# 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. The focus of these contributions is on the generation of complete, finite test suites. Test suites are complete if they can guarantee to uncover every conformance violation of the system under test (SUT) 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 trace refinement, as well as failures equivalence and failures refinement. Complete black-box test suites 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 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, Complete testing theories, CSP, Refinement

## 1 Introduction

**Motivation** Model-based testing (MBT) is an active research field which is currently evaluated and integrated into industrial verification processes by many companies. This holds particularly for the embedded and cyber-physical systems domain. While MBT is applied in different flavours, we consider the variant to be the most effective 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: it guarantees the maximal return of investment for the time and effort invested into creating 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 systems domain, test suites with guaranteed fault coverage are of particular interest. For black-box testing, these guarantees can only be given under certain hypotheses. These hypotheses are usually identified by specifying a *fault domain*; this is 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 has been 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)* [5, 15]; this is a mature process-algebraic approach which has been shown to be well-suited for the description of reactive control systems in many publications over almost 5 decades.

**Contributions** This paper complements work published by two of the authors in [1]. There, fault domains have been specified as collections of processes refining a "most general" fault domain member. With this concept of fault domains, complete test suites may be finite or infinite. While this gives important insight into the theory of complete test suites, we are particularly interested in finite suites when it comes to their practical application.

Therefore, we present a complementary approach to the definition of CSP fault domains in this paper. To this end, 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 [14]. The maximal refusals express the degree of nondeterminism that is present in a given CSP process state which 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 here 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 give constant.

Our main contributions in this paper are as follows.

1. It is proven that for fault domains of the described type, complete test suite generation methods can be given for verifying (1) Trace equivalence, (2) trace refinement, (3) failures equivalence, and (4) failures refinement.
2. We prove that finite complete test suites can be generated in all 4 cases, when using the fault domains based on the size of the members' normalised transition graphs.
3. We present test suite generation techniques for each of the 4 conformance relations by translating algorithms originally elaborated for the FSM domain into the CSP world. This translation preserves the completeness properties that have previously been established for the FSM domain by other authors.

The possibility to translate complete test suites between different formalisms (here FSMs and CSP processes) has been investigated before by two of the authors [6].

**Overview** @todo

## 2 Preliminaries

### 2.1 Complete Test Suites

We use the term *signature* to denote a collection of comparable models represented in an arbitrary formalism. In this article, signatures represent sets of finite state machines over fixed input and output alphabets, or CSP processes with finite state, represented by their normalised transition graphs (see Section 2.3).

Given a signature $Sig$ of models, a *fault model* $\mathcal{F} = (M, \leqslant, Dom)$ specifies a *reference model* $M \in Sig$, a *conformance relation* $\leqslant \subseteq Sig \times Sig$ between models, and a *fault domain* $Dom \subseteq Sig$. This terminology follows [12], where fault models were originally introduced in the context of finite state machine testing. Note that fault domains may contain both models conforming to the reference model and models violating the conformance relation. Note further that the reference model $M$ is not necessarily a member of the fault domain. For example, $M$ could be nondeterministic, while only deterministic implementation behaviours might be considered in the fault domain.

Let $\mathrm{TC}(Sig)$ denote the set of all *test cases* applicable to elements of $Sig$. The abstract notion of test cases defined here only requires the existence of a relation $\underline{\text{pass}} \subseteq Sig \times \mathrm{TC}(Sig)$. For $(M, U) \in \underline{\text{pass}}$, the infix notation $M \ \underline{\text{pass}} \ U$ is used, and interpreted as *'Model $M$ passes the test case $U$'*. If $(M, U) \notin \underline{\text{pass}}$ holds, this is abbreviated by $M \ \underline{\text{fail}} \ U$.

A *test suite* $\mathrm{TS} \subseteq \mathrm{TC}(Sig)$ denotes a set of test cases. A model $M$ *passes the test suite* TS, also written as $M \ \underline{\text{pass}}$ TS, if and only if $M \ \underline{\text{pass}} \ U$ for all $U \in \mathrm{TS}$. A test suite TS is called *complete* for fault model $\mathcal{F} = (M, \leqslant, Dom)$, if and only if the following properties hold.

1. If a member $M'$ of the fault domain conforms to the reference model $M$, it passes the test suite, that is,

$$\forall\, M' \in Dom : M' \leqslant M \Rightarrow M' \ \underline{\text{pass}} \ \mathrm{TS}$$

   This property is usually called *soundness* of the test suite.
2. If a member of the fault domain passes the test suite, it conforms to the reference model, that is,

$$\forall\, M' \in Dom : M' \ \underline{\text{pass}} \ \mathrm{TS} \Rightarrow M' \leqslant M$$

   This property is usually called *exhaustiveness*.

A test suite TS is *finite* if it contains finitely many test cases and every test case $U \in \mathrm{TS}$ is finite in the sense that it terminates after a finite number of steps. It is trivial to see that, if TS is complete for $\mathcal{F} = (M, \leqslant, Dom)$ and $Dom' \subseteq Dom$, then TS is also complete for $\mathcal{F}' = (M, \leqslant, Dom')$.

### 2.2 Translation of Test Cases and Test Suites

Let $Sig_1$ and $Sig_2$ be two signatures with conformance relations $\leqslant_1$ and $\leqslant_2$, and test case relations $\underline{\text{pass}}_1$ and $\underline{\text{pass}}_2$, respectively. A function $T : \underline{Sig_1} \to Sig_2$ defined on a sub-domain $\underline{Sig_1} \subseteq Sig_1$ is called a *model map*, and a function $T^* : \text{TC}(Sig_2) \to \text{TC}(Sig_1)$ is called a *test case map*. Note that models and test cases are mapped in opposite directions (see Fig. 1). The pair $(T, T^*)$ fulfils the *satisfaction condition* if and only if the following conditions **SC1** and **SC2** are fulfilled.

**SC1** The model map is compatible with the conformance relations under consideration, in the sense that

$$\forall \mathcal{S}, \mathcal{S}' \in \underline{Sig_1} : \mathcal{S}' \leqslant_1 \mathcal{S} \Leftrightarrow T(\mathcal{S}') \leqslant_2 T(\mathcal{S}),$$

so the left-hand side diagram in Fig. 1 commutes due to the fact that $T;\ \leqslant_2 = \leqslant_1;\ T$.[4]

**SC2** Model map and test case map preserve the $\underline{\text{pass}}$-relationship in the sense that

$$\forall \mathcal{S} \in \underline{Sig_1}, U \in \text{TC}(Sig_2) : T(\mathcal{S})\ \underline{\text{pass}}_2\ U \Leftrightarrow \mathcal{S}\ \underline{\text{pass}}_1\ T^*(U),$$

so the right-hand side diagram in Fig. 1 commutes, due to the fact that $\underline{\text{pass}}_1 = T;\ \underline{\text{pass}}_2;\ T^*$.

