Under consideration for publication in Formal Aspects of Computing

Finite Complete Suites for CSP Refinement Testing

Ana Cavalcanti, Wen-ling Huang, Jan Peleska, and Adenilso Simao

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 trace refinement, as well as failures equivalence and failures 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, complete testing theories, finite state machines

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. This holds particularly for the embedded and cyber-physical systems domain. 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: 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 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)* [Hoa85, Ros10]; 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 complements work published by two of the authors in [CdSS17]. There, fault domains are 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 [Ros94]. The maximal refusals express the degree of nondeterminism that is present in a given CSP 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 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 four 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 four 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 [HP17].

FiXme Warning: alcc: I miss a paragraph that discusses what is difficult. So far, it sounds like just application of known results, which is not true.

Overview

@todo

2. Preliminaries

In this section, we present the background material relevant to our work.

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 [PYB96], where fault models have been originally introduced in the context of testing of finite state machines. 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, although a model of the SUT behaviour is. 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. This abstract notion of test case requires only 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 $M \ \underline{\text{fail}} \ U$. Our specific notion of test cases for CSP models is elaborated in Section 3.2.3.

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.

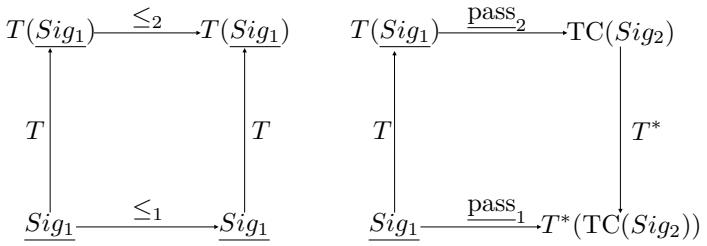

Fig. 1. Commuting diagrams reflecting the satisfaction condition.

1. If a member M' of the fault domain conforms to the reference model M, it passes the test suite, that is,
$$\forall\, M' \in Dom : M' \leqslant M \Rightarrow M' \ \underline{\text{pass}}\ \text{TS}$$
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,
$$\forall\, M' \in Dom : M' \ \underline{\text{pass}}\ \text{TS} \Rightarrow M' \leqslant M$$
This property is usually called *exhaustiveness*.

A test suite TS is *finite* if it contains finitely many test cases and every test case $U \in \text{TS}$ 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). Also, for T, we consider a sub-domain of Sig_1 since it may be the case that the map is not defined for the whole of Sig_1. For example, in our case, we are interested in finite-state CSP processes only and their translation to graphs described in the next section.

The pair (T, T^*) fulfils the *satisfaction condition* if, and only if, the following conditions 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 $T;\ \leqslant_2 = \leqslant_1;\ T$.[1]

SC2 The model map and the 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 $\underline{\text{pass}}_1 = T;\ \underline{\text{pass}}_2;\ T^*$.

The following theorem is a direct consequence of [HP17, Theorem 2.1].

[1] We note that the 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 $g \circ f$.

Theorem 2.1. With the notation introduced above, let (T, T^*) fulfil the satisfaction condition. Suppose that $\mathrm{TS}_2 \subseteq TC(Sig_2)$ 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

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

Then

$$\mathrm{TS}_1 = T^*(\mathrm{TS}_2)$$

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

This theorem supports our justification of testing approaches for finite state machines in the context of CSP.

2.3. CSP and Refinement

Communicating Sequential Processes

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

@todo

Normalised Transition Graphs for CSP Processes

As shown in [Ros94], 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, and only if, (n, e) is in the domain of t, that is, 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 $\mathrm{traces}(P)$. If $s \in \mathrm{traces}(P)$, 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 *fan-out* 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 [Ros10].

The total function r maps each node n to its *refusals* $r(n) = \mathrm{Ref}(n)$. 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. The number of refusal sets in $\mathrm{Ref}(P/s)$ specifies the degree of nondeterminism that 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.

For finite CSP processes, since the refusals of each process state are subset-closed [Hoa85, Ros10], $\mathrm{Ref}(P/s)$ can be re-constructed by knowing the set of *maximal refusals* $\mathrm{maxRef}(P/s) \subseteq \mathrm{Ref}(P/s)$. More formally, the maximal refusals $\mathrm{maxRef}(P/s)$ are defined as

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

Conversely, with the maximal refusals at hand, we can reconstruct the refusals by subset-closure:

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

To see that this approach works only for finite CSP processes, consider the example where Σ is infinite. In this case, $\mathrm{maxRef}(STOP/\langle\rangle)$ is empty, and so we cannot use this set to calculate the refusals of $STOP$, that is, $\mathrm{Ref}(STOP/\langle\rangle)$ 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 $\mathrm{maxRef}(n) = \mathrm{maxRef}(P/s)$.

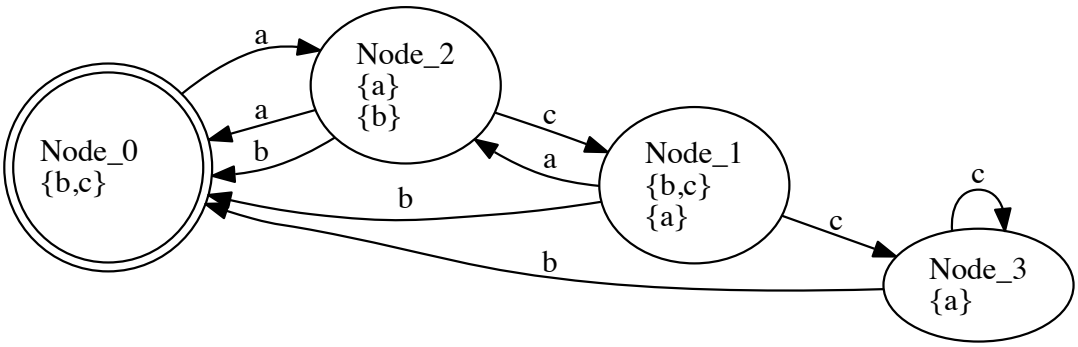

Fig. 2. Normalised transition graph of CSP process P from Example 1.

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

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

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

Example 1. Consider the CSP process P defined below, and that the set of events Σ is $\{a, b, c\}$.

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

Its transition graph $G(P)$ is shown in Fig. 2. Process state $P/\langle\rangle$ is represented there as Node_0, with $\{b, c\}$ as the only maximal refusal, 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 refusal sets derived from Q and R. Since these processes accept their initial events in external choice, $Q \sqcap R$ induces 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 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 maximal refusals labelling Node_1.

Similar considerations explain the other nodes and transitions in Fig. 2.

Note that the node names are generated by the FDR tool (see next paragraph). The node numbering is generated by FDR during the normalisation procedure. Therefore, the node numbers do not reflect the distance from the initial node Node_0. □

Tool Considerations

The FDR tool [GRABR14] supports model checking and semantic analyses of CSP processes. It provides an API [GRABR13] 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 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 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 [PY11, PY14, Hie04].

