Information and Software Technology 74 (2016) 69–85

Contents lists available at ScienceDirect

Information and Software Technology

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

Effective algorithms for constructing minimum cost adaptive distinguishing sequences

Uraz Cengiz Türker*, Tonguç Ünlüyurt, Hüsnü Yenigün

Sabanci University, Orhanli, Tuzla, Istanbul 34956, Turkey

ARTICLE INFO

Article history:
Received 4 January 2015
Revised 1 February 2016
Accepted 2 February 2016
Available online 26 February 2016

Keywords:
Finite State Machines
Adaptive distinguishing sequences
Checking sequences

ABSTRACT

Context: Given a Finite State Machine (FSM), a checking sequence is a test sequence that determines whether the system under test is correct as long as certain standard assumptions hold. Many checking sequence generation methods use an adaptive distinguishing sequence (ADS), which is an experiment that distinguishes the states of the specification machine. Furthermore, it has been shown that the use of shorter ADSs yields shorter checking sequences. It is also known, on the other hand, that constructing a minimum cost ADS is an NP-hard problem and it is NP-hard to approximate. This motivates studying and investigating effective ADS construction methods.

Objective: The main objective of this paper is to suggest new methods that can compute compact ADSs to be used in the construction of checking sequences.

Method: We briefly present the existing ADS construction algorithms. We then propose generalizations of these approaches with a set of heuristics. We also conduct experiments to compare the size of the resultant ADSs and the length of the checking sequences constructed using these ADSs.

Results: The results indicate that when the ADSs are constructed with the proposed methods, the length of the checking sequences may reduce up to 54% (40% on the average).

Conclusions: In this paper, we present the state of the art ADS construction methods for FSMs and we propose generalizations of these methods. We show that our methods are effective in terms of computation time and ADS quality.

1. Introduction

Testing is an important part of the software development process but is typically manual and, as a result, expensive and error prone. Therefore, there has been a significant interest in automating testing from formal specifications. A widely used formal model for the specification is the Finite State Machine (FSM) model. The FSM model and its extensions such as Specification and Description Language (SDL) [1] or State-Charts [2] are also used to model the semantics of the underlying software.

Deriving test sequences from FSM models, therefore, has been an attractive topic for various application domains such as sequential circuits [3], lexical analysis [4], software design [5], communication protocols [6–11], object-oriented systems [12], and web services [13,14]. Such techniques have also been shown to be effective in important industrial projects [15]. The purpose of generating these test sequences is to decide whether an implementation conforms to its specification. An implementation is said to conform to its specification when the implementation has the same behavior as defined by the FSM specification.

In order to determine whether an implementation N has the same behavior as the specification M, a test sequence (an input/output sequence) is derived from M and the input portion of the sequence is applied to N. The final decision is made by comparing the output sequence produced by N (i.e. *the actual output*) and the output portion of the test sequence (i.e. *the expected output*). If there is a difference between the actual and the expected output, then N is a faulty implementation of M. Although, in general, having no difference between the actual and the expected output does not mean that N is a correct implementation of N, it is possible to construct a test sequence with such a guarantee under some conditions on M and N. A test sequence with such a full fault coverage is called *a checking sequence* [5,16].

The literature contains many techniques that automatically generate checking sequences [5,16–21]. In principle, checking

* Corresponding author. Tel.: +90 5073631731.
E-mail addresses: urazc@sabanciuniv.edu, uraz.turker@brunel.ac.uk (U.C. Türker), tonguc@sabanciuniv.edu (T. Ünlüyurt), yenigun@sabanciuniv.edu (H. Yenigün).

http://dx.doi.org/10.1016/j.infsof.2016.02.001
0950-5849/

sequences constructed by these approaches consist of three types of components: *initialization, state identification*, and *transition verification*. As the transition verification components are also based on identifying the starting and ending states of the transitions, a checking sequence incorporates many applications of input sequences to identify the states of the underlying FSM.

For state identification several alternative approaches exist, such as *Distinguishing Sequences (DS), Unique Input Output (UIO) Sequences* or *Characterizing Sets (W-Set)*. Among these alternatives, a checking sequence of polynomial length can be constructed in polynomial time when a DS exists [19,22]. Checking sequences constructed without using a DS, on the other hand, are in general of exponential length [19]. Therefore, many techniques for constructing checking sequences either use a given DS [17,18,23,24], or use both DS and other alternatives together [25–27] for state identification.

There are two types of distinguishing sequences. A *Preset Distinguishing Sequence (PDS)* is a single input sequence for which different states of FSM produce different output sequences. On the other hand, an *Adaptive Distinguishing Sequence (ADS)* (also known as a *Distinguishing Set* [28]) can be thought as a rooted decision tree with n leaves, where n is the number of states of M. The internal nodes of the tree are labeled by input symbols and the leaves are labeled by distinct states. The edges emanating from a common node have different output symbols labeling the edges. The concatenation of input and output labels on a path from the root node a leaf node labeled by a state s, correspond the output sequence that would be obtained when this input sequence is applied to the state s. We present a formal definition of ADS in Section 2.

