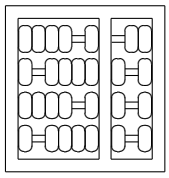

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Necessity and Sufficiency for Checking m-Completeness of Test Suites

Adilson Luiz Bonifacio *Arnaldo Vieira Moura*

Technical Report - IC-13-21 - Relatório Técnico

September - 2013 - Setembro

Necessity and Sufficiency for Checking m-Completeness of Test Suites

Adilson Luiz Bonifacio* Arnaldo Vieira Moura†

Abstract

Test suite generation for Finite State Machines (FSM) has been largely investigated. Some of the previous work in this area found necessary, but not sufficient, conditions for the automatic generation of test suites for this class of models. Yet other set of previous studies obtained sufficient, but not necessary, conditions for the same problem. Many earlier works imposed several conditions upon the specification or on the implementation models. Here, we describe necessary and sufficient conditions for the automatic generation of m-complete test suites when the specification and implementation are modeled as FSMs. Further, we impose only weak *a priori* restrictions on the models, such as determinism and completeness of implementation models. We do not require reduced models nor complete specifications.

1 Introduction

Several approaches for the automatic generation of test suites for models based on the Finite State Machine (FSM) formalism have been proposed in the literature. Many of them focused on the automatic generation of test suites with full fault detection capabilities. Such test suites are named complete [2, 4, 10, 11, 22, 3, 8, 6, 12, 13, 20, 15, 1]. All these methods prove sufficient conditions that guarantee the completeness of the test suites that are generated. Few other works show necessary conditions [17, 23] for the completeness of the test suites that are obtained. An approach that yields necessary and sufficient conditions for the completeness of the test suites has been lacking.

Further, often specifications are required to be reduced models with n states, while the corresponding implementation FSMs are required to have $m \geq n$ states [20]. Test suites extracted from these methods are called m-complete because they can detect any fault on implementations with at most m states. Some approaches are even more restrictive and require $m = n$ [18, 22, 21]. Several methods also assume that the implementation FSM provides a reliable reset operation, which means that a test suite can be described by just one test, formed by the concatenation of smaller test cases. Other approaches require the specification to be completely specified and to have diagnostic sequences able to identify states in order to generate a complete test suite as a single sequence [12, 8, 13, 10, 22].

*Computing Department, University of Londrina, Londrina, Brazil, *email: bonifacio@uel.br*. Supported by FAPESP, process 2012/23500-6.

†Computing Institute, University of Campinas, Campinas, Brazil, *email: arnaldo@ic.unicamp.br*.

Several methods are derived from the so-called W method [3], as is the case of the HSI [19] and the Wp [14] methods. These methods are arranged in two phases. First, they derive tests to check if each state in the specification is also present in the implementation. The second phase then derives tests that explore all transitions in the implementation, produce the correct output and end in the same state as in the specification. Clearly, such methods require the notions of state identification and state distinguishing input sequences. However, sometimes a diagnostic sequence may not exist, given a reduced FSM. Hence, in such cases, methods that do not require the existence of diagnostic sequences must be used [5].

In order to establish their correctness, most of the methods mentioned above lean on sufficient conditions, as shown in [4, 22]. Necessary conditions are treated in a few other works [17]. But, in these cases, other constraints are imposed on the formalisms, such as reducibility and completeness of the specification FSMs, or that the test suite satisfies other properties, such as being a state cover or transition cover, or even constraints on the number of states in the implementation model must be obeyed.

In this paper, we show strict necessary and sufficient conditions for test suite completeness in a more general way. We prove that our approach is stronger than other existing approaches, in the sense that there are infinite families of specifications and corresponding test suites where our method always succeeds in determining the completeness of the test suites. We also argue other proposals require stronger assumptions about the models involved in order to yield positive verdicts for test completeness.

The paper is organized as follows. In Section 2 we survey, with no claim of exhaustiveness, some recent works that more directly discuss the completeness problem for test suites under the FSM formalism. Some basic definitions are given in Section 3. In Section 4 we show necessary and sufficient conditions for test suite completeness. In Section 5 we propose an algorithm to check test suite completeness for FSMs. Finally, in Section 7, we state some concluding remarks.

2 Related Works

In this section, we discuss some more recent works that address necessary or sufficient conditions for test suite completeness. In a general way, these other approaches can be seen as special cases of our work since the latter leads to the less restrict necessary and stronger sufficient conditions for checking completeness of test suites based on FSM models.

In a recent study, Simão and Petrenko [21] present sufficient conditions for test suite completeness. They considered implementation FSMs with at most as many states as in the specification FSM. In this case, the test suite is usually called n-complete, where n is the number of states in the specification. Simão and Petrenko use the notion of confirmed sets. A set of input sequences is said confirmed when its sequences are such that when two of them lead to a same state in the specification, then they also lead to a common state in the implementation. Moreover, using the sequences in the given set, one can reach any state in the specification machine. Simão and Petrenko then establish that a sufficient condition for test completeness is that the given test contains a confirmed subset which is

also a transition cover for the specification. Their approach also requires the specification and implementation machines to be reduced, with the implementation also being complete. The method can determine only n-completeness, and is inconclusive when a test suite does not satisfy these conditions.

Dorofeeva et. al. [4] also show sufficient conditions for test suite completeness. Their approach applies to initially connected FSMs and implementation FSMs are assumed to have a reliable reset operation. The method was also devised as a test for n-completeness due to the reset operations. This method was superseded by that of Simão and Petrenko [21].

Ural et al. [22] also give conditions under which a sequence can be checked for completeness. They define a checking sequence as one that can distinguish a complete strongly connected deterministic and reduced FSM from each other FSM with at most as many states as the original one, and not isomorphic to it. Their method also rely on the existence of a diagnostic sequence [12, 13] and was also argued to be weaker than the method discussed in [21].