A *Finite State Machine (FSM)* is a tuple $M = (Q, \underline{q}, \Sigma_I, \Sigma_O, h)$ with state space Q, input alphabet Σ_I, and output alphabet Σ_O, where Q, Σ_I, and Σ_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, and $(q, x, y, q') \in h$ 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.

The set of all FSMs with input alphabet Σ_I and output alphabet Σ_O is denoted $\mathrm{FSM}(\Sigma_I, \Sigma_O)$.

We call x a *defined* input in state q, if there is a transition from q with input x. If every input of Σ_I is defined in every state, M is *completely specified*. If in every state q and for every output $y \in \Sigma_O$, an input x and 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, $h(q, x, y, q')$ and $h(q, x, y', q'')$ implies $(y, q') = (y', q'')$. Intuitively speaking, a specific input applied to a specific state uniquely determines both the 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 ε, and $(q, \varepsilon, \varepsilon, q) \in h$, for any state q. The *language* $L(M)$ of an FSM M is the set consisting of all possible pairs of input and output traces in M. $L(M) = LM(\underline{q})$, where for $q \in Q$

$$L_M(q) = \{\overline{x}/\overline{y} \mid \exists\, q' \in Q : h(q, \overline{x}, \overline{y}, q')\}$$

For an input trace $\overline{x} = a_1, ..., a_n$ and an output trace $\overline{y} = b_1, ..., b_n$, of the same length, x/y denotes a trace of input-output (I/O) pairs, that is, $x/y = (a_1, b_1), ..., (a_n, b_n)$. Formally, for $x/y \in (\Sigma_I \times \Sigma_O)^*$, x/y is an I/O trace $(a_1, b_1), ..., (a_n, b_n)$, with $x = a_1, ..., a_n \in \Sigma_I^*$ and $y = b_1, ..., b_n \in \Sigma_O^*$.

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

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

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 [PH17]. The algorithm that can be used to make this transformation operates analogously to the algorithm for normalising transition graphs of CSP processes.

Two FSM M_1, M_2 are *I/O-equivalent*, written $M_1 \sim M_2$, if, and only if, their languages coincide, i.e., $L(M_1) = L(M_2)$. FSM M_1 is a *reduction of* M_2, written $M_1 \preceq M_2$ 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., [LvBP94].

FSMs can be composed by synchronising over their common input and 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 and output alphabets. Then the *intersection of* M_1 *and* M_2 is specified by

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

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, $L(M_1 \cap M_2) = L(M_1) \cap L(M_2)$; this motivates the term 'intersection'. 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

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

In the next section, we show how normalised transition graphs can be encoded as FSMs.

3. Finite Complete Test Suites for CSP

Here, we present a model map and a test case map pair for CSP processes that fulfils the satisfaction condition. In Section 3.1 we present the model map, and in Section 3.2, the test case map.

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

We 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 events 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, or (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. In the FSM, 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 if it refuses an event.

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

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

The graph and the machine have the same nodes and initial nodes. The inputs of the machine are the non-empty sets of events, and the outputs include all events and $\perp$. In the machine, between every two states n and n', we have transitions for all inputs A and outputs e in A that are accepted in n and lead to n' according to the graph. In addition, for $\perp$, we have a self-transition for every non-empty refusal R of n.

We say that an FSM trace $(x/s) \in L(T(P))$ and CSP trace $s' \in \text{traces}(P)$ are *corresponding traces*, if $s' = s \upharpoonright \Sigma$. Observe that the FSM output trace s may contain deadlock events $\perp$ that are not contained in the process alphabet Σ. So, we have corresponding traces if they only differ by the presence or absence of $\perp$.

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 use 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

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

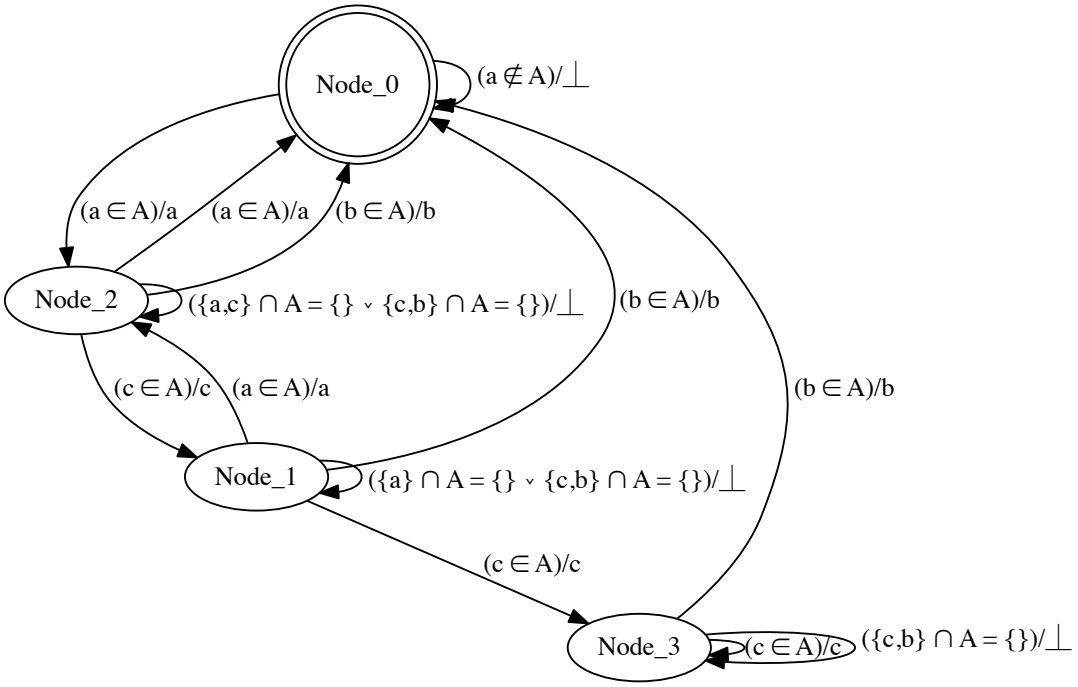

Fig. 3. FSM resulting from applying the model map to CSP process P from Example 1.

for example, stands for FSM transitions

$$\text{Node\_0} \xrightarrow{\{a\}/a} \text{Node\_2}$$
$$\text{Node\_0} \xrightarrow{\{a,b\}/a} \text{Node\_2}$$
$$\text{Node\_0} \xrightarrow{\{a,c\}/a} \text{Node\_2}$$
$$\text{Node\_0} \xrightarrow{\{a,b,c\}/a} \text{Node\_2}$$

□

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.

