Finite Complete Suites for CSP Refinement Testing

Ana Cavalcanti[1], Wen-ling Huang[2], and Jan Peleska[2]

[1] University of York, United Kingdom
ana.cavalcanti@york.ac.uk
[2] University of Bremen, Germany
{peleska,huang}@uni-bremen.de

Abstract. In this paper, two new contributions for model-based testing with Communicating Sequential Processes (CSP) are presented. For a given CSP process representing the reference model, test suites checking the conformance relations trace refinement and failures refinement are constructed, and their finiteness and completeness (that is, capability to uncover conformity violations) is proven. While complete test suites for CSP processes have been previously investigated by several authors, a sufficient condition for their finiteness is presented here for the first time. It is shown that the test suites are optimal with respect to two aspects: (a) the maximal length of test traces cannot be reduced, and (b) the nondeterministic behaviour cannot be tested with smaller or fewer sets of events, without losing the test suite's completeness property.

Keywords: Model-based testing, CSP, Trace Refinement, Failures Refinement, Complete Test Suites

FiXme Fatal: alcc: Is the technique applying the test suites?
FiXme Note: jp: this sentence needed to be re-phrased – the new version should be better understandable

1 Introduction

Motivation Model-based testing (MBT) is an active research field; results are currently being evaluated and integrated into industrial verification processes by many companies worldwide. This holds particularly for the embedded and cyber-physical systems domains, where critical systems require rigorous testing.

In the safety-critical domain, test suites with guaranteed fault coverage are of particular interest. For black-box testing, guarantees can be given only if certain hypotheses are satisfied. These hypotheses are usually specified by a *fault domain*: a set of models that may or may not conform to a given reference model. *Complete* test strategies guarantee to accept every system under test (SUT) conforming to the reference model, and uncover every conformance violation, provided that the SUT behaviour is captured by a member of the fault domain.

Generation techniques for complete test suites have been developed for various formalisms. Here, we tackle *Communicating Sequential Processes (CSP)* [10,19]. This is a mature process algebra that has been shown to be well-suited for the description of reactive control systems in many publications over almost five decades. Many of these applications are described in [19] and in the references there. Industrial success has also been reported; see, for example, [7,21,1].

Contributions This paper presents complete black-box test suites for software and systems modelled using CSP. They can be generated for non-terminating, divergence-free, finite-state CSP processes with finite alphabets, interpreted both in the trace and the failures semantics. Divergence freedom is usually assumed in black-box testing, since it cannot distinguish between divergence and deadlock.

Our results complement work in [3]. There, fault domains are specified as collections of processes refining a "most general" fault domain member. With that concept, complete test suites may be finite or infinite. This result gives important insight into the theory of fault-domain testing for CSP, but we are particularly interested in *finite* suites when it comes to practical application. While [3] may require additional criteria to select tests from still infinite test suites, here, we further restrict fault domains using a graph representation of processes (originally elaborated for model checking) to obtain finite test suites.

Our approach to the definition of CSP fault domains is presented in this paper. We take advantage of results on model checking of CSP processes, where the failures semantics of a finite-state CSP process is represented as a finite normalised transition graph, whose edges are labelled by the events the process engages in, and whose nodes are labelled by minimal acceptances or, alternatively, maximal refusals [17]. The maximal refusals express the degree of nondeterminism present in a process state that is in one-one-correspondence to a node of the normalised transition graph. Inspired by the way fault-domains are specified for finite state machines (FSMs), we define them here as the set of CSP processes whose normalised transition graphs do not exceed the order (that is, the number of nodes) of the reference model's graph by more than a given constant.

The main contribution of this paper is the construction of two test suites to verify the conformance relations *trace refinement* and *failures refinement* for any given non-terminating, non-divergent, finite-state, finite alphabet CSP process. We prove their completeness with respect to fault domains of the described type. The existence of – possibly infinite – complete test suites has been established for process algebras, and for CSP in particular, by several authors [8,20,14,12,2,3]. To the best of our knowledge, this article is the first to present *finite*, complete test suites associated with this class of fault domains and conformance relations.

FiXme Fatal: alcc: I'm slightly uncomfortable with this claim. Can the environment really control outputs? Do you not have tau (internal) events in FSM? I'm unsure, but you guys know about FSMs much more than I do. FiXme Note: jp: I think that this description is ok. With FSMs as used in the testing community, there are no tau events, and parallel composition $M_1 \cap M_2$ is just language intersection. Therefore, when composing M_1 and M_2 to $M_1 \cap M_2$, M_1 cannot refuse to engage into I/O-event x/y, if M_2 has x/y as the only initial event in its language.

Our result is not a simple transcription of existing knowledge about finite, complete test suites for FSMs. The capabilities of CSP to express nondeterminism in a more fine-grained way than it is possible for FSMs requires a more complex approach to testing for conformity in the failures model than required for model-based testing against nondeterministic FSMs, as published, for example, in [9,16]. CSP distinguishes between external choice, where the environment can control the behaviour of a process, and internal choice where the behaviour is decided internally and cannot be influenced by the environment. In contrast, FSMs specify nondeterministic behaviour by offering more than one possible output for a given input, and this output can be controlled by an FSM representing the environment. Loosely speaking, this corresponds to external choice in CSP, while there is no equivalent to internal choice in nondeterministic FSMs.

Finally, we prove two optimality results here. (1) We show that our approach to testing the admissibility of an SUT's nondeterministic behaviour is optimal in the sense that it cannot be established with fewer "probes" investigating the SUT's degree of nondeterminism. These probes are minimal sets of events offered by the tests to the SUT, such that a refusal of the SUT to engage into at least one of these events reveals a violation of failures refinement. (2) Furthermore, it is shown that the maximal length of traces to be investigated in a complete test suite cannot be further reduced without loosing the suite's capability to uncover any violation of trace or failures refinement.

Overview In Section 2, we present the background relevant to our work. In Section 3, finite complete test suites for verifying failures refinement are presented. Test suites checking trace refinement are a simplified version of the former class; they are presented in Section 5. The optimality results are presented in Section 6, together with further complexity considerations. A sample test suite is presented in Section 4. Our results are discussed in Section 7, where we also conclude. References to related work are given throughout the paper where appropriate.

2 Preliminaries

We present CSP (Section 2.1) and the concept of minimal hitting sets (Section 2.2), which is central to our novel notion of test for failures refinement. To study complexity, we also introduce the concept of Sperner families (Section 2.3).

2.1 CSP, Refinement, and Normalised Transition Graphs

Communicating Sequential Processes (CSP) is a process algebra supporting system development by refinement. Using CSP, we model both systems and their components using processes. They are characterised by their patterns of interactions, modelled by synchronous, instantaneous, and atomic events.

Throughout this paper, the alphabet of the processes, that is, the set of events that are in scope, is denoted by Σ and supposed to be finite. The FDR tool [5] supports model checking and semantic analyses of finite-state CSP processes.

A prefixing operator $e \rightarrow P$ defines a process that is ready to engage in the event e, pending agreement of its environment to synchronise. After e occurs, the process behaves as defined by P. The environment can be other processes, in parallel, or the environment of a system as a whole.

Two forms of choice support branching behaviour. An external choice $P \,\Box\, Q$ between processes P and Q offers to the environment the initial events of P and Q. Once synchronisation takes place, the process that has offered this event is chosen and the other is discarded. In an internal choice $P \sqcap Q$, the environment does not have an opportunity to interfere: the choice is made by the process.

Example 1. We consider P, Q, and R below. P is initially ready to engage in the event a, and then makes an internal choice to behave like either Q or R.

$$P = a \rightarrow (Q \sqcap R)$$
$$Q = a \rightarrow P \,\Box\, c \rightarrow P$$
$$R = b \rightarrow P \,\Box\, c \rightarrow R$$

Q, for instance, offers to the environment the choice to engage in a again or c. In both cases, afterwards, we have a recursion back to P. In R, if b is chosen, we also have a recursion back to P. If c is chosen, the recursion is to R. □

Iterated forms $\Box\, i : I \bullet P(i)$ and $\sqcap\, i : I \bullet P(i)$ of the external and internal choice operators define a choice over a collection of processes $P(i)$. If the index set I is empty, the external choice is the process *Stop*, which deadlocks: does not engage into any event or terminate. For an external choice $\Box\, e : A \bullet e \rightarrow P(e), A \subseteq \Sigma$ over a set A of events, we use the abbreviation $e : A \rightarrow P(e)$. An iterated internal choice is not defined for an empty index set.

There are several parallelism operators. A widely used form of parallelism $P \,|[\, cs \,]|\, Q$ defines a process in which the behaviour is characterised by those of P and Q in parallel, synchronising on the events in the set cs. Other forms of parallelism available in CSP can be defined using this operator.

Interactions that are not supposed to be visible to the environment can be hidden. The operator $P \setminus H$ defines a process that behaves as P, with the interactions modelled by events in the set H hidden. Frequently, hiding is used in conjunction with parallelism: it is often desirable to make actions of each process in a network of parallel processes, perhaps used for coordination of the network, invisible, while events happening at their interfaces remain observable.

A rich collection of process operators allows us to define networks of parallel processes in a concise and elegant way, and reason about safety, liveness, and divergences. A comprehensive account of the notation is given in [19].

A distinctive feature of CSP is its treatment of refinement (as opposed to bisimulation), which is convenient for reasoning about program correctness, due to its treatment of nondeterminism and divergence. A variety of semantic models capture different notions of refinement. The simplest model characterises a process by its possible *traces*; the set $traces(P)$ denotes the sequences of (non-hidden) events in which P can engage. We say that a process P *is trace-refined by another process* Q, written $P \sqsubseteq_T Q$, if $traces(Q) \subseteq traces(P)$.

In fact, in every semantic model, subset containment is used to define refinement. The model we focus on first is the failures model, which captures both sequences of interactions and deadlock behaviour. A *failure* of a process P is a pair (s, X) containing a trace s of P and a *refusal*: a set X of events in which P may refuse to engage, after having performed the events of s. The failures model of a process P records all its failures in a set $failures(P)$.

Semantic definitions specify, for each operator, how the traces or failures of the resulting process can be calculated from the traces or failures of each operand.

For example, for internal choice, $\mathit{failures}(P \sqcap Q) = \mathit{failures}(P) \cup \mathit{failures}(Q)$; see [18, p. 210] for a comprehensive list of these definitions.

Using the notation P/s to denote the process P after having engaged into trace s, the set $\mathit{Ref}(P/s) \mathrel{\widehat{=}} \{\, X \mid (s, X) \in \mathit{failures}(P) \,\}$ contains the refusals of P after s. Refusals are subset-closed [10,19]: if (s, X) is a failure of P and $Y \subseteq X$, then $(s, Y) \in \mathit{failures}(P)$ and $Y \in \mathit{Ref}(P/s)$ follows.

For divergence-free processes, failures refinement, $P \sqsubseteq_F Q$, is defined as expected by set containment $\mathit{failures}(Q) \subseteq \mathit{failures}(P)$. Since refusals are subset-closed, $P \sqsubseteq_F Q$ implies $(s, \varnothing) \in \mathit{failures}(P)$ for all traces $s \in \mathit{traces}(Q)$. So, for divergences-free processes, failures refinement implies trace refinement. Therefore, using the conformance relation *conf* below

$$Q \ \mathit{conf} \ P \mathrel{\widehat{=}} \forall\, s \in \mathit{traces}(P) \cap \mathit{traces}(Q) : \mathit{Ref}(Q/s) \subseteq \mathit{Ref}(P/s), \tag{1}$$

failures refinement can be expressed by $\sqsubseteq_T$ and *conf*.

$$(P \sqsubseteq_F Q) \Leftrightarrow (P \sqsubseteq_T Q \wedge Q \ \mathit{conf} \ P) \tag{2}$$

For finite CSP processes, since refusals are subset-closed, $\mathit{Ref}(P/s)$ can be constructed from the set of *maximal refusals* defined by

$$\mathit{maxRef}(P/s) = \{R \in \mathit{Ref}(P/s) \mid \forall\, R' \in \mathit{Ref}(P/s) - \{R\} : R \not\subseteq R'\} \tag{3}$$

Conversely, with the maximal refusals $\mathit{maxRef}(P/s)$ at hand, we can reconstruct the refusals in the set $\mathit{Ref}(P/s)$ as described by

$$\mathit{Ref}(P/s) = \{R' \in \mathbb{P}(\Sigma) \mid \exists\, R \in \mathit{maxRef}(P/s) : R' \subseteq R\}. \tag{4}$$

Deterministic process states P/s have exactly the one maximal refusal defined by $\Sigma - [P/s]^0$, where $[P/s]^0$ denotes the *initials* of P/s, that is, the events that P/s may engage into. The maximal refusals in combination with the initials of a process express its degree of nondeterminism.