In other works [17, 23], certain necessary conditions to check completeness of test suites are presented. The specification and implementation FSMs are required to be complete and reduced besides also being deterministic. They also bound the number of states in implementation FSMs, giving rise to the notion of m-completeness. Moreover, their necessary conditions rely on the notions of state and transition covers. Their strategy is to construct a so-called tree machine, using the given set of defined sequences in the test suite as guides. Then, all the possible reduced forms of the tree machine are obtained, using an algorithm for partial FSM minimization [9, 16]. If at least one of the obtained reduced FSMs is distinguishable from the specification, then the given test is not n-complete; otherwise, the test is n-complete. We notice that these necessary conditions are not necessarily also sufficient conditions for test completeness, even under the required conditions on the specification and implementation machines.

We show that our method is more general than other approaches presented in the literature, and in different ways. First we show that our conditions for test suite completeness are not only sufficient but also necessary. Further we treat the more general notion of m-completeness under necessary and sufficient conditions. Finally, we do need further only the assumption about the completeness of implementation FSMs besides the determinism.

3 Basic concepts

In this section we present some basic concepts and partial results that will be useful later. Let $\mathcal{I}$ be an alphabet. The length of any finite sequence of symbols α over $\mathcal{I}$ is indicated by $|\alpha|$. The set of all sequences of length k over $\mathcal{I}$ is denoted by $\mathcal{I}^k$, while $\mathcal{I}^\star$ names the set of all finite sequences from $\mathcal{I}$. When we write $x_1x_2\cdots x_n \in \mathcal{I}^\star$ $(n \geq 0)$ we mean $x_i \in \mathcal{I}$ $(1 \leq i \leq n)$, unless noted otherwise. For any set C, $\mathcal{P}(C)$ denotes its power set. The empty sequence will be indicated by ε.

Next, we write the definition of a Finite State Machine [7, 21].

Definition 1 (FSM) *A FSM is a system* $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ *where*

- S *is a finite set of* states

- $s_0 \in S$ *is the initial state*
- $\mathcal{I}$ *is a finite set of* input actions *or* input events
- $\mathcal{O}$ *is a finite set of* output actions *or* output events
- $D \subseteq S \times \mathcal{I}$ *is a* specification domain
- $\delta : D \to S$ *is the* transition function
- $\lambda : D \to \mathcal{O}$ *is the* output function □

In order to ease the notation we will use the following conventions:

1. M and N will denote the FSMs $(S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ and $(Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$, respectively.
2. s, q, p, r will indicate states; x, y will indicate input actions; and a, b will indicate output actions. We may also use decorations, like s_1, x' or a'_3.

Let M be a FSM and let $\sigma = x_1 x_2 \cdots x_n \in \mathcal{I}^\star$, $\omega = a_1 a_2 \cdots a_n \in \mathcal{O}^\star$ $(n \geq 0)$. If there are states $r_i \in S$ $(0 \leq i \leq n)$ such that $\delta(r_{i-1}, x_i) = r_i$ and $\lambda(r_{i-1}, x_i) = a_i$ $(1 \leq i \leq n)$, then we may write $r_0 \stackrel{\sigma/\omega}{\to} r_n$. Note that, according to Definition 1, if such states exist, then states r_i are unique $(1 \leq i \leq n)$. That is, all FSMs treated here are deterministic. When the input sequence σ, or the output sequence ω, is not important, then we may write $r_0 \stackrel{\sigma/}{\to} r_n$, or $r_0 \stackrel{/\omega}{\to} r_n$, respectively. If both sequences are not important, we may write $r_0 \to r_n$. In these notations, we can also drop the target state, when it is not important, *e.g.* $r_0 \stackrel{\sigma/\omega}{\to}$ or $r_0 \to$. When we need to indicate the FSM we write $\stackrel{\sigma/\omega}{\underset{M}{\to}}$, and similarly for the other variants of the notation. The function $U : S \to \mathcal{P}(\mathcal{I}^\star)$ will be useful, where $U(s) = \{\sigma \mid s \stackrel{\sigma/}{\to}\}$.

We define $\widehat{\delta}, \widehat{\lambda} : S \to \mathcal{P}(\mathcal{I}^\star)$ by letting $\widehat{\delta}(s, K) = \{r \mid s \stackrel{\sigma/}{\to} r, \text{ for some } \sigma \in K\}$ and $\widehat{\lambda}(s, K) = \{w \mid s \stackrel{\sigma/w}{\to}, \text{ for some } \sigma \in K\}$. We will often write $\widehat{\delta}(s, \sigma) = r$ instead of $\widehat{\delta}(s, \{\sigma\}) = \{r\}$ and, when there is no ambiguity, we may write δ instead of $\widehat{\delta}$. Similarly for $\widehat{\lambda}$.

Next, we state some properties of FSMs that will be important throughout the paper.

Definition 2 (Reachability) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM, $V \subseteq S$ and $r \in S$. We say that r is* reachable from V *iff*[1] *$s \stackrel{\sigma/}{\to} r$ for some $s \in V$ and $\sigma \in \mathcal{I}^\star$. We say that r is* reachable *iff it is reachable from s_0.* □

Now we say what it means for a FSM to be complete.

Definition 3 (Complete FSMs) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM. Then M is* complete *iff for all reachable $s \in S$ and all $x \in \mathcal{I}$ we have $(s, x) \in D$.* □

[1] We write 'iff' for 'if and only if'.

Thus M is complete if $D = R \times \mathcal{I}$ where R is the subset of reachable states. If M is not complete, we say that we have a partial FSM. We can always modify a FSM $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ in order to get a new complete FSM. Just choose a new state $t \notin S$, a new output action $a \notin \mathcal{O}$ and extend the transition and output functions of M according to $s \xrightarrow{x/a} t$ when $(s, a) \notin D$, and according to $t \xrightarrow{x/a} t$ for all $x \in \mathcal{I}$.

