On the automatic verification of non-standard measures of consistency*

Christie Bolton Gavin Lowe

Abstract

Consistency between a process and its specification expressed in CSP is typically presented as a refinement check. Within the traces model consistency is measured by examining only the traces of the systems, whilst in the finer stable failures model the possibility of subsequently refusing a combination of events is also taken into consideration.

The contribution of this paper is to identify and motivate the need for alternative measures of consistency, and to present and prove the soundness and completeness of general techniques for automatically verifying such consistencies. We achieve this by masking all failures information other than that associated with the measure in question.

More concretely, we describe methods for automatically checking: that a process can refuse a given set of events after any trace only if its specification can refuse the same set of events after the same trace; that a process might deadlock after any trace only if its specification might also deadlock after the same trace; and that a process might after any trace refuse a given number of events of a particular class only if its specification might also refuse the same set after the same trace.

Keywords: Refinement; formal verification techniques; model-checking; CSP.

1 Introduction

Refinement is a standard technique that is used to ascertain whether or not a model of a system is in some way consistent with, or satisfies a property captured by, its specification. The measure of consistency depends on the language used and often on the choice of semantic model. Process algebras such as CSP [Hoa85, Ros97] and CCS [Mil80, Mil89] can be used to model concurrent systems as can modal logics [Pnu77, CE81, EH86] and Petri nets [Pet81].

In the process algebra CSP a typical refinement check is of the form $SPEC \sqsubseteq P$ for process P and specification $SPEC$. Non-trivial problems are generally verified using the the model-checker FDR [Ros94, For99]. FDR checks that all behaviours of process P are possible behaviours of specification $SPEC$. The class of behaviours taken into consideration is determined by the choice of semantic model: the traces model [Ros97] simply records all possible *traces*, or sequences of interaction whereas the stable failures model [Ros97] records also the potential to *refuse* sets of events after each such trace.

The purpose of this paper is to identify and motivate the need for alternative measures of consistency between a process and its specification, and to present (and prove the correctness of) techniques for using existing tools to automatically verify

*Authors' address: Oxford University Computing Laboratory Wolfson Building, Parks Road, Oxford OX1 3QD, UK; {christie,gavinl}@comlab.ox.ac.uk. This work is partly funded by the Ministry of Defence through QinetiQ.

such consistencies. We restrict ourselves to measures that are finer than the traces model but courser than the stable failures model; we achieve our goal by masking all failures information other than that associated with the measure in question.

Example 1

The implementation of a pantomime horse[1] is modelled in the self-explanatory UML [RJB97] state diagram in Figure 1 below. The description says that the

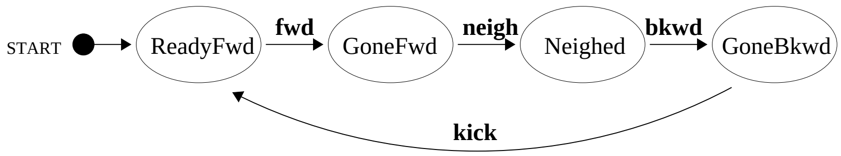

Figure 1: A state diagram illustrating the implementation of a pantomime horse

pantomime horse will repeatedly go forwards, neigh, go backwards and then kick. The original specification of the pantomime horse is modelled in Figure 2. The

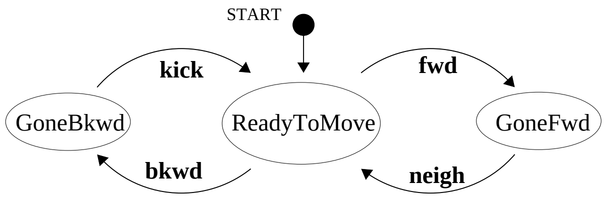

Figure 2: A state diagram illustrating the specification of a pantomime horse

specification can do any combination of going forwards then neighing and going backwards then kicking. Our question is "Is the implementation consistent with the specification?"

If we are are working in the context of the traces model then the answer is "Yes": every possible sequence of interaction of the implementation is a possible sequence of interaction of the specification. However if we consider full failures information then the answer is "No": the implementation could initially refuse to go backwards whilst the specification could not.

But perhaps neither of these is the correct measure. It might be that in this instance our measure of consistency should be "Can the implementation refuse to move only when the specification can also refuse to move?" In other words, can the implementation refuse both *fwd* and *bkwd* only when the specification could refuse both these events? If that is our measure then the answer is "Yes": the implementation is consistent with the specification.
$\square$

The intention of this paper is to provide general techniques for verifying that a process can exhibit a particular behaviour only when its specification could exhibit the same behaviour. The particular behaviours that concern us are: the refusal of a given combination of events; the potential to deadlock; the refusal of a given number of events; and the refusal of a given number of events from a specificied set or class.

[1]The idea of modelling a pantomime horse was taken from Roscoe's book [Ros97].

The paper begins with a brief introduction to the syntax of CSP, identifying the standard operators that we will be using throughout the rest of the paper. We follow this by a discussion of Roscoe's traces and stable failures models.

In Section 3 we present a general technique for verifying automatically whether or not a process can refuse a particular set of events only when its specification can. As well as proving the soundness and completeness of this technique, we discuss practical applications of such a measure of consistency, observing that the results presented in this section could be used to verify automatically (assuming the correct translation to CSP) that the implementation of the pantomime horse as defined in Example 1 is consistent with its specification.

Sections 4, 5 and 6 adopt a structure similar to Section 3: we identify a potential application for the measure of consistency under consideration; and we prove formally how consistency within this context might be automatically verified. These proofs apply arguments similar to those used in Theorem 1 of Section 3 and are therefore included in the appendix rather than the main body of the paper.

The measure of consistency with which we are concerned in Section 4 is that a process might deadlock after some trace only when its specification might deadlock after the same trace. In Section 5 we are concerned only with failures in which the cardinality of the refusal set is at least a specified value. And in Section 6 we are concerned only with failures in which the refusal is a subset of a given set and is of at least a specified size.

We conclude the paper in Section 7 with a discussion of related work.

2 Overview of CSP

In this section we give a brief overview of the syntax and semantics for CSP, as defined by Roscoe [Ros97], that we will be using throughout the rest of the paper.

2.1 Syntax

The process algebra CSP can be used to model systems such as networks, protocols and machines. Furthermore, simple components may be combined to create complex composite entities. A CSP *process* is a pattern of communication that describes the behaviour of a system. Behaviour is described in terms of *events* or synchronous atomic communications, marking points in the evolution of the system.

