THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

A VERIFIED OCPP v2.01 SERVER FOR ELECTRIC VEHICLE CHARGER NETWORKS

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.

Daniel McInnes
s4231125@student.uq.edu.au

May 7, 2020

Prof Amin Abbosh
Acting Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Q 4072

Dear Professor Abbosh,

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

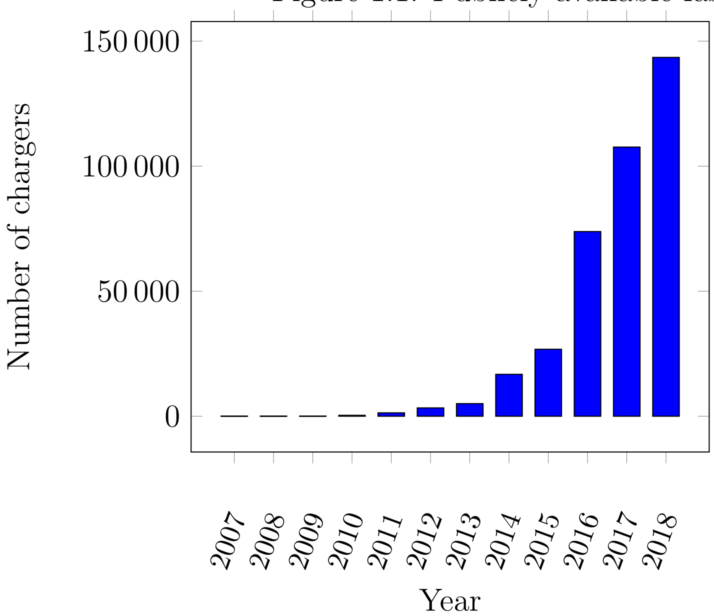
List of Tables

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.

Figure 1.1: Publicly available fast chargers



The Open Charge Alliance [69] 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 Dafny, KeY, OpenJML, SPARK 2014, Spec#, VCC, VeriFast, Viper, Whiley, and Why3.

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

- the absence of memory leaks
- 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 (RAM) 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 wanted a tool that has no known unsound behaviour. 'Soundness' may be further broken down into the following levels.

  Level 1: No false positives: The tool should not report that the software is correct when there are in fact errors. This is fundamental, there is not much point in using the tool if you can't trust it.

  Level 2: No false negatives: The tool should not report that the software is incorrect when it is in fact correct. This is desiriable, but not as important as level 1, as the developer's attention is drawn to the issue prior to the software being released.

Chapter 2

Literature review / prior art

——————————————————— Background (20%): Background material for the thesis should likely include reviews, analyses and discussions of the literature in the area of the thesis and about methods applicable to achieving the thesis goals. This background should not only help the reader understand the rest of the document, but should illustrate to the reader a clear mastery of the material in the topic area and an ability to synthesize and abstract knowledge from other sources.

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.

———————————————————

Numerous software verification tools were considered for use in implementing the OCPP server. These tools include Dafny, KeY, OpenJML, SPARK 2014, Spec#, VCC, Verifast, Viper, Whiley, and Why3. SPARK 2014 was found to best fulfill the criteria 1.

2.1 Dafny

2.1.1 Home Page

https://rise4fun.com/Dafny

2.1.2 Features

Dafny is both a language and a verifier [62]. Dafny supports feature verification via preconditions, postconditions, loop invariants and loop variants. It uses the 'Boogie' intermediate language and the Z3 theorem prover. The Dafny compiler produces executable files for the .NET platform [64].

2.1.3 Soundness

Dafny is designed to be sound but incomplete, and is known to report errors on correct programs [65].

2.1.4 Supported Platforms

Windows, Linux, OSX host, for a .NET target platform.

2.1.5 License Information

MIT.

2.1.6 Evidence of successful use in commercial software development

Internet searches failed to find any evidence of Dafny being used in commercial software development.

2.1.7 Existing Libraries

There is a 'mathematics' library for Dafny.

2.1.8 Multithreaded Application Support

Internet searches failed to find evidence of multithreaded application support.

2.1.9 Supported Languages

Dafny.

2.2 KeY

2.2.1 Home Page

https://www.key-project.org/

2.2.2 Features

KeY offers functional verification for Java programs. The specifications are written as comments in JML in the Java source code. KeY is built on a formal logic called ‘Java Card DL’, which is itself a first-order dynamic logic, and an extension of Hoare logic. It is targeted at JavaCard programs.

2.2.3 Soundness

KeY is thought to be sound. Internet searches failed to find examples of unsoundness.

2.2.4 Supported Platforms

Windows, Linux, and OSX hosts, for a JVM target.

2.2.5 License Information

GPL.

2.2.6 Evidence of successful use in commercial software development

Internet searches failed to find any evidence of KeY being used in commercial software development.

2.2.7 Existing Libraries

There are extensive libraries available for Java programs.

2.2.8 Multithreaded Application Support

No.

2.2.9 Supported Languages

Java.