In the first lemma, we consider every trace s of a process P and the state n that it leads to in the normalised graph $G(P)$ of P. We show that, for every transition from n in the FSM, there are two possibilities. The first is that the output e is in the input A, so it is not $\perp$, and, in this case, the trace $s.e$ is a trace of P, and the target state n' is that identified in $G(P)$. We use $s.e$ to denote the trace obtained by adding the event e to the end of s. The second possibility is that the output is $\perp$, and in this case, we have a self-transition, that is, the target state is n itself, and (s, A) is a failure of P.

Lemma 3.1. Let $s \in \text{traces}(P)$ and $n = G(P)/s$ be the node of $G(P)$ corresponding to process state P/s. Then for any $A \in \Sigma_I$, $e \in \Sigma_O$:

$$(n, A, e, n') \in h \Leftrightarrow (e \in A \wedge s.e \in \text{traces}(P) \wedge n' = G(P)/s.e) \vee (e = \perp \wedge (s, A) \in \text{failures}(P) \wedge n' = n)$$

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

1. Suppose $e = e$ for some $e \in \Sigma$. Then $s' = s.e$. By definition of h, we have
$$\begin{aligned}(n, A, e, n') \in h \Leftrightarrow & e \in A \wedge (n, e) \in \operatorname{dom} t \wedge n' = t(n, e)\\ \Leftrightarrow & e \in A \wedge s.e \in \operatorname{traces}(P) \wedge n' = G(P)/s.e\end{aligned}$$
2. Suppose $e = \perp$. Then $s' = s$, and we can derive
$$\begin{aligned}(n, A, \perp, n') \in h \Leftrightarrow & A \in \operatorname{Ref}(P/s) \wedge n' = n\\ \Leftrightarrow & (s, A) \in \operatorname{failures}(P) \wedge n' = n\end{aligned}$$

□

Next, we use Lemma 3.1 to relate traces and after states of the process and the machine of arbitrary length.

Lemma 3.2. Let $x/s \in (\Sigma_I \times \Sigma_O)^*$ and $s' = s \restriction \Sigma$. Then

$$x/s \in L(T(P)) \Rightarrow s' \in \operatorname{traces}(P) \wedge \underline{q}\text{-}\underline{\text{after}}\text{-}x/s = G(P)/s'.$$

Proof. We prove the lemma by induction on $\#(x/y)$, the length of x/y.

For the base case, suppose that $x/s = \varepsilon$, then $s = s' = \varepsilon$, which is a trace of every process, and $\underline{q}\text{-}\underline{\text{after}}\text{-}x/s = \underline{n} = G(P)/\varepsilon$. We recall that $\underline{n}$ is the initial node.

For the induction hypothesis, suppose the statement holds for all $x/s \in (\Sigma_I \times \Sigma_O)^*$ with $\#(x/s) = k$, for some $k \geqslant 0$. Let $x/s \in (\Sigma_I \times \Sigma_O)^*$ with $\#(x/s) = k$ and $A/e \in \Sigma_I \times \Sigma_O$. To perform the induction step, let $s \restriction \Sigma = s'$ and $(s.e) \restriction \Sigma = s''$. We need to prove that $s'' \in \operatorname{traces}(P)$ and $q\text{-}\underline{\text{after}}\text{-}x.A/s.e = G(P)/s''$. By the induction hypothesis, we have $s' \in \operatorname{traces}(P)$ and $\underline{q}\text{-}\underline{\text{after}}\text{-}x/s = G(P)/s'$. Since $x.A/s.e \in L(T(P))$, there is a transition $(G(P)/s', A, e, n) \in h$ with $n = \underline{q}\text{-}\underline{\text{after}}\text{-}x.A/s.e$. We consider two cases. In the first, $e \in A$, and so $e \in \Sigma$. We obtain the required results as follows. We note that

$$s'' = (s.e) \restriction \Sigma = (s \restriction \Sigma).e = s'.e$$

and $s'.e \in \operatorname{traces}(P)$ follows from Lemma 3.1. For $\underline{q}\text{-}\underline{\text{after}}\text{-}x.A/s.e$, we note that it is n, which, by Lemma 3.1 is $G(P)/s'.e = G(P)/s''$ as shown above. In the second case, $e = \perp$. So, we note

$$s'' = (s.\perp) \restriction \Sigma = s \restriction \Sigma = s'$$

Lemma 3.1 gives that $(s', A) \in \operatorname{failures}(P)$ and so $s' \in \operatorname{traces}(P)$. Finally, for $\underline{q}\text{-}\underline{\text{after}}\text{-}x.A/s.e$, we note that it is n, which, by Lemma 3.1 is $G(P)/s' = G(P)/s''$ as shown above. □

With the above two lemmas, we below relate the language of the FSM with the trace and failure semantics of the CSP process that originated the FSM.

Lemma 3.3. Let $x/s \in L(T(P))$. Let $A/e \in \Sigma_I \times \Sigma_O$. Then

$$x.A/s.e \in L(T(P)) \Leftrightarrow \big(e \in A \wedge (s.e) \restriction \Sigma \in \operatorname{traces}(P)\big) \vee \big(e = \perp \wedge (s \restriction \Sigma, A) \in \operatorname{failures}(P)\big).$$

Proof. Let $x/s \in L(T(P))$, and $A/e \in \Sigma_I \times \Sigma_O$. Then by Lemma 3.2, we have $s' = s \restriction \Sigma \in \operatorname{traces}(P)$ and $G(P)/s' = \underline{q}\text{-}\underline{\text{after}}\text{-}x/s$. From Lemma 3.1 we have the following:

$$\begin{aligned}& x.A/s.e \in L(T(P))\\ \Leftrightarrow\ & \exists\, n' \in Q : (G(P)/s, A, e, n') \in h\\ \Leftrightarrow\ & (e \in A \wedge s.e \in \operatorname{traces}(P)) \vee \big(e = \perp \wedge (s, A) \in \operatorname{failures}(P)\big)\\ \Leftrightarrow\ & \big(e \in A \wedge (s.e) \restriction \Sigma \in \operatorname{traces}(P)\big) \vee \big(e = \perp \wedge (s \restriction \Sigma, A) \in \operatorname{failures}(P)\big)\end{aligned}$$

□

Next we show that every trace of a process is represented in the FSM defined by our map.

Lemma 3.4. For any $s \in \Sigma^*$,

$$s \in \operatorname{traces}(P) \Leftrightarrow \exists\, x \in \Sigma_I^* : x/s \in L(T(P)).$$

Proof. Let $s \in \Sigma^*$. We prove the lemma by induction on $\#s$. Suppose $s = \varepsilon$, then $s \in \operatorname{traces}(P)$ because the empty trace is a trace of every process. Also, we recall that $(q, \varepsilon, \varepsilon, q) \in h$ for every q, by convention. So, $\varepsilon/\varepsilon \in L(T(P))$. Therefore, $\exists\, x \in \Sigma_I^* : x/\varepsilon \in L(T(P))$ as required for the base case.

