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, new contributions to testing Communicating Sequential Processes (CSP) are presented, with focus on the generation of complete, finite test suites. A test suite is complete if it can uncover every conformance violation of the system under test with respect to a reference model. Both reference models and implementation behaviours are represented as CSP processes. As conformance relation, we consider trace equivalence and refinement, as well as failures equivalence and refinement. Complete black-box test suites here rely on the fact that the SUT's true behaviour is represented by a member of a fault-domain, that is, a collection of CSP processes that may or may not conform to the reference model. We define fault domains by bounding the number of excessive states occurring in a fault domain member's representation as a normalised transition graph, when comparing it to the number of states present in the graph of the reference model. This notion of fault domains is quite close to the way they are defined for finite state machines, and these fault domains guarantee the existence of *finite* complete test suites.

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)* [6, 10]. 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 CSP processes that are divergence-free[4] and interpreted both in the trace and the failures semantics. Our results complement work by two of the authors 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. Yhis 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] requires additional criteria to select tests from 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 semantically 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 [9]. 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, 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, 11, 8, 7, 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.

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

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

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

Communicating Sequential Processes @todo

FiXme Warning: alcc: I can make this small contribution.

Throughout this paper, the alphabet of CSP processes P is denoted by Σ. Since we are only considering "classical" CSP processes, the alphabet is always finite.

- traces and failures model
- refinement and equivalence

The FDR tool [4] supports model checking and semantic analyses of CSP processes.

Normalised Transition Graphs for CSP Processes As shown in [9], 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 from n with label e. Normalisation of $G(P)$ is reflected by the fact that t is a function.

A finite sequence of events $s \in \Sigma^*$ is a *trace* of P, if there is a path through $G(P)$ starting at $\underline{n}$ whose edge labels coincide with s. The set of traces of P is denoted by $\mathrm{trc}(P)$. If $s \in \mathrm{trc}(P)$, then the process corresponding to P after having executed s is denoted by P/s. 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.

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

We also use this notation for CSP processes: $[P]^0$ is the set of events P may engage into, in other words, the initials of P after the empty trace of events, that is, $\mathit{initials}(P/\langle\rangle)$ as defined in [10], for example.

The total function r maps each node n to its *refulsals* $r(n) = \mathrm{Ref}(n)$. Each element of $r(n)$ is a set of events that the CSP process P might refuse to engage

into, when in a process state corresponding to n. If P/s is deterministic, its refusals coincide with the set of subsets of $\Sigma - [P/s]^0$, including the empty set.

For finite CSP processes, since the refusals of each process state are subset-closed [6, 10], $\mathrm{Ref}(P/s)$ can be constructed from the set of *maximal refusals* $\mathrm{maxRef}(P/s) \subseteq \mathrm{Ref}(P/s)$. More formally, $\mathrm{maxRef}(P/s)$ is defined as follows.

$$\mathrm{maxRef}(P/s) = \{R \in \mathrm{Ref}(P/s) \mid \forall\, R' \in \mathrm{Ref}(P/s) - \{R\} : R \not\subseteq R'\} \quad (1)$$

Conversely, with the maximal refusals $\mathrm{maxRef}(P/s)$ at hand, we can reconstruct the refusals $\mathrm{Ref}(P/s)$ by subset-closure as follows.

$$\mathrm{Ref}(P/s) = \{R' \in \mathbb{P}(\Sigma) \mid \exists\, R \in \mathrm{maxRef}(P/s) : R' \subseteq R\}. \quad (2)$$

The cardinality of $\mathrm{maxRef}(P/s)$ reflects the degree of nondeterminism that is present in process state P/s: the more maximal refusal sets contained in $\mathrm{maxRef}(P/s)$, the more nondeterministic is the behaviour in state P/s. Deterministic process states P/s have exactly the one maximal refusal $\Sigma - [P/s]^0$.

