Qualification of a Model Checker for Avionics Software Verification

Lucas Wagner[1], Alain Mebsout[2], Cesare Tinelli[2], Darren Cofer[1], and Konrad Slind[1]

[1] Advanced Technology Center, Rockwell Collins
{lucas.wagner, darren.cofer, konrad.slind}@rockwellcollins.com
[2] The University of Iowa
{alain-mebsout, cesare-tinelli}@uiowa.edu

Abstract. Formal methods tools have been shown to be effective at finding defects in safety-critical systems, including avionics systems in commercial aircraft. The publication of DO-178C and the accompanying formal methods supplement DO-333 provide guidance for aircraft manufacturers and equipment suppliers who wish to obtain certification credit for the use of formal methods for software development and verification. However, there are still a number of issues that must be addressed before formal methods tools can be injected into the design process for avionics systems. DO-178C requires that a tool used to meet certification objectives be *qualified* to demonstrate that its output can be trusted. The qualification of formal methods tools is a relatively new concept presenting unique challenges for both formal methods researchers and software developers in the aerospace industry.

This paper presents the results of a recent project studying the qualification of formal methods tools. We have identified potential obstacles to their qualification and proposed mitigation strategies. We have conducted two case studies based on different qualification approaches for an open source formal verification tool, the Kind 2 model checker. The first case study produced a qualification package for Kind 2. The second demonstrates the feasibility of independently verifying the output of Kind 2 through the generation of proof certificates and verifying these certificates with a qualified proof checker, in lieu of qualifying the model checker itself.

Keywords: qualification, certification, model checking, software verification

1 Introduction

Civilian aircraft must undergo a rigorous certification process to establish their airworthiness. Certification encompasses the entire aircraft and all of its components, including the airframe, engines, and on-board computing systems. Many of these systems utilize software. Guidance for the certification of airborne software is provided in *DO-178C: Software Considerations in Airborne Systems and Equipment Certification* [1].

Formal methods tools have been shown to be effective at finding and eliminating defects in safety-critical software [2]. In recognition of this, when DO-178C was published it was accompanied by *DO-333: Formal Methods Supplement to DO-178C and DO-278A* [3]. This document provides guidance on how to acceptably use formal methods to satisfy DO-178C certification objectives. However, there are a number of issues that must be addressed before formal methods tools can be fully integrated into the development process for aircraft software. For example, most developers of aerospace 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. Further, evidence must be provided of a formal method's soundness, a concept that is not well understood by most practicing engineers. Similarly, most developers of formal methods tools are unfamiliar with certification requirements and processes.

DO-178C requires that a tool used to meet its objectives must be *qualified* in accordance with the tool qualification document *DO-330: Software Tool Qualification Considerations* [4]. The purpose of the tool qualification process is to obtain confidence in the tool functionality. The effort required varies based on the potential impact a tool error could have on system safety. The qualification of formal verification tools poses unique challenges for both tool developers and aerospace software engineers.

Previous NASA-sponsored work has described in detail how one might use various formal methods tools to satisfy DO-178C certification objectives [5]. This paper presents the results of a subsequent study designed to address the qualification of formal methods tools. The goal of the effort was to interpret the guidance of DO-330 and DO-333 and provide critical feedback to the aerospace and formal methods research communities on potential pitfalls and best practices to ensure formal methods tool users and developers alike can successfully qualify their tools.

We are aware of several commercial tool vendors who have successfully qualified formal methods tools. For example, Polyspace by MathWorks and Astreé by AbsInt both have DO-178C qualification kits available. In the early stages of this project we helped to organize a Dagstuhl Seminar on Qualification of Formal Methods Tools [6] to engage both formal methods researchers and certification experts. The seminar included presentations on qualification work for the Alt-Ergo theorem prover [7], SPARK verification tools [8], and the CompCert compiler [9], as well as experience reports on qualification guidance and efforts in other industries. A good summary of tool qualification requirements in other domains is found in [10].

In this paper we examine the qualification of a model checker for use in verification of avionics software. The success of model checking is largely due to the fact that it is a highly automated process, generally requiring less expertise than an interactive theorem prover [11]. One clear strength of model checkers is their ability to return precise error traces witnessing the violation of a given safety property. However, most model checkers are currently unable to return any form of corroborating evidence when they declare a safety property to be satisfied. When used to satisfy certification objectives for aircraft software, a model checking tool would therefore need to qualified.