Now we define distinguishability and equivalence.

Definition 4 (Distinguishability and equivalence) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ and $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$ be FSMs and let $s \in S$, $q \in Q$. Let $C \subseteq \mathcal{I}^\star$. We say that s and q are* C-distinguishable *iff $\lambda(s, \sigma) \neq \tau(q, \sigma)$ for some $\sigma \in U(s) \cap U(q) \cap C$, denoted $s \not\approx_C q$. Otherwise, s and q are* C-equivalent*, denoted $s \approx_C q$. We say that M and N are* C-distinguishable *iff $s_0 \not\approx_C q_0$, and they are* C-equivalent *iff $s_0 \approx_C q_0$.* □

When C is not important, or when it is clear from the context, we might drop the index. When there is no mention to C, we understand that we are taking $C = \mathcal{I}^\star$. Note that the notions of distinguishability and equivalence also apply when M and N are the same FSM.

We can now recall the notion of reduced FSMs.

Definition 5 (Reducibility) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM. Then M is* reduced *iff every state is reachable and every pair of distinct states in S are distinguishable.* □

Now we are in a position to define test cases and test suites.

Definition 6 (Tests) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM. A* test suite for M *is any finite subset of $U(s_0)$. Any element of a test suite is a* test case*. The* length *of a test suite is the sum of the lengths of all its test cases.* □

Since a test suite must be applied from initial states, implementations under test must be brought to their initial states before the application of each test case. This can be achieved by a reset operation, which always brings the machine back to its initial state [4, 3].

Next we introduce the concept of m-complete test suites.

Definition 7 (m-complete test suite) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM. A test suite T is* m-complete for M *iff for any complete FSM $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$ with $|Q| \leq m$, if $M \approx_T N$ then $M \approx N$.* □

That is, for any complete implementation under test N with at most m states, if M and N are T-equivalent, then they are equivalent. A common case occurs when we also have $|S| = m$.

4 Test suite completeness

In this section we show necessary and sufficient conditions for test suite completeness. In order to obtain necessary and sufficient conditions for m-complete test suites, we start with the basic notion of a simulation.

Definition 8 (Simulation) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ and $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$ be FSMs. We say that a relation $R \subseteq S \times Q$ is a* simulation *(*of M by N*) iff $(s_0, q_0) \in R$, and whenever we have $(s, q) \in R$ and $s \xrightarrow[M]{x/a} r$, then there is a state $p \in Q$ such that $q \xrightarrow[N]{x/a} p$ and with $(r, p) \in R$.* □

The following result will be useful to demonstrate the existence of simulation relations.

Lemma 1 (State equivalence) *Consider the FSMs $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ and $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$. Let $n \geq 1$, $s_i \in S$, $p_i \in Q$ $(1 \leq i \leq n)$ and $x_i \in \mathcal{I}$, $a_i \in \mathcal{O}$, $b_i \in \mathcal{O}'$ $(1 \leq i < n)$ be such that $s_i \xrightarrow{x_i/a_i} s_{i+1}$ and $p_i \xrightarrow{x_i/b_i} p_{i+1}$ $(1 \leq i < n)$. Assume further that $s_1 \approx p_1$. Then $s_i \approx p_i$ $(1 \leq i \leq n)$ and $a_1 a_2 \cdots a_{n-1} = b_1 b_2 \cdots b_{n-1}$.*

Proof Let $\sigma = x_1 x_2 \cdots x_{n-1}$, $\omega_1 = a_1 a_2 \cdots a_{n-1}$ and $\omega_2 = b_1 b_2 \cdots b_{n-1}$. We clearly have $s_1 \xrightarrow{\sigma/\omega_1} s_n$ and $p_1 \xrightarrow{\sigma/\omega_2} p_n$. But, since $s_1 \approx p_1$ Definition 4 immediately gives $\omega_1 = \omega_2$.

To see that $s_i \approx p_i$ $(1 \leq i \leq n)$ we go by induction on $n \geq 1$. The basis is trivial and we proceed with the induction step. Let $1 \leq k < n$ and assume $s_k \approx p_k$. If $s_{k+1} \not\approx p_{k+1}$ then we must have $\rho \in \mathcal{I}^\star$, $\varphi_1 \in \mathcal{O}^\star$, $\varphi_2 \in \mathcal{O}'^\star$, with $\varphi_1 \neq \varphi_2$, and such that $s_{k+1} \xrightarrow{\rho/\varphi_1} s'$ and $p_{k+1} \xrightarrow{\rho/\varphi_2} p'$, for some states $s' \in S$ and $p' \in Q$. Let $\eta = x_k \rho$, $\phi_1 = a_k \varphi_1$ and $\phi_2 = b_k \varphi_2$. We get $s_k \xrightarrow{\eta/\phi_1} s'$ and $p_k \xrightarrow{\eta/\phi_2} p'$. Since $a_k \varphi_1 \neq b_k \varphi_2$ we have a contradiction to $s_k \approx p_k$, thus extending the induction. □

The next lemma states half of our desired result.

Lemma 2 (Simulation relation) *Let T be a m-complete test suite for a FSM $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$. Then any T-equivalent and complete FSM with at most m states is able to simulate M.*

Proof Let $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$ be a complete FSM with at most m states such that $M \approx_T N$. Define a relation $R \subseteq S \times Q$ by letting $(s, q) \in R$ iff $\delta(s_0, \alpha) = s$ and $\mu(q_0, \alpha) = q$ for some $\alpha \in \mathcal{I}^\star$, $s \in S$ and $q \in Q$.

Since $\delta(s_0, \varepsilon) = s_0$ and $\mu(q_0, \varepsilon) = q_0$ we get $(s_0, q_0) \in R$.

