocol testing.

2.2 Implementation Relations and Interoperability

Consider the implementation relations defined above on the example ST-II origin specification S and the sample implementations $I1$ and $I2$ of this specification, depicted in Figure 2 (a) and (b).

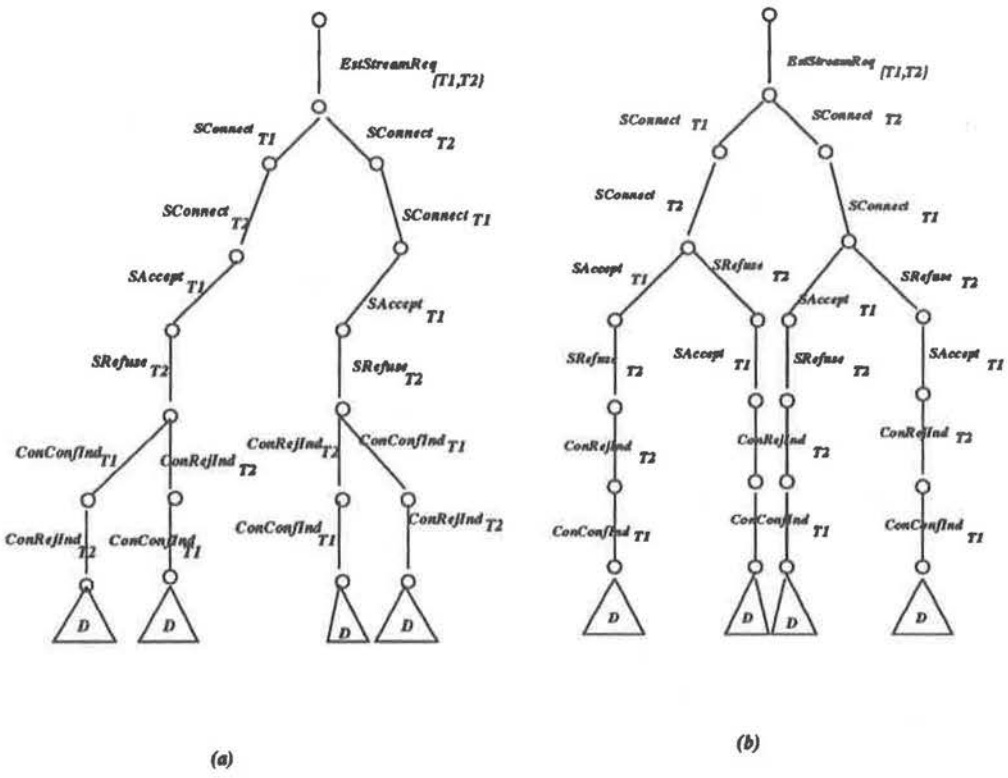

Figure 2: Different implementations of a process S

Example 2.2 *The process I1 (Figure 2(a))*

$$
\begin{aligned}
I1 := {} & EstStreamReq_{\{T1,T2\}}; \\
& ((SConnect_{T1}; SConnect_{T2}; I1') \\
& [] \\
& (SConnect_{T2}; SConnect_{T1}; I1')) \\
I1' := {} & (SAccept_{T1}; SRefuse_{T2}; S'')
\end{aligned}
$$

is neither in relation *conf* nor *red* with the process (specification) S. This is because the refusal set $Ref(I1, \sigma_1)$, where $\sigma_1 = EstStreamReq_{\{T1,T2\}}.\ SConnect_{T1}.\ SConnect_{T2}$ (or $\sigma_1' = EstStreamReq_{\{T1,T2\}}.\ SConnect_{T2}.\ SConnect_{T1}$) includes, among other sets, also the set $\{SRefuse_{T2}\}$, which is not in the refusal set for the same trace in the process (specification) S.

Example 2.3 *The process I2 (Figure 2 (b))*

$$
\begin{aligned}
I2 &:= EstStreamReq_{\{T1,T2\}}; \\
&\quad ((SConnect_{T1}; SConnect_{T2}; I2') \\
&\quad [] \\
&\quad (SConnect_{T2}; SConnect_{T1}; I2')) \\
I2' &:= (SAccept_{T1}; SRefuse_{T2}; I2'') \\
&\quad [] \\
&\quad (SRefuse_{T2}; SAccept_{T1}; I2'') \\
I2'' &:= ConRejInd_{T2}; ConConfInd_{T1}; D
\end{aligned}
$$

is neither in relation *conf* nor *red* with the process (specification) S. This is because the refusal set $Ref(I2, \sigma_2)$, where $\sigma_2 = EstStreamReq_{\{T1,T2\}}.\ SConnect_{T1}.\ SConnect_{T2}.\ SAccept_{T1}.\ SRefuse_{T2}$, contains, among other sets, also the set $\{ConConfInd_{T1}\}$, which is not in the refusal set for the same trace in the specification of the process S. (It is easy to see that there are three more traces which share the same characteristic with the trace σ_2.)

Notice also, that neither of the equivalences generated by these implementation relations holds.