The simplest process is *Stop*. It will not perform any events and marks the end of a pattern of communication. By contrast, the process div represents a divergent process, which performs unboundedly many internal events and never reaches a stable state.

For any event a and process P, the process $a \rightarrow P$ is willing to communicate event a and, if that event occurs, will subsequently behave as P. If A is a set of events, then $?a : A \rightarrow P_a$ represents the process that is willing to communicate any event from A; if event $a \in A$, which can be thought of as an input, is performed, subsequent behaviour is that of P_a.

CSP has two choice operators: $P \sqcap Q$ and $P \mathbin{\Box} Q$ respectively represent the *internal* (or non-deterministic) choice and *external* choice between processes P and Q. In addition, there are indexed version of these operators: process $\bigsqcap i : I \mid p(i) \bullet P_i$ represents an indexed internal choice between the processes P_i where i ranges over those members of I such that $p(i)$ holds.

Given processes P and Q and sets of events A and B (their respective interfaces), the process $P \,{}_A\|_B\, Q$ denotes the *parallel* combination of P and Q. In such a parallel combination, a process can perform only those events that are in its interface and its cooperation is required if such an event is to occur; hence the processes synchronise on events in the intersection of their interfaces. By contrast, $P \mid\mid\mid Q$ represents an interleaving of P and Q, i.e. a parallel composition with no synchronisation.

The process $P \bigtriangleup Q$ represents a process that initially acts like P, but at any point, P can be interrupted and control passed to Q. If P is a process and R a relation over events, then $P\,[\![R]\!]$ represents the process that can perform an event b whenever P could perform an event a such that $a\,R\,b$. If P is a process and A a set of events, then the process $P \setminus A$ behaves as P except that events from A are hidden (or made internal) so cannot be observed and do not require the cooperation of any other process.

For later convenience we define also $\mathit{Offer}(A) \mathrel{\widehat=} ?a : A \rightarrow \mathsf{div}$, the process that offers the events from A and then diverges, and $\leadsto$, the non-deterministic interrupt operator defined as follows: $P \leadsto Q \equiv P \sqcap (P \bigtriangleup Q)$. These were introduced in [BL03].

2.2 Semantics

Given Σ, the set of all possible events, a *trace* describes a sequence of interaction and is an element of the type Σ^*. A *failure* is an element of the type $\Sigma^* \times \mathbb{P}\,\Sigma$; the first component is a trace and the second a *refusal set*; the refusal set is a set of events that might collectively be refused from a stable state (i.e. where no internal activity is possible) reached after the given trace.

Roscoe's stable failures model represents each process P by a pair in which the first component is a set of *traces* and the second a set of *failures*. The semantics of a process P is given by the pair $(\mathit{traces}(P), \mathit{failures}(P))$ where the functions *traces* and *failures* must satisfy the following healthiness conditions:

$$\langle\rangle \in \mathit{traces}(P), \tag{T1}$$

$$tr \frown tr' \in \mathit{traces}(P) \;\Rightarrow\; tr \in \mathit{traces}(P), \tag{T2}$$

$$(tr, X) \in \mathit{failures}(P) \;\Rightarrow\; tr \in \mathit{traces}(P), \tag{F1}$$

$$(tr, X \cup Y) \in \mathit{failures}(P) \;\Rightarrow\; (tr, X) \in \mathit{failures}(P), \tag{F2}$$

$$\begin{aligned}&(tr, Y) \in \mathit{failures}(P) \wedge \forall\, x \in X \bullet tr \frown \langle x \rangle \notin \mathit{traces}(P) \;\Rightarrow \\ &\qquad (tr, X \cup Y) \in \mathit{failures}(P).\end{aligned} \tag{F3}$$

Condition (T1) states that the empty trace is a possible trace of every process and condition (T2) states that the set of traces of any process is prefix-closed. Condition (F1) ensures consistency between failure and trace information whilst condition (F2) states that the set of refusal sets for every possible trace is subset closed. Finally, condition (F3) states that events that cannot be performed in a particular state may be added to a corresponding refusal set.

The functions *traces* and *failures* satisfy the following equations[2]

$$
\begin{aligned}
\mathit{traces}(\mathit{Stop}) &= \{\langle\rangle\}, \\
\mathit{failures}(\mathit{Stop}) &= \{(\langle\rangle, X) \mid X \in \mathbb{P}\,\Sigma\}, \\[1ex]
\mathit{traces}(\mathsf{div}) &= \{\langle\rangle\}, \\
\mathit{failures}(\mathsf{div}) &= \{\}, \\[1ex]
\mathit{traces}(\mathit{Offer}(A)) &= \{\langle\rangle\} \cup \{\langle a\rangle \mid a \in A\} \\
\mathit{failures}(\mathit{Offer}(A)) &= \{(\langle\rangle, X) \mid X \in \mathbb{P}\,\Sigma \wedge X \cap A = \emptyset\}, \\[1ex]
\mathit{traces}(a \to P) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown tr \mid tr \in \mathit{traces}(P)\}, \\
\mathit{failures}(a \to P) &= \{(\langle\rangle, X) \mid X \in \mathbb{P}\,\Sigma \wedge a \notin X\} \;\cup \\
&\qquad \{(\langle a\rangle \frown tr, X) \mid (tr, X) \in \mathit{failures}(P)\}, \\[1ex]
\mathit{traces}(?a : A \to P_a) &= \{\langle\rangle\} \cup \{\langle a\rangle \frown tr \mid a \in A \wedge tr \in \mathit{traces}(P_a)\}, \\
\mathit{failures}(?a : A \to P_a) &= \{(\langle\rangle, X) \mid X \in \mathbb{P}\,\Sigma \wedge A \cap X = \{\}\} \cup \\
&\qquad \{(\langle a\rangle \frown tr, X) \mid a \in A \wedge (tr, X) \in \mathit{failures}(P_a)\}, \\[1ex]
\mathit{traces}(P \,\Box\, Q) &= \mathit{traces}(P) \cup \mathit{traces}(Q), \\
\mathit{failures}(P \,\Box\, Q) &= \{(tr, X) \in \mathit{failures}(P) \cap \mathit{failures}(Q) \mid tr = \langle\rangle\} \cup \\
&\qquad \{(tr, X) \in \mathit{failures}(P) \cup \mathit{failures}(Q) \mid tr \neq \langle\rangle\}, \\[1ex]
\mathit{traces}(P \sqcap Q) &= \mathit{traces}(P) \cup \mathit{traces}(Q), \\
\mathit{failures}(P \sqcap Q) &= \mathit{failures}(P) \cup \mathit{failures}(Q), \\[1ex]
\mathit{traces}(\textstyle\bigsqcap i : I \mid p(i) \bullet P_i) &= \bigcup \{\mathit{traces}(P_i) \mid i \in I \wedge p(i)\}, \\
\mathit{failures}(\textstyle\bigsqcap i : I \mid p(i) \bullet P_i) &= \bigcup \{\mathit{failures}(P_i) \mid i \in I \wedge p(i)\}, \\[1ex]
\mathit{traces}(P \,\triangle\, Q) &= \{tr \frown tr' \mid tr \in \mathit{traces}(P) \wedge tr' \in \mathit{traces}(Q)\}, \\
\mathit{failures}(P \,\triangle\, Q) &= \{(tr \frown tr', X) \mid tr \in \mathit{traces}(P) \wedge \\
&\qquad\qquad (tr', X) \in \mathit{failures}(Q)\}, \\[1ex]
\mathit{traces}(P \;{}_A\|_B\; Q) &= \{tr \in (A \cup B)^* \mid tr \upharpoonright A \in \mathit{traces}(P) \wedge \\
&\qquad\qquad tr \upharpoonright B \in \mathit{traces}(Q)\}, \\
\mathit{failures}(P \;{}_A\|_B\; Q) &= \{(tr, X) \in (A \cup B)^* \times \mathbb{P}\,\Sigma \mid \\
&\qquad \exists\, Y \in \mathbb{P}\,A;\; Z \in \mathbb{P}\,B;\; W \in \mathbb{P}(\Sigma - A - B) \bullet \\
&\qquad\quad (tr \upharpoonright A, Y) \in \mathit{failures}(P) \wedge \\
&\qquad\quad (tr \upharpoonright B, Z) \in \mathit{failures}(Q) \wedge \\
&\qquad\quad X = Y \cup Z \cup W\}, \\[1ex]
\mathit{traces}(P \;|||\; Q) &= \{tr \mid \exists\, tr_P \in \mathit{traces}(P), tr_Q \in \mathit{traces}(Q) \bullet \\
&\qquad\qquad tr \in tr_P \;|||\; tr_Q\},
\end{aligned}
$$