An alternative is to instrument the model checker so that in addition to its safety claims, it generates a *proof certificate*, which is an artifact embodying a proof of the claims. Such a certificate can then be validated by a qualified certificate checker. By reducing the trusted core to the certificate checker, this approach facilitates the integration of formal method tools into the development processes for aircraft software. It redirects tool qualification requirements from a complex tool, the model checker, to a much simpler one, the certificate checker.

The main contribution of this paper is presentation of these two approaches to qualification as applied to the Kind 2 model checker [12]. Section 2 provides a brief overview of the certification guidance for software in commercial aircraft. Section 3 describes the tool qualification process that is used to establish trust in the tools that are used in avionics software development. Sections 4 and 5 describe two case studies that illustrate different approaches to qualification: direct qualification of the Kind 2 model checker and qualification of the certificate checker for a proof-generating enhancement of the model checker. Section 6 provides conclusions and lessons learned from the project. The complete NASA technical report and qualification artifacts are available at [13].

2 Aircraft Software and Certification

Certification is defined in DO-178C as legal recognition by the relevant certification authority that a product, service, organization, or person complies with its requirements. In the context of commercial aircraft, the relevant certification authority is the FAA in the U.S. or EASA in Europe. The requirements referred to are the government regulations regarding the airworthiness of aircraft operating in the National Airspace System (NAS). In practice, certification consists primarily of convincing representatives of a government agency that all required steps have been taken to ensure the safety, reliability, and integrity of the aircraft. Certification differs from verification in that it focuses on evidence provided to a third party to demonstrate that the required activities were performed completely and correctly, rather on performance of the activities themselves.

The stakeholders in the civil aviation domain (regulators, airframers, equipment manufacturers) have developed a collection of guidance documents defining a certification process which has been accepted as the standard means to comply with regulations. The process includes system development, safety assessment, and design assurance. DO-178C focuses on design assurance for software, and is intended to make sure that software components are developed to meet their requirements without any unintended functionality.

DO-178C does not prescribe a specific development process, but instead identifies important activities and design considerations throughout a development process and defines objectives for each of these activities. It identifies five software levels, with each level based on the impact of a software failure on the

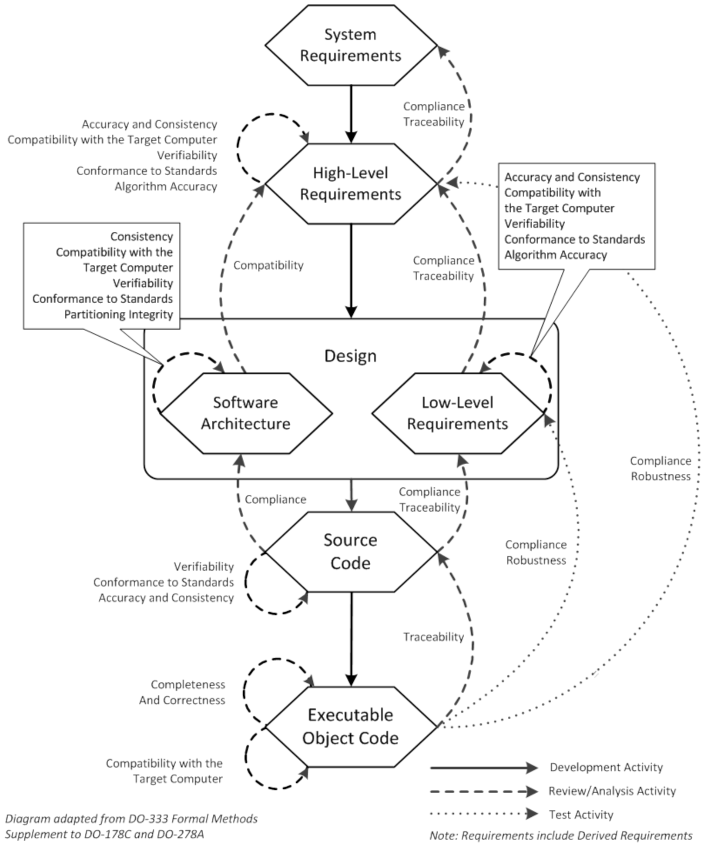

Fig. 1. DO-178C certification activities required for Level A software.

overall aircraft function. As the software criticality level increases, so does the number of objectives that must be satisfied. Depending on the associated software level, the process can be very rigorous (Level A) or non-existent (Level E). Objectives are summarized in a collection of tables covering each phase of the development process. Figure 1 shows the objectives required for the most critical avionics software, Level A.

