International Journal of Rough Sets and Data Analysis
Volume 6 • Issue 1 • January-March 2019

Conditioned Slicing of Interprocedural Programs

Madhusmita Sahu, National Institute of Technology, Odisha, India

ABSTRACT

Program slicing is a technique to decompose programs depending on control flow and data flow amongst several lines of code in a program. Conditioned slicing is a generalization of static slicing and dynamic slicing. A variable, the desired program point, and a condition of interest form a slicing criterion for conditioned slicing. This paper proposes an approach to calculate conditioned slices for programs containing multiple procedures. The approach is termed Node-Marking Conditioned Slicing (NMCS) algorithm. In this approach, first and foremost step is to build an intermediate symbolization of a given program code and the next step is to develop an algorithm for finding out conditioned slices. The dependence graph, termed System Dependence Graph (SDG), is used to symbolize intermediate presentation. After constructing SDG, the NMCS algorithm chooses nodes that satisfy a given condition by the process of marking and unmarking. The algorithm also finds out conditioned slices for every variable at every statement during the process. NMCS algorithm employs a stack to save call context of a method. Few edges in SDG are labeled to identify the statement that calls a method. The proposed algorithm is implemented, and its performance is tested with several case study projects.

KEYWORDS

Conditioned Slice, Dynamic Slice, Program Slicing, Static Slice, System Dependence Graph

Note: This is an extended version of the paper published in *Proceedings of 3rd International Conference on Computational Intelligence in Data Mining (ICCIDM 2016)*, Bhubaneswar, 2016 (Sahu et al., 2016).

1. INTRODUCTION

Program slicing is a decomposition technique utilized to decompose programs depending on control flow and data flow amongst several lines of code in a program code. It is a kind of program analysis technique. It takes out statements related to computation of a variable's value at a specified point in program. The pulled out statements, containing assignment and predicate statements, constitute a program slice. These statements may affect or be affected by value of variable *v* at program location *l*. A slice is computed by employing a slicing criterion. The tuple $< l, v >$ is regarded as a slicing criterion. The slice may be static or dynamic according to input to program code. It is said to be static when it extracts all statements from a program code w.r.t. a slicing criterion regardless input to program (Weiser, 1981). On the other hand, it is said to be dynamic when all statements from a program are extracted w.r.t. a slicing criterion for a specific input to program code (Korel & Laski, 1988).

Program slices are computed in two steps. The first and foremost step is concerned with the construction of an intermediate symbolization of program code. In next step, an algorithm is applied

DOI: 10.4018/IJRSDA.2019010103

to that intermediate representation to find out slices. Program slicing has been employed in many areas of software engineering like debugging, software maintenance, testing, functional cohesion, software refactoring, software quality assurance, etc.

Conditioned slicing is a generalization of static slicing and dynamic slicing (Canfora et al., 1998). A conditioned slice is in the form of a tuple $<Pr,lc,q>$, where *Pr* is a condition, *lc* is a required statement in program code and *t* is a variable. Conditioned slicing puts away those chunks of original program that cannot affect variables at required statement upon satisfaction of conditions. A conditioned slice is computed in two steps: first, the program is simplified with respect to condition provided in slicing criterion. Thus, statements, not satisfying given condition, are removed. Then, a slice is computed on the reduced program. The reduced program is referred to as a conditioned program. More details on conditioned slicing can be obtained in (Canfora et al., 1998; Cheda et al., 2008; Danicic et al., 2000; (Danicic et al., 2004; Fox et al., 2004; Harman et al., 2001; Hierons et al., 2002).

Motivation

Static slicing does not take into account the information about execution state of the program code. Thus, static slices are constructed irrespective of the input to the program code. Dynamic slicing utilizes the complete information about execution behavior of the program. Thus, dynamic slices are dependent on input to program code. There must be a slicing technique that preserves the execution behavior of the program and is independent of input to the program. Conditioned slicing solves this problem by computing the slices at a particular program point for a variable with respect to a condition. Nowadays, most of the programs are interprocedural in nature. There is hardly any work done on conditioned slicing of interprocedural programs. This paper demonstrates a technique to find out conditioned slices of programs containing multiple procedures.

Objectives

The objective of this work is to propose an algorithm to determine conditioned slices of interprocedural programs using an intermediate representation, a dependence graph. The authors also aim at computing slice time for various programs of different lines of code.

The structure of the rest of paper is done as per following ways. Section 2 delivers some background details of the proposed technique. In Section 3, literature survey is discussed. In Section 4, the proposed approach, i.e., Node-Marking Conditioned Slicing (NMCS) algorithm for interprocedural programs is discussed. Section 5 outlines complexity analysis of NMCS algorithm. In Section 6, the correctness of NMCS algorithm is established. Section 7 provides the implementation and experimental results of the proposed technique. Section 8 provides conclusions and future works.

2. BACKGROUND

This section discusses some basic concepts required to understand the proposed work.

2.1. System Dependence Graph (SDG)

Several researchers have contributed towards the area of program slicing. For a given slicing criterion, a slice can be determined manually for a simple program with less complexity. But, with increasing size and complexity of the programs, automatic slice computation is of

greatest importance. Current automated slicing techniques require that the information available in a program source code be first transformed into some intermediate representation and then the slicing technique be applied. The different types of program representations include control flow graph (CFG), program dependence graph (PDG), system dependence graph (SDG). Details of program representations can be discovered in (Binkley & Gallagher, 1996; Horwitz et al., 1990; Mohapatra, 2005).