[2]In the equations for $P \;|||\; Q$, the notation $tr_P \;|||\; tr_Q$ represents all ways of interleaving the traces tr_P and tr_Q; see [Ros97]. Similarly, the notation $tr \upharpoonright X$ represents trace tr restricted to those events in set X.

If R is a relation over events, then R^* is the corresponding lifted relation over traces, and $R(\!|\, X \,|\!)$ is the relational image of set X.

Note that the version of the interrupt operator presented here is not that implemented in FDR. The version presented here can, however, be modelled in FDR as follows: `Interrupt(P,Q) = (P /\ (i->Q)) \ {i}` for some fresh event i.

Finally, observe that `div` has no stable failures, because it never reaches a stable state, and that $P \,\triangle\, Q$ has no failures corresponding to P, because in such states it can be interrupted at any point, so never stabilises.

$$
\begin{aligned}
failures(P \mid\mid\mid Q) &= \{(tr, X) \mid \exists\, tr_P, tr_Q \bullet (tr_P, X) \in failures(P) \wedge \\
&\qquad\qquad\qquad\qquad\qquad (tr_Q, X) \in failures(Q) \wedge \\
&\qquad\qquad\qquad\qquad\qquad tr \in tr_P \mid\mid\mid tr_Q\},
\end{aligned}
$$

$$
\begin{aligned}
traces(P\,[\![R]\!]) &= \{\, tr' \mid \exists\, tr \in traces(P) \bullet tr\ R^*\ tr' \,\}, \\
failures(P\,[\![R]\!]) &= \{\, (tr', X) \mid \exists\, tr \bullet (tr, R(\!|\ X\ |\!)) \in failures(P) \wedge \\
&\qquad\qquad\qquad\qquad tr\ R^*\ tr' \,\},
\end{aligned}
$$

$$
\begin{aligned}
traces(P \setminus A) &= \{\, tr \setminus A \mid tr \in traces(P) \,\}, \\
failures(P \setminus A) &= \{(tr \setminus A, X) \mid (tr, A \cup X) \in failures(P)\},
\end{aligned}
$$

$$
\begin{aligned}
traces(P \rightsquigarrow Q) &= traces(P) \cup \\
&\qquad \{tr \frown tr' \mid tr \in traces(P) \wedge tr' \in traces(Q)\}, \\
failures(P \rightsquigarrow Q) &= failures(P) \cup \\
&\qquad \{(tr \frown tr', X) \mid tr \in traces(P)\ \wedge \\
&\qquad\qquad\qquad (tr', X) \in failures(Q)\}.
\end{aligned}
$$

Given these definitions, refinement and equivalence in the traces and stable failures models can be defined as follows:

$$
\begin{aligned}
P \sqsubseteq_{\mathcal{T}} Q &\Leftrightarrow traces(Q) \subseteq traces(P), \\
P \equiv_{\mathcal{T}} Q &\Leftrightarrow traces(P) = traces(Q), \\
P \sqsubseteq_{\mathcal{F}} Q &\Leftrightarrow traces(Q) \subseteq traces(P) \wedge failures(Q) \subseteq failures(P), \\
P \equiv_{\mathcal{F}} Q &\Leftrightarrow traces(P) = traces(Q) \wedge failures(P) = failures(Q).
\end{aligned}
$$

3 Refusal of a specific set of events

In this section we present an automatable refinement check that will identify whether or not there is some trace after which a process can refuse a specified set of events but after which its specification could not.[3]

3.1 Practical application

A test of the form described above will be useful whenever we have a system in which we are concerned about the availability of at least one of a given class of operations, although which operation is actually performed does not concern us.

Suppose for instance that a service provider has a collection of processors indexed by some set I. The acceptance and completion of a job by processor i are respectively modelled by events $acceptJob.i$ and $completeJob.i$. An initial specification might be of the following form:

$$
\begin{aligned}
ServiceProviderSpec &\mathrel{\widehat{=}} \mid\mid\mid_{\,i:I}\ Processor(i) \\
Processor(i) &\mathrel{\widehat{=}} acceptJob.i \rightarrow completeJob.i \rightarrow Processor(i).
\end{aligned}
$$

To maximise efficiency, the service provider decides to incorporate a scheduler into the system when it comes to the actual implementation. The scheduler's role is to allocate jobs to processors. How it makes its decisions is not important for the purpose of this example. The final implementation is modelled as follows:

$$
ServiceProviderImpl \mathrel{\widehat{=}} Scheduler \underset{\{acceptJob.i \mid i \in I\}}{\|} ServiceProviderSpec.
$$

[3] Bill Roscoe [Ros03a] has come up with an alternative method of using FDR to determine whether one process can refuse a given set of events where another could not. We discuss this further in Section 7.