One of the foundational principles of DO-178C is requirements-based testing. This means that the verification activities are centered around explicit demonstration that each requirement has been met. A second principle is complete coverage, both of the requirements and of the code that implements them. This means that every requirement and every line of code must be examined in the verification process. Furthermore, several metrics are defined which specify the degree of structural coverage that must be obtained in the verification process, depending on the criticality of the software being verified. A third principle is traceability among all of the artifacts produced in the development process.

Together, these objectives provide evidence that all requirements are correctly implemented and that no unintended function has been introduced.

When DO-178C was developed, guidance specific to new software technologies was provided in associated documents called *supplements* which could add, modify, or replace objectives in the core document. New supplements were developed in the areas of model-based development, object-oriented design, and formal methods, as well as an additional document containing expanded guidance on tool qualification. DO-178C and its associated documents were published in 2011 and accepted by the FAA as a means of compliance with airworthiness regulations in 2013.

3 Qualification

Guidance governing tool qualification is provided in Section 12.2 of DO-178C. A tool must be qualified if the following two conditions are met:

1. Any of the processes of DO-178C are eliminated, reduced, or automated by the use of a software tool, and
2. The output of the tool is used without being verified.

This means that if a tool is used to identify software defects rather than, for example, demonstrating that source code satisfies its low-level requirements (a DO-178C objective), then qualification is not required. Similarly, if a tool is used to generate test cases, but those test cases will be manually reviewed for correctness, then qualification is not required.

When it is determined that tool qualification is required, the purpose of the qualification process is to ensure that the tool provides confidence at least equivalent to the processes that were eliminated, reduced, or automated by the tool.

Tool qualification is context-dependent. If a tool previously qualified for use on one system is proposed for use on another system, it must be re-qualified in the context of the new system.

DO-330 outlines a process for demonstrating a tool's suitability for satisfying DO-178C objectives that it is being used to eliminate, reduce, or automate. The qualification process is similar to the software verification process defined in DO-178C. Qualification amounts to accomplishing a set of activities with corresponding objectives to:

- Identify the DO-178C objectives that the tool is eliminating, reducing, or automating
- Specify which functions of the tool are being relied upon
- Create a set of requirements that precisely identify those functions
- Develop a set of test cases showing that the tool meets those requirements.

3.1 Tool Qualification Level

As in the certification process itself, there are varying levels of rigor associated with tool qualification. The Tool Qualification Level (TQL) is similar to the software level in DO-178C and defines the level of rigor required by the qualification process. TQL-1 is the most rigorous, while TQL-5 is the least rigorous.

The required TQL is determined by identifying the tool's impact on the software development process. The impact is characterized by determining the impact of a error in the tool. DO-178C provides three criteria to characterize the impact of an error in the tool:

Criterion 1. A tool whose output is part of the airborne software and thus could insert an error.

Criterion 2. A tool that automates verification processes and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:

- Verification processes other than those automated by the tool, or
- Development processes that could have an impact on the airborne software.

Criterion 3. A tool that, within the scope of its intended use, could fail to detect an error.

A code generator in a model-based development process is an example of a Criterion 1 tool. We expect that most formal methods tools will be used as part of the software verification process and will, therefore, fall into Criteria 2 or 3. That is, they will not be used to generate airborne software, but will be used to verify that the airborne software is correct.

The distinction between Criteria 2 and 3 depends on exactly which processes the tool is eliminating, reducing, or automating. For example, if an abstract interpretation tool determines that division-by-zero cannot occur and this is used to satisfy DO-178C objectives related to the accuracy and consistency of the source code (Objective A-5.6), then the tool is Criterion 3. However, if those results are also used to justify elimination of robustness testing related to division-by-zero in the object code (Objectives A-6.2 and A-6.4), then the tool becomes a Criterion 2 tool. An unofficial rule of thumb is that when a tool addresses objectives from multiple tables of DO-178C (corresponding to different development phases), it is likely a Criterion 2 tool.

