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 refinement and failures refinement are constructed, and their finiteness and completeness (i.e. 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.

While MBT is applied in different flavours, we consider the most effective variant to be the one where test cases and concrete test data, as well as checkers for the expected results (*test oracles*), are automatically generated from a reference model. This guarantees maximal return of the investment of time and effort to create the test model. The test suites generated in this way, however, usually have different test strength, depending on the generation algorithms applied.

For 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 the SUT. The so-called *complete* test strategies guarantee to uncover every conformance violation of the

SUT with respect to a reference model, provided that the true SUT behaviour is captured by a member of the fault domain.

Generation methods for complete test suites have been developed for various modelling formalisms. In this paper, we use *Communicating Sequential Processes (CSP)* [7, 13]. This is a mature process-algebraic approach that has been shown to be well-suited for the description of reactive control systems in many publications over almost five decades. Industrial success has also been reported.

Contributions This paper presents complete black-box test suites for software and systems modelled using the *Communicating Sequential Processes (CSP)* process algebra [7, 12]. This process algebra has been widely used for the specification and verification of communicating concurrent control systems. Many of these applications have been described in [13] and in the references there.

The complete testing theory presented here applies to CSP processes that are divergence-free[4] and interpreted both in the trace and the failures semantics. Our results complement work published in [2]. 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 [2] may require additional criteria to select tests from still infinite test suites, here, we further restrict fault domains using a graph representation of processes used in model checking to obtain test suites that are finite.

Our complementary approach to the definition of CSP fault domains is presented in this paper. We observe that every finite-state CSP process can be 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 [11]. The maximal refusals express the degree of nondeterminism present in a given process state that is in one-one-correspondence to a node of the normalised transition graph. Inspired by the way that fault-domains are specified for finite state machines (FSMs), we define them as the set of CSP processes whose normalised transition graphs do not exceed the size of the reference model's graph by more than a given constant.

The main contribution of this paper is the proof that for fault domains of the described type, finite, complete test suite generation methods can be given for testing against trace and failures refinement and equivalence. The existence of – possibly infinite – complete test suites has been established for process algebras, and for CSP in particular, by several authors [5, 14, 9, 8, 2]. 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

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

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 [6, 10].

It should be noted that the presented results are of a theoretical nature: despite being finite, the resulting test suite size will still be too large to be applied in practice to real-world problems. We discuss a number of promising options how the test suite size can be reduced to become practically applicable.

Overview In Section 2, we present the background material relevant to our work.

@todo

2 Preliminaries

2.1 CSP and Refinement

First, we introduce the CSP notation and refinement notions. We then discuss a graph-based semantics for finite-state CSP processes.

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

A prefixing operator $e \to P$ defines a process that is ready to engage in the event e, pending agreement of its environment to synchronising. 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 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 itself.

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

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

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

Iterated forms $\Box\, i : I \bullet P(i)$ and $\sqcap\, i : I \bullet P(i)$ of the external and internal choice operators allow us to define a choice over a collection of processes $P(i)$.

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 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 may be desirable to make interal actions of each process in a 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 [13].

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 $\mathit{traces}(P)$ 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 $\mathit{traces}(Q) \subseteq \mathit{traces}(P)$.

In fact, in every semantic model, subset containment is used to define refinement. The model we focus here 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 $\mathit{failures}(P)$. Using notation P/s to denote the process P after having engaged into trace s, the set $\mathit{Ref}(P/s) \mathrel{\widehat{=}} \{\, X \mid (s, X) \in \mathit{failures}(P) \,\}$ contains the *refusals* of P after the trace t. Refusals are subset-closed [7, 13]: 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.

Failures refinement, $P \sqsubseteq_F Q$, is defined by set containment $\mathit{failures}(Q) \subseteq \mathit{failures}(P)$. Since refusals are subset-closed, this implies $(s, \varnothing) \in \mathit{failures}(P)$ for all traces $s \in \mathit{traces}(Q)$. As a consequence, failures refinement implies trace refinement. Therefore, using the additional conformance relation

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

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

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

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 [4] supports model checking and semantic analyses of finite-state CSP processes.

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

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

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

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

The cardinality of $\mathit{maxRef}(P/s)$ reflects the degree of nondeterminism that is present in process state P/s: the more maximal refusal sets contained in $\mathit{maxRef}(P/s)$, the more nondeterministic is the behaviour in state P/s. Deterministic process states P/s have exactly the one maximal refusal $\Sigma - [P/s]^0$, where $[P/s]^0$ denotes the *initials* of P/s, i.e., the events that P/s may engage into.