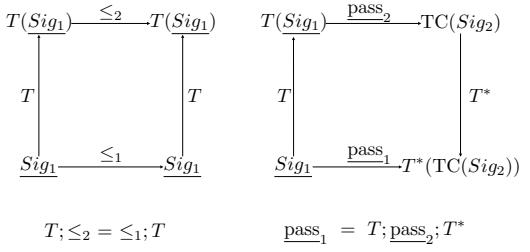


**Fig. 1.** Commuting diagrams reflecting the satisfaction condition.

The following theorem is a direct consequence of [6, Theorem 2.1].

**Theorem 1.** *With the notation introduced above, let* $(T, T^*)$ *fulfil the satisfaction condition. Suppose that* $TS_2 \subseteq TC(Sig_2)$ *is a complete test suite for fault model* $\mathcal{F}_2 = (\mathcal{S}_2, \leqslant_2, Dom_2)$*. Define fault model* $\mathcal{F}_1$ *on* $\underline{Sig}_1$ *by*

$$\mathcal{F}_1 = (\mathcal{S}_1, \leqslant_1, Dom_1),\ \textit{such that}\ T(\mathcal{S}_1) = \mathcal{S}_2\ \textit{and}\ Dom_1 = \{\mathcal{S} \mid T(\mathcal{S}) \in Dom_2\}.$$

[4] Operator ";" denotes the relational composition defined for functions or relations $f \subseteq A \times B$, $g \subseteq B \times C$ by $f;\ g = \{(a, c) \in A \times C \mid \exists\, b \in B : (a, b) \in f \wedge (b, c) \in g\}$. Note that $f;\ g$ is evaluated from left to right (like composition of code fragments), as opposed to right-to-left evaluation which is usually denoted by $g \circ f$.

*Then*

$$TS_1 = T^*(TS_2)$$

*is a complete test suite with respect to fault model* $\mathcal{F}_1$. □

### 2.3 CSP and Refinement

**Communicating Sequential Processes** @todo

**Normalised Transition Graphs for CSP Processes** As shown in [14], 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 $\Sigma$. The partial *transition function* $t$ maps a node $n$ and an event $e \in \Sigma$ to its successor node $t(n, e)$, if and only if $(n, e)$ are in the domain of $t$. 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 $\mathrm{tr}(P)$. If $s \in \mathrm{tr}(P)$, 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 to be reached by applying the events from $s$ one by one, starting in $\underline{n}$. Therefore, the notation $G(P)/s$ is also well-defined.

By $[n]^0$ we denote the *fan-out* of $n$, that is, the set of events occurring as transition 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.

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 CSP process $P$ might refuse to engage into, when in a process state corresponding to $n$. The number of refusal sets contained in $\mathrm{Ref}(P/s)$ specifies the degree of nondeterminism which is present in process state $P/s$: the more refusal sets contained in $\mathrm{Ref}(P/s)$, the more nondeterministic is the behaviour in state $P/s$. If $P/s$ is deterministic, its refusals coincide with the set of subsets of $\Sigma - [P/s]^0$, including the empty set. Since the refusals of each process state are subset-closed [5, 15], $\mathrm{Ref}(P/s)$ can be reconstructed by knowing the set of *maximal refusals* $\mathrm{maxRef}(P/s) \subseteq \mathrm{Ref}(P/s)$ (we also write $\mathrm{maxRef}(n) \subseteq r(n)$ using transition graph nodes instead of process states) and then building the subset closure. More formally, the maximal refusals are defined as

$$\mathrm{maxRef}(n) = \{R \in \mathrm{Ref}(n) \mid \forall\, R' \in r(n) - \{R\} : R \not\subseteq R'\}$$

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

$$\forall\, n \in N, e \in \Sigma : (n, e) \in \mathrm{dom}\ t \Rightarrow \exists\, R \in \mathrm{maxRef}(n) : e \notin R \tag{1}$$

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

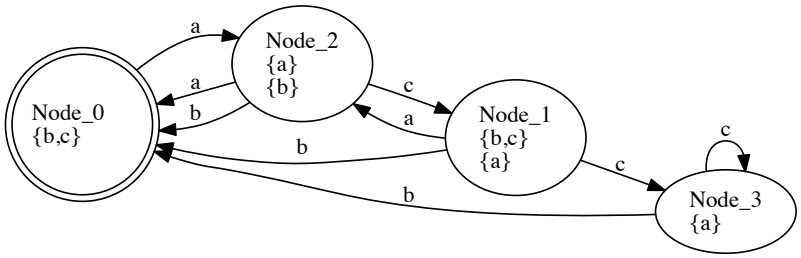


**Fig. 2.** Normalised transition graph of CSP process $P$ from Example 1.

*Example 1.* Consider CSP process

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

Its transition graph $G(P)$ is shown in Fig. 2. Process state $P$ is represented there as Node_0, with $\{b, c\}$ as the only maximal refusal, since event $a$ 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 refusal sets derived from $Q$ and $R$. Since these processes accept their initial events with external choice, process $Q \sqcap R$ induces just two maximal refusal sets $\{b\}$ and $\{a\}$. Note that event $c$ can never be refused, since it is not a member of any maximal refusal.

Having engaged into $c$, the next process state is represented by Node_1. Due to normalisation, there was 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 maximal refusals labelling Node_1. □

**Tool Considerations** The FDR tool [3] supports model checking and semantic analyses of CSP processes. It provides an API [2] that can be used to construct

normalised transition graphs for CSP processes. The graph nodes are labelled by *minimal acceptances*. 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 creating the complements of all minimal acceptances and then building all subsets of these complements. 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.

### 2.4 Finite State Machines

To make this paper sufficiently self-contained, we introduce definitions, notation, and facts about finite state machines (FSMs) that have been originally described in contributions on FSM testing, such as [11, 13, 4].

