QUALIFICATION OF PROOF ASSISTANTS, CHECKERS, AND GENERATORS: WHERE ARE WE AND WHAT NEXT?*

A PREPRINT

Mario Gleirscher University of Bremen Bibliothekstrasse 5 28359 Bremen, Germany gleirsch@uni-bremen.de Robert Sachtleben University of Bremen Bibliothekstrasse 5 28359 Bremen, Germany rob_sac@uni-bremen.de Jan Peleska Verified Systems International Am Fallturm 1 28359 Bremen, Germany peleska@uni-bremen.de

February 21, 2023

ABSTRACT

Cyber-physical systems, such as learning robots and other autonomous systems, employ highintegrity software in their safety-critical control. This software is developed using a range of tools some of which need to be qualified for this purpose according to international standards. In this article, we first evaluate the state of the art of tool qualification for proof assistants, checkers (e.g., model checkers), and generators (e.g., code generators, compilers) by means of a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis. Our focus is on the qualification of tools in the three mentioned categories. Our objective is to assess under which conditions these tools are already fit or could be made fit for use in the practical engineering and assurance of high-integrity control software. In a second step, we derive a viewpoint and a corresponding range of suggestions for improved tool qualification from the results of our SWOT analysis.

Keywords Deductive verification · Model checking · Certified compilers · Cyber-physical systems · Control software engineering

1 Introduction

High-integrity software and control engineering is about the engineering of electric, electronic, or programmable electronic systems required to operate at high degrees of dependability. Dependability may include, for example, highly reliable components, highly secure treatment of data and signals, and highly robust enforcement of safe machine behaviour. The more demanding such requirements are, the more sophisticated methods, techniques, and technologies are needed in control systems engineering [1]. Control software and systems subjected to such requirements are used in many cyber-physical systems, such as intelligent collaborative robots or autonomous vehicles.

The most demanding trustworthiness requirements can be tackled by the paradigm of *formal design and verification*. This paradigm fosters the use of formal logic and mathematics in the construction and verification of key engineering artefacts (e.g., hardware, software, models thereof). Modern formal design and verification is highly automated, that is, its results (e.g., proofs, verdicts, models, code, executables) are automatically or interactively produced by tools and, thus, rely on the correctness of these tools. In this context, tools can be single points of failure with severe impact on control system assurance and operation. For example, performance-increasing optimisations in untrustworthy compilers [2] have to remain switched off until these compiler components are sufficiently verified.

Hence, the standards applicable to high-integrity systems development, verification and validation (V&V), and certification agree on the fact that the trustworthiness of tools automating essential steps of the life cycle process needs to be established in a way that is comprehensible and can be checked by independent parties, for example, the certification

^{*}CC BY-NC-ND 4.0, the authors. This preprint was accepted for publication in the Science of Computer Programming journal and is now available under https://doi.org/10.1016/j.scico.2023.102930.

authorities [3, 4, 5, 6]. The process for establishing this trustworthiness is called *tool qualification* (TQ) [4] or *tool validation* [5]. We use the former term (or 'qualification' for short) to refer to the assurance of design and verification tools or their results. Qualification requirements in certification [4, 7, 6, 8] are justified by the fact that faulty tools automating parts of the development process (e.g., compilers and other code generators) may directly cause erroneous software to be deployed in the target system. Faulty tools automating parts of the V&V process (e.g., test automation tools, model checkers, and proof assistants) may cause errors in the target software to be overlooked.

Challenges. Reinforced by certification needs, the qualification of *formal* design and verification tools has gained traction in the formal methods researcher and practitioner communities [9, 10], such as with proof assistants [11], model checkers [12], compilers [13, 14], abstract interpreters, SATisfiability/Modulo Theory (SAT/SMT) solvers [15], and conformance testers [16, 17]. Despite these efforts, tool qualification is still not necessarily a key issue, even in recent industry-focused surveys [18]. Where qualification is seriously considered, such as in the aforementioned works, efforts are either new [19], limited in their scope [12], or in an early stage. Most importantly, results are hence not yet broadly transferred into high-integrity systems practice, and important questions are still to be explored.

Research Hypothesis and Question. Motivated by the expectation that formal design and verification will become a key enabler on the pathway to trustworthy autonomous control, we focus on the following question:

What are the key issues in getting formal design and verification tools qualified, so that certification credit can be obtained for their generated results according to the industrial standards applicable in high-integrity control systems and software engineering?

To limit the *scope of our assessment*, we focus on proof assistants, checkers, and generators as characterised below. Hence, we decided to exclude abstract interpretation and conformance testing tools from our assessment, but take into account SAT/SMT solvers as far as employed in the three focused categories.