2.3 OpenJML

2.3.1 Home Page

http://www.openjml.org/

2.3.2 Features

OpenJML is a suite of tools for verifying Java programs that are annotated with JML statements. It is based on OpenJDKv1.8. It detects illegal memory access at compile time. It verifies preconditions and postconditions. It arguably guarantees the absence of undefined behaviour for single threaded applications. It does not constrain information flow.

2.3.3 Soundness

Yes

2.3.4 Supported Platforms

Windows, Linux, OSX.

2.3.5 License Information

GPLv2.

2.3.6 Evidence of successful use in commercial software development

Internet searches failed to find any evidence of OpenJML being used in commercial software development.

2.3.7 Existing Libraries

There are extensive libraries available for Java programs.

2.3.8 Multithreaded Application Support

No.

2.3.9 Supported Languages

Java (only OpenJDK v1.8, may become unsupported in December 2020)

2.4 SPARK 2014

SPARK 2014 is both a formally defined programming language and a set of verification tools. In typical use, a programmer writes SPARK code, which is compiled by the GNAT compiler, then analyzed by the GNATprove tool to produce numerous verification conditions.

GNATprove uses Alt-Ergo (OCamlPro, 2014), CVC4 (NYU, 2014), YICES (Dutertre, 2014) and Z3 (Bjorner, 2012) to prove the verification conditions.

Features

Formally verifies:

- information flow
- freedom from runtime errors
- functional correctness

Safety Standards

SPARK 2014 satisifes:

- DO-178B/C
- Formal Methods supplement DO-333
- CENELEC 51028
- IEC 61508
- DEFSTAN 00-56

Soundness

SPARK is thought to be sound. Internet searches failed to find examples of unsoundness.

Supported Platforms

Windows, Linux, OSX.

License Information

Dual license:

SPARK GPL is available for free from http://libre.adacore.com under the GPL.

SPARK PRO is available under a commercial license from http://www.adacore.com.

Evidence of successful use in commercial software development

There is abundant evidence of the successful use of SPARK in high integrity software development. See: [31], [28], [29], [2], [8], [23], [39], [5], [38], [17], [4], [10], [44], [25], [32], [37], [34], [22], [21], [19], [35], [36], [43], [24], [12], [33], [20], [14], [40], [26], [27], [42], [9], [46], [30], [7], [11], [47], [13], [6], [16], [15], [45], [18], [3].

Of particular relevance is the experience of the CubeSat Laboratory at Vermont Technical College[41]. Cubesats are small cubes launched into space with various sensors onboard, in this case without post-launch software update capabilities. This means the software must be fault free at the time of launch. The students (mostly third and fourth year undergraduates, with no prior knowledge of SPARK or Ada, and a high turnover rate) proved the software to be free of runtime errors. 14 Cubesats were launched in November 2013. Most were never heard from again, but the SPARK Cubesat worked for 2 years until it reentered Earth's atmosphere as planned in November 2015.

Existing Libraries

SPARK has a minimal container library. SPARK interfaces easily with Ada, which has an extensive standard library.

Multithreaded Application Support

Unsupported.

Supported Languages

SPARK 2104 supports a subset of Ada 2012.

See [63, p.18]

"The following Ada 2012 features are not currently supported by Spark:

Aliasing of names; no object may be referenced by multiple names

Pointers (access types) and dynamic memory allocation

Goto statements

Expressions or functions with side effects

Exception handlers

Controlled types; types that provide fine control of object creation, assignment, and destruction

Tasking/multithreading (will be included in future releases)

2.5 Spec#

2.5.1 Home Page

https://www.microsoft.com/en-us/research/project/spec/

2.5.2 Features

Spec# consists of the Spec# programming language, the Spec# compiler, and the Spec# static program verifier. The programming language is an extension of C#, adding non-null types, checked exceptions, preconditions, postconditions, and object invariants [66]. It uses Boogie for verification.

2.5.3 Soundness

Spec# claims to be sound[50].

2.5.4 Supported Platforms

Windows host platform, .NET target platform.

2.5.5 License Information

Internet searches failed to find Spec# license information.

2.5.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of Spec# being used in commercial software development.

2.5.7 Existing Libraries

Internet searches failed to find Spec# libraries. However, Spec# offers interoperability with the .NET platform, which has an extensive standard library, although the soundness of this library is not guaranteed.

2.5.8 Multithreaded Application Support

Yes[66].

2.5.9 Supported Languages

Spec#.

2.6 VCC

2.6.1 Home Page

https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/

2.6.2 Features

VCC verifies preconditions and postconditions written in the form of special comments in the source code. It detects data races in multithreaded applications and illegal memory access at compile time. It does not guarantee the absence of undefined behaviour.

2.6.3 Soundness

Yes

2.6.4 Supported Platforms

Windows

2.6.5 License Information

MIT [68]

2.6.6 Evidence of successful use in commercial software development

VCC has been used successfully in at least one major commercial project, the verification of the Microsoft Hypervisor, the virtualization kernel of Hyper-V [53].

