White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of York.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/145963/

Version: Accepted Version

Article:
Peleska, Jan, Huang, W.-l. and Cavalcanti, A. L. C. (2019) Finite complete suites for CSP refinement testing. Science of Computer Programming. pp. 1-23. ISSN: 0167-6423

https://doi.org/10.1016/j.scico.2019.04.004

Reuse
This article is distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) licence. This licence only allows you to download this work and share it with others as long as you credit the authors, but you can't change the article in any way or use it commercially. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

UNIVERSITY OF LEEDS
University of Sheffield
UNIVERSITY of York

Finite Complete Suites for CSP Refinement Testing

Jan Peleska, Wen-ling Huang[1]

Bremen, Germany

Ana Cavalcanti[2]

York, United Kingdom

Abstract

In this paper, new contributions for model-based testing using Communicating Sequential Processes (CSP) are presented. For a finite non-terminating CSP process representing the reference model, finite test suites for checking the conformance relations traces and failures refinement are presented, and their completeness (that is, capability to uncover conformity violations) is proven. The fault domains for which complete failure detection can be guaranteed are specified by means of normalised transition graphs representing the failures semantics of finite-state CSP processes. 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. Moreover, it is shown that the test suites are optimal in two aspects: (a) the maximal length of test traces cannot be further 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, Traces Refinement, Failures Refinement, Complete Test Suites

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 [1, 2].

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; we mention here representative work for finite state machines [3, 4], timed automata [5], *Circus* [6], Timed CSP [7], general labelled transition systems [8], symbolic state machines [9], and Kripke structures [10]. In this article, we tackle *Communicating Sequential Processes (CSP)* [11, 12]. 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 [12] and in the references there. Industrial success has also been reported; see, for example, [13, 14, 15].

Main Contributions. This article 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 using testing.

The main novel contributions in this article may be summarised as follows.

[1]University of Bremen, {peleska,huang}@uni-bremen.de

[2]University of York, ana.cavalcanti@york.ac.uk

1. It is shown that trace or failures conformance can be established with finitely many test cases, provided suitable fault domains are chosen, so that the true behaviour of the SUT is reflected by members of these domains.
2. The definition of these fault domains is based on the well-known normalised transition graphs [16] representing the trace and failures semantics of finite-state CSP processes. A fault domain contains all CSP processes over a given alphabet, whose normalised transition graphs have at most q nodes for some $q \in \mathbb{N}$.
3. Worst-case complexity bounds for the number of test executions to be performed are given.
4. It is shown that the maximal length of the test traces involved cannot be further reduced without losing the test suite's completeness property.
5. Likewise, is shown that the non-deterministic behaviour of the SUT cannot be checked for admissibility with smaller or fewer sets of events.

Related Work. Our results complement and extend work previously published in [17, 18, 19, 20, 21]. None of these provide sufficient conditions for constructing finite complete test suites. So, they also do not provide complexity bounds on the number of test executions needed to establish conformance between an SUT and a reference process. In [21], fault domains are used, but these contain all processes refining a "top" fault domain process. This concept is orthogonal to the one investigated here: members of our fault domain need not be in refinement relation to any other process in the domain. They just adhere to the same upper bound q of nodes in their normalised transition graphs.

The minimal sets of events used for checking nondeterministic behaviour of the SUT used in our article were already suggested in [18, 19, 20]; in the present article, however, they have been identified for the first time as *minimal hitting sets* [22] of minimal acceptances in a given process state, and we establish an upper bound stating how many of these sets need to be checked for the "most extreme form of nondeterminism" that may be exhibited by the SUT.

In [17, 20, 21], the authors devised linear test cases: after running through a preset trace s, test cases for traces refinement check for illegal acceptance of a specific event e, and test cases verifying nondeterministic behaviour check for acceptance of events from some set A. In the present article, we follow the alternative approach proposed in [18, 19] and use *adaptive* test cases. This means that each test case adapts its trace execution to the nondeterministic behaviour of the SUT, checks for trace violations at any point during the test execution, and checks for the acceptance of a given minimal hitting set of events after any legal trace of a test case-specific length.

The adaptive test cases have the advantage that test executions only lead to an inconclusive result if the reference process allows for a nondeterministic choice between deadlock and trace continuation in a certain state. In contrast to this, the linear test cases may lead to many more futile executions with inconclusive results, if the SUT refuses to engage into the next event e from the preset trace s, due to legal nondeterministic choices leading to a refusal of e. Moreover, the preset traces s need to be executed twice according to the strategies devised in [17, 20, 21], because traces refinement and correctness of nondeterministic behaviour are checked by two different sets of test cases.

The approach to specifying fault domains by means of normalised transition graphs has been inspired by the typical method used in the construction of complete test suites for finite state machines (FSMs). There, fault domains typically contain all FSMs over a given alphabet whose number of states does not exceed a given value q [23, 24, 25].

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. A sample test suite is presented in Section 4. Test suites checking traces 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. Our results are discussed in Section 7, where we also conclude. References to further 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 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). This 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 in scope, is denoted by Σ and supposed to be finite. The FDR tool [26] 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 a synchronisation takes place, the process that has offered the event that has occurred 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 the processes P, Q, and R defined below. P is initially ready to engage in the event a, and then makes an internal choice to behave like either Q or R.

$$\begin{aligned} P &= a \rightarrow (Q \sqcap R) \\ Q &= a \rightarrow P \,\Box\, c \rightarrow P \\ R &= b \rightarrow P \,\Box\, c \rightarrow R \end{aligned}$$

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.

The branches of an external choice can be protected by *guards*. The process

$$P(x_1, \ldots, x_n) = \mathrm{bexpr}_1(x1, \ldots, x_n) \& Q(x_1, \ldots, x_n) \,\Box\, \mathrm{bexpr}_2(x1, \ldots, x_n) \& R(x_1, \ldots, x_n)$$

parametrised over $x_1, \ldots, x_n$ branches according to Boolean expressions $\mathrm{bexpr}_1(x1, \ldots, x_n)$ and $\mathrm{bexpr}_2(x1, \ldots, x_n)$ over variables from $\{x_1, \ldots, x_n\}$: if $\mathrm{bexpr}_1(x1, \ldots, x_n)$ evaluates to `true` and $\mathrm{bexpr}_2(x1, \ldots, x_n)$ to `false`, P behaves like Q. If $\mathrm{bexpr}_1(x1, \ldots, x_n)$ evaluates to `false` and $\mathrm{bexpr}_2(x1, \ldots, x_n)$ to `true`, P behaves like R. If both Boolean expressions evaluate to `true`, P behaves as $Q \,\Box\, R$. Finally, if both Boolean expressions evaluate to `false`, P behaves like *Stop*.

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 parallelism 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 [12].

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 those of each operand. For example, for internal choice, $failures(P \sqcap Q) = failures(P) \cup failures(Q)$; see [27, p. 210] for a comprehensive list of these definitions covering $traces(P)$ and $failures(P)$.

Using the notation P/s to denote the behaviour ot the process P after having engaged into the events in the 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 [11, 12]: 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 by $\mathit{failures}(Q) \subseteq \mathit{failures}(P)$. 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 divergence-free processes, failures refinement implies traces 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* as proven in [20].

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

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

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

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

$$\mathit{Ref}(P/s) = \{R' \in 2^{\Sigma} \mid \exists\, R \in \mathit{maxRef}(P/s) : R' \subseteq R\}. \tag{4}$$

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. Nondeterministic behaviour in a given process state is reflected by non-empty intersections between initials and maximal refusals. This is illustrated by the following example.

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 [16], 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 [16] implies that all nodes n in N are reachable by sequences of edges labelled by $e_1 \ldots e_k$ and connecting states $\underline{n}, n_1, \ldots, n_{k-1}, n$, such that

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

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 the events in s in the order they appear. In analogy to $\mathit{traces}(P)$, we use the notation $\mathit{traces}(G(P))$ for the set of finite, initialised paths through $G(P)$, each path represented by its finite sequence of edge labels. We note that $\mathit{traces}(P) = \mathit{traces}(G(P))$. 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 [16] 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,

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

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 useful. While the original introduction of acceptances presented in [17, pp. 75] was independent of refusals, we use the definition from [27, 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

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

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

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

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 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 the set $[P/s]^0 - A$ and *always* refusing those in the set $\Sigma - [P/s]^0$.

Exploiting (6), the nodes of a normalised transition graph can alternatively be labelled with minimal acceptances; this captured the same information conveyed by 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, it can be re-constructed from its graph $G(P)$ with initial state $\underline{n}$ and transition function t, using P's normalised syntactic representation [27, 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, it is established that P is semantically equivalent to normalised(P) in the 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 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.