The Three Tool Categories. We hope that readers will find the following characterisation of formal design and verification tools as a useful guide to the understanding of our assessment below.

- **Proof Assistants** are formal verification tools that help engineers with the *interactive* deductive proving of theorems, for example, about a system model or the correctness of control software. Prominent examples for such tools include Coq [20], Isabelle [21], KeYmaera X [22], Lean [23], and PVS [24].²
- **Checkers** are formal verification tools that allow one to *automatically* check whether a (*model*, *property*)-pair is element of a particular satisfaction relation (⊨) or whether a *theorem* can be automatically deduced from a set of *axioms* given the *inference rules* of a certain theory or formal system. Prominent examples for such tools include FDR4 [25], KIND 2 [26], NUXMV [27], PRISM [28], PROB [29], PROLOG [30], SPIN [31], and UPPAAL [32].³
- **Generators** are formal design tools allowing to automatically translate or transform artefacts of an input language (e.g., C++, Mathworks Simulink⁴) into artefacts of an output language (e.g., an ARM executable,⁵ SPARK [33]), while *provably preserving* a range of correctness *properties* (e.g., type and memory safety). Prominent examples for such tools include certified compilers (e.g., CompCert [14]) and model-to-code translators (e.g., the SCADE-to-X code generator [13]).

Overview. In Sec. 2, we recapitulate key requirements on tool qualification from the standards. We assess the current situation by means of a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis in Sec. 3 and elaborate our viewpoint with respect to the above question in Sec. 4. A summary with concluding remarks is presented in Sec. 5.

2 What Do Standards Require?

This section deals with the question:

What do certification authorities (and industries) hope for in regard to tool qualification of proof assistants, checkers, and generators?

²Note that their code generation capabilities make these tools to be generators as well.

³As already mentioned and to limit the scope, we exclude solvers such as CVC4 and Z3.

⁴https://www.mathworks.com/products/simulink.html

⁵https://developer.arm.com/documentation/100166/0001/Programmers-Model

There are several standards with guidance for tool qualification that apply to the three tool categories mentioned above. According to our assessment, the guidelines in the avionic standard RTCA DO-178C [4] and its addendum RTCA DO-330 [7] currently specify the most comprehensive and strictest qualification requirements (see Wagner et al. [12] for a summary). The standard introduces *tool qualification levels* TQL-1 to TQL-5 [4, Sec. 12.2.2], [7]. A TQL corresponds to a *set of qualification requirements* that depends on (a) the criticality of the target system to be developed, and (b) the impact that tool failures could have on the software life cycle. The impact is classified according to *Criteria 1*: the tool output becomes part of the target software, *Criteria 2*: the tool automates a verification activity, and its output is also used to eliminate or reduce parts of other verification activities and/or parts of the development, and *Criteria 3*: the tool automates a verification activity, so it can only overlook errors, but never inject errors into the target software.

Depending on the applicable TQL, the effort to be invested into the qualification of a tool varies considerably: the lowest level TQL-5 just requires (**R1**) configuration management for the tool software, (**R2**) documentation, (**R3**) verification, and (**R4**) validation of the tool operational requirements, and (**R5**) a test suite showing that these requirements have been adequately implemented and deployed in the production environment where the tool is used. In contrast to that, the highest level TQL-1 requires (**R6**) a fully documented software life cycle and (**R7**) verification activities corresponding to those applicable to target system software development for the highest design assurance level (DAL) A [34, 4], where software errors may lead to catastrophic consequences for the aircraft and its passengers [7, Annex A].

With rigour and detail decreasing from TQL-1 to TQL-5, (R1) to (R5) can include the following of certain more detailed development standards. For example, (R5) may include integration tests to guarantee the correct execution of the tool's object code in the run-time environment. (R6) can imply the documentation of tool requirements and the tool architecture (e.g., specifications at function or method level). Moreover, (R7) requires the verification of a tool's behaviour against these requirements and the tool's robustness in error situations, so that it is ensured that run-time errors are detected. The results from (R7) need to be checked for correctness and completeness. At the source code level, algorithmic correctness might be shown by formally verifying the algorithms used by the tool.

Code generators producing C-code from control models, for example, need to be qualified according to TQL-1, if they generate DAL-A code, and no additional verification measures checking the consistency between code and model are applied. In contrast to this, model checkers and proof assistants automating a single step of the verification process are classified according to Criteria 3. Therefore, they need to be qualified only according to TQL-5, regardless of the criticality of the target system [4, Table 12.1].