A *Finite State Machine (FSM)* is a tuple $M = (Q, \underline{q}, \Sigma_I, \Sigma_O, h)$ with state space $Q$, input alphabet $\Sigma_I$, output alphabet $\Sigma_O$, where $Q, \Sigma_I, \Sigma_O$ are finite and nonempty sets. $\underline{q} \in Q$ denotes the initial state. $h \subseteq Q \times \Sigma_I \times \Sigma_O \times Q$ is the transition relation, $(q, x, y, q') \in h$ if and only if there is a transition from $q$ to $q'$ with input $x$ and output $y$. We use both set notation $(q, x, y, q') \in h$ and Boolean notation $h(q, x, y, q')$ for specifying that $(q, x, y, q')$ is a transition in $h$. We call $x$ a *defined* input in state $q$, if there is a transition from $q$ with input $x$. If every input of $\Sigma_I$ is defined in every state, $M$ is *completely specified*. If in every state $q$ and for every output $y \in \Sigma_O$, and input $x$ a post-state $q'$ satisfying $h(q, x, y, q')$ exists, the FSM is called *output complete*.

FSM $M$ is called a *deterministic FSM (DFSM)*, if for any state $q$ and defined input $x$, $h(q, x, y, q') \wedge h(q, x, y', q'')$ implies $(y, q') = (y', q'')$. Intuitively speaking, a specific input applied to a specific state uniquely determines both post-state and associated output. If $M$ is not deterministic, it is called a *nondeterministic FSM (NFSM)*. If there is no emanating transition for $q \in Q$, this state is called a *deadlock state*, and $M$ *terminates in* $q$. The set of deadlock states is denoted by $\mathit{deadlock}(Q) \subseteq Q$. The set of states that do not deadlock is denoted by $\mathrm{DF}(Q) = \{q \in Q \mid \exists (q', x, y, q'') \in h : q' = q\}$.

The transition relation $h$ can be extended in a natural way to input traces: let $\overline{x}$ be an input trace and $\overline{y}$ an output trace. Then $(q, \overline{x}, \overline{y}, q') \in h$, if and only if there is a transition sequence from $q$ to $q'$ with input trace $\overline{x}$ and output trace $\overline{y}$. If $q$ is the initial state $\underline{q}$, such a transition sequence is called an *execution* of $M$. Executions are written in the notation

$$q_0 \xrightarrow{x_1/y_1} q_1 \xrightarrow{x_2/y_2} \ldots \xrightarrow{x_k/y_k} q_k$$

with $q_0 = \underline{q}$, $h(q_{i-1}, x_i, y_i, q_i)$ for $i = 1, \ldots, k$, and $\overline{x} = x_1 \ldots x_k$ and $\overline{y} = y_1 \ldots y_k$.

The empty trace is denoted by $\varepsilon$, and $(q, \varepsilon, \varepsilon, q) \in h$, for any state $q$. A *language* of an FSM $M$ is the set consisting of all possible input/output traces in $M$; we use notation $L_M(q) = \{\overline{x}/\overline{y} \mid \exists\, q' \in Q : h(q, \overline{x}, \overline{y}, q')\}$ for $q \in Q$, and $L(M) = L_M(\underline{q})$. By $\mathrm{FSM}(\Sigma_I, \Sigma_O)$ we denote the set of all FSMs with input alphabet $\Sigma_I$ and output alphabet $\Sigma_O$.

An FSM $M$ is called *observable* if in every state $q$, every existing post-state $q'$ is uniquely determined by the I/O pair $x/y$ satisfying $h(q, x, y, q')$. For observable state machines, the partial function

$$\_\text{-after-}\_/\_ : Q \times \Sigma_I \times \Sigma_O \nrightarrow Q; \quad q\text{-after-}(x/y) = q' \Leftrightarrow h(q, x, y, q')$$

is well-defined. Again, it can be extended to I/O-traces $\overline{x}/\overline{y}$ by repetitive application. Deterministic FSMs are always observable. Every non-observable FSM can be transformed into an observable one that has the same language [9]. The algorithm operates in analogy to the algorithm normalising transition graphs of CSP processes.

Two FSM $M_1, M_2$ are *I/O-equivalent* ($M_1 \sim M_2$) if and only if their languages coincide, i.e. $L(M_1) = L(M_2)$. FSM $M_1$ is a *reduction of* $M_2$ ($M_1 \preceq M_2$), if and only if $L(M_1) \subseteq L(M_2)$. I/O-equivalence is also called *trace equivalence* by some authors, see, e.g. [7].

FSMs can be composed in parallel by synchronising over common input/output events: Let FSMs $M_i = (Q_i, \underline{q_i}, \Sigma_I, \Sigma_O, h_i), i = 1, 2$ be defined over the same input/output alphabets. Then

$$M_1 \cap M_2 = (Q_1 \times Q_2, (\underline{q_1}, \underline{q_2}), \Sigma_I, \Sigma_O, h)$$

where the transition relation is specified by

$$h((q_1, q_2), x, y, (q_1', q_2')) \Leftrightarrow h_1(q_1, x, y, q_1') \wedge h_2(q_2, x, y, q_2')$$

By construction, $L(M_1 \cap M_2) = L(M_1) \cap L(M_2)$. Every execution

$$(\underline{q_1}, \underline{q_2}) \xrightarrow{x_1/y_1} (q_1^1, q_2^1) \xrightarrow{x_2/y_2} \ldots \xrightarrow{x_k/y_k} (q_1^k, q_2^k)$$

of $M_1 \cap M_2$ is composed of executions

$\underline{q_1} \xrightarrow{x_1/y_1} q_1^1 \xrightarrow{x_2/y_2} \ldots \xrightarrow{x_k/y_k} q_1^k$ of $M_1$ and $\underline{q_2} \xrightarrow{x_1/y_1} q_2^1 \xrightarrow{x_2/y_2} \ldots \xrightarrow{x_k/y_k} q_2^k$ of $M_2$

## 3 Finite Complete Test Suites for CSP

### 3.1 A Model Map from CSP Processes to Finite State Machines

We will now construct a model map for associating CSP processes represented by normalised transition graphs to finite state machines. The intuition behind this construction is that the finite state machine's input alphabet corresponds to *sets of inputs* that may be offered to a CSP process. Depending on the events contained in this set, the process may (1) accept all of them, (2) accept some of them while refusing others, and (3) refuse all of them. This is reflected in the FSM by outputs representing events that the process really has engaged in and an extra event $\perp$ representing refusal, if the set of events has been blocked. Blocked sets of events are always associated with self-loop transitions: the state

is not changed, because the corresponding CSP process also remains blocked in its current state.