This is because an observer (in the environment of these implementations) will be unable to observe some traces at the interface of these implementations, although they should be present in all implementations of S that are valid in the sense of *conf* and *red*. But are the implementations $I1$ and $I2$ equal with respect to their ability to interoperate, as source agents, with other ST-II implementations? A brief informal investigation (we defer the formal interoperability analysis for a later section) and some knowledge of the semantics of the ST-II protocol specification shows that the implementation $I1$ may fail to interoperate with a full implementation of ST-II, if it is presented with the event $SRefuse_{T2}$ before the event $SAccept_{T1}$. However, the implementation $I2$ will always successfully interoperate with any other implementation which fully implements ST-II. Even more significantly, the implementation $I3$ of the same specification S, depicted in Figure 3, also possesses the same ability to interoperate with full ST-II implementations under all circumstances, although it is not even strings equivalent to S for very short traces (refer to traces $Tr(I_S)$ in Example 2.1).

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 S where similar choices have been made. We quote some of those:

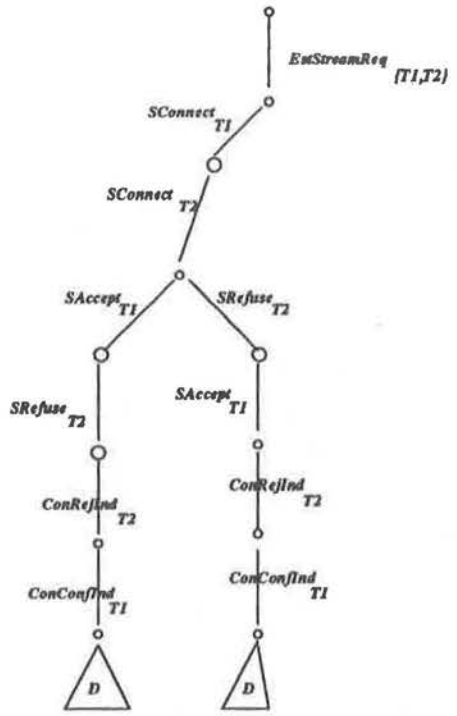

Figure 3: An interoperable implementation of S

- *one* external stimuli (of the type $EstStream$ request) which generates multiple protocol events which *may* be arbitrarily interleaved (e.g., events $SConnect_{T1}$, $SConnect_{T2}$, ..., $SConnect_{Tn}$ and implementation $I3$)
- *accumulation* of external stimuli (of the type $SAccept$ or $SRefuse$ Protocol Data Units in the example) which generates protocol events which *may* be arbitrarily temporally ordered, provided they individually satisfy timing correctness (e.g., events $ConConfInd$ and $ConRejInd$ and implementation $I2$)
- *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: Notice that, although many events initiated by the implementation under observation will be in one of these categories, not all are: consider, for example, certain priority control PDUs or express PDUs.

The observed problem is due to the fact that the formal theory of protocol testing based on observation fails to distinguish between different aspects of the

inability of an implementation to evolve via specific events. The consequence is that, in testing which is based on these equivalences and implementation relations, the inability of implementations such as $I1$ and $I2$ to synchronize on some specific observable events is treated in the same manner.

This characteristic has a profound impact on the design of the testing process itself.

1. In such a theory a tester must be able and will try to synchronize on all traces and observable events as described by the specification of the tested system, which may result in numerous inconclusive test runs if an implementation fails to react with the exact ordering of events that the tester expects to see. Other solutions [APRS92] may include imposing artificial requirements on the testing process (such as establishing a *stable state* before proceeding with testing), which may reduce the error exposition probability of the testing process.

2. In addition, test selection under such premises becomes difficult to solve effectively within the theory itself. A possible test selection scenario could drop some traces that are crucial for interoperability (intuitively, an implementation, such as $I1$, must be tested to be able to synchronize on $SAccept$ PDU followed by a $SRefuse$ PDU at any time, as well as vice versa), in favour of such traces which cannot even be observed by testing for a particular implementation (for instance, a $ConConfInd$ followed by $ConRejInd$, in case of the implementation $I2$), and which may be irrelevant for the interoperability of such implementation.

To overcome these problems, we propose a new formal approach aimed at resolving some practical test design issues and improve the efficiency of multiconnection protocol testing in particular. The testing will still be based on observation, but the underlying formal relation of validity will be greatly relaxed to include such implementations which, on experimentation, are observed to have many fewer traces than the specification. The successful termination of the testing process (if the theoretical upper bound were reachable) would guarantee that all implementations of the same specification which pass will be able to interoperate.

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 much more efficient. We delay these practical considerations for the next section, and concentrate on theoretical comparison next.

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:

$$L_{req} \cap L_{alt} = \phi \text{ and } L_{req} \cup L_{alt} = L$$

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 "*req*uired" 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 "*alt*ernative" synchronization).

Similarly, let $Ref_{req}(P,\sigma) = Ref(P,\sigma) \cap \mathcal{P}(L_{req})$ and $Ref_{alt}(P,\sigma) = Ref(P,\sigma) \cap \mathcal{P}(L_{alt})$ (where $\mathcal{P}(A)$ 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. Observe that the refusal sets satisfy the following properties.