The required TQL is determined by the combination of its impact and the DO-178C software level to which the tool is being applied, as shown in Table 1.

In summary, formal methods tools used to satisfy verification process objectives of DO-178C will usually need to be qualified at TQL-5. TQL-4 qualification would only be required if the tool is determined to fall into Criterion 2 and it is being used in the verification of Level A or B software.

3.2 DO-330 and Tool Qualification Objectives

Once the TQL is determined, the required tool qualification objectives are defined by DO-330. Like DO-178C, these objectives are summarized in a collection

Table 1. Determination of Tool Qualification Level.

Software Level	**Criterion**		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

of tables. Table 2 shows the number of objectives to be satisfied in each area for TQL-4 and TQL-5. Note that objectives for a particular TQL are cumulative, so that the TQL-5 objectives are a subset of the TQL-4 objectives.

Table 2. DO-330 tool qualification objectives.

DO-330 Qualification Objectives	**TQL-4**	**TQL-5**
T-0: Tool Operational Processes	7	6
T-1: Tool Planning Processes	2	
T-2: Tool Development Processes	5	
T-3: Verification of Outputs of Tool Requirements Process	8	
T-4: Verification of Outputs of Tool Design Process	1	
T-5: Verification of Outputs of Tool Coding & Integ. Process		
T-6: Testing of Output of Integration Process	2	
T-7: Verification of Outputs of Tool Testing	2	
T-8: Tool Configuration Management	5	2
T-9: Tool Quality Assurance Process	2	2
T-10: Tool Qualification Liaison Process	4	4
Total number of objectives	38	14

Table 2 highlights an important distinction between the qualification objectives. The gray rows (qualification objective tables T-1 through T-7) are objectives related to the development processes of the tool itself. The other rows (T-0 and T-8 through T-10) are objectives related only to the use of the tool. Thus there is a clear distinction between the tool developer context and the tool user context. Furthermore, TQL-5 qualification only requires objectives from the tool user context. This means that TQL-5 qualification is significantly simpler than TQL-4 because it does not require information about how the tool was developed. If a tool was built by a third party, TQL-4 qualification may be difficult to achieve. In particular, since many formal methods tools arise from academic research activities, the artifacts required for TQL-4 qualification may not be available.

Another interesting point is that tool qualification is always performed in the context of a particular aircraft development effort. This means that certain tool functions may not be utilized or addressed in a qualification. For example, qualification of a model checker may only need to cover variables of primitive data types while ignoring composite types such as arrays, records, and tuple types, if those are not relevant for the given application.

Once the proper TQL is determined and the objectives have been identified, qualification is simply a matter of demonstrating that each objective is satisfied. For a TQL-5 qualification, the bulk of this effort is associated with DO-330 Table T-0, Tool Operational Processes, and involves defining and verifying Tool Operational Requirements which describe tool capabilities necessary to satisfy the claimed certification objectives.

4 Case Study: Kind 2 Model Checker

The first case study describes the activities and artifacts necessary to complete a TQL-5 qualification of the Kind 2 model checker based on the guidance in DO-330. Our goal is to provide a concrete example that illustrates the qualification process for a typical formal methods tool and could be used as a pattern by others. We also identify challenges or lessons learned in the process. The qualification package is available as part of the NASA final report for the project.

Kind 2 [14] is an open-source, multi-engine, SMT-based automatic model checker for safety properties of programs written in the synchronous dataflow language Lustre [15]. It takes as input a Lustre file annotated with properties to be proved, and outputs for each property either a confirmation or a counterexample, a sequence of inputs that falsifies the property.

This case study is based on earlier work [5] in which various formal methods were used to satisfy DO-178C and DO-333 objectives for verification of a representative Flight Guidance System (FGS). In one of the examples, the Kind 2 model checker was used to verify that a model of the FGS mode logic satisfies its high-level requirements. This qualification case study extends that work by performing the activities needed to qualify Kind 2 for accomplishing the certification objectives described in the earlier work.