We are concerned not with which processor performs which job, but that the implementation cannot block the acceptance of a job where the specification would not. We need to verify that the following predicate holds:

$$\begin{aligned}&\{(tr, X) \in \mathit{ServiceProviderImpl} \mid X = \{\mathit{acceptJob}.i \mid i \in I\}\}\\&\quad\subseteq \{(tr, X) \in \mathit{ServiceProviderSpec} \mid X = \{\mathit{acceptJob}.i \mid i \in I\}\}\end{aligned}$$

and hence need the result presented in Theorem 1 below.

3.2 Automatic verification

The following predicate captures that there is no trace of process P after which it can refuse the set of events A, but after which its specification SPEC cannot refuse that same set:

$$\{(tr, X) \in \mathit{failures}(P) \mid X = A\} \subseteq \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid X = A\}. \qquad (1)$$

To verify automatically whether or not this is the case, we must reduce the problem to an equivalent one expressed in terms of refinement within the established semantic models. We must retain failures information when the refusal set is A, and mask all other failures information. Provided SPEC is trace-refined by P we can achieve this by:

- Preserving the trace information of both P and SPEC;
- Removing all trace-refusal pairs (tr, X) where $X \not\subseteq A$ from the failures sets of both P and SPEC.
- Flooding the failures sets of both P and SPEC with all trace-refusal pairs (tr, X) where $tr \in \mathit{traces}(P)$ and $X \subset A$; and

If the resulting processes are P' and SPEC', then Equation (1) will hold precisely when $P' \sqsubseteq_{\mathcal{F}} \mathit{SPEC}'$.

Removing failures can be achieved by interleaving with a suitable process—in this case a process in which every event outside A is initially available. Flooding with additional failures at each trace can be achieved by non-deterministically allowing interruption by a suitable process—in this case a process which can refuse every *strict* subset of A.

In Theorem 1 below we prove our desired result for an arbitrary non-empty set A.

Theorem 1 For any process P defined over finite set of events Σ, and with specification SPEC such that $\mathit{SPEC} \sqsubseteq_{\mathcal{T}} P$, and for any given set of events $A \neq \emptyset$, Equation 1 is satisfied precisely when

$$(\mathit{SPEC} \mathbin{|||} \mathit{Offer}(\Sigma - A)) \rightsquigarrow R_A \sqsubseteq_{\mathcal{F}} (P \mathbin{|||} \mathit{Offer}(\Sigma - A)) \rightsquigarrow R_A$$

where $R_A = \bigsqcap Y \mid Y \subset A \bullet \mathit{Offer}(\Sigma - Y)$.

Proof First observe that, by definition of $|||$ and Offer, for any process Q,

$$\begin{aligned}&\mathit{failures}(Q \mathbin{|||} \mathit{Offer}(\Sigma - A))\\&\quad= \{(tr, X) \mid \exists\, tr', tr'' \mid (tr', X) \in \mathit{failures}(Q) \wedge\\&\qquad\qquad (tr'', X) \in \mathit{failures}(\mathit{Offer}(\Sigma - A)) \wedge\\&\qquad\qquad tr \in (tr' \mathbin{|||} tr'')\}\\&\quad= \{(tr, X) \mid \exists\, tr', tr'' \mid (tr', X) \in \mathit{failures}(Q) \wedge\\&\qquad\qquad (tr'', X) \in \{(\langle\rangle, X) \mid X \subseteq A\} \wedge\\&\qquad\qquad tr \in (tr' \mathbin{|||} tr'')\}\\&\quad= \{(tr, X) \in \mathit{failures}(Q) \mid X \subseteq A\}. \qquad (2)\end{aligned}$$

Furthermore, since $A \neq \emptyset$,

$$failures(R_A) \;=\; \{(\langle\rangle, X) \mid X \subset A\}.$$

Hence, by definition of $\triangle$, we deduce that for any Q

$$\begin{aligned}
&failures((Q \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)\\
&\quad = \{(tr \frown tr', X) \mid tr \in traces(Q \mathbin{|||} Offer(\Sigma - A)) \wedge\\
&\qquad\qquad (tr', X) \in failures(R_A)\}\\
&\quad = \{(tr \frown tr', X) \mid tr \in traces(Q \mathbin{|||} Offer(\Sigma - A) \wedge tr' = \langle\rangle \wedge X \subset A\}\\
&\quad = \{(tr, X) \mid tr \in traces(Q \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\}. \qquad (3)
\end{aligned}$$

We are now equipped with the necessary building blocks to complete our proof.