Now assume $(s, q) \in R$ and let $s \xrightarrow{x/a} r$ for some $r \in S$, $x \in \mathcal{I}$ and $a \in \mathcal{O}$. Since T is m-complete for M, Definition 7 gives $M \approx N$. Hence, $s_0 \approx q_0$. Since $(s, q) \in R$, the construction of R gives some $\alpha \in \mathcal{I}^\star$ such that $\delta(s_0, \alpha) = s$ and $\mu(q_0, \alpha) = q$. Composing, we get $\delta(s_0, \alpha x) = \delta(s, x) = r$. Since N is complete, we get $\mu(q, x) = p$, for some $p \in Q$. Then $\mu(q_0, \alpha x) = \mu(q, x) = p$.

Recalling, we have $\delta(s_0, \alpha) = s$, $\mu(q_0, \alpha) = q$ and $s_0 \approx q_0$. Using Lemma 1 we conclude that $s \approx q$, and then we must have $\lambda(s, x) = \tau(q, x)$. So, from $(s, q) \in R$ and $s \xrightarrow{x/a} r$ we obtained $p \in Q$ such that $q \xrightarrow{x/a} p$. Finally, we note that we also have $\mu(q_0, \alpha x) = p$ and $\delta(s_0, \alpha x) = r$. The definition of R now gives $(r, p) \in R$. This shows that R is a simulation relation, concluding the proof. □

We now show the converse, that is, if there is a simulation of M by any complete FSM that it is T-equivalent to it, then the m-completeness of T follows.

Lemma 3 (Test completeness) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM and let T be a test suite for M. Assume that any complete FSM with at most m states and T-equivalent to M is able to simulate M. Then T is m-complete for M.*

Proof We proceed by contradiction. Assume that T is not m-complete for M. Then, by Definition 7, there exists a complete T-equivalent FSM $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$ with at most m states and such that $M \not\approx N$. Using Definition 4, we get an input sequence $\sigma = x_1 \dots x_n$ $(n \geq 0)$ and an input symbol $y \in \mathcal{I}$ such that $\lambda(s_0, \alpha) = \tau(q_0, \alpha)$ and $\lambda(s_0, \alpha y) \neq \tau(q_0, \alpha y)$. Clearly, there are $s_i \in S$, $q_i \in Q$ $(1 \leq i \leq n)$ such that $\delta(s_{i-1}, x_i) = s_i$ and $\mu(q_{i-1}, x_i) = q_i$ with $\lambda(s_{i-1}, x_i) = \tau(q_{i-1}, x_i)$ $(1 \leq i \leq n)$. Further, we get $s \in S$ and $q \in Q$ such that $\delta(s_n, y) = s$, $\mu(q_n, y) = q$ and $\lambda(s_n, y) \neq \tau(q_n, y)$.

The hypothesis gives a relation $R \subseteq S \times Q$ that is a simulation of M by N.

Claim: $(s_i, q_i) \in R$ $(0 \leq i \leq n)$.
Proof of the Claim. We go by induction on $i \geq 0$.

Basis: we get $(s_0, q_0) \in R$ directly from Definition 8.

Induction step: assume that $(s_i, q_i) \in R$ for some $i < n$. Since $\delta(s_i, x_{i+1}) = s_{i+1}$, Definition 8 gives a $q \in Q$ such that $\mu(q_i, x_{i+1}) = q$, $\lambda(s_i, x_{i+1}) = \tau(q_i, x_{i+1})$ and $(s_{i+1}, q) \in R$. But $\mu(q_i, x_{i+1}) = q_{i+1}$ and, since N is deterministic, we get $q = q_{i+1}$. Thus $(s_{i+1}, q_{i+1}) \in R$ extending the induction and establishing the Claim. △

Using the Claim, we get $(s_n, q_n) \in R$. Now, since $\delta(s_n, y) = s$, Definition 8 gives a $p \in Q$ such that $(s, p) \in R$, $\mu(q_n, y) = p$, and $\lambda(s_n, y) = \tau(q_n, y)$, reaching a contradiction. □

Putting together the previous results we obtain necessary and sufficient conditions for the m-completeness of test suites.

Theorem 1 (Necessity and sufficiency conditions) *Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be a FSM and T be a test suite for M. Then, T is m-complete for M iff any T-equivalent and complete FSM with at most m states is able to simulate M.*

Proof Immediate using Lemmas 2 and 3. □

5 Checking completeness

In this section we describe a process for checking test suite completeness using Theorem 1. The process consists of two phases, described in the next subsections.

5.1 Growing a T-tree

Given a test suite T and a FSM M, we first construct a layered structure, called a T-tree for M. In a T-tree, each node represents a FSM that is being constructed. We can also say that a node contains a table describing the transition and output functions of the FSM that is represented at the node. In addition, each node is labeled by a pair of states (s, q) where s is a state of M and q is a state of the FSM represented at the node. Also, we assume that

we have a specified upper bound of $m \geq 1$ on the number of states of any implementation that will be put under test.

We first describe how to apply the tree-growing process, given a T-tree G with maximum level k, and given that the current test is $\sigma = x\rho$, where x is an input symbol. We proceed in a breadth-first fashion. For each node at level k in G in turn, say in a left-to-right order, we repeat the following cycle. Let u be the next node at level k to be processed, let (s, q) be its label and let N be the FSM it represents. Assume that we have $s \stackrel{x/a}{\rightarrow} r$ in the specification M. If we have a transition $q \stackrel{x/b}{\rightarrow} p$ in N, with $a \neq b$, then we are done with node u and we proceed to the next node of G at level k. Otherwise node u will get several descendants. For each state p of N we add a descendant v to u in the tree G. The new node v is labeled (r, p) and the FSM it represents is obtained by extending the transition and output functions in N according to $q \stackrel{x/a}{\rightarrow} p$. In addition, if the number of states in N is less than the bound m, we add another descendant w to node u in G. The new node w is labeled (r, z), where z is a new state not already in N. The FSM represented by the new node w is obtained by adding state z to N and extending its transition and output functions according to $q \stackrel{x/a}{\rightarrow} z$. When all nodes in G at level k have been considered, we declare ρ to be the new current test and repeat the breadth-first tree-growing procedure, now taking the nodes in G at level $k + 1$. This growing process continues by taking one symbol of the current test at a time and growing one level in the tree. It ends when the current test reduces to the ε, *i.e.* the empty string. When that happens the maximum level in G would be $k + |\sigma|$.