The situation is more subtle, if Criteria 2 apply. Suppose, for example, that a code verification tool has proven that no array boundary violations can occur in a software function. If this verification result is then used to conclude that the robustness tests concerning illegal array index values provided as function call parameters can be omitted, the tool would be classified according to Criteria 2, and this would result in TQL-4 if the target system code is associated with DAL-A or B. The qualification effort needed for TQL-4 is considerably higher than that for TQL-5. It requires, in particular, an extensively documented software life cycle which is hardly ever available for tools developed over decades in academic communities.

In summary, depending on the extent to which verification tools, such as model checkers or proof assistants, *replace manual verification or testing*, they can either be subjected to just TQL-5 or, for example, for airborne DAL-A software, even to TQL-4 if they automate critical parts of the V&V process.

The EN 50128 standard for railway applications [5] mandates a scheme similar to DO-178C, also with three tool categories, T1, T2, and T3. The integrity level of the target system is, however, left implicit. According to the supplement EN 50126 [8], T3 with most rigorous implications on qualification applies to tools, such as code generators and compilers, that can insert errors into train system components. T2 and T1 represent less rigorous requirements (akin to TQL-4 and 5) to be followed, for example, by tools unable to insert faults but prone to overlook faults (e.g., verification tools) and tools neither automatically inserting nor detecting faults (e.g., editing tools).

It should be emphasised that tool qualification is not a once-and-for-all activity [4, Sec. 12.2.1]: the standards stress that the qualification needs to be re-assessed with every new product development undertaking, because different target system characteristics may require different tool properties. The product-specific tool qualification, however, can usually rely on reusable tool components applicable to every target system type, so that *tool qualification kits* can be prepared as templates facilitating the target system-specific qualification.

Tool qualification according to these requirements cannot guarantee complete correctness of a tool. This, however, is not the intention of the standards defining them. Instead, following the specified tool qualification process guarantees *accountability* and *liability*: In case of an undetected tool malfunction leading to a severe failure in the target system, the artefacts generated during tool qualification allow one to determine whether the tool has not been adequately verified. If this is the case, the system developers having applied and qualified the tool are liable for the consequences

of the target system failure. Conversely, if tool qualification has been comprehensively performed according to the requirements of the applicable standard, the developers avoid liability by arguing that all measures that can be "reasonably expected" to ensure tool correctness have been performed.

3 What Has Been Done?

Following the SWOT analysis guidelines [35], we present some *Strengths* (Sec. 3.1), including achievements in the qualification of proof assistants, checkers, and generators. Likewise, we summarise *Weaknesses* (Sec. 3.2) known from the literature and from our academic and industrial experience. Finally, we indicate *Opportunities* (Sec. 3.3) and *Threats* (Sec. 3.4) suggesting new research areas and possibilities for enhancing existing work. Below, we identify each SWOT item with its initial—**S**, **W**, **O**, or **T**—and a sequence number.

3.1 Strengths

One can observe that the tools in the considered categories (e.g., checking, deduction, compilation) have become quite powerful and some of these tools are increasingly easy to use, not only from the viewpoints of their developers but also suggested by expert users in academia [36] and industry [37]. Hence, this section addresses the question:

What are the key achievements in the qualification of proof assistants, checkers, and generators?

Regarding the qualification of proof assistants, (S1) proof terms or objects, capturing basic inferences constituting a mechanised proof, can be exported [38, 39] to enable the checking of proofs performed in them by additional proof checkers [40, 41, 42, 43, 44]. (S2) Some assistants even allow one to reason about their underlying logics or implementations, enabling self-formalisation efforts in order to raise confidence in the their correctness [45, 46, 41]. Such efforts have led, for example, to the HOL Light proof assistant, mechanically verified down to the level of machine code [47, 48].

In the context of the qualification of checkers, **(S3)** Bendisposto et al. [49] report on the effort of validating the ProB model checker as an EN 50128 class T2 tool. Bendisposto et al.'s qualification argument includes a provenin-use claim, extensive coverage-oriented testing, static analysis, and cross-checking with another B parser (Atelier B bcomp [50]). In contrast, Wagner et al. [12] argue that it is sufficient for a model checker (e.g., KIND 2 [26]) to achieve TQL-5 (similar to T1). After deriving 111 requirements for that checker and further requirements for a proof checker, they perform corresponding (language coverage) tests and peer reviews according to DO-330 [7] (cf. Sec. 2).

Wagner et al. demonstrate the (S4) independent verification of KIND 2's output with a simple-to-qualify proof checker (i.e., Check-It based on the LFSC⁶ [51]) based on proof certificates of an SMT solver. In their study, the CVC4 solver [52] provides a *k*-induction-based certificate that unsafe states cannot be reached [53]. Wagner et al. then apply their TQL-5 argument to Check-It instead. For reachability in timed automata, Wimmer and von Mutius [54] show how proof certificates can be represented as compressed sets of abstracted symbolic states. In addition to certificates for the class of safety properties [53, 54], liveness certificates can also be generated in SAT-based linear temporal logic (LTL) model checkers [55] and for timed automata model checkers [56].

