Finite Complete Suites for CSP Refinement Testing

Ana Cavalcanti[1], Wen-ling Huang[2], Jan Peleska[2], and Adenilso Simao[3]

[1] University of York, United Kingdom
ana.cavalcanti@york.ac.uk
[2] University of Bremen, Germany
{peleska,huang}@uni-bremen.de
[3] University of São Paulo, Brazil
adenilso@icmc.usp.br

Abstract. In this paper, two new contributions for model-based testing with Communicating Sequential Processes (CSP) are presented. For a given CSP process representing the reference model, test suites checking the conformance relations trace and failures refinement are constructed, and their finiteness and completeness (that is, capability to uncover conformity violations) is proven. While complete test suites for CSP processes have been previously investigated by several authors, a sufficient condition for their finiteness is presented here for the first time. Moreover, an optimal technique for testing the admissibility of an implementation's nondeterministic behaviour is described.

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

1 Introduction

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

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

Generation techniques for complete test suites have been developed for various formalisms. Here, we tackle *Communicating Sequential Processes (CSP)* [9, 18]. 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 [18] and in the references there. Industrial success has also been reported; see, for example, [6, 20, 1].

Contributions This paper presents complete black-box test suites for software and systems modelled using CSP. They can be generated for divergence-free[4] finite-state CSP processes with finite alphabets, interpreted both in the trace and the failures semantics. Our results complement work in [3]. There, fault domains are specified as collections of processes refining a "most general" fault domain member. With that concept, complete test suites may be finite or infinite. This gives important insight into the theory of fault-domain testing for CSP, but we are particularly interested in *finite* suites when it comes to practical application. While [3] may require additional criteria to select tests from still infinite test suites, here, we further restrict fault domains using a graph representation of processes (originally elaborated for model checking) to obtain finite test suites.

FiXme Fatal: alcc: I think termination is not handled either?

Our approach to the definition of CSP fault domains is presented in this paper. We take advantage of results on model checking of CSP processes, where the failures semantics of a finite-state CSP process is represented as a finite normalised transition graph, whose edges are labelled by the events the process engages in, and whose nodes are labelled by minimal acceptances or, alternatively, maximal refusals [16]. The maximal refusals express the degree of nondeterminism present in a process state that is in one-one-correspondence to a node of the normalised transition graph. Inspired by the way fault-domains are specified for finite state machines (FSMs), we define them here as the set of CSP processes whose normalised transition graphs do not exceed the order (that is, the number of nodes) of the reference model's graph by more than a given constant.

The main contribution of this paper is the construction of two test suites to verify the conformance relations *trace refinement* and *failures refinement* for any given non-divergent, finite-state, finite alphabet CSP process. We prove their completeness with respect to fault domains of the described type. The existence of – possibly infinite – complete test suites has been established for process algebras, and for CSP in particular, by several authors [7, 19, 13, 11, 2, 3]. To the best of our knowledge, however, this article is the first to present *finite*, complete test suites associated with this class of fault domains and conformance relations.

Moreover, our result is not a simple transcription of existing knowledge about finite, complete test suites for finite state machines: the capabilities of CSP to express nondeterminism in a more fine-grained way than it is possible for FSMs requires a more complex approach to testing systems for conformity in the failures-model than it is required for model-based testing against nondeterministic FSMs, as published, for example, in [8, 15]. We show that the approach to testing the admissibility of an SUT's nondeterministic behaviour is optimal in the sense that it cannot be established with fewer test cases.

FiXme Fatal: alcc: I did not understand this?

Overview In Section 2, we present the background material relevant to our work. In Section 3, finite complete test suites for verifying failures refinement are presented. Test suites checking trace refinement are a simplified version of the former class; they are presented in Section 4. A sample test suite is presented

[4] The assumption of divergence freedom is usually applied in black-box testing, since it cannot distinguish between divergence and deadlock.

in Section 5. Our results are discussed in Section 6, where we also conclude. References to related work are given throughout the paper where appropriate.

2 Preliminaries

In this section, we present CSP and the concept of minimal hitting sets, which is central to our novel notion of test for failures refinement.

2.1 CSP, Refinement, and Normalised Transition Graphs

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

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

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

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

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

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

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. $\Box$

Iterated forms $\Box\, i : I \bullet P(i)$ and $\bigsqcap i : I \bullet P(i)$ of the external and internal choice operators allow us to define a choice over a collection of processes $P(i)$. By convention, if the index set I is empty, external choices evaluate to the process *Stop*, which deadlocks: does not engage into any event and does not terminate. An iterated internal choice is not defined for an empty index set. For an external choice indexed over a set of events in the form $\Box\, e : A \bullet e \rightarrow P(e), A \subseteq \Sigma$, we use abbreviation $e : A \rightarrow P(e)$.

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

Interactions that are not supposed to be visible to the environment can be hidden. The operator $P \setminus H$ defines a process that behaves as P, with the interactions modelled by events in the set H hidden. Frequently, hiding is used in conjunction with parallelism: it is often desirable to make internal 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 [18].

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

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

Semantic definitions specify, for each operator, how the traces or failures of the resulting process can be calculated from the traces or failures of each operand. For example, for internal choice, $failures(P \sqcap Q) = failures(P) \cup failures(Q)$; see [17, p. 210] for a comprehensive list of these definitions.

Using the notation P/s to denote the process P after having engaged into trace s, the set $Ref(P/s) \mathrel{\widehat=} \{ X \mid (s, X) \in failures(P) \}$ contains the refusals of P after s. Refusals are subset-closed [9, 18]: if (s, X) is a failure of P and $Y \subseteq X$, then $(s, Y) \in failures(P)$ and $Y \in Ref(P/s)$ follows.

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

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

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

$$(P \sqsubseteq_F Q) \Leftrightarrow (P \sqsubseteq_T Q \wedge Q \ conf \ P) \qquad (2)$$

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

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

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

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

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

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

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

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

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

Normalised Transition Graphs for CSP Processes As shown in [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 $e_1 \dots e_k$ and connecting states $\underline{n}, n_1, \dots, n_{k-1}, n$, such that

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

By construction, $s \in \Sigma^*$ is a trace of P, if, and only if, there is a path through $G(P)$ starting at $\underline{n}$ whose edge labels coincide with s. In analogy to $\mathit{traces}(P)$, 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

$traces(P) = traces(G(P))$. Since $G(P)$ is normalised, there is a unique node reached by applying 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.