More formally, we fix a finite CSP process alphabet $\Sigma$ and consider a finite-state process $P$ over this alphabet with normalised transition graph $G(P) = (N, \underline{n}, \Sigma, t : N \times \Sigma \nrightarrow N, r : N \to \mathbb{PP}(\Sigma))$. Then the model map $T$ maps $P$ to the following observable FSM $T(P) = (Q, \underline{q}, \Sigma_I, \Sigma_O, h)$ specified by

$$
\begin{aligned}
Q &= N \\
\underline{q} &= \underline{n} \\
\Sigma_I &= \mathbb{P}(\Sigma) - \{\varnothing\} \\
\Sigma_O &= \Sigma \cup \{\bot\} \\
h &= \{(n, A, e, n') \mid A \in \Sigma_I \wedge e \in A \wedge (n, e) \in \text{dom } t \wedge t(n, e) = n'\} \cup \\
&\quad \{(n, A, \bot, n) \mid A \in r(n) - \{\varnothing\}\}
\end{aligned}
$$

We say that FSM trace $(x/s) \in L(T(P))$ and CSP trace $s' \in \text{tr}(P)$ are *corresponding traces*, if $s' = s \upharpoonright \Sigma$. Observe that the FSM output trace $s$ may contain deadlock events $\bot$ that are not contained in the process alphabet $\Sigma$.

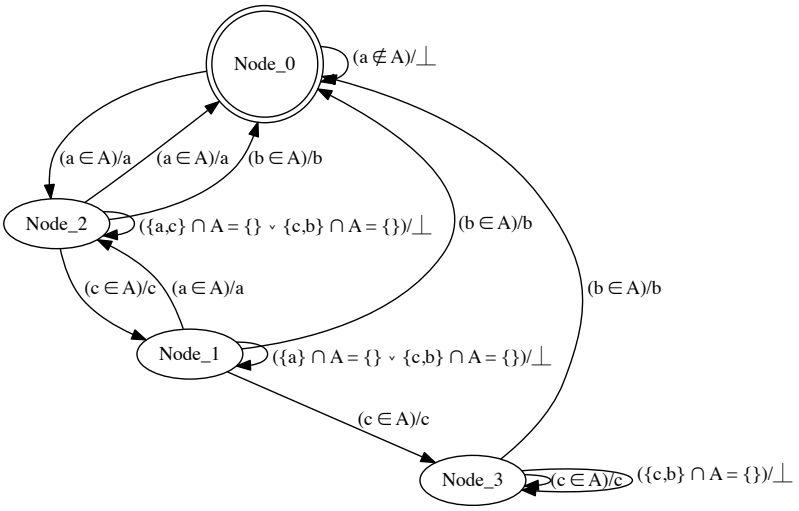


**Fig. 3.** FSM resulting from applying the model map to CSP process $P$ from Example 1.

*Example 2.* For the CSP process $P$ and its transition graph $G(P)$ discussed in Example 1, the FSM $T(P)$ is depicted in Fig. 3. For displaying its transitions,

we used notation

$$(\text{condition}(A))/e$$

which stands for a set of transitions between the respective nodes: one transition per non-empty set $A \subseteq \Sigma$ fulfilling the specified condition. The arrow

$$\text{Node\_0} \xrightarrow{(a \in A)/a} \text{Node\_2},$$

for example, stands for FSM transitions

$$\begin{aligned}
&\text{Node_0} \xrightarrow{\{a\}/a} \text{Node_2}\\
&\text{Node_0} \xrightarrow{\{a,b\}/a} \text{Node_2}\\
&\text{Node_0} \xrightarrow{\{a,c\}/a} \text{Node_2}\\
&\text{Node_0} \xrightarrow{\{a,b,c\}/a} \text{Node_2}
\end{aligned}$$

□

We are now in the position to state and prove the theorem about the model map fulfilling the satisfaction condition **SC1** introduced in Section 2.2. To this end, we first introduce five lemmas.

**Lemma 1.** *Let $s \in tr(P)$ and $n = G(P)/s$ be the node of $G(P)$ corresponding the process state $P/s$. Then for any $x \in \Sigma_I$, $y \in \Sigma_O$ :*

$$\begin{aligned}
(n, x, y, n') \in h \Leftrightarrow &(y \in x \wedge s.y \in tr(P) \wedge n' = G(P)/s.y) \vee\\
&(y = \bot \wedge (s, x) \in \text{Fail}(P) \wedge n' = n)
\end{aligned}$$

*Proof.* Let $s' = s.y \restriction \Sigma$. We consider the following two cases:

1. Suppose $y = e$ for some $e \in \Sigma$. Then $s' = s.e$. By definition of $h$, we have

$$\begin{aligned}
(n, x, e, n') \in h &\Leftrightarrow e \in x \wedge (n, e) \in \text{dom}\, t \wedge n' = t(n, e)\\
&\Leftrightarrow e \in x \wedge s.e \in \text{tr}(P) \wedge n' = G(P)/s.e
\end{aligned}$$

2. Suppose $y = \bot$. Then $s' = s$, and we can derive

$$\begin{aligned}
(n, x, \bot, n') \in h &\Leftrightarrow x \in \text{Ref}(P/s) \wedge n' = n\\
&\Leftrightarrow (s, x) \in \text{Fail}(P) \wedge n' = n
\end{aligned}$$

□

**Lemma 2.** *Let $x/y \in (\Sigma_I \times \Sigma_O)^*$ and $s = y \restriction \Sigma$. Then*

$$x/y \in L(T(P)) \Rightarrow s \in tr(P) \wedge \underline{q}\text{-}after\text{-}x/y = G(P)/s.$$

*Proof.* We prove the lemma by induction on $\#(x/y)$, the length of $x/y$. For the base case, suppose that $x/y = \varepsilon$, then $s = \varepsilon$ and $\underline{q}$-after-$x/y = \underline{n} = G(P)/\varepsilon$. For the induction hypothesis, suppose the statement holds for all $x/y \in (\Sigma_I \times \Sigma_O)^*$ with $\#(x/y) = k$, for some $k \geqslant 0$. Let $x/y \in (\Sigma_I \times \Sigma_O)^*$ with $\#(x/y) = k$ and $x'/y' \in \Sigma \times \Sigma_O$. To perform the induction step, let $y \upharpoonright \Sigma = s$ and $y.y' \upharpoonright \Sigma = s'$. Then by the induction hypothesis we have $s \in \mathrm{tr}(P)$ and $\underline{q}$-after-$x/y = G(P)/s$. Since $x.x'/y.y' \in L(T(P))$, there is a transition $(G(P)/s', x', y', n) \in h$ and $n = \underline{q}$-after-$x/y$. By Lemma 1 we have $s' = s.y' \in \mathrm{tr}(P) \wedge \underline{q}$-after-$x.x'/y.y' = G(P)/s'$. □