Proposition 3.1 (Properties of refusal sets):

1. $Ref_{req}(P,\sigma)$ *and* $Ref_{alt}(P,\sigma)$ *are subset closed*
2. $Ref_{req}(P,\sigma) \cup Ref_{alt}(P,\sigma) \subseteq Ref(P,\sigma)$ *(not necessarily =)*
3. $Ref_{req}(P,\sigma) = \phi \Rightarrow Ref_{alt}(P,\sigma) = Ref(P,\sigma)$ *and*
   $Ref_{alt}(P,\sigma) = \phi \Rightarrow Ref_{req}(P,\sigma) = Ref(P,\sigma)$
4. $(Ref_{req}(P,\sigma) \subset Ref(P,\sigma) (\vee Ref_{alt}(P,\sigma) \subset Ref(P,\sigma)\ ) \Rightarrow Ref_{req}(P,\sigma) \cup Ref_{alt}(P,\sigma) \subset Ref(P,\sigma)$ *(proper subsets)*
5. $Ref(P,\sigma) \subseteq Ref(Q,\sigma) \Rightarrow (\ Ref_{req}(P,\sigma) \subseteq Ref_{req}(Q,\sigma) \wedge Ref_{alt}(P,\sigma) \subseteq Ref_{alt}(Q,\sigma))$ *(the opposite does not necessarily hold)*
6. $\mathcal{P}(\{\cup_{R \in Ref_{req}(P,\sigma)} R\} \cup \{\cup_{R \in Ref_{alt}(P,\sigma)} R\}) \supseteq Ref(P,\sigma)$

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. $L_{alt} \cap Out(S, \sigma) \cap Out(I, \sigma) \neq \phi \vee L_{alt} \in Ref_{alt}(S, \sigma)$
3. For the set $A = L_{alt} \cap (Out(S, \sigma) \backslash Out(I, \sigma))$ we have

$$Ref(I, \sigma) \backslash \{R \mid R \cap A \neq \phi\} \subseteq Ref(S, \sigma)$$

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.

As an example, consider the validity of the implementation $I3$ in the sense of *intop* (it is easy to observe that $I3$ is neither in *conf* nor *red* relation with S).

Example 3.1 *Let* $L_{req} = \{EstStreamReq_{\{T1,T2\}}\ ,\ SAccept_{T1}\ ,\ SRefuse_{T2}\}$ *and* $L_{alt} = \{SConnect_{T1}\ ,\ SConnect_{T2}\ ,\ ConConfInd_{T1}\ ,\ ConRejInd_{T2}\}$.
Consider the implementation $I3$ *of* S, *after the trace* $\sigma = EstStreamReq_{\{T1,T2\}}$.
$Ref_{req}(I3, \sigma) = \mathcal{P}(\{EstStreamReq_{\{T1,T2\}}\ ,\ SAccept_{T1}\ ,\ SRefuse_{T2}\})$
$Ref_{alt}(I3, \sigma) = \mathcal{P}(\{SConnect_{T2}\ ,\ ConConfInd_{T1},\ ConRejInd_{T2}\})$

Then,

1. $Ref_{req}(I3, \sigma) \subseteq \mathcal{P}(\{EstStreamReq_{\{T1,T2\}}\ ,\ SAccept_{T1}\ ,\ SRefuse_{T2}\} = Ref_{req}(S, \sigma)$, *and*
2. $SConnect_{T1} \in L_{alt}$, $\{SConnect_{T1}\} \in Out(S, \sigma)$ *and* $\{SConnect_{T1}\} \in Out(I, \sigma)$, *and*
3. $Ref(I3, \sigma) = \mathcal{P}\ ((\cup_{R \in Ref_{alt}(I,\sigma)} R) \cup (\cup_{R \in Ref_{req}(I,\sigma)} R)) = \mathcal{P}(\{SConnect_{T2}\ ,\ ConConfInd_{T1},\ ConRejInd_{T2}\} \cup \{\ EstStreamReq_{\{T1,T2\}}\ ,\ SAccept_{T1}\ ,\ SRefuse_{T2}\})$,
   and $A = \{SConnect_{T2}\}$. *Therefore,*
   $Ref(I3, \sigma) \backslash \{R \mid R \cap \{SConnect_{T2}\} \neq \phi\} = \mathcal{P}(\{\ ConConfInd_{T1}, ConRejInd_{T2},\ EstStreamReq_{\{T1,T2\}}\ ,\ SAccept_{T1}\ ,\ SRefuse_{T2}\}) \subseteq Ref(S, \sigma)$.

Therefore, I intop S by the Definition 3.1.

3.2 Properties of the *intop* Relation

We collect some easy facts about the *intop* relation.

Proposition 3.2 Let I *intop* S and $I = \sigma \Rightarrow$.

1. $Ref_{alt}(I, \sigma) \subseteq Ref_{alt}(S, \sigma) \Rightarrow Ref(I, \sigma) \subseteq Ref(S, \sigma)$
2. $(\forall \sigma \in Tr(S) \cap Tr(I),\ Ref_{alt}(I, \sigma) \subseteq Ref_{alt}(S, \sigma)) \Rightarrow I\ conf\ S.$
3. $I\ conf\ S \Rightarrow I\ intop\ S$ (i.e. $intop \supset conf$)
4. *intop* is reflexive
5. *intop* is not transitive

Proof: The proof of this theorem highlights some of the properties of the relation *intop* and can be found in the APPENDIX 3.

