Information and Software Technology 000 (2017) 1–10

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Experimental comparison of approaches for checking completeness of test suites from finite state machines[☆]

Adilson Bonifacio[a,*], Arnaldo Moura[b], Adenilso Simao[c]

[a] Computing Department, University of Londrina, Londrina, Brazil
[b] Computing Institute, University of Campinas, Campinas, Brazil
[c] Institute of Mathematics and Computer Science, University of São Paulo, São Carlos, Brazil

ARTICLE INFO

Article history:
Received 16 November 2016
Revised 9 July 2017
Accepted 23 July 2017
Available online xxx

Keywords:
Test suite completeness
Experiments
Completeness conditions
Confirmed sets
Simulation relations

ABSTRACT

Context: Many approaches have been proposed for checking test suite completeness for Finite State Machines (FSMs). Some approaches provide sufficient conditions whereas others give necessary and sufficient conditions for test suite completeness. One method, called the *CONF* method, is based on sufficient conditions, and relies on a search for confirmed sets when checking completeness. If a confirmed set cannot be found, then the outcome is inconclusive. Another method, the *SIM* method, is based on the notion of simulation relations, and relies on necessary and sufficient conditions when checking test suite completeness. The *SIM* method always returns conclusive verdicts about suite completeness.
Objective: In this work, we describe experimental results comparing these two methods. We also investigate when both methods can be combined for checking completeness of test suites.
Method: We evaluate both strategies according to different parameters of the FSMs, such as the number of states and the number of transitions in the FSM models, the size of input and output alphabets of the FSM models, as well as the size of the test suites. We also report on the relative rates of conclusive and inconclusive verdicts when using both methods.
Results: We see that these methods are complementary, which allows for a combined strategy: the *CONF* method is the fastest in terms of processing time, while the *SIM* method is not as scalable in terms of the size of the specifications.
Conclusion: The experimental results indicated a substantial difference for the rate of positive verdicts obtained by the *SIM* method when compared with the number of positive answers returned by the *CONF* method.

1. Introduction

Test generation from Finite State Machines is a long-standing problem, with many contributions throughout the last decades [1–4]. The main goal is to obtain test suites that are complete with respect to a set of faulty candidate FSMs [5–10]. Usually, the completeness of the generated test suites is demonstrated based on sufficient conditions, i.e., conditions that, once satisfied, ensure that all faulty implementations will be detected [3,11]. Thus, the completeness of the methods is guaranteed by construction by checking that the method always produces a test suite that satisfies the sufficient conditions. In other situations, the generation method may not guarantee that the test suites thus produced are always complete. In these cases, it will be necessary to check completeness a posteriori.

A method that relies on sufficient conditions for checking suite completeness was proposed by Simao and Petrenko [12]. They introduce the notion of confirmed sets. A set of input sequences T is confirmed with respect to a test suite T and a FSM M if input sequences in T leading to a same state in M also lead to a same state in any FSM that has the same output responses as M does to test cases in T, and which has as many states as M. If one can find a confirmed set which includes the empty string and whose sequences are enough to exercise each transition in M, then the test suite is guaranteed to be complete for M. Their method also requires implementation candidates to be completely specified FSMs, and which have at most as many states as the specification M. If a confirmed set cannot be found, then the method is inconclusive about test suite completeness.

By contrast, Bonifacio and Moura [13] have proposed a method for checking test suite completeness which is based on necessary

☆ In collaboration with Computing Institute - Unicamp
* Corresponding author.
E-mail addresses: bonifacio@uel.br, adilsonbonifacio@gmail.com (A. Bonifacio), arnaldo@ic.unicamp.br (A. Moura), adenilso@icmc.usp.br (A. Simao).

http://dx.doi.org/10.1016/j.infsof.2017.07.012
0950-5849/© 2017 Published by Elsevier B.V.

and sufficient conditions. If the method is applied to a specification M and a test suite T, the outcome is always conclusive, that is, it always yields either a 'yes' or a 'no' answer, thus always deciding whether or not the test suite is complete for the given specification FSM. The method relies on the notion of simulation relations. Completeness is guaranteed if and only if any T-equivalent candidate implementation FSM simulates M. The approach also requires implementations to be completely specified. An extension [14] generalized the method to partial implementations. We remark that the fault domains here are as wide as possible, that is, a complete test suite must detect any misbehavior between the given specification and any implementation no matter what input string is submitted to them, provided that the implementation under test has at most an arbitrary, but fixed, number of states.

It is not surprising that algorithms that give a conclusive verdict only when some sufficient conditions are present can be more efficient in practice. However, they may miss completeness when the required sufficient conditions cannot be ascertained. On the other hand, if a conclusive verdict is important, *e.g.* due to the nature of the application under test, we need to consider algorithms based on necessary and sufficient conditions, which will always yield a definitive verdict about test suite completeness.

In this paper, we present experimental results comparing the method proposed by Simao and Petrenko [12] and the approach in [13]. We randomly generate different test suites and different groups of FSMs. Each experiment evaluates different aspects of both methods, such as the number of states allowed for specifications, the number of transitions in the models, the number of input and output actions, and the size of test suites. We also investigate when both methods can be combined in mutually reinforcing ways. It turns out that these methods are somehow complementary, which allows for a combined strategy: one method is the fastest but does not always terminate with conclusive verdicts, while the other is not as scalable but it always returns conclusive answers.

The remaining of this paper is organized as follows. Section 2 contains basic notions and lays down some notation. Both methods are presented in Section 3 and experimental results are reported in Section 4. Section 5 offers concluding remarks.