**Lemma 3.** *Let $x/y \in (\Sigma_I \times \Sigma_O)^*$ and $x/y \in L(T(P))$. Let $x'/y' \in \Sigma_I \times \Sigma_O$. Then*

$$x.x'/y.y' \in L(T(P)) \Leftrightarrow \big(y' \in x' \wedge (y.y') \upharpoonright \Sigma \in tr(P)\big) \vee \big(y' = \bot \wedge (y \upharpoonright \Sigma, x') \in \mathrm{Fail}(P)\big).$$

*Proof.* Let $x/y \in (\Sigma_I \times \Sigma_O)^*$ and $x/y \in L(T(P))$. Let $x'/y' \in \Sigma_I \times \Sigma_O$. Then by Lemma 2, we have $s = y \upharpoonright \Sigma \in \mathrm{tr}(P)$ and $G(P)/s = \underline{q}$-after-$x/y$. From Lemma 1 we have the following:

$$\begin{aligned}
& x.x'/y.y' \in L(T(P)) \\
\Leftrightarrow\ & \exists\, n' \in Q : (G(P)/s, x', y', n') \in h \\
\Leftrightarrow\ & (y' \in x' \wedge s.y' \in \mathrm{tr}(P)) \vee (y' = \bot \wedge (s, x') \in \mathrm{Fail}(P)\big) \\
\Leftrightarrow\ & (y' \in x' \wedge (y.y') \upharpoonright \Sigma \in \mathrm{tr}(P)\big) \vee \big(y' = \bot \wedge (y \upharpoonright \Sigma, x) \in \mathrm{Fail}(P)\big)
\end{aligned}$$

**Lemma 4.** *For any $s \in \Sigma^*$,*

$$s \in tr(P) \Leftrightarrow \exists\, x \in \Sigma_I^* : x/s \in L(T(P)).$$

*Proof.* Let $s \in \Sigma^*$. We prove the lemma by induction on $\#s$ the length of $s$. Suppose $s = \varepsilon$, then $s \in \mathrm{tr}(P) \wedge x = \varepsilon \Leftrightarrow x/s \in L(T(P))$. Suppose the statement holds for all $s \in \Sigma^*$ with $\#s = k$, for some $k \geqslant 0$. Let $s \in \Sigma^*$ with $\#s = k$ and $e \in \Sigma$. Then

$$\begin{aligned}
& s.e \in \mathrm{tr}(P) \\
\Leftrightarrow\ & s \in \mathrm{tr}(P) \wedge s.e \in \mathrm{tr}(P) \\
\Leftrightarrow\ & \exists\, x \in \Sigma_I^* : x/s \in L(T(P)) \wedge s.e \in \mathrm{tr}(P) \\
\Rightarrow\ & \exists\, x \in \Sigma_I^* : x.\{e\}/s.e \in L(T(P)) && \text{[Lemma 3]} \\
\Rightarrow\ & \exists\, x \in \Sigma_I^* : x/s.e \in L(T(P))
\end{aligned}$$

$$\begin{aligned}
& \exists\, x \in \Sigma_I^* : x/s.e \in L(T(P)) \\
\Leftrightarrow\ & \exists\, x \in \Sigma_I^* \wedge \exists\, x' \in \Sigma_I : x/s \in L(T(P)) \wedge x.x'/s.e \in L(T(P)) \\
\Rightarrow\ & s.e \in \mathrm{tr}(P) && \text{[Lemma 3]}
\end{aligned}$$

□

**Lemma 5.** *For any $s \in \Sigma^*$ and $x' \in \Sigma_I$,*

$$(s, x') \in \text{Fail}(P) \Leftrightarrow \exists\, x \in \Sigma_I^* : x.x'/s.\bot \in L(T(P))$$

*Proof.* $\forall\, s \in \Sigma^*, x' \in \Sigma_I$:

$$\begin{array}{ll} & \exists\, x \in \Sigma_I^* : x.x'/s.\bot \in L(T(P)) \\ \Leftrightarrow & \exists\, x \in \Sigma_I^* : x/s \in L(T(P)) \wedge x.x'/s.\bot \in L(T(P)) \\ \Leftrightarrow & (s, x') \in \text{Fail}(P) \qquad \text{[Lemma 3 and 4]} \end{array}$$

□

**Theorem 2.** *Let $P, Q$ be two CSP processes over the same alphabet $\Sigma$. Then $T(Q) \preceq T(P) \Leftrightarrow P \sqsubseteq_F Q$*

*Proof.* Suppose $T(Q) \preceq T(P)$. Let $s \in \Sigma^*, x' \in \Sigma_I$. Then by Lemma 4 and Lemma 5

$$\begin{array}{rl} s \in \text{tr}(Q) & \Leftrightarrow \exists\, x \in \Sigma_I^* : x/s \in L(T(Q)) \\ & \Rightarrow \exists\, x \in \Sigma_I^* : x/s \in L(T(P)) \\ & \Leftrightarrow s \in \text{tr}(P) \end{array}$$

$$\begin{array}{rl} (s, x') \in \text{Fail}(Q) & \Leftrightarrow \exists\, x \in \Sigma_I^* : x.x'/s.\bot \in L(T(Q)) \\ & \Rightarrow \exists\, x \in \Sigma_I^* : x.x'/s.\bot \in L(T(P)) \\ & \Leftrightarrow (s, x') \in \text{Fail}(P) \end{array}$$