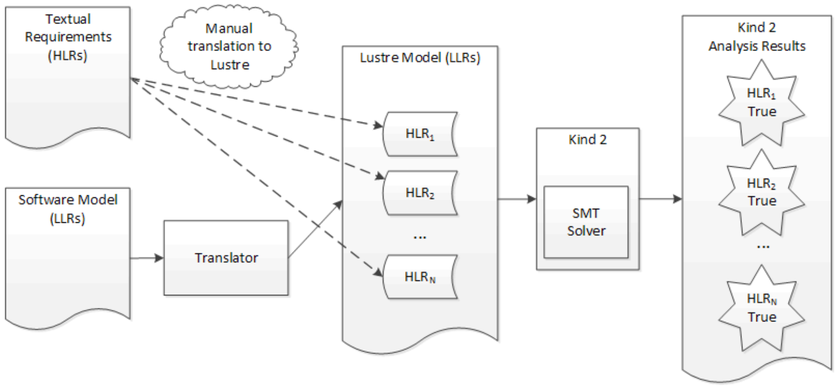

Fig. 2. Verification using qualified Kind 2 model checker.

In this example, the mode logic was expressed as a state machine model in Simulink Stateflow, and serves as low-level requirements for the source code that will be generated from it. A Rockwell Collins tool was used to translate this model into Lustre for analysis by the Kind 2 model checker. Textual high-level requirements for the model logic were manually translated to Lustre and merged with the mode logic Lustre model. The overall tool chain is shown in Figure 2. This case study is limited to qualification of the model checker and ignores (for now) the model translation tools.

4.1 Need for Tool Qualification

In this case study Kind 2 is being used to automate processes that satisfy the objectives of Verification of Outputs of Software Design Process (DO-178C Table A-4). This includes, for example:

- **A-4.1** Low-level requirements comply with high-level requirements.
- **A-4.2** Low-level requirements are accurate and consistent.
- **A-4.7** Algorithms are accurate.

Furthermore, the outputs of Kind 2 will not be independently verified. This establishes the need for qualification.

The required TQL is established by determining the impact of Kind 2 on the software development process. In this context the tool:

- Cannot insert an error into the airborne software.
- Could fail to detect an error in the airborne software.
- Is not used to justify the elimination or reduction of other verification processes or development processes that could have an impact on the airborne software.

Therefore, Criterion 3 applies so Kind 2 should be qualified to TQL-5.

4.2 Tool Qualification Objectives

The work performed to satisfy TQL-5 qualification objectives is summarized below:

T-0.1 Tool qualification need is established. (Rationale for tool qualification and determination of the required TQL is described in Section 4.1.)

T-0.2 Tool Operational Requirements are defined. Definition of the Tool Operational Requirements (TOR) and their verification in objective T-0.5 are the key qualification activities. The Tool Operational Requirements identify how the tool is to be used within the software life cycle process. This objective requires the identification of the tool usage context, tool interfaces, the tool operational environment, tool inputs and outputs, tool operational requirements, and the operational use of the tool. The focus here is on the tool performance from the perspective of the tool user and what capabilities the tool provides in the software development process.

We have specified 111 TORs that must be verified for Kind 2. These requirements cover:

- The features of the Lustre language used by Kind 2 in this context
- Input validation features
- Properties that must be correctly analyzed as true or false.

Since the requirements will be verified by testing performed on Kind 2, they cover a finite subset of the Lustre grammar. Conservative bounds on the length of inputs are established and validated.

T-0.3 Tool Executable Object Code is installed in the tool operational environment. Identification of the specific versions of the tool and its dependencies, instructions of how to install the tool, and a record of actually installing the tool are required to meet this objective. Qualification was based on Kind 2 version 1.0.1 and the Z3 SMT solver [16] (version 4.4.2).

T-0.5 Tool operation complies with the Tool Operational Requirements. This objective demonstrates that the tool complies with its TORs. This objective is covered in three parts. First, the review and analysis procedures used to verify the TORs are defined. Secondly, we identify a set of tests, referred to as the Tool Operational Test Cases and Procedures, that when executed, demonstrate that Kind 2 meets its TORs. Finally, the results of actually executing the test procedures within the Tool Operational Environment must be collected.

T-0.6 Tool Operational Requirements are sufficient and correct. This objective is satisfied by ensuring that the TORs adequately address the tool usage context, the tool operational environment, the input accepted by the tool, the output produced by the tool, required tool functions, applicable tool user information, and the performance requirements for the tool.

T-0.7 Software life cycle process needs are met by the tool. This objective is satisfied by the review, analysis, and testing results used to satisfy the TORs.

