

Qualification of Formal Methods Tools

Edited by

Darren Cofer¹, Gerwin Klein², Konrad Slind³, and Virginie Wiels⁴

1 Rockwell Collins - Minneapolis, US, darren.cofer@rockwellcollins.com

2 NICTA - Sydney, AU, gerwin.klein@nicta.com.au

3 Rockwell Collins - Minneapolis, US, konrad.slind@rockwellcollins.com

4 ONERA - Toulouse, FR, virginie.wiels@onera.fr

Abstract

Formal methods tools have been shown to be effective at finding defects in and verifying the correctness of safety-critical systems, many of which require some form of certification. However, there are still many issues that must be addressed before formal verification tools can be used as part of the certification of safety-critical systems. For example, most developers of avionics systems are unfamiliar with which formal methods tools are most appropriate for different problem domains. Different levels of expertise are necessary to use these tools effectively and correctly. In most certification processes, a tool used to meet process objectives must be *qualified*. The qualification of formal verification tools will likely pose unique challenges.

Seminar 26-29 April, 2015 – <http://www.dagstuhl.de/15182>

1998 ACM Subject Classification D.2.4 Software/program verification, F.3.1 Specifying and Verifying and Reasoning about Programs, G.4 Mathematical Software

Keywords and phrases Dependable systems, Certification, Qualification, Formal methods, Verification tools

Digital Object Identifier 10.4230/DagRep.1.1.1

1 Executive Summary

Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels

License  Creative Commons BY-NC-ND 3.0 Unported license
© Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels

1.1 Motivation and objectives

Dagstuhl Seminar 13051, *Software Certification: Methods and Tools*, convened experts from a variety of software-intensive domains (automotive, aircraft, medical, nuclear, and rail) to discuss software certification challenges, best practices, and the latest advances in certification technologies. One of the key challenges identified in that seminar was tool qualification. Tool qualification is the process by which certification credit may be claimed for the use of a software tool. The purpose of tool qualification is to provide sufficient confidence in the tool functionality so that its output may be trusted. Tool qualification is, therefore, a significant aspect of any certification effort. Seminar participants identified a number of needs in the area of formal methods tool qualification. Dagstuhl Seminar 15182 *Qualification of Formal Methods Tools*, was organized to address these needs.

Software tools are used in development processes to automate life cycle activities that are complex and error-prone if performed by humans. The use of such tools should, in principle, be encouraged from a certification perspective to provide confidence in the correctness of the



Except where otherwise noted, content of this report is licensed under a Creative Commons BY-NC-ND 3.0 Unported license
Qualification of Formal Methods Tools, *Dagstuhl Reports*, Vol. 1, Issue 1, pp. 1–19
Editors: Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels



Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

software product. Therefore, we should avoid unnecessary barriers to tool qualification which may inadvertently reduce the use of tools that would otherwise enhance software quality and confidence.

Most software tools are not used in isolation, but are used as part of a complex tool chain requiring significant integration effort. In general, these tools have been produced by different organizations. We need to develop better and more reliable methods for integrating tools from different vendors (including university tools, open source tools, and commercial tools).

A given software tool may be used in different application domains having very different requirements for both certification and tool qualification. Furthermore, the methods and standards for tool development varies across domains. Consistent qualification requirements across different domains would simplify the process.

Despite the additional guidance provided for the avionics domain in recently published standards (DO-178C, DO-330, and DO-333), there are still many questions to be addressed. For one thing, most practicing engineers are unaware of how to apply different categories of formal verification tools. Even within a particular category, there are a wide variety of tools, often based on fundamentally different approaches, each with its own strengths and weaknesses.

If formal verification is used to satisfy DO-178C objectives, DO-333 requires the applicant to provide evidence that the underlying method is sound, i.e., that it will never assert something is true when it is actually false, allowing application software errors to be missed that should have been detected. Providing an argument for the soundness of a formal verification method is highly dependent on the underlying algorithm on which the method is based. A method may be perfectly sound when used one way on a particular type of problem and inherently unsound when used in a different way or on a different type of problem. While these issues may be well understood in the research community, they are not typically collected in one place where a practitioner can easily find them. It is also not realistic to expect avionics developers to be able to construct an argument for the soundness of a formal method without help from experts in the field.

At the same time, it is also important to not make the cost of qualification of formal methods tools so great as to discourage their use. While it is tempting to hold formal verification tools to a higher standard than other software tools, making their qualification unnecessarily expensive could do more harm than good.

The objectives of this Dagstuhl Seminar were to

- investigate the sorts of assurances that are necessary and appropriate to justify the application of formal methods tools throughout all phases of design in real safety-critical settings,
- discuss practical examples of how to qualify different types of formal verification tools, and
- explore promising new approaches for the qualification of formal methods tools for the avionics domain, as well as in other domains.

