ABSTRACT

BEAT is a Web-based system that generates fault-based test cases from Boolean expressions. It is based on the integration of our several fault-based test case selection strategies. The generated test cases are considered to be fault-based, because they are aiming at the detection of particular faults. For example, when the Boolean expression is in irredundant disjunctive normal form, test cases generated by BEAT can guarantee the detection of seven common types of operator and operand faults, which may occur during the implementation process. Apart from being an automated test case generation tool developed for software testing practitioners, it also can be used as a self-learning and training tool for students as well as software testing practitioners.

Keywords: Boolean logic — Boolean expression; CASE tools — test case generator; information and communication technologies — software testing; technology-enhance learning; Web-based applications — test case generator; Web-based learning — Boolean logic; Web-based learning — software testing techniques

INTRODUCTION

Boolean expression is a fundamental and important concept in the Information Technology area. In particular, it plays several important roles in the engineering of software. In the context of specifications, it can be found in the specifications of life-critical and safety-critical software. For example, Leveson et al. (1994, 2000) use an AND-OR table to specify the requirements of an aircraft Traffic Collison Avoidance System, TCAS II, which is required to be installed on every aircraft flying in U.S. airspace. As pointed out by Weyuker et al. (1994), an AND-OR table is just one of many possible representations of a Boolean expression. In addition, Boolean expression can be used to model various conditions in software systems using specification techniques, such as the
Software Cost Reduction (SCR) table (Offutt et al., 2003). On the other hand, program predicates such as assertions, pre-conditions, conditional statements, and post-conditions can be modeled as Boolean expressions. For example, Chilenski and Miller (1994) studied an industrial aircraft control system and found that it is not uncommon for a program predicate to have more than 15 simple conditions, each of which can be modeled by Boolean variables.

Various fault-based test case selection strategies have been proposed (Chen & Lau, 2001a; Chilenski & Miller, 1994; Kobayashi et al., 2002; Tai, 1996; Weyuker et al., 1994). These strategies are considered to be fault-based, because test cases are generated aiming at the detection of particular types of faults. For example, Tai (1996) proposes to use attribute grammar to generate test cases that can detect Boolean operator faults and relational operator faults related to the predicates in source code.

Weyuker et al. (1994) propose to generate test cases from specifications rather than from the implementation (or the program source). They propose a family of strategies to automatically generate test cases from Boolean specifications, which are specifications of software that can be written in Boolean expressions. The test cases are aiming at the detection of a particular type of fault. One of the advantages of this approach is that the generated test cases do not depend on the source code of the program. So, as long as the specifications do not change, the generated test cases can be reused.

Chen and Lau (2001a) developed three strategies to generate test cases from Boolean specifications. The test cases generated using the three strategies can guarantee to detect seven types of faults when the Boolean expression is in irredundant disjunctive normal form (IDNF). These three strategies are the MUTP strategy, the CUTPNFP strategy, and the MNFP strategy.

Since the test set that satisfies these three strategies can guarantee to detect seven types of faults when the Boolean expression is in IDNF, experimental study has been performed to evaluate whether the strategies are good in detecting faults when the Boolean expression is in general form (Chen & Lau, 2001b). Twenty Boolean expressions in general form are selected from Weyuker et al. (1994). Seven types of faults, described in a later section, are seeded into the expression, creating one faulty expression for each fault injected. The total number of faulty expressions that are not equivalent to the original expression for each fault type ranges from one to 1,225. It is found that test sets that can satisfy these three strategies can detect 97.8% to 100% of the faulty expressions with an average of 99.5% (Chen & Lau, 2001b). The size of the test sets ranges from 2.0% to 65.6% with an average of 25.7% of the entire input domain.

This article describes the BEAT system, which is a Web-based Boolean Expression fAult-based Test case generation tool. It allows users to select which of these three strategies to be applied in generating test cases from Boolean expression. Since a different application order of these three strategies may result in different test sets, BEAT also allows users to choose the order of application of the selected strategies when users decide to apply more than one strategy. Besides being a test case generation tool, BEAT also can serve as a self-learning tool. In a later section, we demonstrate how BEAT can be used to generate test cases and how the system can be used to help students and software testing practitioners to acquire the knowledge of generating test cases from Boolean expressions.

In this article, we first present the background knowledge. This includes the notation, the terminology, and the seven types of faults considered in this article. We then describe the three fault-based test case selection strategies that were developed by Chen and Lau (2001a) and implemented in BEAT. We also propose guidelines on using these three strategies and on integrating these strategies together for the generation of more comprehensive test sets. Third, we describe the BEAT system. Fourth, we illustrate with an example how BEAT can be used as a test case generator and as a learning tool. Finally, we conclude the article and discuss future work.
Related Content

Robotic Teaching Assistance for the “Tower of Hanoi” Problem
Nguyen Duc Thien, Annalisa Terracina, Luca Iocchi and Massimo Mecella (2016). International Journal of Distance Education Technologies (pp. 64-76).
www.igi-global.com/article/robotic-teaching-assistance-for-the-tower-of-hanoi-problem/143252?camid=4v1a

Data Modeling: A Vehicle for Teaching Creative Problem Solving and Critical Appraisal Skills
www.igi-global.com/chapter/data-modeling-vehicle-teaching-creative/6529?camid=4v1a

Education Mirrors Industry: On the Not-So Surprising Rise of Internet Distance Education
Donald A. Hantula and Darleen M. Pawlowicz (2004). The Distance Education Evolution: Issues and Case Studies (pp. 142-162).
www.igi-global.com/chapter/education-mirrors-industry/30304?camid=4v1a

The Challenge of Teaching Effectively from a Distance
Valerie E. Polichar and Christine Bagwell (2002). The Design and Management of Effective Distance Learning Programs (pp. 93-107).
www.igi-global.com/chapter/challenge-teaching-effectively-distance/30288?camid=4v1a