Other objectives (T-8, T-9, T-10). Tool configuration management, quality assurance, and qualification liaison process. Most of the data required by these objectives are highly dependent on the context and the processes of the applicant organization and can only be meaningfully defined for an actual software development and tool qualification effort.

4.3 Results

The purpose of this qualification package was to provide a complete case study containing a detailed set of tool operational requirements and test procedures. It is anticipated that this qualification package contains all of the necessary information such that it could be used within an avionics certification effort. No barriers were found that would prevent qualification of Kind 2.

One interesting result from the Tool Qualification Liason process is T-10.4 Impact of Known Problems on TORs. During verification of the TORs, some errors were identified.These have either been corrected or will be corrected in the near future. However, such errors do not preclude use of the tool in certification activities, as long as the impact and functional limitations on tool use are identified.

The qualification package and results were reviewed by certification experts at Rockwell Collins and determined to meet the requirements of DO-330. Successfully using it would require an applicant to provide detailed information to support the tool qualification objectives from Table T-8, T-9, and T-10, which are specific to an organization's configuration management, quality assurance, and certification practices respectively. We expect that it could be used as the starting point for tool qualification in an actual avionics software development effort or as a pattern for qualification of another tool.

5 Case Study: Proof-Generating Model Checker

The second qualification case study is based on a proof-generating version of the Kind 2 model checker that is supported by a separate proof checker [14]. In this approach, the proof checker verifies the output of the model checker. This removes the need to qualify a complex tool (the model checker) and instead requires qualification of a much simpler one (the proof checker). By reducing the trusted core to the proof checker, we may be able to reduce the qualification effort required and enhance the overall assurance.

This case study is based on the same software development context as the first, and involves using the model checker to satisfy the same certification objectives for verifying the FGS mode logic. The qualification package developed for the proof checker tool is available as part of the project final report.

5.1 Development of a Proof-Generating Version of Kind 2

For this effort we have used the SMT solver CVC4 [17] with Kind 2. CVC4 is a solver for first-order propositional logic modulo a set of background theories such as integer or real linear arithmetic. Our work relies heavily on the proof production capabilities of CVC4. A unique aspect of CVC4 proofs is that they are fine grained. This means they are very detailed and checking them is only a matter of carefully following and replaying the steps in the proof certificate. In contrast, proofs produced by other solvers require the final proof checker to perform substantial reasoning to reconstruct missing steps.

The proof checker which was qualified in this case study, named Check-It, is an instantiation of the Logical Framework with Side Conditions (LFSC) proof checker [18]. The resulting tool architecture is shown in Figure 3, which includes both the unqualified Kind 2 model checker and the qualified Check-it proof checker.

Kind 2 is used to generate two separate proof certificates:

- A proof certificate (PC) for safety properties of the transition system corresponding to Lustre model being verified.
- A front-end certificate (FEC) that provides evidence that two independent tools have accepted the same Lustre input model and produced the same first order logic (FOL) internal representation.

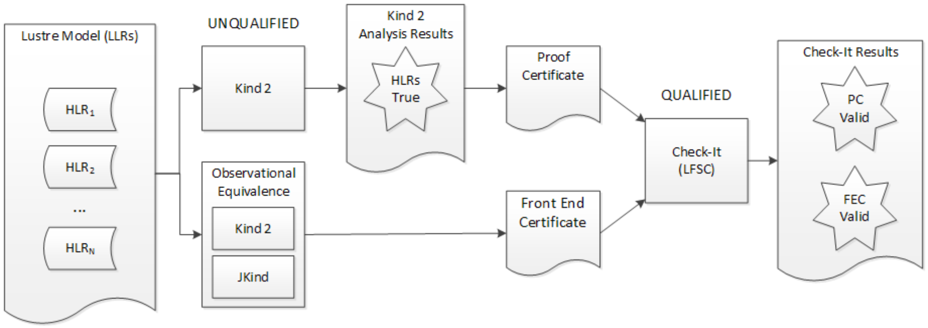

Fig. 3. Verification using Kind 2 and a qualified proof checker.

The PC summarizes the work of the different analysis engines used in Kind 2. This includes bounded model checking (BMC), k-induction, IC3, as well as additional invariant generation strategies. In practice it takes the form of a k-inductive strengthening of the properties.

This intermediate certificate is checked by CVC4, from which we extract proofs to reconstruct safety arguments using the rules of k-induction. Proofs are produced in the language of LFSC.