Example 2. $P = (\mathit{Stop} \sqcap Q)$ has maximal refusals $\mathit{maxRef}(P) = \{\Sigma\}$, because *Stop* refuses to engage in any event, and this is carried over to P by the internal choice. However, P is distinguished from *Stop* by its initials, which are defined by $[P]^0 = [\mathit{Stop} \sqcap Q]^0 = [Q]^0$. So P may engage nondeterministically in any initial event of Q, but also refuse everything, due to internal selection of *Stop*.

Assuming an alphabet $\Sigma = \{a, b, c, d\}$, the process

$$Q = (e : \{a, b\} \to \mathit{Stop}) \sqcap (e : \{c, d\} \to \mathit{Stop})$$

has maximal refusals $\mathit{maxRef}(Q) = \{\{c, d\}, \{a, b\}\}$ and initials $[Q]^0 = \Sigma$. In contrast to P, nondeterminism is reflected here by two maximal refusals. □

Normalised Transition Graphs for CSP Processes As shown in [17], the failures semantics of any finite-state CSP process P can be represented by a *normalised transition graph* $G(P)$ defined by a tuple

$$G(P) = (N, \underline{n}, \Sigma, t : N \times \Sigma \nrightarrow N, r : N \to \mathbb{PP}(\Sigma)),$$

with nodes N, initial node $\underline{n} \in N$, and process alphabet Σ. The partial *transition function* t maps a node n and an event $e \in \Sigma$ to its successor node $t(n, e)$. If (n, e) is in the domain of t, then there is a transition, that is, an outgoing edge, from n with label e, leading to node $t(n, e)$. Normalisation of $G(P)$ is reflected by the fact that t is a function. The graph construction in [17] implies that all nodes n in N are reachable by sequences of edges labelled by $e_1 \dots e_k$ and connecting states $\underline{n}, n_1, \dots, n_{k-1}, n$, such that

$$n_1 = t(\underline{n}, e_1), \quad n_i = t(n_{i-1}, e_i),\ i = 2, \dots, k-1, \quad n = t(n_{k-1}, e_k).$$

By construction, $s \in \Sigma^*$ is a trace of P, if, and only if, there is a path through $G(P)$ starting at $\underline{n}$ whose edge labels coincide with s. In analogy to $traces(P)$, we use the notation $traces(G(P))$ for the set of finite, initialised paths through $G(P)$, each path represented by its finite sequence of edge labels. We note that $traces(P) = traces(G(P))$. Since $G(P)$ is normalised, there is a unique node reached by following the events from s one by one, starting in $\underline{n}$. Therefore, $G(P)/s$ is also well defined. By $[n]^0$ we denote the *initials* of n: the set of events occurring as labels in any outgoing transitions.

$$[n]^0 = \{e \in \Sigma \mid (n, e) \in \text{dom } t\}$$

The graph construction from [17] used to define $G(P)$ for any process P guarantees that $[G(P)/s]^0 = [P/s]^0$ for all traces s of P.

The total function r maps each node n to a non-empty set of (possibly empty) subsets of Σ. The graph construction guarantees that $r(G(P)/s)$ represents the maximal refusals of P/s for all $s \in traces(P)$. As a consequence,

$$(s, X) \in failures(P) \Leftrightarrow s \in traces(G(P)) \wedge \exists\, R \in r(G(P)/s) : X \subseteq R, \qquad (5)$$

so $G(P)$ allows us to re-construct the failures semantics of P.

Acceptances When investigating tests for failures refinement, the notion of *acceptances*, which is dual to refusals, is also useful. While the original introduction of acceptances presented in [8, pp. 75] was independent of refusals, we use the definition from [18, pp. 278]. A *minimal acceptance* of a CSP process state P/s is the complement of a maximal refusal of the same state. The set of minimal acceptances of P/s is denoted by $minAcc(P/s)$ and formally defined as

$$minAcc(P/s) = \{\Sigma - R \mid R \in maxRef(P/s)\} \qquad (6)$$

With this definition, a (not necessarily minimal) *acceptance* of P/s is a superset of some minimal acceptance and a subset of the initials $[P/s]^0$. Denoting the acceptances of P/s by $Acc(P/s)$, this leads to the formal definition

$$Acc(P/s) = \{B \subseteq [P/s]^0 \mid \exists\, A \in minAcc(P/s) : A \subseteq B\} \qquad (7)$$

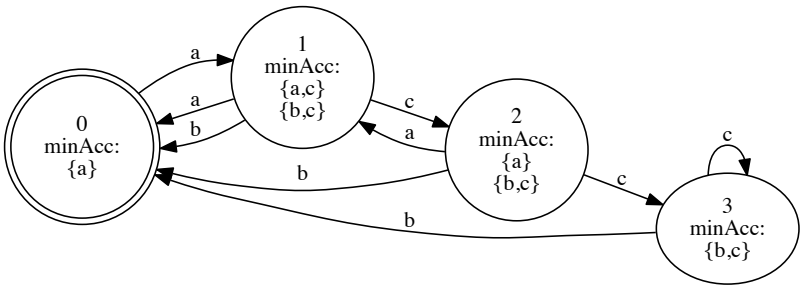

Fig. 1. Normalised transition graph of CSP process P from Example 3.

Acceptances have the following intuitive interpretation. If the behaviour of P/s is deterministic, its only acceptance equals $[P/s]^0$, because P/s never refuses any of the events contained in this set. If P/s is nondeterministic, it internally chooses one of its *minimal acceptance sets* A and never refuses any event in A, while *possibly* refusing the events from $[P/s]^0 - A$ and *always* refusing those from $\Sigma - [P/s]^0$.

Exploiting (6), the nodes of a normalised transition graph can alternatively be labelled with their minimal acceptances, and this information is equivalent to that contained in the maximal refusals. Since process states P/s are equivalently expressed by states $G(P)/s$ of P's normalised transition graph, we also write $minAcc(G(P)/s)$ and note that (5) and (6) imply

$$minAcc(G(P)/s) = \{\Sigma - R \mid R \in r(G(P)/s)\} = minAcc(P/s). \tag{8}$$

Given any non-diverging, non-terminating, finite-state process P, the process can be re-constructed from its transition graph $G(P)$ with initial state $\underline{n}$ and transition function t, using P's normalised syntactic representation [18, pp. 277] specified as follows.

$$\begin{aligned} \text{normalised}(P) &= P_N(\underline{n}) \\ P_N(n) &= \prod_{A \in minAcc(n) \cup \{[n]^0\}} e : A \to P_N(t(n,e)) \end{aligned}$$

With this definition, P is semantically equivalent to normalised(P) in the CSP failures semantics.

Example 3. We consider the process P in Example 1; its transition graph $G(P)$ is shown in Fig. 1. The process state P/ε (where ε denotes the empty trace) is represented as node 0, with $\{a\}$ as the only minimal acceptance, since a is never

FiXme Fatal: alcc: proof in the report perhaps? What do you need (5) for?

FiXme Note: jp: I have clarified the introduction of acceptances and removed the statements we do not need in the rest of the paper. We need (5) here to show that the set of maximal refusals of P/s equals $r(G(P/s))$.

refused and no other events are accepted. Having engaged in a, the transition from node 0 leads to node 1 representing the process state $P/a = Q \sqcap R$. The internal choice induces several minimal acceptances derived from Q and R. Since these processes accept their initial events in external choice, $Q \sqcap R$ induces minimal acceptance sets $\{a, c\}$ and $\{b, c\}$. We note that the event c can never be refused, since it is contained in each minimal acceptance set.

FiXme Fatal: alcc: the a from node 2 should go back to 0, I think.

Having engaged in c, the next process state is represented by node 2. Due to normalisation, there is only a single transition satisfying $t(1, c) = 2$. This transition, however, can have been caused by either Q or R engaging into c, so node 2 corresponds to process state $Q/c \sqcap R/c = P \sqcap R$. This is reflected by the two minimal acceptance sets labelling node 2. From node 2, event c leads to node 3. Since P does not engage into c, the R-component of $P \sqcap R$ must have processed c, so node 3 corresponds to process state $R/c = R$, and therefore it is labelled by R's minimal acceptance $\{b, c\}$. □

FiXme Note: jp: No, this is correct: state 0 corresponds to P, 1 to $Q \sqcap R$, 2 to $P \sqcap R$, and 3 to R. Therefore, 2 goes with a to $(P/a) = Q \sqcap R$, that is, to node 1. I think that this is now explained sufficiently clear in the example.

Summarising, refinement relations between finite-state CSP processes P, Q can be be expressed by means of their normalised transition graphs

$$G(P) = (N_P, \underline{n}_P, \Sigma, t_P : N_P \times \Sigma \nrightarrow N_P, r_P : N_P \to \mathbb{PP}(\Sigma))$$
$$G(Q) = (N_Q, \underline{n}_Q, \Sigma, t_Q : N_Q \times \Sigma \nrightarrow N_Q, r_Q : N_Q \to \mathbb{PP}(\Sigma))$$

as established by the results in the following lemma. There, result (9) reflects trace refinement in terms of graph traces; (10) expresses failures refinement in terms of traces refinement and *conf*; (11) states how *conf* can be expressed by means of the maximal refusal functions of the graphs involved; and (12) states the same in terms of the minimal acceptances that can be derived from the maximal refusal functions by means of (8).

Lemma 1.

$$P \sqsubseteq_T Q \Leftrightarrow \mathit{traces}(G(Q)) \subseteq \mathit{traces}(G(P)) \tag{9}$$
$$P \sqsubseteq_F Q \Leftrightarrow P \sqsubseteq_T Q \wedge Q \ \mathit{conf}\ P \tag{10}$$
$$Q \ \mathit{conf}\ P \Leftrightarrow \forall\, s \in \mathit{traces}(G(Q)) \cap \mathit{traces}(G(P)), R_Q \in r_Q(G(Q)/s) : \exists\, R_P \in r_P(G(P)/s) : R_Q \subseteq R_P \tag{11}$$
$$\Leftrightarrow \forall\, s \in \mathit{traces}(G(Q)) \cap \mathit{traces}(G(P)), A_Q \in \mathit{minAcc}(G(Q)/s) : \exists\, A_P \in \mathit{minAcc}(G(P)/s) : A_P \subseteq A_Q \tag{12}$$

□

Proof. Recall that $P \sqsubseteq_T Q$ is defined as $\mathit{traces}(Q) \subseteq \mathit{traces}(P)$ and that $\mathit{traces}(P) = \mathit{traces}(G(P))$ and $\mathit{traces}(Q) = \mathit{traces}(G(Q))$. This shows (9). Formula (10) has been shown in [2].

FiXme Fatal: alcc: (11) is already proved in my paper with mcg. No need to repeat it here. I suggest just to add a reference.
FiXme Note: jp: done

To prove (11), we derive

$Q \ \mathit{conf}\ P$

$$\Leftrightarrow \forall\, s \in \mathit{traces}(P) \cap \mathit{traces}(Q) : \mathit{Ref}(Q/s) \subseteq \mathit{Ref}(P/s)$$
[Definition of *conf* (1)]
$$\Leftrightarrow \forall\, s \in \mathit{traces}(G(P)) \cap \mathit{traces}(G(Q)) : \mathit{Ref}(Q/s) \subseteq \mathit{Ref}(P/s)$$
[$\mathit{traces}(P) = \mathit{traces}(G(P))$, $\mathit{traces}(Q) = \mathit{traces}(G(Q))$]
$$\Leftrightarrow \forall\, s \in \mathit{traces}(G(P)) \cap \mathit{traces}(G(Q)), R_Q \in r_Q(G(Q)/s) :$$
$$\exists\, R_P \in r_P(G(P)/s) : R_Q \subseteq R_P$$
[Property of r_P, r_Q and (4)]

Finally, (12) follows from (11) using (6) and the fact that $R_Q \subseteq R_P$ is equivalent to $\Sigma - R_P \subseteq \Sigma - R_Q$. □

Reachability Under Sets of Traces Given a finite-state CSP process P and its normalised transition graph $G(P)$, we suppose that $V \subseteq \Sigma^*$ is a prefix-closed set of sequences of events. By $t(\underline{n}, V)$ we denote the set

$$t(\underline{n}, V) = \{n \in N \mid \exists\, s \in V : s \in \mathit{traces}(P) \wedge G(P)/s = n\}$$

of nodes in N that are reachable in $G(P)$ by applying traces of V. The lemma below specifies a construction method for such sets V reaching *every* node of N.