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

The graph construction from [16] used to define $G(P)$ for any process P guarantees that $[G(P)/s]^0 = [P/s]^0$ for all traces s of P.

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

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

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

Acceptances When investigating tests for failures refinement, the notion of *acceptances* [7], which is dual to refusals, is also useful. An acceptance set of P/s is a subset of its initials $[P/s]^0$; equivalently, it is a subset of events labelling outgoing transitions of $G(P)/s$. If the behaviour of P/s is deterministic, its only acceptance equals $[P/s]^0$, because P/s never refuses any of the events contained in this set. If P/s is nondeterministic, it internally chooses one of its *minimal acceptance sets* A and never refuses any event in A, while *possibly* refusing the events from $[P/s]^0 - A$ and *always* refusing those from $\Sigma - [P/s]^0$. The acceptances of P/s are denoted by $Acc(P/s)$, and the minimal acceptances by $minAcc(P/s)$. They satisfy the following properties.

$$A \in minAcc(P/s) \Leftrightarrow \exists R \in maxRef(P/s) \land A = \Sigma - R \tag{6}$$

$$\bigcup\{A \mid A \in Acc(P/s)\} = [P/s]^0 \tag{7}$$

$$X \in Acc(P/s) \Leftrightarrow \exists A \in minAcc(P/s) : A \subseteq X \subseteq [P/s]^0 \tag{8}$$

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

$$minAcc(G(P)/s) = \{\Sigma - R \mid R \in r(G(P)/s)\}. \tag{9}$$

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_2 representing the process state $P/a = Q \sqcap R$.

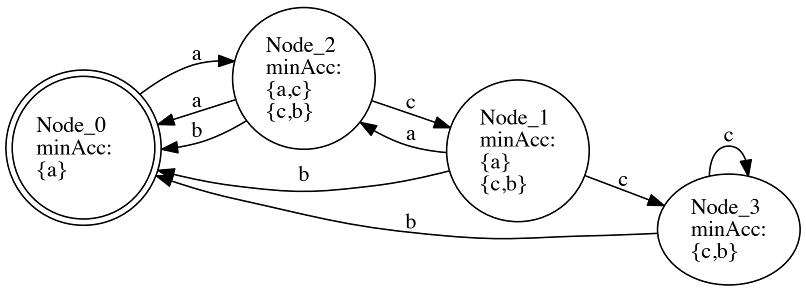

Fig. 1. Normalised transition graph of CSP process P from Example 3.

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_1. Due to normalisation, there is only a single transition satisfying $t(\text{Node\_2}, c) = \text{Node\_1}$. This transition, however, can have been caused by either Q or R engaging into c, so Node_1 corresponds to process state $Q/c \sqcap R/c = P \sqcap R$. This is reflected by the two minimal acceptance sets labelling Node_1. Similar considerations explain the other nodes and transitions in Fig. 1.

Note that the node names including their number suffixes are generated by the FDR tool. The numbering is generated during the normalisation procedure. So, the node numbers do not reflect the distance from the initial node Node_0. □

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

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

as established by the results in the following lemma. There, result (10) reflects trace refinement in terms of graph traces; (11) expresses failures refinement in terms of traces refinement and *conf*; (12) states how *conf* can be expressed by means of the maximal refusal functions of the graphs involved; and (13) states the same in terms of the minimal acceptances that can be derived from the maximal refusal functions by means of (9).

Lemma 1.

$$P \sqsubseteq_T Q \Leftrightarrow \mathit{traces}(G(Q)) \subseteq \mathit{traces}(G(P)) \tag{10}$$

$$P \sqsubseteq_F Q \Leftrightarrow P \sqsubseteq_T Q \wedge Q \ \mathit{conf} \ P \tag{11}$$

$$\begin{aligned} Q \ \mathit{conf} \ P \Leftrightarrow & \forall s \in \mathit{traces}(G(Q)) \cap \mathit{traces}(G(P)), R_Q \in r_Q(G(Q)/s) : \\ & \exists R_P \in r_P(G(P)/s) : R_Q \subseteq R_P \qquad (12) \\ \Leftrightarrow & \forall s \in \mathit{traces}(G(Q)) \cap \mathit{traces}(G(P)), A_Q \in \mathit{minAcc}(G(Q)/s) : \\ & \exists A_P \in \mathit{minAcc}(G(P)/s) : A_P \subseteq A_Q \qquad (13) \end{aligned}$$

□

Proof. Recall that $P \sqsubseteq_T Q$ is defined as $\mathit{traces}(Q) \subseteq \mathit{traces}(P)$ and that $\mathit{traces}(P) = \mathit{traces}(G(P))$ and $\mathit{traces}(Q) = \mathit{traces}(G(Q))$. This shows (10).

To prove (11), we derive

$$\begin{aligned} & P \sqsubseteq_F Q \\ \Leftrightarrow & \mathit{failures}(Q) \subseteq \mathit{failures}(Q) && [\text{definition of } \sqsubseteq_F] \\ \Leftrightarrow & \forall (s,R) \in \mathit{failures}(Q) : (s,R) \in \mathit{failures}(P) && [\text{property of } \subseteq] \\ \Leftrightarrow & \forall (s,\varnothing) \in \mathit{failures}(Q) : (s,\varnothing) \in \mathit{failures}(P) \wedge \\ & \quad \forall s \in \mathit{traces}(Q), R \in \mathit{Ref}(Q/s) : R \in \mathit{Ref}(P/s) \\ & && [\text{Definition of failures, refusals, subset closure}] \\ \Leftrightarrow & \mathit{traces}(Q) \subseteq \mathit{traces}(P) \wedge \\ & \quad \forall s \in \mathit{traces}(Q), R \in \mathit{Ref}(Q/s) : R \in \mathit{Ref}(P/s) \\ & && [\text{Definition of failures and traces}] \\ \Leftrightarrow & \mathit{traces}(Q) \subseteq \mathit{traces}(P) \wedge \\ & \quad \forall s \in \mathit{traces}(Q) \cap \mathit{traces}(P) : \mathit{Ref}(Q/s) \subseteq \mathit{Ref}(P/s) \\ & && [\text{Property of } \cap \text{ and } \subseteq] \\ \Leftrightarrow & P \sqsubseteq_T Q \wedge Q \ \mathit{conf} \ P && [\text{Definition of } \sqsubseteq_T \text{ and } \mathit{conf}] \end{aligned}$$

