Applied Soft Computing xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

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

Heuristics for deriving distinguishing experiments of nondeterministic finite state machines

Khaled El-Fakih[a,*], Abdul Rahim Haddad[b], Nassima Aleb[c], Nina Yevtushenko[d]

[a] Computer Science & Engineering Department, American University of Sharjah, P.O. Box 26666, Sharjah, United Arab Emirates
[b] Computer Science & Engineering Department, American University of Sharjah, Sharjah, United Arab Emirates
[c] University of Science and Technology Houari Boumediene, Algeria
[d] Tomsk State University, Russia

ARTICLE INFO

Article history:
Received 9 January 2016
Received in revised form 3 July 2016
Accepted 11 July 2016
Available online xxx

Keywords:
Software engineering
Functional testing
Conformance testing
Distinguishing experiments
Nondeterministic finite state machines
Mutation testing
Heuristics
Evolutionary algorithms
Genetic algorithms

ABSTRACT

Derivation of input sequences for distinguishing states of a finite state machine (FSM) specification is well studied in the context of FSM-based functional testing. We present three heuristics for the derivation of distinguishing sequences for nondeterministic FSM specifications. The first is based on a cost function that guides the derivation process and the second is a genetic algorithm that evolves a population of individuals of possible solutions (or input sequences) using a fitness function and a crossover operator specifically tailored for the considered problem. The third heuristic is a mutation based algorithm that considers a randomly generated input sequence as a candidate solution, and if the candidate is not a distinguishing sequence, then the algorithm tries to find a solution by appropriately mutating the considered candidate. Experiments are conducted to assess the performance of the considered algorithms with respect to execution time, virtual memory consumption, and quality (length) of obtained sequences. Experiments are conducted using randomly generated machines with a various number of states, inputs, outputs, and degrees of non-determinism. Further, we assess the impact of varying the number of states, inputs, outputs, and degree of non-determinism.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Testing is a crucial yet expensive part of the software development process. One of the most promising approaches for reducing the costs of testing is to consider the derivation of tests from formal specifications (or models) as such test derivation can be automated. Furthermore, formal specifications provide a rigorous discipline for functional testing of communication protocols and other reactive systems. Accordingly, in the past years, many functional (conformance) test derivation methods have been developed for deriving tests when the system specification and implementation are represented as (Mealy) Finite State Machines (FSMs). For some related methods, experiments and surveys, the reader may refer to [1–8]. An FSM is a state transition system which has a finite number of inputs, outputs, states and a finite number of transitions each labeled by an input/output pair. FSMs are widely used in various application domains such as communication protocols and other reactive systems. Moreover, FSMs are the underlying models for formal description techniques, such as statecharts, SDL, and UML.

In FSM-based testing, we have a black-box FSM Implementation Under Test (IUT) about which we lack some information, and we want to deduce this information by conducting experiments on this IUT. An experiment consists of applying input sequences to the IUT, observing corresponding output responses, and drawing a conclusion about the IUT. Well-known types of experiments that are widely used in FSM-based testing are distinguishing experiments. An input sequence applied when performing a distinguishing experiment is called a *distinguishing sequence*. Distinguishing sequences have been constructed for different types/classes of FSMs. Given an FSM, assuming that the initial state is unknown, a distinguishing sequence determines the initial state of the FSM before the experiment and such a sequence is widely used in FSM-based conformance testing [1–8] for checking the correspondence between transitions of an IUT and those of the specification FSM. In addition, distinguishing sequences are used in FSM-based mutation [9–11] and fault diagnosis techniques [12–15]. For example, test derivation from a given specification machine can be done by constructing appropriate mutants from the given

* Corresponding author.
E-mail addresses: kelfakih@aus.edu (K. El-Fakih), abdulrahim.haddad@gmail.com (A.R. Haddad), na.aleb@gmail.com (N. Aleb), yevtushenko@sibmail.com (N. Yevtushenko).

http://dx.doi.org/10.1016/j.asoc.2016.07.019
1568-4946/© 2016 Elsevier B.V. All rights reserved.

Please cite this article in press as: K. El-Fakih, et al., Heuristics for deriving distinguishing experiments of nondeterministic finite state machines, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.07.019

specification machine considering selected types of faults, and then distinguishing sequences (test cases) are derived to separate the derived mutants from the given specification. Furthermore, in FSM and extended FSM based fault diagnosis, several nondeterministic FSMs, called *mutation machines* or *fault functions*, can be constructed from a given specification machine considering selected types of faults. For example, one could derive several mutation machines considering transfer and output faults at selected transitions of the specification machine then distinguishing sequences (*diagnostic tests*) can be derived for these mutation machines such that when applied to the given faulty IUT these tests help identifying the faults of the IUT.

Given a state of a nondeterministic FSM, there can be several output responses at the state to the same input sequence. Given two states of the FSM, a distinguishing sequence is called a *separating sequence* [16] if the sets of output responses (or sequences) produced at the considered states in response to the input sequence are disjoint.

Nowadays, analysis of nondeterministic systems is capturing a lot of attention. Nondeterminism occurs due to various reasons such as performance, flexibility, limited controllability, and/or abstraction. Accordingly, some research work has been done on the derivation of distinguishing sequences for nondeterministic FSMs [17–20]. In particular, for separating sequences, Alur et al. [17] have shown that, in general, the length of a separating sequence can be exponential. Spitsyna et al. [18] proposed a method that can be used for deriving a shortest separating sequence (if such a sequence exists) of two given FSMs (or for two states of a given FSM). Given two states of an FSM with n states, the length of a shortest separating sequence can reach $2^{(\frac{n}{2})^2-1}$ [18]. Alur et al. [17] studied the complexity of the problem of deriving a separating sequence and showed that the problem is PSPACE-complete.