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 $R/c = R$, and so it is labelled by R's minimal acceptance $\{b, c\}$. □

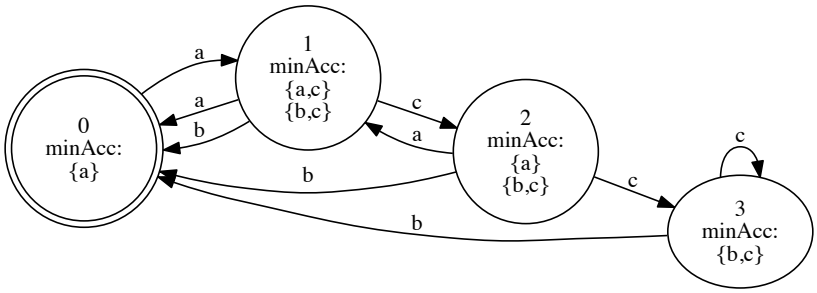

Figure 1: Normalised transition graph of CSP process P from Example 3.

Summarising, refinement between finite-state CSP processes P, Q can be expressed using their normalised graphs

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

as established by the results in the following lemma. There, result (9) reflects traces refinement in terms of graph traces; result (2) expresses failures refinement in terms of traces refinement and *conf*; result (10) states how *conf* can be expressed by means of the maximal refusal functions of the graphs involved; and result (11) states the same in terms of the minimal acceptances that can be derived from the maximal refusal functions by means of (8).

Lemma 1.

$$P \sqsubseteq_T Q \quad \Leftrightarrow \quad traces(G(Q)) \subseteq traces(G(P)) \tag{9}$$

$$\begin{aligned} Q \; conf \; P \quad &\Leftrightarrow \quad \forall\, s \in traces(G(Q)) \cap traces(G(P)), R_Q \in r_Q(G(Q)/s) : \\ &\qquad \exists\, R_P \in r_P(G(P)/s) : R_Q \subseteq R_P \end{aligned} \tag{10}$$

$$\begin{aligned} &\Leftrightarrow \quad \forall\, s \in traces(G(Q)) \cap traces(G(P)), A_Q \in minAcc(G(Q)/s) : \\ &\qquad \exists\, A_P \in minAcc(G(P)/s) : A_P \subseteq A_Q \end{aligned} \tag{11}$$

□

Proof. To prove (9), we recall that $P \sqsubseteq_T Q$ is defined as $traces(Q) \subseteq traces(P)$ and, moreover, $traces(P) = traces(G(P))$ and $traces(Q) = traces(G(Q))$. To prove (10), we derive

$$\begin{aligned} &Q \; conf \; P \\ &\Leftrightarrow \forall\, s \in traces(P) \cap traces(Q) : Ref(Q/s) \subseteq Ref(P/s) && \text{[Definition of } conf \text{ (1)]} \\ &\Leftrightarrow \forall\, s \in traces(G(P)) \cap traces(G(Q)) : Ref(Q/s) \subseteq Ref(P/s) \\ &\qquad\qquad && [traces(P) = traces(G(P)), traces(Q) = traces(G(Q))] \\ &\Leftrightarrow \forall\, s \in traces(G(P)) \cap traces(G(Q)), R_Q \in r_Q(G(Q)/s) : \exists\, R_P \in r_P(G(P)/s) : R_Q \subseteq R_P \\ &\qquad\qquad && \text{[Property of } r_P, r_Q \text{ (subset closure) and (4)]} \end{aligned}$$

Finally, (11) follows from (10) 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 $G(P)$ with nodes in set N, we suppose that $V \subseteq \Sigma^*$ is a prefix-closed set of sequences of events. By $t(\underline{n}, V)$ we denote the set

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

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

$$G_1 \times G_2 = (N_1 \times N_2, (\underline{n}_1, \underline{n}_2), t : (N_1 \times N_2) \times \Sigma \nrightarrow (N_1 \times N_2)) \tag{12}$$

$$\text{dom } t = \{((n_1, n_2), e) \in (N_1 \times N_2) \times \Sigma \mid (n_1, e) \in \text{dom } t_1 \wedge (n_2, e) \in \text{dom } t_2\} \tag{13}$$

$$t((n_1, n_2), e) = (t_1(n_1, e), t_2(n_2, e)) \text{ for } ((n_1, n_2), e) \in \text{dom } t \tag{14}$$

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 results regarding its semantics 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.

$$\forall A \in C : H \cap A \neq \varnothing. \tag{15}$$

A *minimal hitting set* is a hitting set that cannot be further reduced without losing the characteristic property (15). 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 using minimal hitting sets, we can reduce 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 Minimal Acceptances. In this article, we are interested in the minimal hitting sets of minimal acceptances; for these, the abbreviated notation $minHit(P/s) = minHit(minAcc(P/s))$ is used. The minimal hitting sets of minimal acceptances may be alternatively characterised by means of the failures of a process as is done in [20]. To this end, in [20], the authors define, for any collection $C \subseteq 2^\Sigma$ of subsets from Σ

$$min_\subseteq(C) = \{A \in C \mid \forall B \in C : B \subseteq A \Rightarrow B = A\}. \tag{16}$$

The collection $min_\subseteq(C)$ contains all those sets of C that are not true supersets of other members of C. With this definition, the relation between failures and minimal hitting sets of minimal acceptances is established in the following lemma.

Lemma 4. *For any trace s of CSP process P, define $\mathcal{A}_s = \{A \subseteq \Sigma \mid (s, A) \notin failures(P)\}$. Then $minHit(P/s) = min_\subseteq(\mathcal{A}_s)$ for all traces s of P.*

Proof. We derive

$$\begin{aligned}
&(s, A) \notin failures(P)\\
&\Leftrightarrow A \notin Ref(P/s)\\
&\Leftrightarrow \forall R \in maxRef(P/s) : A \not\subseteq R\\
&\Leftrightarrow \forall B \in minAcc(P/s) : A \not\subseteq (\Sigma - B) && \text{[this follows from (6)]}\\
&\Leftrightarrow \forall B \in minAcc(P/s) : A \cap B \neq \varnothing\\
&\Leftrightarrow A \text{ is a (not necessarily minimal) hitting set of } minAcc(P/s)
\end{aligned}$$

This derivation is valid for arbitrary A satisfying $(s, A) \notin$ failures(P). Specialising it to *minimal* sets A satisfying $(s, A) \notin$ failures(P) proves the statement of the lemma. □

Minimal Hitting Sets of Normalised Transition Graphs. Since, as previously explained, minimal acceptances can be used to label the nodes of a normalised transition graph, and since $minAcc(P/s) = minAcc(G(P)/s)$ by (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

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

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 5. *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 necessarily minimal) hitting set of $minAcc(Q/s)$.*

Proof. We apply (8) and (17), so that $minAcc(P/s)$ and $minAcc(G(P)/s)$, as well as $minHit(P/s)$ and $minHit(G(P)/s)$ are used interchangeably. For showing "(1) ⇒ (2)", we assume Q *conf* P and $s \in traces(P) \cap traces(Q)$. Lemma 1 (11), 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. So, each $H \in minHit(P/s)$ is also a hitting set for $minAcc(G(Q)/s)$ as required.

To prove "(2) ⇒ (1)", we assume that (2) holds, but that P *conf* Q does *not* hold. According to Lemma 1, (11), 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. □

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 (11) 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 5 covers the situations where $minAcc(P/s) = \{\varnothing\}$ and so $minHit(P/s) = \varnothing$. Trivially,

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

holds.

2.3. Sperner Families

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

A *Sperner Family* is a collection $\mathcal{S} \subseteq 2^{\Sigma}$ 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 $H_1 \neq H_2 \in \mathcal{S}$. Specialising antichains known from partial orders to finite sets partially ordered by $\subseteq$ results in Sperner families. Given an *arbitrary* collection of subsets $C \subset 2^{\Sigma}$, the sub-collection $min_{\subseteq}(C)$ defined in (16) is a Sperner Family contained in C.

We further observe that

- 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 $P = \bigsqcap_{A \in S} (e : A \rightarrow P(e))$.