To prove (12), we derive

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

Finally, (13) follows from (12) using (6) and the fact that $R_Q \subseteq R_P$ is equivalent to $\Sigma - R_P \subseteq \Sigma - R_Q$. □

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

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

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

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

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

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

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

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

over the same alphabet Σ. Their product is defined by

$$G_1 \times G_2 = (N_1 \times N_2, (\underline{n}_1, \underline{n}_2), t : (N_1 \times N_2) \times \Sigma \nrightarrow (N_1 \times N_2)) \quad (14)$$

$$\text{dom } t = \{((n_1, n_2), e) \in (N_1 \times N_2) \times \Sigma \mid (n_1, e) \in \text{dom } t_1 \wedge (n_2, e) \in \text{dom } t_2\} \quad (15)$$

$$t((n_1, n_2), e) = (t_1(n_1, e), t_2(n_2, e)) \text{ for } ((n_1, n_2), e) \in \text{dom } t \quad (16)$$

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 $V' = V \cup V.\varSigma \cup \cdots \cup V.\varSigma^{(pq-1)}$. The maximal length of traces in V' is $(pq-1)$. □

2.2 Minimal Hitting Sets

The main idea of the underlying test strategy for failures refinement can be 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 $\varSigma$, a *hitting set* $H \subseteq \varSigma$ is a set satisfying the following property.

$$\forall\, A \in C : H \cap A \neq \varnothing. \tag{17}$$

A *minimal hitting set* is a hitting set that cannot be further reduced without losing the characteristic property (17). By $\mathit{minHit}(C)$ we denote the collection of minimal hitting sets for a collection C. For the pathological case where C contains an empty set, $\mathit{minHit}(C)$ is also empty.

The problem of determining minimal hitting sets is NP-hard [21]. We see below, however, that it reduces the effort of testing for failures refinement from a factor of $2^{|\varSigma|}$ to a factor that equals the number of minimal hitting sets.

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

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

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

Proof. For showing "1 ⇒ 2", assume P *conf* Q and $s \in \mathit{traces}(P) \cap \mathit{traces}(Q)$. Lemma 1 (13), states that

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

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

To prove "2 ⇒ 1", assume that 2 holds, but that P *conf* Q does *not* hold. According to Lemma 1, (13), there exists $s \in \mathit{traces}(P) \cap \mathit{traces}(Q)$ such that

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

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

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

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

We note that $\mathit{minAcc}(P) = \{\varnothing\}$ if $P = Q \sqcap \mathit{Stop}$. Since *Stop* accepts nothing, its minimal acceptance is $\varnothing$, and this carries over to $Q \sqcap \mathit{Stop}$. From (13) we conclude that $\varnothing \in \mathit{minAcc}(P)$ implies $\mathit{minAcc}(P) = \{\varnothing\}$. This clarifies that $\mathit{minHit}(P/s)$ is empty if, and only if, $\mathit{minAcc}(P) = \{\varnothing\}$. The proof of Lemma 4 covers the situations where $\mathit{minAcc}(P/s) = \{\varnothing\}$ and so $\mathit{minHit}(P/s) = \varnothing$. Trivially,

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

holds.

Since minimal acceptances can be used to label the nodes of a normalised transition graph, and since $\mathit{minAcc}(P/s) = \mathit{minAcc}(G(P)/s)$, the notation of minimal hitting sets also carries over to graphs: we write $\mathit{minHit}(n)$ for nodes n of $G(P)$ and observe that

$$\mathit{minHit}(G(P)/s) = \mathit{minHit}(P/s) \text{ for all } s \in \mathit{traces}(P). \tag{19}$$

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 process [7]. They synchronise with the SUT over its visible events and use some additional events outside the SUT'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 for each integer $p \geqslant 0$, we define a CSP test process for failures refinement as shown below.

$$
\begin{aligned}
U_F(p) &= U_F(p, 0, \underline{n}) && (20)\\
U_F(p, k, n) &= \big(e : (\Sigma - [n]^0) \to \mathit{fail} \to \mathit{Stop}\big) && (21)\\
&\quad \Box\\
&\quad (\mathit{minHit}(n) = \varnothing) \& \big(\mathit{pass} \to \mathit{Stop}\big) && (22)\\
&\quad \Box\\
&\quad (k < p) \& \big(e : [n]^0 \to U_F(p, k+1, t(n, e)\big) && (23)\\
&\quad \Box\\
&\quad (k = p \wedge \mathit{minHit}(n) \neq \varnothing) \&\\
&\qquad \big(\textstyle\bigsqcap_{H \in \mathit{minHit}(n)} (e : H \to \mathit{pass} \to \mathit{Stop})\big) && (24)
\end{aligned}
$$

Before explaining the intuition behind this test definition in the paragraphs below, we state its basic properties in the following lemma, to show the relationships between $U_F(p)$ and the reference process P from which the former has been derived.

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

$$
\begin{aligned}
U_F(p)/s &= U_F(p, \#s, G(P)/s) && (25)\\
e \notin [P/s]^0 &\Rightarrow U_F(p)/s.e = (\mathit{fail} \to \mathit{Stop}) && (26)\\
U_F(p)/s = U(p, \#s, n) &\Rightarrow [n]^0 = [P/s]^0 && (27)\\
U_F(p)/s = U(p, \#s, n) &\Rightarrow \mathit{minHit}(n) = \mathit{minHit}(P/s) && (28)
\end{aligned}
$$

Proof. We prove (25) by induction over the length of s. For $\#s = 0$, the statement holds because $U_F(p)$ starts with the initial node $\underline{n}$ of $G(P)$. Suppose that the statement holds for all traces s with length $\#s \leqslant k < p$, so that $U_F(p)/s = U_F(p, \#s, G(P)/s)$. Now let $s.e$ be a trace of P. Then $e \in [P/s]^0$. Since $[G(P)/s]^0 = [P/s]^0$ for all traces s of P, we conclude that $e \in [G(P)/s]^0$, so $U_F(p)/s = U_F(p, \#s, G(P)/s)$ can engage into e by executing branch (23). Since t is the transition function of $G(P)$ and $e \in [G(P)/s]^0$, $t(G(P)/s, e)$ is defined, and $t(G(P)/s, e) = G(P)/s.e$ holds. This leads to a new recursion $U_F(p)/s.e = U_F(p, \#s, G(P)/s)/e = U_F(p, \#s + 1, G(P)/s.e)$ and proves the induction step.

