An Efficient Relevant Slicing Method for Debugging

Tibor Gyimóthy¹, Árpád Beszédés¹, and István Forgács²

¹ Research Group on Artificial Intelligence, Hungarian Academy of Sciences
Aradi vértanuk tere 1., H-6720 Szeged, Hungary, +36 62 454139
{gyimi,beszedes}@inf.u-szeged.hu

² Computer and Automation Institute, Hungarian Academy of Sciences
Kende u. 13-17., H-1502 Budapest, Hungary, +36 1 4665644
forgacs@sztaki.hu

Abstract. Dynamic program slicing methods are widely used for debugging, because many statements can be ignored in the process of localizing a bug. A dynamic program slice with respect to a variable contains only those statements that actually had an influence on this variable. However, during debugging we also need to identify those statements that actually did not affect the variable but could have affected it had they been evaluated differently. A relevant slice includes these potentially affecting statements as well, therefore it is appropriate for debugging. In this paper a forward algorithm is introduced for the computation of relevant slices of programs. The space requirement of this method does not depend on the number of different dynamic slices nor on the size of the execution history, hence it can be applied for real size applications.

Keywords: Dynamic slicing, relevant slicing, debugging

1 Introduction

Program slicing methods are widely used for debugging, testing, reverse engineering and maintenance (e.g. [7], [17], [5], [8]). A slice consists of all statements and predicates that might affect the variables in a set \( V \) at a program point \( p \) [19]. A slice may be an executable program or a subset of the program code. In the first case the behaviour of the reduced program with respect to a variable \( v \) and program point \( p \) is the same as the original program. In the second case a slice contains a set of statements that might influence the value of a variable at point \( p \). Slicing algorithms can be classified according to whether they only use statically available information (static slicing) or compute those statements which influence the value of a variable occurrence for a specific program input (dynamic slice).

In this paper we are concerned with using slicing methods in program debugging. During debugging we generally investigate the program behaviour under the test case that revealed the error, not under any generic test case. Therefore,
dynamic slicing methods are more appropriate than static ones. By using a dynamic slice many statements can be ignored in the process of localizing a bug. Different dynamic slicing methods for debugging are introduced in e.g. [12], [11], [3]. In [3] Agrawal and Horgan presented a precise dynamic slicing method which is based on the graph representation of the dynamic dependences. This graph, called a Dynamic Dependence Graph (DDG) includes a distinct vertex for each occurrence of a statement. A dynamic slice created from the DDG with respect to a variable contains those statements that actually had an influence on this variable. (We refer to this slice as the DDG slice). However, during debugging we also need to identify those statements that actually did not affect the variable but could have affected it had they been evaluated differently. In the next section we are going to present a simple example, where the DDG slice with respect to a variable does not contain such statements which may affect the value of the variable. (We can say that the value of the variable is potentially dependent on those statements.)

In [4] Agrawal et al. introduced a new type of slicing, called relevant slicing for incremental regression testing. The relevant slice is an extension of the DDG slice with potentially dependent predicates and their data dependences. They suggested relevant slicing to solve the problem of determining the test cases in a regression test suit on which the modified program may differ from the original program. Since a relevant slice with respect to a variable occurrence includes all statements which may affect the value of the variable, and this slice is as precise as possible, therefore we would like to emphasize that relevant slices are very suitable for debugging as well. The DDG slice component of the relevant slice is precise, but to determine potential dependences we need to use static dependences as well. The static dependence may be imprecise for a program including e.g. unconstrained pointers, hence we may not obtain a precise relevant slice using imprecise static information. Note that not only the relevant slicing method requires static information. Any other algorithm for computing potential dependences has to use the same imprecise static dependences.

In the method introduced in [4] the DDG representation is used for the computation of the relevant slice. The major drawback of this approach is that the size of the DDG is unbounded. Although Agrawal and Horgan suggested a method for reducing the size of the DDG [10], even this reduced DDG may be very huge for an execution which has many different dynamic slices. Therefore, the method of Agrawal et al. could not apply for real size applications where for a given test case millions of execution steps may be performed.

In this paper we introduce a forward computation method for relevant slices. First, we give a forward algorithm to compute a dynamic slice which is equivalent to the DDG slice, then we augment this algorithm with the computation of the potential dependences. The main advantage of our approach is that the space requirement of this algorithms is very limited compared to that of Agrawal et al. In our case the space requirement for the computation of the dynamic slice is $O(m^2)$, where $m$ is the total number of the different variables, predicate statements and output statements in the program. To determine potential de-