Now suppose that the breadth-first growing process has just ended because the current test reduced to ε. Suppose that the T-tree is now G, with maximum level k. We have to restart the tree-growing process. We do so by taking another, yet not considered, test in $t \in T$ and make it the current test. We also relabel all nodes with (s_0, q_0) at level k since a new test must be exhausted from initial states of M and N. If all tests in T have already been constructed, the T-tree for M is complete and the whole procedure terminates. Otherwise, we just restart the growing of G, with maximum level k, and with t as the new current test.

Initially, the T-tree is formed just by a root node, labeled (s_0, q_0), that represents the trivial FSM whose only state is q_0 and with empty transition and output functions. We say that this node is at level zero. We choose a test in T and declare it the current test. The tree-growing process is ready to start for the first time. An example illustrating this processes appears in Section 6.

We remark that the growing procedure cannot end prematurely, since we are assuming $T \subseteq U(s_0)$, where s_0 is the initial state in M. In fact, when the whole growing process terminates, the complete T-tree for M will have maximum level $\sum_{\sigma \in T} |\sigma|$. Further, when the procedure terminates, its leaves at maximum level represent FSMs that are T-equivalent to M. Moreover, any FSM that happens to be T-equivalent to M will be an extension of a machine represented in one of these leaves.

Here, the first phase for checking completeness terminates.

5.2 Checking for simulations

In the last phase of the process for checking completeness of test suites, we must check whether a complete candidate implementation FSM with at most m states can simulate a given specification FSM. Let $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ be the given specification.

We will build a set $\mathcal{M}$ of pairs (N, R) where N is a FSM and R is a candidate simulation relation (of M by N). We start with an initial set $\mathcal{M}$ containing all pairs (N, R), where N is a machine represented at a leaf of the T-tree and R contains just the unmarked pair (s_0, q_0), with q_0 the initial state of N.

Now, while there are pairs in $\mathcal{M}$, we proceed as follows. Remove a pair (N, R) from $\mathcal{M}$ and examine it as described below. If all pairs in $\mathcal{M}$ are exhausted, the procedure terminates successfully.

Let (N, R) be the pair to be examined, where $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$. While there remains unmarked pairs in R, we repeat the following cycle. Take any unmarked pair (s, q) in R and mark it as visited. For all input symbols $x \in \mathcal{I}$ such that $(s, x) \in D$, if $(q, x) \in D'$ but $\lambda(s, x) \neq \tau(q, x)$, immediately terminate the whole procedure unsuccessfully. When $(q, x) \in D'$ and $\lambda(s, x) = \tau(q, x)$, then add the pair $(\delta(s, x), \mu(x, s))$ to R as an unmarked pair, if it not already there. Finally, if $(q, x) \notin D'$ then we must insert new pairs in $\mathcal{M}$ as follows. Let $\delta(s, x) = r$ and $\lambda(s, x) = a$. For each state $p \in Q$, extend N according to $q \xrightarrow{x/a} p$, thus obtaining a new FSM N', and extend R by adding (r, p) to it as an unmarked pair, thus obtaining a new relation R'. Then, add the pair (N', R') to $\mathcal{M}$ and stop the cycle for the pair (N, R).

This is the last phase of the entire procedure to check completeness of a test suite T for a given specification M. If this last phase terminates unsuccessfully, then we deem T as not m-complete for M, where m is the upper bound on the number of states of any implementation that will be put under test, as assumed at the start of the first phase. If this last phase terminates successfully, we declare that T is m-complete for M. The whole process is supported by Theorem 1.

6 An example

We illustrate the procedure describe in Section 5. Let M be the FSM M depicted in Figure 1 and let $T = \{00, 1000\}$ be a test suite for M. Assume that we are treating implementation machines with up to $m = 3$ states. A T-tree for M is partially illustrated in Figure 2.

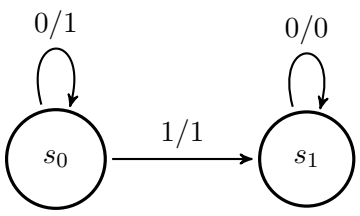

Figure 1: FSM specification M.

We start with a root labeled (s_0, q_0), at level zero, representing a trivial machine Z_0 with the empty transition and output functions. Let $\sigma = 00$ be the first test case chosen from T. It is now the current test case. Since the first input symbol of σ is a 0, we get the two descendant nodes at level one. The first corresponds to extending Z_0 according to $q_0 \stackrel{0/1}{\rightarrow} q_0$ in Z_0, and the second corresponds to adding a new state q_1 to Z_0, and extending it as dictated by $q_0 \stackrel{0/1}{\rightarrow} q_1$. The current test case is now reduced to $\sigma = 0$, and we proceed by considering all nodes at level 1, in turn. The leftmost node at level 1, says node u, represents a machine Z_1, as indicated in the figure. Since we already have $q_0 \stackrel{0/}{\rightarrow} q_0$ in Z_1, node u gets a single descendant that also represents Z_1. Its label is obtained from $s_0 \stackrel{0/1}{\rightarrow} s_0$ in machine M and $q_0 \stackrel{0/1}{\rightarrow} q_0$ in machine Z_1. Next, we examine the second node at level 1, say node w, representing a machine Z_2. Since we do not have $q_1 \stackrel{0/}{\rightarrow}$ in Z_2, we must generate new descendants for w by extending Z_2 according to $q_1 \stackrel{0/1}{\rightarrow} q$ for all states $q \in \{q_0, q_1\}$ that are already in Z_2. These are the leftmost two descendants of w. Further, since Z_2 has two states and we are treating implementation FSMs with at most $m = 3$ states, node w gets another descendant by adding a new state q_2 to Z_2 and extending it as required by $q_1 \stackrel{0/1}{\rightarrow} q_2$. This is the rightmost descendant of w.