3.3 The $intop_{red}$ Relation

The results of Proposition 3.1 (iv) and (v) are in keeping with the theory developed in [Led92], 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). However, notice that similarly to $conf$, the relation *intop* allows that additional traces may exist in the implementation, that are not part of the specification. This feature becomes even more critical when interoperability of different implementations is examined, since, in general, such implementations may 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 which restricts the traces in an implementation to those of the specification. Unlike *intop*, this relation is also transitive.

Definition 3.2 $I\ intop_{red}\ S$ *iff*

1. $Tr(I) \subseteq Tr(S)$
2. $I\ intop\ S$

Proposition 3.3 The relation $intop_{red}$ has the following properties:

1. $intop \supset intop_{red}$
2. $intop_{red} \supset red$

3. $intop_{red}$ 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 the APPENDIX 4.

The above theoretical considerations are sufficient as a basis for specifying the necessary architectural and design requirements in interoperability testing. We do however note that the formal notion of interoperability as an implementation relation can be extended in the sense of *imp-eq* and other formal theory given in [Led92].

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 *intop* between an Implementation Under Test (IUT) and its specification S.

4.1 Architectural considerations

The general theory of protocol testing allows for different test architectures and different test interfaces. Consider the test architecture given in Figure 4. 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. 4, 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. 4.) in order to ensure the proper internetworking of different implementations in all environments. In this report, we concentrate on the interoperability of different implementations of the same (peer) protocols. (It may be interesting to study to what extent the same theory applies towards the interoperability of any two implementations that share the same interface.) 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

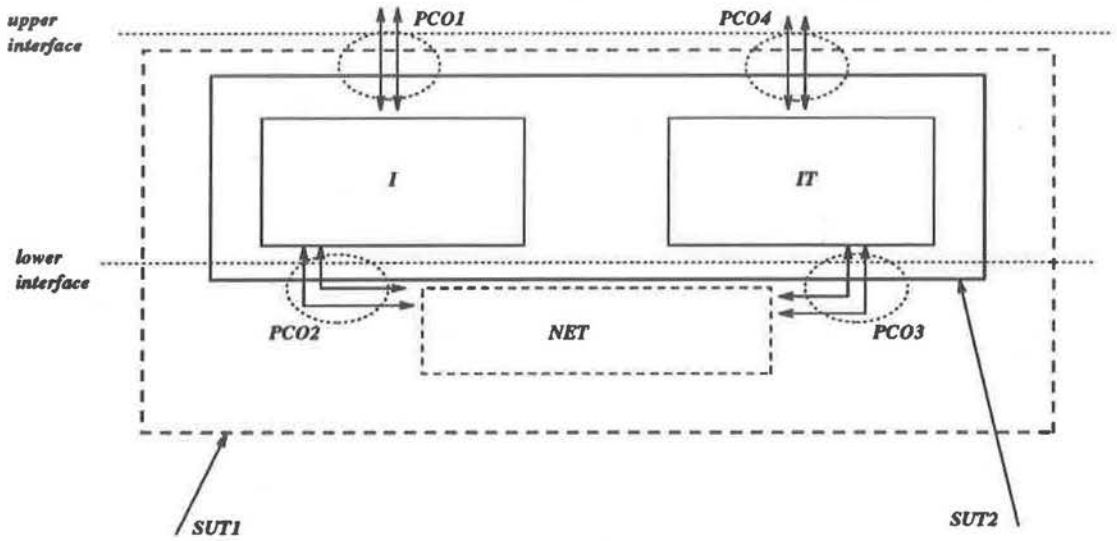

Figure 4: Interoperability Test Architecture

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_{req} as the next event is eliminated. This may or may not hold for actual implementations, but will simplify our further analysis.

[1] In our example specification, we adopt a simple strategy to prefix the name of each primitive by the address at which it occurs including S for the source (calling) ST-II agent and T for the target (called) ST-II agent, in order to allow for a simple NET process modelling. Other specification possibilities that provide distinctness of interactions exist and are equally suitable for this setting (see [Got92] for architectural details of interaction points in various specification formalisms).

4.2 Formal Network Protocol Specification Issues

The requirements of interoperability testing regarding the observability and controllability of PCOs impose strict requirements on 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 illustration, we complete our example ST-II specification of the stream establishment phase with the specification of the target process, supplying the remaining visible interactions at these gates. We simplify the specification in the manner similar to Example 2.1. (We will nor worry about the details of the negotiation of connection establishment with the multimedia application above and let the ST-II target agent decide on acceptance or refusal of the connection itself.)

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

$$T := (TConnect_{T1}; TAccept_{T1}; D)$$
$$|||$$
$$(TConnect_{T2}; TRefuse_{T2})$$

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 synchronization trees given in Figure 1.