1.2 Accomplishments

Qualification is not a widely understood concept outside of those industries requiring certification for high-assurance, and different terminology is used in different domains. The seminar was first a way of sharing knowledge from certification experts so that formal methods

researchers could better understand the challenges and barriers to the use of formal methods tools.

The seminar also included presentations from researchers who have developed initial approaches to address qualification requirements for different classes of formal methods tools. We were especially interested in sharing case studies that are beginning to address tool qualification challenges. These case studies include tools based on different formal methods (model checking, theorem proving, abstract interpretation).

As a practical matter, we focussed much of our discussion on the aerospace domain since there are published standards addressing both formal methods and tool qualification for avionics software. The seminar also included researchers from other domains (nuclear, railway) so we could better understand the challenges and tool qualification approaches that are being discussed in those domains.

We managed to bridge a lot of the language between the certification domains, mostly railway, avionics, and nuclear, and bits of automotive, and related the qualification requirements to each other. Some of the otherwise maybe less stringent schemes (e.g. automotive) can end up having stronger qualification requirements, because formal methods are not specifically addressed in them. There is some hope that DO-333 might influence those domains, or be picked up by them in the future, to increase the use of FM tools which would increase the quality of systems.

For the academic tool provider side, we worked out and got the message across that tool qualification can be a lot easier and simpler than what we might strive for academically, and discussed specific tools in some detail, clarifying what would be necessary for a concrete qualification. Finally, we also investigated tool architectures that make tools easier to qualify (verification vs code generation).

1.3 Participants

- Andronick June, NICTA, Australia
- Arthan Rob, Lemma 1, UK
- Blanchette Jasmin Christian, TU Munich, Germany
- Blazy Sandrine, INRIA, France
- Bordin Matteo, AdaCore, France
- Cofer Darren, Rockwell Collins, USA
- Cok David, Grammatech, USA
- Delmas Remi ONERA France
- Dierkes Michael, Rockwell Collins, France
- Engstrom Eric, SIFT, USA
- Klein Gerwin, NICTA, Australia
- Kumar Ramana, Cambridge University, UK
- Lawford Mark, McMaster University, Canada
- Leroy Xavier, INRIA, France
- Leue Stefan, University of Konstanz, Germany
- Mebsout Alain, University of Iowa, USA
- Merz Stephan, INRIA Nancy, France
- Munoz Cesar, NASA, USA
- Myreen Magnus, Cambridge University, UK
- Owens Scott, University of Kent, UK

- Pantel Marc, IRIT, France
- Pister Markus, AbsInt, Germany
- Schuetz Werner, Thales, Austria
- Slind Konrad, Rockwell Collins, USA
- Tudor Nick, D-RisQ, UK
- Wagner Lucas, Rockwell Collins, USA
- Whalen Michael, University of Minnesota, USA
- Wiels Virginie, ONERA, France

2 Table of Contents

Executive Summary

<i>Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels</i>	1
Motivation and objectives	
<i>Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels</i>	1
Accomplishments	
<i>Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels</i>	2
Participants	
<i>Darren Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels</i>	3

Overview of Talks

Please check my 500K LOC of Isabelle	
<i>June Andronick</i>	7
Compiling avionics software with the CompCert formally verified compiler	
<i>Sandrine Blazy</i>	9
Qualification of Formal Methods Tools and Tool Qualification with Formal Methods	
<i>Matteo Bordin</i>	9
Are You Qualified for This Position? An Introduction to Tool Qualification	
<i>Darren Cofer</i>	10
Sharing experience on SAT-based formal verification toolchain qualification in the railway domain	
<i>Rémi Delmas</i>	10
Qualification of PVS for Systematic Design Verification of a Nuclear Shutdown System	
<i>Mark Lawford</i>	11
How much is CompCert's proof worth, qualification-wise?	
<i>Xavier Leroy</i>	12
Certificates for the Qualification of the Model Checker Kind 2	
<i>Alain Mebsout</i>	14
Towards Certification of Network Calculus	
<i>Stephan Merz</i>	14
Tool Qualification Strategy for Abstract Interpretation-based Static Analysis Tools	
<i>Markus Pister</i>	14
Tool Qualification in the Railway Domain	
<i>Werner Schuetz</i>	15
FM Tool Trust Propositions	
<i>Konrad Slind</i>	16
DO-330 Tool Qualification: An experience report	
<i>Lucas Wagner</i>	16

Discussion Groups

Why qualify a formal methods tool?	17
--	----


6 15182 – Qualification of Formal Methods Tools

How to qualify a formal methods tool?	17
Compiler qualification strategies	18
Comparison of qualification in different domains	18

3 Overview of Talks

3.1 Please check my 500K LOC of Isabelle

June Andronick (UNSW - Sydney, AU)

License  Creative Commons BY-NC-ND 3.0 Unported license
© June Andronick

The seL4 microkernel has been formally proved correct [2], from binary code, up to high level requirements, using the Isabelle theorem prover [5]. In this talk we first gave an overview of seL4 development and proof guarantees and assumptions. We then explored what would be needed for a (hypothetical) certification of seL4 according to DO-178 (the software certification standard for airborne systems on commercial aircraft [6]), including a potential qualification of Isabelle according to DO-330 (tool qualification guidelines [7]).