At this point we have reduced the current test to $\sigma = \varepsilon$. We must now take another test case from T. The last test case in T that was not already used is $\sigma = 1000$ and we make it the current test case. Now, because we are restarting the tree-growing process anew with a new test case, we must relabel all nodes at level 2 with the root label, (s_0, q_0). That is why all nodes at level 2 in Figure 2 appear so relabeled.

We proceed by considering all nodes at level 2, in turn, with the current test case as $\sigma = 1000$. We start with the leftmost node at level 2, say node v, also representing machine Z_1. Since we do not have $q_0 \stackrel{1/}{\rightarrow}$ in Z_1, we must generate new descendants for node v by extending Z_1 according to $q_0 \stackrel{1/1}{\rightarrow} q_0$ in Z_1 and by adding a new state q_1 to Z_1 according to $q_0 \stackrel{1/1}{\rightarrow} q_1$. Next, we consider the second node at level 2, say node r, representing a machine Z_3, as indicated in the figure. Since we do not have $q_0 \stackrel{1/}{\rightarrow}$ in Z_3 we must generate new descendants for r by extending Z_3 according to $q_0 \stackrel{1/1}{\rightarrow} q$ for all states $q \in \{q_0, q_1\}$ that are already in Z_3, and also by adding a new state q_2 to Z_3 and further extending it according to $q_0 \stackrel{1/1}{\rightarrow} q_2$. In order to keep the figure size under control, we suppressed the tree-growing process at the remaining two nodes at level 2, marked as * and †. The current test case will be reduced again to $\sigma = 000$, and we restart the growing process at level 3. When we consider the leftmost node at level 3, labeled (s_1, q_0), representing machine Z_4, we notice that we have $s_1 \stackrel{0/0}{\rightarrow} s_1$ in machine M but $q_0 \stackrel{0/1}{\rightarrow} q_0$ in Z_4. This indicates an incompatibility and we terminate the growing process at this node, leaving it behind.

The growing process continues up to when we have exhausted the current test case $\sigma = 000$. When the procedure terminates we will have obtained six machines that are T-equivalent to M. Machines N_0 to N_3 are indicated in the figure. A fifth machine would result from the subtree rooted at node marked by a *, and a sixth one from the subtree

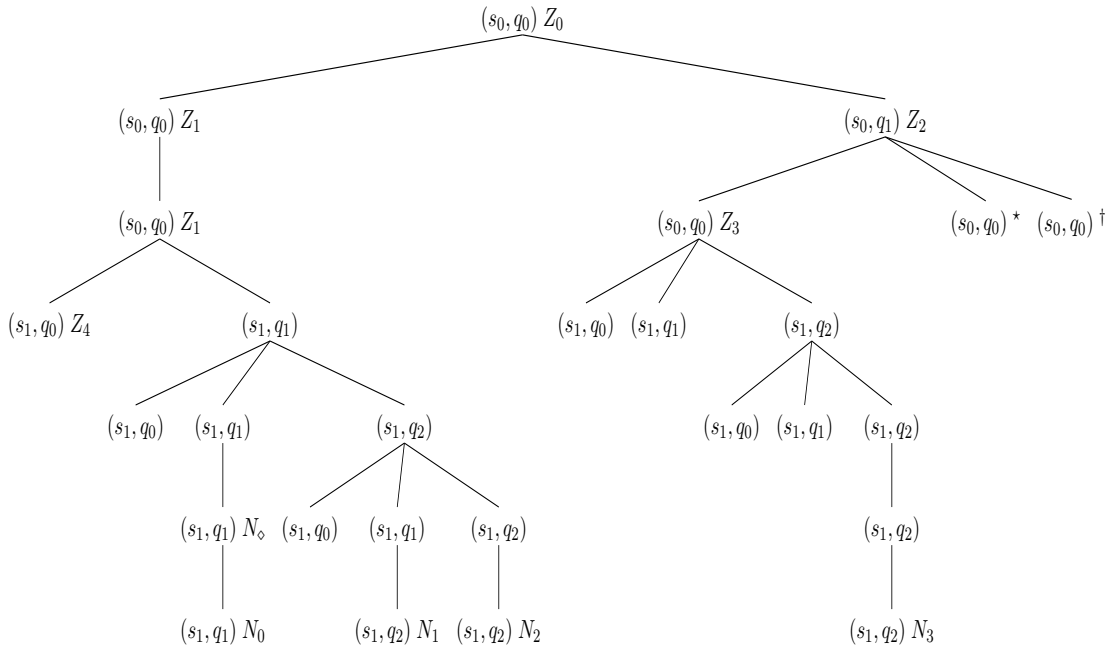

Figure 2: T-tree for FSM depicted in Figure 1.

rooted at node marked by a †. We note that machine N_0 is isomorphic to M, whereas machines N_1, N_2 and N_3 are not, since they have 3 states.

