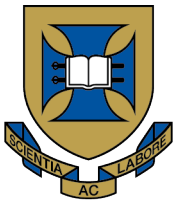


THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

A VERIFIED SERVER FOR AN ELECTRIC VEHICLE CHARGER NETWORK

by

DANIEL HUGH MCINNES

School of Information Technology and Electrical Engineering,
The University of Queensland.

Submitted for the degree of
Master of Engineering
in the field of Software Engineering

October 2019.

15 Blackthorn St
Brisbane, 4077
Tel. 0415208480

June 27, 2019

Prof Michael Brünig
Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Q 4072

Dear Professor Brünig,

In accordance with the requirements of the degree of Master of Engineering in the division of Software Engineering, I present the following thesis entitled "A Verified Server for an Electric Vehicle Charger Network". This work was performed under the supervision of A/Prof. Graeme Smith.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours sincerely,

Daniel McInnes.

Acknowledgments

I wish to acknowledge the support of my supervisor, Associate Professor Graeme Smith, whose expertise and assistance were greatly appreciated.

Abstract

Currently, networks of publicly available electric vehicle fast chargers communicate with servers using the Open Charge Point Protocol (OCPP). Ideally, the software running on these servers would be error free and never crash. Numerous software verification tools exist to prove desirable properties of the server software, such as functional correctness, the absence of race conditions, memory leaks, and certain runtime errors. I compare the different features of several verification tools, and use one of them to partially implement an OCPP server.

Contents

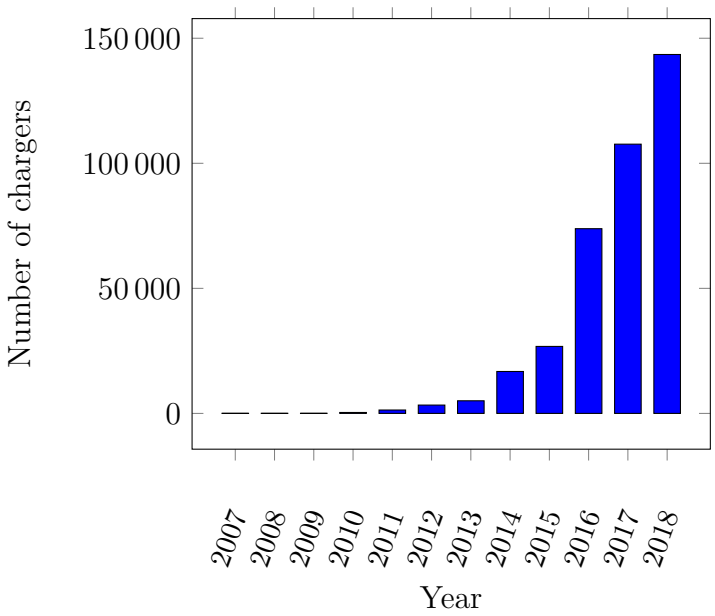
List of Figures

List of Tables

Chapter 1

Introduction

The International Energy Agency [1] reports that the number of publicly available fast chargers (> 22kW) increased from 107 650 in 2017 to 143 502 in 2018.



The Open Charge Alliance [2] reports that more than 10 000 charging stations[1] in over 50 countries are managed using the OCPP protocol.

In this paper I investigate and compare the features of several different software verifiation tools and choose one to partially implement an OCPP server. The tools include VeriFast, VCC, OpenJML, KeY, Spec#, Saphney, Whiley, and AdaSPARK.

The tools vary in what guarantees they provide. Desirable guarantees that I was interested in are:

- the absence of memory leaks

[1] Note that a "charging station" may consist of multiple fast chargers, thus the disparity between 143 502 "fast chargers" and "more than 10 000 charging stations"

- the program should never access uninitialized memory (for example, should never read past the end of an array)
- the program should never crash or exit unexpectedly
- the tool should verify the absense of stack overflows. i.e. the program should be bounded in terms of memory usage at runtime
- the program should verifiably meet its requirements. These requirements are typically in the form of preconditions and postconditions.
- the program should never exhibit undefined behaviour
- the program should constrain information flow, i.e. not leak sensitive information such as passwords
- The tool should be sound. Many of the tools claim to be sound "modulo bugs in the tool", and have lengthy lists of known bugs. I want a tool that has no known unsound behaviour.