Normalised Transition Graphs for CSP Processes As shown in [11], the failures semantics of any finite-state CSP process P can be represented by a *normalised transition graph*

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

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 [11] implies that all nodes n in N are reachable 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 s. In analogy to $\mathit{traces}(P)$, we use 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, and we note that $\mathit{traces}(P) = \mathit{traces}(G(P))$. 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 [11] guarantees that $[G(P)/s]^0 = [P/s]^0$ for all traces of P.

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

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

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

Acceptances When investigating tests for failures refinement, the notion of *acceptances* [5], 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 refusing the events in $\Sigma - A$. The acceptances of P/s are denoted by $\mathit{Acc}(P/s)$, and the minimal acceptances by $\mathit{minAcc}(P/s)$. They satisfy the following properties.

$$A \in \mathit{minAcc}(P/s) \Leftrightarrow \exists\, R \in \mathit{maxRef}(P/s) \wedge A = \Sigma - R \quad (6)$$

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

$$X \in \mathit{Acc}(P/s) \Leftrightarrow A \in \mathit{minAcc}(P/s) \wedge A \subseteq X \subseteq [P/s]^0 \quad (8)$$

Exploiting formulas (6), (7), and (8), 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 $\mathit{minAcc}(G(P)/s)$ and note that (5) and (6) imply

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

Example 2. Consider the CSP process P in Example 1. Its transition graph $G(P)$ is shown in Fig. 1. Process state P/ε (ε denotes the empty trace) is represented there as Node_0, with $\{a\}$ as the only minimal acceptance, since a can never be refused, and no other events are accepted. Having engaged into a, the transition emanating from Node_0 leads to Node_2 representing the process state $P/a = Q \sqcap R$. The internal choice operator 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 $\{b, c\}$. Note that event c can never be refused, since it is contained in each minimal acceptance set.

Having engaged into 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.

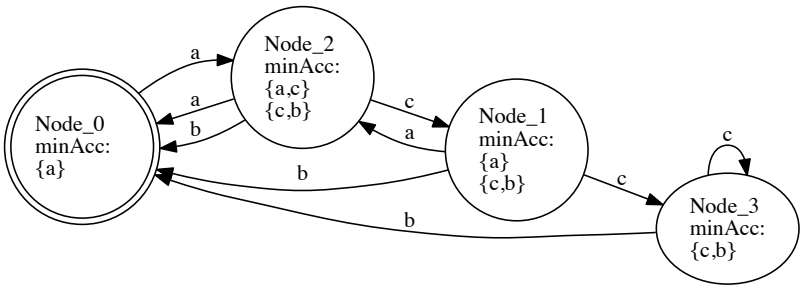

Fig. 1. Normalised transition graph of CSP process P from Example 2.

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

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

and

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

in the following way.

Lemma 1.

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

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

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

$$\begin{aligned} \Leftrightarrow{}& \forall\, s \in \mathit{traces}(G(Q)) \cap \mathit{traces}(G(P)), A_Q \in \mathit{minAcc}(G(Q)/s) : \\ & \exists\, A_P \in \mathit{minAcc}(G(P)/s) : A_P \subseteq A_Q \end{aligned} \tag{13}$$

□

Formula (10) reflects trace refinement in terms of graph traces. Formula (12) states how the *conf* conformance relation 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).

Given a finite-state CSP process P and its normalised transition graph

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

suppose that $V \subseteq \Sigma^*$ is a prefix-closed set of sequences of events. By $t(\underline{n}, V)$ we denote the set

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

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

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

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

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

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. Therefore, the labelling of nodes with maximal refusals or minimal acceptances are disregarded in the product construction. 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 will be 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 $(pq-1)$. □

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 Σ, a *hitting set* $H \subseteq \Sigma$ 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). The problem of determining minimal hitting sets is known to be NP-hard [1], but we will see below that it reduces the effort of testing for failures refinement from a factor of 2^{Σ} to a factor that equals the number of minimal hitting sets.

For testing, the following lemma about hitting sets are required.

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

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

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

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

Therefore, $H \in \text{minHit}(P/s)$ 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 $H \in \text{minHit}(P/s)$ is also a hitting set for $minAcc(G(Q)/s)$.

To prove "2 ⇒ 1", assume that that Statement 2 of the lemma holds but that P *conf* Q does *not* hold. According to Lemma 1, (13), 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

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

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 $H \in \text{minHit}(P/s)$ which is *not* a hitting set of $\mathit{minAcc}(G(Q)/s)$, a contradiction to Assumption 2. This completes the proof of the lemma. □

This result is used in the sequel in Section 3.

