Clauser: Clause Slicing Tool for C Programs

Mohammad M. A. Abdallah\(^1\)* and Hanadi A. Tamimi\(^2\)

\(^1\)Department of Software Engineering
Al-Zaytoonah University of Jordan
Amman, Jordan
\(^2\)Department of Computer Science
University of Koblenz-Landau
Koblenz, Germany
\(^1\)m.abdallah@zuj.edu.jo, \(^2\)htamimi@uni-koblenz.de

Abstract

The Clause slicing technique is a static slicing technique. The clause slice criterion is the clause, which is the smallest part of the code line, with the clause number. In this paper, we introduce the "Clauser," which is a new clause slicing tool for C programs.

The Clauser is a slicing tool that divides the program code lines into clauses, depending on certain rules, and then it slices the clauses by applying the rules of clause slicing and returns the slices of the slice criterion. Compared to other static slicing techniques, clause slicing is more accurate because it considers all code phrases that may affect the program flow. Implementation results showed that the Clauser succeeded in automating a part of clause slicing.

Keywords: Program slicing; Clause slicing; Clauser

1. Introduction

Program slicing was introduced in 1979 by Weiser [1] to address the points of interest in a program depending on the slicing criterion. Few years later, other slicing techniques were developed; such as dynamic slicing, object oriented slicing, etc.

Clause slicing is a recent slicing technique. It was introduced in 2012 by Abdallah [2] as a part of a program robustness measurement technique. Clause slicing concentrates on the code syntax of the program, which makes it a static slicing technique.

Clause slicing considers most of the syntax words in the code as potential slicing criteria, which make it more useful in testing and measuring program quality.

However, clause slicing is a manual technique. There is no tool for applying clause slicing automatically. In this research, we present a tool called Clauser. Clauser was built to identify the clauses that can be sliced in the program, give them a number, let the user choose one of them to be sliced, and return the formed slice. Nevertheless, Clauser is still in the early stages of development. So far, it only slices the variables. In the future work, we plan to make it slice every single clause in the code.

This paper is divided into six sections; Section 2 will explore related program slicing and tools that applied them. In Section 3, the clause slicing technique will be described in details showing how it is applied to chosen clause rules. Section 4 introduces the Clauser; the clause slicing tool for C programs. The evaluation of the Clauser, compared with another slicing tools in Section 5. The conclusions and future work will be presented in Section 6.

* Corresponding Author
2. Program Slicing

Program Slicing is “a method used for abstracting from computer programs” [1]. In terms of formula; in Program P, the program slice has a criteria which is represented as \(<s, v>\), where s is the statement number and v is the variable. With respect to slicing criteria, the slice includes only those statements of P needed to capture the behavior of v at s [3]. Program slicing criterion has the main influence on the program slicing classification. More details about program slicing types, applications, tools, and techniques are discussed below.

2.1. Static Slicing

“Static” means that only statically available information is used for computing slices (i.e., all possible executions of the program are taken into account) [4].

Static slicing is a construct built by assigning a point of interest and deleting all statements irrelevant to this point [5-7]. A point of interest is the statement to be sliced. It is signed by the variable v and the line numbered l. This is called the slicing criterion. The construct for variable v at Line l is expressed as s (l, v), where s is the slice we are interested in [8, 9]. Static slicing is considered a code preserving analysis, since it retrieves the code lines without any change [10].

Static slicing can be executable or non-executable [9]. An executable slice is the code produced after the slicing operation (the slice) is compiled and run as a program.

Weiser [1, 6, 7, 11] introduced a program slicing which became known later as Executable Forward Static Slicing. It is Executable because the slice is required to be an executable program. It is Forward because of the direction edges are traversed when the slice is computed using a dependence graph. It is Static because slices they are computed as the solution to a static analysis program (i.e., without considering the program input) [3].

Backward Slicing is computed by gathering statements and control predicts by way of a backward traversal of the programs starting at the slicing criteria [9]. Backward slicing contains the statements of the program which have effect on the criteria slice and it answers the question: “What program components might effect a selected computation?” [10] A static backward slice preserves the meaning of the variable(s) in the slicing criterion for all possible inputs to the program. [10]

Another form of static slicing is Forward Slicing. Forward Slicing traverses data and control dependence edges in Forward direction and answers the question: “What program components might be effected by a selected computation?” [10].

Forward slice captures the impact of its slicing criteria and it is considered as a kind of ripple effect analysis [4, 12]. It contains the set of statements and control predicts that were effect by the computation of the slicing criterion that was presented by the variable v at the program point p [8, 9, 13, 14]. A statement is dependent on the values of the slicing criterion if the values of the statement depend on the values computed at the slicing criterion or if the values computed at the slicing criterion determine whether the slice is executable or not [9].

Forward Slicing is often not executable because the challenge caused by Forward Slicing is defining the semantics captured by a forward slice [5, 9, 15].

Backward and Forward Slicing are computed in a similar way. Binkley and Harman [16] proved that “For a large class of programs, the distribution of forward slices will contain a significantly larger proportion of small slices when compared to the distribution of backward slices.”