2.6.7 Existing Libraries

Internet searches failed to find libraries verified with VCC. However, applications written with VCC can link against unverified libraries, effectively giving access to extensive library support.

2.6.8 Multithreaded Application Support

Yes

2.6.9 Supported Languages

C

2.7 Verifast

2.7.1 Home Page

https://github.com/verifast/verifast

2.7.2 Features

Verifast verifies preconditions, and postconditions in the form of special comments in the source code. It detects race conditions in multithreaded applications. It does not guarantee the absence of undefined behaviour, or verify the absence of stack overflows.

2.7.3 Soundness

No, see https://github.com/verifast/verifast/blob/master/soundness.md

2.7.4 Supported Platforms

Windows, Linux, OSX.

2.7.5 License Information

MIT [61].

2.7.6 Evidence of successful use in commercial software development

There is evidence of some use in industrial applications[77].

2.7.7 Existing Libraries

Internet searches failed to find libraries verified by / written with VeriFast annotations. However, applications written with VeriFast can link against unverified libraries, effectively gaining access to extensive library support.

2.7.8 Multithreaded Application Support

Yes.

2.7.9 Supported Languages

C, Java

2.8 Viper

2.8.1 Home Page

https://www.pm.inf.ethz.ch/research/viper.html

2.8.2 Features

Viper consists of the Viper intermediate verification language, automatic verifiers, and example front end tools [57]. It is more of a tool for creating other verification tools than a tool for verifying software. In practice, a developer would write code in Python and use the 'Nagini' front end (based on Viper) to verify the code [55]. A corresponding front end for Rust exists, 'Prusti' [49].

2.8.3 Soundness

Yes.

2.8.4 Supported Platforms

Windows, MacOS, Linux[60].

2.8.5 License Information

https://bitbucket.org/viperproject/carbon/src/default/LICENSE.txt Mozilla Public License Version 2.0[56]

2.8.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of Viper being used in commercial software development.

2.8.7 Existing Libraries

Internet searches failed to find libraries verified by Viper front ends. However, applications written with these front ends can link against unverified libraries, effectively gaining access to extensive library support.

2.8.8 Multithreaded Application Support

Yes.

2.8.9 Supported Languages

Python, Rust, Java, OpenCL, Chalice.

2.9 Whiley

2.9.1 Home Page

http://whiley.org/

2.9.2 Features

Whiley consists of the Whiley language, the Whiley Build System, the Whiley Compiler, the Whiley Intermediate Language, the Whiley-2-Java Compiler, the Whiley-2-C Compiler, and the Whiley Constraint Solver[76].

Whiley uses a variant of first-order logic called the Whiley Assertion Language for verification.

In typical use, a developer will write source code in Whiley, build with the Whiley Build system, and execute the resulting Java class file on the JVM.

2.9.3 Soundness

Internet searches did not find evidence to say that Whiley is unsound.

2.9.4 Supported Platforms

JVM.

2.9.5 License Information

BSD.

2.9.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of Whiley being used in commercial software development.

2.9.7 Existing Libraries

Internet searches failed to find libraries verified by Whiley. However, applications written with Whiley can link against unverified Java functions, effectively gaining access to extensive library support.

2.9.8 Multithreaded Application Support

No.

2.9.9 Supported Languages

Whiley. The Whiley-2-Java Compiler (WyJC) can convert verified Whiley programs into JVM class files. Whiley can import Java functions, and export Whiley functions for use in Java programs.

2.10 Why3

2.10.1 Home Page

http://why3.lri.fr/

2.10.2 Features

The Why3 deductive program verification platform includes the WhyML language, a standard library of logical theories, and basic programming data structures[83].

In typical use, a developer will write software in WhyML, and get correct-by-construction OCaml programs through an automated extraction mechanism. It verifies preconditions and postconditions.

WhyML is also used by numerous popular verification tools (FramaC, SPARK2014, Krakatoa) as an intermediate language for verification of C, Java and Ada programs.

2.10.3 Soundness

Yes.

2.10.4 Supported Platforms

Windows, Linux, OSX. Why3 is distributed as a Debian package and as an OPAM package.

2.10.5 License Information

GNU LGPL 2.1.

2.10.6 Evidence of successful use in commercial software development

Internet searches failed to find evidence of WhyML being used in commercial software development. However, there are countless cases of commercial software development using WhyML as an intermediate language, as it is used by FramaC, SPARK, and others.

2.10.7 Existing Libraries

Why3 comes with a standard library of logical theories and basic programming data structures.

2.10.8 Multithreaded Application Support

No.

2.10.9 Supported Languages

WhyML.

2.11 Conclusion of Review of Background and Associated Work

Based on the properties of the verification tools available, it has been decided to use SPARK 2014 for the implementation and verification of the OCPP server. It has wide use in industry, which gives me confidence that it is a practical choice. It verifies all of the properties that are important to me, and has good library support.

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.