To prove (26), we apply (25) to conclude that $U_F(p)/s = U_F(p, \#s, G(P)/s)$, 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 $U_F(p, \#s, G(P)/s)$ can engage in e by entering branch (21). The specification of this branch implies that

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

Statement (27) follows trivially from (25), because $[G(P)/s]^0 = [P/s]^0$ for all traces s of P.

Finally, statement (28) follows trivially from (25), because, according to (19), $\mathit{minHit}(G(P)/s) = \mathit{minHit}(P/s)$ for all traces of P. □

Note that it is not guaranteed for $U_F(p)$ to run through the traces $s, s.e$ specified in Lemma 5, if $\mathit{minHit}(P/u) = \varnothing$ for some prefix u of s: in such a case, $U_F(p)$ may stop with a *pass*-event by entering branch (22). Therefore, Lemma 5 just states the existence of $U_F(p)$-executions $s, s.e$ satisfying the properties stated there.

Explanation of the Test Definition A test is performed by running $U_F(p)$ concurrently with any SUT process Q, synchronising over Σ. So, a test execution is a trace of the concurrent process

$$Q \,|[\,\Sigma\,]|\, U_F(p).$$

It is assumed that the events *fail* and *pass*, indicating a verdict FAIL and PASS for the test execution, are not included in Σ. Since we assume that Q is free of livelocks, it is guaranteed that events *fail* or *pass* always become visible, if they are the only events $U_F(p)/s$ is ready to engage in.

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

$$Q\ \underline{\text{pass}}\ U_F(p) \mathrel{\widehat{=}} (\mathit{pass} \to \mathit{Stop}) \sqsubseteq_F (Q \,|[\,\Sigma\,]|\, U_F(p)) \setminus \Sigma \tag{29}$$

This type of pass relation is often called *must test*, because every test execution must end with the *pass* event [7]. Note that it is necessary to use the failures-refinement relation in this condition, and not the trace-refinement relation: $(Q \,|[\,\Sigma\,]|\, U_F(p)) \setminus \Sigma$ 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 $U_F(p, k, n)$ executing branch (24), because $(k = p \wedge \mathit{minHit}(P/s) \neq \varnothing)$. This is explained further in the next paragraphs.

The specification of $U_F(p, k, n)$ implies that the test always stops after having engaged into a trace $s \in \mathit{traces}(Q)$ of maximal length p or $p + 1$: If branch (21) is the last to be entered, the maximal length of s is $p + 1$, and the test execution stops with *fail*. If branch (22) is the last to be entered, the maximal length of s is p, and the execution stops with *pass*. If branch (24) is the last to be entered, the process either accepts another event e of some minimal hitting set $H \in \mathit{minHit}(P/\mathit{front}(s))$. Then the final length of s is $p + 1$, and the execution terminates with *pass*. Or the test execution $(Q \,|[\,\Sigma\,]|\, U_F(p))/s$ deadlocks, the final length of s is p, and the execution stops without a PASS or FAIL event. Such an execution is also interpreted as FAIL, because it reveals that $(\mathit{pass} \to \mathit{Stop}) \not\sqsubseteq_F (Q \,|[\,\Sigma\,]|\, U_F(p)) \setminus \Sigma$.

We observe that the number of possible executions of $Q \,|[\,\Sigma\,]|\, U_F(p)$ is finite, because the number of traces s with maximal length $(p + 1)$ is finite and the sets $[n]^0$, $(\Sigma - [n]^0)$, and $\mathit{minHit}(n)$ are finite. We further recall that $\mathit{minHit}(n)$ may be empty, so that the indexed internal choice construct in (24) is undefined. Therefore, the associated guard condition contains the requirement $\mathit{minHit}(P/s) \neq \varnothing$, otherwise this branch cannot be taken, but branches (21) or (22) can be taken in this situation.

Intuitively speaking, $U_F(p)$ is able to perform any trace s of P, up to a length p. If, after having already run through $s \in \mathit{traces}(P)$ with $\#s \leqslant p$, an event is accepted by the SUT that is outside the initials of P/s (recall from Lemma 5 that $[n]^0 = [P/s]^0$ for $U_F(p)/s$), the test immediately terminates with FAIL-event *fail*. This is handled by the branch (21) of the external choice in the process $U_F(p, s)$ defined above.

If P/s is the *Stop* process or has *Stop* as an internal choice, this is revealed by $\mathit{minHit}(n) = \varnothing$ (recall (18) and Lemma 5). In this case, the test may terminate successfully (branch (22) of the external choice in $U_F(p, s)$). If P/s may also nondeterministically engage into events in such a situation, branches (21) and (23) of the test are simultaneously enabled.

If the length of s is still less than p, the test accepts any event e from the initials $[P/s]^0 = [G(P)/s]^0$ and continues recursively as $U_F(p, \#s{+}1, G(P)/s.e)$ in branch (23), this follows again from Lemma 5). A test of this type is called *adaptive*, because it accepts any legal behaviour of the SUT and adapts its consecutive behaviour to the event selected by the SUT.

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

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 of 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.

In the sequel, we establish the completeness of our test suite which is stated in the following main theorem.

Theorem 1. *Let P be a 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*

$$TS_F = \{U_F(k) \mid 0 \leqslant k < pq\}$$

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

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

Lemma 6. *Test suite TS_F generated from a CSP process P, as specified in Theorem 1, is passed by every CSP process Q satisfying $P \sqsubseteq_F Q$.*

Proof. **Step 1.** Suppose that $P \sqsubseteq_F Q$, so $P \sqsubseteq_T Q$ and P *conf* Q according to (11). Since $\mathit{traces}(Q) \subseteq \mathit{traces}(P)$, any adaptive test $U_F(p)$ running in parallel with Q will always enter the branches (22), (23), or (24) of the external choice construction for $U_F(p, s)$. Branch (21) can never be entered in the parallel execution of Q and $U_F(p)$, because $[Q/s]^0 \subseteq [P/s]^0$ for all traces of Q.