To see that this approach works only for finite CSP processes, we consider the example where Σ is infinite. In this case, $\mathrm{maxRef}(\mathit{Stop}/\langle\rangle)$ is empty, and so we cannot use this set to calculate the refusals of *Stop*, that is, $\mathrm{Ref}(\mathit{Stop}/\langle\rangle)$ as defined above. As with refusals, we also use the transition graph-oriented notation $\mathrm{maxRef}(n) \subseteq r(n)$ to denote the maximal refusals associated with graph state n: if n is the state reached in the transition graph by following the edge labels in trace s, then $\mathrm{maxRef}(n) = \mathrm{maxRef}(P/s)$.

Well-formed normalised transition graphs must not refuse an event of the initials of a state in *every* refusal applicable in this state; more formally,

$$\forall\, n \in N, e \in \Sigma : (n, e) \in \mathrm{dom}\ t \Rightarrow \exists\, R \in \mathrm{maxRef}(n) : e \notin R \quad (3)$$

By construction, normalised transition graphs reflect the *failures semantics* of finite-state CSP processes: the traces s of a process are defined by the sequences of transitions associated with paths through its graph, starting at $\underline{n}$. The pairs (s, R) with $s \in \mathrm{trc}(P)$ and $R \in r(G(P)/s)$ represent the failures of P.

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 the initials $[P/s]^0$, i.e., 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 $\mathrm{Acc}(P/s)$, and the minimal acceptances by $\mathrm{minAcc}(P/s)$. They satisfy the following properties.

$$A \in \mathrm{minAcc}(P/s) \Leftrightarrow \exists\, R \in \mathrm{maxRef}(P/s) \wedge A = \Sigma - R \quad (4)$$

$$\bigcup \{A \mid A \in \mathrm{Acc}(P/s)\} = [P/s]^0 \quad (5)$$

$$X \in \mathrm{Acc}(P/s) \Leftrightarrow A \in \mathrm{minAcc}(P/s) \wedge A \subseteq X \subseteq [P/s]^0 \quad (6)$$

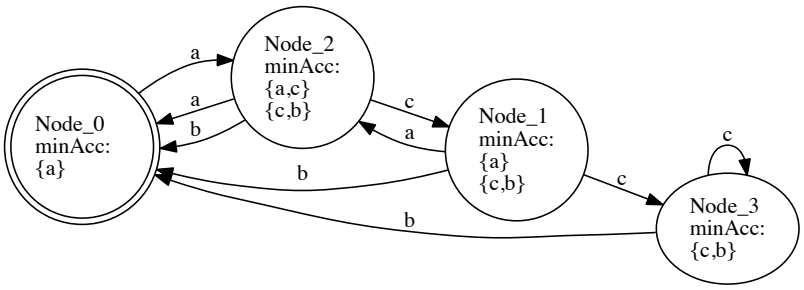

Fig. 1. Normalised transition graph of CSP process P from Example 1.

Note that (4) can be regarded as a definition of minimal acceptances by means of maximal refusals. In this case, (5) and (6) are consequences of this definition and the fact that refusals are subset-closed.

FiXme Fatal: alcc: are you sure? (4) does not talk about Acc. Is this important?

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. This is the representation used by FDR.

Example 1. Consider the CSP process P defined below, and $\Sigma = \{a, b, c\}$.

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

Its transition graph $G(P)$ is shown in Fig. 1. Process state $P/\langle\rangle$ 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.

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. □

Refinement relations between finite-state CSP processes P, Q can be be expressed by means of their normalised transition graphs in the following way.

FiXme Fatal: alcc: trc was defined for P, not G(P) and this lemma is used later for trc(P). So, we need to be clearer as to what we mean.

Lemma 1.

$$P \sqsubseteq_T Q \Leftrightarrow trc(G(Q)) \subseteq trc(G(P)) \tag{7}$$

$$P \sqsubseteq_F Q \Leftrightarrow trc(G(Q)) \subseteq trc(G(P)) \wedge \forall\, s \in trc(G(Q)), R_Q \in maxRef(G(Q)/s) : \exists\, R_P \in maxRef(G(P)/s) : R_Q \subseteq R_P \tag{8}$$

$$P \sqsubseteq_F Q \Leftrightarrow trc(G(Q)) \subseteq trc(G(P)) \wedge \forall\, s \in trc(G(Q)), A_Q \in minAcc(G(Q)/s) : \exists\, A_P \in minAcc(G(P)/s) : A_P \subseteq A_Q \tag{9}$$