2.12 Summary of Verification Tool Features

Features								
Tool	No memory leaks	Never accesses uninitialized memory	Never crashes / exits unexpectedly	Bounded RAM	Never hang	Prove correct	No undefined	No data leak
Dafny	2.12	2.12	A.1		2.12		2.12	A.1
JML	?	?	?	?	?	?	?	?
KeY	?	?	?	?	?	?	?	?
SPARK	?	?	?	?	?	?	?	?
Spec #	?	?	?	?	?	?	?	?
Verifast	?	?	?	?	?	?	?	?
VCC	?	?	?	?	?	?	?	?
Viper	?	?	?	?	?	?	?	?
Whiley	?	?	?	?	?	?	?	?
Why3	?	?	?	?	?	?	?	?

Internet searches failed to find evidence that this is supported.

Chapter 3

OCPP Server Implementation

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 2ε `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.

3.1 OCPP Overview

An OCPP server acts as a websocket server. There may be many individual charging stations, which act as websocket clients. The clients establish a websocket connection with the server, and once this is complete, JSON formatted packets are exchanged between the client and the server.

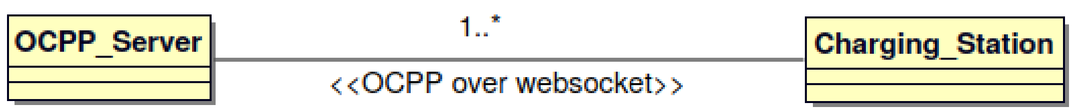


Figure 3.1: OCPP overview.

3.2 Minimal OCPP implementation

The OCPP protocol supports many use cases which are not required for a basic implementation. The following table [70] shows a minimal subset of OCPP messages required to support basic functionality.

Table 3.1: Use cases for a basic implementation

Functionality	Use Case	Messages	Complete
Booting a charge station	B01 - B04	BootNotification	
Configuring a charge station	B05-B07	SetVariables, GetVariables, GetReportBase	
Resetting a charge station	B11-B12	Reset	
Authorization Options	One of C01, C02, C04	Authorize	
Transaction Mechanism	E01 (one of S1-S6), E02-E03, E05, E06 (one of S1-S6), E07- E08, One of E09-E10, E11-E13	TransactionEvent	
Availability	G01, G03-G04	ChangeAvailability, StatusNotification	
Monitoring Events	G05, N07	NotifyEvent	
Meter Values	J02	TransactionEvent	
Data Transfer	P01-P02	DataTransfer	

3.2.1 Boot Notification

Whenever a charger is powered on in the field, it establishes a websocket connection with the server, and sends a 'BootNotificationRequest' packet to announce itself. A high level view of a typical boot notification sequence is given in Figure 3.2.1.

The ‘BootNotificationRequest’ packet contains information describing the charger. If the server recognises the charger, it returns a ‘BootNotificationResponse’ message with a status of ‘Accepted’. The contents of these packets is described in figure 3.2.1.

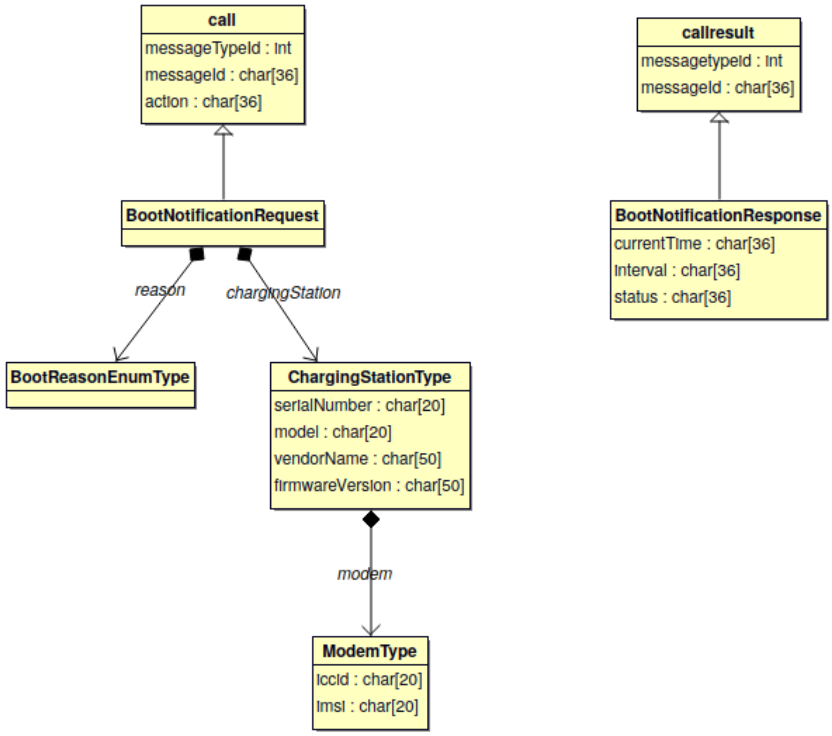


Figure 3.2: ‘Boot Notification’ class diagram.