Step 2. Also, Lemma 4 implies that for all traces $s \in \mathit{traces}(Q) \cap \mathit{traces}(P)$, every H in $\mathit{minHit}(P/s)$ is a hitting set for $\mathit{minAcc}(Q/s)$. Branch (22) of test $U_F(p, s)$ leads always to a PASS verdict, and branch (23) to test continuation without a verdict. For the last branch, we note that any selected minimal hitting set $H \in \mathit{minHit}(P/s)$ has a non-empty intersection with each of the minimal acceptances of Q/s. As a consequence, Q/s never blocks when offered events from H, and the test terminates with PASS event *pass*. Note that this argument requires that Q is free of livelocks, because otherwise the *pass*-events might not become visible, due to unbounded sequences of hidden events performed by Q. □

Lemma 7. *Test suite TS_F specified in Theorem 1 is exhaustive for the fault model specified there.*

Proof. Consider a process $Q \in \mathcal{D}$ with $P \not\sqsubseteq_F Q$, According to (11), this non-conformance can be caused in two possible ways corresponding to the cases $P \not\sqsubseteq_T Q$ and $\neg(P$ *conf* $Q)$, respectively:

Case 1 $\mathit{traces}(Q) \not\subseteq \mathit{traces}(P)$

Case 2 There exists a joint trace $s \in \mathit{traces}(Q) \cap \mathit{traces}(P)$ and a minimal acceptance A_Q of $\mathit{minAcc}(Q/s)$, such that (see Lemma 1, (13)).

$$\forall\, A_P \in \mathit{minAcc}(P/s) : A_P \not\subseteq A_Q, \tag{30}$$

It has to be shown for each of the two possibilities that at least one test execution of some $(Q \,|[\,\Sigma\,]|\, U_F(k))$ with $k < pq$ ends with the *fail*-event or without giving any verdict. The latter case is also interpreted as FAIL, since then the process $\mathit{pass} \to \mathit{Stop}$ is no longer failures-refined by the test execution.

For Case 1, consider a trace $s.e \in \mathit{traces}(Q)$ such that $s \in \mathit{traces}(P)$, but $s.e \notin \mathit{traces}(P)$. 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. Suppose that $G/s = (n_P, n_Q)$. 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, (n_P, n_Q) can be reached by a trace $u \in \mathit{traces}(G)$ of length $\#u < pq$. Now the construction of the transition function of G implies that u is also a trace of P and Q. 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, $U_F(pq-1)/u = U_F(pq-1, u)$. Since $s.e \notin \mathit{traces}(P)$, e is an element of $\Sigma - [P/u]^0$. So, in at least one execution, $U_F(pq-1, u)$ executes its first branch (21) with this event e, so that the test fails. Again, the assumption of non-divergence of Q is needed for this conclusion.

For Case 2, we note that trace s is again a trace of the product graph G, but we do not know its length. Again, by applying Lemma 3, we know that the state G/s can be reached by a trace $u \in \mathit{traces}(Q) \cap \mathit{traces}(P)$ of maximal length $\#u < pq$. Consider test $U_F(\#u)$, which satisfies $U_F(\#u)/u = U_F(\#u, u)$, because it always performs branch (23) until the trace u has been completely processed. $U_F(\#u, u)$ may execute branches (21) or (24) only: assumption (30) in Case 2 implies that P/s has at least one non-empty minimal acceptance, so the guard condition $(\mathit{minAcc}(P/s) = \{\varnothing\})$ of branch (22) evaluates to `false` for $U_F(\#u, u)$. Moreover, the guard condition $(\#s < p)$ for branch (23) evaluates to `false` for $U_F(\#u, u)$, too. If branch (21) is executed, the test always fails. If branch (24) is executed, the test fails for the execution where a minimal hitting set $H \in \mathit{minHit}(P/u)$ is chosen by $U_F(\#u, u)$ that has an empty intersection with the minimal acceptance A_Q from condition (30). The existence of such an H is guaranteed because of Lemma 4. As a consequence, there exists a test execution where Q/u selects acceptance A_Q and $U_F(\#u, u)$ selects H. This execution deadlocks in process state $(Q \,|[\, \Sigma \,]|\, U_F(\#u))/u$, so it cannot produce the *pass*-event; this means that the test fails and concludes the proof. $\square$

3.3 Complexity Considerations

As can be seen from the specification of the test cases $U_F(k)$, the number of executions ending in a *pass*-event corresponds to the number ℓ of traces s of P with length equal to[5] k, multiplied by the number h of minimal hitting sets in $\mathit{minHit}(P/s)$.

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

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

The normalised transition graph of this process has a single state, and its initials are $[RUN(\Sigma)]^0 = \Sigma$. Therefore, the associated test process $U_F(k)$ can never

[5] In this estimation, we disregard the case where the test terminates earlier due to entering branch (22).

enter branches (21) and (22), but there are exactly $\mid \Sigma \mid^k$ different traces of length k exercising branch (23) for each of their events.

Estimation of h Given a set $minAcc(P/s) = \{A_1, \ldots, A_\alpha\}$ of minimal acceptances, the cardinality h of $minHit(P/s)$ is maximal for the case where all A_i are disjoint. In this case, $h = \mid minHit(P/s) \mid = \prod_{i=1}^{\alpha} \mid A_i \mid$. To find an upper bound for h, we use the result that we prove in the following lemma.

Lemma 8. *Let $n \geqslant 2$ be any positive integer. We call a sequence*

$$0 < a_1 \leqslant a_2 \leqslant \ldots \leqslant a_\alpha, a_i \in \mathbb{N}, i = 1, \ldots, \alpha$$

a partition of n, if $n = \Sigma_{i=1}^{\alpha} a_i$. Define

$$p(n) = \begin{cases} 3^k & \textit{if } n = 3k \\ 2^2 \cdot 3^{k-1} & \textit{if } n = 3k+1 \\ 2 \cdot 3^k & \textit{if } n = 3k+2 \end{cases}$$

Then

$$p(n) = \max\Big\{\prod_{i=1}^{\alpha} a_i \mid n = \Sigma_{i=1}^{\alpha} a_i, \ a_i, \alpha \in \mathbb{N}\Big\}.$$

Proof. Let $n \geqslant 2$. Obviously,

$$n \leqslant p(n). \tag{31}$$

Define $\pi(n)$ to be the following partition of n:

$$\pi(n) = \begin{cases} \underbrace{3, \ldots, 3}_{k \text{ times}} & \text{if } n = 3k \\ 2, 2, \underbrace{3, \ldots, 3}_{k-1 \text{ times}} & \text{if } n = 3k+1 \\ 2, \underbrace{3, \ldots, 3}_{k \text{ times}} & \text{if } n = 3k+2 \end{cases}$$

Then the product of the numbers in $\pi(n)$ equals $p(n)$. In the remainder of the proof we have to show that $p(n)$ is really maximal in the sense of the lemma.

