Survey of Software verification tools for real world applications

Candidate No: 105936

17th January 2019

Contents

1 Introduction **3**

2 Software Verification Overview **3**

3 Formal Program Verification Tools **3**
3.1 Extended Static Checker for Java[5] 3
3.2 Static Driver Verifier/SLAM[6] 3
3.3 Krakatoa [8], [9] 4

4 Verifiable Languages **4**
4.1 Spec# 4
4.2 Spark Ada 4

5 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 Software Verification Overview

Software verification can be looked at as the automatic analysis of a program. One commonplace example of program verification is type checking which has been implemented in many programming languages. There are also more complex areas of program verification such as extended static checking and full functional program verification. For the purposes of this report I will be calling Languages which have been designed to be verified as a part of compilation "Verifiable Languages". There are a number of these languages which I will go on to outline further in the paper. There are also tools which have been developed to verify existing languages

3 Formal Program Verification Tools

3.1 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.

3.2 Static Driver Verifier/SLAM[6]

This was designed to allow Microsoft to verify drivers for their operating systems. The actual tool used to verify drivers was called Static Driver Verifier while slam is the analysis engine it uses. SLAM is a tool which can "check that a C program correctly uses the interface to an external library"[6]. I expect that this is one of the best examples of mainstream software verification uses. This is probably

because the SDV(Static Driver Verifier) tool is included as a part of the driver developer kit for windows this means that developers have to make sure that their driver is verified before they can release it.

3.3 Krakatoa [8], [9]

This is a verification tool which runs can be run on programs written in Java. Contracts for the code are written in the comments for a method the come from the Java Modelling Language[10]. This tool is integrated into the Why IDE which was designed to make it easier to verify programs.

4 Verifiable Languages

4.1 Spec#

This is not a tool as such it is a language which adds various static analysis features which are checked during compilation. This runs on the .NET framework therefore can be used with other .NET languages. Many developers have easy access to the language through visual studio "The Spec# system is fully integrated into the Microsoft Visual Studio environment."[7]. Barnett et al. also lists several features which Spec# adds to C# such as:

- "type support for distinguishing non-null object references from possibly-null object references"[7]
- "method specifications like pre- and postconditions"[7]
- "support for constraining the data fields of objects"[7]

Because this works on the .NET framework functions can be called from other languages meaning it is possible to call a method with parameters which violate the preconditions in this case Spec# provides the option to specify an exception to be thrown in the case where a precondition is not met.

4.2 Spark Ada

Spark is a Verifiable language based on Ada it uses pre and post conditions as contracts for verification. It appears to be used on a number of mission critical pieces of software mainly on embedded systems.

5 Conclusion

In conclusion there seems to be many tools and many verifiable languages although there doesn't appear to be any which compile to the JVM which could be the reason why it is not more widespread. I would expect formal verification to grow in the future but as mentioned earlier there several people already predicted this and this obviously has not happened.

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.

[6] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, "SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft," in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2004, pp. 1–20. DOI: 10.1007/978-3-540-24756-2_1. [Online]. Available: https://www.microsoft.com/en-us/research/publication/slam-and-static-driver-verifier-technology-transfer-of-formal-methods-inside-microsoft/.

[7] M. Barnett, K. R. M. Leino, and W. Schulte, "The Spec# programming system: An overview," in *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*, Springer Berlin Heidelberg, 2005, pp. 49–69. DOI: 10.1007/978-3-540-30569-9_3.

[8] C. Marché, *The Krakatoa Verification Tool for JAVA programs*, Version 2.41, INRIA Team Toccata, Batiment 650, Université Paris-Sud 91405 Orsay cedex, France, Jun. 2018. [Online]. Available: krakatoa.lri.fr/krakatoa.pdf.

[9] J.-C. Filliâtre and C. Marché, "The Why/Krakatoa/Caduceus platform for deductive program verification," in *Computer Aided Verification*, Springer Berlin Heidelberg, 2007, pp. 173–177. DOI: 10.1007/978-3-540-73368-3_21.

[10] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll, "An overview of JML tools and applications," *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 3, pp. 212–232, Dec. 2004. DOI: 10.1007/s10009-004-0167-4.

[11] I. Feinerer and G. Salzer, "A comparison of tools for teaching formal software verification," *Formal Aspects of Computing*, vol. 21, no. 3, pp. 293–301, Jun. 2008. DOI: 10.1007/s00165-008-0084-5.

[12] R. Feldt, R. Torkar, E. Ahmad, and B. Raza, "Challenges with software verification and validation activities in the space industry," in *2010 Third International Conference on Software Testing, Verification and Validation*, IEEE, 2010. DOI: 10.1109/icst.2010.37.

[13] S. D. Nelson and C. Pecheur, "Formal verification for a next-generation space shuttle," in *Formal Approaches to Agent-Based Systems*, Springer Berlin Heidelberg, 2003, pp. 53–67. DOI: 10.1007/978-3-540-45133-4_5.

[14] G. Hunt, J. Larus, M. Abadi, M. Aiken, P. Barham, M. Fähndrich, C. Hawblitzel, O. Hodson, S. Levi, N. Murphy, B. Steensgaard, D. Tarditi, T. Wobber, and B. Zill, "An overview of the Singularity project," Microsoft, Microsoft Research One Microsoft Way Redmond, WA 98052, Tech. Rep., Oct. 2005. [Online]. Available: https://www.microsoft.com/en-us/research/publication/an-overview-of-the-singularity-project/.

[15] A. Romero and J. Divasón, "Experiences and new alternatives for teaching formal verification of Java programs," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018*, ACM Press, 2018. DOI: 10.1145/3197091.3205811.