Science of Computer Programming 94 (2014) 93–108

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Test generation for sequential nets of Abstract State Machines with information passing

Paolo Arcaini, Angelo Gargantini *

Dipartimento di Ingegneria, Università degli Studi di Bergamo, viale Marconi 5, 24044 Dalmine (BG), Italy

HIGHLIGHTS

- We propose a test generation approach for distributed systems with information passing.
- The generation is performed only considering the single subsystems.
- The tests for the single subsystems are combined to obtain system valid traces.
- The policy used to combine the tests influences the completeness of the approach.
- The approach, compared to techniques considering the whole system, is very efficient.

ARTICLE INFO

Article history:
Received 20 February 2013
Received in revised form 11 October 2013
Accepted 5 February 2014
Available online 17 February 2014

Keywords:
Test case generation
State explosion problem
Information passing
Abstraction
State machines

ABSTRACT

Model-based test generation consists in deriving system traces from specifications of systems under test. There exist several techniques for test generation, which, however, may suffer from scalability problems. In this paper, we assume that the system under test can be divided in several subsystems such that only one subsystem is active at the time. Moreover, each subsystem decides when and to which other subsystem to pass the control, by also initializing the initial state of the next subsystem in a desired way. We model these systems and we show how it is possible to generate tests in a very efficient way that exploits the division of the entire system in subsystems. Test generation for the whole system is performed by visiting each subsystem and generating tests for it. The tests are combined in order to obtain valid system traces. We show how several visiting policies influence the completeness of the test generation process.

1. Introduction

Model-based automatic test generation consists in automatically generating tests from (abstract) models of systems under test. In this way, models and specifications are reused for software testing without the need for testers to write test suites by hand. The advantage of using models instead of code, is mainly the possibility to use specifications as test oracles and that specifications provide an abstract view of the system without all the implementation details. Although model-based test generation techniques have been successfully employed [1–3] even for complex systems, the scalability of these approaches is still a challenge.

We have worked on test generation from Abstract State Machines (ASMs) and used a tool for test generation for several years. However, since the test generation algorithm is based on model checking [4], one of the main obstacles has been the scalability of the approach and soon we encountered the well known *state space explosion problem*. Indeed, the problem

* Corresponding author.
E-mail address: angelo.gargantini@unibg.it (A. Gargantini).

http://dx.doi.org/10.1016/j.scico.2014.02.007
0167-6423/

of the model checking method is that the computational complexity increases exponentially together with the size of the model. Several techniques exist to overcome this limitation, like symbolic representation of states, compact storing of states, and efficient state space exploration. However, these techniques may still fail or weaken the coverage of the state space.

On the other hand, the system under test may have some peculiarities that can be exploited to limit the state explosion. We focus on systems that are composed of several subsystems that pass the control to each other such that only one subsystem is active at any time. This topology can be exploited for generating the test sequences over the single subsystems and combining them later, instead of generating the tests over the entire system. So, since the state space exponentially grows with the size of the system, decomposing the system exponentially reduces the complexity of the problem.

In this paper, we extend the approach in [5] by allowing information passing among the machines: the active machine decides the next machine and also its initial state by setting some location values as in classical value-passing of programming languages. This situation often occurs when modeling complex systems. For instance, the reader can think of a robotic system in which multiple small robots work in the same environment and pass to each other a job to be completed. The same scenario is common also in programming: a program is divided in multiple subprograms, but at every time only one subprogram is active and every subprogram calls another subprogram by passing some information.

Such systems can be modeled in an abstract way as *sequential nets* of ASMs, defined in Section 3, that are sets of ASMs having some features including that only one ASM is active at every time.

The test generation for the entire system modeled by a whole unique specification may be infeasible, but the topology of the system can be exploited by the test generation algorithm. In this paper, we present a technique in Section 4, that builds tests for single submodels and combines them in order to obtain valid system traces. Differently from [5], the test generation and test combination are performed at the same time, in order to visit the ASM only starting from valid initial states. We present several policies in which the activities of test generation and combination can be performed together. The *basic* strategy implements a classical depth first search, while the *retrying* method performs some extra visits in order to improve the testing coverage. Moreover, we present a *backward* search which is able to build tests by backward visiting the net and possibly improving the coverage.

The paper is organized as follows. Section 2 presents the ASM formalism and the use of model checkers for test generation. In Section 3 we formalize the concept of sequential nets of ASMs. Section 4 reports the three strategies we propose for generating test suites for sequential nets. Section 5 presents the relations existing between the three strategies. Section 6 describes the experiments conducted on three versions of the running case study that we use throughout the paper. Section 7 discusses the limitations of the presented approach. Section 8 relates our work with similar contributions, and Section 9 concludes the paper.

2. Background

2.1. Abstract State Machines

Abstract State Machines (ASMs) [6] are an extension of FSMs, where unstructured control states are replaced by states with arbitrary complex data. The *states* of an ASM are multi-sorted first-order structures, i.e., domains of objects with functions and predicates defined on them. *Static* functions never change during any run of the machine. *Dynamic* functions are distinguished between *monitored* (only read by the machine and modified by the environment), and *controlled* (read and written by the machine).

ASM states are modified by *transition relations* specified by "rules" describing the modification of the functions interpretation from one state to the next one. There is a limited but powerful set of *rule constructors* including guarded actions (`if-then`) and simultaneous parallel actions (`par`). The constructor `choose` expresses nondeterminism in a compact way. A rule can be declared with a name (*macro* rule) and called in another rule simply by its name.

A *computation* of an ASM is a finite or infinite sequence $s_0, s_1, \ldots, s_n, \ldots$ of states of the machine, where s_0 is an initial state and each s_{i+1} is obtained from s_i by executing the machine (unique) *main rule*. An ASM can have more than one *initial state*. Because of the nondeterminism of the choose rule and of the environment moves, an ASM can have several different runs starting in the same initial state.

An ASM state s_i is represented by a set of couples (*location*, *value*). ASM *locations*, namely pairs (*function-name*, *list-of-parameter-values*), represent the abstract ASM concept of basic object containers (memory units). Location *updates* represent the basic units of state change and they are given as assignments, each of the form $loc := v$, where *loc* is a location and v its new value.

2.2. Test generation for ASMs

In model based testing [2,1], the specification describing the expected behavior of the system is used as a test oracle to assess the correctness of the implementation. Tests are derived from specifications and used generally in conformance testing. In the following we give some basic definitions about test generation from ASMs.