2. Basics and notation

If X is a set, then $\mathcal{P}(X)$ indicates the power set of X. Let A be an alphabet. The length of a finite sequence of symbols α over A is indicated by $|\alpha|$. The set of all finite sequences over A is denoted by $A^\star$. The empty sequence will be indicated by ε. When we write $x_1x_2\ldots x_n \in A^\star$, it is implicitly assumed that $n \geq 0$ and that $x_i \in A$, $1 \leq i \leq n$, unless explicitly noted otherwise. When $n = 0$, $x_1x_2\ldots x_n$ denotes the empty sequence, ε. The classical notion of Finite State Machines [12,15] is as follows:

Definition 1. A Finite State Machine (FSM) is a tuple $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$, 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*
- δ: $D \to S$ is the *transition function*
- $\lambda : D \to \mathcal{O}$ is the *output function.*

Note that all FSMs treated here are deterministic. The following conventions will ease the notation:

- We fix $M = (S, s_0, \mathcal{I}, \mathcal{O}, D, \delta, \lambda)$ and $N = (Q, q_0, \mathcal{I}, \mathcal{O}', D', \mu, \tau)$, as FSMs with the same input action alphabet, $\mathcal{I}$.
- s, q, p, r will indicate states; x, y will indicate input actions; and a, b will indicate output actions; σ will indicate input action sequences, and ω will denote output action sequences. We may also use decorations, like s_1, x' or a'_3.

Let M be a FSM and let $\sigma = x_1x_2\cdots x_n \in \mathcal{I}^\star$, $\omega = a_1a_2\cdots a_n \in \mathcal{O}^\star$. 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 $r_0 \overset{\sigma/\omega}{\to} r_n$. Since all FSMs treated here are deterministic, when such states r_i exist, they are unique, given r_0. When the input sequence σ, or the output sequence ω, is not important we may write $r_0 \overset{\sigma/}{\to} r_n$, or $r_0 \overset{/\omega}{\to} r_n$, respectively. If both sequences are not important, we may write $r_0 \to r_n$. We can also drop the target state when it is not important, *e.g.* $r_0 \overset{\sigma/\omega}{\to}$ or $r_0 \to$. When we need to indicate the FSM we write $\underset{M}{\overset{\sigma/\omega}{\to}}$, and similarly for the other variants of the notation. The function $U : S \to \mathcal{P}(\mathcal{I}^\star)$ will be useful, where $U(s) = \{\sigma \,|\, s \overset{\sigma/}{\to}\}$.

It will be useful to extend the domains of the functions δ and λ so as to accommodate sets of strings. Define $\widehat{\delta} : S \times \mathcal{P}(\mathcal{I}^\star) \to \mathcal{P}(S)$ by letting $\widehat{\delta}(s, K) = \{r \,|\, s \overset{\sigma/}{\to} r$, for some $\sigma \in K\}$. We may write $\widehat{\delta}(s, \sigma) = r$ instead of $\widehat{\delta}(s, \{\sigma\}) = \{r\}$ and, when there is no ambiguity, we may simply write δ instead of $\widehat{\delta}$. Similarly, we extend the function λ to $\widehat{\lambda} : S \times \mathcal{P}(\mathcal{I}^\star) \to \mathcal{P}(\mathcal{O}^\star)$ by letting $\widehat{\lambda}(s, K) = \{w \,|\, s \overset{\sigma/w}{\to}$, for some $\sigma \in K\}$, and we may relax the notation likewise.

Now we can say when a FSM is complete.

Definition 2. A FSM M is *complete* if and only if $(s, x) \in D$, for all $s \in S$ and all $x \in \mathcal{I}$.

If M is not complete, it is a *partial* FSM.

The next notions of distinguishability and equivalence are crucial to understand suite completeness. Recall our notational conventions at page 5.

Definition 3. Let M and N be FSMs, $s \in S$, $q \in Q$, and $C \subseteq \mathcal{I}^\star$. Then s and q are *C-distinguishable*, written $s \not\approx_C q$, if and only if $\lambda(s, \sigma) \neq \tau(q, \sigma)$ for some $\sigma \in U(s) \cap U(q) \cap C$. Else, they are *C-equivalent*, written $s \approx_C q$. We say that M and N are *C-distinguishable*, written $M \not\approx_C N$, if and only if $s_0 \not\approx_C q_0$, otherwise they are *C-equivalent*, written $M \approx_C N$.

When C is omitted in the relation $\approx$, we mean $C = \mathcal{I}^\star$. Also, we may simply say that M is equivalent to N when $M \approx N$.[1]

We now say when a FSM is reduced.

Definition 4. We say that a FSM M is *reduced* if and only if every pair of distinct states in S are distinguishable, and for all $s \in S$ there is an input sequence $\sigma \in \mathcal{I}^\star$ with $\delta(s_0, \sigma) = s$.

An input sequence σ such that $\delta(s_0, \sigma) = s$ is also called a *transfer sequence* for s. We note that the definition of reduced FSMs in [12] is written in slightly different, but equivalent, terms when compared to Definition 4, since they assume at the outset that their models are *initially connected* FSMs, that is, they are FSMs where there is a transfer sequence for each state. We also note that some earlier notions of reducibility for FSMs did not require the reachability condition $\delta(s_0, \sigma) = s$. Since we do not assume FSMs to be initially connected, this condition prevents the consideration of FSMs that may have a block of states unreachable from the initial state. Clearly, those states could not be probed during a test run.

Since for any two states that are distinguishable there is a shortest input sequence that distinguishes them, we can effectively check if a FSM is reduced.

Given sequences $\alpha, \beta, \gamma \in \mathcal{I}^\star$, if $\beta = \alpha\gamma$, then α is said to be a *prefix* of β. Moreover, if γ is not empty, then α is a *proper* prefix

[1] Note that C-equivalence is not an equivalence relation in the usual sense.