□

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

$$G_1 \times G_2 = (N_1 \times N_2, (\underline{n}_1, \underline{n}_2), t : (N_1 \times N_2) \times \Sigma \nrightarrow (N_1 \times N_2)) \tag{10}$$

$$\text{dom } t = \{((n_1, n_2), e) \in (N_1 \times N_2) \times \Sigma \mid (n_1, e) \in \text{dom } t_1 \wedge (n_2, e) \in \text{dom } t_2\} \tag{11}$$

$$t((n_1, n_2), e) = (t_1(n_1, e), t_2(n_2, e)) \text{ for } ((n_1, n_2), e) \in \text{dom } t \tag{12}$$

Tool Considerations FDR provides an API [3] that can be used to construct normalised transition graphs for CSP processes. The FDR graph nodes are labelled by *minimal acceptances* instead of maximal refusals as described above. Since such a minimal acceptance set is the complement of a maximal refusal, the function r mapping states to their refusals can be implemented by taking the complements of all minimal acceptances and then building all their subsets. For practical applications, the subset closure is never constructed in an explicit way; instead, sets are checked with respect to containment in a maximal refusal.

@todo

2.2 Test Cases and Complete Test Suites

@todo

2.3 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.

$$\forall\, A \in C : H \cap A \neq \varnothing. \tag{13}$$

A *minimal hitting set* is a hitting set that cannot be further reduced without losing the characteristic property (13). 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 2. *Let P, Q be two finite-state CSP processes satisfying $P \sqsubseteq_T Q$. For each $s \in trc(P)$, let minHit(P/s) denote the collection of all minimal hitting sets of minAcc(P/s). Then the following statements are equivalent.*

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

FiXme Fatal: Note that if traces refinement is already given, this is conf.

Proof. For showing "1 $\Rightarrow$ 2", assume that $P \sqsubseteq_F Q$ and suppose that $s \in \mathrm{trc}(P) \cap \mathrm{trc}(Q)$. Lemma 1, (9), states that

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

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

To prove "2 $\Rightarrow$ 1", assume $P \sqsubseteq_T Q$, but $P \not\sqsubseteq_F Q$. According to Lemma 1, (9), there exists $s \in \mathrm{trc}(P) \cap \mathrm{trc}(Q)$ such that

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

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

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

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

FiXme Fatal: alcc: minaccs is used for P above.

This result is used in the sequel in Section **??**.

3 Finite Complete Test Suites for CSP Failures Refinement

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

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

of nodes in N that are reachable in $G(P)$ by applying traces of V.

FiXme Fatal: alcc: can a normalised graph for a process have states that are unreachable?

Lemma 3. *Let P be a CSP process with normalised transition graph $G(P)$, such that all states in N are reachable. 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. □

FiXme Fatal: alcc: Explain the structure of the section? Can the above result be in a section, rather than the introduction of the section?

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.

$$U_F(p) = U_F(p, \varepsilon) \tag{14}$$

$$U_F(p, s) = \big(\ \Box\ e : (\Sigma - [P/s]^0) \bullet e \to \mathit{fail} \to \mathit{Stop}\big) \tag{15}$$

$$\Box$$

$$([P/s]^0 = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{16}$$

$$\Box$$

$$(\#s < p)\&\big(\ \Box\ e : [P/s]^0 \bullet e \to U_F(p, s.e)\big) \tag{17}$$

$$\Box$$

$$(\#s = p)\&\big(\ \sqcap_{H \in \mathrm{minHit}(P/s)}\ (\Box\ e : H \bullet e \to \mathit{pass} \to \mathit{Stop})\big) \tag{18}$$

FiXme Fatal: alcc: add an example?