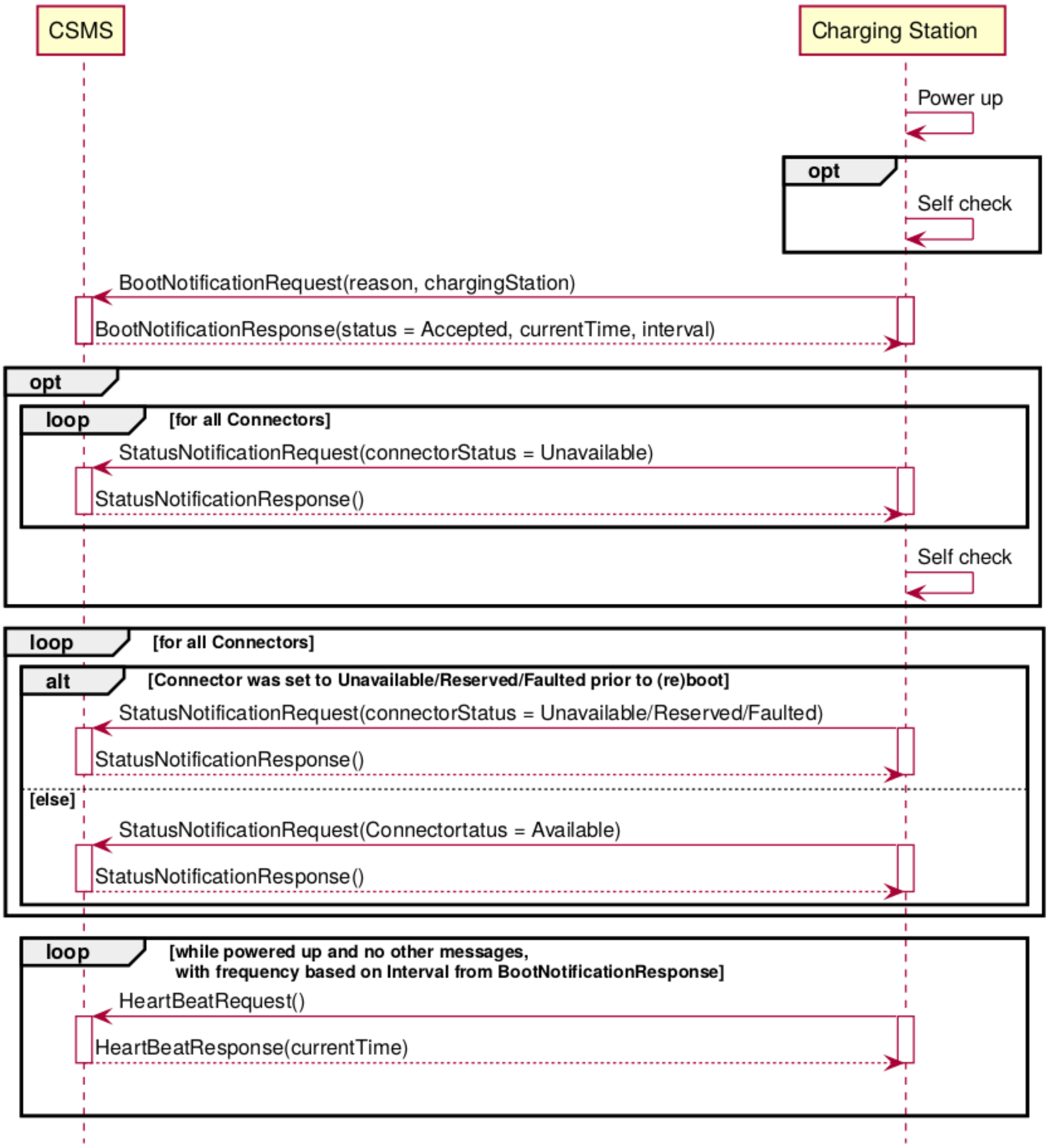


Figure 3.3: Cold Booting a Charging Station [73].