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

Theorem 1 (Sperner's Theorem [28]). *Given a Sperner family $\mathcal{S}$ over an n-element universe Σ, its cardinality $|\mathcal{S}|$ is bound by*

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

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

1. *For even n, if $\mathcal{S}$ consists of all subsets of Σ with cardinality $n/2$;*
2. *For odd n, if one of the following cases holds;*
   (a) *$\mathcal{S}$ consists of all subsets of Σ with cardinality $(n+1)/2$; or*
   (b) *$\mathcal{S}$ 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 the Sperner Family containing the 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 [17]. 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.

$$\begin{aligned}
U_F(j) &= U_F(j, 0, \underline{n}) && (19)\\
U_F(j,k,n) &= (e : (\Sigma - [n]^0) \to \mathit{fail} \to \mathit{Stop}) && (20)\\
&\quad \Box && \\
&\quad (\mathit{minHit}(n) = \varnothing) \& (\mathit{pass} \to \mathit{Stop}) && (21)\\
&\quad \Box && \\
&\quad (k < j) \& (e : [n]^0 \to U_F(j, k+1, t(n,e)) && (22)\\
&\quad \Box && \\
&\quad (k = j \wedge \mathit{minHit}(n) \neq \varnothing) \& \quad (\textstyle\bigsqcap_{H \in \mathit{minHit}(n)} (e : H \to \mathit{pass} \to \mathit{Stop})) && (23)
\end{aligned}$$

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 verdicts 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, since the only process executing concurrently with $U_F(j)$ is livelock-free Q.

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*, because every test execution must end with the *pass* event [17].

It is necessary to use failures refinement in the definition above, and not just traces 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 whose length $\#s$ satisfies $\#s < j$, such that $\mathit{minHit}(G(P)/s) \neq \varnothing$, so that the process $(Q \,|[\,\Sigma\,]|\, U_F(j))/s$ deadlocks as well.

Intuitively, $U_F(j)$ is able to perform any trace s of P, up to a length j. If, after having already run through s with $\#s \leqslant j$, the SUT accepts an event outside the initials of P/s (recall from Lemma 6 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 6). In this case, the test may terminate successfully (branch (21) of the choice in $U_F(j, \#s, G(P)/s)$). If P/s may also nondeterministically engage into events, branch (22) is simultaneously enabled. If Q/s is able to engage into an event in $\Sigma - [P/s]^0$, 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 $U_F(j, \#s+1, G(P)/s.e)$ in branch (22); this follows from Lemma 6 (note that $G(P)/s.e = t(G(P)/s, e)$). A test of this type is called *adaptive*, because it accepts any legal behaviour of the SUT, here any event from $[P/s]^0$, and adapts its consecutive behaviour to the event selected by the SUT, here $U_F(j, \#s+1, G(P)/s.e)$.

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 $H \in minHit(n)$ 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 5, a conforming SUT should accept at least one event of each minimal hitting set in $minHit(n)$. Therefore, the test execution terminates with *pass*, only if such an event is accepted. 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 $s \in traces(Q)$ 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, then there are two possibilities. The first is that the process accepts another event e of some minimal hitting set $H \in minHit(n)$ with $n = G(P)/s$ according to Lemma 6. In this case, the final length of s is $j+1$, and the execution terminates with *pass*. Alternatively, 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 $(pass \rightarrow Stop) \not\sqsubseteq_F (Q \,|[\,\Sigma\,]|\, U_F(j)) \setminus \Sigma$.

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 $minHit(n)$ are finite. Moreover, we further recall that $minHit(n)$ may be empty, in which case the indexed internal choice in (23) would be undefined. The guard in that branch, however, requires $minHit(n) \neq \varnothing$, and branches (20) or (21) can be taken in this situation.

The following lemma establishes relationships between $U_F(j)$ and the reference process P from which it is derived.

Lemma 6. *If $s \in traces(P)$ satisfies $\#s \leqslant j$, then $s, s.e \in traces(U_F(j))$ for all $e \in \Sigma$, and the following properties hold.*

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

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

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

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

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 $t(G(P)/s, e) = G(P)/s.e$. So, the new recursion in branch (22) is 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 $U_F(j)/s.e = U_F(j, \#s, G(P)/s)/e = (fail \rightarrow Stop)$.

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), $minHit(G(P)/s) = minHit(P/s)$ 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 6, if $minHit(P/u) = \varnothing$ 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 6 just states the existence of $U_F(j)$-executions $s, s.e$ satisfying the properties stated there.

Complete Testing Assumption. As explained above, passing a test case $U_F(j)$ requires that none of the possible executions $(Q \,|[\,\Sigma\,]|\, U_F(j))$ stops after *fail* or stops without having produced the event *pass*. Therefore, it is necessary to determine whether all possible executions have been covered in the repeated runs of $(Q \,|[\,\Sigma\,]|\, U_F(j))$. The theoretical completeness results are, therefore, based on a *complete testing assumption* [3, 20], 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 behaviours of the SUT have been observed. Therefore, testing nondeterministic SUTs comes at the price of having to apply some grey-box testing techniques.

3.2. A Finite Complete Test Suite for Failures Refinement

A CSP *fault model* $\mathcal{F} = (P, \sqsubseteq, \mathcal{D})$ consists 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*

$$TS_F = \{U_F(j) \mid 0 \leqslant j < pq\} \quad \textit{with } U_F(j) \textit{ as specified in (19)}$$

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

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

Lemma 7. *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 (2). 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 6 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. For proving that a test execution can never deadlock without a *pass* event, it has to be shown that a test execution can neither block at branch (22) nor at branch (23). These cases are considered separately below.

Step 2.1. Suppose that the test execution blocks at branch (22) after having run through a trace s with $\#s < j$. Since $P \sqsubseteq_T Q$ by assumption, s is a trace of P, thus $U_F(j)/s = U_F(j, \#s, G(P)/s)$ according to Lemma 6. Therefore, $U_F(j)/s$ can enter branch (22) with any event from $[G(P)/s]^0$. Since we assume that $(Q \,|[\, \Sigma \,]|\, U_F(j))/s$ deadlocks, this means that $[G(P)/s]^0$ is *not* a hitting set of $minAcc(Q/s)$, because otherwise at least one $e \in [G(P)/s]^0$ would be accepted by Q/s and the test execution would not deadlock. Now suppose that $minHit(G(P)/s) = \varnothing$. Then branch (21) can be entered, and the test stops after *pass*. Otherwise, if $minHit(G(P)/s) \neq \varnothing$, let $H \in minHit(G(P)/s)$. Since H contains only elements that are contained in some minimal acceptance of P/s, and all these minimal acceptances are subsets of $[G(P)/s]^0$, H is a subset of $[G(P)/s]^0$ as well. Since $[G(P)/s]^0$, however, is not a hitting set of $minAcc(Q/s)$, also H is not a hitting set of $minAcc(Q/s)$. Now this is a contradiction to Lemma 5, since Q *conf* P by assumption, so H should also be a (not necessarily minimal) hitting set in $minAcc(Q/s)$. This proves that the test execution cannot block at branch (22) without being able to pass the test by entering branch (21).

Step 2.2. Suppose that the execution blocks at branch (23) after having run through some $s \in traces(Q) \subseteq traces(P)$ with $\#s = j$. From Lemma 6 we know that $U_F(j)/s = U_F(j,k,n) = U_F(j,\#s,G(P)/s)$, so $minHit(n) = minHit(P/s)$. Branch (21) of $U_F(j,k,n)$ leads always to a PASS verdict and is taken if $minHit(n) = \varnothing$. If $minHit(n) \neq \varnothing$, the assumption that $(Q \,|[\,\Sigma\,]|\, U_F(j))/s$ blocks at branch (23) implies that there exists some $H \in minHit(n)$ that is not a hitting set of $minAcc(Q/s)$. Again, by Lemma 5, this contradicts the assumption that Q *conf* P. As a consequence, the test execution can never deadlock at branch (23) without entering branch (21) and passing the test.

Note that the line of reasoning in this proof requires that Q is free of livelocks, because otherwise a *pass* event might not become visible, due to unbounded sequences of hidden events performed by Q. □

Lemma 8. *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 (2), 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, (11))

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

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 $[P/s]^0$ (the violation described by Case 1) or identifying an illegal acceptance A_Q (as in Case 2). It is not guaranteed, however, that s is short enough to be executed by one of the test cases $U_F(j)$ with $0 \leqslant j < pq$. So, it has to be shown that for any s leading to an error situation, 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 6, $U_F(pq-1)/u = U_F(pq-1, \#u, G(P)/u)$. 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.

Case 2. We note again that s is a trace of the product graph G, but we do not know its length. Again, by Lemma 3, 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 6. $U_F(\#u)$ can always perform branch (22) until the trace u has been completely processed. $U_F(\#u, \#u, G(P)/u)$ may execute branches (20) or (23) only: (29) implies that P/s has at least one non-empty minimal acceptance. By (18) 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 $(k < j)$ 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 deadlocks and therefore fails for the execution where Q/u selects the minimal acceptance A_Q as specified in (29) and $U_F(\#u, \#u, G(P)/u)$ selects a minimal hitting set $H \in minHit(P/u)$ that has an empty intersection with A_Q. The existence of such an H is guaranteed because of Lemma 5. As a consequence, $(Q \,|[\,\Sigma\,]|\, U_F(\#u))/u$ cannot produce the *pass* event in this execution; this means that the test fails. The complete testing assumption guarantees that this execution really occurs if $(Q \,|[\,\Sigma\,]|\, U_F(\#u))$ is executed sufficiently often. This concludes the proof. □

Our notion of test can be specialised to deal with traces refinement (see Section 5). We next present an example.