$$\begin{aligned}
&(SPEC \mathbin{|||} Offer(\Sigma - A)) \rightsquigarrow R_A \sqsubseteq_{\mathcal{F}} (P \mathbin{|||} Offer(\Sigma - A)) \rightsquigarrow R_A\\
&\Leftrightarrow \qquad [\text{def}^{\,n} \text{ of } \rightsquigarrow]\\
&\quad (SPEC \mathbin{|||} Offer(\Sigma - A)) \sqcap ((SPEC \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)\\
&\qquad \sqsubseteq_{\mathcal{F}} (P \mathbin{|||} Offer(\Sigma - A)) \sqcap ((P \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)\\
&\Leftrightarrow \qquad [\text{def}^{\,n} \text{ of } \sqsubseteq_{\mathcal{F}}]\\
&\quad traces((P \mathbin{|||} Offer(\Sigma - A)) \sqcap ((P \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)) \subseteq\\
&\qquad traces((SPEC \mathbin{|||} Offer(\Sigma - A)) \sqcap\\
&\qquad\qquad ((SPEC \mathbin{|||} Offer(\Sigma - A)) \triangle R_A))\\
&\quad \wedge\\
&\quad failures((P \mathbin{|||} Offer(\Sigma - A)) \sqcap ((P \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)) \subseteq\\
&\qquad failures((SPEC \mathbin{|||} Offer(\Sigma - A)) \sqcap\\
&\qquad\qquad ((SPEC \mathbin{|||} Offer(\Sigma - A)) \triangle R_A))\\
&\Leftrightarrow \qquad [SPEC \sqsubseteq_{\mathcal{T}} P, \text{ monotonicity}]\\
&\quad failures((P \mathbin{|||} Offer(\Sigma - A)) \sqcap ((P \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)) \subseteq\\
&\qquad failures((SPEC \mathbin{|||} Offer(\Sigma - A)) \sqcap\\
&\qquad\qquad ((SPEC \mathbin{|||} Offer(\Sigma - A)) \triangle R_A))\\
&\Leftrightarrow \qquad [\text{def}^{\,n} \text{ of } \sqcap]\\
&\quad failures(P \mathbin{|||} Offer(\Sigma - A)) \cup failures((P \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)\\
&\qquad \subseteq\\
&\qquad failures(SPEC \mathbin{|||} Offer(\Sigma - A)) \cup\\
&\qquad\qquad failures((SPEC \mathbin{|||} Offer(\Sigma - A)) \triangle R_A)\\
&\Leftrightarrow \{(tr, X) \in failures(P) \mid X \subseteq A\} \;\cup \qquad [\text{Equations (2) and (3)}]\\
&\qquad\qquad \{(tr, X) \mid tr \in traces(P \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\} \subseteq\\
&\qquad \{(tr, X) \in failures(SPEC) \mid X \subseteq A\} \;\cup\\
&\qquad\qquad \{(tr, X) \mid tr \in traces(SPEC \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\}\\
&\Leftrightarrow \qquad [\text{set theory, def}^{\,n} \text{ of } |||, \text{ condition F1}]\\
&\quad \{(tr, X) \in failures(P) \mid X = A\} \;\cup\\
&\qquad\qquad \{(tr, X) \mid tr \in traces(P \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\} \subseteq\\
&\qquad \{(tr, X) \in failures(SPEC) \mid X = A\} \;\cup\\
&\qquad\qquad \{(tr, X) \mid tr \in traces(SPEC \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\}\\
&\Leftrightarrow \{(tr, X) \in failures(P) \mid X = A\} \subseteq \qquad [\text{set theory}]\\
&\qquad \{(tr, X) \in failures(SPEC) \mid X = A\}\\
&\quad \wedge\\
&\quad \{(tr, X) \mid tr \in traces(P \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\} \subseteq\\
&\qquad \{(tr, X) \mid tr \in traces(SPEC \mathbin{|||} Offer(\Sigma - A)) \wedge X \subset A\}\\
&\Leftrightarrow \qquad [SPEC \sqsubseteq_{\mathcal{T}} P, \text{ monotonicity, set theory}]\\
&\quad \{(tr, X) \in failures(P) \mid X = A\} \subseteq\\
&\qquad \{(tr, X) \in failures(SPEC) \mid X = A\}.
\end{aligned}$$

□

4 Potential deadlock

In this section we present a technique for determining automatically whether a process could reach a state of deadlock after a particular sequence of interaction, whilst, after the same sequence of interaction, its specification could not. This is a special case of the previous section, taking A to be the whole of Σ.

4.1 Practical application

A means of automatically verifying whether one system can deadlock when another would not is necessary whenever we are designing a system that might reach a deadlocked state but should do so only under certain conditions: for instance after reporting an error or successfully completing a task.

Consider an inference tool that can make a variety of deductive steps depending on its current knowledge. The specification might permit any possible deductive step from any given state whilst, for efficiency reasons, the implementation might prioritise techniques: all possible deductive steps of one class must be completed before any deductive steps of another class may be performed. We see that every possible trace of the implementation must be a possible trace of the specification, but are not concerned that the implementation will have more refusals than the specification. Our concern is that whenever the implementation can make no more deductive steps and hence deadlocks, the specification might also deadlock. We need to verify that

$$\begin{aligned}&\{(tr, X) \in \mathit{failures}(\mathit{InferenceToolImpl}) \mid X = \Sigma\}\\ &\qquad \subseteq\ \{(tr, X) \in \mathit{failures}(\mathit{InferenceToolSPEC}) \mid X = \Sigma\},\end{aligned}$$

and hence need the result presented in Theorem 2 below.

4.2 Automatic verification

The following predicate captures that there is no trace of process P after which it can deadlock, but its specification SPEC cannot:

$$\{(tr, X) \in \mathit{failures}(P) \mid X = \Sigma\}\ \subseteq\ \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid X = \Sigma\}.$$

In order to ascertain automatically whether this is or is not the case, we must equivalently express the above predicate in terms of refinement checks within established semantic models. To achieve this we must flood the failures sets of both P and SPEC with all trace refusal pairs (tr, X) such that tr is a trace of the process and X is a strict subset of Σ. As we show in Theorem 2 below, a corollary to Theorem 1 taking A to be the whole of Σ, this can be done by non-deterministically interrupting with an appropriate process.

Theorem 2 For any process P defined over finite set of events Σ and with specification SPEC such that $\mathit{SPEC} \sqsubseteq_{\mathcal{T}} P$, there is no trace of P after which it might deadlock but after which SPEC will not deadlock precisely when

$$\mathit{Spec} \rightsquigarrow R_\Sigma \ \sqsubseteq_{\mathcal{F}}\ P \rightsquigarrow R_\Sigma,$$

where $R_\Sigma = \bigsqcap Y \mid Y \subset \Sigma \bullet \mathit{Offer}(\Sigma - Y)$. $\Box$

5 Refusal of a given number of events

In this section we present a technique for determining automatically whether, for any given integer $N > 0$, a process could refuse any set of N or more events after a particular sequence of interaction, whilst, after the same sequence of interaction, its specification could not.

5.1 Practical application

Verifying that a set of at least N events can be refused by a process only when its specification could also refuse those events is of particular interest when we are considering support or safety systems, systems that under usual circumstances require a built-in margin for error.

A large office has an indexed set of E emergency exits. The Health and Safety Division of the company are required to do an annual check to verify that, provided a fire does not simultaneously break out in more than F places, there are at least R escape routes. Equivalently, no more than $E - R$ exits can be blocked unless a fire has broken out in more than F places.