Chapter 2

Literature review / prior art

VeriFast claims to be sound (Vogels et al, p1. "VeriFast is a sound modular formal verification approach for single-threaded and mul- tithreaded imperative programs being developed at KU Leuven".)

Verifast guarantees (modulo bugs in the tool) the absence of illegal memory accesses such as null pointer dereferences, accesses of uninitialized memory, accessing outside of the bounds of an array, data race conditions, violations of the user specified function contracts.

You will need to review previous work in the field, which may include books and papers ("literature"), patents and commercial products ("prior art"), and earlier work in your Department. This information is usually (but not always) collected in a single chapter, whose title should preferably be more specific and interesting than the one above.

Chapter 3

Theory

A scientific paper is likely to be read by people who are not specialists in the same field as the author(s), but who nevertheless may need to use the results of the paper in their own fields. Similarly, the examiners of your thesis will probably include at least one academic who does not teach or conduct research in the subject area of your thesis. In an early chapter of your thesis, therefore, you should quote any theoretical results which are necessary for the understanding of later chapters. Examiners who are not specialists in your area will know whether you have given sufficient theoretical information. They will also know whether you have insulted their status by presenting material which is familiar to every half-competent graduate in every field of ECE.

Chapter 4

Methodology, procedure, design, etc.

This may be one chapter or several. Again, titles should be more informative than the above.

You will almost certainly need diagrams to clarify your meaning. The LaTeX 2_ε `graphics` package allows the inclusion of PostScript graphics, as in . The inclusion of LaTeX `picture` graphics, as in , requires no auxiliary packages and allows the mathematical formatting features of LaTeX to be used in diagrams; but the `picture` files, unlike PostScript files, usually require manual editing.

Figure 4.1: Experimental two-way active crossover (op-amp version)

Chapter 5

Results and discussion ...

... or perhaps the discussion should be a separate chapter.

In any case, you will probably need to include tabulated results. Table 5.1 illustrates the use of various LaTeX environments to include a computer printout (plain text file) in a document. The `verbatim` environment, which encloses the formatted text, is also useful for program listings.

Table 5.1: *Fraction of air volume involved in heat exchange for second mode (right column) vs. filling factor (left column). The plain-text headings represent f, m, μ_2 and f_2.*

f(%)	m	mu2	f2(%)
0.016	80.00	0.05400	4.874
0.031	56.57	0.07732	5.438
0.062	40.00	0.11103	6.125
0.125	28.28	0.16001	6.970
0.250	20.00	0.23175	8.020
0.500	14.14	0.33799	9.329
1.000	10.00	0.49789	10.967
2.000	7.07	0.74444	13.008
4.000	5.00	1.13919	15.525
8.000	3.54	1.81095	18.568
19.237	2.28	3.61958	23.174
37.180	1.64	7.28635	27.094
57.392	1.32	14.63631	29.813
74.316	1.16	29.35160	31.453
85.734	1.08	58.79364	32.360

Chapter 6

Conclusions

6.1 Summary and conclusions

6.2 Possible future work

Appendix A

Dummy appendix

Appendices are useful for supplying necessary details or explanations which do not seem to fit into the main text, perhaps because they are too long and would distract the reader from the central argument. Appendices are also used for program listings.

Notice that appendices are "numbered" with capital letters, not numerals. When the `\appendix` command in LaTeX [12, p. 175] is used with the `book` document class, it causes subsequent chapters to be treated as appendices.

Appendix B

Program listings

B.1 First program

Some initial explanatory notes may precede the listing.

B.2 Second program

B.3 Etc.

Appendix C

Companion disk

If you wish to make some computer files available to your examiners, you can list and describe the files here. The files can be supplied on a disk and inserted in a pocket fixed to the inside back cover.

The disk will not be needed if you can specify a URL from which the files can be downloaded.

Bibliography

[1] International Energy Agency, "Global EV Outlook 2019", *International Energy Agency*, May 2019. Available: https://webstore.iea.org/global-ev-outlook-2019. [Accessed June 25, 2019].

2.

3.

4.

5.

6.

7.

8.

[2] Open Charge Alliance, "Appraisal OCPP", todo - undated. [online]. Available: https://www.openchargealliance.org/about-us/appraisal-ocpp/. [Accessed June 22, 2019].