The seL4 microkernel is a small operating system kernel, of roughly 10,000 lines of C code, designed to be a high-performance, secure, safe, and reliable foundation for a wide variety of application domains. It provides isolation and controlled communication to applications running on top of it, allowing trusted applications to run alongside untrusted, legacy code such as a whole Linux instance.

seL4 is the world's most verified kernel [2], with a full functional correctness proof, showing that the binary code is a correct implementation of the high-level functional specification, plus security proofs, showing that seL4 enforces integrity and confidentiality. All the proofs have been conducted in the Isabelle/HOL theorem prover, apart from the binary-to-C correctness proof, which uses some SMT solvers and HOL4 models and proofs. The combined Isabelle proofs amount to about 500,000 lines of Isabelle models and proof scripts.

For this Dagstuhl seminar of tool qualification, we have put ourselves in the situation of wanting to certify seL4 for use in an avionics context, and therefore needing to qualify the tools used in its formal verification, here mainly Isabelle, according to DO-330. Following the discussions and presentations from the seminar, we investigated the following question:

what would be needed to qualify Isabelle, for the objective of using the proof of functional correctness of seL4 to justify that the code is complete and correct with respect to its high-level specification?

From our understanding of the qualification process, we propose to answer the following questions.

1. Justify that the *method* (Interactive Theorem Proving) is *suitable*:

Since the property we are showing is functional correctness, it requires a high-level of expressiveness to precisely model the code and specification; such high level of expressiveness implies a loss of decidability, and therefore requires user's input to perform the proof. Interactive theorem proving fits precisely with those requirements. To justify this to a certifier, we could refer to peer-reviewed papers or point to examples of projects using interactive theorem provers to prove functional correctness.

2. Justify that the *method* (Isabelle-style deduction) is *sound*:

Isabelle's logic is based on a very small kernel that needs to be trusted: a dozen axioms, that have been manually validated. All extensions are derived from first principles

and checked by this kernel. The only ways of adding axioms is through (conservative) definitions and through explicit axioms and tracked oracles (e.g. `sorried` lemmas). To justify this to a certifier, we could again refer to peer-reviewed papers, the *HOL-report* [1], or the formally verified HOL-light [3] and CakeML implementations [4].

3. Justify that the *tool* (Isabelle) correctly implements the method:

This would require us to show that only the standard distribution theory HOL is used, that no *axiom* commands are used after the theory HOL, that no “sorry” and “`cheat_tac`” commands are used, and other technical corner-cases that should be documented. When these conditions are met, only true theorems in HOL can be derived. Evidences for this question would ideally be a small verified proof checker for Isabelle (using e.g. cakeML and providing efficient proof terms).

4. Justify the *correct use* of the tool (Isabelle):

This would consist in checking that the above conditions (no axioms, no sorries, etc) are satisfied in the specific example of the proof under consideration. This is where the title of this talk comes from.

5. Justify that the tool (Isabelle) is helping *meeting the objective*:

This would require showing that the model of C used is a correct representation of C, that the model of the specification is a correct representation of the expected behavior, and that the formalisation of the property (here refinement) is a correct representation of the objective (here that the code is complete and correct with respect to its high-level specification). The seL4 verification includes high-level security proofs, which aim at justifying that the specification satisfies the expected behaviors. Evidence for the C model and refinement statement could be done by review, inspection and testing. As a community, it would also be helpful to provide documentation and training material on how to *read* formal specification, to allow certifiers and non-experts to convince themselves that the statements and properties make sense. Then they only need to trust the experts and peer-reviewed papers that the proof script will indeed provide an evidence that the statement is true, that the property is satisfied.

3.1.0.1 Acknowledgments:

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.


References

- 1 Mike Gordon. HOL: A machine oriented formulation of higher-order logic. Technical report, University of Cambridge Computer Laboratory, 1985.
- 2 Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. Comprehensive formal verification of an OS microkernel. *ACM Transactions on Computer Systems*, 32(1):2:1–2:70, February 2014.
- 3 Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. HOL with definitions: Semantics, soundness, and a verified implementation. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving (ITP)*, pages 308–324. Springer, 2014.
- 4 Ramana Kumar, Magnus Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In Peter Sewell, editor, *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 179–191, San Diego, jan 2014. ACM Press.