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.

$$
\begin{array}{lcl}
[0]^0 & = & \{a\} \\
[1]^0 & = & \{a,b,c\} \\
[2]^0 & = & \{a,b,c\} \\
[3]^0 & = & \{b,c\}
\end{array}
\qquad
\begin{array}{lcl}
minHit(0) & = & \{\{a\}\} \\
minHit(1) & = & \{\{a,b\},\{c\}\} \\
minHit(2) & = & \{\{a,b\},\{a,c\}\} \\
minHit(3) & = & \{\{b\},\{c\}\}
\end{array}
\qquad
\begin{array}{lcl}
t(0,a) & = & 1 \\
t(1,a) & = & 0 \\
t(1,b) & = & 0 \\
t(1,c) & = & 2
\end{array}
\qquad
\begin{array}{lcl}
t(2,a) & = & 1 \\
t(2,b) & = & 0 \\
t(2,c) & = & 3 \\
t(3,b) & = & 0 \\
t(3,c) & = & 3
\end{array}
$$

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

$$
\begin{array}{rcl}
Z & = & a \rightarrow (Q_1 \sqcap R_1(r_{max}, 0)) \\
Q_1 & = & a \rightarrow Z \,\Box\, c \rightarrow Z \\
R_1(r_{max}, k) & = & (k < r_{max}) \& (b \rightarrow Z \,\Box\, c \rightarrow R_1(r_{max}, k+1)) \\
 & & \Box \\
 & & (k = r_{max}) \& (b \rightarrow Z \sqcap c \rightarrow R_1(r_{max}, r_{max}))
\end{array}
$$

It 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 $(Z \,[|\, \Sigma \,|]\, U_F(4))/s$

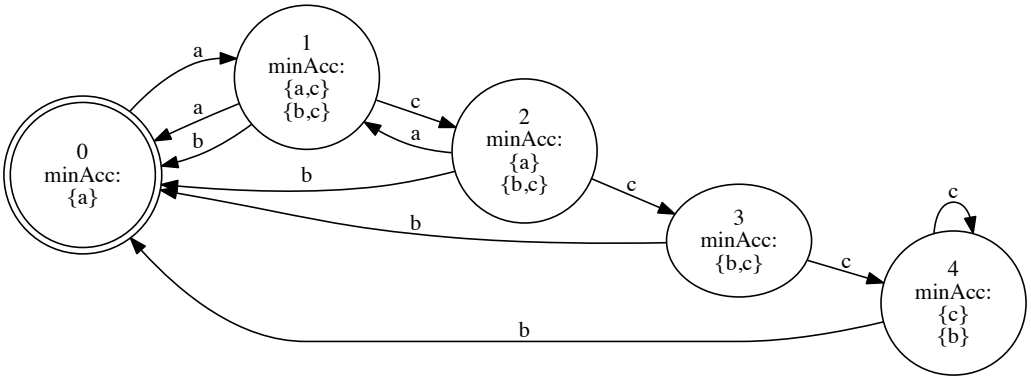

Figure 3: Normalised transition graph of faulty implementation Z for $r_{\max} = 3$ from Example 4.

deadlocks, and the *pass* event cannot be produced. Another failing execution arises if Z/s chooses to accept only $\{c\}$, while $U_F(4)/s$ chooses to accept only $\{a, b\}$. Therefore, $(pass \rightarrow Stop) \not\sqsubseteq_F (Z \,[\![\, \Sigma \,]\!]\, U_F(4)) \setminus \Sigma$, and the test fails. □

5. Finite Complete Test Suites for CSP Traces Refinement

For establishing traces refinement, the class of adaptive test cases specified below in (30) – (34) is 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 traces refinement are derived from the reference model's transition graph

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

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. We use the condition on acceptances $minAcc(n) = \{\varnothing\}$ instead of the condition on hitting sets $minHit(n) = \varnothing$ in branch (32). From (18) we know that these conditions are equivalent, but, by using expression $minAcc(n) = \{\varnothing\}$, we make it unnecessary to calculate hitting sets for generating these tests from $G(P)$, which is expensive.

$$\begin{aligned}
U_T(j) &= U_T(j, 0, \underline{n}) && (30)\\
U_T(j, k, n) &= (e : (\Sigma - [n]^0) \rightarrow fail \rightarrow Stop) && (31)\\
&\quad \Box && \\
&\quad (minAcc(n) = \{\varnothing\}) \& (pass \rightarrow Stop) && (32)\\
&\quad \Box && \\
&\quad (k < j) \& (e : [P/s]^0 \rightarrow U_T(j, k + 1, t(n, e))) && (33)\\
&\quad \Box && \\
&\quad (k = j) \& (pass \rightarrow Stop) && (34)
\end{aligned}$$

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

$$\begin{aligned}
U_T(j)/s &= U_T(j, \#s, G(P)/s) && (35)\\
e \notin [P/s]^0 \quad &\Rightarrow \quad U_T(j)/s.e = (fail \rightarrow Stop) && (36)
\end{aligned}$$

proven in Lemma 6 for $U_F(j)$ for traces $s \in traces(P)$ with $\#s \leqslant j$.

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

$$Q \underline{\text{ pass }} U_T(j) \mathrel{\widehat{=}} (pass \rightarrow Stop) \sqsubseteq_T (Q \,[\![\, \Sigma \,]\!]\, U_T(j)) \setminus \Sigma \qquad (37)$$

If the SUT process Q deadlocks after a trace s, and in this case the reference process P is also in a state where deadlock is possible, this is captured by the fact that $minAcc(n) = \{\varnothing\}$ for $n = G(P)/s$. Therefore, 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' and 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 $(Q \,[\![\, \Sigma \,]\!]\, U_T(j)) \setminus \Sigma = Stop$, the test passes, because $(pass \rightarrow Stop) \sqsubseteq_T Stop$ holds.

The existence of complete, finite test suites is expressed in analogy to Theorem 2. A noteworthy difference is that the complete suite for traces 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 for Theorem 2, the proof is structured in two lemmas, the first ensuring soundness, and the second exhaustiveness.

Lemma 9. *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 conclude

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

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 $minAcc(G(P)/s) = \{\varnothing\}$. In this case, the test $U_T(pq-1, \#s, G(P)/s)$ enters branch (32), and its execution stops after *pass*.

Case 2. Q/s deadlocks, but $minAcc(G(P)/s) \neq \{\varnothing\}$. In this case, the whole test execution deadlocks, 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 $\#s < pq-1$. In this case, the test $U_T(pq-1)$ in state $U_T(pq-1, \#s, G(P)/s)$ can also engage in e by entering branch (33), and its execution continues without producing a *pass* or a *fail* event.

Case 4. $\#s = pq-1$ holds. In this case, $U_T(pq-1, \#s, G(P)/s)$ can 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 10. *A test suite TS_T specified as in Theorem 3 is exhaustive for the fault model specified there.*

Proof. As in the proof for failures testing, we construct the product graph $G = G(P) \times G(Q)$ and recall that every trace $s \in traces(P) \cap traces(Q)$ 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 the 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 $P \not\sqsubseteq_T Q$. Since the empty trace is a trace of every process, there exists a trace $s \in traces(Q) \cap traces(P)$ and an event $e \in [Q/s]^0$ such that $e \notin [P/s]^0$. Let $u \in traces(Q) \cap traces(P)$ 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 $U_T(pq-1, \#u, G(P)/s)$ can enter branch (31), and the test execution stops after having produced *fail*. This proves that Q fails test $U_T(pq-1)$. □

Having established completeness of our test suites, we consider complexity of a testing technique that uses them.

6. Complexity Considerations

With finite complete CSP test suites test suites at hand, it is now possible to calculate how many test executions are needed when using them. Previous work did not consider sufficient conditions for finiteness, so estimates for the maximal number of executions could not be calculated. We answer the following questions. (1) What is the worst-case bound on the number of test executions to be performed to verify an SUT with respect to failures refinement, when we use our test suite? (2) What is the worst-case bound for traces refinement? (3) Is it possible to reduce the maximal length of traces when testing for failures or traces refinement? We consider the first question (1) in Section 6.1, where we also discuss whether it is possible to reduce the number of test executions with a different test suite. With the answer to question (1), question (2) is a fairly simple consequence we discuss briefly at the end of Section 6.1. Question (3) is the subject of Section 6.2.