Furthermore, (S5) some core algorithms of a variety of model checkers have been verified using proof assistants, for example, PRISM's probabilistic computation tree logic checking algorithm [57], a UPPAAL-compatible formalisation of timed automata checking [58] together with a multi-tier qualification argument [19, Sec. 4.2], and a proof of the critical LTL-to-Büchi automaton translation [59], the three approaches using Isabelle/HOL.

Concerning the qualification of generators, (**S6**) the optimising, semantics-preserving C compiler *CompCert*, formally verified in Coq, has been successfully qualified for use in the development and certification according to IEC 60880 of a high-integrity control system in the nuclear power domain [60].

Underlining DO-178C's testing-driven qualification, (**S7**) Taft et al. [61, Sec. 5] suggest that TQL-1 for a Simulinkto-SPARK or -MISRA C generator can be achieved by integrated unit testing with sufficient coverage of the generator input language. Here, the language of Simulink models is covered by traversing its block grammar rules. The authors describe how input-language coverage can drive test requirements (i.e., input equivalence classes for code units of a generator), the creation of assertion checkers with oracles (i.e., outputs of the units expected for certain inputs to these units), and, ultimately, test coverage (i.e., sufficiently covering the input/output behaviour of each of the units).

(**S8**) The commercial ANSYS SCADE Suite creates code from SCADE models and offers a TQL-1-conforming tool qualification kit [62, 13]. Moreover, the generated object code can be verified against the generated C-code. Both

⁶Logical Framework with Side Conditions

tool qualification and object code validation rely on tests achieving Modified Condition/Decision Coverage (MC/DC). These suffice according to DO-178B to obtain TQL-1 for the tool and to verify the production object code for DAL-A. In other cases, (**S9**) code generation can be certified (in class T3) when using diverse implementations with code produced by at least two independent code generators and the resulting executables running on diverse hardware [63].

In summary, there are several efforts to reduce the code base responsible for proof construction to a minimal *trusted core* of verifiable routines. Examples include the concentration of KIND 2's qualification on the verification of CheckIt [12] as well as the focus of qualification efforts in Isabelle on its kernel, which is orientated towards the compact Logic-of-Computable-Functions (LCF) principle. On the side of code generators, Yuan et al. [64] follow a refinement-based approach (proving a forward simulation relation) to certify semantics-preserving C-code generation from a functional program previously security-verified in Coq. They establish confidence in their certification by a careful minimisation and validation of the trusted computing base. On the other side, the approaches to qualifying proof assistants and model checkers, desired to be more trustworthy [65], seem to focus on artefacts (e.g., validating the tool's result by an additional checker) rather than the tools themselves (e.g., testing the tool and verifying its algorithms).

3.2 Weaknesses

This section is centred around the question:

Why can tool qualification sometimes fail? What can go wrong when using a proof assistant, checker, or generator in an industrial certification setting?

(W1) Arguments based on the soundness of the underlying logic of a proof assistant may be invalidated if user-defined axioms are added [66, Sec. 3]. Some assistants allow suspension of certain proof obligations in their interactive modes [39] or by disabling guard conditions [67]. (W2) Moreover, proof checkers developed to check proofs of the same proof assistant they have been implemented and verified in, such as [41, 42], may not be sufficiently diverse to be applied in the independent checking of proofs. (W3) Importantly, if proof assistants are not qualified and their proofs are not checked by a sufficiently diverse⁷ proof checker—that is, if they constitute a single point of failure—then it may be necessary to fall back to a manual (independent) repetition of these proofs [68, Sec. 2].

(W4) For formal method applications, Taft et al. [61] state that "the co-development of formal specifications, proofs, and program code raises significant 'common mode' concerns". (W5) For model checkers, Wagner et al. [12] stress that common cause errors in an unqualified model checker and a qualified proof checker are easy to be overseen.

Concerning the qualification of generators, (W6) code generated from provably correct types (i.e., specifications of data structures and algorithms) may not be ensured to conform to those types if the generation mechanism is unqualified. This can be partially tackled by verified code generators such as [69] and [70] for Coq and Isabelle/HOL. (W7) Though conforming to the qualification requirements discussed in Sec. 2, the form of generator testing suggested in [61, Sec. 5], even if applied with MC/DC tests to achieve TQL-1 (see also S8), is not a valid replacement for rigorous proofs of tool correctness or equivalence checks of generated code against models.