Hence

$$T(Q) \preceq T(P) \Rightarrow P \sqsubseteq_F Q$$

Now suppose $P \sqsubseteq_F Q$. We prove by induction on $\#(x/y)$ that for any $x/y \in (\Sigma_I \times \Sigma_O)^*$, $x/y \in L(T(Q)) \Rightarrow x/y \in L(T(P))$ holds. It is trivial if $\#(x/y) = 0$. Suppose $x/y \in L(T(Q)) \Rightarrow x/y \in L(T(P))$ holds for any $x/y \in (\Sigma_I \times \Sigma_O)^*$ of length $k \geqslant 0$. For any $x/y \in (\Sigma_I \times \Sigma_O)^*$ of length $k$ and for any $x'/y' \in \Sigma_I \times \Sigma_O$. Let $s = y \upharpoonright \Sigma$ and $s' = (y.y') \upharpoonright \Sigma$. Then by Lemma 3 and $P \sqsubseteq_F Q$ we have

$$\begin{array}{ll} & x.x'/y.y' \in L(T(Q)) \\ \Rightarrow & x/y \in L(T(Q)) \wedge \big((y' \in x' \wedge s' \in \text{tr}(Q)) \vee (y' = \bot \wedge (s, x) \in \text{Fail}(Q))\big) \\ \Rightarrow & x/y \in L(T(P)) \wedge \big((y' \in x' \wedge s' \in \text{tr}(P)\big) \vee (y' = \bot \wedge (s, x) \in \text{Fail}(P))\big) \\ \Rightarrow & x.x'/y.y' \in L(T(P)) \end{array}$$

Hence

$$P \sqsubseteq_F Q \Rightarrow T(Q) \preceq T(P)$$

. □

### 3.2 A Test Case Map from Finite State Machines to CSP Processes

**FSM Test Cases** Following [13], an *adaptive FSM test case*

$$tc_{\mathrm{FSM}} = (Q, \underline{q}, \Sigma_I, \Sigma_O, h, in)$$

is a nondeterministic, observable, output-complete, acyclic FSM which only provides a single input in a given state. Running in FSM intersection mode with the SUT, the test case provides a specific input to the SUT; this input is determined by the current state of the test case. It accepts every output and transits either to a fail-state FAIL, if the output is wrong according to the test objectives, or to the next test state uniquely determined by the processed input/output pair. Another state PASS indicates that the test has been completed without failure. Both FAIL and PASS are termination states, that is, they do not have any outgoing transitions.

Since the test case state determines the input for all of its outgoing transitions, this input is typically used as a state label, and the outgoing transitions are just labelled by the possible outputs. A function $in : Q - \{\mathrm{PASS}, \mathrm{FAIL}\} \to \Sigma_I$ maps the states to these inputs. Termination states of the FSM are not labelled with further inputs.

There is no requirement that an FSM test case running in intersection mode against an FSM $M$ acting as SUT should *always* reach the PASS state. Since the SUT may be nondeterministic, it may perform a joint test execution that blocks before the test case's PASS state is reached. In such a case, the result of the test execution is *inconclusive*. Due to nondeterminism, for $M$ to pass a test case, it has to checked that *every possible* test execution of $tc_{\mathrm{FSM}}$ against $M$ terminates either in PASS, or that the result remains inconclusive. If one execution ends in FAIL, the test verdict is FAIL. More formally,

$$M \ \underline{\mathrm{pass}} \ tc_{\mathrm{FSM}} \equiv \big(\forall\, x/y \in L(M) \cap L(tc_{\mathrm{FSM}}) : \underline{q}\text{-after-}(x/y) \neq \mathit{FAIL}\big).$$

*Example 3.* Consider the FSM test case depicted in Fig. 4 which is specified for the same input and output alphabets as defined for the FSM presented in Example 2. The test case is passed by the FSM from Example 2, because intersecting the two state machines results in an FSM which always reaches the PASS state. □

**CSP Test Cases** A *CSP test case* $tc_{\mathrm{CSP}}$ is a terminating process with alphabet $\Sigma \cup \{\dagger, \bot, \checkmark\}$, where the extra events stand for (1) test verdict FAIL ($\dagger$), (2) timeout ($\bot$), and (3) test verdict PASS ($\checkmark$). The test case runs in parallel with the SUT $P$, synchronising over all events from the visible alphabet $\Sigma$ of $P$. This is expressed by the formula

$$P \,|[\, \Sigma \,]|\, tc_{\mathrm{CSP}}.$$

In analogy to FSM test cases, a CSP process $P$ passes a test case $tc_{\mathrm{CSP}}$ if the traces of the parallel composition do not contain the failure event, that is,

$$P \ \underline{\mathrm{pass}} \ tc_{\mathrm{CSP}} \equiv \big(\forall\, s \in \mathrm{tr}(P \,|[\, \Sigma \,]|\, tc_{\mathrm{CSP}}) : (s \upharpoonright \{\dagger\}) = \varepsilon\big)$$

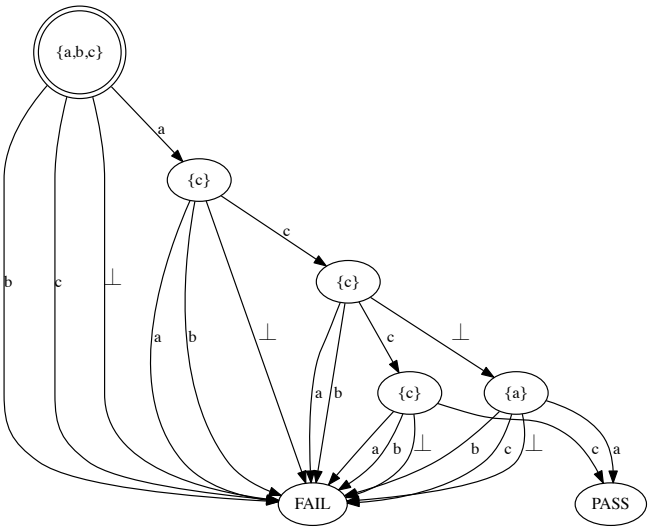


**Fig. 4.** An FSM test case which is passed by the FSM presented in Example 2.

In principle, very general classes of CSP processes can be used for testing, as introduced, for example, in [10, 8]. For the purpose of this paper, however, we can restrict the possible variants of CSP test cases to the ones that are in the range of the test case map which is constructed next.