3 Finite Complete Test Suites for CSP Failures Refinement

3.1 Test Cases for Verifying CSP Failures Refinement

For a given reference process P and for each integer $p \geqslant 0$, we define a CSP test process for failures refinement as shown below.

$$U_F(p) = U_F(p, \varepsilon) \tag{18}$$

$$U_F(p, s) = \big(\ \Box\ e : (\Sigma - [P/s]^0) \bullet e \to \mathit{fail} \to \mathit{Stop}\big) \tag{19}$$

$$\Box$$

$$([P/s]^0 = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{20}$$

$$\Box$$

$$(\#s < p)\&\big(\ \Box\ e : [P/s]^0 \bullet e \to U_F(p, s.e)\big) \tag{21}$$

$$\Box$$

$$(\#s = p)\&\big(\ \sqcap_{H \in \text{minHit}(P/s)}\ (\Box\ e : H \bullet e \to \mathit{pass} \to \mathit{Stop})\big) \tag{22}$$

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

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

It is assumed that the events *fail* and *pass*, denoting FAIL and PASS of the test execution, are events outside Σ. Since we assume that Q is free of livelocks, it is guaranteed that each test execution terminates after some $s \in \mathit{traces}(P)$ with length $(p+1)$ at the latest. 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 $\overline{\text{PASS}}$ event *pass*. This can also be expressed by means of a failures refinement.

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

This type of pass relation is often called *must test*, because every test execution must end with the *pass* event [5]. 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 $\langle pass \rangle$ as the "Test Passed Process" ($pass \rightarrow Stop$). However, the former may nondeterministically refuse *pass*, due to a deadlock occurring when a faulty SUT process executes concurrently with $U_F(p, s)$ executing branch (22), because $\#s = p$. This is explained further in the next paragraphs.

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 $s \in traces(P)$ with $\#s < p$, an event is accepted by the SUT that is outside the initials of P/s, the test immediately terminates with FAIL-event *fail*. This is handled by the branch (19) of the external choice in the process $U_F(p, s)$ defined above.

If P/s is the $STOP$ process, this is revealed by its initials being empty. In this case, the test may terminate successfully (branch (20) of the external choice in $U_F(p, s)$). Note that at the same time, any (illegal) event of the alphabet is also accepted by the test in branch (19). So, if the SUT accepts an event in a state where P/s is supposed to have stopped, there exists a test execution that terminates with FAIL by choosing the first branch of the external choice.

If the length of s is still less than p, the test accepts any event from the initials $[P/s]^0$ and continues recursively as $U_F(p, s.e)$ in branch (21). 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 successfully through a trace of length p, 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 of $minAcc(P/s)$ and only offers the events contained in this set. If the SUT refuses to engage into any of these events, this reveals a violation of the failures refinement: according to Lemma 4, a conforming SUT should accept at least one event of each minimal hitting set in $\text{minHit}(P/s)$. Therefore, the test only terminates with success *pass*, if such an event is accepted by the SUT.

3.2 Fault Models and Complete Test Suites

A *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.

3.3 A Finite Complete Test Suite for Failures Refinement

After the informal explanation of tests representing adaptive test cases, we are ready to state the main theorem of this paper.

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

Proof. 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 (20), (21), or (22) of the external choice construction for $U_F(p, s)$. Branch (19) 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.

Moreover, Lemma 4 implies that for all traces $s \in \mathit{traces}(Q) \cap \mathit{traces}(P)$, every H in $\mathrm{minHit}(P/s)$ is a hitting set for $\mathit{minAcc}(Q/s)$. Branch (20) of test $U_F(p, s)$ leads always to a PASS verdict, and branch (21) to test continuation without a verdict. For the last branch, we note that any selected minimal hitting set $H \in \mathrm{minHit}(P/s)$ 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 6. *Test suite TS_F specified im 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 $\neg(P \mathit{\ conf\ } Q)$, 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)).

$$\forall\, A_P \in \mathit{minAcc}(P/s) : A_P \not\subseteq A_Q, \tag{23}$$

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 *fail* or without giving any verdict. The latter case is also interpreted as FAIL, since then the process $\mathit{pass} \rightarrow \mathit{Stop}$ 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 (19) 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 (21) until the trace u has been completely processed. $U_F(\#u, u)$ may execute branches (19) or (22) only: assumption (23) in Case 2 implies that P/s has at least one non-empty minimal acceptance, so the guard condition $([P/s]^0 = \varnothing)$ of branch (20) evaluates to `false` for $U_F(\#u, u)$. Moreover, the guard condition $(\#s < p)$ for branch (21) evaluates to `false` for $U_F(\#u, u)$, too. If branch (19) is executed, the test always fails. If branch (22) is executed, the test fails for the execution where a minimal hitting set $H \in \mathrm{minHit}(P/u)$ is chosen by $U_F(\#u, u)$ that has an empty intersection with the minimal acceptance A_Q from condition (23). 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 *pass*; this means that the test fails. This concludes the proof. □