For the induction step, 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 we prove the equivalence by considering implication in each direction.

$$
\begin{array}{ll}
& s.e \in \mathrm{traces}(P) \\
\Leftrightarrow & s \in \mathrm{traces}(P) \land s.e \in \mathrm{traces}(P) & \text{[prefix closure of CSP traces]} \\
\Leftrightarrow & \exists\, x \in \Sigma_I^* : x/s \in L(T(P)) \land s.e \in \mathrm{traces}(P) & \text{[induction hypothesis]} \\
\Rightarrow & \exists\, x \in \Sigma_I^* : x.\{e\}/s.e \in L(T(P)) & \text{[Lemma 3.3 and } (s.e) \upharpoonright \Sigma = s.e\text{]} \\
\Rightarrow & \exists\, x \in \Sigma_I^* : x/s.e \in L(T(P)) & \text{[predicate calculus]}
\end{array}
$$

For the implication in the other direction, we have the following.

$$
\begin{array}{ll}
& \exists\, x \in \Sigma_I^* : x/s.e \in L(T(P)) \\
\Leftrightarrow & \exists\, x \in \Sigma_I^* \land \exists\, x' \in \Sigma_I : x/s \in L(T(P)) \land x.x'/s.e \in L(T(P)) & \text{[in } x/s.e\text{, we have } \#x = \#s.e\text{]} \\
\Rightarrow & s.e \in \mathrm{traces}(P) & \text{[Lemma 3.2 and } (s.e) \upharpoonright \Sigma = s.e\text{]}
\end{array}
$$

□

Similarly, every failure of a process is represented in the FSM defined by our map.

Lemma 3.5. For any $s \in \Sigma^*$ and $R \in \Sigma_I$,

$$(s, R) \in \mathrm{failures}(P) \Leftrightarrow \exists\, x \in \Sigma_I^* : x.R/s.\bot \in L(T(P))$$

Proof.

$$
\begin{array}{ll}
& \exists\, x \in \Sigma_I^* : x.R/s.\bot \in L(T(P)) \\
\Leftrightarrow & \exists\, x \in \Sigma_I^* : x/s \in L(T(P)) \land x.R/s.\bot \in L(T(P)) & \text{[definition of } L\text{]} \\
\Leftrightarrow & (s, R) \in \mathrm{failures}(P) & \text{[Lemma 3.3 and Lemma 3.4]}
\end{array}
$$

□

We finally get to the main result of this section, which establishes that our map satisfies **SC1**. The conformance relations are failures refinement and FSM reduction.

Theorem 3.1. Let P, Q be two CSP processes over the same alphabet Σ. Then $T(Q) \preceq T(P) \Leftrightarrow P \sqsubseteq_F Q$.

Proof. First, suppose $T(Q) \preceq T(P)$. Let $s \in \Sigma^*, A \in \Sigma_I$.

$$
\begin{array}{lcll}
s \in \mathrm{traces}(Q) & \Leftrightarrow & \exists\, x \in \Sigma_I^* : x/s \in L(T(Q)) & \text{[Lemma 3.4]} \\
& \Rightarrow & \exists\, x \in \Sigma_I^* : x/s \in L(T(P)) & [T(Q) \preceq T(P)] \\
& \Leftrightarrow & s \in \mathrm{traces}(P) & \text{[Lemma 3.4]}
\end{array}
$$

$$
\begin{array}{lcll}
(s, A) \in \mathrm{failures}(Q) & \Leftrightarrow & \exists\, x \in \Sigma_I^* : x.A/s.\bot \in L(T(Q)) & \text{[Lemma 3.5]} \\
& \Rightarrow & \exists\, x \in \Sigma_I^* : x.A/s.\bot \in L(T(P)) & [T(Q) \preceq T(P)] \\
& \Leftrightarrow & (s, A) \in \mathrm{failures}(P) & \text{[Lemma 3.4]}
\end{array}
$$

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

For the implication in the other direction, now suppose $P \sqsubseteq_F Q$. We prove by induction on $\#(x/s)$ that for any $x/s \in (\Sigma_I \times \Sigma_O)^*$, $x/s \in L(T(Q)) \Rightarrow x/s \in L(T(P))$ holds. It is trivial if $\#(x/s) = 0$ because $x = s = \varepsilon$, which is in the language of every FSM. Suppose $x/s \in L(T(Q)) \Rightarrow x/s \in L(T(P))$ holds for any $x/s \in (\Sigma_I \times \Sigma_O)^*$ of length $k \geqslant 0$. For any $x/s \in (\Sigma_I \times \Sigma_O)^*$ of length k and for any $A/e \in \Sigma_I \times \Sigma_O$. Let $s' = s \upharpoonright \Sigma$ and $s'' = (s.e) \upharpoonright \Sigma$.

$$
\begin{array}{ll}
& x.A/s.e \in L(T(Q)) \\
\Rightarrow & x/s \in L(T(Q)) \land \big((e \in A \land s' \in \mathrm{traces}(Q)) \lor (e = \bot \land (s', A) \in \mathrm{failures}(Q))\big) \\
& \qquad \text{[prefix closure of } L(T(Q)) \text{ and Lemma 3.3]} \\
\Rightarrow & x/s \in L(T(P)) \land \big((e \in A \land s' \in \mathrm{traces}(P)\big) \lor (e = \bot \land (s', A) \in \mathrm{failures}(P))\big) \\
& \qquad \text{[induction hypothesis and } P \sqsubseteq_F Q\text{]} \\
\Rightarrow & x.A/s.e \in L(T(P)) & \text{[Lemma 3.3]}
\end{array}
$$

Hence $P \sqsubseteq_F Q \Rightarrow T(Q) \preceq T(P)$. □

In the next section, we consider **SC2** for the test case map.

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

In this section, we first of all define a notion of test cases for FSM and then for CSP. Afterwards, we define a map between these test cases that satisfies **SC2** as required.

3.2.1. FSM Test Cases

Following [PY14], 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 that provides only a single input in any given state. Running in intersection mode with the SUT, the test case provides a specific input to the SUT 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 deadlock states.