A test is performed by running $U_F(p)$ concurrently with any SUT process Q that operates on the same alphabet as P, 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 $s \in \mathrm{trc}(P)$ 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 PASS event *pass*. This can also be expressed by means of a failures refinement.

$$Q \;\underline{\text{pass}}\; U_F(p) \equiv (\mathit{pass} \rightarrow \mathit{Stop}) \sqsubseteq_F (Q \,|[\,\Sigma\,]|\, U_F(p)) \setminus \Sigma$$

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 $\langle \mathit{pass} \rangle$ as the "Test Passed Process" $(\mathit{pass} \rightarrow \mathit{Stop})$. However, the former may nondeterministically refuse *pass*, due to a deadlock occurring when a faulty SUT process executes concurrently with $U_F(p, s)$ executing branch (18), 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 $s \in \mathrm{trc}(P)$ 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 (15) 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 (16) 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 (15). 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 (17). 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 $\mathrm{minAcc}(P/s)$ 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 2, a conforming SUT should accept at least one event of each minimal hitting set in $\mathrm{minHit}(P/s)$. Therefore, the test only terminates with success *pass*, if such an event is accepted by the SUT.

After this informal explanation of tests representing adaptive test cases, we are ready to prove 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 $\mathcal{F} = (P, \sqsubseteq_F, \mathcal{D})$.

Proof. To prove soundness of TS_F, suppose that $Q \in \mathcal{D} \wedge P \sqsubseteq_F Q$. In this case, $\mathrm{trc}(Q) \subseteq \mathrm{trc}(P)$, and Lemma 2 implies that for all traces s of Q, every H in $\mathrm{minHit}(P/s)$ is a hitting set for $\mathrm{minAcc}(Q/s)$. So, when running in parallel with Q, any adaptive test $U_F(p)$ will always enter the branches (16), (17), or (18) of the external choice construction for $U_F(p, s)$. Branch (16) leads always to a PASS verdict, and branch (17) 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. Note further, that this proof did not refer in any way to the size of Q's normalised transition graph, so the test suite is sound for *all* non-divergent CSP process refining P.

FiXme Fatal: alcc: Soundness does not depend on the fault domain. I would have it as a separate theorem.

To prove exhaustiveness, consider a process $Q \in \mathcal{D}$ with $P \not\sqsubseteq_F Q$. This non-conformance can be caused in two possible ways.

Case 1 $\mathrm{trc}(Q) \not\subseteq \mathrm{trc}(P)$

Case 2 There exists a joint trace $s \in \mathrm{trc}(Q) \cap \mathrm{trc}(P)$ and a minimal acceptance A_Q of $\mathrm{minAcc}(Q/s)$, such that (see Lemma 1, (9)).

$$\forall\, A_P \in \mathrm{minAcc}(P/s) : A_P \not\subseteq A_Q, \tag{19}$$

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 $pass \rightarrow Stop$ is no longer failures-refined by the test execution.

For the first case, consider a trace $s.e \in \mathrm{trc}(Q)$ such that $s \in \mathrm{trc}(P)$, but $s.e \notin \mathrm{trc}(P)$. 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. From the construction of G described there, we know that G has at most pq reachable states, because $G(P)$ has p states, and $G(Q)$ has at most q states. Suppose that $G/s = (n_P, n_Q)$. Applying Lemma 3 implies that this state can be reached by a trace $u \in \mathrm{trc}(G)$ 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 $s.e \notin \mathrm{trc}(P)$, e is an element of $\Sigma - [P/u]^0$. Therefore, in at least one execution, $U_F(pq-1, u)$

executes its first branch (15) with this event e, so that the test fails with event *fail*. Again, the assumption of non-divergence of Q is needed for this conclusion.

FiXme Fatal: alcc: I didn't see how Lemma 3 is being applied here. Also, given the definition of the test as a process, I'm not sure why you need to make the argument using graph product.