Lemma 2. *Let P be a CSP process with normalised transition graph $G(P)$. Let $V \subseteq \Sigma^*$ be a finite prefix-closed set of sequences of events. Suppose that $G(P)$ reaches $k <| N |$ nodes under V, that is, $| t(\underline{n}, V) |= k$. Let $V.\Sigma$ denote the set of all sequences from V, extended by any event of Σ. Then $G(P)$ reaches at least $(k+1)$ nodes under $V \cup V.\Sigma$.*

Proof. Suppose that $n' \in (N - t(\underline{n}, V))$. Since all nodes in N are reachable, there exists a trace s such that $G(P)/s = n'$. Decompose $s = s_1.e.s_2$ with $s_i \in \Sigma^*, e \in \Sigma$, such that $G(P)/s_1 \in t(\underline{n}, V)$ and $G(P)/s_1.e \notin t(\underline{n}, V)$. Such a decomposition always exists, because V is prefix-closed and therefore contains the empty trace ε. Note, however, that it is not necessarily the case that $s_1 \in V$.

Since $G(P)$ reaches $G(P)/s_1$ under V, there exists a trace $u \in V$ such that $G(P)/u = G(P)/s_1 = \overline{n}$. Since $s = s_1.e.s_2$ is a trace of P and $G(P)/s_1 = \overline{n}$, then $(\overline{n}, e)$ is in the domain of t. So, $G(P)/u.e = G(P)/s_1.e = n$ is a well-defined node of N not contained in $t(\underline{n}, V)$. Since $u.e \in V \cup V.\Sigma$, $G(P)$ reaches at least the additional node n under $V \cup V.\Sigma$. This completes the proof. □

Graph Products For proving our main theorems, it is necessary to consider the *product* of normalised transition graphs. We need this only for the investigation of corresponding traces in reference processes and processes for SUTs. So, the labelling of nodes with maximal refusals or minimal acceptances are disregarded in the product construction. We consider two normalised transition graphs

$$G_i = (N_i, \underline{n}_i, \Sigma, t_i : N_i \times \Sigma \nrightarrow N_i, r_i : N_i \to \mathbb{PP}(\Sigma)), \qquad i = 1, 2,$$

over the same alphabet Σ. Their product is defined by

$$G_1 \times G_2 = (N_1 \times N_2, (\underline{n}_1, \underline{n}_2), t : (N_1 \times N_2) \times \Sigma \nrightarrow (N_1 \times N_2)) \quad (13)$$

$$\text{dom } t = \{((n_1, n_2), e) \in (N_1 \times N_2) \times \Sigma \mid (n_1, e) \in \text{dom } t_1 \wedge (n_2, e) \in \text{dom } t_2\} \quad (14)$$

$$t((n_1, n_2), e) = (t_1(n_1, e), t_2(n_2, e)) \text{ for } ((n_1, n_2), e) \in \text{dom } t \quad (15)$$

The following lemma is used in the proof of our main theorem.

Lemma 3. *If G_1 has p states and G_2 has q states, then every reachable state (n_1, n_2) of the product graph $G_1 \times G_2$ can be reached by a trace of maximal length $(pq - 1)$.*

Proof. The product graph $G_1 \times G_2$ has at most pq states. The empty trace ε reaches its initial state $(\underline{n}_1, \underline{n}_2)$. Applying Lemma 2 $(pq-1)$ times with $V = \{\varepsilon\}$ implies that $G_1 \times G_2$ reaches all of its reachable states (there are at most pq of them) under $V' = V \cup V.\Sigma \cup \cdots \cup V.\Sigma^{(pq-1)}$. The maximal length of traces in V' is $(pq - 1)$. □

This concludes our presentation of CSP and of some results regarding its semantics and normalised transition graphs that are used in the next section.

2.2 Minimal Hitting Sets

Definition The main idea of the underlying test strategy for failures refinement is based on solving a *hitting set problem*. Given a finite collection of finite sets $C = \{A_1, \ldots, A_n\}$, such that each A_i is a subset of a universe Σ, a *hitting set* $H \subseteq \Sigma$ is a set satisfying the following property.

$$\forall A \in C : H \cap A \neq \varnothing. \quad (16)$$

A *minimal hitting set* is a hitting set that cannot be further reduced without losing the characteristic property (16). By *minHit*(C) we denote the collection of minimal hitting sets for a collection C. For the pathological case where C contains an empty set, *minHit*(C) is also empty.

The problem of determining minimal hitting sets is NP-hard [22]. We see below, however, that it reduces the effort of testing for failures refinement from a factor of $2^{|\Sigma|}$ to a factor that equals the number of minimal hitting sets.

Minimal Hitting Sets of Normalised Transition Graphs Since minimal acceptances can be used to label the nodes of a normalised transition graph, and since *minAcc*(P/s) = *minAcc*$(G(P)/s)$ according to (8), the notation of minimal hitting sets also carries over to graphs: we write *minHit*(n) for nodes n of $G(P)$ and observe that

$$\textit{minHit}(G(P)/s) = \textit{minHit}(P/s) \text{ for all } s \in \textit{traces}(P). \quad (17)$$

FiXme Fatal: You already used this above. Perhaps you want to move this to before Lemma 4?
FiXme Note: jp: done

Characterisation of *conf* by Minimal Hitting Sets The following lemma establishes that the *conf* relation specified in (1) can be characterised by means of minimal acceptances and their minimal hitting sets.

Lemma 4. *Let P, Q be two finite-state CSP processes. For each $s \in traces(P)$, let $minHit(P/s)$ denote the collection of all minimal hitting sets of $minAcc(P/s)$. Then the following statements are equivalent.*

1. *Q conf P*
2. *For all $s \in traces(P) \cap traces(Q)$ and $H \in minHit(P/s)$, H is a (not necessariliy minimal) hitting set of $minAcc(Q/s)$.*

Proof. Throughout the proof, we apply (8) and (17), so that $minAcc(P/s)$ and $minAcc(G(P)/s)$, as well as $minHit(P/s)$ and $minHit(G(P)/s)$ can be used interchangeably.

For showing "1 $\Rightarrow$ 2", assume Q *conf* P and $s \in traces(P) \cap traces(Q)$. Lemma 1 (12), states that

$$\forall\, A_Q \in minAcc(G(Q)/s) : \exists\, A_P \in minAcc(G(P)/s) : A_P \subseteq A_Q$$

Therefore, $H \in minHit(P/s)$ not only implies $H \cap A_P \neq \varnothing$ for all minimal acceptances A_P, but also $H \cap A_Q \neq \varnothing$ for every minimal acceptance A_Q, because $A_P \subseteq A_Q$ for at least one A_P. As a consequence, each $H \in minHit(P/s)$ is also a hitting set for $minAcc(G(Q)/s)$ as required.

To prove "2 $\Rightarrow$ 1", assume that 2 holds, but that P *conf* Q does *not* hold. According to Lemma 1, (12), there exists $s \in traces(P) \cap traces(Q)$ such that

$$\exists\, A_Q \in minAcc(G(Q)/s) : \forall\, A_P \in minAcc(G(P)/s) : A_P \not\subseteq A_Q \qquad (*)$$

Let A be such an acceptance set A_Q fulfilling (*). Define

$$\overline{H} = \bigcup \{ A_P \setminus A \mid A_P \in minAcc(G(P)/s) \}.$$

Since $A_P \setminus A \neq \varnothing$ for all A_P because of (*), $\overline{H}$ is a hitting set of $minAcc(G(P)/s)$ which has an empty intersection with A. Minimising $\overline{H}$ yields a minimal hitting set $H \in minHit(P/s)$ which is *not* a hitting set of $minAcc(G(Q)/s)$, a contradiction to Assumption 2. This completes the proof of the lemma. □

FiXme Fatal: Please, check that you're happy with this observation.

We note that $minAcc(P) = \{\varnothing\}$ if $P = Q \sqcap Stop$. Since *Stop* accepts nothing, its minimal acceptance is $\varnothing$, and this carries over to $Q \sqcap Stop$. From (12) we conclude that $\varnothing \in minAcc(P)$ implies $minAcc(P) = \{\varnothing\}$. This clarifies that $minHit(P/s)$ is empty if, and only if, $minAcc(P) = \{\varnothing\}$. The proof of Lemma 4 covers the situations where $minAcc(P/s) = \{\varnothing\}$ and so $minHit(P/s) = \varnothing$. Trivially,

$$minAcc(P/s) = \{\varnothing\} \Leftrightarrow minHit(P/s) = \varnothing \qquad (18)$$

holds.

FiXme Note: jp: I think the new first sentence of the proof should now be sufficient. Therefore, I removed your addition.

2.3 Sperner Families

In preparation for complexity results to be presented in Section 6, we consider how many minimal hitting sets can maximally exist for a given collection of minimal acceptances. To this end, the following definitions and results are useful.

A *Sperner Family* is a collection $S \subseteq 2^{\Sigma}$ of sets from a given finite universe Σ that do not contain each other, that is, $H_1 \not\subseteq H_2 \wedge H_2 \not\subseteq H_1$ holds for each pair $H_1 \neq H_2 \in S$. Specialising antichains known from partial orders to finite sets partially ordered by $\subseteq$ results in Sperner families. We observe that

FiXme Fatal: Shouldn't the footnote go to the acknowledgements?
FiXme Note: jp: done

- the maximal refusals of a CSP process state,
- the minimal acceptances of a CSP process state, and
- the minimal hitting sets of a given collection of sets

are Sperner families. Moreover, given any finite alphabet Σ with $|\Sigma| = n$, every collection S of subsets with identical cardinality $k \leqslant n$ is a Sperner family, because $A_1, A_2 \in S \wedge A_1 \subseteq A_2 \wedge |A_1| = |A_2|$ implies $A_1 = A_2$. Given any Sperner Family S of Σ, S represents the minimal acceptances in the initial state of the CSP process $P = \bigsqcap_{A \in S} \big(e : A \to P(e)\big)$.

The cardinality of Sperner Families is determined by the following theorem.

Theorem 1 (Sperner's Theorem [24]). *Given a Sperner family S over an n-element universe Σ, its cardinality is bound by*

$$|S| \leqslant \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

The upper bound is reached if, and only if, one of the following cases apply:

1. *For even n, if S consists of all subsets of Σ with cardinality $n/2$;*
2. *For odd n, if one of the following cases holds;*
   (a) *S consists of all subsets of Σ with cardinality $(n+1)/2$; or*
   (b) *S consists of all subsets of Σ with cardinality $(n-1)/2$.*

□

It is shown in Section 6 that this upper bound can actually be reached by hitting sets associated with the minimal acceptances of a CSP process state.

3 Finite Complete Test Suites for CSP Failures Refinement

Here, we define our notion of tests for failures refinement, and then prove completeness of our suite. Finally, we study to complexity of our approach by identifying a bound on the number of tests we need in a complete suite.

3.1 Test Cases for Verifying CSP Failures Refinement

Test Definition and Basic Properties In the domain of process algebras, test cases are typically represented by processes interacting concurrently with the SUT [8]. Considering an (unknown) process that represents the behaviour of the SUT, we say that tests synchronise with the process for the SUT over its visible events and use some additional events outside the SUT process's alphabet to express whether the test execution passed or failed.