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 be checked that *every possible* test execution of tc_{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,

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

It will turn out in the exposition below that this pass criterion is appropriate for testing the reduction conformance relation. For equivalence testing, it is also of interest whether *all* pass-traces of an adaptive test case are contained in the language of the FSM under test. This induces a second pass criterion specified by

$$\begin{aligned} M \ \underline{\mathrm{pass}}_{=} \ tc_{\mathrm{FSM}} \equiv \big(&\forall\, x/y \in L(M) \cap L(tc_{\mathrm{FSM}}) : \underline{q}\text{-}\underline{\mathrm{after}}\text{-}(x/y) \neq \mathrm{FAIL}\big) \wedge \\ \big(&\forall\, x/y \in L(tc_{\mathrm{FSM}}) : \underline{q}\text{-}\underline{\mathrm{after}}\text{-}(x/y) = \mathrm{PASS} \Rightarrow x/y \in L(M)\big) \end{aligned}$$

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. □

While adaptive test cases are an operational representation of tests, an alternative, equivalent, denotational description can be given by means of *pass-traces* and *fail-traces*, as introduced in [HP18]: a test case $U = (U_{\mathrm{pass}}, U_{\mathrm{fail}})$ consists of two sets of traces $U_{\mathrm{pass}}, U_{\mathrm{fail}} \subseteq L(M)$, such that $U_{\mathrm{pass}} \cap U_{\mathrm{fail}} = \varnothing$. We introduce two pass-criteria that will be used later on to distinguish tests for equivalence and tests for refinement.

$$M \ \underline{\mathrm{pass}}_{=} \ U \equiv U_{\mathrm{pass}} \subseteq L(M) \wedge L(M) \cap U_{\mathrm{fail}} = \varnothing \tag{2}$$

$$M \ \underline{\mathrm{pass}}_{\subseteq} \ U \equiv L(M) \cap U_{\mathrm{fail}} = \varnothing \tag{3}$$

By definition, both pass criteria require that none of the fail-traces are contained in the language of the FSM under test. This suffices when testing for language inclusion, as defined in the pass-criterion (2). When testing for language equivalence as specified in (2), it is additionally required that *all* pass traces need to be contained in $L(M)$.

It is easy to see how adaptive test cases can be transformed into their denotational representation. Conversely, it is easy to transform a test case $U = (U_{\mathrm{pass}}, U_{\mathrm{fail}})$ into one or more adaptive test cases. Since

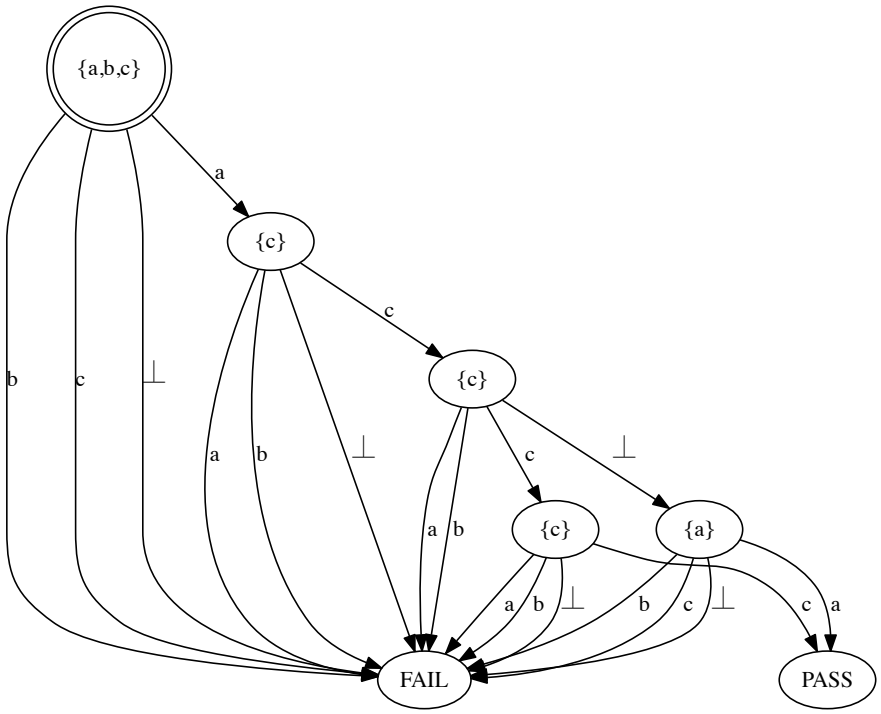

Fig. 4. An FSM test case which is passed by the FSM presented in Example 2.

the denotational test case representation is very general, however, it is not guaranteed that they can always be transformed into *a single* adaptive test case: this is due to the fact that adaptive test cases are required to be single-input.

Example 4. Consider again the adaptive FSM test case depicted in Fig. 4. It pass-traces and fail-traces are given by

$$\begin{aligned} U_{\text{pass}} = \{&(\{a,b,c\}/a).(\{c\}/c).(\{c\}/\bot).(\{a\}/a),\\ &(\{a,b,c\}/a).(\{c\}/c).(\{c\}/c).(\{c\}/c)\} \end{aligned}$$

$$\begin{aligned} U_{\text{fail}} = \{&(\{a,b,c\}/b),(\{a,b,c\}/c),(\{a,b,c\}/\bot),\\ &(\{a,b,c\}/a).(\{c\}/a),(\{a,b,c\}/a).(\{c\}/b),(\{a,b,c\}/a).(\{c\}/\bot),\\ &(\{a,b,c\}/a).(\{c\}/c).(\{c\}/a),(\{a,b,c\}/a).(\{c\}/c).(\{c\}/b),\\ &(\{a,b,c\}/a).(\{c\}/c).(\{c\}/\bot).(\{a\}/b),(\{a,b,c\}/a).(\{c\}/c).(\{c\}/\bot).(\{a\}/c),\\ &(\{a,b,c\}/a).(\{c\}/c).(\{c\}/\bot).(\{a\}/\bot),\\ &(\{a,b,c\}/a).(\{c\}/c).(\{c\}/c).(\{c\}/a),(\{a,b,c\}/a).(\{c\}/c).(\{c\}/c).(\{c\}/b),\\ &(\{a,b,c\}/a).(\{c\}/c).(\{c\}/c).(\{c\}/\bot)\} \end{aligned}$$

This can be directly derived by performing a reachability analysis for the states PASS and FAIL, respectively, using the graph representation in Fig. 4. □

3.2.2. FSMs in the Range of the Model Map

We will now state some facts about FSMs in the range of the model map; these will be exploited in the definition of effective CSP test cases for failures equivalence and failures refinement. They are also needed in the proof of the satisfaction condition **SC2** showing the correctness of the test case map to be constructed below.

