A Verified OCPP v2.0 Server for Electric Vehicle Charger Networks

An Annotated Bibliography

Daniel McInnes
University of Queensland

August 12, 2019

References

[1] *Deductive Software Verification The KeY Book: From Theory to Practice*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, vol. 10001.

> This book describes the KeY software verification framework. The framework uses the Java Modeling Language to formally specify Java programs, and provides both automatic and interactive functional verification. The book was useful for evaluating the different verification tools available.

[2] "Global EV Outlook 2019," International Energy Agency, Paris, France, Tech. Rep., May 2019, Accessed: June 25, 2019. [Online]. Available: https://webstore.iea.org/global-ev-outlook-2019

> This annual publication describes the state of electric mobility around the world. It includes statistics for the number of publicly available fast chargers, which increased 33% from 107 650 in 2017 to 143 502 in 2018. These numbers show the potential usefulness of a verified OCPP v2.0 server.

[3] 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: International Workshop, CASSIS 2004, Marseille, France, March 10-14, 2004, Revised Selected Pa-*

pers, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 3362, pp. 49–69.

This article describes the Spec# software verification system. The Spec# language is a superset of C#, which uses Boogie to check specifications statically. The article was useful for evaluating the different verification tools available.

[4] F. Bobot, J.-C. Filliâtre, C. Marché, and A. Paskevich, "Why3: Shepherd Your Herd of Provers," in *Boogie 2011: First International Workshop on Intermediate Verification Languages*, Wroclaw, Poland, 2011, pp. 53–64. [Online]. Available: https://hal.inria.fr/hal-00790310

This article describes the Why3 software verification platform. All of the authors are members of the "Toccata" project, which developed Why3, and therefore have a vested interest in portraying the platform in a positive light. It was useful for evaluating the different verification tools available.

[5] E. Cohen, M. Hillebrand, S. Tobies, M. Moskal, and W. Schulte, "Verifying C Programs: A VCC Tutorial," University of Freiburg, Leuven, Belgium, Tech. Rep., July 10 2015, accessed: June 22, 2019. [Online]. Available: https://swt.informatik.uni-freiburg.de/teaching/SS2015/swtvl/Resources/literature/vcc-tutorial-col2.pdf

This tutorial gives useful examples of how to use the VCC verification tool. It is not obvious from the project webpage what features the tool provides, and the tutorial was used to infer these features. The tutorial indicates that is a "working draft, version 0.2", and it is not available on the Microsoft website, rather it is available indirectly via the University of Freiburg. In spite of this, it is well written, and it was easy to follow the examples and perform the proofs.

[6] E. Cohen, S. Tobies, M. Moskal, and W. Schulte, "A Practical Verification Methodology for Concurrent Programs," Microsoft Research, 1 Microsoft Way, Redmond, WA, Tech. Rep., February 12 2009, accessed: June 22, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-practical-verification-methodology-for-concurrent-programs/

This paper describes the verification methodology used in 'VCC', an automated, sound C verifier, which translates annotated C code into BoogiePL. Boogie generates verification conditions, which are then fed into the Z3 SMT solver. It was useful for evaluating the different verification tools available.

[7] D. Cok, "Openjml: Software verification for java 7 using jml, openjdk, and eclipse," vol. 149. Open Publishing Association, 2014, pp. 79–92.

This paper describes 'OpenJML', a verification tool for Java programs. It has a section on the soundness of the tool, which is vague and inconclusive. The paper is useful for evaluating the different verification tools available.

[8] J.-C. Filliâtre and A. Paskevich, "Why3 – Where Programs Meet Provers," in *ESOP'13 22nd European Symposium on Programming*, ser. LNCS, vol. 7792. Rome, Italy: Springer, Mar. 2013. [Online]. Available: https://hal.inria.fr/hal-00789533

This article describes the Why3 software verification platform, and the WhyML language. WhyML is used for both specification and programming. It was useful for evaluating the different verification tools available.

[9] B. Jacobs, J. Smans, and F. Piessens, "The Verifast Program Verifier: A Tutorial," Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, Tech. Rep., November 2017, Accessed: June 20, 2019. [Online]. Available: https://people.cs.kuleuven.be/~bart.jacobs/verifast/tutorial.pdf

This report gives useful examples of how to use the VeriFast Program Verifier to prove the absence of buffer overflows, data races and functional correctness as specified by preconditions and postconditions.

[10] U. Juhasz, I. T. Kassios, P. Mller, M. Novacek, M. Schwerhoff, and A. J. Summers, "Viper: A verification infrastructure for permission-based reasoning," 2014.

