International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278–9960; ISSN(E): 2278–9979
Vol. 9, Issue 2, Feb–Mar 2020; 1–6
© IASET

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

A STUDY OF TYPES OF DEAD CODES AND THEIR SOLUTIONS

***Rekha Naug*[1] *& Kavita*[2]**

[1]*Research Scholar, Jayoti Vidhyapeeth Women University, Jaipur, Rajasthan, India*

[2]*Associate Processor, Jayoti Vidyapeeth Woman University, Jaipur, Rajasthan, India*

ABSTRACT

Whenever a complex code is written to provide computational solutions for some task or to make some business activities faster and accurate, or to create some new features to existing software application, it is possible that some piece of code later become dead code. This dead code does not play any role in result of the software application, but makes the program bulkier and slow. To make the complex code less bulkier and faster execution, we should identify and remove the dead code. Sometimes, it is troublesome to identify and eliminate it because some other pure code might have dependency on it. Many researchers have worked on this problem and concluded their results. The Adam Fischbach, John Hannan[1] proposed their research work to eliminate useless variables and utilization of weak form of dependent variables. The researchers have used type interface based approach for useless variable elimination and useless code elimination. We have defined two algorithms to identify and eliminate dead code of the c program.

KEYWORDS: *Computational, Dead, Complex, Troublesome, Bulkier*

Article History

Received: ***02 Mar 2020*** **|** ***Revised:*** ***06 Mar 2020*** **|** ***Accepted:*** ***13 Mar 2020***

INTRODUCTION

Dead code identification and elimination play significant role for the program execution. This helps the programmer to create pure useful code that has faster execution. One another necessity of dead code elimination is to optimize memory space allocation. It might be possible that we may not be able to eliminate whole dead code because of dependency of other statement on it. Identification and elimination of partial dead code can be achieved by changing branching structure [2].

Dead code is not written purposely, there are many reasons for the appearance of dead code in a program. Software ageing is one of them. When coder adds some new features to existing code, then occurrence of dead code is possible. Many PHP techniques can be developed for the evaluation and elimination of dead code in web applications [4]. If a program has dead code in it, then it is possible that more test cases are needed for testing. This makes the program testing complex, lengthy and time consuming. Pure code is the solution to overcome from lengthy testing process. In java programs, dead code can be detected and eliminated by enhancing the features of ECJ compiler and by using single assignment property for the variables [5]. Many types of dead code can occur in a program.

A variable can be considered as dead if it is not going to be read or used.

Example

```
Function(x)
{
Y=3;
Return(x^2)
}
```

Here, variable y will never be used, so it is considered as dead variable.

In model checking of large software, state explosion is very big problem. One of the causes of explosion is large input range of variables that is used in software and some variables have no assignment of values [6]. They may have assigned empty value which can indicate missing logic in software.

Parameters of a function can be dead if they are not using by the function. Passing as parameter is slightly time consuming process during the calling, so this leads to slowness in programming.

```
Function(x)
{
int y=6;
y=y*2;
return y
}
```

Dead return value of a method can be considered as dead, as it is not participating in the result of software application.

A method is dead if it functions redundant or similar as another method already exists in the same software application.

If a written code is unreachable, than it will also considered as dead code.

CODE OPTIMIZATION

Elimination of Dead or Redundant of variable

A dead variable must be removed from the code for the best management of memory. The size of hash table can also be reduced by identifying and ignoring dead variables [9]. Many researchers have used different techniques to remove dead variable from the code. The value of dead variable does not contribute anything in the program. Such variables may occur due to program transformation. It is possible that transformation can be reformulated to eliminate dead variable and dead parameters in a procedure [10]. A conservative approximation is helpful in finding and marking useful variable [11]. Elimination of dead or useless variable can make the program lighter and less complex. A higher level optimization can be achieved by removing useless variable. This optimization possibly enables to increase the speed of execution [12].

Redundancy of variable is also an impurity for the good program. It makes the program lengthy and complex. Sometimes, the program cannot be understandable. Optimization of constraint logic programming is also important. Continuous manipulation of redundant variable can slow down the execution speed of the program. Data flow analysis technique can be applied to detect redundancy in variables in CLP [13]. In Java programs, dead code can be detected and eliminated by enhancing the features of ECJ compiler and by using single assignment property for the variables [14].

Elimination of Dead Argument Variable

Generally, programmers and techno persons develop tools and techniques to eliminate dead variables declared in main parts or global sections. It is necessary to identify dead or useless argument of a function and remove it, so that intermediate data structure can be managed properly. RAF and FAR algorithms can be applied to eliminate unnecessary argument list in the context of partial evaluation of logic programs [15]. Dead variables of java code can be eliminated using modified eclipse compiler, which can identify unused variable and eliminate it for optimization of memory [18]. In static programming, a function has two types of arguments, one is calling by value and other is calling by reference. The possibility of dead code generation is higher in Call by value function in comparison to call by reference function; hence it is proven that call by reference is the good option for higher programming language [19].