Lemma 3.6. Let $M = (Q, \underline{q}, \Sigma_I, \Sigma_O, h)$ be an FSM in the range of the model map T. Then the following facts hold for all $q \in Q$, $A, A_i \in \Sigma_I$, $e \in \Sigma$, and $u_i \in (\Sigma_I \times \Sigma_O)^*$.

$$(A_1/\perp).(A_2/\perp) \in L(q) \Leftrightarrow (A_2/\perp).(A_1/\perp) \in L(q) \tag{4}$$

$$u_1.(A/\perp).u_2 \in L(q) \Leftrightarrow u_1.u_2 \in L(q) \wedge (A/\perp) \in L(\underline{q}\text{-}\underline{\text{after}}\text{-}u_1) \tag{5}$$

$$(A_1/\perp) \in L(q) \wedge A_2 \subseteq A_1 \Rightarrow (A_2/\perp) \in L(q) \tag{6}$$

$$(A_1/e) \in L(q) \wedge A_1 \subseteq A_2 \Rightarrow (A_2/e) \in L(q) \tag{7}$$

$$(A/e) \in L(q) \Rightarrow e \in A \tag{8}$$

Proof. Formulas (4) and (5) are direct consequences of the fact that all transitions labelled with output $\perp$ are self-loops. Formula (6) is a direct consequences of the fact that refusals are subset-closed, and $(A_1/\perp) \in L(q)$ if and only if A_1 is contained in some refusal of the CSP transition graph node q (recall that the model map maps graph nodes to FSM states in one-one fashion). For proving Formula (7), we note that $(A_1/e) \in L(q)$ if and only if the CSP transition graph contains an outgoing transition from graph node q which is labelled by e. Since A_2 is supposed to be a super-set of A_1, this property holds for A_2 as well. Formula (8) follows from the definition of the model map T which specifies $e \in A$ as a necessary condition for FSM transitions $h(q, A, e, q')$ with $e \in \Sigma$. □

The following theorem states that complete test suites for FSMs in the range of the model map can be simplified in such a way, that every pass-trace contains at most one self-transition $(A/\perp)$, and this only occurs at the end of the trace.

Theorem 3.2. Let $M = (Q, \underline{q}, \Sigma_I, \Sigma_O, h)$ be an FSM in the range of the model map T. Let TS be a complete test suite for verifying reduction or refinement with a fault domain that is also in the range of the model map. Let TS′ be another test suite satisfying the following conditions.

[Condition 1.]
$$\forall\, U = (U_{\text{pass}}, U_{\text{fail}}) \in \text{TS} : \exists\, U' = (U'_{\text{pass}}, U'_{\text{fail}}) \in \text{TS}' :$$
$$\big(\forall\, u \in U_{\text{pass}} : \exists\, u' \in U'_{\text{pass}} : \mathit{last}(u) = \mathit{last}(u') \wedge \mathit{front}(u \restriction (\Sigma_I \times \Sigma)) = \mathit{front}(u')\big) \wedge$$
$$\big(\forall\, u \in U_{\text{fail}} : \exists\, u' \in U'_{\text{fail}} : \mathit{last}(u) = \mathit{last}(u') \wedge \mathit{front}(u \restriction (\Sigma_I \times \Sigma)) = \mathit{front}(u')\big)$$

[Condition 2.]
$$\forall\, U = (U_{\text{pass}}, U_{\text{fail}}) \in \text{TS}, A \in \Sigma_I : \exists\, U' = (U'_{\text{pass}}, U'_{\text{fail}}) \in \text{TS}' :$$
$$\big(\forall\, u.(A/\perp).v \in U_{\text{pass}} : \exists\, u' \in U'_{\text{pass}} : u' = (u \restriction (\Sigma_I \times \Sigma)).(A/\perp)\big) \wedge$$
$$\big(\forall\, u.(A/\perp).v \in U_{\text{fail}} : \exists\, u' \in U'_{\text{fail}} : u' = (u \restriction (\Sigma_I \times \Sigma)).(A/\perp)\big)$$

Then TS′ is also complete for the same conformance relation and fault domain.

Proof. We first consider the pass-criterion $\underline{\text{pass}}_=$ for language equivalence. It first has to be shown that TS′ is also sound; this means that conforming implementations will pass all test cases of TS′. Suppose therefore that $M' \sim M$ for $M' = (Q', \underline{q}', \Sigma_I, \Sigma_O, h')$ and let $U' = (U'_{\text{pass}}, U'_{\text{fail}}) \in \text{TS}'$. Let $u' \in U'_{\text{pass}}$. We have to show that $u' \in L(\underline{q})$. From Condition 1 and Condition 2 we conclude that there exists a test case $U = (U_{\text{pass}}, U_{\text{fail}}) \in \text{TS}$ and a pass-trace $u \in U_{\text{pass}}$ such that (1) $\mathit{last}(u) = \mathit{last}(u') \wedge \mathit{front}(u \restriction (\Sigma_I \times \Sigma)) = \mathit{front}(u')$ or (2) $u = w.(A/\perp).v$ and $u' = w \restriction (\Sigma_I \times \Sigma).(A/\perp)$. Since TS is sound, we know that $u \in L(\underline{q}')$.

In Case (1), Lemma 3.6, (5) implies that $u' \in L(\underline{q}')$, because u' has been constructed from u by deleting I/O-pairs $(A/\perp)$. In Case (2), $u' = w \restriction (\Sigma_I \times \Sigma).(A/\perp)$ for some $u = w.(A/\perp).v \in U_{\text{pass}}$. Since $w.(A/\perp).v \in L(\underline{q}')$, prefix closure of languages gives us $w.(A/\perp) \in L(\underline{q}')$. Again, Lemma 3.6, (5) implies that $u' \in L(\underline{q}')$.

Now it has to be shown that no fail-trace $a' \in U'_{\text{fail}}$ is contained in the language of M'. To this end, we

observe that a' is the prefix of some fail-trace $a \in U_{\text{fail}}$, where some $(A/\perp)$ I/O-events have been deleted. Lemma 3.6, (5) implies that $a' \in L(M') \Leftrightarrow a \in L(M')$. Since TS is sound and M' conforms to M, a is not an element of $L(M')$. Therefore, $a' \notin L(M')$, so M' also passes the test U'. This completes the proof that TS$'$ is sound for showing language equivalence.

To show that TS$'$ is exhaustive for language equivalence, suppose that $M' \not\sim M$. Since TS is complete by assumption, this implies the existence of a test case $U = (U_{\text{pass}}, U_{\text{fail}}) \in$ TS and (1) a fail-trace $a \in U_{\text{fail}}$ such that $u \in \text{Ł}(\underline{q}')$, or (2) the existence of a pass-trace $u \in U_{\text{pass}}$ which is *not* contained in $L(\underline{q}')$.

For Case (1), Condition 1 ensures the existence of a fail-trace a' which is derived from a by deleting some $(A/\perp)$ I/O-pairs. From Lemma 3.6, (5), we know that a' is also in the language of M'. Therefore, M' also fails the TS$'$-test case containing fail-trace a'. For Case (2), Condition 1 yields the existence of as pass-trace u' in some test case of TS$'$ which is constructed from u by deleting some $(A/\perp)$ I/O-pairs. Now Lemma 3.6, (5) implies that $u' \notin L(\underline{q}')$ because $u \notin L(\underline{q}')$. Therefore, the TS$'$ test case containing u' also fails for M'.

Next, we consider the pass criterion $\underline{\text{pass}}_{\subseteq}$ for reduction. First, the soundness of TS$'$ is shown. To this end, we only have to prove that the fail-traces of U' are not contained in $L(M')$ if M' is a reduction of M. This follows from the same argument as applied above for the language equivalence case. To prove exhaustiveness, assume that $a \in L(M')$ for some fail-trace of a test case of TS. Now we can apply the same argument as for the exhaustiveness Case (1) described for equivalence testing above to prove the existence of a fail-trace a' from test suite TS$'$ which is also contained in M''s language. This completes the proof. □

Next, it is shown that every test case in denotational representation can be transformed into a set of adaptive test cases.

