# Survey of Software verification for real world applications

Candidate No: 105936

15th January 2019

# Contents

**1 Introduction** **3**

**2 Current Problem** **3**

**3 Software Verification Overview** **3**

**4 Existing Software verification techniques** **3**
4.1 Abstract Static Analysis/Interpretation . . . . . 3
4.2 Symbolic Execution . . . . . 4
4.3 Full Deductive (Theorem Proving) . . . . . 4
4.4 Model Checking . . . . . 4
4.4.1 Bounded Model Checking . . . . . 4

**5 Existing Verification Tools** **4**
5.1 Extended Static Checker for Java[5] . . . . . 4

**6 How to bring software verification into the mainstream** **5**

**7 Conclusion** **5**

**References** **6**

# 1 Introduction

Generally software verification is a very interesting topic in research at the moment, however it is currently limited to the field of researchers and it it is only really used as a part of demonstration software and only as a part of verifiable languages. Therefore this paper will look into how the software verification techniques can be applied to applications designed for use in the real world. This will obviously have advantages as it guarantees code free of fatal runtime errors and reduces the likelihood of other errors.

# 2 Current Problem

Currently if we compare software verification to unit testing as a form of guarantee against bugs we see that unit testing is far more popular and there are far more frameworks available to use for unit testing than there are for verifying software. Also, if you look at the current Computer Science Degree at the University of Sussex unit testing is taught in the first year and software verification is not taught until the masters level. This paper will look into the different software verification techniques and why they are not used more often.

# 3 Software Verification Overview

Software verification can be looked at as the automatic analysis of a program. One simple example of program verification is type checking which has been implemented in a number of programming languages and is quite common. There are also more complex areas of program verification such as extended static checking and full functional program verification.

# 4 Existing Software verification techniques

There are three types of static analysis are Abstract static analysis, model checking and, bounded model checking.[1]

## 4.1 Abstract Static Analysis/Interpretation

This verification technique was introduced in [2] as a way of reducing runtime errors, they saw that strong typing was a start in reducing run time errors and then went on to look into how to make pointers safer. Static analysis allows for the analysis of a program without actually executing the program. The way in

which it works is by computing a superset of possible values for each stage of the program. You can then look at the sets for example if one of the set of values for a divisor is zero then you may have a divide by zero error. Obviously if these are not in the set of values you can guarantee that the program does not contain divide by zero errors.

## 4.2 Symbolic Execution

## 4.3 Full Deductive (Theorem Proving)

## 4.4 Model Checking

This involves looking at a program as a set of states and transitions an algorithm can then check the reachable states of the program an find if there is a case where the program may not terminate[1]. It is possible to provide properties to clarify and restrict variables as explained in the following quote. "In general, properties are classified to 'safety' and 'liveness' properties. While the former declares what should not happen (or equivalently, what should always happen), the latter declares what should eventually happen."[3] which allow you to show that bad states are inaccessible.

### 4.4.1 Bounded Model Checking

This was introduced in a 1999 paper by Biere et al.[4]. As an effort to reduce the complexity It is a development on Model checking however due to limited computing power and the growing complexity of programs it became difficult to run model checking on programs therefore BMC allows for checking with a limited number of steps.

# 5 Existing Verification Tools

As a part of looking into how to include software verification in real world applications we need to look at how they can be used in mainstream languages therefore in the following sections I have described a number of tools and what features they support in verifying programs written in mainstream languages.

## 5.1 Extended Static Checker for Java[5]

This is a checker which finds common programming errors, this may be one of the problems with these tools they are labelled as tools which find bugs rather than tools which guarantee code quality if you are a programmer generally you

don't like finding bugs in your code. Programmers need to write annotations for the program. "The checker is powered by verification-condition generation and automatic theorem proving techniques."[5] . This works on a modular level meaning any method or constructor can be verified

Look into how this works more

# 6 How to bring software verification into the mainstream

[5] shows how to add static checking to Java programs but may not go into how to allow for some parts to be checked and others not to be checked. Possibly through the use of multiple languages for example the .NET enviroment has many languages which can be compiled into .NET libraries and used interchangeably and there are also a number of languages which can be compiled to run on the JVM. Therefore is there an example of a language which compiles into .NET or JVM for use with other libraries.

Come up with an idea about how to bring software verification into the mainstream

# 7 Conclusion

# References


[1] V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, Jul. 2008, ISSN: 0278-0070. DOI: 10.1109/TCAD.2008.923410.

[2] P. Cousot and R. Cousot, "Static determination of dynamic properties of generalized type unions," *SIGPLAN Not.*, vol. 12, no. 3, pp. 77–94, Mar. 1977, ISSN: 0362-1340. DOI: 10.1145/390017.808314. [Online]. Available: http://doi.acm.org/10.1145/390017.808314.

[3] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," in *Advances in Computers*, Elsevier, 2003, pp. 117–148. DOI: 10.1016/s0065-2458(03)58003-2.

[4] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *Tools and Algorithms for the Construction and Analysis of Systems*, Springer, Jan. 1, 1999, ISBN: 978-3-540-65703-3. DOI: 10.1007/3-540-49059-0_14. [Online]. Available: http://dx.doi.org/10.1007/3-540-49059-0_14.

[5] C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata, "Extended static checking for java," in *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, ser. PLDI '02, Berlin, Germany: ACM, 2002, pp. 234–245, ISBN: 1-58113-463-0. DOI: 10.1145/512529.512558. [Online]. Available: http://doi.acm.org/10.1145/512529.512558.