Since the number of partitions of a given n is finite,

$$\max\{\prod_{i=1}^{\alpha} a_i \mid n = \Sigma_{i=1}^{\alpha} a_i, a_i \in \mathbb{N}\}$$

exists. Let $a_1, \ldots, a_\alpha$ be any partition of $n \geqslant 2$ with maximal product. Then, obviously, $a_i > 1$ for all $i = 1, \ldots, \alpha$. Suppose there is some a_j, $1 \leqslant j \leqslant \alpha$, with $a_j > 3$. Then from (31) we have

$$\prod_{i=1}^{\alpha} a_i \leqslant a_1 \cdot a_{j-1} \cdot p(a_j) \cdot a_{j+1} \ldots \cdot a_\alpha.$$

Hence we can replace a_j by $\pi(a_j)$, and the new partition has a product value which is not less then the original partition $a_1, \ldots, a_\alpha$. This process can be repeated, until every a_i in the resulting partition is either 2 or 3. Hence there exists a partition of n with maximal product, such that every term is 2 or 3. Since $6 = 2+2+2 = 3+3$ and $2^3 < 3^2$, the number of 2's in such a maximal partition is less than 3. Let $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_\alpha$ with $a_i \in \{2,3\}, i = 1, \ldots, \alpha$, be a partition of n with maximal product. Let $k_1 \leqslant 2$ be the number of terms equal 2 and k_2 be the number of terms equal 3. Every $n \geqslant 2$ can be represented by $n = 2k_1 + 3k_2$, $n \equiv 2k_1 (\mathsf{mod} 3)$ with $k_1 \leqslant 2$, and $\prod_{i=1}^{\alpha} a_i = 2^{k_1} \cdot 3^{k_2}$. We have

$$k_1 = \begin{cases} 0 & \text{if } n = 3k \\ 2 & \text{if } n = 3k+1 \\ 1 & \text{if } n = 3k+2 \end{cases}$$

$$k_2 = \begin{cases} k & \text{if } n = 3k \\ k-1 & \text{if } n = 3k+1 \\ k & \text{if } n = 3k+2 \end{cases}$$

$$\prod_{i=1}^{\alpha} a_i = \begin{cases} 3^k & \text{if } n = 3k \\ 2^2 \cdot 3^{k-1} & \text{if } n = 3k+1 \\ 2 \cdot 3^k & \text{if } n = 3k+2 \end{cases}$$

This proves the lemma. □

From Lemma 8, we conclude that

$$h = \begin{cases} 3^{\lfloor \frac{|\Sigma|}{3} \rfloor} & \text{if } \mid \Sigma \mid \equiv \ 0 \ \mathsf{mod}\ 3 \\ 3^{\lfloor \frac{|\Sigma|}{3} \rfloor} + 3^{\lfloor \frac{|\Sigma|}{3} \rfloor - 1} & \text{if } \mid \Sigma \mid \equiv \ 1 \ \mathsf{mod}\ 3 \\ 2 \cdot 3^{\lfloor \frac{|\Sigma|}{3} \rfloor} & \text{if } \mid \Sigma \mid \equiv \ 2 \ \mathsf{mod}\ 3 \end{cases}$$

We note that h is significantly smaller than $2^{|\Sigma|} - 1$, this can also be intuitively motivated by the fact that for every $H \in \mathit{minHit}(P/s)$, all of its true subsets and true supersets are *not* contained in $\mathit{minHit}(P/s)$.

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

$$h \cdot \left(\tfrac{1-|\Sigma|^{pq}}{1-|\Sigma|}\right), \text{ or, asymptotically, } O(3^{\lfloor \frac{|\Sigma|}{3} \rfloor} \cdot \mid \Sigma \mid^{(pq-1)}).$$

From Lemma 4 we know that the worst-case bound for h above cannot be further improved, since the full collection of minimal hitting sets needs to be checked to verify *conf*. The authors of [7] and [2] suggest to test *every* non-empty subset of Σ whose events cannot be completely refused in a given process state of the reference model; this leads to a worst-case estimate of $2^{|\Sigma|} - 1$ for the number of different sets to be offered to the SUT in the last step of the test execution, so our approach reduces the number of test executions in comparison to [7, 2].

Upper Bound pq for Number of Test Cases According to Theorem 1, the tests $U_F(k)$ need to be executed for $k = 0, \dots, pq-1$ to guarantee completeness. This means that the SUT is verified with test traces up to, and including, length pq. It is interesting to investigate whether this maximal length is really necessary, or whether one could elaborate alternative complete test strategies where the SUT is tested with shorter traces only. Indeed, an example presented in [12, Exercise 5] shows that when testing for equivalence of deterministic FSMs, it is sufficient to test the SUT with traces of significantly shorter length.

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

Example 4. Consider the CSP reference process P and an erroneous implementation Q specified as follows.

$$\begin{aligned} P &= a \rightarrow P_1 \sqcap b \rightarrow P_1 \sqcap c \rightarrow P_1 \\ P_1 &= a \rightarrow P \,\square\, b \rightarrow P \end{aligned}$$

$$\begin{aligned} Q &= a \rightarrow Q_1 \,\square\, b \rightarrow Q_1 \\ Q_1 &= a \rightarrow Q_2 \,\square\, b \rightarrow Q_2 \\ Q_2 &= a \rightarrow Q \sqcap b \rightarrow Q \end{aligned}$$

Obviously, P's normalised transition graph has 2 nodes, while Q's graph has 3. It is easy to see (and can be checked with FDR4) that $P \sqsubseteq_T Q$, but $\neg(P \sqsubseteq_F Q)$. Furthermore, it can also be shown using the FDR4 tool that the "test passed condition"

$$(pass \rightarrow Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U_F(k)) \setminus \Sigma$$

holds for $U_F(0), \dots, U_F(4)$, but fails for $U_F(5)$. This means that the non-conformance of Q cannot be detected by any test trace of length less or equal to 5, but is revealed (as expected from Theorem 1) by a trace of length 6, because the last event offered by the test $U_F(5)$ is refused by Q. □

Generalising Example 4, it can be shown that for any pair $p, q \in \mathbb{N}$ whose greatest common divisor is 1, there exist reference processes with p states and implementation processes with q states, such that non-conformance of the implementation can only be detected with a trace of length pq.