For the Case 2, we note that trace s is again a trace of the product graph G. Therefore, G/s can again be reached by a trace $u \in \mathrm{trc}(Q) \cap \mathrm{trc}(P)$ 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 (17) until the trace u has been completely processed. $U_F(\#u, u)$ may execute branches (15) or (18) only: assumption (19) 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 (16) evaluates to `false` for $U_F(\#u, u)$. Moreover, the guard condition $(\#s < p)$ for branch (17) evaluates to `false` for $U_F(\#u, u)$, too. If branch (15) is executed, the test always fails. If branch (18) 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 (19). The existence of such an H is guaranteed because of Lemma 2. 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 $p \geqslant 0$.

$$U_T(p) = U_T(p, \varepsilon) \tag{20}$$

$$U_T(p, s) = \big(e : (\Sigma - [P/s]^0) \to \mathit{fail} \to \mathit{Stop}\big) \tag{21}$$

$$\Box$$

$$([P/s]^0 = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{22}$$

$$\Box$$

$$(\#s < p)\&\big(e : [P/s]^0 \to U_T(p, s.e)\big) \tag{23}$$

$$\Box$$

$$(\#s = p)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{24}$$

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 consists in the fact that the complete suite for trace refinement just needs the single adaptive test case $U_T(pq - 1)$, while the complete failures test suite 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 the reference process has run through. Instead, we calculate the following auxiliary functions from P's transition graph.

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

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). Then, using the transition function of t in addition to the two auxiliary functions, $U_F(p)$ can be re-written as the failures-equivalent CSP process

$$U_F^1(p) = U_F^1(p, 0, \underline{n}) \tag{25}$$

$$U_F^1(p,k,n) = \big(e : (\Sigma - \text{initials}(n)) \to \mathit{fail} \to \mathit{Stop}\big) \tag{26}$$

$$\Box$$

$$(\text{initials}(n) = \varnothing)\&\big(\mathit{pass} \to \mathit{Stop}\big) \tag{27}$$

$$\Box$$

$$(k < p)\&\big(e : \text{initials}(n) \to U_F^1(p, (k+1), t(n,e))\big) \tag{28}$$

$$\Box$$

$$(k = p)\&\big(\sqcap_{H \in \text{minHit}(n)}\ (e : H \to \mathit{pass} \to \mathit{Stop})\big) \tag{29}$$

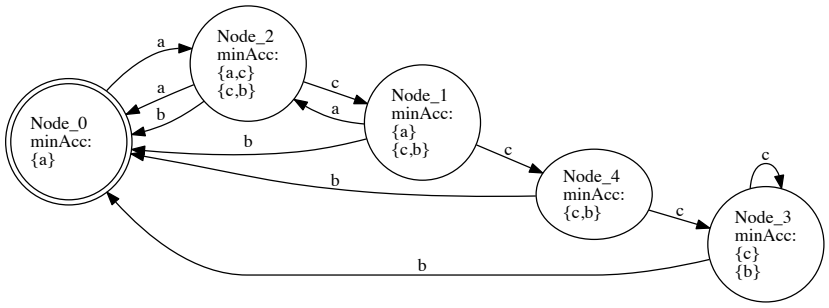

Fig. 2. Normalised transition graph of faulty implementation Z from Example 2.

Example 2. Consider the following erroneous implementation Z of process P from Example 1.

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

It is easy to see (and can be checked with FDR4) that Z is trace-equivalent to P. While $k < r_{max}$, process Z also accepts the same sets of events as P. If, however, sub-process $R_1(r_{max}, k)$ runs through several recursions until condition $k = r_{max}$ is fulfilled, the process uses internal choice instead of external choice, and so this process state no longer refines the corresponding process state R of the reference process P in the failures model. For $r_{max} = 3$, the normalised transition graph of Z is displayed in Fig. 2.

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 only accepts $\{b\}$ due to internal choice and $U_F^1(4)$ – also due to internal choice – only accepts the minimal hitting set $\{c\}$ in union with the event $a \in (\Sigma - [P/s]^0)$. As a consequence, $(Z \,|[\,\Sigma\,]|\, U_F^1(4))/s$ deadlocks for this execution, and the pass-event cannot be produced. Another failing execution would be the situation where Z/s choses 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

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. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
7. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Jounal 19, 53–77 (1997)
8. 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
9. Roscoe, A.W. (ed.): A Classical Mind: Essays in Honour of C. A. R. Hoare. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)
10. Roscoe, A.W.: Understanding Concurrent Systems. Springer, London, Dordrecht Heidelberg, New York (2010)
11. 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