Furthermore, we need to acknowledge that qualification is supposed to not only provide evidence for the correctness of the tools in isolation but also to show the suitability of these tools in regard to the requirements of the particular development and V&V projects. Appreciating the state of the art, however, it needs to be said, that (W8) the mentioned tools have largely been used in environments where certification is not required (e.g., in academic settings). Moreover, certification authorities have still rarely considered formal design and verification as key activities and their results as key artefacts in industrial assurance cases. It might thus be unsurprising that tool providers (e.g., research teams) have only given limited attention to a comprehensive industry-strength qualification.

It is important to note that tool qualification is not about assuring that the input to a proof assistant (i.e., a type and a theorem) or model checker (i.e., a model and a property) is actually valid. Nevertheless, the **(W9)** crucial issues of *translation* and *input validation* can have a tremendous impact on tool qualification.

As observed by Cofer [71, p. 24], the risks stemming from a *translation* required to use a proof assistant or model checker can in certain cases outweigh that tool's benefits, in comparison with the often less involved default translations.

Default translations reflect what might occur in any safety-critical development and usually encompass, for example, the necessary transformation of requirements into an implementation, the derivation of a test suite from these

⁷Some regulators using older versions of IEC 61508 were not even positive about the mitigation of a failure in a proof assistant (e.g., PVS) via the use of a different tool [68, Sec. 2].

requirements, the occasional translation of critical requirement fragments into useful but often free-style mathematical sketches. The *additional translation* required to use a proof assistant or model checker usually includes (i) a careful transcription and abstraction of the requirements and the domain description into a (*model*, *property*)-pair,⁸ and then (ii) an alignment of the verified model with, or a translation into, an implementation.

The aforementioned risks may thus include, for example, issues in understanding an additional model in another language or the costs of aligning that model with the existing descriptions by tracing backward through the corresponding translations and abstractions.

Concerning *input validation*, model checkers and proof assistants do not usually help one in figuring out how much a checked property or deduced theorem contributes to a proof obligation [66, Sec. 3.1.2]. Overall, translation- and input-related issues threaten the essential correctness assertions that (i) the (*model*, *property*)-pair reflects the domain and the intended requirements, and (ii) the specified property is strong enough to rule out undesirable models.

3.3 **Opportunities**

Based on previous studies [37, 36], we are confident that formal design and verification tools have good potential to improve the best practices in the development and assurance of control software. Here, we focus on the question:

Building on the strengths (Sec. 3.1), which opportunities are there for getting these tools qualified?

(**O1**) A qualification of the Isabelle proof assistant would lead to a significant number of software products verified by means of Isabelle to become certifiable. To provide an example, the seL4 case study applies Isabelle comprehensively to verify an operating system kernel [72], focusing in particular on security properties such as the enforcement of strong isolation between applications run on top of it, intending it to serve as the foundation of trustworthy systems [73]. seL4 is not yet certified [74]: As suggested in [9, Talk 3.1], tool qualification of Isabelle is the essential prerequisite to obtain certification credit for the correctness proofs developed for the kernel. Thus, a *reference qualification* for Isabelle, yet missing, could increase the motivation to qualify other proof assistants as well.

(**O2**) Regarding the design of proof assistants, the simplicity of the LCF approach underlying many modern proof assistants could be further exploited [66, 18]. This approach prescribes an inference kernel as the sole soundness-critical component, performing a limited number of primitive inference rules and operations to expand theories. From studying this kernel, we could learn how to qualify the cores of proof assistants by sufficient unit testing and turn them into trusted components.

(O3) Accepting potential fallbacks to manual proofs (W3) as highlighted in [68, Sec. 2], tool-informed manual proofs could still be an improvement over entirely manually performed proofs. In low-criticality contexts, we could then think about using proof assistants for that purpose even without qualification. (O4) Moreover, the qualification of proof assistants could benefit from extended guidance on how to use them to address the requirements of specific standards. For example, Bertot et al. [75, 44, Sec. 3.3] provide recommendations on how to employ Coq in the context of the Common Criteria (CC) standard (e.g., a restricted configuration of Coq when used for the demanding CC evaluation assurance levels EAL 6+/7).

Concerning the qualification of checkers and in the context of TQL-5 qualification, (**O5**) Wagner et al. [12] spread optimism by stating that "the guidance of DO-330 does not require any activities that are especially difficult or costly for qualification of a model checker." So, there is an opportunity for other model checkers to follow their approach.

In regard to the qualification of generators, (**O6**) certified compilers, such as CompCert, can confidently raise the level of abstraction of proof efforts, given the supported language fragment fits the practical needs. For instance, if it can be certified that a compiler preserves the semantics of a given program between source code and compiled code, then it becomes possible to analyse the source code and soundly transfer the results to the compiled code. This reduces the need for additional analysis of the compiled code and costly reviews on whether the compiled code corresponds to the source code [76]. A nearby opportunity would be to try to achieve TQL-1 of the trusted computing base of code generators using semantically-integrated refinement proofs as, for example, shown in [64]. An alternative solution to dealing with simpler generators could be to use a complete testing approach [17].