To make the whole process efficient and scalable, certificates are first minimized before being checked. An iterative process takes care of this phase by efficiently lowering the bound k and removing any superfluous information contained within the certificate.

The FEC is necessary to ensure that the proof in the PC is actually about the input model provided. Without this step, it is possible that the (unqualified) model checker could produce a valid PC that is unrelated to the input model. The FEC is generated in the form of observational equivalence between two internal representations generated by independently developed front ends. In our case, the two front ends are Kind 2 itself and JKind, a Lustre model checker inspired by Kind but independently developed by Rockwell Collins [19]. Observational equivalence between the two FOL representations is recast as an invariant property. Checking that property yields a second proof certificate from which a global notion of safety can be derived and incorporated in the LFSC proof.

The trusted core of this approach consists of:

- The LFSC checker (5300 lines of C++ code).
- The LFSC signatures comprising the overall proof system in LFSC, for a total of 444 lines of LFSC code.
- The assumption that Kind 2 and JKind do not have identical defects that could escape the observational equivalence check. We consider this reasonable since the tools were produced by different development teams using different programming languages.

5.2 Qualification of Check-It

The approach of using a qualified tool to check the results of an unqualified tool is not unprecedented. FAQ D.7 of DO-330 provides guidance for exactly this “two tool” approach. Recall that qualification of a tool is necessary when it is used to eliminate, reduce, or automate DO-178C processes and when the outputs of the tool are not verified. Kind 2 and Check-It are used to satisfy the same objectives for the FGS mode logic as described in Section 4. The outputs of the Kind 2 analysis, a set of proof certificates, are verified using the Check-It proof checking tool. According to the guidance in DO-330 FAQ D.7, this process is acceptable if the Check-It tool is qualified.

Determination of required TQL is the same as in Section 4. Check-it is used only to verify proof certificates produced by Kind 2 and so it is a Criterion 3 tool. Therefore, Check-It must be qualified at TQL-5.

The qualification objectives for Check-It were the same as for Kind 2, so we only address the differences here. Since Check-It is simpler than Kind 2, defining its TORs was comparatively straightforward. Inputs to the tool are proof certificates (PC and FEC) that are composed of proof rules defined in six signature files. We have specified 82 TORs that must be verified for Check-It.

Objectives for verification of tool operation were accomplished by a combination of peer review and testing. Test cases cover presence and validity of certificates, compatibility with certificates produced by Kind 2, performance requirements, and proof rule acceptance. Peer review of the proof rules in the signatures files used by Check-It was conducted to identify any potential trust issues. Results from this review were used to identify additional test cases (for example, to preclude the acceptance of unsound rules).

DO-330, FAQ D.7 provides additional information on the use of a qualified tool (Check-It) to check the results of an unqualified tool (Kind 2). This FAQ identifies factors that should be considered to prevent the possibility of errors in both the unqualified tool and the qualified tool. The primary concern is to identify the interaction between tools in the case of various failures in the unqualified tool (for example, if Kind 2 fails to produce a PC or a FEC, or if either is found to be incorrect by Check-It).

The FAQ also identifies four additional concerns that apply in this situation, and which have been addressed in the qualficiation package:

- Coverage of verification objectives for the unqualified tool’s output
- Operating conditions of the qualified tool
- Common cause avoidance
- Protection between tools

5.3 Results

To summarize, we found nothing about the “two tool” proof-checking approach that would prevent successful tool qualification. Checking the PC validates the Kind 2 analysis and checking the FEC provides an argument that the emitted PC corresponds to the original Lustre file. If Kind 2 produces incorrect, malformed, or missing certificates Check-It highlights the error. The tools use dissimilar technical approaches, one performing model checking and the other proof checking, minimizing the chance for any common cause failure. The TORs for Check-It were much simpler to define and verify than for Kind 2. However, the proof checking approach was more challenging to explain to certification experts and, consequently, would be inherently riskier to implement. We estimate the overall effort of this approach to be about 75% of the effort required to qualify Kind 2 itself. An added benefit, however, is that the qualified proof checker could be reused with future improved versions of Kind 2 (provided the proof format remains the same), or even with other model checkers which would produce certificates in the same format.