Elimination of Dead Return Value

When a method is returning a value and it is not used by calling variable or function, than it is considered as dead return value. Instructions executed by the processor are dynamically dead, if the values generated by them are not used by the program. To avoid the delay execution of program, this should be managed by the compiler [21].

Elimination of Dead Statements

A program can have a statement that will never execute. This code unnecessarily makes the code complex and difficult to understand. Different tools and techniques can be used to identify and remove this type of code. One of the techniques is program static slicing. Program slicing helps to debug and detect dead statements [14]. In case of two functions in a program, when both of them gives same output, then one of them will be considered as dead function. Pattern matching is a useful technique for recognize any character type recognition in a typical pattern; it can be used to identify redundant and useless program statements. In this technique, a list of structure is created and pattern of every loop structure is compared with it, if not matches and found reachable then it is pure, otherwise it is redundant or dead [20].

CONCLUSIONS

In this paper, we introduced different types of research work about dead codes that occurs in the program due to many reasons and its solutions. It is not possible to make the code hundred percent pure, but we can use different types of tools and techniques to eliminate dead code. According to research, pure code uses optimized memory allocation and results in faster execution. Testing of dead code free program is much accurate as compared to program having dead code.

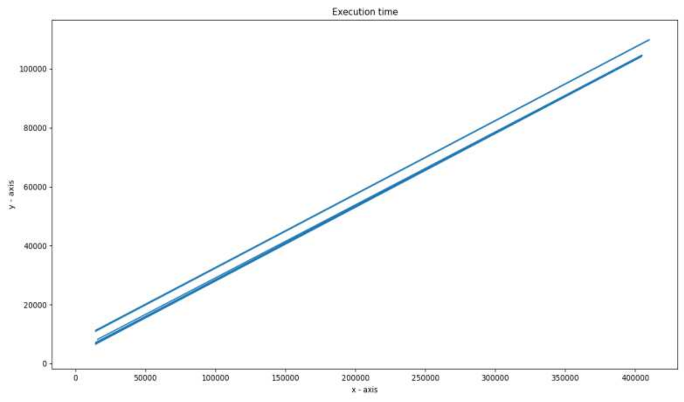

Figure 1.

From the comparison graph, it can be noted that the difference between normal execution of program and execution of dead code included program is in 30-50 nano seconds for different complexity algorithms (O(n), O(n2)).

REFERENCES

1. *Adam Fischbach, John Hannan, "Type systems for useless variable Elimination". (2001).*
2. *Jens Knoop, Oliver Ruthing, Bernhard Steffen. "Partial Dead Code Elimination". (1994).*
3. *Mohamed A.El-Zawawg, "Dead code elimination based on pointer analysis for multithreaded programs", Journal of the Egyptian Mathematical Society, volume 20 page 28-37(2012).*
4. *Niels Groot Obbink, Ivano Malavolta, Gian Luca, Patricia Lago, " An Extensible Approach for Taming the Challenges of JavaScript Dead Code Elimination"(2018).*
5. *Rachana Baldania, "Dead code elimination techniques in eclipse compiler in java",(2016).*
6. *Joel P Self, Eric G. Mercer, "Dead Variable Analysis" On The Fly Dynamic(LNCS volume 4595).*
7. *Mayur Naik,"Dynamic Symbolic Execution", (2018).*
8. *Cristian Cader et al, "KLEE: Unassisted and automatic generation of high coverage test for complex system programs".*
9. *Micah Lewis, Michel Jones, "A dead variable analysis for explicit model checking", PEMP ACM symposium, page 48-57.*
10. *Mitchell Wand, Igor Siveroni, "constraint Systems for useless variable elimination", Symposium on principle of programming languages(1999),291-302.*
11. *Olin Shivers, "Useless variable elimination", Carnegie Mellon University Pittsburgh, Penn. 15213-3890.*
12. *Daniel M. Roy, "Implementation of constraint systems for useless variable elimination", Research Science Institute (1998).*
13. *Andrew D. Macdonald, Peter J. Stuckey, Ronald H. C. Yap, "Redundancy of variables in CLP(R) (1993)*

14. *Nour ALAbwaini, Amal Aldaaje, Tamara Jaber, Mohammad Abdallah, "Using program slicing to detect the dead code", (2018).*
15. *Maria Alpuente et al, "Removing Redundant arguments of a function", Algebraic methodology and software technology, 128-130.*
16. *R.Gupta, D A Banson, J Z Fang "Path profile guided partial dead code elimination using predication", (1997)*
17. *Jagdish Bhatia, "Dead code elimination techniques."*
18. *Rachna Baldania, Hiral Karer, Kinjal Mori "Dead code elimination techniques in Eclipse compiler in java",*
19. *Stefano Berardi,MarioCoppo, Ferruccio Damiani, Paola Giannini, "Useless code elimination for higher order programs", (2000).*
20. *Hongwei Xi, "Dead code elimination through dependent type",(1999).*
21. *MarianneJ. Jantz, PrasadA. Kulkarni, "Understand and Categorize Dynamic Dead Dynamic Dead Instructions for contemporary Architecture."*