6.1. Estimates for the Maximal Number of Failures Test Executions

An arbitrary CSP process P might have $minHit(P/s) = \varnothing$ for some traces s, so that a test case $U_F(j)$ for a j greater than the size of s can enter branch (21). In this case, further executions are needed to consider traces that have s as a prefix. To provide a bound on the number of test executions needed, we first define a process $P_{\max}$ (see (39)), which, when used as a reference process, requires the maximal number of test executions among all reference processes P fulfilling $minHit(P/s) \neq \varnothing$ for all traces s. For $P_{\max}$, we can establish the actual number of test executions required (see (47)). We then show that the order of magnitude of the worst-case bound for the number of test executions is the same also for reference processes P that may have $minHit(P/s) = \varnothing$ for some s.

A Reference Process. Given an alphabet Σ of size $|\Sigma| = n \geqslant 2$, define a collection of subsets of Σ by

$$C = \{A \subseteq \Sigma \mid |A| = n - \lfloor \frac{n}{2} \rfloor + 1\}. \tag{38}$$

With this choice of C, define

$$P_{\max} = \bigsqcap_{A \in C} e : A \to P_{\max} \tag{39}$$

The relevant properties of $P_{\max}$ are summarised in the following lemma.

Lemma 11. *Given alphabet Σ with cardinality $|\Sigma| = n \geqslant 2$, process P_{max} fulfils*

$$[P_{max}/s]^0 = \Sigma \quad \textit{for all } s \in \Sigma^* \tag{40}$$

$$traces(P_{max}) = \Sigma^* \tag{41}$$

$$minAcc(P_{max}/s) = C \quad \textit{for all } s \in \Sigma^* \tag{42}$$

$$minHit(P_{max}/s) = minHit(C) \quad \textit{for all } s \in \Sigma^* \tag{43}$$

$$|minHit(P_{max}/s)| = \binom{n}{\lfloor \frac{n}{2} \rfloor} \quad \textit{for all } s \in \Sigma^* \tag{44}$$

$$minHit(C) = \{H \subseteq \Sigma \mid |H| = \lfloor \frac{n}{2} \rfloor\} \tag{45}$$

Proof. Since $\bigcup_{A \in C} A = \Sigma$ by construction of C, $[P_{\max}/s]^0 = \Sigma$ as stated by (40). Since $P_{\max}/e = P_{\max}$ for all $e \in \Sigma$, this proves statement (41). The internal choice construct used in the specification of $P_{\max}$ implies $minAcc(P_{\max}) = C$. Again, $P_{\max}/e = P_{\max}$ for all $e \in \Sigma$ implies $minAcc(P_{\max}/s) = C$ for all traces of $P_{\max}$, so this shows (42). Statement (43) is a direct consequence of (42). 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; this shows (45) and $|minHit(C)| = \binom{n}{\lfloor \frac{n}{2} \rfloor}$. The latter shows (44) and completes the proof. □

Test Cases of P_{max}. The test cases $U_F(j)$ generated from $P_{\max}$ can never enter branch (20), because $P_{\max}/s$ has initials Σ for all traces $s \in traces(P_{\max})$ according to (40). Moreover, they can never enter branch (21), because $minHit(P_{\max}/s)$ is never empty according to (44). Finally, the minimal hitting sets used to probe the SUT at the end of a non-blocking test execution are always the hitting sets of C according to (43). This results in the following test case structure.

$$\begin{aligned} U_F(j) &= U_F(j, 0, \underline{n}) \\ U_F(j,k,n) &= (k < j)\&(e : \Sigma \rightarrow U_F(j, k+1, t(n,e)) \\ &\quad \Box \\ &\quad (k = j)\&(\bigsqcap_{H \in minHit(C)}(e : H \rightarrow pass \rightarrow Stop)) \end{aligned}$$

This means that the branches of $U_F(j)$ that can lead to an early termination are not feasible. All tests deadlock, or run to the end of a trace of size j and then present a choice of events from a minimal hitting set.

Moreover, Theorem 1 establishes that given an alphabet with n elements, there is no Sperner family consisting of more than $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ members. In addition, we recall the minimal hitting sets calculated from a given set of minimal acceptances are a Sperner family (see discussion in Section 2.3). So, (44) establishes that there is no possibility of providing more choices from minimal hitting sets with a test derived from a process different from $P_{\max}$.

In summary, the tests derived from $P_{\max}$ require the most test executions when compared to tests derived from any other CSP process P whose collections of minimal hitting sets $minHit(P/s)$ are never empty for any trace s.

Maximal Number of Test Executions for P_{max}. When considering the number of test executions to be performed using $U_F(j)$ derived from $P_{\max}$ against some SUT Q for all $j = 0, \ldots, pq - 1$, and considering that we need to cover all the possible behaviours of Q, the maximal number of test executions is only reached if (a) Q is a correct refinement of $P_{\max}$ (or more generally, of the reference process) and (b) its traces are Σ^*. In such a situation, no test execution blocks early, because Q/s can always engage into some $e \in \Sigma$ while $\#s < j$, and never blocks in the last step when $\#s = j$ and a hitting set $H \in minHit(C)$ is offered by the test case. The resulting number of executions in this case is

$$n^j \cdot \binom{n}{\lfloor \frac{n}{2} \rfloor}, \tag{46}$$

because all traces up to length j can be executed with $U_F(j)$ entering branch (22), and each of these traces is followed by one event from each of the hitting sets of C since Q is correct.

The number of executions in (46) is indeed maximal for all reference processes P fulfilling $minHit(P/s) \neq \varnothing$ for all traces s. All these processes can never enter branch (21), and, if an execution with an SUT entered branch (20), this would only lead to early termination of the whole test suite, because a failure has been detected. As a consequence, $|\Sigma|^j$ is the maximal number of traces to be executed up to length j, and from Theorem 1 we know that the number $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ of hitting sets to be tested at the end of each trace of length j is already maximal.

Summing up formula (46) over all test cases $U_F(0), \ldots, U_F(pq-1)$ to be executed and applying the formula for the sum of a geometric progression, this results int

$$\sum_{j=0}^{pq-1} n^j \cdot \binom{n}{\lfloor \frac{n}{2} \rfloor} = \binom{n}{\lfloor \frac{n}{2} \rfloor} \cdot \frac{1 - n^{pq}}{1 - n} \quad \text{with } n = |\Sigma| \tag{47}$$

as the maximal number of test executions to be performed when testing an error-free SUT Q with $traces(Q) = \Sigma^*$ against the reference process $P_{\max}$. If we are interested only in the order of magnitude, we have

$$O(\binom{n}{\lfloor \frac{n}{2} \rfloor} \cdot n^{pq-1}) \quad \text{with } n = |\Sigma|\,. \tag{48}$$

Considering Empty Collections of Minimal Hitting Sets. The argument so far has shown that the tests derived from $P_{\max}$ require the most test executions when considering processes whose collections of minimal hitting sets are never empty. It remains to consider whether reference processes Z possessing failures (s, Σ) may require more test executions for their associated tests $U_F(j)$ than the bound given for $P_{\max}$ in (47), because process states Z/s with $minHit(Z/s) = \varnothing$ allow for test executions entering branch (21). To this end, consider a test case $U_F(j)$ constructed from such a process Z. Every trace $s \in traces(Z)$ with $\#s < j$ ending in a process state Z/s with $minHit(Z/s) = \varnothing$ allows for

- one execution of branch (21), where the test execution $(Z\,|[\,\Sigma\,]|\,U_F(j))/s$ stops after *pass*, and
- $|[Z/s]^0|$ continuations of the test execution with events $e \in [Z/s]^0$.

For every trace $s \in traces(Z)$ with $\#s = j$,

- one execution of branch (21) follows if $minHit(Z/s) = \varnothing$, and otherwise
- $|minHit(Z/s)|$ executions checking acceptance of minimal hitting sets.

For a rough estimate of the worst-case upper bound suppose that

1. $[Z/s]^0 = \Sigma$ for all traces s of Z,
2. all traces s with $\#s < j$ end in a state with empty minimal hitting sets, and
3. all traces s with $\#s = j$ end in a state with a maximal number $\binom{n}{\lfloor n/2 \rfloor}$ of hitting sets.

We note that this scenario is not feasible for all $j \in \{0, \ldots, pq - 1\}$, because the traces of $U_F(p - 1)$ already cover all states of Z's transition graph according to Lemma 2, and if all states of Z have empty hitting sets, there are no acceptance checks to be performed in the last step of the test execution. Therefore, the upper bound calculated next

cannot be reached by a CSP process. With the three assumptions above, nevertheless, we can calculate that

- $U_F(0)$ has $\binom{n}{\lfloor n/2 \rfloor}$ executions,
- $U_F(j)$, for $j > 0$ has $\sum_{i=0}^{j-1} n^i = \frac{n^j-1}{n-1}$ executions of branch (21), ($n = |\Sigma|$), and
- $U_F(j)$, for $j > 0$ has $\binom{n}{\lfloor n/2 \rfloor} \cdot n^j$ executions where the acceptance of hitting sets is checked after having run through a trace of length j.

