Information and Software Technology 52 (2010) 1286–1297

Contents lists available at ScienceDirect

Information and Software Technology

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

FSM-based conformance testing methods: A survey annotated with experimental evaluation

Rita Dorofeeva [a], Khaled El-Fakih [b,*], Stephane Maag [c], Ana R. Cavalli [c], Nina Yevtushenko [a]

[a] Tomsk State University, 36 Lenin Str., 634050, Tomsk, Russia
[b] American University of Sharjah, College of Engineering, P.O. BOX 2666, Sharjah, United Arab Emirates
[c] TELECOM SudParis, CNRS UMR 5157, 9, rue Charles Fourier, 91011, Evry Cedex, France

ARTICLE INFO

Article history:
Received 8 December 2009
Received in revised form 12 April 2010
Accepted 9 July 2010
Available online 30 July 2010

Keywords:
Conformance testing
Protocol testing
Model based testing
Finite state machines

ABSTRACT

The development of test cases is an important issue for testing software, communication protocols and other reactive systems. A number of methods are known for the development of a test suite based on a formal specification given in the form of a finite state machine. In this paper, we overview and experiment with these methods to assess their complexity, applicability, completeness, fault detection capability, length and derivation time of their test suites. The experiments are conducted on randomly generated specifications and on two realistic protocols called the Simple Connection Protocol and the ITU-T V.76 Recommendation.

1. Introduction

The development of test cases based on a formal model is an important issue for software testing including conformance testing of communication protocols and other reactive systems. The purpose of these tests is to determine whether a protocol implementation conforms to (i.e. is correct with respect to) its specification. Usually a conforming implementation is required to have the same input/output behavior as defined by the specification. In various application domains, such as telecommunication systems, communication protocols and other reactive systems, the specification can be represented in the form of a finite state machine (FSM). In particular, FSMs are the underlying models for formal description techniques, such as SDL and UML State Diagrams. A number of methods are known for the development of a test suite based on a specification given in the form of a finite state machine (FSM). Well-known methods are called the W [1,2], Wp [3], UIO [4], UIOv [5], DS [6], HSI [7–9] and the H [10,11] test derivation methods. For related surveys and tools the reader may refer to [12–17]. Moreover, in the last years an increasing research has been developed on the application of these methods to object oriented software [18].

In FSM-based testing, one usually assumes that not only the specification, but also an implementation can be modeled as an FSM. In order to check if a given implementation under test (IUT), assumed to be a black-box, conforms to its specification, input/outputs pairs of test sequences (*test cases*) are derived from the given specification. The inputs of these tests are then applied to an implementation and outputs generated (observed) by the implementation are compared with expected outputs. If the outputs do not match, then the implementation has a fault. A test suite is called *complete* if it detects every faulty implementation w.r.t. the considered fault model. All of the above methods, except of the UIO method, each provides the following *complete fault coverage* guarantee: if the specification can be modeled by a reduced deterministic FSM with n states and if an implementation can be modeled by a complete deterministic FSM with at most m states, then a test suite can be derived by the method (for this given m) such that the implementation passes this test suite if and only if it conforms to the specification (i.e., the implementation and the specification have the same input/output behaviors). Guessing the bound of m is an intuitive process based on the knowledge of a specification and the class of implementations that have to be tested for conformance and their interior structure [12]. Usually, it is assumed that m equals or is greater than n [2,3,5,8,9,12,15].

All of the above methods assume that a reset function (written as '*r*') is available that allows the reliable reset of an implementation under test. This implies that the test suite can be composed of individual test cases, each starting with the reset operation. We note that there is another group of methods [6,19–21] that do not use this assumption. These methods usually yield much longer test suites and are not considered in the paper.

* Corresponding author. Tel.: +971 515 6 5152492; fax: +971 971 6 515 2979.
E-mail addresses: drf@kitidis.tsu.ru (R. Dorofeeva), kelfakih@aus.edu (K. El-Fakih), Stephane.Maag@it-sudparis.eu (S. Maag), Ana.Cavalli@it-sudparis.eu (A.R. Cavalli), ninayevtushenko@yahoo.com (N. Yevtushenko).

0950-5849/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2010.07.001

All the above methods use certain state distinguishing input sequences (called *state identifiers*) for test derivation, and thus, can be only applied when all states of the specification FSM are pair-wise distinguishable (i.e. the specification is reduced). Moreover, the length of a derived test suite essentially depends how state identifiers are selected. The DS method uses a distinguishing sequence of the specification FSM and can be thought of as a particular case of the W method. However, the DS method is not always applicable even for complete reduced specifications [22]. The Wp method is an improvement of the W method and the UIOv method can be thought of as a particular case of the Wp. The UIO method employs part of the UIOv method. However, UIO test suites are not always complete [5]. We also note that the W method (and correspondingly, Wp- and UIOv-methods) is not always applicable for partial reduced specifications [9,16]. The reason is not each even reduced partial FSM has a W set that distinguishes any pair of states of the specification. The HSI method, another improvement of W method, can be applied for each reduced deterministic specification FSM. The H method is an improvement of the HSI method and unlike all other methods, in the H method, for every state of the specification FSM, appropriate state identifiers are derived on-the-fly based on already derived part of a test suite in order to shorten the length of the resulting test suite. Similar to HSI method, H method is always applicable for any reduced specification.