For a given reference process P, its normalised transition graph

$$G(P) = (N, \underline{n}, \Sigma, t : N \times \Sigma \nrightarrow N, r : N \to \mathbb{PP}(\Sigma)),$$

and each integer $j \geqslant 0$, we define a test for failures refinement as shown below.

$$U_F(j) = U_F(j, 0, \underline{n}) \tag{19}$$

$$\begin{aligned} U_F(j,k,n) = {} & \big(e : (\Sigma - [n]^0) \to \mathit{fail} \to \mathit{Stop}\big) && (20)\\ & \Box \\ & (\mathit{minHit}(n) = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) && (21)\\ & \Box \\ & (k < j)\&\big(e : [n]^0 \to U_F(j, k+1, t(n,e))\big) && (22)\\ & \Box \\ & (k = j \wedge \mathit{minHit}(n) \neq \varnothing)\& \\ & \quad \big(\bigsqcap_{H \in \mathit{minHit}(n)} (e : H \to \mathit{pass} \to \mathit{Stop})\big) && (23) \end{aligned}$$

Explanation of the Test Definition A test is performed by running $U_F(j)$ concurrently with any SUT process Q, synchronising over Σ. So, a *test execution* is a trace of the concurrent process $Q \,|[\,\Sigma\,]|\, U_F(j)$.

It is assumed that the events *fail* and *pass*, indicating a verdict FAIL and PASS for the test execution, are not included in Σ. Since we assume that Q is free of livelocks, it is guaranteed that events *fail* or *pass* always become visible, if they are the only events $U_F(j)/s$ is ready to engage in: if $U_F(j)/s$ can only produce *pass* or *fail*, the occurrence of these events can never be blocked due to a livelock in Q occurring in the same step of the execution.

The test is *passed* by the SUT (written $Q \underline{\text{ pass }} U_F(j)$) if, and only if, *every* execution of $Q \,|[\,\Sigma\,]|\, U_F(j)$ terminates with the event *pass*. This can also be expressed by means of a failures refinement as defined below.

$$Q \underline{\text{ pass }} U_F(j) \mathrel{\widehat{=}} (\mathit{pass} \to \mathit{Stop}) \sqsubseteq_F (Q \,|[\,\Sigma\,]|\, U_F(j)) \setminus \Sigma \tag{24}$$

This type of pass relation is often called *must test* in the literature, because every test execution must end with the *pass* event [8].

We note that it is necessary to use failures refinement in the definition above, and not just trace refinement: $(Q \,|[\,\Sigma\,]|\, U_F(j)) \setminus \Sigma$ may have the same visible traces ε and *pass* as the "Test Passed Process" $(\mathit{pass} \to \mathit{Stop})$. However, the

former may nondeterministically refuse *pass*, due to a deadlock occurring when a faulty SUT process executes concurrently with $U_F(j,k,n)$ executing branch (23), when the guard condition $(k = j \wedge \mathit{minHit}(n) \neq \varnothing)$ evaluates to `true`. This is explained further in the next paragraphs. Alternatively, a faulty SUT Q might internally deadlock after a trace s where $\#s < j$ and $\mathit{minHit}(G(P)/s) \neq \varnothing$, so that $(Q \,|[\, \Sigma \,]|\, U_F(j))/s$ deadlocks as well.

Intuitively speaking, $U_F(j)$ is able to perform any trace s of P, up to a length j. If, after having already run through $s \in \mathit{traces}(P)$ with $\#s \leqslant j$, an event is accepted by the SUT that is outside the initials of P/s (recall from Lemma 5 that $[n]^0 = [P/s]^0$ for $U_F(j)/s$), the test immediately terminates with FAIL-event *fail*. This is handled by the branch (20) of the external choice.

If P/s is the *Stop* process or has *Stop* as an internal choice, this is revealed by $\mathit{minHit}(G(P)/s) = \varnothing$ (recall (18) and Lemma 5). In this case, the test may terminate successfully (branch (21) of the external choice in $U_F(j, \#s, G(P)/s)$). If P/s may also nondeterministically engage into events in such a situation, branch (22) of the test is simultaneously enabled. If Q/s is able to engage into an event from $\Sigma - [P/s]^0$, a test execution exists where $U_F(j, \#s, G(P)/s)$ branches into (20) and produces the *fail* event.

If the length of s is still less than j, the test accepts any event e from the initials $[P/s]^0 = [G(P)/s]^0$ and continues recursively as $U_F(j, \#s{+}1, G(P)/s.e)$ (note that $G(P)/s.e = t(G(P)/s, e)$) in branch (22); this follows again from Lemma 5. A test of this type is called *adaptive*, because it accepts any legal behaviour of the SUT and adapts its consecutive behaviour to the event selected by the SUT.

Now suppose that a test execution has run through a trace $s \in \mathit{traces}(P)$ of length j, so that $U_F(j)/s = U_F(j,j,n)$ with $n = G(P)/s$. If $\mathit{minHit}(n) \neq \varnothing$, the test changes its behaviour: instead of offering *all* legal events from $[n]^0$ to the SUT, it nondeterministically chooses a minimal hitting set $H \in \mathit{minHit}(n)$ and only offers the events contained in H. If the SUT refuses to engage into some event of H, this reveals a violation of failures refinement: according to Lemma 4, a conforming SUT should accept at least one event of each minimal hitting set in $\mathit{minHit}(n)$. Therefore, the test execution only terminates with *pass*, if such an event is accepted by the SUT. Otherwise, it deadlocks, and the test fails.

The specification of $U_F(j,k,n)$ implies that the test always stops after having engaged into a trace $s \in \mathit{traces}(Q)$ of maximal length j or $j+1$. If branch (20) is the last to be entered, the maximal length of s is $j+1$, and the test execution stops with *fail*. If branch (21) is the last to be entered, the maximal length of s is j, and the execution stops with *pass*. If branch (23) is the last to be entered, the process either accepts another event e of some minimal hitting set $H \in \mathit{minHit}(n)$ with $n = G(P)/s$ according to Lemma 5. Then the final length of s is $j+1$, and the execution terminates with *pass*. Or the test execution $(Q \,|[\, \Sigma \,]|\, U_F(j))/s$ deadlocks, the final length of s is j, and the execution stops without a PASS or FAIL event. Such an execution is also interpreted as FAIL, because it reveals that $(\mathit{pass} \rightarrow \mathit{Stop}) \not\sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U_F(j)) \setminus \Sigma$.

We observe that the number of possible executions of $Q \,|[\,\Sigma\,]|\, U_F(j)$ is finite, because the number of traces s with maximal length $(j+1)$ is finite and the sets $[n]^0$, $(\Sigma - [n]^0)$, and $\mathit{minHit}(n)$ are finite. Moreover, we further recall that $\mathit{minHit}(n)$ may be empty, in which case the indexed internal choice in (23) would be undefined. The guard in that branch, however, requires $\mathit{minHit}(n) \neq \varnothing$, and branches (20) or (21) can be taken in this situation.

We state basic properties of our tests in the following lemma, to show the relationships between $U_F(j)$ and the reference process P from which it is derived.

Lemma 5. *If $s \in \mathit{traces}(P)$ satisfies $\#s \leqslant j$, then $s, s.e \in \mathit{traces}(U_F(j))$ for all $e \in \Sigma$, and the following properties hold.*

$$U_F(j)/s = U_F(j, \#s, G(P)/s) \tag{25}$$

$$e \notin [P/s]^0 \Rightarrow U_F(j)/s.e = (\mathit{fail} \rightarrow \mathit{Stop}) \tag{26}$$

$$U_F(j)/s = U_F(j, \#s, n) \Rightarrow [n]^0 = [P/s]^0 \tag{27}$$

$$U_F(j)/s = U_F(j, \#s, n) \Rightarrow \mathit{minHit}(n) = \mathit{minHit}(P/s) \tag{28}$$

Proof. We prove (25) by induction over the length of s. For $\#s = 0$, the statement holds because $U_F(j)$ starts with the initial node $\underline{n}$ of $G(P)$. Suppose that the statement holds for all traces s with length $\#s \leqslant k < j$, so that $U_F(j)/s = U_F(j, \#s, G(P)/s)$. Now let $s.e$ be a trace of P, so that $e \in [P/s]^0$. Since $[G(P)/s]^0 = [P/s]^0$ for all traces s of P, we conclude that $e \in [G(P)/s]^0$, so $U_F(j, \#s, G(P)/s)$ can engage into e by executing branch (22). Since t is the transition function of $G(P)$ and $e \in [G(P)/s]^0$, $t(G(P)/s, e)$ is defined, and $t(G(P)/s, e) = G(P)/s.e$. This leads to a new recursion, so that $U_F(j)/s.e = U_F(j, \#s, G(P)/s)/e = U_F(j, \#s + 1, G(P)/s.e)$ as required.

To prove (26), we apply (25) to conclude that $U_F(j)/s = U_F(j, \#s, G(P)/s)$, because s is a trace of P. Noting again that $[G(P)/s]^0 = [P/s]^0$, this implies that $e \notin [G(P)/s]^0$, so $U_F(j, \#s, G(P)/s)$ can engage in e by entering branch (20). The specification of this branch implies that

$$U_F(j)/s.e = U_F(j, \#s, G(P)/s)/e = (\mathit{fail} \rightarrow \mathit{Stop}).$$

Statement (27) follows trivially from (25), because $[G(P)/s]^0 = [P/s]^0$ for all traces s of P. Finally, statement (28) follows trivially from (25), because, according to (17), $\mathit{minHit}(G(P)/s) = \mathit{minHit}(P/s)$ for all traces of P. □

Note that it is not guaranteed for $U_F(j)$ to run through the traces $s, s.e$ in Lemma 5, if $\mathit{minHit}(P/u) = \varnothing$ for some prefix u of s: in such a case, $U_F(j)$ may stop with a *pass* event by entering branch (21). Therefore, Lemma 5 just states the existence of $U_F(j)$-executions $s, s.e$ satisfying the properties stated there.

3.2 A Finite Complete Test Suite for Failures Refinement

A CSP *fault model* $\mathcal{F} = (P, \sqsubseteq, \mathcal{D})$ counsists of a reference process P, a conformance relation $\sqsubseteq \in \{\sqsubseteq_T, \sqsubseteq_F\}$, and a fault domain $\mathcal{D}$, which is a set of CSP processes over P's alphabet that may or may not conform to P.

A test suite TS is called *complete* with respect to fault model $\mathcal{F}$, if, and only if, the following conditions are fulfilled.

1. Soundness If $P \sqsubseteq Q$, then Q passes all tests in TS.
2. Exhaustiveness If $P \not\sqsubseteq Q$ and $Q \in \mathcal{D}$, then Q fails at least one test in TS.

The following main theorem establishes the completeness of our test suite.

Theorem 2. *Let P be a non-terminating, divergence-free CSP process over alphabet Σ whose normalised transition graph $G(P)$ has p states. Define fault domain $\mathcal{D}$ as the set of all divergence-free CSP processes over alphabet Σ, whose transition graph has at most q states with $q \geqslant p$. Then the test suite*

$$TS_F = \{\, U_F(j) \mid 0 \leqslant j < pq \} \quad U_F(j) \text{ specified in (19)}$$

is complete with respect to $\mathcal{F} = (P, \sqsubseteq_F, \mathcal{D})$.

The proof of the theorem follows from the two lemmas below. The first states that test suite TS_F is sound, the second states that the suite is also exhaustive.

Lemma 6. *A test suite TS_F generated from a CSP process P, as specified in Theorem 2, is passed by every CSP process Q satisfying $P \sqsubseteq_F Q$.*

Proof. We make two points in separate steps below. The first is that the test execution cannot reach branch (20) and raise a *fail* event. The second is that it cannot deadlock without raising a *pass* event. This case would also be interpreted as FAIL, since then $pass \rightarrow Stop$ is not failures refined by $(Q \,|[\, \Sigma \,]|\, U_F(j)) \setminus \Sigma$.