(07) Here, the code generation capabilities of proof assistants constitute a particularly motivating target for tool qualification. If they could be qualified to generate code from specifications developed in proof assistants so that the generated code retains properties established over the specifications via the proof assistant, this would foster the qualification of a large number of already existing tools developed via proof assistants, including LTL model

⁸For the sake of simplicity, we consider a (model, property)-pair for a model checker to be the dual of a (type, theorem)-pair for a proof assistant.

checkers [77], SAT solvers [15], and compilers [60, 70]. For instance, Bendisposto et al.'s [49] aim is to achieve a validation of the ProB generation facilities as an EN 50128 class T3 tool.

Following the idea of artefact-based qualification, (**O8**) we could improve the support of redoing interactive proofs or automatic checks in diverse proof assistants [78, 79, 80] and model checkers.

We could further (**O9**) mitigate the risk of a poor cost-benefit ratio from using formal design and verification tools. Regarding the required *additional translations* (W9), two types of benefits are important to be considered: a short-term benefit from early fault detection and a long-term benefit from having reusable domain models and theories. These benefits could be strengthened by methodologies aiding in the translation between requirements or assurance documents and the input languages of model checkers and proof assistants. See the two recent examples in [81, 82].

Part of the *input validation* problem has been addressed by *coverage metrics and sanity checks* [83], for example, the automatic identification of easily breakable preconditions leading to vacuously true implications, *warnings about problematic proof directives* (cf. W1), as reported, for example, to Isabelle users [39], and *model-in-the-loop testing*, for example, to make sure that the model captures domain phenomena in such a way that useful properties can be specified. Along with these measures, tool-specific debugging techniques can help us to increase the confidence and trust in formal design and verification tools. Furthermore, informal-to-formal translations and input validation could be controlled with frameworks such as Isabelle/DOF [84], demonstrating direct support of requirements traceability within proof environments.

Finally, (O10) we expect that proof assistants, checkers, and generators will increasingly be used in high-integrity system assurance, so that their qualification will gain increasing attention.

3.4 Threats

In this section, we focus on the following question:

Which obstacles to the qualification of formal design and verification tools are to be expected?

Weaknesses refer to problems observed or known with the existing state of the art. Threats focus on challenges to cope with and risks to reoccur or to be faced, in particular, when trying to pursue the opportunities listed in Sec. 3.3.

Concerning the qualification of proof assistants, two threats may limit the benefits of tool qualification guidelines (O4): (**T1**) First, the code bases of even the cores of Isabelle and Coq exceed 10,000 lines of code, and future versions of both tools could exhibit soundness flaws, as detailed for earlier versions in [66, Sec. 3.2]. (**T2**) Moreover, components outside of these cores may still need to be qualified to ensure that a proof assistant checks the proofs the user expects it to check. This qualification, in particular, includes printing and parsing capabilities, where many assistants allow (i) syntactic manipulations that could mislead users about what theorems they are actually proving [85] or (ii) to print theorems in ways that, when parsed again, result in different theorems. This aspect has been addressed explicitly only by a few tools, such as HOL Zero [86], which has been developed for this very purpose.

(T3) The well-structured qualification case in [19, Sec. 4.2] for a timed automaton checker code base contains the clause "it is widely accepted within the community that Isabelle/HOL only admits valid theorems (at least on the user level)." As a *proven-in-use* argument, even a well-justified observation like this needs to be underpinned by detailed evidence for a certification authority, as indicated by Adams [66, Sec. 3.1.2]: According to the applicable standards, a proven-in-use argument requires a comprehensive documentation of its *service history* [4, 12.3.4]. This documentation comprises a complete history of configuration management and product failure tracking. Such a history will not usually be available for verification tools developed in the academic communities. Moreover, obtaining certification credit for such a tool in a particular V&V campaign requires that the tool has been used extensively in the past and in domains that are similar to the actual system to be verified. Providing a history of successful use will hardly be feasible for tools developed in academia, since the tool applications—though extensive—are usually on examples stemming from open-source developments or significant academic challenges that were not directly based on V&V campaigns for safety-critical industrial systems.