Theorem 3.3. Let $U = (U_{\text{pass}}, U_{\text{fail}})$ be a test case. Then there exists a set $\mathcal{A}(U)$ of adaptive test cases that are equivalent to U in the sense that

$$M \ \underline{\text{pass}}_{=} \ U \Leftrightarrow \forall\, U_A \in \mathcal{A}(U) : M \ \underline{\text{pass}}_{=} \ U_A$$
$$M \ \underline{\text{pass}}_{\subseteq} \ U \Leftrightarrow \forall\, U_A \in \mathcal{A}(U) : M \ \underline{\text{pass}}_{\subseteq} \ U_A$$

Proof. aa □

Data: Finite test case $U = (U_{\text{pass}}, U_{\text{fail}})$.
Result: Finite set of adaptive test cases equivalent to U.

```
1  begin
2  |  A(U) := ∅;
3  |  P := U_pass ;
4  |  F := U_fail ;
5  |  while P ∪ F ≠ ∅ do
6  |  |  if P ≠ ∅ then
7  |  |  |  tc = createFromPassTrace(P);
8  |  |  |  while extendByPassTrace(tc, P) do ;
9  |  |  |  while extendByFailTrace(tc, P) do ;
10 |  |  |  makeComplete(tc);
11 |  |  |  A(U) := A(U) ∪ {tc};
12 |  |  end
13 |  |  else
14 |  |  |  tc := createFromFailTrace(P);
15 |  |  |  while extendByFailTrace(tc, P) do ;
16 |  |  |  makeComplete(tc);
17 |  |  |  A(U) := A(U) ∪ {tc};
18 |  |  end
19 |  end
20 |  return A(U);
21 end
```

Algorithm 1: Conversion of test case $U = (U_{\text{pass}}, U_{\text{fail}})$ to set of adaptive test cases.

```
1 TestCase createFromPassTrace(inout P : ℙ((Σ_I × Σ_O)*)) begin
2 |   Select trace π from P;
3 |   P := P − {π};
4 |   return createTestCase(π, PASS);
5 end
```

Algorithm 2: Initialisation adaptive test case with pass-trace.

```
1 TestCase createFromFailTrace(inout F : ℙ((Σ_I × Σ_O)*)) begin
2 |   Select trace π from F;
3 |   F := F − {π};
4 |   return createTestCase(π, FAIL);
5 end
```

Algorithm 3: Initialisation adaptive test case with fail-trace.

```
1 TestCase createTestCase(in π : (Σ_I × Σ_O)*, in endState : State) begin
2 |   let π = (x_0 ... x_{#π−1})/(y_0 ... y_{#π−1});
3 |   Q := {q, q_1, ..., q_{#π−1}, endState};
4 |   tc := (Q, q, Σ_I, Σ_O, h, in);
5 |   in := {q ↦ x_0, q_1 ↦ x_1, ..., q_{#π−1} ↦ x_{#π−1}};
6 |   h := {(q, x_0, y_0, q_1)} ∪ {(q_i, x_i, y_i, q_{i+1}) | i = 1, ... #π − 2} ∪ {(q_{#π−1}, x_{#π−1}, y_{#π−1}, endState)};
7 |   return (Q, q, Σ_I, Σ_O, h, in);
8 end
```

Algorithm 4: Initialisation adaptive test case with given trace and end-state.

3.2.3. CSP Test Cases

A *CSP test case* tc_{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 Σ of P. This is expressed by the formula

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

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

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

In principle, very general classes of CSP processes can be used for testing, as introduced, for example, in [PS96, PS97]. 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.

3.2.4. Test Case Map

The test case map $T^* : TC(FSM) \nrightarrow TC(CSP)$ is specified with respect to a fixed CSP process alphabet Σ extended by the events $\{\dagger, \bot, \checkmark\}$ introduced above and the associated FSM input and output alphabets $\Sigma_I = \mathbb{P}(\Sigma) - \{\varnothing\}$ and $\Sigma_O = \Sigma \cup \{\bot\}$. As a consequence of Theorem 3.2, the domain of T^* may be restricted to test cases $U = (U_{\text{pass}}, U_{\text{fail}})$ of the form where every pass-trace $u \in U_{\text{fail}}$ or fail-trace $a \in U_{\text{fail}}$ contains I/O-pairs $(A/\bot)$, $A \in \Sigma_I$, if any, at the very last element of the trace. Theorem 3.2 implies that *any* complete test suite for FSMs in the range of the model map T can be represented by means of test cases fulfilling this condition.

```
1 𝔹 extendByPassTrace(inout tc : TestCase, inout P : ℙ((Σ_I × Σ_O)*)) begin
2 |  for π ∈ P do
3 |  |  if extendByTrace(tc, π, PASS) then
4 |  |  |  P := P − {π};
5 |  |  |  return true;
6 |  |  end
7 |  end
8 |  return false;
9 end
```

Algorithm 5: Extend adaptive test case under construction with pass-trace.

```
1 𝔹 extendByFailTrace(inout tc : TestCase, inout F : ℙ((Σ_I × Σ_O)*)) begin
2 |  for π ∈ F do
3 |  |  if extendByTrace(tc, π, FAIL) then
4 |  |  |  F := F − {π};
5 |  |  |  return true;
6 |  |  end
7 |  end
8 |  return false;
9 end
```

Algorithm 6: Extend adaptive test case under construction with fail-trace.

```
1 𝔹 extendByTrace(inout tc : TestCase, in π : (Σ_I × Σ_O)*, in endState : State) begin
2 |  let π = π1.π2 ∧ π1 ∈ L(tc) ∧ π1.π2(0) ∉ L(tc) ∧ π2 = (x0 ... xm)/(y0 ... ym) ;
3 |  let tc = (Q, q0, Σ_I, Σ_O, h, in) ∧ n =| Q |;
4 |  q := q0-after-π1;
5 |  if in(q) ≠ x0 then return false;
6 |  Q := Q ∪ {q_{n+1}, ..., q_{n+m}, endState};
7 |  h := h ∪ {(q, x0, y0, qn)} ∪ {(q_{n+j}, xj, yj, q_{n+j+1}) | j = 1, ..., m − 1} ∪ {(q_{n+m}, xm, ym, endState)};
8 |  in := in ∪ {q_{n+j} ↦ xj | j = 1, ..., m};
9 |  return true;
10 end
```

Algorithm 7: Try to extend test case by given trace and end-state.

```
1 makeComplete(inout tc : TestCase) begin
2 |  let tc = (Q, q0, Σ_I, Σ_O, h, in) ∧ n =| Q |;
3 |  for q ∈ Q \ {PASS, FAIL, INC} do
4 |  |  for y ∈ Σ_O do
5 |  |  |  if (in(q)/y) ∉ L(q) then  Q := Q ∪ {INC};
6 |  |  |  h := h ∪ {(q, in(q), y, INC)} ;
7 |  |  end
8 |  end
9 end
```

Algorithm 8: Make test case output-complete.