The specification of the NET FIFO process can be given as in [Got92]. For the purposes of our brief example we will informally observe that for every event 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 Consider the application of $EstStreamReq_{\{T1,T2\}}$ at the PCO1, and assume that the additional revealed PCOs are PCO2, PCO3 and PCO4 (the system specified is exactly $SUT2$ in Fig. 4). Then the following trace may be observable at these PCOs:

σ=$EstStreamReq_{\{T1,T2\}}$. $SConnect_{T1}$. $SConnect_{T2}$. $TConnect_{T1}$. $TConnect_{T2}$. $TAccept_{T1}$. $TRefuse_{T2}$. $SAccept_{T1}$. $SRefuse_{T2}$. $ConRejInd_{T2}$. $ConConfInd_{T1}$

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 $IT(S)$ based on the the canonical tester [Bri88, Led92]. 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. $\forall I$ I $conf$ S iff ($\forall \sigma \in L^*$ we have ($L \in Ref(I \parallel X, \sigma) \Rightarrow L \in Ref(X, \sigma)$))

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 [Bri88, Led92], 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 5, where S, T and IT denote the specification, its canonical tester and its interoperability tester. In the derivation of the canonical tester $T(S)$, the purpose of this choice over i is to allow the tester

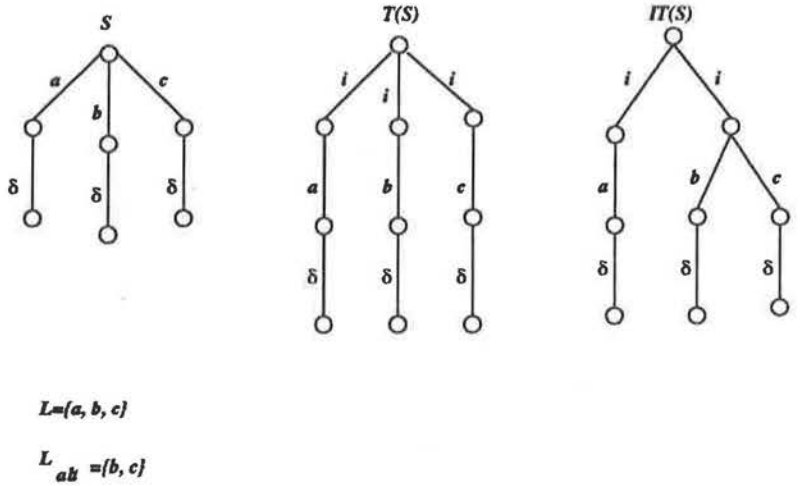

Figure 5: A specification, its canonical tester $T(S)$ and its interoperability tester $IT(S)$

to attempt synchronization on each action (a, b, c in the example) in L at that point in the specification.

By the definition 3.1 (of the *intop* relation), evolving via actions in L_{req} ($I = a \Rightarrow I'$ of the example) is also mandatory for every implementation I, and we therefore leave the design of $T(S)$ intact with respect to such actions (the trace $i.a.\delta$). The tester $IT(S)$ will also require synchronization on each such sequence.

However, by the definition of the *intop* relation, for actions in L_{alt}, not all possible actions need to be observable or observed at that particular point. In the example, either $I = b \Rightarrow I'$ or $I = c \Rightarrow I''$ must be observable. It follows that the tester $IT(S)$ must attempt synchronization on either $IT(S) = b \Rightarrow$ or $IT(S) = c \Rightarrow$. In such cases, $IT(S)$ is derived from the canonical tester $T(S)$ by substituting the n internal 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 $IT(S) := i; a[]i; (b[]c)$. 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

$$[]\{a_p; \ldots \mid p \in P\}[]\{i; b_q; \ldots \mid q \in Q\}$$

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

$$[]\{a_p; \ldots \mid p \in P\}[]\{i; b_r; \ldots \mid b_r \in R \subseteq Q\}[]i; ([]\{b_a; \ldots \mid b_a \in A \subseteq Q\}$$

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

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:

- Within the theory itself, such a selection process can be designed, which will guarantee not to sacrifice traces needed for interoperability in favour of possibly unobservable traces

  The test selection theory itself is beyond the scope of this report. We however observe that all the events that can happen alternatively 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 (other test selection criteria assumed to contribute separately).

- 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

  It follows immediately, from the construction of the tester and the definition of the relation *intop*, 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 test 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 be resolved on interaction of the interoperability tester with the environment

4.4 The interoperability of protocol implementations

We now turn to the formal notion of interoperability of two protocol implementations within our framework. We restrict ourselves to the interoperability of peer protocol implementations only. Therefore, the labelset of actions L_{alt} can only be interpreted in the context of the lower interface of protocol implementations. We first introduce some necessary definitions.

Definition 4.2 The set L_{alt} is said to be well defined if $\neg(\exists a \in L_{alt})$ such that $NET = \sigma \Rightarrow NET' - b.a \rightarrow NET''$ for some $\sigma \in Tr(NET)$ and $b \in L$, where the events a and b are distinguishable only by the calling or called prefix (S or T in our addressing convention) of the address at which they occur.

Informally, a well defined set L_{alt} does not contain elements which can occur as output of the underlying channel NET.[2] Observe that, although we consider the interoperability notion between two implementations only, this definition is general enough to apply also in the context of any number of implementations and any number of connections, because of the properties of the NET channel.