During the past years a lot of research work has been done on developing complete FSM-based testing methods [1,3,5,8,9,15], studying the fault coverage of test suites [23,24], developing testing computer-aided tools [13,25], and using the FSM-based methods in the development of test cases from specifications modeled using other formalisms as extended FSMs (EFSMs) and labelled transition systems LTSs (related survey in [12]). Moreover, these methods have been used in order to derive tests for realistic protocols. For example, Refs. [26] and [15] include a comparison of the application of the UIO, DS and W methods to a subset of the NBS Class 4 transport (TP4) [27]. The application of the Wp method for a control subset of the XTP protocol is presented in [28] and the application of the HSI method [7,9,29] to the OSI session protocol has been presented in [25]. The application of the UIO method [30] to the ISDN protocol LAPD was conducted at AT&T and reported in [31]. These applications and others [26] have demonstrated that formal methods for conformance test generation are quite effective for rather complex communication protocols.

The ongoing research work on the development of complete testing methods focuses on two issues, minimizing the length of derived test sequences and enhancing the applicability of the testing methods. For example, it is known that the Wp method generates shorter tests than its ancestor the W method as demonstrated in [3]. Also, it is known that the UIO and UIOv methods are more applicable than the DS method but there exist complete reduced specifications where the UIO method cannot be applied and thus, UIO method is less applicable than the W, Wp, HSI, and H methods. The W and Wp methods are not always applicable for partial reduced specifications while the H and HSI methods are applicable to any complete or partial reduced specification [9,16]. Moreover, given a reduced FSM M with n states and k input symbols, the worst-case length of a test suite generated using all the above W, Wp, HSI, and H methods (except DS and UIO methods) is of order $O(k^{m-n+1}n^3)$, where m is the upper bound on the number of states of an implementation of M [2].

In [15], Sidhu and Leung offered a proper comparison of the W, DS, and UIO methods. However, to the best of our knowledge, there is no paper that includes a comprehensive (experimental) evaluation of all FSM-based methods using the same set of specification FSMs. To this end, in this paper, we first present a detailed overview of all above methods including recent methods such as the HSI [7–9] and the H [10,11] methods. We describe test derivation using simple formulae and we add corresponding application examples. Second, we implement and experiment, using randomly generated specifications, with the above methods in order to:

a. Determine average length and execution time of test suites for the cases when an implementation has equal or larger number of states than the specification, i.e., when $m = n$ and $m > n$.
b. Study the effects of increasing the number of states and transitions on the length of obtained test suites.
c. Determine how often the UIO and DS methods are not applicable.
d. Determine how often the UIO method (when applicable) generates incomplete test suites and the fault detection capability of the UIO-based test suites.
e. Compare length of obtained test suites with the estimated theoretic worst-case upper-bound length.

Moreover, we apply the above test derivation methods to two realistic protocols, namely the simple connection protocol (SCP) and the V76 protocol [32], and compare the length of test suites derived using randomly generated specifications and specifications of the realistic protocols. A summary of obtained results can be found in Section 6.

We note that a preliminary version of this work appeared in [33]. In this paper we extend that work in many ways. For example, in this paper, we include more experiments and more thorough analysis, for example, we add experiments for the case when an implementation has more states than the specification (i.e. when $m > n$). In addition, we model and include experiments with the V.76 protocol [32] and we provide a detailed description and application example of the recently developed H method [10,11].

This paper is organized as follows. Section 2 defines notations for describing finite state machines and Section 3 includes an overview of the W, Wp, HSI, H, UIOv, UIO, and DS test derivation methods. Sections 4 and 5 include the experimental results and Section 6 concludes the paper.

2. Finite state machines

This section contains the definition of basic concepts that are used in the rest of the paper for demonstrating the test derivation methods.

2.1. Definition

A deterministic *finite state machine* (FSM) is an initialized complete deterministic Mealy machine that can be formally defined as a 6-tuple $M = (S,X,Y,\delta_M,\lambda_M,s_1)$ [22], where S is a finite set of states, s_1 is the *initial state*, X is a finite set of input symbols, Y is a finite set of output symbols, δ_M is the next state (or transition) function: δ_M: $S \times X \to S$, λ_M is the output function: λ_M:$S \times X \to Y$. In usual way, functions δ_M and λ_M are extended to input sequences.

2.2. Definition

A FSM A is called *connected* if for each state $s \in S$ there exists an input sequence α_s that takes FSM A from the initial state to state s. The sequence α_s is called a *transfer* sequence for the state s.

2.3. Definition

The *concatenation* of two sets V_1 and V_2 of input sequences is defined as $V_1 \cdot V_2 = \{v_1 \cdot v_2 \mid v_1 \in V_1,\ v_2 \in V_2\}$, where $v_1 \cdot v_2$ denotes the