Given such an FSM test case $tc_{\mathrm{FSM}} = (Q, \underline{q}, \Sigma_I, \Sigma_O, h, in)$, the image $T^*(tc_{\mathrm{FSM}})$ is the CSP process tc_{CSP} specified as follows.

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

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

@todo update text to new test case version. Intuitively speaking, tc_{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_{FSM}. While the FSM test case offers these events a single set-valued member of the input alphabet Σ_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 $\bot$ 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_{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 5. The FSM test case tc_{FSM} shown in Fig. 4 is mapped by T^* to the following CSP test case.

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

□

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

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

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 $u \restriction \Sigma$ 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 $u \restriction \Sigma$ 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

$$G(P)/(s \restriction \Sigma) = \underline{q}\text{-}\underline{\text{after}}\text{-}(x/s).$$

4. FSM test tc_{FSM} and CSP test $T^*(tc_{\mathrm{FSM}})$ perform consistent state changes, in the sense that

$$x/s \in L(T(P)) \cap L(tc_{\mathrm{FSM}}) \land \underline{q}'\text{-}\underline{\text{after}}\text{-}(x/s) \notin \{\mathrm{PASS}, \mathrm{FAIL}\} \Rightarrow \\ T^*(tc_{\mathrm{FSM}})/(s \restriction \Sigma) = tc(\underline{q}'\text{-}\underline{\text{after}}\text{-}(x/s)),$$

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 ε, 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_{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 $\underline{q}\text{-}\underline{\text{after}}\text{-}(x/s) = q \in N$. Moreover, the FSM test case tc_{FSM} fulfils $\underline{q}'\text{-}\underline{\text{after}}\text{-}(x/s) = q'$ for a uniquely defined state $q' \in Q'$. The induction hypothesis gives us $G(P)/(s \restriction \Sigma) = \underline{q}\text{-}\underline{\text{after}}\text{-}(x/s) = q$ from Assertion 3 and $T^*(tc_{\mathrm{FSM}})/(s \restriction \Sigma) = tc(q')$ from Assertion 4. FSM test tc_{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. $(x.in(q'))/(s.\perp) \in L(\underline{q}\text{-}\underline{\text{after}}\text{-}(x/s))$.
By construction of $T(P)$, this is exactly the case if $in(q') \in r(G(P)/(s \restriction \Sigma)) - \{\varnothing\}$. 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 $G(P)/(s \restriction \Sigma)$. This proves Assertion 3 for the induction step, Case 1.

Since tc_{FSM} is output complete, $in(q')/\perp \in L(q')$. Depending on the nature of the test case, $q'\text{-}\underline{\text{after}}\text{-}(in(q')/\perp)$ is one of the deadlock states PASS, FAIL, or it is a non-blocking FSM state, say, q''. Since tc_{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_{FSM}: if tc_{FSM} terminates in PASS(FAIL), then $T^*(tc_{\mathrm{FSM}})$ will produced the events $\perp$ followed by $\checkmark(\dagger)$ and terminate. If tc_{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 $tc(q'\text{-}\underline{\text{after}}\text{-}(in(q')/\perp)$ with $q'\text{-}\underline{\text{after}}\text{-}(in(q')/\perp) = q''$. Since $\perp, \checkmark, \dagger$ are not contained in Σ, 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_{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. $(x.in(q'))/(s.e) \in L(\underline{q}\text{-}\underline{\text{after}}\text{-}(x/s))$ with $e \in in(q') \subseteq \Sigma$.
By construction of $T(P)$, this case applies whenever $e \in A \land (n, e) \in \mathrm{dom}\ t$. Also by construction, FSM $T(P)$ will transit to $t(q, e)$ (recall that $q = G(P)/(s \restriction \Sigma)$, so $q\text{-}\underline{\text{after}}\text{-}(in(q')/e) = u$ 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_{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 $q'\text{-}\underline{\text{after}}\text{-}(in(q')/e)$. 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

[CdSS17] Ana Cavalcanti and Adenilso da Silva Simão. Fault-based testing for refinement in CSP. In Nina Yevtushenko, Ana Rosa Cavalli, and Hüsnü Yenigün, editors, *Testing Software and Systems - 29th IFIP WG 6.1 International Conference, ICTSS 2017, St. Petersburg, Russia, October 9-11, 2017, Proceedings*, volume 10533 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2017.

[GRABR13] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A.W. Roscoe. *Failures Divergences Refinement (FDR) Version 3*, 2013.

[GRABR14] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A.W. Roscoe. FDR3 — A Modern Refinement Checker for CSP. In Erika brahm and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 187–201, 2014.

[Hie04] Robert M. Hierons. Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Trans. Computers*, 53(10):1330–1342, 2004.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[HP17] Wen-ling Huang and Jan Peleska. Complete model-based equivalence class testing for nondeterministic systems. *Formal Aspects of Computing*, 29(2):335–364, 2017.

[HP18] Wen-ling Huang and Jan Peleska. Model-based testing strategies and their (in)dependence on syntactic model representations. *STTT*, 20(4):441–465, 2018.

[LvBP94] Gang Luo, Gregor von Bochmann, and Alexandre Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Trans. Software Eng.*, 20(2):149–162, 1994.

[PH17] Jan Peleska and Wen-ling Huang. *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.

[PS96] Jan Peleska and Michael Siegel. From testing theory to test driver implementation. In Marie-Claude Gaudel and Jim Woodcock, editors, *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*, volume 1051 of *Lecture Notes in Computer Science*, pages 538–556. Springer, 1996.

[PS97] J. Peleska and M. Siegel. Test automation of safety-critical reactive systems. *South African Computer Jounal*, 19:53–77, 1997.

[PY11] A. Petrenko and N. Yevtushenko. Adaptive testing of deterministic implementations specified by nondeterministic fsms. In *Testing Software and Systems*, number 7019 in Lecture Notes in Computer Science, pages 162–178, Berlin, Heidelberg, 2011. Springer.

[PY14] Alexandre Petrenko and Nina Yevtushenko. 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*, pages 224–228. IEEE Computer Society, 2014.

[PYB96] A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Fault models for testing in context. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX – Theory, application and tools*, pages 163–177. Chapman&Hall, 1996.

[Ros94] A. W. Roscoe, editor. *A Classical Mind: Essays in Honour of C. A. R. Hoare*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.

[Ros10] A. W. Roscoe. *Understanding Concurrent Systems*. Springer, London, Dordrecht Heidelberg, New York, 2010.