Chapter 11
Constraint-Based Techniques for Software Testing

Nikolai Kosmatov
CEA LIST, Software Safety Laboratory, France

ABSTRACT

In this chapter, the authors discuss some innovative applications of artificial intelligence techniques to software engineering, in particular, to automatic test generation. Automatic testing tools translate the program under test, or its model, and the test criterion, or the test objective, into constraints. Constraint solving allows then to find a solution of the constraint solving problem and to obtain test data. The authors focus on two particular applications: model-based testing as an example of black-box testing, and all-paths test generation for C programs as a white-box testing strategy. Each application is illustrated by a running example showing how constraint-based methods allow to automatically generate test data for each strategy. They also give an overview of the main difficulties of constraint-based software testing and outline some directions for future research.

INTRODUCTION

Artificial intelligence (AI) techniques are successfully applied in various phases of software development life cycle. One of the most significant and innovative AI applications is using constraint-based techniques for automation of software testing.

Testing is nowadays the primary way to improve the reliability of software. Software testing accounts for about 50% of the total cost of software development (Ramler & Wolfmaier, 2006). Automated testing is aimed at reducing this cost. The increasing demand has motivated much research on automated software testing. Constraint solving techniques are commonly used in software testing since 1990's. They were applied in the development of several automatic test generation tools.

The underlying idea of constraint-based test generators is to translate the program under test, or its model, and the test criterion, or the test objective, into constraints. Constraint solving allows then to

DOI: 10.4018/978-1-60566-758-4.ch011

find a solution of the constraint solving problem and to obtain test data. The constraint representation of the program, interaction with a constraint solver and the algorithm may be different in each particular tool and depend on its objectives and test coverage criteria.

While learning about constraint-based techniques for the first time, we are often surprised to see that one constraint solver can so efficiently solve so different problems. For example, such as the famous SEND + MORE = MONEY puzzle (Apt, 2003), SUDOKU puzzles, systems of linear equations and many others, where a human would use quite different and sometimes very tricky methods. The intelligence of modern constraint solvers is not only in their ability to solve problems, but also in their ability to solve quite different kinds of problems. Of course, some solvers may be more adapted for specific kinds of problems.

In this chapter, we will discuss some innovative applications of constraint-based techniques to software engineering, in particular, to automatic test generation. We will focus on two particular applications: model-based testing as an example of black-box testing, and all-paths test generation for C programs as a white-box testing strategy. Each application will be illustrated by a running example showing how constraint-based methods allow to automatically generate test data for each strategy. We will also mention the main difficulties of constraint-based software testing and outline some directions for future research.

Organization of the Chapter

The chapter is organized as follows. We start by a short background section on software testing and describe the most popular test coverage criteria. The section on model-based testing contains an overview of the approach, an example of formal model and application of AI techniques to this example. Next, the section on all-paths test generation presents the generation method, its advantages and possible applications. We finish by a brief description of future research directions and a conclusion.

BACKGROUND

The classical book *The Art of Software Testing* by G. J. Myers defines software testing as "the process of executing a program with the intent of finding errors" (Myers, 1979, p.5). In modern software engineering, various testing strategies may be applied depending on the software development process, software requirements and test objectives. In *black-box testing* strategies, the software under test is considered as a black box, that is, test data are derived without any knowledge of the code or internal structure of the program. On the other hand, in *white-box testing*, the implementation code is examined for designing tests. Different testing strategies may be used together for improved software development. For example, one may first use black-box testing techniques for *functional testing* aimed at finding errors in the functionality of the software. Second, white-box testing may be applied to measure the test coverage of the implementation code by the executed tests, and to improve it by adding more tests for non-covered parts.

Significant progress in software testing was done by applications of artificial intelligence techniques. Manual testing being very laborious and expensive, automation of software testing was the focus of much research since 1970's. Symbolic execution was first used in software testing in 1976 by L. A. Clarke (1976) and J. C. King (1976). Automatic constraint-based testing was proposed by R. A. DeMilli and A. J. Offutt (1991). Since then, constraint-based techniques were applied for development of many automatic test generation tools. Like in manual testing, various *test coverage (test selection) criteria* may be used to control automatic test generation and to evaluate the coverage of a given set of test cases. The possibility to express such criteria in con-