We now give a definition of the interoperability between two implementations of the same protocol specification.

Definition 4.3 P $intop_S$ Q iff for every event offered by either P or Q,

1. the concurrent execution of P and Q yields traces in S and
2. the concurrent execution of P and Q will not deadlock after a trace σ unless S can deadlock after that same trace

We are now ready to prove the following result.

Theorem 4.1 Let L_{alt} be well defined in the sense of definition 4.2. Let $I1$ and $I2$ be two implementations and S their common specification. Let NET be a reliable $FIFO$ channel. Then,

$$(I1\ intop_{red}\ S \text{ and } I2\ intop_{red}\ S\ ) \Rightarrow (I1\ intop_S\ I2\ )$$

[2]Notice that the distinctness of interactions (by interaction points including a calling or called agent prefix in this report) influences the definition of well-definedness of the set L_{alt}. Other specification styles may yield different instantiations of the same definition, but it suffices to say that in our architectural model, (with underlying FIFO channel), such a definition is always possible.

Proof: The proof of this theorem can be found in APPENDIX 5.

We finally observe that, if we substitute *intop* wherever $intop_{red}$ occurs in the Theorem 4.1, we obtain a slightly weaker result. We cannot state anything about the traces that $I1$ and $I2$ may generate while concurrently executing, if these traces are not in S. (they may even deadlock on such traces). This is why our final result is stated in terms of the relation $intop_{red}$. As a matter of practical importance, however, it is expected that a tester will only be able to systematically examine the traces in S, and leave the verdict about the extra traces in $I1$, $I2$ inconclusive.

5 Conclusion

In this report 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. However, the framework could benefit both from including more strict theoretical results (especially along the results in [Led92]) as well as more efficient algorithmic solutions of the interoperability tester derivation. For truly rigorous testing of modern network protocols in their full multiconnection capacity, both ingredients are needed.

References

[APRS92] N. Arakawa, M. Phalippou, N. Risser, and T. Soneoka. Combination of conformance and interoperability testing. In *5th International Conference FORTE '92*. Lannion,France, 1992.

[Bri88] B. Brinksma. A theory for the derivation of tests. In Aggarwal and Sabnani, editors, *Protocol Specification, Testing and Verification VIII*. North-Holland, 1988.

[dN87] R. de Nicola. Extensional equivalences for transition systems. *Acta Informatica*, (24):211–237, 1987.

[Got92] R. Gotzhein. Formal definition and representation of interaction points. *Computer Networks and ISDN Systems*, (25):3–22, 1992.

[Led92] Guy Leduc. Conformance relation, associated equivalence, and new canonical tester in lotos. In B. Jonnson, J. Parrow, and B. Pehrson,

editors, *To Appear in: Protocol Specification, Testing and Verification XI.* North-Holland, 1992.

[Mil80] R. Milner. A calculus of communicating systems. In *Lecture Notes in Computer Science 92.* Springer-Verlag, New York/Berlin, 1980.

[Top90] C. ed. Topolcic. Experimental internet stream protocol, version 2 (st-ii); rfc-1190. *Internet Requests for Comments*, (1190), October 1990.

APPENDIX 1: LOTOS

LOTOS (Language of Temporal Ordering Specification)[8] is a Formal Description Technique developed within ISO (International Organization for Standardization) for the formal specification of open distributed systems, and in particular for those related to the Open Systems Interconnection (OSI) computer network architecture. These concurrent real time systems are specified in LOTOS by defining the temporal relation among the interactions that constitute the externally observable behavior of the system. In the Table 1 we give such rules for the subset of LOTOS that is used in this report.

Combinator	**Axioms or Inference Rules**
stop	none
exit	exit - $\delta \rightarrow$ stop
m;B	m; B-m $\rightarrow B$ (m $\in$ L)
i;B	i;B - $\tau \rightarrow B$
$B_1[]B_2$	$B_1 - m \rightarrow B_1' \mid - B_1[]B_2 - m \rightarrow B_1'$
	$B_2 - m \rightarrow B_2' \mid - B_1[]B_2 - m \rightarrow B_2'$
$B_1 \mid\mid B_2$	$B_1 - m \rightarrow B_1', B_2 - m \rightarrow B_2' \mid - B_1 \mid\mid B_2 - m \rightarrow B_1' \mid\mid B_2'$

Table 1

APPENDIX 2: NOTATION

Processes (denoted by T, and ranged over by P, T, T_1,... will be sets of labelled transition systems over an alphabet $L \cup \{i\}$ (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, behave 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, behave like process P'.
$P - a.b \rightarrow P'$ means $\exists P''$, such that $P - a \rightarrow P''$ and $P'' - b \rightarrow P'$
$P = a \Rightarrow P'$ means $\exists k_0, k_1 \in N$ such that $P - i^{k_0}.a.i^{k_1} \rightarrow P'$
$P = a \Rightarrow$means $\exists P'$ such that $P = a \Rightarrow P'$, i.e. P may engage in an action a
$P \neq a \Rightarrow$ means $\neg(P = a \Rightarrow)$ i.e. P cannot engage in an action a
$P = \sigma \Rightarrow P'$ means that process P may engage in a sequence of observable actions σ and, after doing so, behave like process P'.
$P = \sigma \Rightarrow$ means that $\exists P'$ such that $P = \sigma \Rightarrow P'$
$Tr(P)$ is the trace set of P, i.e. $\{\sigma \mid P = \sigma \Rightarrow\}$; $Tr(P)$ is a subset of L^*
$P\ after\ \sigma = \{P' \mid P = \sigma \Rightarrow P'\}$, i.e. the set of all behaviour expressions (or states) reachable by σ
$Out(P, \sigma)$ is the set of possible observable actions after the trace σ, i.e.