- 5 Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
- 6 RTCA. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*.
- 7 RTCA. *DO-330, Software Tool Qualification Considerations*.

3.2 Compiling avionics software with the CompCert formally verified compiler

Sandrine Blazy (IRISA - Rennes, FR)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Sandrine Blazy

Compilers are complicated pieces of software that sometimes contain bugs causing wrong executable code to be silently generated from correct source programs. In turn, this possibility of compiler-introduced bugs diminishes the assurance that can be obtained by applying formal methods to source code.

This talk gives an overview of the CompCert project: an ongoing experiment in developing and formally proving correct a realistic, moderately-optimizing compiler from a large subset of C to PowerPC, ARM and x86 assembly languages. The correctness proof, mechanized using the Coq proof assistant, establishes that the generated assembly code behaves exactly as prescribed by the semantic of the C source, eliminating all possibilities of compiler-introduced bugs and generating unprecedented confidence in this compiler.

3.3 Qualification of Formal Methods Tools and Tool Qualification with Formal Methods


Matteo Bordin (AdaCore - Paris, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Matteo Bordin

This work focuses on the return of experience in the relation between Formal Methods and Tool Qualification. We explored two main application domains: the qualification of formal methods tools and the use of formal methods for tool qualification. In the first case, we present our work in qualifying an abstract interpretation tool (CodePeer) and a formal verification tool (SPARK) in a DO-178 context. In the second case, we focus instead on a lightweight use of formal methods to help the qualification of an automated code generator from Simulink models. This second experience is particularly interesting as it describes how we used Ada 2012 contracts (pre/post-condition) to formally describe in first-order logic the behavior of a code generator. Such specification is not used to statically verify the code generator, but rather as a run-time oracle that checks that the tool executes accordingly to its specifications. Differently from other similar experiences, and quite to our surprise, we realized that the specification in the form of pre/post -conditions significantly differed from the implementation algorithm.

3.4 Are You Qualified for This Position? An Introduction to Tool Qualification

Darren Cofer (Rockwell-Collins - Bloomington, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Darren Cofer


Formal methods tools have been shown to be effective at finding defects in and verifying the correctness of safety-critical systems such as avionics systems. The recent release of DO-178C and the accompanying Formal Methods supplement DO-333 will make it easier for developers of software for commercial aircraft to obtain certification credit for the use of formal methods.

However, there are still many issues that must be addressed before formal verification tools can be injected into the design process for safety-critical systems. For example, most developers of avionics systems are unfamiliar with which formal methods tools are most appropriate for different problem domains. Different levels of expertise are necessary to use these tools effectively and correctly. Evidence must be provided of a formal method's soundness, a concept that is not well understood by most practicing engineers. Finally, DO-178C requires that a tool used to meet its objectives must be qualified in accordance with the tool qualification document DO-330. The qualification of formal verification tools will likely pose unique challenges.

Qualification is not a widely understood concept outside of those industries requiring certification for high-assurance, and different terminology is used in different domains. This talk provided an overview of certification and qualification requirements for the civil aviation domain so that formal methods researchers can better understand the challenges and barriers to the use of formal methods tools. Topics covered included a summary of certification processes and objectives for avionics software, requirements for qualification of tools used in software development and verification, and how formal methods tools fit into the certification environment.

3.5 Sharing experience on SAT-based formal verification toolchain qualification in the railway domain

Rémi Delmas (ONERA - Toulouse, FR)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Rémi Delmas

The goal of the talk is to fuel the reflexion and discussion about formal verification tool qualification in the aerospace domain according to the new DO-333 guidelines, by sharing previous experience on tool qualification in the railway domain under CENELEC SIL-* requirements. The talk describes a formal verification toolchain based on SAT solvers and k-induction used in the railway domain for the verification of safety properties of interlocking and communication-based train control systems. The tool in question has been used to earn certification credits, by replacing tests with formal properties verification, in real world railway control systems. In particular, the talk describes how the tool chain's architecture, development and V&V process was designed in order to meet CENELEC SIL-4 tool qualification requirements, using implementation diversification, semantic equivalence checking, proof-logging/proof-checking. The talk also highlights the various non-technical

issues that surround formal verification tool qualification, which nevertheless must be taken into account to ensure the success of formal verification in industrial applications.

3.6 Qualification of PVS for Systematic Design Verification of a Nuclear Shutdown System

Mark Lawford (McMaster University - Hamilton, CA)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Mark Lawford

The Systematic Design Verification (SDV) process used on the redesign of the Darlington Nuclear Generating Station originated in the difficulties encountered in receiving regulatory approval for Canada's first computer based reactor shutdown system (SDS) [4]. The SDV process for the redesign project made use of tabular expressions for the Software Requirements Specification (SRS) and the Software Design Description (SDD). Completeness and consistency of the tabular expressions and the conformance of the SDD to the SRS were established using the automated theorem prover PVS [3]. The process used to qualify PVS for use in this context is described below and related to the latest version of IEC 61508.