Regarding the qualification of checkers, in particular when pursuing TQL-5 (O5) or proof replay (O8), (T4) it can be challenging to explain the indirect approach to assurance with additional proof checkers to certification authorities [12]. (T5) The effort of qualifying a proof checker for TQL-5 can, although not difficult (W5), be nearly as high as the effort for qualifying the model checker itself for TQL-5 [12]. Similar efforts could raise two questions: Depending on the capabilities of the proof checker, could the latter be used as the model checker in the first place? And, relating to (O3), if a tool is only used to double-check manual proofs, do we need tool qualification at all? On the one hand, Wagner et al. [12, Sec. 6] discuss how TQL-4 might require a much higher effort than TQL-5. On the other hand, it remains to be seen how often TQL-4 applies to tools for verifying DAL-A code if these tools do not replace large parts of the

DAL-A verification procedure. (T6) In addition, the format and size of proof objects and certificates for large systems could make their validation difficult.

Regarding the qualification of generators, (**T7**) as indicated by Yang et al. [87], even compilers having undergone serious certification efforts [14] might still have critical flaws, threatening the preservation of the semantics (O6). Hence, it is crucial that a tool qualification case makes explicit the exact circumstances and the code for which certification credit was obtained.

The following threats pertain to any kind of tool. (**T8**) As an organisational aspect and human factor, there could be a lack of trust on the side of certification assessors into V&V approaches based on formal methods and supporting tools. This lack of trust could hinder qualification. (**T9**) Moreover, tools from the three categories discussed in this article face the usual risk of their code bases getting too complex to be qualified for a targeted TQL. Similarly, discontinuities in tool maintenance (e.g., missing or new developers) could hinder re-qualification. Additionally, changes of both, tools or standards (e.g., DO-178C), could trigger costly re-qualification.

4 Our Viewpoint: What Could Be Done Next?

This section is an attempt to suggest answers to the following three questions:

How can we handle the weaknesses (Sec. 3.2)? How can we use the opportunities (Sec. 3.3) to the maximum benefit? What could we do to mitigate the threats (Sec. 3.4)?

Our *viewpoint* is that a repetitive and exhaustive formal verification of tools (e.g., Isabelle beyond its core or the collection of PRISM's algorithms) is likely to remain difficult if not infeasible, both economically and organisationally. Some tools have just grown too complex and might naturally remain black-boxes, even to their expert users.

In the context of artefact-based tool qualification, our SWOT analysis in Sec. 3 and our experiences with the state of the art suggest several tasks to be considered for a successful qualification of formal design and verification tools.

(V1) Based on the widely accepted *de Bruijn* criterion for checkers [88]⁹ and on Pnueli's idea of generator output validation [89, 90], we advocate the enhancement or replacement of tool verification by artefact-based tool qualification. Successfully qualified artefacts could replace the qualification of the tools producing these artefacts. The conditions stated in DO-178C [4, Sec. 12.2] will then be imposed on the artefact validation tools, ranging from utilities for computing "checksums" (and other properties indicative of the validity of the output) to preferably simple proof checkers or proof certificate analysers certifying the validity of the output. This form of qualification is not only increasingly supported by the research community, as discussed in Sec. 3, but would also address the principle of *Explainability* in the Manifesto for Applicable Formal Methods [91].

(V2) The comprehensive testing according to DO-178B/C for the qualification of a complete model checker or proof assistant might also be infeasible. As discussed in Sec. 3.4, a proven-in-use argument will not be acceptable from the standards' perspective. Hence, following the suggestion in [4, Sec. 12.3.2.4/5], we again stress the achievement of *redundancy* of proof results through the use of *diverse tools*, as also suggested by Fantechi and Gnesi [92] and successfully applied, for example, by Parillaud et al. [93].

(V3) More specifically, trust in tools could be increased by improving proof certificates. In particular, the application of diverse tools to the same problems could be facilitated by developing and adopting shared formats analogous to the TPTP¹⁰ family of exchange formats [94]. At the conceptual level, Chihani et al. [11] characterise core elements of proof certificates to be defined and shared among provers and solvers. For model checkers, Beyer et al. [95] explore a graph-based shared format for correctness witnesses¹¹ produced by these tools.

(V4) We think that, for further industries to gain trust in the mentioned tools, early *qualification cases* should be open-source rather than undisclosed and only shown to certification authorities (e.g., the undisclosed case of the DO-178C/TQL-1-certified SCADE code generator [13]). Generally, we should build up lasting trust in new technologies of any kind by making reference cases open-source (e.g., the seL4 proofs [73, 74])

(V5) Specific guidance for the *auditing of mechanised proofs* could be developed in order to simplify and regulate this process. This could include both tool-specific guidance such as the need to check for the suspension of proof

⁹"A mathematical assistant satisfying the possibility of independent checking by a small program is said to satisfy the de Bruijn criterion." [88]

¹⁰Thousands of Problems for Theorem Provers

¹¹In model checking, *counterexamples* witness the violation of a specification (e.g., a temporal logic formula) and *witnesses* are models of a specification. In contrast, *correctness witnesses* compactly represent complete sets of witnesses, that is, invariants certifying safety of a model.