Summing up over all $U_F(j)$ for $j = 0, \ldots, pq-1$, an upper bound B may be calculated as follows.

$$\begin{aligned} B &= \binom{n}{\lfloor n/2 \rfloor} + \sum_{j=1}^{pq-1} \frac{n^j - 1}{n-1} + \sum_{j=1}^{pq-1} \binom{n}{\lfloor n/2 \rfloor} \cdot n^j \\ &= \sum_{j=1}^{pq-1} \frac{n^j - 1}{n-1} + \sum_{j=0}^{pq-1} \binom{n}{\lfloor n/2 \rfloor} \cdot n^j \\ &= \frac{n^{pq} - npq + pq - 1}{(n-1)^2} + \binom{n}{\lfloor n/2 \rfloor} \cdot \frac{n^{pq} - 1}{n-1} \\ &= \\ &= \frac{\binom{n}{\lfloor \frac{n}{2} \rfloor}(n-1)\,(n^{pq}-1) + n^{pq} - npq + pq - 1}{(n-1)^2} \end{aligned}$$

Since B cannot be reached anyway, we just calculate its order of magnitude, and this results again in $O(\binom{n}{\lfloor \frac{n}{2} \rfloor} \cdot n^{pq-1})$, as calculated already for $P_{\max}$ in (48). Summarising these complexity calculations, we have the following theorem.

Theorem 4. *Given a process alphabet Σ, consider a fault model $\mathcal{F} = (P, \sqsubseteq_F, \mathcal{D})$, such that the normalised transition graph of P has p states, and the fault domain $\mathcal{D}$ contains all processes Q over alphabet Σ, such that $G(Q)$ has at most $q \geqslant p$ states. Then the maximal number of test executions to be performed using the complete test suite $TS_F = \{U_F(j) \mid 0 \leqslant j < pq\}$ created from P as specified in Theorem 2 is of order*

$$O(\binom{n}{\lfloor \frac{n}{2} \rfloor} \cdot n^{pq-1}) \quad \textit{with } n = |\Sigma|\,.$$

For processes P satisfying $(s, \Sigma) \notin \mathit{failures}(P)$ for all traces s, the reachable precise upper bound is given by

$$\binom{n}{\lfloor \frac{n}{2} \rfloor} \cdot \frac{1 - n^{pq}}{1 - n} \quad \textit{with } n = |\Sigma|\,.$$

□

As established by Lemma 5, the number of hitting sets used to probe the SUT cannot be reduced: if the reference process is $P_{\max}$, we need to consider all of them, otherwise illegal blocking may remain undetected. In addition, if $P_{\max}$ defines the behaviour of the SUT, using smaller sets that are no longer hitting sets lead to a rejection of correct implementations. Our explicit definition of $P_{\max}$ to establish this worst-case is useful to illustrate this point.

In [17], 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 $2^{\mathbb{N}} - 1$ for the number of different sets to be offered to the SUT in the last step of the test execution. This number is significantly larger than the worst-case estimate $\binom{n}{\lfloor n/2 \rfloor}$ above. In Fig. 4, the reduction is visualised by plots of the two functions. In [20], the authors also use minimal hitting sets[3], but do not give an estimate for the number of test executions.

[3] However, they are denoted by *minimal acceptances* in [20].

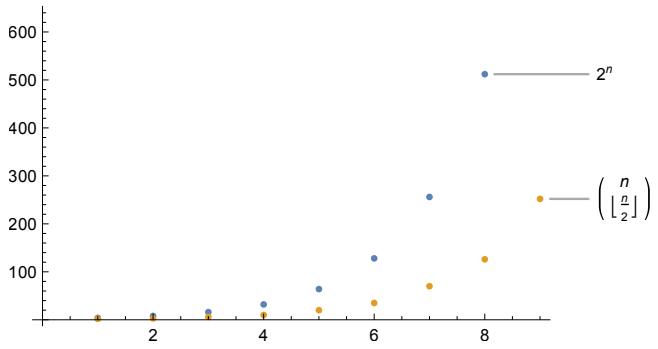

Figure 4: Function plot 2^n versus $\binom{n}{\lfloor \frac{n}{2} \rfloor}$.

Estimate for the Maximal Number of Trace Test Executions. According to Theorem 3, a complete test suite checking traces refinement just contains the adaptive test case $U_T(pq-1)$. As derived for $U_F(j)$ above, the maximal number of executions to be performed by $(Q \,|[\, \Sigma \,]|\, U_T(pq-1))$ is of order $O(|\Sigma|^{pq-1})$.

6.2. 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. So, the SUT is verified with test traces up to, and including, length pq. With branch (20), $U_F(j)$ accepts 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 necessary, or one could elaborate alternative complete test strategies where the SUT is tested with shorter traces only. Indeed, an example in [29, Exercise 5] shows that, when testing for equivalence of deterministic FSMs, it is sufficient to test with traces of significantly shorter length.

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

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

$$\begin{array}{rcl} P & = & P(0) \\ P(k) & = & (k < p-1)\&((a \to P(k)) \sqcap (b \to P(k+1))) \\ & & \Box \\ & & (k = p-1)\&(a \to P(k)) \\ Q & = & Q(0) \\ Q(k) & = & (k < q-1)\&(a \to Q(k+1)) \\ & & \Box \\ & & (k = q-1)\&(a \to Q(0) \;\Box\; b \to Q(0)) \end{array}$$

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"

$$(pass \to Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U_F(j)) \setminus \Sigma \quad \text{and} \quad (pass \to Stop) \sqsubseteq_T (Q \,|[\, \Sigma \,]|\, U_T(j)) \setminus \Sigma$$

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

Q, however, neither conforms to P in the failures refinement relation, nor in the traces-refinement relation. This can only be seen when executing the test $U_F(pq-1)$ and $U_T(pq-1)$, respectively. These tests fail, because they allow for execution of the following trace of length 12

$$s_{\text{fail}} = a.a.a.b.a.a.a.b.a.a.a.b$$

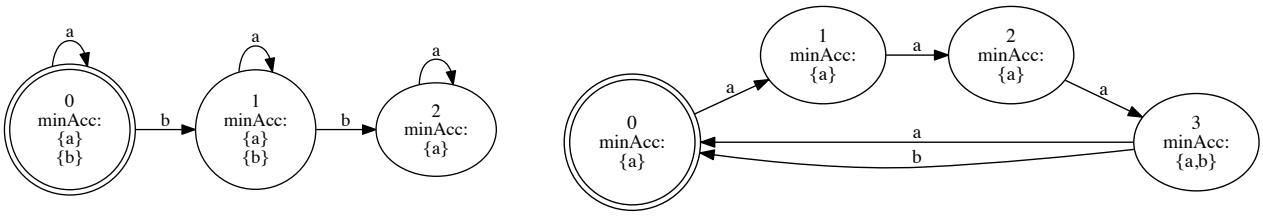

Figure 5: Transition graphs of P (left) and Q (right) from Example 5 for $p = 3$ and $q = 4$.

Only in the very last step, the final b-event can be produced by Q but is refused by P. Therefore, tests $U_F(11)$ and $U_T(11)$ enter their first branch (20) and (31), respectively, when engaging into the last b, and this produces the *fail*-event and shows that $P \not\sqsubseteq_F Q$ and $P \not\sqsubseteq_T Q$ according to Theorem 2 and Theorem 3.. The tests $U_F(j)$ and $U_T(j)$ with $j < pq - 1 = 11$ only execute true prefixes of s_{fail} which are allowed for P, so that these tests pass.

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. □

Generalising Example 5, it can be shown that for any $p, q \geq 2$, there exist reference processes P with p states and implementation processes Q with q states, such that a violation of the traces-refinement property can only be detected with a trace of length pq. This is proven in the following theorem.

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 traces 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 $traces(P) = \mathbf{pref}((a^*b)^{p-1}a^*)$, 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 $traces(Q) = \mathbf{pref}((a^{q-1}(a \mid b))^*)$. 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 traces of P contain at most $p-1$ b-events. This proves statement 3.

Let $s \in traces(Q)$ be any trace of length $\#s = pq - 1$, then $s = (a^{q-1}(a \mid b))^{p-1}a^{q-1} \in \mathbf{pref}((a^{q-1}(a \mid b))^*)$. So, s is also an element of $traces(P)$, because $(a^{q-1}(a \mid b))^{p-1}a^{q-1}$ is also contained in $\mathbf{pref}((a^*b)^{p-1}a^*)$, since $\mathbf{pref}((a^*b)^{p-1}a^*)$ contains all finite sequences of a, where at most $p-1$ events b have been inserted. This proves statement 4.