The qualification required the use of manual proof to mitigate against potential undetected errors that might be caused by a failure of PVS, i.e., all of the proofs performed in the PVS theorem prover also had to be done by hand. The standard IEC 61508 (2nd ed) in part 4 provides a classification of tools according to whether they are *software on-line support tools* that can directly influence system safety at run time, or *software off-line support tools* that support a phase of the software development lifecycle and that cannot directly influence the safety-related system during its run time. Software off-line support tools are further broken down into three subclasses:

- T1** : generates no outputs which can directly or indirectly contribute to the executable code (including data) of the safety related system; (e.g. a text editor, a requirements or design support tool with no automatic code generation capabilities, configuration control tools)
- T2** : supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software; (e.g. a test harness generator, test coverage measurement tool, static analysis tool)
- T3** : generates outputs which can directly or indirectly contribute to the executable code of the safety related system (e.g., an optimising compiler where the relationship between the source code program and the generated object code is not obvious, a compiler that incorporates an executable run-time package into the executable code).

According to this classification, PVS as used on the Darlington Redesign Project would be a T2 tool since it is being used to verify a design and a tool failure could fail to reveal an error but not introduce an error into the executable.

In IEC 61508-3 (2nd ed) it states that:

7.4.4.5 An assessment shall be carried out for offline support tools in classes T2 and T3 to determine the level of reliance placed on the tools, and the potential failure mechanisms of the tools that may affect the executable software. Where such failure mechanisms are identified, appropriate mitigation measures shall be taken.

Since a failure mechanism is that PVS has a bug that causes a proof to succeed when it should have failed, we needed a mitigation strategy. The strategy chosen was to redo all

proofs manually. Although this mitigation strategy might appear to defeat much of the benefit of using a formal methods tool, PVS could still be used to quickly check design iterations and the manual checks only needed to be performed on the final work product to mitigate PVS's failure modes. Still, the final manual proofs were tedious and required significant effort.


A proposal is made for a revised Tabular Expression Toolbox that makes use of PVS and an SMT solver to eliminate the need for manual review in order to gain tool qualification. A prototype implementation of the Tabular Expression Toolbox is described in [1].

References

- 1 Eles, C. and Lawford, M. (2011). A tabular expression toolbox for matlab/simulink. In *3rd NASA Formal Methods Symposium*, volume 6617 of *LNCIS*, pages 494–499. Springer-Verlag.
- 2 Pang, L., Wang, C.-W., Lawford, M., and Wassyng, A. (2014). Formalizing and verifying function blocks using tabular expressions and pvs. In *Second International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2013)*, volume 419 of *Communications in Computer and Information Science*, pages 163–178. Springer.
- 3 Wassyng, A. and Lawford, M. (2003). Lessons learned from a successful implementation of formal methods in an industrial project. In Araki, K., Gnesi, S., and Mandrioli, D., editors, *FME 2003: International Symposium of Formal Methods Europe Proceedings*, volume 2805 of *Lecture Notes in Computer Science*, pages 133–153, Pisa, Italy. Springer-Verlag.
- 4 Wassyng, A., Lawford, M. S., and Maibaum, T. S. (2011). Software certification experience in the canadian nuclear industry: lessons for the future. In *Proceedings of the ninth ACM international conference on Embedded software*, EMSOFT '11, pages 219–226, New York, NY, USA. ACM.

3.7 How much is CompCert's proof worth, qualification-wise?

Xavier Leroy (INRIA - Le Chesnay, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Xavier Leroy

Intuitively as well as experimentally (cf. the Csmith compiler testing project), the formal verification of the CompCert C compiler generates much confidence that it is free of miscompilation issues. How can we derive certification credit from this formal verification, in the context of a DO-330 / DO-333 tool qualification? This question is being investigated within the Verasco project (ANR-11-INSE-03; <http://verasco.imag.fr/>).

Consider first the formally-verified part of the CompCert C compiler. This part goes from abstract syntax for the CompCert subset of C to abstract syntax for the assembly language of the target processor. This part contains all the optimizations and almost all code generation algorithms. For this part, we see a plausible mapping between parts of the Coq development and DO-330 concepts:

- The "specifications" part of the Coq development constitutes most of the (high-level) tool requirements. This part comprises the abstract syntax and operational semantics of the CompCert C and CompCert assembly languages, as well as the high-level statement of compiler correctness, namely preservation of semantics during compilation, with preservation of properties as a corollary.

- The "code" part of the Coq development map to the low-level tool requirements. This part comprises all compilation algorithms (written in pure functional, executable style in Coq's specification language) as well as the abstract syntaxes of the intermediate languages

used. It is comparable to the pseudocode or Simulink/Scade models that are used as low-level requirements in other certifications.