4 Finite Complete Test Suites for CSP Trace Refinement

For establishing trace refinement, the following class of adaptive test cases will be used; again, they are defined for integers $p \geqslant 0$.

$$U_T(p) = U_T(p, \varepsilon) \tag{24}$$

$$U_T(p, s) = \big(\,\Box\, e : (\Sigma - [P/s]^0) \bullet e \to \mathit{fail} \to \mathit{Stop}\big) \tag{25}$$

$$\Box$$

$$([P/s]^0 = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{26}$$

$$\Box$$

$$(\#s < p)\&\big(\,\Box\, e : [P/s]^0 \bullet e \to U_T(p, s.e)\big) \tag{27}$$

$$\Box$$

$$(\#s = p)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{28}$$

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 $(pq-1)$.

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 $\mathcal{F} = (P, \sqsubseteq_T, \mathcal{D})$. □

We skip the proof of Theorem 2, since it is just a simplified version of the proof of Theorem 1.

5 Applications

5.1 Testing for Trace Refinement

5.2 Testing for Failures Refinement

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.

$$\text{initials} : N \to \mathbb{P}(\Sigma)$$
$$\text{minHit} : N \to \mathbb{P}\,\mathbb{P}(\Sigma)$$

In a state $n = G(P)/s$, the set initials(n) equals the events labelling outgoing edges of n, so initials(n) $= [P/s]^0$. Function minHit maps n to the set of all minimal hitting sets associated with the minimal acceptances of n, so minHit(n) = 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.

$$U_F^1(p) = U_F^1(p, 0, \underline{n}) \tag{29}$$
$$U_F^1(p,k,n) = \big(\,\Box\, e : (\Sigma - \text{initials}(n)) \bullet e \to \mathit{fail} \to \mathit{Stop}\big) \tag{30}$$
$$\Box$$
$$(\text{initials}(n) = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{31}$$
$$\Box$$
$$(k < p)\&\big(\,\Box\, e : \text{initials}(n) \bullet e \to U_F^1(p, (k+1), t(n,e))\big) \tag{32}$$
$$\Box$$
$$(k = p)\&\big(\,\sqcap_{H \in \text{minHit}(n)}\, (\Box\, e : H \bullet e \to \mathit{pass} \to \mathit{Stop})\big) \tag{33}$$

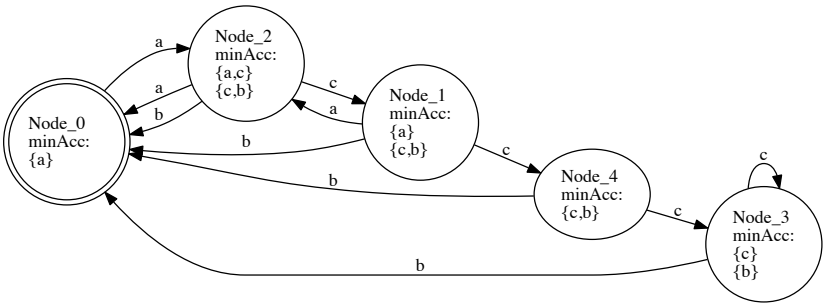

Fig. 2. Normalised transition graph of faulty implementation Z from Example 3.

Example 3. Consider the following erroneous implementation Z of process P from Example 2 from the point of view of failures refinement.

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

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 $k = 0, \ldots, 20$ ($G(P)$ 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 $U_F^1(0), \ldots, U_F^1(3)$ 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,

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

and the test fails. □

6 Related Work

7 Conclusion

Acknowledgements The authors would lime 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.

References

1. Book, R.V.: Karp richard m.. reducibility among combinatorial problems. complexity of computer computations, proceedings of a symposium on the complexity of computer computations, held march 20-22, 1972, at the ibm thomas j. watson center, yorktown heights, new york, edited by miller raymond e. and thatcher james w., plenum press, new york and london 1972, pp. 85–103. Journal of Symbolic Logic 40(4), 618–619 (1975)
2. 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
3. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: Failures Divergences Refinement (FDR) Version 3 (2013), https://www.cs.ox.ac.uk/projects/fdr/
4. 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)
5. Hennessy, M.: Algebraic Theory of Processes. MIT Press, Cambridge, MA, USA (1988)
6. 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
7. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
8. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Jounal 19, 53–77 (1997)
9. 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
10. 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

11. Roscoe, A.W. (ed.): A Classical Mind: Essays in Honour of C. A. R. Hoare. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)
12. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
13. Roscoe, A.W.: Understanding Concurrent Systems. Springer, London, Dordrecht Heidelberg, New York (2010)
14. 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