A decomposition slicing is a slice used to decompose the program in different components. It is a union of certain slices taken at certain line numbers on a given variable [7, 17].
Decomposition slicing has two parts; the slice and the complement. The slice “captures all relevant computations involving a given variable” [17], where a decomposition slice depends on the variable name only and does not depend on the statement number. The complement is the rest of the program code, which may be considered as a slice that corresponds to the rest of slicing criteria [17].

In this research, we are only interested in Forward and Backward static slicing techniques. Other program slicing techniques such as Conditioned Slicing and Dynamic Slicing will not be discussed.

2.2. Program Slicing Tools

Researchers tried to apply program slicing by introducing new tools or modifying existing tools. Three tools that have been used to compute program slices for C programs are: CSurf, frama-C, and Wisconsin Program-Slicing.

CodeSurfer is related to CodeSonar, which is GrammaTech's automated source-code analysis tool that finds bugs. CodeSurfer is a program-understanding tool that makes manual review of code easier and faster. CodeSonar is an automated bug finder that generates a report of defects in the code.

Many program-understanding tools interpret code loosely. In contrast, CodeSurfer does a precise analysis. Program constructs, including pre-processor directives, macros, and C++ templates, are analysed correctly. CodeSurfer calculates a variety of representations that can be explored through the graphical user interface or accessed through the optional programming API [18].

Frama-C is a code analysis tool which is used only for programs written in the C programming language. It supports static slicing techniques; forward and backward slicing. It also provides dependency analysis.

Frama-C comes with plugins such as Slicing and Value analysis. Frama-C allows these plugins to collaborate together. It also allows the users to insert and run their own plugins and to connect them with other plugins in Frama-C. However, it still needs to be improved to support other types of slicing such as dynamic slicing. [19]

Wisconsin Program-Slicing is also a program slicing tool. It can perform Forward Slicing, Backward slicing, and Chopping. In addition, it includes a package for building and manipulating control-flow graphs and program dependence graphs. The Wisconsin Program-Slicing tool was developed and tested only on Sun OS 5.5.1 which made it less known than other tools.[20]

3. Clause Slicing

Clause slicing is a new slicing technique that was introduced and defined in [2]. Clause slicing was introduced to facilitate the Robustness Measurement of a C program.

Clause slicing is a syntax-reserved technique, which makes it a static slicing technique. For this reason, clause slicing can be a forward, backward, or decomposition clause type of slice. In this paper, only the forward clause slice will be discussed.

A Clause can be defined as the minimum piece of code that can be sliced. Some clauses are not sliceable, such as #include and break, and they are called the un-sliceable clauses.

The slicing criterion for the clause slicing is <C, n>, where C is the clause, and n is the clause number. The Clause slicing (Cn) is all clauses in the program that depend on clause slicing criterion <C, n>.

Clause slicing was originally introduced for the C programming language. Therefore, the following rules may not be applicable to other programming languages. However, the clause slicing idea may be applicable to other programming languages.

There are 12 main rules that can be applied to a C program to clause slice it. These rules can be defined as follows [2]:
break-statement, continue-statement, goto-statement, reserved words, and punctuations are not Clauses.

Type-variable name has one Clause.

\[expression = expression\] has the sum of the number of clauses in the both expressions.

compound-statement = \{declaration-list, statement-list\} has one clause.

return-statement has one clause.

while-statement = while (expression) \{statement(s)\} has the number of clauses in the statement plus the number of clauses in the expression.

do-statement = do statement while (expression) has the number of clauses in the statement plus the number of clauses in the expression.

for-statement = for (initialization-expression; control-expression; iteration-expression) \{statements\} has the number of clauses in the initialization-expression plus control-expression, and iteration-expression added to the number of clauses in the statements.

if-statement = if(expression) \{statements\} has the number of clauses in the statement added to the number of clauses in the expression.

if-else-statement = if(expression) \{statements\} else statement has the number of clauses in the statement plus the number of clauses in the expression.

switch-statement= switch(expression) \{declaration-list, statement-list, case-list\} has the number of clauses in the expression plus the number of clauses in the declaration-list plus the number of clauses in the statement-list.

Type-function name (parameters-set) \{statements\} has one clause, the Type-function name, added to the number of clauses in (parameters-set) which is equal to the number of parameters, plus the number of clauses in the statement.

4. Clauser

The clause slicing algorithm can be divided into four main steps: Enter the C program, number it, slice it, and show the results.

For the first step, some conditions for the C program need to be satisfied by the program for it to be qualified to enter the Clauser. The program should be compiled with no errors using the gcc compiler where all warnings are ignored. The program must be saved as a text file (.txt extension) to be able to enter the clause slicing tool.

The second step, the numbering technique, is done as follows:

- The C program file (in text format) is read line by line.
- Each line is broken into tokens of strings.
- Add spaces before and after each token, except \#include and return.
- The token is then pushed into a stack.
- The stack will be read, the clauses identified (following the rules mentioned earlier), and saved in an array.
- The array will be printed with an integer counter (clause number).
- Finally, the file is saved under the name (out.txt).

