Theoretical Feasibility of Conditional Invariant Detection

Mohammad Hani Fouladgar, Hamid Parvin*, and Behrouz Minaei,
Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran
hamidparvin@mamasaniiau.ac.ir, {fouladgar,b_minaei}@iust.ac.ir

Abstract. All software engineering process, which includes designing, implementing and modifying of software, are done to develop a software as fast as possible and also to reach a high quality, efficient and maintainable software. Invariants, as rather always true properties of program context, can help developers to do some aspect of software engineering more easily; therefore any improvement in extracting of more relevant invariant can help software engineering process. Conditional invariant is a novel kind of invariant which is turned in when some conditions are provided in program execution. Conditional invariant can exhibit program behavior much better. In order to extract this kind of invariants, it might be used some technique of data mining such as association rule mining or using decision tree to obtain rules. This paper spans feasibility of conditional invariant and advantageous of this kind of invariant compared to ordinary invariant.

Keywords: Daikon, Invariant, Association Rules, Variable Relations, Decision tree, Program point, Data mining, Software engineering, Predicate, Verification.

1 Introduction

In last decade, program invariants have had significant effect on software engineering and especially in software testing and verification. Invariants are variables properties in and relationships between these variables in a specific line of code which is called program point. For example assume a subprogram that its task is to sort array of integers. In the post-condition of the mentioned subprogram, invariant (array a sorted >=) exists. This invariant means all element of array a is sorted by descending order. Extraction of invariants is a significant key in program verification. In all executions, these properties and relationships among the program variables or constants are always true; thus invariants help programmer or tester to be able to determine the behavior of program in different program points. Software behavioral model [1] uses invariant also is used in generating hence this can be mentioned as another usage of invariant in software engineering. Software behavioral model is used to perform design, validation, verification, and

* Corresponding author.

© Springer-Verlag Berlin Heidelberg 2011
maintenance. One of the most prominent contributions of invariants is in code modifying that properties help programmer to verify the code. Software testing takes a considerable time in software development life cycle. Although software testing is done automatically in present day, but traditionally the onus of software testing was human’s obligation [2]. Invariants might be used in automatically software testing. Invariants are detected and extracted by different methods which are divided into two major approaches, static and dynamic [4].

Static approach analyzes syntactic structures and runtime behavior of program without actual running of code [5]. Static analysis is a thoroughly automatic process. Compilers traditionally analyze Data-flows as a static analysis to collect necessary information for code optimization. Data-flows analysis extracts some needful invariants in each program point and uses these invariants to assess the program behavior. This kind of behavior is used in compilers for optimization. Abstract interpretation is a theoretical framework for static analysis [6]. The most precise imaginable abstract interpretation is called the static semantics or accumulating semantics.

On the other hand, Dynamic approaches elicit program properties and relationships by the help of actual executing of the program code [7]. In these approaches, program is executed with different inputs and test suits and variable properties and relations are detected according to variables value during execution time. Dynamic invariant detection first appeared in Daikon [4]. By using different test suits in different executions, in each program point, variable properties and relationships are extracted. These Program points are usually the points of entry and exist of program functions. Extracted properties and relations are invariants. These invariants are not certainly true but indeed they are true in all executions in test suit. One of important advantages of dynamic invariant detection is the inferred invariants not only show the properties and relations of variables via execution but also utter the inputs properties and relations. This is because invariants are extracted from some real inputs in actual executions. This attitude of dynamically detected invariants causes double usage of it.

In this paper we introduce dynamic inference of conditional invariant. Conditional invariant is a new sort of invariant which are revealed in specific situation and not in all runs of program. These invariants are extracted through a fully automatic process. This kind of invariant is more relevant and helps programmer or tester to have better view about program behavior. In order to extract invariants, we encounter with two significant issues [2]: we would be able to determine the beneficial invariants; and then to exert inference on program context. In this paper we solve these two issues and declare two models to elicit the conditional invariants. In the following we discuss about some related work (section 2) and some frequently used definitions (section 3). Then we introduce what exactly conditional invariant is (section 4). Section 5 proposes two different models to extract conditional invariant then it continues with predominance of conditional in section 6. Finally, we talk about conclusion and future work.