Implementation

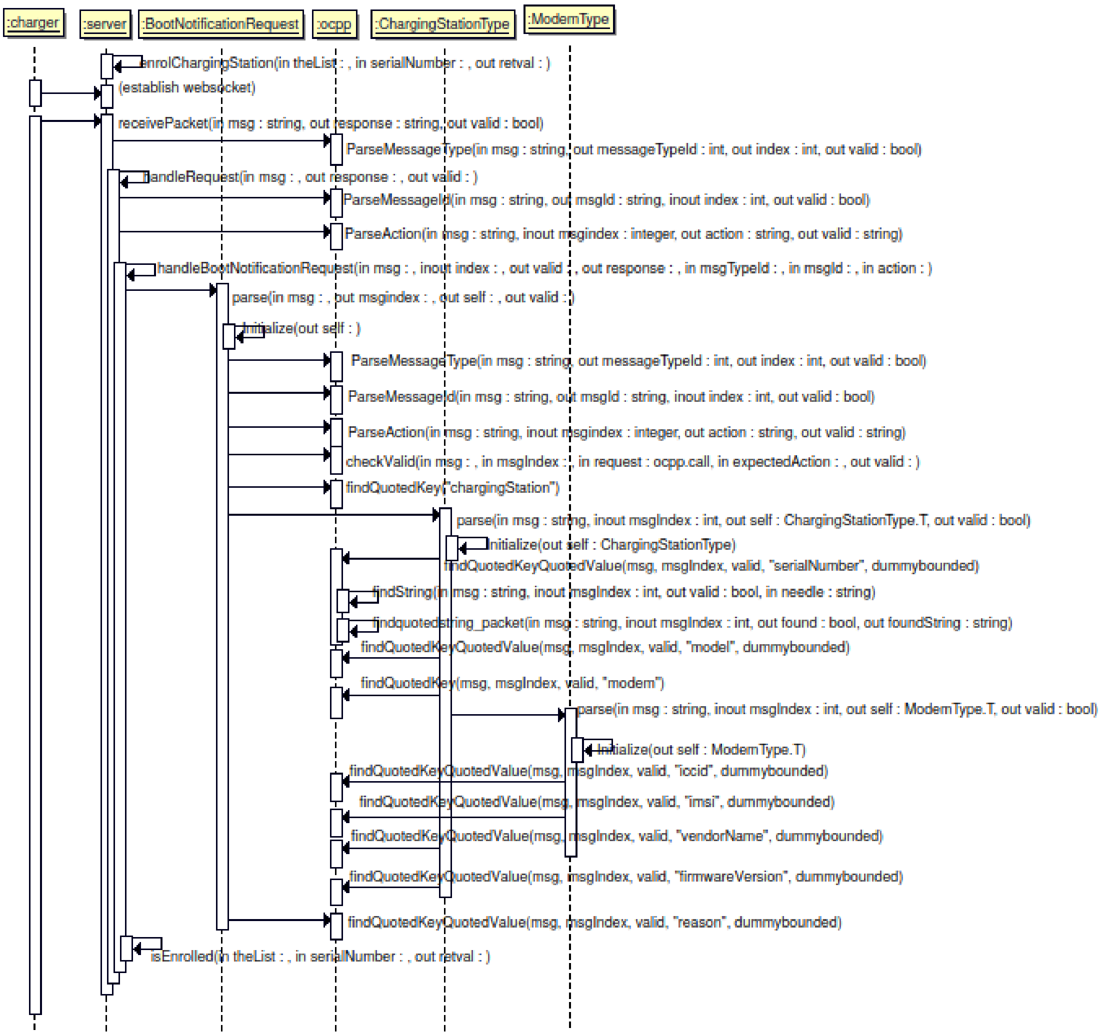


Figure 3.4: Cold Booting a Charging Station [73].

Chapter 4

Results and discussion . . .

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

In any case, you will probably need to include tabulated results. Table **??** 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.

Chapter 5

Conclusions

5.1 Summary and conclusions

5.2 Possible future work

5.2.1 Websocket Implmentation

Appendix A

Dummy appendix

A.1 Dafny

```
https://github.com/dafny-lang/dafny/issues/532
Simulated type set crashes at run-time \#532
An attempt to do a dynamic type test causes a crash when the compiled program is

Repro: Here is the output on the program below:

\$ dafny /compile:3 test.dfy
Dafny 2.3.0.10506
test.dfy(17,19): Warning: /!\ No terms found to trigger on.

Dafny program verifier finished with 2 verified, 0 errors
Running...

t.x=5  The given Tr is a C, and c.y=6
t.x=100  Error: Execution resulted in exception: Exception has been thrown by the
System.InvalidCastException: Specified cast is not valid.
%  at _module.__default+<M>c__AnonStorey0.<>m__0 () [0x00023] in <73b9bd36ee6e47f
%  at _module.__default.M (_module.Tr t) [0x0003b] in <73b9bd36ee6e47fba3617ec618
%  at _module.__default.Main () [0x00058] in <73b9bd36ee6e47fba3617ec618048be5>:0
%  at (wrapper managed-to-native) System.Reflection.MonoMethod.InternalInvoke(Sys
  at System.Reflection.MonoMethod.Invoke (System.Object obj, System.Reflection.Bi
```

And here is the program:

```
trait Tr {
  var x: int
}

class C extends Tr {
  var y: int
}

class D extends Tr {
  var z: int
}

method M(t: Tr)
  modifies t
{
  print "t.x=", t.x, "  ";
  var s: set<C> := set c: C | c == t;  // this line crashes for the call M(d)
  if s == {} {
%    print "The given Tr is not a C\n";
  } else {
    var c :| c in s;
%    print "The given Tr is a C, and c.y=", c.y, "\n";
    c.y := c.y + 10;
  }
}

method Main() {
  var c := new C;
  var d := new D;
  c.x, c.y := 5, 6;
  d.x, d.z := 100, 102;

  M(c);
  M(d);
  M(c);
}
```

```
https://gitter.im/dafny-lang/community?at=5d90c402086a72719e848f24
Bryan Parno
@parno
Mar 14 05:41
We use reference counting via shared_ptr Which means it is possible to create mem
```

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 [**?**, 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

See https://github.com/DanielMcInnes/thesis.git .