Step 1. Suppose that $P \sqsubseteq_F Q$, so $P \sqsubseteq_T Q$ and Q *conf* P according to (10). Since $traces(Q) \subseteq traces(P)$, any adaptive test $U_F(j)$ running in parallel with Q will always enter the branches (21), (22), or (23) of the external choice construction for $U_F(j,k,n)$. To see this, consider $U_F(j,k,n) = U_F(j)/s$ with $s \in traces(Q)$. Lemma 5 implies $U_F(j,k,n) = U_F(j,k,G(P)/s)$, so $[n]^0 = [G(P)/s]^0 = [P/s]^0$. As a consequence, $[Q/s]^0 \subseteq [P/s]^0 = [n]^0$, so branch (20) can never be entered in the parallel execution of Q and $U_F(j)$, and the *fail* event cannot occur.

Step 2. Since Q *conf* P, Lemma 4 implies that for all $s \in traces(Q) \cap traces(P)$, every $H \in minHit(P/s)$ is a hitting set for $minAcc(Q/s)$. From Lemma 5 we know that $U_F(j)/s = U_F(j,k,n)$ with $U_F(j,k,n) = U_F(j, \#s, G(P)/s)$, so $minHit(n) = minHit(P/s)$. The branch (21) of the test $U_F(j,k,n)$ leads always to a PASS verdict and is taken if $minHit(n) = \varnothing$. The branch (22) always leads to test continuation without a verdict. For the last branch, we note that it can only be entered if $minHit(P/s) = minHit(n) \neq \varnothing$. In this case, any selected minimal hitting set $H \in minHit(n)$ has a non-empty intersection with each of the minimal acceptances of Q/s. As a consequence, Q/s never blocks when offered events from $H \in minHit(n)$, and the test terminates with event *pass*. Note that this argument requires that Q is free of livelocks, because otherwise the *pass*-events might not become visible, due to unbounded sequences of hidden events performed by Q. □

FiXme Fatal: I think we need a stronger argument here, as to why it cannot deadlock.
FiXme Note: jp: I do not understand the question – let's discuss this on the phone.

Lemma 7. *A test suite TS_F specified as in Theorem 2 is exhaustive for the fault model specified there.*

Proof. Consider a process $Q \in \mathcal{D}$ with $P \not\sqsubseteq_F Q$. According to (10), this non-conformance can be caused in two possible ways corresponding to the cases $P \not\sqsubseteq_T Q$ and $\neg(Q \; conf \; P)$. These cases can be characterised as follows:

Case 1 $traces(Q) \not\subseteq traces(P)$

Case 2 There exists a joint trace $s \in traces(Q) \cap traces(P)$ and a minimal acceptance A_Q of $minAcc(Q/s)$, such that (see Lemma 1, (12)).

$$\forall\, A_P \in minAcc(P/s) : A_P \not\subseteq A_Q, \tag{29}$$

It has to be shown for each of these cases that at least one test execution of some $(Q \,|[\, \Sigma \,]|\, U_F(j))$ with $j < pq$ ends with the *fail* event or deadlocks. We do this by analysing the product graph of the reference process P and the SUT process Q: any trace $s \in traces(Q) \cap traces(P)$ gives rise to a path labelled by the events of s through this product graph. Any error can be detected after running through such a trace and then either observing an event outside $[P/s]^0$ (this is the violation described by Case 1) or identifying an illegal acceptance A_Q as specified in Case 2. It is not guaranteed, however, that trace s is short enough to be executed by one of the test cases $U_F(j)$ with $0 \leqslant j < pq$. Therefore, it has to be shown that for any s leading to an error situation specified by Case 1 or Case 2, there exists a trace u of maximal length $pq - 1$ leading to the same error.

Case 1. Consider a trace $s.e \in traces(Q)$ with $s \in traces(P)$, but $s.e \notin traces(P)$. Such a trace always exists because ε is a trace of every process. In this case, s is also a trace of the product graph $G = G(P) \times G(Q)$ defined in Section 2.1, and $G/s = (G(P)/s, G(Q)/s)$ holds. The length of s is not known, but from the construction of G, we know that G has at most pq reachable states, because $G(P)$ has p states, and $G(Q)$ has at most q states. By Lemma 3, $(G(P)/s, G(Q)/s)$ can be reached by a trace $u \in traces(G)$ of length $\#u < pq$. Now the construction of the transition function of G implies that u is also a trace of P and Q, which means that $(G(P)/s, G(Q)/s) = (G(P)/u, G(Q)/u)$. Since test $U_F(pq-1)$ accepts all traces of P up to length $pq - 1$, u is also a trace of this test, and, by construction and by Lemma 5, $U_F(pq-1)/u = U_F(pq-1, \#u, G(P)/u)$. Since $s.e \notin traces(P)$, e is an element of $\Sigma - [P/u]^0 = \Sigma - [G(P)/s]^0$. Hence, in at least one execution, $U_F(pq-1, \#u, G(P)/u)$ executes its first branch (20) with this event e, so that the test fails. Again, the assumption of non-divergence of Q is needed for this conclusion.

FiXme Fatal: There is an assumption here that the graph of a test execution is given by the cross-product of graphs. But, test execution is not defined in this way. It is defined by parallelism and hiding. I think the definition of test execution needs to change.

Case 2. We note that trace s is again a trace of the product graph G, but we do not know its length. Again, by applying Lemma 3, we know that the state G/s can be reached by a trace $u \in traces(Q) \cap traces(P)$ of maximal length $\#u < pq$. We consider the test $U_F(\#u)$, for which $U_F(\#u)/u = U_F(\#u, \#u, G(P)/u)$, because of Lemma 5. Test $U_F(\#u)$ always performs branch (22) until the trace u has been completely processed. $U_F(\#u, \#u, G(P)/u)$ may execute branches

FiXme Note: jp: this is a misunderstanding: we are not looking at the product graph of Q and $U_F(j)$, but at the product graph of Q and P. I have added more explanatory text in the paragraph before Case 1.

FiXme Fatal: Again, nothing here is about this process, but about the graph, and the hiding that is needed is discarded.

FiXme Note: jp: see my explanation above

(20) or (23) only: assumption (29) in Case 2 implies that P/s has at least one non-empty minimal acceptance. From (18) we know that this is equivalent to $minHit(P/s) = minHit(G(P)/s) \neq \varnothing$, and we observe that $G(P)/s = G(P)/u$, so $minHit(G(P)/u) \neq \varnothing$. As a consequence, branch (21) cannot be taken because its guard condition evaluates to `false` for $U_F(\#u, \#u, G(P)/u)$. The guard condition $(k < p)$ for branch (22) evaluates to `false` for $U_F(\#u, \#u, G(P)/u)$, too. If branch (20) is executed, the test always fails. If branch (23) is executed, the test fails for the execution where a minimal hitting set $H \in minHit(P/u)$ is chosen by $U_F(\#u, \#u, G(P)/u)$ that has an empty intersection with the minimal acceptance A_Q from condition (29). The existence of such an H is guaranteed because of Lemma 4. As a consequence, there exists a test execution where Q/u selects acceptance A_Q and $U_F(\#u, \#u, G(P)/u)$ selects H. This execution deadlocks in process state $(Q \,|[\,\Sigma\,]|\, U_F(\#u))/u$, so it cannot produce the *pass*-event; this means that the test fails. This concludes the proof. □

FiXme Fatal: I think the example should come before more theory.

Our notion of tests can be specialised to deal with traces refinement, as we explain in Section 5. The next section presents and example.

FiXme Note: jp: done

4 Testing for Failures Refinement – an Example

Generating the test cases $U_F(p)$ specified in (19) for the reference process P discussed in Example 1, results in the instantiations of initials, minimal hitting sets, and transition function shown in Fig. 2; this can be directly derived from P's normalised transition graph with nodes $N = \{0, 1, 2, 3\}$ displayed in Fig. 1.

$[0]^0 = \{a\}$	$minHit(0) = \{\{a\}\}$	$t(0, a) = 1$	$t(2, a) = 1$
$[1]^0 = \{a, b, c\}$	$minHit(1) = \{\{a, b\}, \{c\}, \}$	$t(1, a) = 0$	$t(2, b) = 0$
$[2]^0 = \{a, b, c\}$	$minHit(2) = \{\{a, b\}, \{a, c\}, \}$	$t(1, b) = 0$	$t(2, c) = 3$
$[3]^0 = \{b, c\}$	$minHit(3) = \{\{b\}, \{c\}\}$	$t(1, c) = 2$	$t(3, b) = 0$
			$t(3, c) = 3$

Fig. 2. Initials, minimal hitting sets, and transition function of the normalised transition graph displayed in Fig. 1.

Example 4. Consider the following implementation Z of process P from Example 1 that is erroneous from the point of view of failures refinement. In the

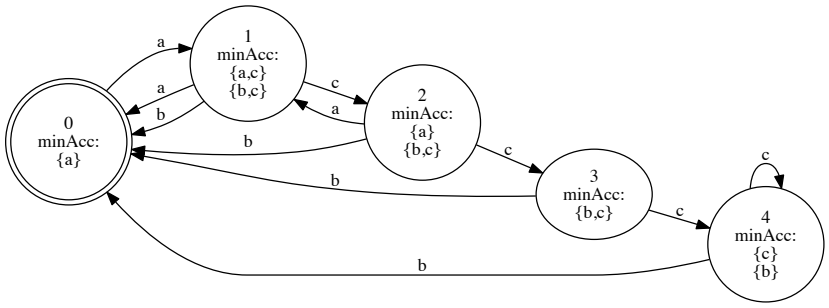

Fig. 3. Normalised transition graph of faulty implementation Z from Example 4.

specification of Z, it is assumed that $r_{max} \geqslant 0$.

$$\begin{aligned} Z &= a \to (Q_1 \sqcap R_1(r_{max}, 0)) \\ Q_1 &= a \to Z \;\Box\; c \to Z \\ R_1(r_{max}, k) &= (k < r_{max}) \& \big(b \to Z \;\Box\; c \to R_1(r_{max}, k+1)\big) \\ &\quad \Box \\ &\quad (k = r_{max}) \& \big(b \to Z \sqcap c \to R_1(r_{max}, r_{max})\big) \end{aligned}$$

It is easy to see (and can be checked with FDR) that Z is trace-equivalent to P. While $k < r_{max}$, Z also accepts the same sets of events as P. When $R_1(r_{max}, k)$ runs through several recursions and $k = r_{max}$, however, $R_1(r_{max}, k)$ makes an internal choice, instead of offering an external choice, so $P \not\sqsubseteq_F Z$. Fig. 3 shows the normalised transition graph of Z for $r_{max} = 3$.

Running the test $U_F(j)$ against Z for $j = 0, \ldots, 19$ ($G(P)$ has $p = 4$ states and $G(Z)$ has $q = 5$, so $pq - 1 = 19$ is the index of the last test to be executed according to Theorem 2), tests $U_F(0), \ldots, U_F(3)$ are passed by Z, but Z fails $U_F(4)$, because after execution of the trace

$$s = a.c.c.c, \qquad \text{(note that } G(P)/s = \text{node 3 according to Fig. 1)},$$

the test $U_F(4)$ offers hitting sets from $minHit(3) = \{\{b\}, \{c\}\}$ in branch (23). Therefore, there exists one test execution where Z/s accepts only $\{b\}$ due to the internal choice (note from Fig. 3 that $G(Z)/s = $ node 4), while $U_F(4)/s$ only offers $\{c\}$ in branch (23) or $\{a\} = \Sigma - [3]^0$ for branch (20). As a consequence, this execution of $(Z \,|[\, \Sigma \,]|\, U_F(4))/s$ deadlocks, and the *pass* event cannot be produced. Another failing execution arises if Z/s chooses to accept only $\{c\}$, while $U_F(4)/s$ choses to accept only $\{a, b\}$. Therefore,

$$(pass \to Stop) \not\sqsubseteq_F (Z \,|[\, \Sigma \,]|\, U_F(4)) \setminus \Sigma,$$

FiXme Fatal: The hiding is missing.
FiXme Note: jp: done

and the test fails. □

5 Finite Complete Test Suites for CSP Trace Refinement

For establishing trace refinement, the following class of adaptive test cases will be used for a given reference process P and integers $j \geqslant 0$. Just as for the tests developed in Section 3 to verify failures refinement, the tests for trace refinement are derived from the reference model's transition graph

$$G(P) = (N, \underline{n}, \Sigma, t : N \times \Sigma \nrightarrow N, r : N \to \mathbb{PP}(\Sigma)).$$

In contrast to the tests for failures refinement (19), however, we do not need to check the SUT with respect to its acceptance of hitting sets. Therefore, these do not occur in the specification of the test cases below, and we use the condition $minAcc(n) = \{\varnothing\}$ instead of $minHit(n) = \varnothing$ in branch (32) to indicate that minimal hitting sets need not to be calculated for generating these tests from $G(P)$. From (18) we know that these conditions are equivalent.

$$U_T(j) = U_T(j, 0, \underline{n}) \tag{30}$$

$$\begin{aligned} U_T(j,k,n) = {} & \big(e : (\Sigma - [n]^0) \to \mathit{fail} \to \mathit{Stop}\big) && (31)\\ & \Box \\ & (\mathit{minAcc}(n) = \{\varnothing\}) \& \big(\mathit{pass} \to \mathit{Stop}\big) && (32)\\ & \Box \\ & (k < j) \& \big(e : [P/s]^0 \to U_T(j, k+1, t(n,e))\big) && (33)\\ & \Box \\ & (k = j) \& \big(\mathit{pass} \to \mathit{Stop}\big) && (34) \end{aligned}$$

It is easy to see that the tests $U_T(j)$ satisfy the properties

$$U_T(j)/s = U_T(j, \#s, G(P)/s) \tag{35}$$

$$e \notin [P/s]^0 \Rightarrow U_T(j)/s.e = (\mathit{fail} \to \mathit{Stop}) \tag{36}$$

FiXme Fatal: In (37), I think there is an implicit assumption about p and $\#s$?