This paper describes the "Viper" verification infrastructure, the "Silver" intermediate language and the "Silicon" and "Carbon" back-end verifiers. The "Viper" tool provides a verification infrastructure based on permission logics. Three of

the six authors are current members of the Viper project, and therefore have a vested interest in portraying Viper in a positive light. The paper was useful for evaluating the different verification tools available.

[11] K. Leino, "Dafny: An automatic program verifier for functional correctness," vol. 6355, 2010, pp. 348–370.

This article describes the Dafny Automatic Program Verifier. The Dafny verifier translates Dafny source code into Boogie2, which generates first order verification conditions, which are proved by the Z3 SMT solver. The article was useful for evaluating the different verification tools available.

[12] J. W. McCormick, *Building high integrity applications with SPARK*, 2015.

This book describes the Spark programming language. It is not written by the tool vendor, the authors are users and teachers of the technology, so therefore have some vested interest in portraying the technology in a positive light. The book has a more practical approach than many scientific articles. The book was useful for evaluating the different verification tools available.

[13] "VCC: A Verifier for Concurrent C". Microsoft. Accessed:June 22, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/

This is the website for the VCC verification tool. It gives an overview of the product, installation instructions and links to more documentation. It is useful for evaluating the different verification tools available.

[14] "Appraisal OCPP". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: www.openchargealliance.org/about-us/appraisal-ocpp/

This webpage gives the goals of the OCPP protocol. It includes statistics for the number of charging stations (which may consist of 1 or more chargers) using the OCPP protocol, which currently number of 10 000 in over 50 different countries. These numbers show the potential usefulness of a

verified OCPP v2.0 server. The web site is operated by the Open Charge Alliance, whose stated mission is to "Uphold OCPP and OSCP as vital standards, with implementations widely adopted and deployed", i.e., they have a vested interest in portraying the adoption of the OCPP protocol in a positive light. In spite of this vested interest, the data seems trustworthy.

[15] "Does your program do what it is supposed to do?". OpenJML. Accessed: June 22, 2019. [Online]. Available: https://github.com/verifast/verifast/

This is the website for the OpenJML verification tool. It gives an overview of the product, installation instructions and links to more documentation. It is useful for evaluating the different verification tools available.

[16] D. J. Pearce and L. Groves, "Whiley: A platform for research in software verification," in *Software Language Engineering*, M. Erwig, R. F. Paige, and E. Van Wyk, Eds. Cham: Springer International Publishing, 2013, pp. 238–248.

[17] T. Runge, I. Schaefer, L. Cleophas, T. Thüm, D. Kourie, and B. Watson, "Tool support for correctness-by-construction," in *Fundamental Approaches to Software Engineering - 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), R. Hähnle and W. van der Aalst, Eds. Germany: Springer, 1 2019, pp. 25–42.

This paper introduces 'CorC', a graphical and textual IDE for creating software using the 'correctness by construction' methodology. 'CorC' uses the 'KeY' theorem prover. The authors found that when a set of algorithms were developed using the 'correctness by construction' approach, verification was much faster than post-hoc verification. The graphical component is surprisingly effective at making the software and its proof easier to understand.

[18] M. Utting, D. Pearce, and L. Groves, "Making whiley boogie!" vol. 10510. Springer Verlag, 2017, pp. 69–84.

This article describes the Whiley programming language. The article was useful for evaluating the different verification tools available.

[19] "Research prototype tool for modular formal verification of C and Java programs". VeriFast. Accessed: June 22, 2019. [Online]. Available: https://github.com/verifast/verifast/

This website is the home page of the VeriFast project. It gives an overview of the product, installation instructions and links to tutorials and papers. It was useful when evaluating the different verification tools available, particularly the list of known bugs in the tool.

[20] F. Vogels, B. Jacobs, and F. Piessens, "Featherweight Verifast," *Logical Methods in Computer Science*, vol. 11, no. 3:19, p. 157, 2015. [Online]. Available: https://lmcs.episciences.org/1595

In this article Vogels *et al.* discuss VeriFast, a tool for modular verification of safety and correctness properties of single threaded and multithreaded C and Java programs. The authors present a formal definition and soundness proof of a core subset of the VeriFast program verification approach. The article provides a detailed description of what VeriFast does and how it works, then introduces a simplified version of Verifast, "Featherweight Verifast", as well as another variant called "Mechanised Featherweight Verifast". The article is useful mainly for its description of what guarantees VeriFast provides. Surprisingly, these features are not obvious from the VeriFast website https://github.com/verifast/verifast or from internet searches. The main limitation of the article is that it focuses on a subset of VeriFast. The authors list future work to be done to create formal definitions and proofs for those features of VeriFast that are not included in Featherweight Verifast. This article provides useful supplementary information for comparing different software verification tools.