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. Keywords: Model-based testing, Complete testing theories, CSP, Refinement

1 Introduction

Motivation

Main Contributions

Overview

2 Preliminaries

2.1 Complete Testing Theories

Fault Models, Test Cases, Test Suites, and Completeness 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.

Given a signature *Sig* of models, a *fault model* $\mathcal{F} = (M, \leqslant, \mathit{Dom})$ specifies a *reference model* $M \in \mathit{Sig}$, a *conformance relation* $\leqslant \subseteq \mathit{Sig} \times \mathit{Sig}$ between models, and a *fault domain* $\mathit{Dom} \subseteq \mathit{Sig}$. This terminology follows [7], 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. By $F(\mathit{Sig}, \leqslant)$ we denote the set of all fault models $\mathcal{F}$ defined for signature *Sig* and conformance relation $\leqslant$.

Let $\mathrm{TC}(\mathit{Sig})$ 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 $\underline{\mathrm{pass}} \subseteq \mathit{Sig} \times \mathrm{TC}(\mathit{Sig})$. For $(M, U) \in \underline{\mathrm{pass}}$, the infix notation $M \ \underline{\mathrm{pass}}\ U$ is used, and interpreted as *'Model M passes the test case U'*. If $(M, U) \notin \underline{\mathrm{pass}}$ holds, this is abbreviated by $M \ \underline{\mathrm{fail}}\ U$.

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

2.2 Translation of Testing Theories

Let Sig_1 and Sig_2 be two signatures with conformance relations $\leqslant_1$ and $\leqslant_2$, and test case relations $\underline{\mathrm{pass}}_1$ and $\underline{\mathrm{pass}}_2$, respectively. A function $T : \underline{\mathit{Sig}_1} \to \mathit{Sig}_2$ defined on a sub-domain $\underline{\mathit{Sig}_1} \subseteq \mathit{Sig}_1$ is called a *model map*, and a function $T^* : \mathrm{TC}(\mathit{Sig}_2) \to \mathrm{TC}(\mathit{Sig}_1)$ 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
$$\forall \mathcal{S}, \mathcal{S}' \in \underline{\mathit{Sig}_1} : \mathcal{S}' \leqslant_1 \mathcal{S} \Leftrightarrow T(\mathcal{S}') \leqslant_2 T(\mathcal{S}),$$
so the left-hand side diagram in Fig. 1 commutes due to the fact that $T;\ \leqslant_2 = \leqslant_1;\ T$.[4]

[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 $g \circ f$.

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

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

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

$$\begin{tikzcd}
T(\underline{Sig_1}) \arrow[r, "\leq_2"] & T(\underline{Sig_1}) \\
\underline{Sig_1} \arrow[u, "T"] \arrow[r, "\leq_1"] & \underline{Sig_1} \arrow[u, "T"']
\end{tikzcd}
\qquad
\begin{tikzcd}
T(\underline{Sig_1}) \arrow[r, "\underline{\mathrm{pass}}_2"] & \mathrm{TC}(Sig_2) \arrow[d, "T^*"] \\
\underline{Sig_1} \arrow[u, "T"] \arrow[r, "\underline{\mathrm{pass}}_1"] & T^*(\mathrm{TC}(Sig_2))
\end{tikzcd}$$

$$T; \leq_2\, = \,\leq_1; T \qquad\qquad \underline{\mathrm{pass}}_1 \ = \ T; \underline{\mathrm{pass}}_2; T^*$$

Fig. 1. Commuting diagrams reflecting the satisfaction condition.

The following theorem is a direct consequence of [2, 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*

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

Then

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

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

2.3 CSP and Refinement

Normalised Transition Graphs As shown in [9], any finite-state CSP process P can be represented by a *normalised transition graph*

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

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) are in the domain of t. Normalisation of $G(P)$ is reflected by the fact that t is a function. The total function a maps each node to its set of *minimal acceptances*: if $n \in N$ corresponds to a deterministic process state of P, $a(n)$ contains a single acceptance $A \subseteq \Sigma$, and every $e \in A$ is in one-one-correspondence with a transition $t(n, e)$. If n corresponds to a nondeterministic process state, $a(n)$ contains at least two acceptances $A_1, A_2, \ldots, A_k$. This reflects the fact that in a nondeterministic state, P must accept all events of one acceptance $A_i, i \in \{1, \ldots, k\}$, but may refuse all events e from $A_j \setminus A_i, j \neq i$.

Each well-defined transition graph $G(P)$ fulfils the following condition. The union of all minimal acceptances in each node corresponds to the set of events labelling its outgoing transitions.

$$\forall\, n \in N : (n, e) \in \text{dom } t \Leftrightarrow e \in \bigcup a(n) \tag{1}$$

In this condition, dom t denotes the domain of function t.