FiXme Note: jp: yes, I added $\#s \leqslant p$ in the text.

FiXme Fatal: The proofs don't use this definition. It should change.

FiXme Note: jp: I do not understand this remark – we should discuss it on the phone. Please note the additional explanations I have added here below (37) and please read the full proof for Theorem3.

proven in Lemma 5 for $U_F(j)$ for traces $s \in \mathit{traces}(P)$ with $\#s \leqslant j$.

Since $U_T(j)$ never blocks any Q-event before terminating, the pass criterion can be based on trace refinement instead of failures refinement as required in (24).

$$Q \underline{\text{ pass }} U_T(j) \mathrel{\widehat{=}} (\mathit{pass} \to \mathit{Stop}) \sqsubseteq_T (Q \,|[\,\Sigma\,]|\, U_T(j)) \setminus \Sigma \tag{37}$$

If the SUT process Q deadlocks internally after a trace s where also the reference process P is in a state where deadlock is possible, this is captured by the fact that $minAcc(n) = \{\varnothing\}$ for $n = G(P)/s$. As a consequence, branch (32) of a test case execution state $U_T(j, k, n)$ with $\#s = k \leqslant j$ can be entered and the test execution terminates with *pass*. If, however, Q blocks after a trace s'

where the reference process satisfies $minAcc(P/s') \neq \varnothing$, branch (32) cannot be taken, and the test execution stops without producing *pass* or *fail*. In contrast to the test for failures refinement, this is interpreted here as a successful test execution, because unexpected blocking of the SUT does not violate the trace refinement relation, as long as all traces executed by the SUT are traces of the reference process. In particular, if neither *pass* nor *fail* is ever produced, so that $(Q \,|[\, \Sigma \,]|\, U_T(j)) \setminus \Sigma = \mathit{Stop}$, the test passes, because $(\mathit{pass} \rightarrow \mathit{Stop}) \sqsubseteq_T \mathit{Stop}$ holds.

The existence of complete, finite test suites is expressed in analogy to Theorem 2. A noteworthy difference is that the complete suite for trace refinement just needs the single adaptive test case $U_T(pq-1)$, while failures refinement requires the execution of $\{U_F(0), \ldots, U_F(pq-1)\}$. The reason is that $U_T(pq-1)$ identifies trace errors for all traces up to length pq, while $U_F(pq-1)$ only probes for erroneous acceptances at the end of each trace of length $(pq-1)$.

Theorem 3. *Let P be a non-terminating, divergence-free CSP process over alphabet Σ whose normalised transition graph $G(P)$ has p states. Define fault domain $\mathcal{D}$ as the set of all non-terminating, divergence-free CSP processes over alphabet Σ, whose transition graph has at most q states with $q \geqslant p$. Then the test suite*

$$TS_T = \{U_T(pq-1)\}$$

is complete with respect to $\mathcal{F} = (P, \sqsubseteq_T, \mathcal{D})$. □

As before in the context of Theorem 2, the proof of Theorem 3 is structured into two lemmas, the first establishing soundness, the second exhaustiveness.

FiXme Note: jp: Wen-ling and I have now provided a full proof for Theorem 3

Lemma 8. *A test suite TS_T generated from a CSP process P, as specified in Theorem 3, is passed by every CSP process Q satisfying $P \sqsubseteq_T Q$.*

Proof. Suppose that $P \sqsubseteq_T Q$, so that $traces(Q) \subseteq traces(P)$, and assume that $s \in traces(Q)$ with $\#s < pq$. Since s is also a trace of P, we can conlude

$$U_T(pq-1)/s = U_T(pq-1, \#s, G(P)/s)$$

because of (35). Now $traces(Q) \subseteq traces(P)$ implies $[Q/s]^0 \subseteq [P/s]^0 = [G(P)/s]^0$, so $U_T(pq-1, \#s, G(P)/s)$ cannot enter branch (31) and produce a *fail*-event when running in parallel with Q and synchronising over Σ. Therefore, only four options are available for the test execution $(Q \,|[\, \Sigma \,]|\, U_T(j))/s$ to continue.

Case 1. Q/s deadlocks and $minAcc(G(P)/s) = \{\varnothing\}$. Then $U_T(pq-1, \#s, G(P)/s)$ enters branch (32), and the test execution stops after event *pass*.

Case 2. Q/s deadlocks, but $minAcc(G(P)/s) \neq \{\varnothing\}$. Then the whole test execution blocks, and this means that neither a *pass* nor a *fail* event is produced, so the test execution is passed.

Case 3. Q/s selects an event $e \in [Q/s]^0$ and $\#s < pq-1$. Then the test case $U_T(pq-1)$ in state $U_T(pq-1, \#s, G(P)/s)$ can also engage in e by entering

branch (33), and the test execution continues without having produced a *pass* or a *fail* event.

Case 4. $\#s = pq-1$ holds. Then $U_T(pq-1, \#s, G(P)/s)$ can only enter branch (34), and the test execution stops after *pass*.

This case analysis shows that every execution of $(Q \,|[\,\Sigma\,]|\, U_T(j))$ either stops after *pass* or produces neither *pass* nor *fail*. This proves that Q passes test $U_T(pq-1)$ according to the pass criterion (37). □

Lemma 9. *A test suite TS_T specified as in Theorem 3 is exhaustive for the fault model specified there.*

Proof. As before in the proofs for failures testing, we construct the product graph $G = G(P) \times G(Q)$ and recall that every trace $s \in \mathit{traces}(P) \cap \mathit{traces}(Q)$ is associated with a path through G labelled with the same events as s, such that $G/s = (G(P)/s, G(Q)/s)$. Furthermore, we recall from Lemma 2 that graph state $(G(P)/s, G(Q)/s)$ can always be reached by a trace u of length less or equal $pq-1$, where the order of $G(P)$ is p and that of $G(Q)$ is q.

Suppose that $P \not\sqsubseteq_T Q$. Then there exists a trace $s \in \mathit{traces}(Q) \cap \mathit{traces}(P)$ and an event $e \in [Q/s]^0$ such that $e \notin [P/s]^0$. Let $u \in \mathit{traces}(Q) \cap \mathit{traces}(P)$ be a trace with $\#u < pq$ and $G/u = (G(P)/s, G(Q)/s)$. Then

$$U_T(pq-1)/u = U_T(pq-1, \#u, G(P)/s).$$

By assumption, $e \in (\Sigma - [P/s]^0) = (\Sigma - [G(P)/s]^0)$. Since $G(Q)/u = G(Q)/s$, Q/u can engage into e. Then $U_T(pq-1, \#u, G(P)/s)$ will enter branch (31), and the test execution stops after having produced *fail*. This proves that Q fails test $U_T(pq-1)$. □

6 Complexity Considerations

FiXme Fatal: In that section, k is used for a different purpose. It is confusing. I would use p here.

FiXme Note: jp: ok – I also had to change this in Theorem 2, where k was used as index from 0 to $pq-1$. I now use j as index for $U_F(j)$, because p is the number of states in $G(P)$.

In this section, we will calculate upper bounds for the total number of test executions to be performed when testing for failures refinement. Theorem 2 specifies that all tests $U_F(j)$, $0 \leqslant j < pq$ need to be executed, where p denotes the number of nodes in the transition graph of the reference process P, and $q \geqslant p$ is an estimate for the number of nodes in the SUT's transition graph. Therefore, we will first calculate a bound for the number of test executions to be performed for test $U_F(j)$ and then summarise these bounds over all j from 0 to $pq-1$.

For the worst-case estimate, we assume that P never allows for early deadlock (so $\mathit{minHit}(P/s)$ is never empty) and that the SUT Q is a correct failures refinement. Then all test executions $(Q \,|[\,\Sigma\,]|\, U_F(j))$ stop after having run through a Q-trace of length $j+1$, because their is no early termination due to entering branches (20), (21), or due to an illegal deadlock of Q. As can be seen from the specification of the test cases $U_F(j)$ (see Section 3), the number of executions ending in a *pass* event corresponds to the number ℓ of traces s of P with length

equal to j, multiplied by the number h of minimal hitting sets in $\mathit{minHit}(P/s)$. For the tests $U_T(j)$ verifying trace refinement (see Section 5), the number of executions equals ℓ, since there is no equivalent in $U_T(j)$ to checking different hitting sets in the last step of a test execution.

FiXme Fatal: But doesn't this add to the number of executions?
FiXme Note: jp: should be clear now from the revised test

6.1 Estimation of ℓ

The first factor ℓ has worst-case upper bound $\ell \leqslant |\Sigma|^k$. As an example where this upper bound is really met, we consider the reference process

$$RUN(\Sigma) = e : \Sigma \to RUN(\Sigma).$$

The normalised transition graph of this process has a single state, and its initials are $[RUN(\Sigma)]^0 = \Sigma$. Therefore, the associated test process $U_F(j)$ can never enter branches (20) and (21), but there are exactly $|\Sigma|^j$ different traces of length j exercising branch (22) for each of their events.

6.2 Estimation of h

Given a set $\mathit{minAcc}(P/s) = \{A_1, \ldots, A_\alpha\}$ of minimal acceptances, the cardinality $h = |\mathit{minHit}(P/s)|$ has the upper bound

$$h \leqslant \binom{n}{\lfloor \frac{n}{2} \rfloor}, \quad \text{where } n = |\Sigma|.$$

This follows from Theorem 1, since $\mathit{minHit}(P/s)$ is a Sperner Family (see Section 2.3). The next theorem shows that this upper bound can really be reached by collections of minimal hitting sets associated with a CSP process state.

Theorem 4. *Let Σ be an alphabet with cardinality $n \geqslant 2$. Then there exists a CSP process P with*

$$|\mathit{minHit}(P)| = \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

Proof. Let C be the collection of all subsets of Σ with cardinality $n - \lfloor \frac{n}{2} \rfloor + 1$. Then the CSP process

$$P = \bigsqcap_{A \in C} e : A \to P(e)$$

fulfils $\mathit{minAcc}(P) = C$. Let H be any minimal hitting set of C. Then H contains at least $\lfloor \frac{n}{2} \rfloor$ elements, because otherwise $|\Sigma \setminus H| > n - \lfloor \frac{n}{2} \rfloor$, and any subset $A \subseteq \Sigma \setminus H$ with cardinality $n - \lfloor \frac{n}{2} \rfloor + 1$ would be contained in C, but satisfy $A \cap H = \varnothing$. Since $\lfloor \frac{n}{2} \rfloor + n - \lfloor \frac{n}{2} \rfloor + 1 = n + 1$, we conclude that any $\lfloor \frac{n}{2} \rfloor$-element subset of Σ intersects every element of C. Therefore, every minimal hitting set of C has exactly $\lfloor \frac{n}{2} \rfloor$ elements, so $|\mathit{minHit}(C)| = \binom{n}{\lfloor \frac{n}{2} \rfloor}$. □

To get an approximation of the maximal size of $minHit(P/s)$ for large n, recall Stirling's approximation [6, p. 112]

$$m! \approx \sqrt{2\pi m} \cdot \left(\frac{m}{e}\right)^m, \quad \text{where } e \text{ denotes the Euler Number.}$$

Applying this approximation when $n = |\Sigma|$ is even and $m = (n/2)$ results in

$$h \leqslant \binom{2m}{m} \approx \frac{4^m}{\sqrt{\pi m}} = \frac{2^{|\Sigma|}}{\sqrt{2\pi |\Sigma|}}.$$

We note that, due to this approximation result, h is smaller than $2^{|\Sigma|} - 1$, the cardinality of the non-empty subsets of Σ.