**Test Case Map** The test case map $T^* : TC(FSM) \to TC(CSP)$ is specified with respect to a fixed CSP process alphabet $\Sigma$ extended by the events $\{\dagger, \bot, \checkmark\}$ introduced above and the associated FSM input and output alphabets $\Sigma_I = \mathbb{P}(\Sigma) - \{\varnothing\}$ and $\Sigma_O = \Sigma \cup \{\bot\}$. Given an FSM test case $tc_{\text{FSM}} = (Q, \underline{q}, \Sigma_I, \Sigma_O, h, in)$, the image $T^*(tc_{\text{FSM}})$ is the CSP process $tc_{\text{CSP}}$ specified as follows.

$$
\begin{aligned}
tc_{\text{CSP}} &= tc(\underline{q}) \\
tc(q) &= \big(e : A(q) \to tc(q\text{-after-}(in(q)/e))\big) \\
&\quad \Box \, \big(e : A_{\text{PASS}}(q) \to \checkmark \to \mathit{Skip}\big) \\
&\quad \Box \, \big(e : A_{\text{FAIL}}(q) \to \dagger \to \mathit{Skip}\big)
\end{aligned}
$$

$$
\begin{aligned}
A(q) &= \{a \in in(q) \cup \{\bot\} \mid q\text{-after-}(in(q)/a) \notin \{\text{PASS}, \text{FAIL}\}\} \\
A_{\text{PASS}}(q) &= \{a \in in(q) \cup \{\bot\} \mid q\text{-after-}(in(q)/a) = \text{PASS}\} \\
A_{\text{FAIL}}(q) &= \{a \in in(q) \cup \{\bot\} \mid q\text{-after-}(in(q)/a) = \text{FAIL}\}
\end{aligned}
$$

Intuitively speaking, $tc_{\mathrm{CSP}}$ offers in each test step the same events to the CSP process to be tested as the FSM test case offers to the FSM under test. These events are specified in each non-terminating test step by $in(q)$, where $q$ is the current state of the FSM test case $tc_{\mathrm{FSM}}$. While the FSM test case offers these events a single set-valued member of the input alphabet $\Sigma_I$ to the FSM under test, the CSP test offers the same to the SUT by means of an external choice, so that it just depends on the SUT which event to choose. In addition to the events from $in(q)$, the CSP test case accepts the event $\perp$ which is not shared with the SUT but represents a timeout event provided by the testing environment to indicate that the SUT is blocked without accepting any of the events in $in(q)$.

The events offered/accepted by the CSP test in state $tc(q)$ are partitioned into 3 sets $A(q)$, $A_{\mathrm{PASS}}(q)$, and $A_{\mathrm{FAIL}}(q)$. The disjointness of these sets is a consequence of the fact that the FSM test case is observable: if $tc_{\mathrm{FSM}}$ can transit, for example, from $q$ to FAIL with I/O $in(q)/a$, then there exists no other transition from $q$ which is also labelled by $in(q)/a$.

*Example 4.* The FSM test case $tc_{\mathrm{FSM}}$ shown in Fig. 4 is mapped by $T^*$ to the following CSP test case.

$$
\begin{aligned}
T^*(tc_{\mathrm{FSM}}) &= P_1 \\
P_1 &= \big(e : \{b, c, \perp\} \bullet e \rightarrow \dagger \rightarrow \mathit{Skip}\big) \,\Box\, \big(a \rightarrow P_2\big) \\
P_2 &= \big(e : \{a, b, \perp\} \bullet e \rightarrow \dagger \rightarrow \mathit{Skip}\big) \,\Box\, \big(c \rightarrow P_3\big) \\
P_3 &= \big(e : \{a, b\} \bullet e \rightarrow \dagger \rightarrow \mathit{Skip}\big) \,\Box\, \big(\perp \rightarrow P_4\big) \,\Box\, \big(c \rightarrow P_5\big) \\
P_4 &= \big(e : \{b, c, \perp\} \bullet e \rightarrow \dagger \rightarrow \mathit{Skip}\big) \,\Box\, \big(a \rightarrow \checkmark \rightarrow \mathit{Skip}\big) \\
P_5 &= \big(e : \{a, b, \perp\} \bullet e \rightarrow \dagger \rightarrow \mathit{Skip}\big) \,\Box\, \big(c \rightarrow \checkmark \rightarrow \mathit{Skip}\big)
\end{aligned}
$$

□

The following theorem shows the validity of the satisfaction condition **SC2** regarding the test case map, the model map, and the pass conditions for tests on CSP level and FSM level.

**Theorem 3.** *Fixing a CSP process alphabet $\Sigma$, the model map $T : \underline{Sig}_1 \rightarrow FSM$ and the test case map $T^* : TC(FSM) \rightarrow TC(CSP)$ fulfil satisfaction condition* **SC2** *in the sense that*

$$\forall\, P \in \underline{Sig}_1, tc_{FSM} \in TC(FSM) : T(P)\ \underline{pass}_2\ tc_{FSM} \Leftrightarrow P\ \underline{pass}_1\ T^*(tc_{FSM})$$

*Proof.* Let $T(P) = (Q, \underline{q}, \Sigma_I, \Sigma_O, h)$ and $tc_{\mathrm{FSM}} = (Q', \underline{q}', \Sigma_I, \Sigma_O, h', in)$. We show by induction over the length of $s \in \Sigma_O^*$ that the following assertions hold for all $s$.

1. For every pass-trace $x/s \in L(T(P)) \cap L(tc_{\mathrm{FSM}})$, there exists a pass-trace $u \in \mathrm{tr}(P \,|[\, \Sigma \,]|\, T^*(tc_{\mathrm{FSM}}))$, such that $u \upharpoonright \Sigma$ corresponds to $x/s$.

2. For every fail-trace $x/s \in L(T(P)) \cap L(tc_{\mathrm{FSM}})$, there exists a fail-trace trace $u \in \mathrm{tr}(P \,|[\,\Sigma\,]|\, T^*(tc_{\mathrm{FSM}}))$, such that $u \upharpoonright \Sigma$ corresponds to $x/s$.
3. For all $x/s \in L(T(P))$, the graph nodes of $G(P)$ and the states of $T(P)$ are related by
$$G(P)/(s \upharpoonright \Sigma) = \underline{q}\text{-after-}(x/s).$$
4. FSM test $tc_{\mathrm{FSM}}$ and CSP test $T^*(tc_{\mathrm{FSM}})$ perform consistent state changes, in the sense that
$$x/s \in L(T(P)) \cap L(tc_{\mathrm{FSM}}) \wedge \underline{q}'\text{-after-}(x/s) \notin \{\mathrm{PASS}, \mathrm{FAIL}\} \Rightarrow T^*(tc_{\mathrm{FSM}})/(s \upharpoonright \Sigma) = tc(\underline{q}'\text{-after-}(x/s)),$$
where $tc(q)$ has been defined above with the test case map.