The use of ADS is straightforward: to identify the current state of an FSM, one applies the input symbol at the root and follows the outgoing edge labeled by the output symbol that is produced by the FSM. The procedure is repeated for the root of the subtree reached in this way, as long as the current node is an internal node of the ADS. When a leaf node is reached, the state label of the node gives the initial state that the experiment started.

In this paper, we consider *deterministic* and *completely specified* FSMs[1]. For constructing a checking sequence for such FSMs, using an ADS rather than a PDS is advantageous. Lee and Yannakakis show that checking the existence of and computing a PDS is a PSPACE-complete problem. On the other hand, for a given FSM M with n states and m input symbols, the existence of an ADS can be decided in $O(mn\log n)$ time [29].

1.1. Literature review

This section reviews previous work on ADSs. There are many computational complexity results regarding ADSs for deterministic and complete FSMs. Although earlier bounds for the height of ADSs are exponential in the number of states [30], Sokolovskii proved that if an FSM M with n states has an ADS, then it has an ADS with height $\leq \pi^2 n^2/12$ [31]. Moreover, Kogan claimed that, for a given n state FSM, the length of an ADS is bounded above by $n(n-1)/2$ [32]. Later Rystsov proved this claim [33]. Lee and Yannakakis proposed an algorithm (LY algorithm) that constructs an ADS with upper bound of $n(n-1)/2$ in the worst case in $O(mn^2)$ time [29]. It was proven that minimizing the height of an ADS (in fact minimizing ADS size with respect to some other metrics as well) is an NP-hard problem [34]. Türker and Yenigün proposed two heuristics as a modification of the LY algorithm for minimizing ADSs [34]. Recently Türker et al. also presented an enhanced version of successor tree algorithm called the lookahead based algorithm (LA) for ADS minimization [35].

Unfortunately, not all FSMs possess an ADS. For such cases, Hierons and Türker introduced the notion of incomplete ADSs [36]. They showed that the optimization problems and the corresponding approximation problems related to incomplete ADSs are PSPACE-complete. A greedy algorithm to construct incomplete ADSs is also given in this work.

Besides these results for deterministic and complete FSMs, there are also works on ADSs for non-deterministic and incomplete FSMs. Kushik et al. present an algorithm for constructing ADSs for non-deterministic observable FSMs [37]. Since the class of deterministic FSMs is a subclass of nondeterministic observable FSMs, the algorithm can also be used to construct ADSs for a given FSM M.

It was recently shown that for partial FSMs, checking the existence of an ADS can be done in polynomial time and checking the existence of a PDS is PSPACE-complete [38]. The height of a minimum ADS for a partial FSM is known to be at most $(n-1)^2$, although it is not known if this bound is tight [39]. Finally in [40] the authors propose a brute-force massively parallel algorithm for deriving ADSs/PDSs from partial observable nondeterministic FSMs.

1.2. Motivation and problem statement

As the length of the checking sequence determines the duration and hence the cost of testing, there exists a line of work to reduce the length of checking sequences. In these works, the goal is to generate a shorter checking sequence, by putting the pieces that need to exist in such a checking sequence together in a better way [17,18,21,23,41–43]. However in [34] Türker and Yenigün show the potential enhancements of constructing minimum cost ADSs on the length of checking sequences and examined the computational complexity of constructing minimum cost ADSs.

In their work, they define the "cost" of an ADS as (i) the height of the ADS (MinHeightADS problem), (ii) the sum of the depths of all leaves in the ADS (*external path length*) (MinADS problem), and (iii) the weighted sum of the depths of the leaves in the ADS (MinWeightADS problem). They showed that constructing a minimum ADS with respect to these cost metrics are NP-complete and NP-hard to approximate. They proposed two different modifications for the LY algorithm called GLY1 and GLY2 for constructing compact ADSs with respect to minimum height and minimum external path length.

As shown in Section 1.1, except for the exponential time algorithms [30,35,44], there have been no polynomial time algorithm proposed for constructing minimum cost ADSs. Besides there have been no work reported for constructing ADSs with minimum weight and there exists no work that shows the effect of using such ADSs for constructing checking sequences. This paper is mainly motivated by these observations.

In this paper, we first provide a brief summary for the existing ADS construction algorithms including STA, LY, GLY1, GLY2 and LA algorithms and then we propose generalizations of these approaches: (1) *Low-cost ST construction approach* (LCST) (2) *Splitting Forest Algorithm* (SFA), and (3) *Splitting Graph Algorithm* (SGA) for constructing reduced size ADSs. Furthermore, we present a set of new heuristics to construct ADSs with minimum height, minimum external path length and minimum weight.

LCST is a generalization of GLY1 and GLY2 algorithms. SFA makes use of a splitting forest (SF) to construct an ADS, and SGA makes use of a splitting graph (SG) to construct an ADS. Construction of STs, SFs and SGs are guided by different heuristics based on the objective, such as minimizing the height, the external path length or the weight of the ADS. LCST and SFA are polynomial time methods but SGA may require exponential time (with the number of states of the underlying FSM) to construct an ADS.

[1] Please see Section 2.1 for the definitions of these terms.