6.3 Upper Bounds of Test Executions for Checking Failures Refinement

By Theorem 2, we need to execute the tests $U_F(j)$ for $j = 0, \ldots, (pq-1)$; this results in a worst-case bound defined below, where we use the formula for the sum of the geometric progression.

$$h \cdot \left(\frac{1-|\Sigma|^{pq}}{1-|\Sigma|}\right), \text{ or, asymptotically, } O(\frac{2^{|\Sigma|}}{\sqrt{2\pi |\Sigma|}} \cdot |\Sigma|^{(pq-1)}).$$

FiXme Fatal: Theorem 4 is not about *conf*. I don't think you have tests for *conf*.

From Theorem 4 we know that the worst-case bound for h above cannot be further reduced, since the full collection of minimal hitting sets needs to be checked at the end of each execution of $U_F(j)$.

In [8], it is suggested to test *every* non-empty subset of Σ whose events cannot be completely refused in a given process state of the reference model; this leads to a worst-case estimate of $2^{|\Sigma|} - 1$ for the number of different sets to be offered to the SUT in the last step of the test execution, so our approach reduces the number of test executions in comparison to [8] by a factor of $1/\sqrt{2\pi |\Sigma|}$. In Fig. 4, the reduction is visualised by a function plot of $2^{|\Sigma|}$ versus $\frac{2^{|\Sigma|}}{\sqrt{2\pi |\Sigma|}}$.

In [2], the authors also use minimal hitting sets[3], but they do not give an upper bound for the number of test executions to be performed.

FiXme Note: jp: I rephrased this – hope that it's better understandable now.

FiXme Fatal: In [2], we work with minimal acceptances. I didn't check [8].

FiXme Note: jp: we have studied [2] now in more detail – you are right, of course, but there is a confusion of terms: you are also using the minimal hitting sets we handle here in our joint paper, but you used the term "minimal acceptance" for them. This is not in line with the definition of minimal acceptances as presented in [8] and in Roscoe's books. Here, in our joint paper, minimal acceptances are introduced as defined in Roscoe's books. I have changed the text accordingly.

6.4 Upper Bounds of Test Executions for Checking Trace Refinement

According to Theorem 3, a complete test suite checking trace refinement just contains the adaptive test case $U_T(pq-1)$. As derived for $U_F(j)$ above, the number of executions performed by $(Q\,|[\,\Sigma\,]|\,U_T(pq-1))$ is bounded by $|\Sigma|^{pq-1}$.

[3] However, they are denoted by *minimal acceptances* in [2].

FiXme Fatal: Please, see comments above

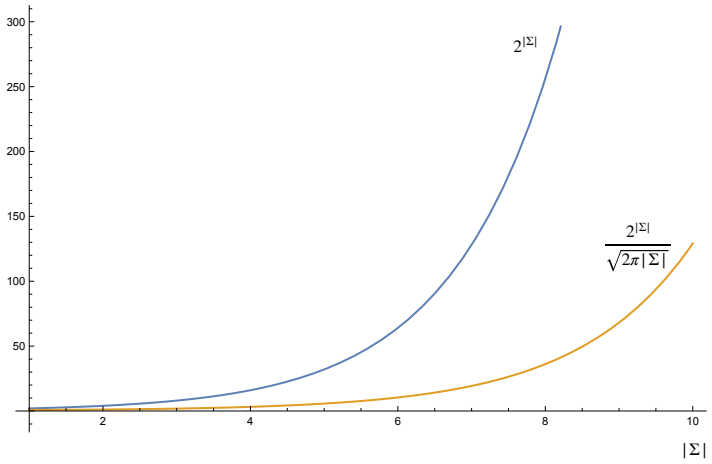

Fig. 4. Function plot $2^{|\Sigma|}$ versus $\frac{2^{|\Sigma|}}{\sqrt{2\pi|\Sigma|}}$.

6.5 Upper Bound pq for the Maximal Length of Test Traces

According to Theorem 2, the tests $U_F(j)$ need to be executed for $j = 0, \ldots, pq-1$ to guarantee completeness. This means that the SUT is verified with test traces up to, and including, length pq: recall from the test specification, branch (20), that $U_F(j)$ will accept all traces $s.e$ with $s \in traces(P), \#s = j, e \notin traces(P/s)$, so erroneous traces up to length $j+1$ are detected.

It is interesting to investigate whether this maximal length is really necessary, or whether one could elaborate alternative complete test strategies where the SUT is tested with shorter traces only. Indeed, an example presented in [13, Exercise 5] shows that when testing for equivalence of deterministic FSMs, it is sufficient to test the SUT with traces of significantly shorter length.

The following example, however, shows that the maximal length pq is really required when testing for refinement.

FiXme Fatal: Confusing use of p and q again.
FiXme Note: jp: fixed

Example 5. Consider the CSP reference process P and an erroneous implementation Q specified as follows.

FiXme Note: jp: I have updated the example with P and Q as used in the proof of the theorem. This should make thinks more readable.
FiXme Fatal: The internal choice became external. To test for traces refinement, yo put forward just one adaptive tests. So, there is more going on here I think it is

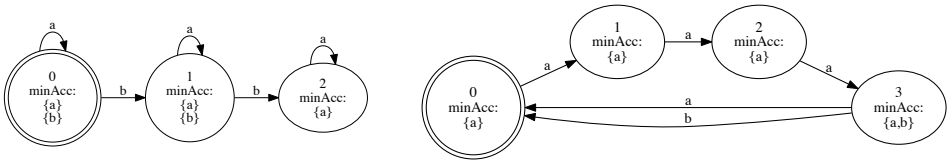

Fig. 5. Transition graphs of P (left) and Q (right) from Example 5 for $p = 3$ and $q = 4$.

$$
\begin{aligned}
P &= P(0) \\
P(k) &= (k < p-1)\&\big((a \to P(k)) \sqcap (b \to P(k+1))\big) \\
&\quad \Box \\
&\quad (k = p-1)\&(a \to P(k)) \\
Q &= Q(0) \\
Q(k) &= (k < q-1)\&\big(a \to Q(k+1)\big) \\
&\quad \Box \\
&\quad (k = q-1)\&\big(a \to Q(0) \,\Box\, b \to Q(0)\big)
\end{aligned}
$$

The normalised transition graphs of P and Q are depicted in Fig. 5 for the case $p = 3,\ q = 4$. Using FDR4, it can be shown for concrete values of p and q that the "test passed conditions"

$$(pass \to Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U_F(j)) \setminus \Sigma$$

and

$$(pass \to Stop) \sqsubseteq_T (Q \,|[\, \Sigma \,]|\, U_T(j)) \setminus \Sigma$$

hold for $j = 0, \ldots, pq - 2$. This means, that none of the test cases $U_F(j)$ and $U_T(j)$ are capable of detecting failures and trace refinement violations, respectively, if they only check traces up to length $pq - 1$ (recall that this corresponds to $j \leqslant pq - 2$).

Process Q, however, neither conforms to P in the failures refinement relation, nor in the trace refinement relation. This can only be seen when executing the test $U_F(pq-1)$ and $U_T(pq-1)$, respectively. These tests fail, so this shows that $P \not\sqsubseteq_F Q$ and $P \not\sqsubseteq_T Q$ according to Theorem 2 and Theorem 3. Moreover, this shows that the maximal trace length pq to be investigated in the tests cannot be further reduced without losing the completeness property of the test suites.

□

FiXme Fatal: But, there is no argument that the proofs based on graph products corresponds to this process. In particular the hiding does not seem to play a role.

FiXme Note: jp: should be clear now, since we have explained in Section 3 that only the graph product $G(P) \times G(Q)$ is needed in the proofs.

Generalising Example 5, it can be shown that for any pair $2 \leqslant p, q \in \mathbb{N}$, there exist reference processes P with p states and implementation processes Q with q states, such that a violation of the trace refinement property can only be detected with a trace of length pq. This is proven in the following theorem. In the proof, we use the processes P and Q introduced in Example 5.

Theorem 5. *Let $2 \leqslant p, q \in \mathbb{N}$. Then there exists a reference process P and an implementation process Q with the following properties.*

1. *$G(P)$ has p states.*
2. *$G(Q)$ has q states.*
3. *$P \not\sqsubseteq_T Q$, and therefore, also $P \not\sqsubseteq_F Q$.*
4. *$\forall\, s \in traces(Q) : \#s < pq \Rightarrow s \in traces(P)$.*
5. *Q conf P.*

As a consequence, the upper bound pq for the length of traces to be tested when checking for failures refinement or trace refinement cannot be reduced without losing the test suite's completeness property.

Proof. Given $2 \leqslant p, q \in \mathbb{N}$, define reference process P and implementation process Q as in Example 5. It is trivial to see that $G(P)$ has p nodes and $G(Q)$ has q nodes, so statements 1 and 2 of the theorem hold.

Using regular expression notation, the traces of P can be specified as

$$traces(P) = \mathbf{pref}\big((a^*b)^{p-1}a^*\big),$$

where $\mathbf{pref}(M)$ denotes the set of all prefixes of traces in $M \subseteq \Sigma^*$, including the traces of M themselves. The traces of Q can be specified by

$$traces(Q) = \mathbf{pref}\big((a^{q-1}(a \mid b))^*\big).$$

It is easy to see that $traces(Q) \not\subseteq traces(P)$; for example, the trace $(a^{q-1}b)^p$ is in $traces(Q) \setminus traces(P)$, because P-traces may contain at most $p-1$ b-events. This proves statement 3 of the theorem.

Let $s \in traces(Q)$ be any trace of length $\#s = pq - 1$. Then s can be represented by $s = (a^{q-1}(a \mid b))^{p-1}a^{q-1} \in \mathbf{pref}\big((a^{q-1}(a \mid b))^*\big)$. Then s is also an element of $traces(P)$, because $(a^{q-1}(a \mid b))^{p-1}a^{q-1}$ is also contained in $\mathbf{pref}\big((a^*b)^{p-1}a^*\big)$: this is easy to see, since $\mathbf{pref}\big((a^*b)^{p-1}a^*\big)$ contains all finite sequences of a-events, where at most $p-1$ events b have been inserted. This proves statement 4 of the theorem.

To prove statement 5, we observe that the specification of P implies (the expression $(s \downarrow b)$ denotes the number of b-events occurring in trace s)

$$minAcc(P/s) = \begin{cases} \{\{a\},\{b\}\} & \text{for all } s \in traces(P) \text{ with } (s \downarrow b) < p-1. \\ \{\{a\}\} & \text{for all } s \in traces(P) \text{ with } (s \downarrow b) = p-1. \end{cases}$$

and

$$minAcc(Q/s) = \begin{cases} \{\{a\}\} & \text{for all } s \in traces(Q) \text{ with } \#s \neq 0 \bmod (q-1). \\ \{\{a,b\}\} & \text{for all } s \in traces(P) \text{ with } \#s = 0 \bmod (q-1). \end{cases}$$

As a consequence, the minimal acceptance set $A_P = \{a\}$ which is contained in every $minAcc(P/s)$ fulfils $A_P \subseteq A_Q$ for any $A_Q \in minAcc(Q/s)$, when $s \in traces(P) \cap traces(Q)$. Now Lemma 1, (12) can be applied to conclude that Q *conf* P. □

Since Theorem 5 just states that a violation of trace refinement may remain undetected if only traces shorter than pq are checked during tests, it can also be applied to our trace refinement tests. Therefore, test suites $\{U_T(z)\}$ with $z < pq - 1$ are not complete. It is discussed in Section 7 how the number of test traces to be executed by complete test suites for failures or trace refinement can still be reduced *without* reducing the maximal length.