To prove statement 5, we observe that the specifications of P and Q lead to the following sets of minimal acceptances. In these definition, the expression $(s \downarrow b)$ denotes the number of b events occurring in trace s.

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

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

So, the minimal acceptance set $A_P = \{a\}$ 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, in particular (11), can be applied to conclude that Q *conf* P. □

It is discussed next in Section 7 how the number of test traces to be executed by complete test suites for failures or traces refinement can still be reduced *without* reducing the maximal length.

7. Discussion and Conclusions

Further Reductions of the Test Effort. As shown in Theorem 5, the maximal length pq of traces to be tested for either failures refinement or traces refinement cannot be further reduced. It is noteworthy, however, that when testing FSMs for equivalence, considerably shorter traces can be used. From the classical results published in [23, 24], for example, it follows that the maximal trace length to be executed is less or equal to $3p - q$, which is considerably smaller than pq for $p, q \geqslant 3$. As a consequence, the investigation of complete test suites establishing failures or trace equivalence is of considerable interest and will be discussed in a future paper.

It is also known from FSM testing that it is not necessary to check *all* traces up to length pq when testing for reduction of FSMs (which corresponds to trace refinement). Notable complete strategies supporting this fact have been presented, for example, in [3, 30, 31, 4]. From [10] 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. These will effectively reduce the upper bound for the number of test executions to be performed, which has been shown to be of the order $O(|\Sigma|^{pq-1})$ for our traces-refinement tests in Section 6. The bound $\binom{n}{\lfloor n/2 \rfloor}$ for the number of sets to be used in probing the SUT for illegal deadlocks, however, cannot be further reduced, as established in Lemma 5.

Adaptive Test Cases. The tests suggested in [17, 20] were *preset* in the sense that the trace to be executed was predefined for each test. As a consequence, the authors of [20] 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. We consider this as a disadvantage, since, when aiming at executing a specific trace s before being able to check the test objective—for example, the absence of deadlocks when offering a hitting set H of the minimal acceptances—it may take several tries until the full trace s is accepted by the SUT. Considering the complete testing assumption described in Section 3, it may even take $c^{\#s}$ tries to reach the end of the trace s, if the SUT can legally block every event of s due to nondeterminism, so that c trials are required for each event is accepted.

Those authors, later, in the context of a richer algebra based on CSP, have considered a framework similar to adaptive testing [32]. They have, however, considered only traces refinement, and have not studied complexity.

In contrast to that, our test cases specified in Section 3 and 5 are adaptive. This has the advantage that test executions $(Q \,[\![\, \Sigma \,]\!]\, U_F(j))$ for failures refinement may only stop early with *pass* after traces s satisfying $minHit(P/s) = \varnothing$, and may deadlock (a) after a trace s where the SUT illegally deadlocks (so $minHit(P/s) \neq \varnothing$ for the reference process, but $minHit(Q/s) = \varnothing$ for the implementation Q), or (b) in the final step when—just as in the corresponding test cases specified in [20]— hitting sets H are offered to the SUT and it refuses their acceptance. In both situations (a) and (b) the test fails. As a consequence, far less test repetitions are required according to the complete testing assumption than for the successful execution of all test cases specified in [20].

Another distinction of our failures test cases $U_F(j)$ to the tests specified in [20] consists in the fact that the former test both traces refinement and the *conf* conformance relation in one go, whereas the latter use separate test suites to establish these two correctness conditions. Again, we consider the structure of the test cases $U_F(j)$ as advantageous, since, when checking acceptance of a hitting set H after a trace s for c times according to the complete testing assumption, any acceptance of an illegal event $e \in \Sigma - [P/s]^0$ should also be revealed within these c tries.

Fault Domains. As already mentioned, the work in [21] 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, but is restricted to finite and nonterminating reference processes. In addition, [21] presents an algorithm for test generation that can be 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 an arbitrary behaviour specified by RUN.

Implications for CSP Model Checking. As explained in the previous sections, passing a test is characterised by the failures-refinement check $(pass \rightarrow Stop) \sqsubseteq_F (Q \,[\![\, \Sigma \,]\!]\, U_F(j)) \setminus \Sigma$ 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 of[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.

For a large implementation process 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, our testing approach based on fault domains may still be used as efficient bug finders: use the number of nodes of the normalised transition graph of the reference process as the initial value for q and increment q from there, as long as each increment reveals new errors.

Practical Application to Embedded Systems Testing. Test models developed in CSP are particularly useful for testing embedded control systems; see [33, 34], for example, for reports on practical application. Using complete test methods is particularly attractive for justifying the test case selection when verifying safety-critical control systems. The complexity estimates in Section 6, however, indicate that completeness will result in considerable test effort depending exponentially on the number of states in reference model and implementation. Moreover, the complexity considerations showed that nondeterminism increases the number of test cases further by a significant factor.

Therefore, we expect that the test strategy presented in this paper will only be applicable to rather small control systems, when used directly for HW/SW integration testing, where tests are executed on the target hardware and in real physical time. There are, however, several options to circumvent or at least mitigate the complexity problem.

(1) Recent research on the certifiability of autonomous systems advocates a multi-level approach to testing. In the first stage, very many tests are executed concurrently in a cloud configuration.[5] In the second stage, a smaller subset of these tests is then executed against the real physical embedded system. We expect that the complete testing approach described here would typically be applied for stage 1 in the cloud configuration.

(2) The number of test cases to consider in a complete test suite can be significantly reduced by the introduction of equivalence classes. It has been shown in [10] that complete testing theories for finite state machines induce likewise complete equivalence class testing strategies for more complex modelling formalisms, such as Kripke Structures or CSP. Experimental evaluations have shown that with the help of such classes, complete test suites with manageable size can be generated [35, 36]. The experiments considered on-board speed monitors for trains, airbag controllers, and route controllers for railway interlocking systems.

(3) A further promising option is to eliminate nondeterminism. For safety-critical systems, nondeterminism usually occurs in environment models or in high-level models of the target system for specifying behavioural options. The implementation itself is typically deterministic. As a consequence, the high-level model can be refined until all design decisions are included, so that the lower-level models are deterministic. Now the SUT could be tested for *equivalence* to the lower-level model. The lower-level model would typically have more states than the higher-level one, but – as

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

[5] An informal overview describing the ongoing discussion about virtualisation of tests and simulations in the cloud can be found in https://www.linkedin.com/pulse/virtual-testing-qualification-autonomous-vehicles-lopez-rodriguez.

mentioned at the beginning of this section – maximal trace length would now be only $3q - q$ instead of pq (recall that p denotes the number of states in the reference model, and q the maximal number of states in the implementation). Therefore, complete test suites checking trace or failures equivalence are of considerable interest as well.

(4) If the ultimate goal is to establish failures refinement, there is a useful heuristic for reducing the test effort while the SUT is probably still faulty (e.g. for newly developed components). We start with the longest test cases $U_F(pq - 1), U_F(pq - 2), \ldots$, and only perform the shorter tests $U_F(j)$ with small j when the longer ones do not reveal any errors. This is very effective, since longer test cases completely check trace refinement, before hitting sets are applied to check admissibility of nondeterministic behaviour at the end of the longest traces. As a consequence, trace errors will be quickly revealed before fully covering the failures-related tests.

The question whether the SUT is inside the specified fault domain can usually be answered by performing static analyses of the SUT code or by monitoring state coverage and decision coverage during test executions. If the code has been automatically generated from a low-level model, this is further simplified when the code generator's strategy for introducing auxiliary variables (if any) is known. Even in the case where it cannot be determined whether the SUT is inside the fault domain, the experiments from [35, 36] have shown that the test suites generated according to a complete strategy still have superior test strength, when compared to naive heuristic testing approaches.

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 traces 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 in our test cases is optimal, so that it is not possible to guarantee exhaustiveness with fewer probes. Moreover, the maximal length of the test traces cannot be reduced without losing the test suite's completeness property; this holds both for traces and failures refinement.

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 also 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 for several helpful suggestions. The work of Ana Cavalcanti is funded by the Royal Academy of Engineering and UK EPSRC Grant EP/R025134/1.