Suppose that a process *Layout* models the current layout of the building along with access to exits, and that *Fires* is a process that non-deterministically chooses to block certain routes. The Health and Safety executives construct a process

$$\mathit{Scenario} \;\widehat{=}\; (\mathit{Layout} \underset{\mathit{Routes}}{\|} \mathit{Fires}) \setminus \mathit{Routes}$$

that determines which exits are accessible given the current layout and the non-deterministic blocking of routes by *Fires*. They then construct a process that specifies that if fewer than F fires have broken out, then *all* exits are accessible (the parameter $\mathit{fs} \subseteq \mathit{Fire}$ represents the set of fires that have broken out):

$$\begin{aligned} \mathit{SafetySpec} \;&\widehat{=}\; \mathit{SS}(\{\}), \\ \mathit{SS}(\mathit{fs}) \;&\widehat{=}\; \Box\, f : \mathit{Fire} \bullet \mathit{fire}.f \rightarrow \mathit{SS}(\mathit{fs} \cup \{f\}) \\ &\quad\; \Box \\ &\quad\; (\text{if } \#\mathit{fs} \leqslant F \text{ then } \Box\, i : \mathit{Index} \bullet \mathit{accessible}.i \rightarrow \mathit{SS}(\mathit{fs}) \text{ else } \mathit{STOP}). \end{aligned}$$

Since they are only interested in cases where more than $E - R$ exits are blocked (and *fire* events are never refused by *SafetySpec*), they need to verify that the following condition is met:

$$\{(tr, X) \in \mathit{Scenario} \mid \#X > E - R\} \;\subseteq\; \{(tr, X) \in \mathit{SafetySpec} \mid \#X > E - R\}.$$

The result presented in Theorem 3 would facilitate such a check.

5.2 Automatic verification

For any strictly positive integer N, the following predicate captures that process P can refuse a set of at least N events only when *SPEC* can refuse the same set of events:

$$\{(tr, X) \in \mathit{failures}(P) \mid \#X \geq N\} \;\subseteq\; \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid \#X \geq N\}.$$

We can perform this check using existing tools by masking all failures information where the cardinality of the refusal set is strictly less than N. We do this by using the non-deterministic interrupt operator to flood both sides with all such failures. The proof is included in the appendix.

Theorem 3 For any process P defined over finite set of events Σ and with specification *SPEC* such that $\mathit{SPEC} \sqsubseteq_{\mathcal{T}} P$, and for any strictly positive integer N, there is no trace of P after which it might refuse a set X such that $\#X \geq N$ but after which *SPEC* will not refuse X precisely when

$$\mathit{SPEC} \rightsquigarrow S_N \;\sqsubseteq_{\mathcal{F}}\; P \rightsquigarrow S_N,$$

where $S_N = \bigsqcap Y : \mathbb{P}\,\Sigma \mid \#Y < N \bullet \mathit{Offer}(\Sigma - Y)$. $\Box$

6 Refusal of a given number of events of a specific class

In this section we present a technique for determining automatically whether or not, for some strictly positive integer N, a process could refuse a set of N or more events from a particular set or class, whilst its specification could not, after the same sequence of interaction, refuse that same set of events.

6.1 Potential application

A means of automatically verifying whether one system can refuse at least a certain number of events of a given class when another could not is of particular interest when we are modelling distributed systems involving the parallel combination or interleaving of an indexed collection of identical components.

Suppose that a company is running the software for a large safety-critical distributed system comprising a dynamic network of R indexed nodes. The requirements imposed on the company state that (modulo an agreed latency period after any node crashes) the central monitor must always be able to communicate with at least k nodes. The ability of the central monitor to be able to communicate with node $i \in R$ is modelled by the event $inContact.i$.

Suppose that the process $Network$ below satisfies the necessary constraints:

$$Network \;\widehat{=}\; MonitorProcess \underset{\{inContact.i \mid i \in R\}}{\|} Nodes.$$

The company now decides, for whatever reason, to update its nodes:

$$NewNetwork \;\widehat{=}\; MonitorProcess \underset{\{inContact.i \mid i \in R\}}{\|} UpdatedNodes$$

but then needs to verify that the new system can communicate with at least k of its processors at least as often as the old system could. They need to verify that whenever the new system can refuse $N = R - K + 1$ or more $inContact$ events the old system could also have refused these events. They need to verify that

$$\begin{aligned}&\{(tr, X) \in failures(NewNetwork) \mid \#X \geq N \wedge X \subseteq \{contact.i \mid i \in R\}\}\\ &\quad\subseteq \{(tr, X) \in failures(Network) \mid \#X \geq N \wedge X \subseteq \{contact.i \mid i \in R\}\}.\end{aligned}$$

We prove in Theorem 4 below how such a verification might be performed automatically using existing tools.

6.2 Automatic verification

Given set of events $A \subseteq \Sigma$ and integer $N > 0$, the following predicate captures that there is no trace of process P after which it can refuse a subset of A of size N or greater but its specification $SPEC$ cannot:

$$\begin{aligned}&\{(tr, X) \in failures(P) \mid X \subseteq A \wedge \#X \geq N\}\\ &\quad\subseteq \{(tr, X) \in failures(SPEC) \mid X \subseteq A \wedge \#X \geq N\}.\end{aligned}$$

To verify automatically whether or not this is the case, we must reduce the problem to one expressed in terms of refinement within the established models. We must retain failures information when the refusal set is a subset of A of cardinality N or more, and we must mask all other failures information: we remove all failures in which the refusal set is not a subset of A and we flood the failures sets of both P and $SPEC$ with all trace refusal pairs (tr, X) such that tr is a trace of the process and X is a subset of Σ of size strictly less than N.

As we show in Theorem 4 below, this can be done through a combination of interleaving and non-deterministically interrupting with the appropriate processes. The proof is included in the appendix.

Theorem 4 For any process P defined over finite set of events Σ and with specification $SPEC$ such that $SPEC \sqsubseteq_{\mathcal{T}} P$, for any strictly positive integer N and for any given set of events $A \neq \emptyset$, P can refuse a set of N events from A only if $SPEC$ can precisely when

$$(SPEC \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N} \ \sqsubseteq_{\mathcal{F}} \ (P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N},$$

where $T_{A,N} = \bigsqcap Y \mid \#Y < N \wedge Y \subseteq A \bullet \mathit{Offer}(\Sigma - Y)$. $\Box$

7 Discussion

In this paper we have considered various non-standard measures for determining whether a process modelled in CSP is consistent with its specification, presenting motivating examples in which the full strength of the stable failures model is too prescriptive but traces model not prescriptive enough.

The first situation we considered was one is which we were concerned only about the potential refusal of a particular set of events: we wished to verify that a process could refuse a fixed set of events after any trace only if its specification could refuse the same set after the same trace. The criterion for consistency in our second model was that a process might deadlock after any trace only if its specification might also deadlock after the same trace. Our third measure was to ascertain whether a process could refuse a set of at least N events when its specification could not. And the final situation we considered was one in which we were concerned about the potential to refuse N or more events from a given set.