7 Discussion and Conclusions

Discussion of Further Reductions of the Test Effort It is known from complete testing strategies for finite state machines that the upper bound pq for the lengths of the traces used in our tests to investigate the SUT behaviour can be reduced. It is also known from FSM testing that it is not necessary to test *all* traces up to this maximal length. Notable complete strategies supporting this fact have been presented, for example, in [9,4,15,23]. From [11] it is known that complete FSM testing theories can be translated to other formalisms, such as Extended Finite State Machines, Kripke Structures, or CSP, resulting in likewise complete test strategies for the latter. We intend to study translations of several promising FSM strategies to CSP in the future. These will effectively reduce the upper bound $\ell \leqslant |\ \Sigma\ |^k$ introduced in Section 6. The bound h for the number of sets to be used in probing the SUT for illegal deadlocks, however, cannot be further reduced, as has been established in Lemma 4.

Discussion of Adaptive Test Cases The tests suggested in [8,2] were *preset* in the sense that the trace to be executed was pre-defined for each test. As a consequence, the authors of [2] introduced *inconclusive* as a third test result, applicable to the situations where the intended trace of the execution was blocked, due to legal, but nondeterministic behaviour of the SUT.

In contrast to that, our test cases specified in Section 3 and 5 are adaptive. This has the advantage that test executions $(Q\,|[\,\Sigma\,]|\,U_F(p))$ for failures refinement never block before the final step specified by branch (23), and so we do not need inconclusive test results. It should be noted, however, that it is necessary for our test verdicts to recognise also deadlocks in the final test step and interpret them as FAIL, as described in Section 3. In practice, this is realised by adding a timeout event to the testing environment which indicates deadlock situations. For real-time systems, this is an accepted technique, because the SUT has to respond within a pre-defined latency interval, otherwise its behaviour is considered to be blocked and regarded as a failure. The tests executions $(Q\,|[\,\Sigma\,]|\,U_T(p))$ for trace refinement never block at all before stopping after the verdict *pass*

FiXme Fatal: From the proofs of the main theorems, and what you say below, I thought you did have verdicts coming from deadlock, and you simply chose not to mark it with an inc event.

FiXme Fatal: Missing hiding.

FiXme Note: jp: Here, the hiding operator is not necessary: the possible test executions are really the traces of Q parallel $U_T(p)$. The hiding is only needed to specify the verdict.

or *fail*, unless the SUT process Q has an unexpected deadlock state. Recall from the explanations given in Section 5 that the blocking situation also leads to passing the test, because $(pass \rightarrow Stop) \sqsubseteq_T (Q \,|[\, \Sigma \,]|\, U_T(p)) \setminus \Sigma$ still holds if $(Q \,|[\, \Sigma \,]|\, U_T(p)) \setminus \Sigma$ only produces the empty trace.

The adaptive behaviour of both our test case types $U_F(p)$ and $U_T(p)$, however, induces the obligation to check that *all* possible executions have been performed before the test can be considered as passed. Typically, it is therefore assumed that a *complete testing assumption* [9] holds, which means that every possible behaviour of the SUT is performed after a finite number of test executions. In practice, this is realised by executing each test several times, recording the traces that have been performed, and using hardware or software coverage analysers to determine whether all possible test execution behaviours of the SUT have been observed. Therefore, adaptive test cases come at the price of having to apply some grey-box testing techniques enabling us to decide whether all SUT behaviours have been observed.

FiXme Fatal: I think you need this for the failures refinement tests as well.
FiXme Note: jp: yes, I re-arranged the text accordingly.

FiXme Fatal: And also, if the SUT blocks, the test should block, no?
FiXme Note: jp: you are right, I changed the text here and added explanations above after the verdict explanation in formula (37).

Discussion of Fault Domains As already mentioned, the work in [3] defines a fault domain as the set of processes that refine a given CSP process. In that context, only testing for traces refinement is considered, and the complete test suites may not be finite. So, the work presented here goes well beyond what is achieved there. On the other hand, [3] presents an algorithm for test generation that can be easily adapted to consider additional selection and termination criteria, like, for example, the length of the traces used to construct tests. It would be possible, for instance, to use the bound indicated here. Moreover, specifying a fault domain as a CSP process allows us to model domain-specific knowledge using CSP. For example, if an initialisation component defined by a process I can be regarded as correct without further testing, we can use $I;\ RUN$ as a fault domain, to indicate that any SUT of interest implements I correctly, but afterwards has a potentially arbitrary behaviour specified by RUN. In addition, the work in [3] is not restricted to finite or non-terminating processes.

Implications for CSP Model Checking As explained in the previous sections, passing a test is specified by $(pass \rightarrow Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U(F(j)) \setminus \Sigma$ for failures testing. If the SUT Q is not a programmed piece of software or an integrated hardware or software system, but just another CSP process specification, it is of course possible to verify the pass criterion using the FDR4 model checker. For checking the refinement relation $P \sqsubseteq_F Q$, the pass criterion has to be verified for $j = 0, \ldots, pq - 1$, where p and q indicate the number of nodes in P's transition graph and the maximal number of nodes in Q's graph, respectively (Theorem 2). Since the test cases $U(F(j)$ have such a simple structure, it is an interesting question for further research whether checking $(pass \rightarrow Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U(F(j)) \setminus \Sigma$ for $j = 0, \ldots, pq - 1$ can be faster than directly checking $P \sqsubseteq_F Q$, as one would do in the usual approach with FDR4. This is of particular interest, since the checks could be parallelised on several

CPUs. Alternatively it is interesting to investigate whether the check[4]

$$(pass \rightarrow Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, \bigsqcap_{j=0}^{pq-1} U_F(j)) \setminus \Sigma$$

may perform better than the check of $P \sqsubseteq_F Q$, since the former allows for other optimisations in the model checker than the latter.

For large and complex CSP processes Q, it may be too time consuming to generate its normalised transition graph, so that its number q of nodes is unknown. In such a case the suggested options may still be used as efficient bug finders: use p of reference process P as initial q-value and increment q from there, as long as each increment reveals new errors.

Conclusion In this paper, we have introduced finite complete testing strategies for model-based testing against finite, non-terminating CSP reference models. The strategies are applicable to the conformance relations failures refinement and trace refinement. The underlying fault domains have been defined as the sets of CSP processes whose normalised transition graphs do not have more than a given number of additional nodes, when compared to the transition graph of the reference process. For these domains, finite complete test suites are available. We have shown for the strategy to check failures refinement that the way of probing the SUT for illegal deadlocks used in our test cases is optimal in the sense that it is not possible to guarantee exhaustiveness with fewer probes.

Acknowledgements The authors would like to thank Bill Roscoe and Thomas Gibson-Robinson for their advice on using the FDR4 model checker and for very helpful discussions concerning the potential implications of this paper in the field of model checking. We are grateful to Li-Da Tong from National Sun Yat-sen University, Taiwan, for suggesting the applicability of Sperner's Theorem in the context of the work presented here. Moreover, we thank Adenilso Simao from the University of São Paulo for several helpful suggestions. The work of Ana Cavalcanti is funded by the Royal Academy of Engineering and UK EPSRC Grant EP/R025134/1.

References

1. Buth, B., Kouvaras, M., Peleska, J., Shi, H.: Deadlock analysis for a fault-tolerant system. In: Johnson, M. (ed.) Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1349, pp. 60–74. Springer (1997), https://doi.org/10.1007/BFb0000463
2. Cavalcanti, A., Gaudel, M.: Testing for refinement in CSP. In: Butler, M.J., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) Formal Methods and Software Engineering, 9th International Conference on Formal Engineering Methods, ICFEM 2007, Boca Raton, FL, USA, November 14-15, 2007, Proceedings. Lecture Notes

[4] We are grateful to Bill Roscoe for suggesting this option.

in Computer Science, vol. 4789, pp. 151–170. Springer (2007), https://doi.org/10.1007/978-3-540-76650-6_10
3. Cavalcanti, A., da Silva Simão, A.: Fault-based testing for refinement in CSP. In: Yevtushenko, N., Cavalli, A.R., Yenigün, H. (eds.) Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10533, pp. 21–37. Springer (2017), https://doi.org/10.1007/978-3-319-67549-7_2
4. Dorofeeva, R., El-Fakih, K., Yevtushenko, N.: An improved conformance testing method. In: Wang, F. (ed.) Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3731, pp. 204–218. Springer (2005), https://doi.org/10.1007/11562436_16
5. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: FDR3 — A Modern Refinement Checker for CSP. In: brahm, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 8413, pp. 187–201 (2014)
6. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edn. (1994)
7. Hall, A., Chapman, R.: Correctness by construction: developing a commercial secure system. IEEE Software 19(1), 18–25 (Jan 2002)
8. Hennessy, M.: Algebraic Theory of Processes. MIT Press, Cambridge, MA, USA (1988)
9. Hierons, R.M.: Testing from a nondeterministic finite state machine using adaptive state counting. IEEE Trans. Computers 53(10), 1330–1342 (2004), http://doi.ieeecomputersociety.org/10.1109/TC.2004.85
10. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
11. Huang, W.l., Peleska, J.: Complete model-based equivalence class testing for nondeterministic systems. Formal Aspects of Computing 29(2), 335–364 (2017), http://dx.doi.org/10.1007/s00165-016-0402-2
12. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Jounal 19, 53–77 (1997)
13. Peleska, J., Huang, W.l.: Test Automation - Foundations and Applications of Model-based Testing. University of Bremen (January 2017), lecture notes, available under http://www.informatik.uni-bremen.de/agbs/jp/papers/test-automation-huang-peleska.pdf
14. Peleska, J., Siegel, M.: From testing theory to test driver implementation. In: Gaudel, M., Woodcock, J. (eds.) FME '96: Industrial Benefit and Advances in Formal Methods, Third International Symposium of Formal Methods Europe, Co-Sponsored by IFIP WG 14.3, Oxford, UK, March 18-22, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1051, pp. 538–556. Springer (1996), https://doi.org/10.1007/3-540-60973-3_106
15. Petrenko, A., Yevtushenko, N.: Adaptive testing of deterministic implementations specified by nondeterministic fsms. In: Testing Software and Systems. pp. 162–178. No. 7019 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2011)
16. Petrenko, A., Yevtushenko, N.: Adaptive testing of nondeterministic systems with FSM. In: 15th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2014, Miami Beach, FL, USA, January 9-11, 2014. pp. 224–228. IEEE Computer Society (2014), http://dx.doi.org/10.1109/HASE.2014.39

17. Roscoe, A.W.: Model-checking csp. In: Roscoe, A.W. (ed.) A Classical Mind: Essays in Honour of C. A. R. Hoare, chap. 21, pp. 353–378. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)
18. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
19. Roscoe, A.W.: Understanding Concurrent Systems. Springer, London, Dordrecht Heidelberg, New York (2010)
20. Schneider, S.: An operational semantics for timed csp. Inf. Comput. 116(2), 193–213 (Feb 1995), http://dx.doi.org/10.1006/inco.1995.1014
21. Shi, H., Peleska, J., Kouvaras, M.: Combining methods for the analysis of a fault-tolerant system. In: 1999 Pacific Rim International Symposium on Dependable Computing (PRDC 1999), 16-17 December 1999, Hong Kong. pp. 135–142. IEEE Computer Society (1999), https://doi.org/10.1109/PRDC.1999.816222
22. Shi, L., Cai, X.: An exact fast algorithm for minimum hitting set. In: 2010 Third International Joint Conference on Computational Science and Optimization. vol. 1, pp. 64–67 (May 2010)
23. Simo, A., Petrenko, A., Yevtushenko, N.: On reducing test length for FSMs with extra states. Software Testing, Verification and Reliability 22(6), 435–454 (Sep 2012), https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.452
24. Sperner, E.: Ein Satz über Untermengen einer endlichen Menge. Mathematische Zeitschrift 27(1), 544–548 (Dec 1928), https://doi.org/10.1007/BF01171114