- The "proof" part of the Coq development automates the verification activities between the (high-level) tool requirements and the low-level tool requirements. This part contains the proofs of semantic preservation for every compilation pass, the proofs of semantic soundness for every static analysis, as well as the operational semantics for the intermediate languages.

A first difficulty is that the "specifications", "code" and "proof" parts are not clearly separated in CompCert's Coq development, owing to good mathematical style (theorems and their proofs come just after definitions) and also to the use of dependently-typed data structures. It would be useful to develop a "slicing" tool for Coq that extracts the various parts of the development by tracing dependencies.

The source code for the compiler, in DO-330 parlance, corresponds to the OCaml code that is generated from the "code" part of the Coq development by Coq's extraction facility. The executable compiler, then, is obtained by OCaml compilation. Here, we are in familiar territory: automatic code generation followed by compilation. However, suitable confidence arguments must be provided for Coq's extraction and for OCaml's compilation. Several approaches were discussed during the meeting, ranging from dissimilar implementations to Coq-based validation of individual runs of the executable compiler.

At the other end of the DO-330 sequence of refinements, we are left with the tool operational requirements, which have to be written in informal prose, with references to the ISO C 1999 language standard, the ISA reference manuals for the target architecture, and coding standards such as MISRA C. The verification activities here are essentially manual, and include for example relating the CompCert C formal semantics with the informal specifications in ISO C 1999 and MISRA. Such a relation can be built from appropriate tests, since CompCert provides a reference interpreter that provides an executable, testable form of its C formal semantics.


All in all, the formal proof of CompCert does not eliminate the need for manual verifications, but it reduces their scope tremendously: from manual verification of a full optimizing compiler to manual verification of formal semantics for C and assembly languages. For example, changes to the "code" part of the compiler (e.g. adding new optimizations, modifying the intermediate languages, etc) need no new manual verification activities, as long as the "specification" part of the compiler is unchanged.

To finish, we need to consider the parts of the CompCert C compiler that are not formally verified yet: uphill of the verified part, the transformations from C source text to CompCert C abstract syntax (preprocessing, tokenization, parsing, type-checking, pre-simplifications, production of an abstract syntax tree); downhill, the transformation from assembly abstract syntax to ELF executables (assembling and linking). CompCert provides an independent checker that validates a posteriori the assembling and linking phases. Likewise, some of the uphill passes were formally verified recently (parsing and type-checking). Nonetheless, many of the uphill passes lack formal specifications and therefore must be verified by conventional, test-based means.

In conclusions, the qualification of an optimizing compiler to the highest quality levels has never been attempted before, and might very well be too expensive to be worth the effort. A formal compiler verification such as CompCert's has high potential to reduce these costs. However, much work remains to take full advantage of this potential.

3.8 Certificates for the Qualification of the Model Checker Kind 2

Alain Mebsout (University of Iowa - Iowa City, US)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Alain Mebsout

Joint work of Mebsout, Alain; Tinelli, Cesare;

This talk presents a technique for generating proof certificates in the model checker Kind 2 as an alternate path of qualification with respect to DO-178C. This is put in perspective with the qualification that was conducted for the SMT solver Alt-Ergo at Airbus for use in the development of the A350. Alt-Ergo was qualified wrt DO-178B as a backend solver for Caveat to verify C code of the pre-flight inspection. On the other hand, Kind 2 generates proof certificates which allows to shift the trust from the model checker to the proof checker (LFSC). Certificates for the actual model checking algorithm are generated as SMT2 files and verified by an external SMT solver. The translation from Lustre to the internal first-order logic representation is verified in a lightweight way by proving observational equivalence between independent frontends (for the moment JKind and Kind 2). This proof is actually carried by Kind 2 itself and generates in turn SMT2 certificates.

3.9 Towards Certification of Network Calculus

Stephan Merz (INRIA Nancy - Villers-lès-Nancy, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Stephan Merz


Joint work of Boyer, Marc; Fejoz, Loïc; Mabilie, Etienne; Merz, Stephan;

URL http://dx.doi.org/10.1007/978-3-642-39634-2_37

Network Calculus (NC) is an established theory for determining bounds on message delays and for dimensioning buffers in the design of networks for embedded systems. It is supported by academic and industrial tool sets and has been widely used, including for the design and certification of the Airbus A380 AFDX backbone. However, tool sets used for developing certified systems need to be qualified, which requires substantial effort and makes them rigid, even when deficiencies are subsequently detected. Result checking may be a worthwhile complement, since the use of a qualified (and highly trustworthy) checker could replace qualifying the analysis tool itself. In this work, we experimented an encoding of the fundamental theory of NC in the interactive proof assistant Isabelle/HOL and used it to check the results of a prototypical NC analyzer.