[1] J. Peleska, Model-based avionic systems testing for the airbus family, in: 23rd IEEE European Test Symposium, ETS 2018, Bremen, Germany, May 28 - June 1, 2018, IEEE, 2018, pp. 1–10. doi:10.1109/ETS.2018.8400703.
URL https://doi.org/10.1109/ETS.2018.8400703

[2] J. Peleska, J. Brauer, W. Huang, Model-based testing for avionic systems proven benefits and further challenges, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV, Vol. 11247 of Lecture Notes in Computer Science, Springer, 2018, pp. 82–103. doi:10.1007/978-3-030-03427-6_11.
URL https://doi.org/10.1007/978-3-030-03427-6_11

[3] R. M. Hierons, Testing from a nondeterministic finite state machine using adaptive state counting, IEEE Trans. Computers 53 (10) (2004) 1330–1342. doi:10.1109/TC.2004.85.
URL http://doi.ieeecomputersociety.org/10.1109/TC.2004.85

[4] A. Simão, A. Petrenko, N. Yevtushenko, On reducing test length for FSMs with extra states, Software Testing, Verification and Reliability 22 (6) (2012) 435–454. doi:10.1002/stvr.452.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.452

[5] J. Springintveld, F. Vaandrager, P. D'Argenio, Testing timed automata, Theoretical Computer Science 254 (1-2) (2001) 225–257.

[6] A. Cavalcanti, M.-C. Gaudel, Testing for refinement in *circus*, Acta Inf. 48 (2) (2011) 97–147.

[7] S. Schneider, An operational semantics for timed csp, Inf. Comput. 116 (2) (1995) 193–213. doi:10.1006/inco.1995.1014.
URL http://dx.doi.org/10.1006/inco.1995.1014

[8] J. Tretmans, Conformance testing with labelled transition systems: Implementation relations and test generation, Computer Networks and ISDN Systems 29 (1) (1996) 49–79.

[9] A. Petrenko, Checking experiments for symbolic input/output finite state machines, in: Ninth IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2016, Chicago, IL, USA, April 11-15, 2016, IEEE Computer Society, 2016, pp. 229–237. doi:10.1109/ICSTW.2016.9.
URL http://dx.doi.org/10.1109/ICSTW.2016.9

[10] W.-l. Huang, J. Peleska, Complete model-based equivalence class testing for nondeterministic systems, Formal Aspects of Computing 29 (2) (2017) 335–364. doi:10.1007/s00165-016-0402-2.
URL http://dx.doi.org/10.1007/s00165-016-0402-2

[11] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[12] A. W. Roscoe, Understanding Concurrent Systems, Springer, London, Dordrecht Heidelberg, New York, 2010.

[13] A. Hall, R. Chapman, Correctness by construction: developing a commercial secure system, IEEE Software 19 (1) (2002) 18–25. doi:10.1109/52.976937.

[14] H. Shi, J. Peleska, M. Kouvaras, 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, IEEE Computer Society, 1999, pp. 135–142. doi:10.1109/PRDC.1999.816222.
URL https://doi.org/10.1109/PRDC.1999.816222

[15] B. Buth, M. Kouvaras, J. Peleska, H. Shi, Deadlock analysis for a fault-tolerant system, in: M. Johnson (Ed.), Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings, Vol. 1349 of Lecture Notes in Computer Science, Springer, 1997, pp. 60–74. doi:10.1007/BFb0000463.
URL https://doi.org/10.1007/BFb0000463

[16] A. W. Roscoe, Model-checking csp, in: A. W. Roscoe (Ed.), A Classical Mind: Essays in Honour of C. A. R. Hoare, Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994, Ch. 21, pp. 353–378.

[17] M. Hennessy, Algebraic Theory of Processes, MIT Press, Cambridge, MA, USA, 1988.

[18] J. Peleska, M. Siegel, From testing theory to test driver implementation, in: M. Gaudel, J. Woodcock (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, Vol. 1051 of Lecture Notes in Computer Science, Springer, 1996, pp. 538–556. doi:10.1007/3-540-60973-3_106.
URL https://doi.org/10.1007/3-540-60973-3_106

[19] J. Peleska, M. Siegel, Test automation of safety-critical reactive systems, South African Computer Jounal 19 (1997) 53–77.

[20] A. Cavalcanti, M. Gaudel, Testing for refinement in CSP, in: M. J. Butler, M. G. Hinchey, M. M. Larrondo-Petrie (Eds.), Formal Methods and Software Engineering, 9th International Conference on Formal Engineering Methods, ICFEM 2007, Boca Raton, FL, USA, November 14-15, 2007, Proceedings, Vol. 4789 of Lecture Notes in Computer Science, Springer, 2007, pp. 151–170. doi:10.1007/978-3-540-76650-6_10.
URL https://doi.org/10.1007/978-3-540-76650-6_10

[21] A. Cavalcanti, A. da Silva Simão, Fault-based testing for refinement in CSP, in: N. Yevtushenko, A. R. Cavalli, H. Yenigün (Eds.), Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings, Vol. 10533 of Lecture Notes in Computer Science, Springer, 2017, pp. 21–37. doi:10.1007/978-3-319-67549-7_2.
URL https://doi.org/10.1007/978-3-319-67549-7_2

[22] L. Shi, X. Cai, An exact fast algorithm for minimum hitting set, in: 2010 Third International Joint Conference on Computational Science and Optimization, Vol. 1, 2010, pp. 64–67. doi:10.1109/CSO.2010.240.

[23] T. S. Chow, Testing software design modeled by finite-state machines, IEEE Transactions on Software Engineering SE-4 (3) (1978) 178–186.

[24] M. P. Vasilevskii, Failure diagnosis of automata, Kibernetika (Transl.) 4 (1973) 98–108.

[25] G. Luo, G. von Bochmann, A. Petrenko, Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method, IEEE Trans. Software Eng. 20 (2) (1994) 149–162. doi:10.1109/32.265636.
URL http://doi.ieeecomputersociety.org/10.1109/32.265636

[26] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, A. Roscoe, FDR3 — A Modern Refinement Checker for CSP, in: E. Ábrahám, K. Havelund (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Vol. 8413 of Lecture Notes in Computer Science, 2014, pp. 187–201.

[27] A. W. Roscoe, The Theory and Practice of Concurrency, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[28] E. Sperner, Ein Satz über Untermengen einer endlichen Menge, Mathematische Zeitschrift 27 (1) (1928) 544–548. doi:10.1007/BF01171114.
URL https://doi.org/10.1007/BF01171114

[29] J. Peleska, W.-l. Huang, Test Automation - Foundations and Applications of Model-based Testing, University of Bremen, 2017, lecture notes, available under http://www.informatik.uni-bremen.de/agbs/jp/papers/test-automation-huang-peleska.pdf.

[30] R. Dorofeeva, K. El-Fakih, N. Yevtushenko, An improved conformance testing method, in: F. Wang (Ed.), Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5, 2005, Proceedings, Vol. 3731 of Lecture Notes in Computer Science, Springer, 2005, pp. 204–218. doi:10.1007/11562436_16.
URL https://doi.org/10.1007/11562436_16

[31] A. Petrenko, N. Yevtushenko, Adaptive testing of deterministic implementations specified by nondeterministic fsms, in: Testing Software and Systems, no. 7019 in Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, pp. 162–178.

[32] A. L. C. Cavalcanti, M.-C. Gaudel, Test selection for traces refinement, Theoretical Computer Science 563 (0) (2015) 1 – 42. doi:10.1016/j.tcs.2014.08.012.
URL https://www-users.cs.york.ac.uk/~alcc/publications/papers/CG15.pdf

[33] J. Peleska, Hardware/software integration testing for the new airbus aircraft families, in: I. Schieferdecker, H. König, A. Wolisz (Eds.), Testing of Communicating Systems XIV. Application to Internet Technologies and Services, Kluwer Academic Publishers, 2002, pp. 225–351.

[34] J. Peleska, Formal methods for test automation - hard real-time testing of controllers for the airbus aircraft family, in: Proc. of the Sixth Biennial World Conference on Integrated Design & Process Technology (IDPT2002), Pasadena, California, June 23-28, 2002, Society for Design and Process Science, 2002, iSSN 1090-9389.

[35] J. Peleska, W. Huang, F. Hübner, A novel approach to HW/SW integration testing of route-based interlocking system controllers, in: T. Lecomte, R. Pinger, A. Romanovsky (Eds.), Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification,

and Certification - First International Conference, RSSRail 2016, Paris, France, June 28-30, 2016, Proceedings, Vol. 9707 of Lecture Notes in Computer Science, Springer, 2016, pp. 32–49. doi:10.1007/978-3-319-33951-1_3.
URL http://dx.doi.org/10.1007/978-3-319-33951-1_3

[36] F. Hübner, W.-l. Huang, J. Peleska, Experimental evaluation of a novel equivalence class partition testing strategy, Software & Systems ModelingDoi=10.1007/s10270-017-0595-8. doi:10.1007/s10270-017-0595-8.
URL https://doi.org/10.1007/s10270-017-0595-8