For each measure we identifed various problem domains in which the proposed measure of consistency would be useful and appropriate. Furthermore, in each case we explored in greater detail a particular example within one of these domains. We expressed the predicate defining consistency for each measure in terms of refinement checks within the traces model and the stable failures model, thereby enabling automatic verification of consistency using existing tools. Finally, we proved formally the soundness and completeness of each of these results.

The work presented here has built upon and developed the very abstract ideas presented in [BL03]. There we identified a family, in fact a complete lattice, of failures models each recording all trace information and some failures information. The amount of failures information recorded in each model was determined by its associated predicate on refusal sets. In this paper we have taken a further step, considering practical applications of some of these models and presenting explicitly their associated refinement checks.

An interesting corollary of a result in that paper is that the preorders corresponding to the measures of consistency in Sections 4 and 5 are precongruences with respect to all the CSP operators except for parallel composition; while the preorders corresponding to the measures of consistency in Sections 3 and 6 are precongruences with respect to all the CSP operators except for parallel composition and hiding. In other words, for any CSP context $F(\_)$ built using CSP operators other than (respectively) parallel composition or parallel composition and hiding, if P satisfies the condition then so does $F(P)$. This allows a degree of compositional reasoning.

In [Ros03b], Roscoe considers which predicates R over the semantics of processes can be captured as refinement checks. He shows that all predicates that are closed under the metric topology can be captured as refinement checks. He gives simplified refinement checks in the cases that R is refinement-closed (i.e. such that if P satisfies R and $P \sqsubseteq Q$ then Q satisfies R) and/or distributive (i.e. such that if each process in the set S satisfies R, then so does $\bigsqcap S$). However, his refinement checks are not, in general, finite-state, and so cannot be automated using a tool such as FDR.

It is interesting to observe that the techniques for verifying consistency presented in this paper are not unique: other operators and combinations could be used to mask the unwanted failures. Indeed, as observed in Section 3, Roscoe has come up with an alternative method for determining whether a process can refuse a set of events A only when its specification could refuse the same set of events [Ros03a]. His work arose from a problem posed by Reed and Sinclair following on from [RS01] in which their predicate "is as live as" could be verified using techniques similar to those presented in Section 3.

Roscoe's technique is as follows. Let $e \notin \Sigma$ be a fresh event, and let R_A be the renaming relation that maps each element a of A to both itself and e, and is otherwise the identity function. Let

$$\begin{aligned} \mathit{CheckAll}_A(Q) \;&\mathrel{\widehat{=}}\; Q\,[\![R_A]\!] \underset{\Sigma\cup\{e\}}{\parallel} \mathit{CantBlock\_e} \\ \mathit{CantBlock\_e} \;&\mathrel{\widehat{=}}\; (?x : \Sigma \rightarrow \mathit{CantBlock\_e} \sqcap \mathit{STOP}) \,\Box\, e \rightarrow \mathit{STOP}. \end{aligned}$$

$\mathit{CheckAll}_A(Q)$ can perform any trace of Q, but can also perform an e when Q can do an event from A, after which it must stop. After any trace it can refuse anything except for e; it can refuse e only when Q can refuse the whole of A. Hence the requirement that P refuses A only when SPEC refuses A can be captured by the following refinement check:

$$\mathit{CheckAll}_A(\mathit{SPEC}) \sqsubseteq_{\mathcal{F}} \mathit{CheckAll}_A(P).$$

This check and the one introduced in Section 3 take comparable times.

References

[BL03] C. Bolton and G. Lowe. A hierarchy of failures-based models, 2003. Submitted for publication.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronisation skeletons using branching-time temporal logic. In *Proceedings of the International Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, 1981.

[EH86] E. A. Emerson and J. Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[For99] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR 2 User Manual*, 1999. Available via URL `http://www.fsel.com/fdr2_manual.html`.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil89] R. Milner. *Communications and concurrency*. Prentice Hall, 1989.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall International, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th International Smposium on Foundations of Computer Science*, pages 46–57, 1977.

[RJB97] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language reference manual*. Addison-Wesley, 1997.

[Ros94] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[Ros03a] A. W. Roscoe. Personal Communication, 2003.

[Ros03b] A. W. Roscoe. On the expressiveness of CSP refinement checking. In *Proceedings of the Third Workshop on Automated Verification of Critical Systems (AVoCS 2003)*, 2003.

[RS01] J. N. Reed and J. E. Sinclair. Combining independent specifications. In *Proceedings of Fundamental Approaches to Software Engineering*, volume 2029 of *Lecture Notes in Computer Science*, 2001.

A Proofs

Proof of Theorem 3 First observe that since $N > 0$

$$failures(S_N) = \{(\langle\rangle, X) \mid \#X < N\}.$$

Hence, by definition of $\triangle$, we deduce that for any Q

$$\begin{aligned}
&failures(Q \triangle S_N) \\
&\quad = \{(tr \frown tr', X) \mid tr \in traces(Q) \wedge (tr', X) \in failures(S_N)\} \\
&\quad = \{(tr \frown tr', X) \mid tr \in traces(Q) \wedge tr' = \langle\rangle \wedge \#X < N\} \\
&\quad = \{(tr, X) \mid tr \in traces(Q) \wedge \#X < N\}. \qquad (4)
\end{aligned}$$

We now reason as follows:

$$\begin{aligned}
&SPEC \rightsquigarrow S_N \sqsubseteq_{\mathcal{F}} P \rightsquigarrow S_N \\
&\Leftrightarrow \; SPEC \sqcap (SPEC \triangle S_N) \sqsubseteq_{\mathcal{F}} P \sqcap (P \triangle S_N) \qquad [\text{def}^n \text{ of } \rightsquigarrow] \\
&\Leftrightarrow \qquad [\text{def}^n \text{ of } \sqsubseteq_{\mathcal{F}}] \\
&\quad traces(P \sqcap (P \triangle S_N)) \subseteq traces(SPEC \sqcap (SPEC \triangle S_N)) \\
&\quad \wedge \\
&\quad failures(P \sqcap (P \triangle S_N)) \subseteq failures(SPEC \sqcap (SPEC \triangle S_N)) \\
&\Leftrightarrow \qquad [SPEC \sqsubseteq_{\mathcal{T}} P, \text{ monotonicity, def}^n \text{ of } \sqcap] \\
&\quad failures(P) \cup failures(P \triangle S_N)) \subseteq \\
&\qquad failures(SPEC) \cup failures(SPEC \triangle S_N)) \\
&\Leftrightarrow \qquad [\text{Equation (4)}] \\
&\quad failures(P) \cup \{(tr, X) \mid tr \in traces(P) \wedge \#X < N\} \subseteq \\
&\qquad failures(SPEC) \cup \{(tr, X) \mid tr \in traces(SPEC) \wedge \#X < N\}
\end{aligned}$$