$$Out(P, \sigma) = \{a \in L \mid \sigma.a \in Tr(P)\}$$

$Ref(P, \sigma)$ is the refusal set of P after trace σ, i.e.

$$Ref(P, \sigma) = \{X \subseteq L \mid \exists P' \in P\ after\ \sigma,\ such\ that\ P' \neq a \Rightarrow, \forall a \in X\}$$

APPENDIX 3

Proof of the Proposition 3.2

The proof for 1. follows directly from the observation that in the case when $Ref_{alt}(I,\sigma) \subseteq Ref_{alt}(S,\sigma)$ then the set A from Def. 3.1, condition 3., is empty. Since also I *intop* S, the condition in Def. 3.1, 3., holds which proves that $Ref(I,\sigma) \subseteq Ref(S,\sigma)$ holds. (Notice that this is exactly the condition needed for the opposite implication in Prop. 3.1, 5).

2. follows directly from 1. and the definition of $conf$ relation.

3. Assume that $Ref(I,\sigma) \subseteq Ref(S,\sigma) \forall \sigma \in Tr(S) \cap Tr(I)$.

- We have $Ref_{req}(I,\sigma) = Ref(I,\sigma) \cap \mathcal{P}(L_{req})$ and $Ref_{req}(S,\sigma) = Ref(S,\sigma) \cap \mathcal{P}(L_{req})$. Therefore,
  $Ref_{req}(I,\sigma) = Ref(I,\sigma) \cap \mathcal{P}(L_{req}) \subseteq Ref(S,\sigma) \cap \mathcal{P}(L_{req}) = Ref_{req}(S,\sigma)$.
- Assume that the first part of Def. 3.1, 2., is not true. Then $\forall a \in L_{alt}$ such that $a \in Out(S,\sigma)$ we have $a \notin Out(I,\sigma)$. Hence $\{a\} \in Ref_{alt}(I,\sigma) \subseteq Ref_{alt}(S,\sigma)$ by Prop. 3.1, 5. We clearly have that the union of $Out(S,\sigma) \cap L_{alt}$ and all one-element subsets of $Ref_{alt}(S,\sigma)$ must be equal to L_{alt}. However, since $Out(S,\sigma) \cap L_{alt} \in Ref_{alt}(S,\sigma)$, it follows that $L_{alt} \in Ref_{alt}(S,\sigma)$, which proves that the second condition of the Def. 3.1 holds.
- The part 3. of Def. 3.1 holds trivially.

This proves that $intop \supset conf$.

4. and 5. follow directly from the definition of *intop* relation.

APPENDIX 4

Proof of Proposition 3.3

We will prove that $intop_{red}$ is a preorder.

1. $intop_{red}$ is reflexive since $Tr(I) \subseteq Tr(I)$ and I *intop* I (by Proposition 3.1., 4.)

2. $intop_{red}$ is transitive: Let $I\ intop_{red}\ J$ and $J\ intop_{red}\ K$. Then:

   (a) ($Tr(I) \subseteq Tr(J)$ and $Tr(J) \subseteq Tr(K)$) $\Rightarrow Tr(I) \subseteq Tr(K)$

   (b) Let $\sigma \in Tr(K)$. Then

      i. $Ref_{req}(I,\sigma) \subseteq Ref_{req}(J,\sigma)$ (from I *intop* J) and similarly $Ref_{req}(J,\sigma) \subseteq Ref_{req}(K,\sigma)$. These two relations imply $Ref_{req}(I,\sigma) \subseteq Ref_{req}(K,\sigma)$.

      ii. By the definition of the *intop* relation, $\exists a \in L_{alt} \cap Out(J,\sigma)$ such that $a \in Out(I,\sigma)$. Since $\sigma.a \in Tr(J) \subseteq Tr(K)$ we conclude that

      $a \in L_{alt} \cap Out(K,\sigma)$ and $a \in Out(I,\sigma)$. Therefore, the second requirement of Def. 3.1 is satisfied.