On the second phase we start with a set $\mathcal{M} = \{(N_i, R_i) \mid 0 \leq i \leq 5\}$, where R_i contains only the initial pair of states for M and N_i. We illustrate the procedure by examining the pair (N_1, R_1). Machine N_1 can be read from the T-tree in Figure 2, and is depicted in Figure 3. Initially, R_1 contains just (s_0, q_0), unmarked. We immediately mark it as visited and proceed. For the input symbol 0 we get $s_0 \xrightarrow{0/1} s_0$ and $q_0 \xrightarrow{0/1} q_0$. Since the pair (s_0, q_0) is already in R, we consider the other input symbol, 1. We have $s_0 \xrightarrow{1/1} s_1$ and $q_0 \xrightarrow{1/1} q_1$. Since (s_1, q_1) is not in R, we add it to R as an unmarked pair. Now, $R = \{(s_0, q_0)^\ddagger, (s_1, q_1)\}$, where we indicate marked pairs by a ‡. We mark (s_1, q_1) as visited and consider the input symbols again. We have $s_1 \xrightarrow{0/0} s_1$ in M and $q_1 \xrightarrow{0/0} q_2$ in N_1. Because (s_1, q_2) is not in R, we add it to R as unmarked. Since $(s_1, 1) \notin D$ we are done with the pair (s_1, q_1). Now, $R = \{(s_0, q_0)^\ddagger, (s_1, q_1)^\ddagger, (s_1, q_2)\}$. We mark (s_1, q_2) as visited and need only consider the input symbol 0, since $(s_1, 1) \notin D$. We get $s_1 \xrightarrow{0/0} s_1$ in M and $q_2 \xrightarrow{0/0} q_1$ in N_1. Since (s_1, q_1) is already in R, it remains as is. Now, $R = \{(s_0, q_0)^\ddagger, (s_1, q_1)^\ddagger, (s_1, q_2)^\ddagger\}$. We have no more unmarked pairs in R and the examination of the pair (N_1, R_1) terminates without reaching a conflict.

For another, contrasting, situation consider the last leaves indicated in Figure 2. The machine represented by that node, N_3, can be extracted from the T-tree and is depicted in Figure 4. So, the pair (N_3, R_3) would also be inserted into the set $\mathcal{M}$, with $R_3 = \{(s_0, q_0)\}$. After marking (s_0, q_0) as visited and considering the input symbols 0 and 1, we would

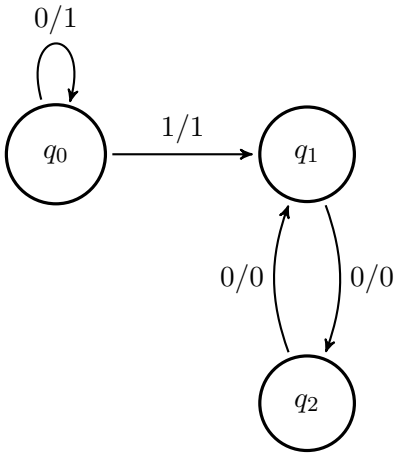

Figure 3: FSM N_1.

have $R_3 = \{(s_0, q_0)^{\ddagger}, (s_0, q_1), (s_1, q_2)\}$. We now mark (s_0, q_1) as visited. With the input symbol 0 no new pairs would be add to R_3, since we have $s_0 \stackrel{0/1}{\rightarrow} s_0$ in M, $q_1 \stackrel{0/1}{\rightarrow} q_0$ in N_3 and the pair (s_0, q_0) is already in R_3. But when we consider the input symbol 1, we find that we have $s_0 \stackrel{1/1}{\rightarrow} s_1$ in M, but we do not have $q_1 \stackrel{1/}{\rightarrow}$ in N_3. We extend N_3 according to $q_1 \stackrel{1/1}{\rightarrow} q_j$, thus generating three new machines $N_{3,j}$ $(j = 0, 1, 2)$. The companion relation sets would be $R_{3,j} = \{(s_0, q_0)^{\ddagger}, (s_0, q_1)^{\ddagger}, (s_1, q_j)\}$ $(j = 0, 1, 2)$ and we would include in $\mathcal{M}$ the pairs $(N_{3,j}, R_{3,j})$ $(j = 0, 1, 2)$. In its due turn, the pair $(N_{3,1}, R_{3,1})$ would be selected for examination. Machine $N_{3,1}$ is illustrated in Figure 5, and we have $R_{3,1} = \{(s_0, q_0)^{\ddagger}, (s_0, q_1)^{\ddagger}, (s_1, q_1)\}$. Now, we mark (s_1, q_1) as visited in $R_{3,1}$ and consider

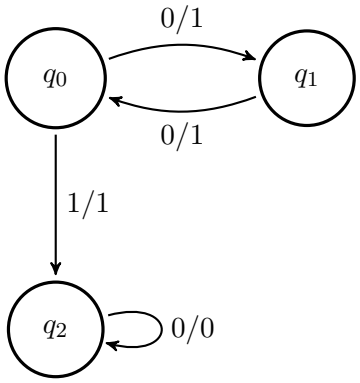

Figure 4: FSM N_3.

the input symbols. With 0 we have $s_1 \stackrel{0/0}{\rightarrow} s_1$ in M and $q_1 \stackrel{0/1}{\rightarrow} q_0$ in $N_{3,1}$. This is a conflict and the whole procedure for checking the 3-completeness of T for M terminates unsuccessfully. We, thus, declare T not to be 3-complete for M, as expected.

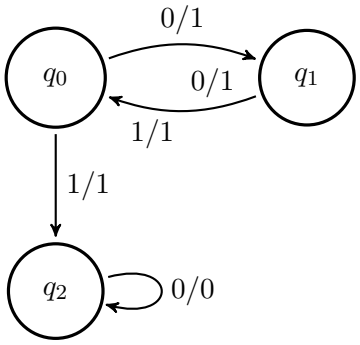

Figure 5: FSM $N_{3,1}$.