3.

4.

5.

6.

7.

8.

[3] Open Charge Alliance, "Open Charge Point Protocol 2.0", todo - undated. [online]. Available: https://www.openchargealliance.org/protocols/ocpp-20/. [Accessed June 22, 2019].

3.

4.

5.

6.

7.

8.

[4] VeriFast, "Research prototype tool for modular formal verification of C and Java programs", todo - undated. [online]. Available: https://github.com/verifast/verifast/. [Accessed June 22, 2019].
2. This is the website for the VeriFast verification tool.
3.
4.
5.
6.
7.
8.

[5] F. Vogels, B. Jacobs, and F. Piessens, "Featherweight Verifast", *Logical Methods in Computer Science*, Vol. 11(3:19), pp. 1–57, 2015. [Online serial]. Available: https://lmcs.episciences.org/1595. [Accessed June 20, 2019].
2.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.
3. The authors present a formal definition and soundness proof of a core subset of the VeriFast program verification approach.
4. 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".
5. The article is useful to my project 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.
6. The main limitation of the article is that it focuses on a subset of VeriFast.
7. 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.
8. This article will not form the basis of my research, however it will provide useful supplementary information for comparing different software verification tools.

[6] B. Jacobs and F. Piessens, "The Verifast Program Verifier", Department of Computer Science, Katholieke Universiteit Leu-

ven, Belgium, Tech. Report. CW-520, August 2008. Available: https://people.cs.kuleuven.be/ bart.jacobs/verifast/verifast.pdf. [Accessed June 20, 2019].

2. This report describes some aspects of an early version of the VeriFast Program Verifier.

3. Based on the title of this paper, I had hoped it would help me understand the advantages of using VeriFast, however this was not the case. This report is not particularly useful for my project, as it describes the design of an early version of the tool, and does not explain the guarantees that using the tool provides.

4.

5.

6.

7.

8.

[7] B. Jacobs, J. Smans, and F. Piessens, "The Verifast Program Verifier: A Tutorial", Department of Computer Science, Katholieke Universiteit Leuven, Belgium, [online document], November 28, 2017. Available: https://people.cs.kuleuven.be/ bart.jacobs/verifast/tutorial.pdf. [Accessed June 20, 2019].

2. 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.

3.

4.

5.

6.

7.

8.

[8] E. Cohen, M. Moskal, W. Schulte, and S. Tobies, "A Practical Verification Methodology for Concurrent Programs", Microsoft Research, [online document], February 12, 2009. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/concurrency3.pdf. [Accessed June 22, 2019].

2. This document focusses on a methodology for verifying concurrent programs using VCC. It covers some theory behind the tool, but is not particularly

useful for my thesis. It does not provide a list of the features the tool provides, or simple examples to demonstrate its use.
3.
4.
5.
6.
7.
8.

[9] E. Cohen, M. Hillebrand, S. Tobies, M. Moskal, and W. Schulte, "Verifying C Programs: A VCC Tutorial", University of Freiburg, [online document], July 10, 2015. Available: https://swt.informatik.uni-freiburg.de/teaching/SS2015/swtvl/Resources/literature/vcc-tutorial-c [Accessed June 22, 2019].
2. This tutorial gives useful examples of how to use the VCC verification tool.
3.
4.
5.
6. The tutorial indicates that is a "working draft, version 0.2", and I notice that it is not available on the Microsft website, rather it is available indrectly via the University of Freiburg. In spite of this, I found it well written, and it was easy to follow the examples and perform the proofs myself.
7.
8.

[10] M. Moskal, "VCC: A Verifier for Concurrent C", microsoft.com, December 10, 2008. [online]. Available: https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/. [Accessed June 22, 2019].
2. This is the website for the VCC verification tool.
3.
4.
5.
6.
7.
8.

[11] OpenJML, "Does your program do what it is supposed to do?", http://www.openjml.org, todo - undated. [online]. Available: http://www.openjml.org/. [Accessed June 22, 2019].
2. This is the website for the OpenJML verification tool.
3.
4.
5.
6.
7.
8.

[12] L. Lamport, *LaTeX: A Document Preparation System*, 2nd ed. (Addison-Wesley, 1994).

[13] REFERENCE 2

[14] Etc.