$$
\begin{aligned}
&\Leftrightarrow && \text{[set theory]}\\
&\quad \{(tr, X) \in \mathit{failures}(P) \mid \#X \geq N\} \subseteq \\
&\qquad \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid \#X \geq N\}\\
&\quad \wedge\\
&\quad \{(tr, X) \mid tr \in \mathit{traces}(P) \wedge \#X < N\} \subseteq \\
&\qquad \{(tr, X) \mid tr \in \mathit{traces}(\mathit{SPEC}) \wedge \#X < N\}\\
&\Leftrightarrow && [\mathit{SPEC} \sqsubseteq_{\mathcal{T}} P, \text{ set theory}]\\
&\quad \{(tr, X) \in \mathit{failures}(P) \mid \#X \geq N\} \subseteq \\
&\qquad \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid \#X \geq N\}.
\end{aligned}
$$

Proof of Theorem 4 First observe that, since $N > O$,

$$
\mathit{failures}(T_{A,N}) = \{(\langle\rangle, X) \mid \#X < N \wedge X \subseteq A\}.
$$

Hence, by definition of $\triangle$, we deduce that for any Q

$$
\begin{aligned}
&\mathit{failures}(Q \triangle T_{A,N})\\
&\quad = \{(tr \frown tr', X) \mid tr \in \mathit{traces}(Q) \wedge (tr', X) \in \mathit{failures}(T_{A,N})\}\\
&\quad = \{(tr \frown tr', X) \mid tr \in \mathit{traces}(Q) \wedge tr' = \langle\rangle \wedge \#X < N \wedge X \subseteq A\}\\
&\quad = \{(tr, X) \mid tr \in \mathit{traces}(Q) \wedge \#X < N \wedge X \subseteq A\}. \qquad (5)
\end{aligned}
$$

We now reason as follows:

$$
\begin{aligned}
&(\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N} \sqsubseteq_{\mathcal{F}} (P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N}\\
&\Leftrightarrow && [\mathrm{def}^{\,n} \text{ of } \sqsubseteq_{\mathcal{F}}]\\
&\quad \mathit{traces}((P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N}) \subseteq\\
&\qquad \mathit{traces}((\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N})\\
&\quad \wedge\\
&\quad \mathit{failures}((P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N}) \subseteq\\
&\qquad \mathit{failures}((\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N})\\
&\Leftrightarrow && [\mathit{SPEC} \sqsubseteq_{\mathcal{T}} P, \text{ monotonicity}]\\
&\quad \mathit{failures}((P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N}) \subseteq\\
&\qquad \mathit{failures}((\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \rightsquigarrow T_{A,N})\\
&\Leftrightarrow && [\mathrm{def}^{\,n} \text{ of } \rightsquigarrow, \sqcap]\\
&\quad \mathit{failures}(P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \cup\\
&\qquad \mathit{failures}((P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \triangle T_{A,N})\\
&\qquad \subseteq\\
&\qquad \mathit{failures}(\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \cup\\
&\qquad\quad \mathit{failures}((\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \triangle T_{A,N})\\
&\Leftrightarrow \{(tr, X) \in \mathit{failures}(P) \mid X \subseteq A\} \cup && \text{[Equations (2) and (5)]}\\
&\qquad \{(tr, X) \mid tr \in \mathit{traces}(P \mid\mid\mid \mathit{Offer}(\Sigma - A)) \wedge\\
&\qquad\qquad \#X < N \wedge X \subseteq A\}\\
&\qquad \subseteq\\
&\qquad \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid X \subseteq A\} \cup\\
&\qquad\quad \{(tr, X) \mid tr \in \mathit{traces}(\mathit{SPEC} \mid\mid\mid \mathit{Offer}(\Sigma - A)) \wedge\\
&\qquad\qquad\quad \#X < N \wedge X \subseteq A\}
\end{aligned}
$$

$$
\begin{array}{ll}
\Leftrightarrow & \hfill [\text{set theory, def}^{\,n} \text{ of } |||, \text{condition F1}] \\
& \{(tr, X) \in \mathit{failures}(P) \mid \#X \geq N \wedge X \subseteq A\} \;\cup \\
& \qquad \{(tr, X) \mid tr \in \mathit{traces}(P \;|||\; \mathit{Offer}(\Sigma - A)) \;\wedge \\
& \qquad\qquad\qquad \#X < N \wedge X \subseteq A\} \\
& \qquad \subseteq \\
& \qquad \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid \#X \geq N \wedge X \subseteq A\} \;\cup \\
& \qquad\qquad \{(tr, X) \mid tr \in \mathit{traces}(\mathit{SPEC} \;|||\; \mathit{Offer}(\Sigma - A)) \;\wedge \\
& \qquad\qquad\qquad\qquad \#X < N \wedge X \subseteq A\}
\end{array}
$$

$$
\begin{array}{ll}
\Leftrightarrow & \hfill [\text{set theory}] \\
& \{(tr, X) \in \mathit{failures}(P) \mid \#X \geq N \wedge X \subseteq A\} \subseteq \\
& \qquad \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid \#X \geq N \wedge X \subseteq A\} \\
& \wedge \\
& \{(tr, X) \mid tr \in \mathit{traces}(P \;|||\; \mathit{Offer}(\Sigma - A)) \wedge \#X < N \wedge X \subseteq A\} \;\subseteq \\
& \qquad \{(tr, X) \mid tr \in \mathit{traces}(\mathit{SPEC} \;|||\; \mathit{Offer}(\Sigma - A)) \wedge \\
& \qquad\qquad\qquad \#X < N \wedge X \subseteq A\}
\end{array}
$$

$$
\begin{array}{ll}
\Leftrightarrow & \hfill [\mathit{SPEC} \sqsubseteq_{\mathcal{T}} P, \text{ monotonicity, set theory}] \\
& \{(tr, X) \in \mathit{failures}(P) \mid \#X \geq N \wedge X \subseteq A\} \subseteq \\
& \qquad \{(tr, X) \in \mathit{failures}(\mathit{SPEC}) \mid \#X \geq N \wedge X \subseteq A\}.
\end{array}
$$