6 Conclusions

In this paper we have explored the qualification of formal methods tools within the context of avionics certification. This effort produced useful examples and artifacts for two qualification case studies, and also provided insight into the qualification process for formal methods tools that should be useful to software developers, tool developers, tool users, and certification experts. Combined with the prior work on Formal Methods Case Studies for DO-333, it provides a comprehensive set of case studies for using and qualifying formal method tools for avionics software development.

The work reveals that qualification at TQL-5 can be a straightforward task. The guidance of DO-330 does not require any activities that are especially difficult or costly for qualification of a model checker. However, the guidance does suggest that tools from the research community may be difficult to qualify at TQL-4 due to the requirements for tool development artifacts including tool requirements, test cases, tool design, and architectural descriptions. Formal methods tool developers who desire to have their tools used in the avionics industry should keep this in mind.

In addition, this work highlights the need for good software engineering practices for formal methods tools used in certification. The relatively high complexity of internal translations, optimizations, and analysis algorithms increases the likelihood that defects will be identified. Bug tracking facilities are absolutely essential for users to understand a tool's limitations.

Lastly, we developed a proof-generating enhancement of the Kind 2 model checker, and explored the impact of this capability on tool qualification. We produced qualification packages for both Kind 2 and for the proof checker for certificates generated by Kind 2. We determined that the "two tool" proof checker approach was viable from a qualification standpoint and provides increased assurance. However, it was not dramatically easier or less costly to qualify and was definitely more difficult to explain and justify to certification experts.

Based purely on cost and perceived risk, we expect that TQL-5 qualfication of a model checker would be the approach preferred by most avionics software developers. The qualified proof checker approach provides significant advantages in terms of greater assurance and modularity, which may be attractive for developers interested in “future-proofing” their verification process. By keeping the model checker separate and free from the need for qualification, improved features and functionality can be more easily incorporated without impacting the qualified (and therefore less flexible) proof checker.

Acknowledgments. This work was funded by NASA contract NNL14AA06C.

References

1. RTCA DO-178C: Software Considerations in Airborne Systems and Equipment Certification, Washington, DC. (2011)
2. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: Practice and experience. ACM Comput. Surv. **41** (2009)
3. RTCA DO-333: Formal Methods Supplement to DO-178C and DO-278A, Washington, DC. (2011)
4. RTCA DO-330: Software Tool Qualification Considerations, Wash., DC. (2011)
5. Cofer, D., Miller, S. In: DO-333 Certification Case Studies. Springer International Publishing, Cham (2014) 1–15
6. Cofer, D., Klein, G., Slind, K., Wiels, V.: Qualification of Formal Methods Tools (Dagstuhl Seminar 15182). Dagstuhl Reports **5** (2015) 142–159
7. OCamlPro: Alt-ergo (2013) `https://alt-ergo.ocamlpro.com/`.
8. AdaCore: SPARK Pro (2014) `http://www.adacore.com/sparkpro/`.
9. Leroy, X.: A formally verified compiler back-end. Journal of Automated Reasoning **43** (2009) 363–446
10. Camus, J.L., DeWalt, M.P., Pothon, F., Ladier, G., Boulanger, J.L., Blanquart, J.P., Quere, P., Ricque, B., Gassino, J.: Tool qualification in multiple domains: Status and perspectives. In: Embedded Real Time software and Systems, Springer (2014)
11. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Commun. ACM **53** (2010) 58–64
12. Champion, A., Mebsout, A., Sticksel, C., Tinelli, C.: The Kind 2 model checker. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. (2016) 510–517
13. NASA: Qualification of Formal Methods Tools Under DO-330 (2017) `https://shemesh.larc.nasa.gov/fm/FMinCert/DO-330-case-studies-RC.html`.
14. Mebsout, A., Tinelli, C.: Proof certificates for SMT-based model checkers for infinite-state systems. In: Proceedings of 16th Conference on Formal Methods in Computer-Aided Design (FMCAD), Springer (2016)
15. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D.: The synchronous dataflow programming language lustre. In: Proceedings of the IEEE. (1991) 1305–1320
16. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th Int’l Conference, TACAS 2008. Volume 4963 of LNCS., Springer (2008) 337–340
17. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV, Springer (2011) 171–177
18. Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods in System Design **41** (2013) 91–118
19. Gacek, A.: JKind - a Java implementation of the KIND model checker (2014) `https://github.com/agacek/jkind`.