The third step, clause slicing, also has several smaller steps:
• The generated text file (“out.txt”) is read back as an array once again to calculate the number of lines in the code.
• Then, the user is asked to enter the Clause slicing criteria. The input strings are case sensitive.

The Tool takes the entered strings and uses them as parameters passed to the slice function, as \( \text{Slice}(C, N) \) where \( C \) is the clause and \( N \) is the clause number. This function will handle the Clause slicing method and bring it into practice. Depending on the position of the chosen clause on the data dependency, the slice function will retrieve all the clauses that depend on the chosen clause.

Finally, in the fourth step, the array will be printed as the required slice.

5. Evaluation

Clauser is the first tool to apply the Clause slicing technique. It is still in the early stages of development, where at this stage it only slices the variables. Slicing methods of other clauses are still under development. Nonetheless, Clauser succeeded in proving that clause slicing can be applied automatically.

Since there are no other tools that apply clause slicing, Clauser will be compared with other slicing tools that apply forward static slicing.

Figures 1, 2 and 3 show the difference between clause slicing and Forward Slicing. In Figure 1, a simple C program, which calculates the factorial of a number, is to be sliced. Two types of numbers are shown in Figure 1: line numbers, which number each line of code and used in Forward Slicing applied by frama-C, and Clause number, which is written in superscript format and used for Clause slicing based on rules addressed and applied by Clauser.

By applying the same slicing criteria (fact =1) in Line 5 using forward static slicing and the same one with Clause 5 for clause slicing it shows the differences between the Slices using the two different techniques.

In Figure 2, the program is sliced using Forward Slicing, and Figure 3 shows how the program is sliced using clause slicing.

As noticed in Figures 2 and 3, clause slicing has more chunks in the slice than Forward slicing. However, that will cause clause slicing to have more details than Forward slicing. Also, Clauser specifies exactly what is going to be effected by the clause slice criteria, where forward slicing only highlights the statement line which may have more than one logical statement.

These details help in applying clause slicing to applications where slicing techniques with less details cannot be applied. Since Clauser is a slicing tool that runs the clause slicing techniques, it may help in measuring program robustness automatically. In addition, Clauser can be used to measure code quality regarding some standard criteria such as MISRA C [2], which considers every single word in the code syntax.

```c
#include <stdio.h>
int main()
{
    int c;
    int n;
    int fact = 1;
    printf("Enter a number\n");
    scanf("%d", &n);
    return 0;
}
```
for (c^11 = 1; c^12 <= n^13; c++^14)
    fact^15 = fact^16 * c^17;
printf("Factorial of %d = %d\n", n^19, fact^20);
return 0^22;
}

The Program has 11 lines and 22 Clauses

Figure 1. Factorial Program

5     int fact = 1;
9     fact = fact * c;
10    printf("Factorial of %d = %d\n", n, fact);

Forward Slicing on C= (fact, 5) has produced 3 lines.

Figure 2. Forward Slice on (fact, 5)

int fact^5 = 1;
fact^14 = fact^15 * ;
printf("Factorial of %d = %d\n", n^19, fact^21);

Clause slicing on (C5 {int fact=1}) has produced 6 Clauses.

Figure 3. Clause Slice on C5= (fact=1, 5)

Clauser has the advantage of allowing the developer to number the code without slicing it, which can be useful for further practice such as code analysis.

For now, our Clauser tool can slice variables in all different positions where it may be applied, such as in an initialization statement, as a parameter in a function, in an assignment statement, or in the definition condition. Thus, the Clauser tool still needs more work to be able to slice more language features.

6. Conclusion and Future Work

Clauser is a static slicing technique that was introduced recently. The Clauser tool introduced in this paper is the only tool that automates Clause slicing. Clauser divides the program into clauses depending on the 12 rules of clause slicing and numbers them. Then, it allows the user to choose the clause to be sliced and returns the selected slice.

Clauser can be used to number code clauses to help in applying rules of coding standards such as MISRA C. It can also help analyze the code, which is useful in code maintenance, regression testing, and debugging.

Clauser still needs to be improved to become applicable to all C programs. Now, it only slices the variables. In future work, Clauser will be developed to slice all clauses in a C program.
References


Authors

Mohammad M. A. Abdallah, He received his B.S. degree in Computer Science from Al-Zaytoonah University of Jordan and his MSc degree in Software Engineering from Bradford University. He received PhD in Software Engineering from Durham University. He is the Chair of Software Engineering Department in Al-Zaytoonah University of Jordan. His research interest is Software Quality and Program Analysis.

Hanadi A. Tamimi, She received her BSc degree in Computer Science from Jordan University in January 2014 and worked as QA tester for a year. Nowadays, she is preparing her MSc Web Sciences in University of Koblenz – Landau, Germany. Her research interest is Software Quality and Web development.