By construction, normalised transition graphs reflect the failures semantics of finite-state CSP processes: the traces s of a process are exactly the paths through the transition graph, starting at $\underline{n}$. The maximal refusals in each process state P/s are the complements of the minimal acceptances of the node n corresponding to P/s. As a consequences, all failures of P are represented by some (s, R), where s is an initialised path through the transition graph and $R \subseteq (\Sigma - A)$ for some minimal acceptance $A \in a(n)$, such that n is the node corresponding to P/s.

Example 1. Consider CSP process

$$\begin{aligned} P &= a \to (Q \sqcap R) \\ Q &= a \to P \,\square\, c \to P \\ R &= b \to P \,\square\, c \to R \end{aligned}$$

Its transition graph $G(P)$ is shown in Fig. 2. Process state P is represented there as Node_0, with $\{a\}$ as the only acceptance, 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 acceptance sets derived from Q and R. Since these processes accept their initial events with external choice, process $Q \sqcap R$ induces just two minimal acceptance sets $\{a, c\} = [Q]^0$ and $\{b, c\} = [R]^0$. Note that event c can never be refused, since it is a member of all minimal acceptances.

Having engaged into c, the next process state is represented by Node_1. Due to normalisation, there was only a single transition satisfying $t(\text{Node\_2}, c) =$ Node_1. This transition, however, can have been caused by either Q or R engaging into c, so Node_1 corresponds to process state $Q/c \sqcap R/c = P \sqcap R$. This is reflected by the two minimal acceptances labelling Node_1. □

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 [6, 8, 1].

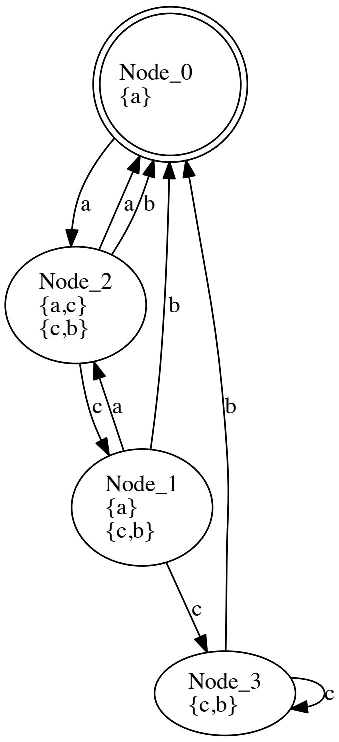

Fig. 2. Normalised transition graph of CSP process P from Example 1.

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, output alphabet Σ_O, where Q, Σ_I, 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, $(q, x, y, q') \in h$ 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 Σ_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 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, $h(q, x, y, q') \wedge h(q, x, y', q'')$ 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 *non-deterministic 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

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

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

The empty trace is denoted by ε, 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 Σ_I and output alphabet Σ_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

$$h_1 : Q \times \Sigma_I \times \Sigma_O \nrightarrow Q; \;\; h_1(q, x, y) = q' \Leftrightarrow h(q, x, y, q')$$

is well-defined. Deterministic FSMs are always observable.

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. [3].

3 Finite Complete Testing Theories 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 output events that represent events that the process really has engaged in and an extra event $\perp$ representing deadlock, if the set of events has been refused.

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

$$\begin{aligned}
Q &= N \cup \{\mathrm{DL}\} \\
\underline{q} &= \underline{n} \\
\Sigma_I &= \mathbb{P}(\Sigma) - \{\varnothing\} \\
\Sigma_O &= \Sigma \cup \{\perp\} \\
h &= \{(n, A, e, n') \mid A \in \Sigma_I \land e \in A \land (n, e) \in \mathrm{dom}\ t \land t(n, e) = n'\} \cup \\
&\quad\ \{(n, A, \perp, \mathrm{DL}) \mid A \in \Sigma_I \land \exists\, A' \in a(n) \land A \cap A' = \varnothing\}
\end{aligned}$$

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

$$\forall\, A : \text{condition}/e$$

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 Node_0 $\longrightarrow$ Node_2 labelled by $\forall\, A : a \in A/a$, for example, stands for FSM transitions

$$\begin{aligned}
&\text{Node\_0} \xrightarrow{\{a\}/a} \text{Node\_2} \\
&\text{Node\_0} \xrightarrow{\{a,b\}/a} \text{Node\_2} \\
&\text{Node\_0} \xrightarrow{\{a,c\}/a} \text{Node\_2} \\
&\text{Node\_0} \xrightarrow{\{a,b,c\}/a} \text{Node\_2}
\end{aligned}$$