It is discussed in Section 6 how the number of test traces to be executed by complete test suites for failures refinement can still be reduced *without* reducing the maximal length.

4 Finite Complete Test Suites for CSP Trace Refinement

For establishing trace refinement, the following class of adaptive test cases will be used for a given reference process P and integers $p \geqslant 0$.

$$\begin{aligned}
U_T(p) &= U_T(p, \varepsilon) && (32)\\
U_T(p,s) &= \big(\Box\, e : (\Sigma - [P/s]^0) \bullet e \to \mathit{fail} \to \mathit{Stop}\big) && (33)\\
&\quad \Box\\
&\quad (\mathit{minAcc}(P/s) = \{\varnothing\}) \& \big(\mathit{pass} \to \mathit{Stop}\big) && (34)\\
&\quad \Box\\
&\quad (\#s < p) \& \big(\Box\, e : [P/s]^0 \bullet e \to U_T(p, s.e)\big) && (35)\\
&\quad \Box\\
&\quad (\#s = p) \& \big(\mathit{pass} \to \mathit{Stop}\big) && (36)
\end{aligned}$$

The difference between adaptive tests $U_T(p)$ for trace refinement and $U_F(p)$ for failures refinement consists in the fact that the former do not "probe" the SUT with respect to minimal sets of events to be accepted without blocking.

The existence of complete, finite test suites is expressed in analogy to Theorem 1. A noteworthy difference is that the complete suite for trace refinement just needs the single adaptive test case $U_T(pq-1)$, while failures refinement required the execution of $\{U_F(0), \ldots, U_F(pq-1)\}$. The reason for this 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 $(pq-1)$. Since $U_T(pq-1)$ never blocks any Q-event before terminating, the pass criterion can be based on trace refinement instead of failures refinement as required in (29).

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

Theorem 2. *Let P be a 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*

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

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

Proof. The theorem follows directly from Step 1 in the proof of Lemma 6 and Case 1 in the proof of Lemma 7. □

5 Testing for Failures Refinement – an Example

For implementing the test case $U_F(p)$ with sub-processes $U_F(p,s)$, it is advisable to avoid an enumeration of traces s of the reference process. Instead, we calculate the following auxiliary functions from P's transition graph.

$$\begin{aligned} \mathit{initials} &: N \to \mathbb{P}(\Sigma) \\ \mathit{minHit} &: N \to \mathbb{P}\,\mathbb{P}(\Sigma) \end{aligned}$$

As mentioned, in a state $n = G(P)/s$, the set $\mathit{initials}(n)$ equals the events labelling outgoing edges of n, so $\mathit{initials}(n) = [P/s]^0$. The function minHit maps n to the set of all minimal hitting sets associated with the minimal acceptances of n, so $\mathit{minHit}(n) = \mathit{minHit}(P/s)$. Using the transition function t and the above functions, $U_F(p)$ can be re-written as the failures-equivalent CSP process below.

$$\begin{aligned} U_F^1(p) &= U_F^1(p, 0, \underline{n}) && (38)\\ U_F^1(p,k,n) &= \big(\Box\, e : (\Sigma - \mathit{initials}(n)) \bullet e \to \mathit{fail} \to \mathit{Stop}\big) && (39)\\ &\Box \\ &(\mathit{minHit}(n) = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) && (40)\\ &\Box \\ &(k < p)\&\big(\Box\, e : \mathit{initials}(n) \bullet e \to U_F^1(p, (k+1), t(n,e))\big) && (41)\\ &\Box \\ &(k = p)\&\big(\sqcap_{H \in \mathit{minHit}(n)} (\Box\, e : H \bullet e \to \mathit{pass} \to \mathit{Stop})\big) && (42) \end{aligned}$$

FiXme Fatal: alcc: I think the example should be for the theory presented before, not with a different notion of test. In particular, 33 is very different.

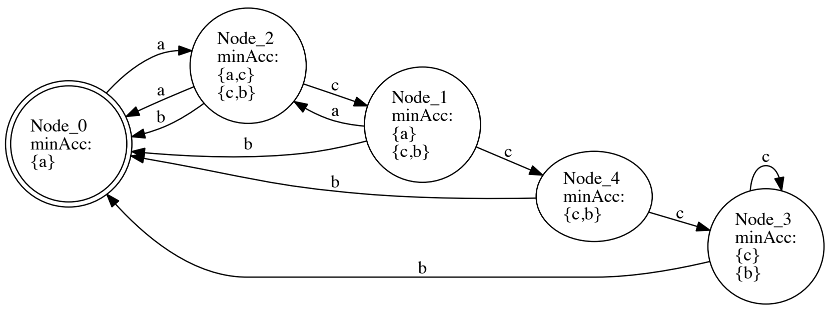

Fig. 2. Normalised transition graph of faulty implementation Z from Example 5.

Example 5. Consider the following implementation Z of process P from Example 1 that is erroneous from the point of view of failures refinement.

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

It is easy to see (and can be checked with FDR) that Z is trace-equivalent to P. While $k < r_{max}$, Z also accepts the same sets of events as P. When $R_1(r_{max}, k)$ runs through several recursions and $k = r_{max}$ is fulfilled, however, $R_1(r_{max}, k)$ makes an internal choice, instead of offering an external choice, and refinement does not hold. Fig. 2 shows the normalised transition graph of Z for $r_{max} = 3$.