Bibliography

[1] "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.

[2] "Ada on Board: GNAT Pro Helps ExoMars Get to the Red Planet". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/ada-on-board-gnat-pro-helps-exomars-get-to-the-red-planet

[3] "Ada on Board: Thales Using AdaCore's GNAT Pro for Critical Avionics Software". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/ada-on-board-thales-using-adacores-gnat-pro-for-critical-avionics-software

[4] "AdaCore and Altran Toolsets Help Launch CubeSat into Orbit". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/cubesat

[5] "AdaCore Development Environment Selected for New Spanish Satellite Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/spanish-satellite-project

[6] "AdaCore Enhances Security-Critical Firmware with NVIDIA". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/adacore-enhances-security-critical-firmware-with-nvidia

[7] "AdaCore Helps AAI Upgrade the T25 SECT Electronic Combat Trainer". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/adacore-helps-aai-upgrade-the-t25-sect-electronic-combat-trainer

[8] "AdaCore's CodePeer Selected for Digital Terrain System Requiring DO-178B Certification". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/codepeer-do178b-certification

[9] "Astrium in the UK Selects GNAT Pro for Environmental Satellite System". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/customers/astute-class-submarine-periscope

[10] "Astrium Selects AdaCore's GNAT Pro and PolyORB for International Space Station". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/astrium-polyorb

[11] "AVIO Selects AdaCore's GNAT Pro Assurance Toolsuite for European Space Agency Program". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/avio-selects-adacores-gnat-pro-assurance-toolsuite-for-european-space-agency-program

[12] "Barco selects GNAT Pro for Advanced Avionics Applications". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/barco

[13] "Case Study BAE Systems Eurofighter Typhoon". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/customers/eurofighter-typhoon

[14] "Case Study EADS CASA In-flight Refuelling Boom System". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/uploads/customers/CaseStudy_Boom.pdf

[15] "Case Study MDA - Canadian Space Arm". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/uploads/customers/CaseStudy_SpaceArm.pdf

[16] "Case Study Pilot Ejection Seat". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/customers/pilot-ejection-seat

[17] "Deep Blue Capital Selects AdaCore Products for Financial System Development". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/deep-blue-capital-financial-system-development

[18] "DENSO Using SPARK Technology for Automotive Research Project". AdaCore. Accessed: Aug 17, 2019.

[19] "Digicomp Shows Continuing Success with Ada and GNAT Pro". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/digicomp-success

[20] "EADS CASA Selects AdaCore Toolset for nEUROn Unmanned Aircraft". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/neuron-unmanned-aircraft

[21] "Embraer Selects Ada and AdaCore's GNAT Pro for AMX Upgrade". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/embraer-amx-upgrade

[22] "Eurocopter Selects GNAT Pro for Military Helicopter ARINC 653 Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/arinc-653-project

[23] "French Agency DGA Selects AdaCore's GNAT Pro with SQUORE Technology". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/dga-gnat-pro-squore-technology

[24] "GNAT Pro Chosen for UK's Next Generation ATC System". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/adacore-gnat-pro-chosen-for-uk-next-generation

[25] "GNAT Pro Safety-Critical used by Terma A/S for Space Monitor Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/terma

[26] "Hamilton Sundstrand Selects GNAT Pro For Boeing 787 Air Conditioning Control Unit". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/hamilton-sundstrand-boeing-air-conditioning

[27] "Lockheed Martin Selects GNAT Pro for C-130J Software". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/a350

[28] "MDA Selects AdaCore's GNAT Pro Assurance Development Platform for International Space Station Software". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: www.adacore.com/press/mda-gnatpro-space-station

[29] "MHI Aerospace Systems Corp. Selects AdaCore's QGen for Model-Based Development". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/mhi-aerospace-systems-qgen

[30] “Protecting Navy Vessels with Ship Anti-Air Warfare Capability”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.raytheon.com/capabilities/products/ssds

[31] “Revolutionary Artificial Heart”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/scandinavian-real-heart-selects-adacore-embedded-software-development-platform-for-revolutiona

[32] “Rockwell Collins Develops SecureOne™ with SPARK Pro and GNAT Pro High-Security”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/secureone

[33] “Rockwell Collins Selects GNAT Pro for Advanced Avionics Display System”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/rockwell-avionics-display

[34] “Saab Electronic Defence Systems Adopts CodePeer”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/saab-electronic-defence-systems-adopts-codepeer

[35] “Siemens Switzerland Selects AdaCore Toolset for Railway Project siemens-railway”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/siemens-railway

[36] “Singo Solution”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.contactcenterworld.com/view/contact-center-case-study/singo-solution.aspx