      iii. Denote by $B = L_{alt} \cap (Out(K,\sigma) \backslash Out(I,\sigma))$ and by $C = L_{alt} \cap (Out(K,\sigma) \backslash Out(J,\sigma))$. $a \in C \Rightarrow \sigma.a \in Tr(K)$, but $\sigma.a \notin Tr(J)$. Therefore, $\sigma.a \notin Tr(I)$. Thus $a \notin Out(I,\sigma)$. Hence $C \subset B$. Since J *intop* K we have
      $Ref(J,\sigma) \backslash \{R \mid R \cap C \neq \phi\} \subseteq Ref(K,\sigma)$
      and consequently $Ref(J,\sigma) \backslash \{R \mid R \cap B \neq \phi\} \subseteq Ref(K,\sigma)$. (*)
      Put $A = L_{alt} \cap (Out(J,\sigma) \backslash Out(I,\sigma))$. Then,
      $Ref(I,\sigma) \backslash \{R \mid R \cap A \neq \phi\} \subseteq Ref(J,\sigma)$.
      We have $a \in A \Rightarrow \sigma.a$, $\sigma.a \notin Tr(I) \Rightarrow \sigma.a \in Tr(K)$, $\sigma.a \notin Tr(I) \Rightarrow a \in B$.
      Thus, $A \subset B$. So, $\{R \mid R \cap A \neq \phi\} \subset \{R \mid R \cap B \neq \phi\}$ and therefore $Ref(I,\sigma) \backslash \{R \mid R \cap B \neq \phi\} \subseteq Ref(I,\sigma) \backslash \{R \mid R \cap A \neq \phi\} \subseteq Ref(J,\sigma)$. Consequently, $Ref(I,\sigma) \backslash \{R \mid R \cap B \neq \phi\} \subseteq R(J,\sigma) \backslash \{R \mid R \cap B \neq \phi\}$. By (*) it follows that
      $R(I,\sigma) \backslash \{R \mid R \cap B \neq \phi\} \subseteq Ref(K,\sigma)$.
      This proves the third property in Def. 3.1, therefore I *intop* K.

(a) and (b) together prove that $intop_{red}$ is transitive. Therefore, $intop_{red}$ is a preorder.

APPENDIX 5

Proof of the Theorem 4.1

First, observe that, by Definition 4.3, the concurrent execution of $I1$ and $I2$ includes both their independent simultaneous execution (i.e., $I1$ and $I2$ are communicating with some other implementations $I3$ and $I4$), and their concurrent execution where $I1$ and $I2$ are communicating with each other.

The first case is represented by independent interleaving in the interleaving model of concurrency. The proof in this case follows directly from the trace set inclusion property for the relation $intop_{red}$ (Def. 3.2., 1.).

Consider now the second case. For the purpose of this proof, we will consider that the underlying NET connection has both sets of its $PCOs$ (i.e. gates T and S) attached to one protocol process I capable of running multiple connections. Also, we assume that the specification S allows multiple concurrent processes corresponding to these connections. Then, without loss of generality, the execution of $I1$ and $I2$ can be viewed as the process I, specified as a parallel composition of the corresponding multiple connections of the protocol specification S.

Let $I1$ $intop_{red}$ S and $I2$ $intop_{red}$ S, and let $\sigma \in Tr(I1 \parallel I2 \parallel NET)$. Then there exists a trace σ' (a prefix of σ) such that $\sigma'.a \in Tr(I1 \parallel I2 \parallel NET)$ and $\sigma' \in Tr(S)$. Denote by $\sigma'_{I1} \in Tr(I1)$ ($\sigma'_{I2} \in Tr(I2)$) the projection of the trace σ' on the events in $I1$ ($I2$ respectively). Observe that $I1 = \sigma'_{I1} \Rightarrow I1' - a \rightarrow$ or $I2 = \sigma'_{I2} \Rightarrow I2' - a \rightarrow$. Then, $\sigma'_{I1}.a \in Tr(I1)$ or $\sigma'_{I2}.a \in Tr(I2)$. It follows that (by the trace set inclusion for $intop_{red}$ relation and the operational semantics of the parallel operator) if

$(I1 \parallel I2 \parallel NET) = \sigma' \Rightarrow (I1' \parallel I2 \parallel NET) - a \rightarrow$ then
$\sigma'.a \in Tr(S)$ and similarly for $I2'$. Therefore, every trace generated by the concurrent execution of $I1$ and $I2$ is also a trace in S.

We next prove that $(I1 \parallel I2 \parallel NET)$ cannot deadlock after some $\sigma.a \in Tr(I1 \parallel I2 \parallel NET)$ if such a deadlock cannot occur after $S = \sigma.a \Rightarrow$.
Suppose that, in our model, a is a lower interface event of the type Sa. Then,

$(I1 \parallel I2 \parallel NET) = \sigma' \Rightarrow (I1' \parallel I2' \parallel NET') = Sa \rightarrow (I1'' \parallel I2' \parallel NET'')$
and

$(I1'' \parallel I2' \parallel NET'') \neq Ta \Rightarrow$
This, in particular, means that there exists a trace $\sigma'_{I2} \in Tr(I2)$ such that $I2 = \sigma'_{I2} \Rightarrow I2' \neq Ta \Rightarrow$. The proof is then completed by observing that,

1. $\sigma'_{I2} \in Tr(S)$ by the first part of the proof, and
2. $Ta \in L_{req}$ (since L_{alt} is well defined) and $Ref_{req}(I2, \sigma'_{I2}) \subseteq Ref_{req}(S, \sigma'_{I2})$, both by the assumptions of the theorem.