Running the test $U_F^1(k)$ against Z for $k = 0, \dots, 20$ ($G(P)$ has $p = 4$ states and $G(Z)$ has $q = 5$, so $pq = 20$ is an upper bound for the test depth to be used according to Theorem 1), tests $U_F^1(0), \dots, U_F^1(3)$ are passed by Z, but Z fails $U_F^1(4)$, because after execution trace

$$s = a.c.c.c, \qquad \text{(note that } G(Z)/s = \text{Node\_3 according to Fig. 2)},$$

it may be the case that Z accepts only $\{b\}$ due to the internal choice and $U_F^1(4)$ – also due to internal choice – accepts only the minimal hitting set $\{c\}$ and the event $a \in (\Sigma - [P/s]^0)$. So, $(Z \,|[\,\Sigma\,]|\, U_F^1(4))/s$ deadlocks, and the *pass* event cannot be produced. Another failing execution arises if Z/s chooses to accept only $\{c\}$, while $U_F^1(4)/s$ choses to accept only $\{a, b\}$. Therefore,

$$(pass \to Stop) \not\sqsubseteq_F (Z \,|[\,\Sigma\,]|\, U_F^1(4)),$$

and the test fails. □

FiXme Fatal: alcc: I would write down the actual test generated.

6 Discussion and Conclusions

Discussion of Further Reductions of the Test Effort It is known from complete testing strategies for finite state machines that the upper bound pq for the lengths of the traces used in our tests to investigate the SUT behaviour can be reduced. It is also known from FSM testing that it is not necessary to test *all* traces up to this maximal length. Notable complete strategies supporting this fact have been presented, for example, in [8, 4, 14, 22]. 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 in the future. These will effectively reduce the upper bound $\ell \leqslant \mid \Sigma \mid^k$ introduced in Section 3.3. The bound h for the number of sets to be used in probing the SUT for illegal deadlocks, however, cannot be further reduced, as has been established in Lemma 4.

Discussion of Adaptive Test Cases The tests suggested in [7, 2] were *preset* in the sense that the trace to be executed was pre-defined for each test. As a consequence, the authors of [2] introduced *inconclusive* as a third test result, applicable to the situations where the intended trace of the execution was blocked, due to legal, but nondeterministic behaviour of the SUT.

In contrast to that, our test cases specified in Section 3 and 4 are adaptive. This has the advantage that test executions $(Q \,|[\, \Sigma \,]|\, U_F(p))$ for failures refinement never blocks before the final step specified by branch (24), and so we do not need inconclusive test results. It should be noted, however, that it is necessary for our test verdicts to recognise also deadlocks in the final test step and interpret them as FAIL, as described in Section 3. In practice, this is realised by adding a timeout event to the testing environment which indicates deadlock situations. For real-time systems, this is an accepted technique, because the SUT has to respond within a pre-defined latency interval, otherwise its behaviour is considered to be blocked and regarded as a failure.

FiXme Fatal: alcc: if deadlock is not acceptable? I think you need instrumentation here as well.

Our tests executions $(Q \,|[\, \Sigma \,]|\, U_T(p))$ for trace refinement never block at all. The adaptive behaviour of our test cases, however, induce the obligation to check that *all* possible executions have been performed before the test can be considered as passed. Typically, it is therefore assumed that a *complete testing assumption* [8] holds, which means that every possible behaviour of the SUT is performed after a finite number of test executions. In practice, this is realised by executing each test several times, recording the traces that have been performed, and using hardware or software coverage analysers to determine whether all possible test execution behaviours of the SUT have been observed. Therefore, adaptive test cases come at the price of having to apply some grey-box testing techniques enabling us to decide whether all SUT behaviours have been observed.

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

Implications for CSP Model Checking As explained in the previous sections, passing a test is specified by $(pass \to Stop) \sqsubseteq_F (Q \,|[\, \Sigma \,]|\, U_F(k)) \setminus \Sigma$ 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 these pass criteria using the FDR4 model checker. For checking the refinement relation $P \sqsubseteq_F Q$, the pass criteria have to be verified for $k = 0, \ldots, pq - 1$, 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 1). Since the test cases $U_F(k)$ have such a simple structure, it is an interesting question for further research whether checking $(\mathit{pass} \to \mathit{Stop}) \sqsubseteq_F (Q \,|[\,\Sigma\,]|\, U_F(k)) \setminus \Sigma$ for $k = 0, \ldots, pq - 1$ 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[6]

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

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

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

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

Acknowledgements The authors would like to thank Bill Roscoe and Thomas Gibson-Robinson for their advice on using the FDR4 model checker and for very helpful discussions concerning the potential implications of this paper in the field of model checking. The work of Cavalcanti is funded by the Royal Academy of Engineering and UK EPSRC Grant EP/R025134/1.

References

1. Buth, B., Kouvaras, M., Peleska, J., Shi, H.: Deadlock analysis for a fault-tolerant system. In: Johnson, M. (ed.) Algebraic Methodology and Software Technology, 6th

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

International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1349, pp. 60–74. Springer (1997), https://doi.org/10.1007/BFb0000463

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

15. Petrenko, A., Yevtushenko, N.: Adaptive testing of nondeterministic systems with FSM. In: 15th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2014, Miami Beach, FL, USA, January 9-11, 2014. pp. 224–228. IEEE Computer Society (2014), `http://dx.doi.org/10.1109/HASE.2014.39`
16. Roscoe, A.W.: Model-checking csp. In: Roscoe, A.W. (ed.) A Classical Mind: Essays in Honour of C. A. R. Hoare, chap. 21, pp. 353–378. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)
17. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
18. Roscoe, A.W.: Understanding Concurrent Systems. Springer, London, Dordrecht Heidelberg, New York (2010)
19. Schneider, S.: An operational semantics for timed csp. Inf. Comput. 116(2), 193–213 (Feb 1995), `http://dx.doi.org/10.1006/inco.1995.1014`
20. Shi, H., Peleska, J., Kouvaras, M.: Combining methods for the analysis of a fault-tolerant system. In: 1999 Pacific Rim International Symposium on Dependable Computing (PRDC 1999), 16-17 December 1999, Hong Kong. pp. 135–142. IEEE Computer Society (1999), `https://doi.org/10.1109/PRDC.1999.816222`
21. Shi, L., Cai, X.: An exact fast algorithm for minimum hitting set. In: 2010 Third International Joint Conference on Computational Science and Optimization. vol. 1, pp. 64–67 (May 2010)
22. Simo, A., Petrenko, A., Yevtushenko, N.: On reducing test length for FSMs with extra states. Software Testing, Verification and Reliability 22(6), 435–454 (Sep 2012), `https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.452`