Suppose now that we have the same specification M illustrated in Figure 1, but were satisfied to test implementations with at most two states, and using the same test suite $T = \{00, 1000\}$. In this case we set $m = 2$. Then phase one of the procedure for checking m-completeness would generate a sub-tree of the T-tree illustrated in Figure 2. The differences would be that the last at levels 2 and 3 would not appear, as well as the third node from left to right at level 4. This is because all these nodes represent machines with three states, while the remaining ones represent machines with at most 2 states. In this case, after the tree-growing process terminates, we would have a single leaf in the T-tree representing machine N_0, as indicated in Figure 2. Then, phase two would start with $\mathcal{M} = \{(N_0, R)\}$, with $R = \{(s_0, q_0)\}$ and with q_0 being the start state in N_0. We would then remove the pair (N_0, R) from $\mathcal{M}$ and would start to examine it. But N_0, when extracted from the T-tree, is clearly seen to be isomorphic to M, and so a simulation relation obviously exist between M and N_0. We would return to $\mathcal{M}$ for another pair, but now we have $\mathcal{M} = \emptyset$. At this point the m-completeness checking procedure would terminate successfully and we would declare T to be 2-complete for M.

Thus we have a situation where a test suite $T = \{00, 1000\}$ is 2-complete, but not 3-complete, for the specification FSM illustrated in Figure 1.

In fact, had we considered the test suite $T' = \{00, 100\}$ when testing for 2-completeness of the same specification M, the final verdict would indicate that T' is also 2-complete for M. This is because the tree-growing process would construct the same sub-tree as before, except that now the only leaf would have been the machine represented by the leftmost node at level 5 in Figure 1, there marked as $N_\diamond$. But $N_\diamond$ is the same as machine N_0, which, as note before, turns out to be isomorphic to M. Hence the second phase would also terminate successfully in this case, and we would also declare T' to be 2-complete for M. Clearly, T' is not m-complete for M, for any $m \geq 3$, since T is not 3-complete for M and all tests in T' are prefixes of tests in T.

7 Conclusions

In this paper, we presented necessary and sufficient conditions for verifying m-completeness of test suites for specification FSMs. We remark that our method imposes no restriction on the specification machine nor in the implementation under test, except for determinism and that the latter be complete.

We proved by a rigorous argument that our method is correct. Using this formal support, we also indicated how to proceed to give definitive answers about the m-completeness of given test suites. The procedure described always terminates with a correct yes or no answer, that is, it is never inconclusive.

Future research might consider implementing the suggested procedure, or a similar algorithm based on Theorem 1, so that its time and space complexities could be measured when taking typical families of specifications and implementations as input instances.

References

[1] Adilson Luiz Bonifacio, Arnaldo Vieira Moura, and Adenilso da Silva Simão. Model partitions and compact test case suites. *Int. J. Found. Comput. Sci.*, 23(1):147–172, 2012.

[2] Rachel Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[3] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978.

[4] Rita Dorofeeva, Khaled El-Fakih, and Nina Yevtushenko. An improved conformance testing method. In *FORTE*, pages 204–218, 2005.

[5] N. V. Evtushenko and A. F. Petrenko. Synthesis of test experiments in some classes of automata. *Autom. Control Comput. Sci.*, 24(4):50–55, April 1991.

[6] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, June 1991.

[7] A. Gill. *Introduction to the theory of finite-state machines.* McGraw-Hill, New York, 1962.

[8] G. Gonenc. A method for the design of fault detection experiments. *IEEE Trans. Comput.*, 19(6):551–558, 1970.

[9] Sezer Gören and F. Joel Ferguson. On state reduction of incompletely specified finite state machines. *Comput. Electr. Eng.*, 33(1):58–69, January 2007.

[10] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design, 11-13 November 1964, Princeton, New Jersey, USA*, pages 95–110. IEEE, 1964.

[11] R. M. Hierons. Separating sequence overlap for automated test sequence generation. *Automated Software Engg.*, 13(2):283–301, 2006.

[12] Robert M. Hierons and Hasan Ural. Reduced length checking sequences. *IEEE Trans. Comput.*, 51(9):1111–1117, September 2002.

[13] Robert M. Hierons and Hasan Ural. Optimizing the length of checking sequences. *IEEE Trans. Comput.*, 55(5):618–629, May 2006.

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

[15] Lucio Felipe de Mello Neto and A Simao. Test suite minimization based on fsm completeness sufficient conditions. In *Proceedings of 9th IEEE Latin-American Test Workshop*, volume 1, pages 93–98, 2008.

[16] Jorge M. Pena and Arlindo L. Oliveira. A new algorithm for the reduction of incompletely specified finite state machines. In *In ICCAD, 482-489*, pages 482–489. IEEE Computer Society Press, 1998.

[17] A. Petrenko and G. V. Bochmann. On fault coverage of tests for finite state specifications. *Computer Networks and ISDN Systems*, 29:81–106, 1996.

[18] Alex Petrenko and Nina Yevtushenko. On test derivation from partial specifications. In *In FORTE*, pages 85–102, 2000.

[19] Alexandre Petrenko and Gregor v. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In Gang Luo, editor, *IWPTS '94: 7th IFIP WG 6.1 international workshop on protocol test systems*, pages 95–110, London, UK, UK, 1995. Chapman & Hall, Ltd.

[20] Alexandre Petrenko and Nina Yevtushenko. Testing from partial deterministic fsm specifications. *IEEE Trans. Comput.*, 54(9):1154–1165, September 2005.

[21] Adenilso da Silva Simao and Petrenko Petrenko. Checking completeness of tests for finite state machines. *IEEE Trans. Computers*, 59(8):1023–1032, 2010.

[22] Hasan Ural, Xiaolin Wu, and Fan Zhang. On minimizing the lengths of checking sequences. *IEEE Trans. Comput.*, 46(1):93–99, January 1997.

[23] Ming Yu Yao, Alexandre Petrenko, and Gregor von Bochmann. Fault coverage analysis in respect to an fsm specification. In *INFOCOM*, pages 768–775, 1994.