3.10 Tool Qualification Strategy for Abstract Interpretation-based Static Analysis Tools

Markus Pister (AbsInt - Saarbrücken, DE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Markus Pister

Joint work of Kästner Daniel, Pister Markus, Gebhard Gernot, Ferdinand Christian

Main reference Kästner Daniel, Pister Markus, Gebhard Gernot, Ferdinand Christian, Reliability of WCET Analysis, Proceedings of the Embedded Real Time Software and Systems Congress ERTSS,

Toulouse, 2014.

In automotive, railway, avionics and healthcare industries more and more functionality is implemented by embedded software. A failure of safety-critical software may cause high costs or even endanger human beings. Also for applications which are not highly safety-critical, a software failure may necessitate expensive updates.

Safety-critical software has to be certified according to the pertinent safety standard to get approved for release. Contemporary safety standards including DO-178C, IEC-61508, ISO-26262, and EN-50128 require the identification of potential functional and non-functional hazards and to demonstrate that the software does not violate the relevant safety goals. If tools are used to satisfy the corresponding verification objectives, an appropriate tool qualification is mandatory to show functional correctness of the tool behavior with respect to the operational context.


To ensure functional program properties, automatic or model-based testing and formal techniques like model checking are becoming more widely used. For non-functional properties identifying a safe end-of-test criterion is a hard problem since failures usually occur in corner cases and full test coverage cannot be achieved.

For some non-functional program properties this problem is solved by abstract interpretation-based static analysis techniques which provide full control and data coverage and yield provably correct results. Like model checking and theorem proving, abstract interpretation belongs to the formal software verification methods. AbsInt provides abstract interpretation-based static analyzers to determine safety-guarantees on the worst-case execution time (aiT) and stack consumption (StackAnalyzer) as well as to prove the absence of runtime errors (Astree) in safety-critical software.

This talk focuses on our tool qualification strategy of the above mentioned verification tools, which are increasingly adopted by industry in their validation activities for safety-critical software. First, we will give an overview of the tools and their role within the analyzed system's certification process. We then outline the required activities for a successful tool qualification of our static analyzers alongside their correspondingly produced data.

3.11 Tool Qualification in the Railway Domain

Werner Schuetz (Thales - Wien, AT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Werner Schuetz

In this presentation we give an overview of the relevant standards applicable to the rail domain. EN50128 is concerned with software, while EN50129 addresses system issues.


This presentation focuses on tool qualification. The 2011 edition of EN50128 is the first to include requirements on "Support Tools and Languages". To this end it defines three tool classes. T3 tools directly or indirectly produce code or data that is used in the safety-related system. T2 tools are verification tools that may fail to detect an error but cannot introduce an error themselves. T1 tools do not contribute directly or indirectly to the executable code or data.

This presentation discusses the requirements on support tools and how they apply to the three tool classes. Comparison with the relevant aerospace standards (DO178C, DO330) is partly given.

In an appendix we briefly analyze which "Formal Methods" are contained in the 2011 edition of EN50128.

3.12 FM Tool Trust Propositions

Konrad Slind (Rockwell-Collins - Bloomington, USA)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Konrad Slind


An interactive theorem proving (ITP) system is a complex piece of software that bundles a great deal of functionality together. Beyond their core theorem proving task, which can employ highly complex algorithms, these systems provide extensibility, rich interfaces for users, interaction with host operating systems, etc. And yet, ITP systems are claimed to provide very high assurance. It is our purpose to take a close look at this state of affairs and explain the justifications for this claim.

We introduce the notion of the *trust proposition* to organize the discussion: it helps the consumer of a theorem prover's output understand what the full assurance story is, by breaking the overall trust proposition down to subcomponents. In particular, we identify the work product of an ITP as a collection of theories, which formalize the artifact under scrutiny, plus properties and proofs. This work product can be trusted, provided the following conditions are met:

1. **Trusted Basis** The support theories are trusted;
2. **Trusted Extension** The newly introduced types, constants, definitions, and axioms are trusted;
3. **Valid Model** The support theories plus newly introduced types, constants, definitions, and axioms accurately model the artifact under scrutiny;
4. **Sound Logic** The proof system is sound;
5. **Correct Implementation** The proof system and extension mechanisms are correctly implemented
6. **Correct Libraries** The libraries used in the implementation are correctly implemented;
7. **Correct Compilation** The compiler correctly compiles the libraries and the implementation of the proof system;
8. **Correct Execution** The machine correctly runs the executable; and
9. **Trusted IO** The input and output of the ITP can be trusted.

3.13 DO-330 Tool Qualification: An experience report

Lucas Wagner (Rockwell Collins - Cedar Rapids, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Lucas Wagner

This presentation gives an overview of the qualification of a test case generation tool that utilized model checking to generate tests. The tool is used to satisfy verification objectives, so it was qualified in accordance with DO-330 Tool Qualification Level 5 (TQL-5).