Proof of 1 and 2 obviously proves the theorem; assertions 3 and 4 are needed to perform the induction argument.

For the base case, $s$ is the empty trace $\varepsilon$, so $x/s$ is empty as well. We have $G(P)/\varepsilon = \underline{n} = \underline{q}$, due to the definition of the model map; this proves Assertion 3 for the base case. FSM test case $tc_{\mathrm{FSM}}$ resides in its initial state $\underline{q}'$. By definition of the test case map above, $T^*(tc_{\mathrm{FSM}})$ has initial CSP process state $tc(\underline{q}')$; this proves Assertion 4 for the base case $\#s = 0$. For assertions 1 and 2, there is nothing to prove: no test can pass or fail on the empty trace.

For the induction hypothesis, assume that the four assertions have been proven for $\#s \leqslant k$ with $0 \leqslant k$.

For the induction step, assume that the FSM $T(P)$ has run through trace $x/s \in L(T(P)) \cap L(tc_{\mathrm{FSM}})$ such that $\#s \leqslant k$ and $\underline{q}\text{-after-}(x/s) = q \in N$. Moreover, the FSM test case $tc_{\mathrm{FSM}}$ fulfils $\underline{q}'\text{-after-}(x/s) = q'$ for a uniquely defined state $q' \in Q'$. The induction hypothesis gives us $G(P)/(s \upharpoonright \Sigma) = \underline{q}\text{-after-}(x/s) = q$ from Assertion 3 and $T^*(tc_{\mathrm{FSM}})/(s \upharpoonright \Sigma) = tc(q')$ from Assertion 4. FSM test $tc_{\mathrm{FSM}}$ will offer input $in(q')$ to the FSM $T(P)$ which is being tested. We distinguish two cases for the outcomes of the resulting test step.

**Case 1.** $(x.in(q'))/(s.\perp) \in L(\underline{q}\text{-after-}(x/s))$.
By construction of $T(P)$, this is exactly the case if $in(q') \in r(G(P)/(s \upharpoonright \Sigma)) - \{\varnothing\}$. By construction of $T(P)$, this FSM will perform a self-loop transition labelled by $in(q')/\perp$ and therefore remain in $q$. For $P$, the corresponding behaviour is to refuse all events from $in(q')$, so $q$ also remains as the current state in $G(P)/(s \upharpoonright \Sigma)$. This proves Assertion 3 for the induction step, Case 1.

Since $tc_{\mathrm{FSM}}$ is output complete, $in(q')/\perp \in L(q')$. Depending on the nature of the test case, $q'\text{-after-}(in(q')/\perp)$ is one of the deadlock states PASS, FAIL, or it is a non-blocking FSM state, say, $q''$. Since $tc_{\mathrm{FSM}}$ is observable, however, this post-state is uniquely determined. Since the sets $A(q)$, $A_{\mathrm{PASS}}(q)$, and $A_{\mathrm{FAIL}}(q)$ are disjoint, $T^*(tc_{\mathrm{FSM}})$ has exactly one choice to proceed and will come to the post-states that are analogous to those of $tc_{\mathrm{FSM}}$: if $tc_{\mathrm{FSM}}$ terminates in PASS(FAIL), then $T^*(tc_{\mathrm{FSM}})$ will produced the events $\perp$ followed by $\checkmark(\dagger)$ and terminate. If $tc_{\mathrm{FSM}}$ transits with $in(q)/e$ to $q''$, CSP test case $T^*(tc_{\mathrm{FSM}})$ will perform $e \in A(q)$ as well and continue like process $tc(q'\text{-after-}(in(q')/\perp)$ with $q'\text{-after-}(in(q')/\perp) = q''$. Since $\perp, \checkmark, \dagger$ are not contained in $\Sigma$, the CSP

process $P$ cannot block these transitions of $T^*(tc_{\mathrm{FSM}})$, since we have assumed $P$ to be non-divergent. This shows that $tc_{\mathrm{FSM}}$ and $T^*(tc_{\mathrm{FSM}})$ engage into the same pass/fail/continue step, as was to be shown. This proves assertions 1,2,4 for Case 1.

**Case 2.** $(x.in(q'))/(s.e) \in L(\underline{q}\text{-after-}(x/s))$ with $e \in in(q') \subseteq \Sigma$.
By construction of $T(P)$, this case applies whenever $e \in A \wedge (n, e) \in \mathrm{dom}\ t$. Also by construction, FSM $T(P)$ will transit to $t(q, e)$ (recall that $q = G(P)/(s \restriction \Sigma)$, so $q$-after-$(in(q')/e) = u$ for a uniquely defined state $u = t(q, e) \in N$. Since $G(P)/(s \restriction \Sigma)$ will also transit to $u$ by definition of $t$, this proves the induction step for Assertion 3.

With the same argument as in Case 1, we conclude that $tc_{\mathrm{FSM}}$ and $tc(q')$ can perform equivalent test steps, leading to PASS or FAIL deadlock state, or to continuation of the test in the new state $q'$-after-$(in(q')/e)$. This proves the induction step for assertions 1,2,4 in Case 2 and completes the proof. □

## 4 Applications

### 4.1 Testing for Trace Equivalence

### 4.2 Testing for Trace Refinement

### 4.3 Testing for Failures Equivalence

### 4.4 Testing for Failures Refinement

## 5 Related Work

## 6 Conclusion

## References

1. 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
2. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: Failures Divergences Refinement (FDR) Version 3 (2013), https://www.cs.ox.ac.uk/projects/fdr/
3. 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)
4. 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

5. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
6. 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
7. Luo, G., von Bochmann, G., Petrenko, A.: Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. IEEE Trans. Software Eng. 20(2), 149–162 (1994), http://doi.ieeecomputersociety.org/10.1109/32.265636
8. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Jounal 19, 53–77 (1997)
9. 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
10. 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
11. 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)
12. Petrenko, A., Yevtushenko, N., Bochmann, G.v.: Fault models for testing in context. In: Gotzhein, R., Bredereke, J. (eds.) Formal Description Techniques IX – Theory, application and tools, pp. 163–177. Chapman&Hall (1996)
13. 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
14. Roscoe, A.W. (ed.): A Classical Mind: Essays in Honour of C. A. R. Hoare. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)
15. Roscoe, A.W.: Understanding Concurrent Systems. Springer, London, Dordrecht Heidelberg, New York (2010)