□

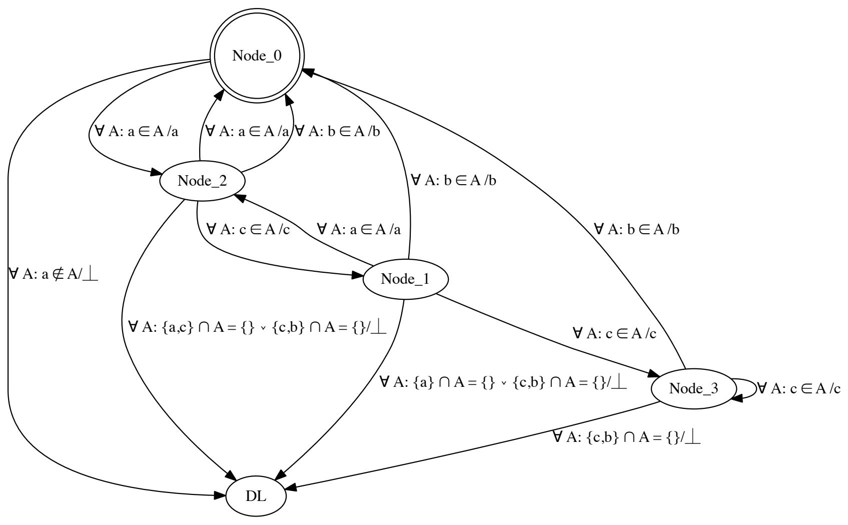

Fig. 3. FSM resulting from applying the model map to CSP process P from Example 1.

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

FSM Test Cases Following [8], an *adaptive FSM test case* $tc_{\text{FSM}} = (Q, \underline{q}, h, \Sigma_I, \Sigma_O, in)$ 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 $in : Q - \{\text{PASS}, \text{FAIL}\} \to \Sigma_I$ maps the states to these inputs. Termination states of the FSM are not labelled with further inputs.

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. □

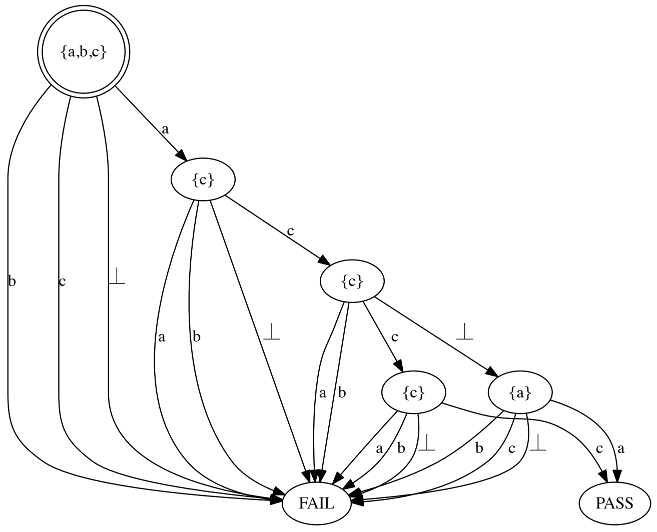

Fig. 4. An FSM test case which is passed by the FSM presented in Example 2.

CSP Test Cases A *CSP test case* is a terminating process with alphabet $\Sigma \cup \{\dagger, \perp, \checkmark\}$, where the extra events stand for (1) test verdict FAIL ($\dagger$), (2) timeout ($\perp$), and (3) test verdict PASS ($\checkmark$). In principle, very general classes of CSP processes can be used for testing, as introduced, for example, in [5, 4]. 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 Σ extended by the events $\{\dagger, \perp, \checkmark\}$ introduced above and the associated FSM input and output alphabets $\Sigma_I = \mathbb{P}(\Sigma) - \{\varnothing\}$ and $\Sigma_O = \Sigma \cup \{\perp\}$. Given an FSM test case $tc_{\mathrm{FSM}} = (Q, \underline{q}, h, \Sigma_I, \Sigma_O, in)$, the image $T^*(tc_{\mathrm{FSM}})$ is the CSP process tc_{CSP} specified as follows.

$$
\begin{aligned}
tc_{\mathrm{CSP}} &= tc(\underline{q}) \\
tc(q) &= \big(e : \{a \in in(q) \mid h_1(q, in(q), e) \notin \{\mathrm{PASS}, \mathrm{FAIL}\}\} \bullet e \to tc(h_1(q, in(q), e))\big) \\
&\quad \Box \\
&\quad \big(e : \{a \in in(q) \mid h_1(q, in(q), e) = \mathrm{PASS}\} \bullet e \to \checkmark \to \mathit{Skip}\big) \\
&\quad \Box \\
&\quad \big(e : \{a \in in(q) \mid h_1(q, in(q), e) = \mathrm{FAIL}\} \bullet e \to \dagger \to \mathit{Skip}\big)
\end{aligned}
$$

4 Case Studies

5 Related Work

6 Conclusion

References

1. 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`
2. 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`
3. 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`
4. Peleska, J., Siegel, M.: Test automation of safety-critical reactive systems. South African Computer Jounal 19, 53–77 (1997)
5. Peleska, J., Siegel, M.: From testing theory to test driver implementation. In: Gaudel, M.C., Woodcock, J. (eds.) FME'96: Industrial Benefit and Advances in Formal Methods, pp. 538–556. No. 1051 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Mar 1996), `http://link.springer.com/chapter/10.1007/3-540-60973-3_106`
6. 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)
7. 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)
8. 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`
9. Roscoe, A.W. (ed.): A Classical Mind: Essays in Honour of C. A. R. Hoare. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)