In this paper, we aim at reducing the efforts (execution time and memory) of deriving separating sequences. In particular, we first implement and experiment with the algorithm, hereafter called *Exact Algorithm* (EA), given in [18] where breadth first search is used while exploring a corresponding search tree (a successor tree). Then, we propose three heuristics for solving the problem. The first, called *Heuristic Algorithm* (HA), is similar to EA; with an alteration that the exploration of the search tree is guided by a cost function that selects the most promising nodes of the tree while conducting the search for a solution. Similar to EA, HA guarantees the derivation of a separating sequence (when such a sequence exists) but in general, HA might not return a shortest separating sequence. However, according to the conducted experiments, HA not only outperforms EA in terms of execution time but also it always derives a shortest separating sequence as EA. Furthermore, we present and experiment with two other heuristics that can be used for deriving separating sequences, namely, a *Genetic Algorithm* (GA) that works on evolving a set of possible solutions (input sequences) using a fitness function and a crossover operator appropriately developed for the considered problem. GAs were used for efficiently solving a variety of complex engineering and testing problems [21–25]. We also present a *Mutation Based* (MA) algorithm that initially randomly derives a candidate solution to the problem (input sequence) of particular length and then if the derived candidate is not a separating sequence, MA continues searching for a solution by appropriately mutating the derived candidate. Unlike EA and HA, both GA and MA may not provide a solution even when a solution exists; in addition, similar to HA, when GA/MA finds a solution it might not provide the shortest one. However, our experiments show that in most cases GA and MA find a separating sequence (if it exists). Furthermore, GA and MA can process bigger machines compared to EA and HA and also the execution time of GA and MA does not tend to significantly increase when considering bigger machines. We assess the performance of the considered algorithms with respect to execution time and quality (length) of obtained solutions using randomly generated machines with various attributes (states, inputs, outputs, and minimum degrees of non-determinism). The minimum (maximum) degree of non-determinism represents the minimum (maximum) number of outgoing transitions at each state under each input of the generated machine. Further, we assess the impact of varying the number of states, inputs, outputs, and minimum degree of non-determinism on execution time and solution quality. We also consider stress testing EA and HA and assess their execution time using special machines which have two states that are separable only by a sequence of exponential length. A summary of experimental results is reported in Section 5.

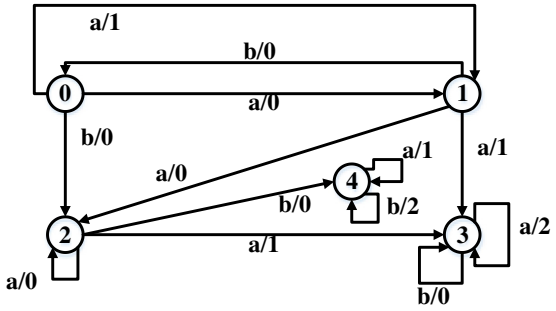

Fig. 1. Nondeterministic FSM M.

This paper is organized as follows. Section 2 contains preliminaries and the exact algorithm. Section 3 includes the proposed algorithms while experimental evaluation is presented and summarized in Sections 4 and 5, respectively. Section 6 concludes the paper.

2. Preliminaries

2.1. Finite state machines and related definitions

A *finite state machine* (*FSM*) S, simply called a *machine* throughout the paper, is a 4-tuple $\langle S, I, O, h\rangle$, where S is a finite nonempty set of states; I and O are input and output alphabets; and $h \subseteq S \times I \times O \times S$ is a behavior relation. The behavior relation defines all possible transitions of the machine. Given a current state s_j and input symbol i, a 4-tuple $(s_j, i, o, s_k) \in h$ represents a possible transition from state s_j under the input i to the next state s_k with the output o. A machine is called *deterministic* if for each pair $(s, i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h$; otherwise, the machine is called *nondeterministic*. If for each pair $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h$ then FSM S is said to be *complete*; otherwise, the machine is called *partial*. FSM S is *initialized* if it has the designated initial state s_1, also written S/s_1. Thus, an initialized FSM is a 5-tuple $\langle S, I, O, h, s1\rangle$. Given FSMs $S = \langle S, I, O, h, s1\rangle$ and $T = \langle T, I, O, g, t1\rangle$, FSM T is a submachine of S if $T \subseteq S$, $t1 = s1$ and $g \subseteq h$. A complete nondeterministic FSM is *observable* if at each state, the machine has at most one transition under a given input/output pair, otherwise, it is *non-observable*. In the following, we consider observable and complete FSMs if the contrary is not explicitly stated.

As an example, consider the machine M in Fig. 1. The machine is defined over the sets of inputs $I = \{a, b\}$, outputs $O = \{0, 1, 2\}$, and states $S = \{1, 2, 3, 4, 5\}$, respectively. The machine is nondeterministic as for example, from state 2 under input a there are two outgoing transitions leading to states 2 and 3, respectively.

Given FSM $S = \langle S, I, O, h\rangle$, state s and an input i, the *successor* of state s under the input i or simply the *i-successor* of state s contains

Download English Version:

https://daneshyari.com/en/article/4963627

Download Persian Version:

https://daneshyari.com/article/4963627

Daneshyari.com