[37] “SmartSide Adopts Ada and GNAT Pro for Smart Devices Platform”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/smartside-adopts-ada-gnat-pro

[38] “SmartWard Pty Ltd Selects AdaCore Tools for Hospital Information System Development”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/smartward-hospital-information-system

[39] “SPARK Going to the Moon”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/spark-going-to-the-moon

[40] “SPARK Pro Adopted by secunet”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/spark-pro-secunet

[41] “ten years of using spark to build cubesat nano satellites with students”. AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://blog.adacore.com/ten-years-of-using-spark-to-build-cubesat-nano-satellites-with-students

[42] "Thales Aerospace Division Selects GNAT Pro for Airbus A350 XWB (Xtra Wide-Body)". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/astrium

[43] "Thales Selects AdaCore Toolset for Argos Satellite Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/thales-argos-satellite

[44] "TOYOTA ITC Japan Selects SPARK Pro Language and Toolset for High-Reliability Research Project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/toyota-itc-japan-selects-spark-pro-language-and-toolset-for-high-reliabilit

[45] "University of Colorado's Laboratory for Atmospheric and Space Physics adopts Ada and GNAT Pro for NASA project". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/lasp-selects-gnat-pro-for-clarreo

[46] "Wind River Powers New Generation of Periscope on Royal Navy's Next-Generation Astute Class Submarines". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/uploads/customers/adacore_casestudy_uret.pdf

[47] "Wind River Teams with AdaCore on Safety-Critical ARINC 653 for use in Boeing 7E7". AdaCore. Accessed: Aug 17, 2019. [Online]. Available: https://www.adacore.com/press/arinc-653

[48] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, and M. Ulbrich, Eds., *Deductive Software Verification - The KeY Book: From Theory to Practice*, ser. Lecture Notes in Computer Science. Springer, 2016, vol. 10001. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-49812-6

> 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.

[49] V. Astrauskas, P. Müller, F. Poli, and A. J. Summers, "Leveraging Rust types for modular specification and verification," ETH Zurich, Tech. Rep., 2019.

[50] 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 Papers, 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] M. Eilers and P. Müller, "Nagini: A static verifier for python," in *CAV*, 2018.

[56] "Carbon License". ETH Zurich. Accessed: Aug 18, 2019. [Online]. Available: https://bitbucket.org/viperproject/carbon/src/default/LICENSE.txt

[57] "Viper". ETH Zurich. Accessed: Aug 18, 2019. [Online]. Available: https://www.pm.inf.ethz.ch/research/viper.html

[58] 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.

[59] 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.

[60] U. Juhasz, I. T. Kassios, P. Müller, 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.

[61] "VeriFast License". Katholieke Universiteit Leuven. Accessed: Aug 18, 2019. [Online]. Available: https://github.com/verifast/verifast/blob/master/LICENSE.md

[62] 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.

[63] 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.

[64] "Dafny: A Language and Program Verifier for Functional Correctness". Microsoft. Accessed: Aug 17, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/

[65] "Dafny FAQ". Microsoft. Accessed: Aug 17, 2019. [Online]. Available: https://github.com/dafny-lang/dafny/wiki/FAQ

[66] "Spec#". Microsoft. Accessed: Aug 18, 2019. [Online]. Available: https://www.microsoft.com/en-us/research/project/spec/

[67] "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.

[68] "VCC License". Microsoft. Accessed: Aug 18, 2019. [Online]. Available: https://github.com/microsoft/vcc/blob/master/LICENSE

[69] "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.

[70] "OCA OCPP 2.0 Part 0 - Introduction". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: https://www.openchargealliance.org/downloads/

[71] "OCA OCPP 2.0 Part 1 - Architecture & Topology". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: https://www.openchargealliance.org/downloads/

[72] "OCA OCPP 2.0 Part 2 - Appendices". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: https://www.openchargealliance.org/downloads/

[73] "OCA OCPP 2.0 Part 2 - Specification". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: https://www.openchargealliance.org/downloads/

[74] "OCA OCPP 2.0 Part 4 - OCPP-J Specification". OpenChargeAlliance.org. Accessed: June 22, 2019. [Online]. Available: https://www.openchargealliance.org/downloads/

[75] "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.

[76] 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.

[77] P. Philippaerts, J. T. Mühlberg, W. Penninckx, J. Smans, B. Jacobs, and F. Piessens, "Software verification with verifast: Industrial case studies," *Science of Computer Programming*, vol. 82, no. C, pp. 77–97, 2014.

[78] 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.

[79] 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.

[80] "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.

[81] F. Vogels, B. Jacobs, and F. Piessens, "Featherweight Verifast," *Logical Methods in Computer Science*, vol. 11, no. 3:19, p. 1–57, 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.

[82] "Whiley A Programming Language with Extended Static Checking". Whiley. Accessed: Aug 18, 2019. [Online]. Available: http://whiley.org/

[83] "Why3 Where Programs Meet Provers". Why3. Accessed: Aug 18, 2019. [Online]. Available: http://why3.lri.fr/