The presentation covers the rationale used for classifying the tool as a TQL-5 tool, the applicable DO-330 objectives for a TQL-5 tool, and examples of how the major objectives

were satisfied, including examples of test cases used in the qualification package developed for the test generation tool.

The purpose of this presentation was to give a concrete example and demonstrate that qualification of a tool is not overly complicated, but rather a straightforward, manageable process.

4 Discussion Groups

In addition to individual presentations, the seminar included four discussion groups organized around specific questions that arose during these presentations.

4.1 Why qualify a formal methods tool?

DO-178 (certification standard for software in civil aviation) states that qualification of a tool is needed when certification processes are eliminated, reduced, or automated by the use of a software tool without its output being verified.

For formal methods tools, two questions arise:

- Why use formal methods tools?
- Is qualification necessary?

One difficulty with DO-178 is that structural coverage testing is connected to many different certification objectives. Only some of these objectives can be mitigated using formal methods tools. A careful look at objectives is necessary to determine the economic benefit of using formal methods tools. In some cases, the business case may be derived from a new capability enabled by the use of a formal methods tool. For example:

- The ability to optimize code by using the CompCert compiler (see presentation by Xavier Leroy)
- The ability to increase processor utilization by performing worst case execution time (WCET) analysis with AiT
- The ability to host software at multiple criticality levels on same processor using a verified microkernel such as seL4

Formal methods qualification may, therefore, be a means to justify using the new capability.

Sometimes it is also possible to realize value without qualifying the tool. The use of a formal methods tool to detect and remove errors earlier in the development process is an example. Therefore, the benefit to be derived from a formal methods tool and how it is used in the development process should be carefully evaluated before assuming that qualification is needed.

4.2 How to qualify a formal methods tool?

In this group, we discussed qualification considerations for formal methods tools in the civil aviation context.

DO-333, the formal methods supplement for DO-178C, makes a distinction between a formal method and the tool which implements the method. Additional objectives for formal

methods are defined in DO-333 (appearing in tables A-3 through A-5). These objectives apply to the underlying method, and are in addition to any tool qualification activities that may be required. For each formal method used, the following activities should be done:

- Verification that the method has precise unambiguous, mathematically defined syntax and semantic
- Justification of the soundness of the analysis method
- Description and justification of any assumptions that are made in the analysis performed

Concerning tool qualification, there is nothing specific for formal methods tools required by the tool qualification document, DO-330. For verification tools (called TQL-5 tools), the main activities have to do with definition and verification of Tool Operational Requirements. These describe operation of the tool from a user perspective and demonstrate that the tool can satisfy the certification objectives for which it is being used. Some verification must be done showing that the tool does what the requirements say it should do (for example by the use of adequate test cases).

4.3 Compiler qualification strategies

Some formal methods are more difficult to classify in terms of how they fit in to a certification process and what kind of qualification is needed. A good example is the CompCert tool [1]. CompCert is a formally verified C compiler and thus could be seen as a development tool. However, DO-178 is designed to *not* require that the compiler be trusted. Instead, it assumes that executable object code will be verified by means of test (for compliance and robustness with respect to the requirements and to demonstrate structural coverage). The question is thus what is the certification objective that is automated by CompCert?

A possible answer is property preservation between source code and object code. In that case, CompCert could be considered as a verification tool automating this objective, and thus it would be qualified as a TQL-5 tool (according to DO-330). It would, however, be necessary to separate the code production part from the proof part inside the CompCert tool, which is not easy given the nature of the technique used (Coq).

Of course, CompCert could also be qualified as a development tool (TQL-1). In that case, since its assurance story is based on a formal proof, DO-333 (the formal methods supplement to DO-178C) could be applied for the qualification objectives concerning the tool development process. This combination of using formal methods to qualify a formal methods development tool has not been previously considered. In that case, the issue is to justify qualification of CompCert as a development tool on an economic point of view. Since a TQL-1 qualification is costly, it is necessary to determine what can we put in the balance to motivate the use of CompCert in place of a traditional compiler.

References

- 1 Leroy, X. (2009). Formal verification of a realistic compiler. In *Communications of the ACM*, volume 52, number 7, pages 107–115.

4.4 Comparison of qualification in different domains

In this discussion group we discussed the similarities and difference among qualification standards in different domains. The standards considered were:

- DO-178C Software Considerations in Airborne Systems and Equipment Certification and DO-330 Software Tool Qualification Considerations
- IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements
- ISO 26262 Road vehicles — Functional safety —Part 8: Supporting processes

The comparison concerned the following questions:

- When is tool qualification required?
- What levels of qualification are defined and what is the purpose of each?
- What activities are required to achieve qualification?