obligations or disabling of guards, which may affect soundness, as well as general guidelines for the examination of theorems and their dependencies as suggested in [66, Sec. 3]. Analogously, guidelines for the use of proof assistants such as [44] for Coq could be extended to other tools and checked against in proof audits. Considering the possibility of tool-informed manual proofs, proof auditing could thus foster the use of mechanised proofs in qualification efforts instead of only relying on proof assistants themselves. Hence, we would like to advocate tool-informed over fully manual proofs. Human-friendly proof languages such as Isabelle/Isar [39] can enable such a form of interaction.¹² Generally, an improved tool support for input validation of and translation between artefacts (W9, O9) would devitalise the occasional "garbage-in/garbage-out" argument against formal design and verification tools.

(V6) To prevent us from constructing circular qualification arguments,¹³ an alternative would be to strive for sufficiently diverse and independent proof checkers, for example, one for Isabelle/HOL verified and implemented in Coq and vice versa.

(V7) Given a verified model checker (e.g., as shown in [58, 57]) or a simple proof checker, and apart from the issues of an additional translation (as described in Sec. 3.2 and to be addressed by, e.g., model-in-the-loop simulation), we think that there are no hard-to-overcome obstacles to getting a model checker qualified up to TQL-5.

Moreover, (V8) proof certificates could complement certified compilers with certifying compilers [65].

In a survey of benefits and limits of tool qualification according to DO-330 and DO-178, Ibrahim and Durak [96] allude to the trend "of not qualifying development tools [but rather] qualifying verification tools". To avoid costly re-qualification (T9), they suggest (**V9**) the definition of use cases within which a tool can be transferred from one project to another. This idea could perhaps be adopted in the qualification of formal verification and design tools.

5 Summary

In this work, we have assessed some key issues for getting proof assistants, checkers, and generators qualified according to the applicable standards. Table 1 provides a summary of our findings from the SWOT analysis as well as the suggestions accompanied with our viewpoint. We hope that our discussion contributes to the next generation of qualified formal design and verification tools in order for them to be used in the development and certification of control software used in cyber-physical systems, such as intelligent robots and autonomous systems [1]. Finally, a wider assessment of the field under discussion could include (i) tools for abstract interpretation, further solvers and checkers, and (ii) proof-carrying code could be similarly discussed as a form of proof certificates.

Acknowledgements

Jan Peleska has been partially funded by the German Ministry of Economics, Grant Agreement 20X1908E. We would like to thank Jörg Brauer for helpful feedback and highlighting the issue of tool suitability as well as Michael Leuschel for his kind suggestions concerning EN 50128-based tool qualification.

References

- [1] Matt Webster, Neil Cameron, Michael Fisher, and Mike Jump. Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. *Journal of Aerospace Information Systems*, 11(5):258–279, 2014.
- [2] Ricardo Bedin França, Denis Favre-Felix, Xavier Leroy, Marc Pantel, and Jean Souyris. Towards Formally Verified Optimizing Compilation in Flight Control Software. In *Predictability and Performance in Embedded Systems*, volume 18 of *OpenAccess Series in Informatics*, pages 59–68, Grenoble, France, March 2011. Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik.
- [3] IEC 61508. Functional safety of electric/electronic/programmable electronic safety-related systems. International Electrotechnical Commission, 2006.
- [4] RTCA SC-205/EUROCAE WG-71. *RTCA DO-178C Software Considerations in Airborne Systems and Equipment Certification*. 1140 Connecticut Avenue, N.W., Suite 1020, Washington, D.C. 20036, December 2011.

¹²In a recent comment on improving the interoperability between proof assistants, Paulson stresses the role of a humanfriendly proof language in the porting of proofs at a conceptual level rather than at the low level of mechanical inferences. See https://lawrencecpaulson.github.io/2022/09/14/Libraries.html.

¹³Verifying a proof checker in the very proof assistant whose proofs it is to check may raise issues about circular verification.

Strengths (Sec. 3.1)	Weaknesses (Sec. 3.2)
S1. Support for independent check of deductive proof objects	W1. Invalidation of theorems by user-defined axioms
S2. Introspective reasoning	W2. Mutual dependencies between checkers reduce diversity
S3. TQL-5/T2 qualification guidance for model checkers	W3. Undesired manual repetition of proof work
S4. Safety & liveness certificates for model checkers &	W4. Common errors in co-development of specs & programs
solvers	W5. Common errors in tools and their artefact checkers
S5. Verified model checking algorithms	W6. Unqualified generators jeopardise artefact correctness
S6. IEC 60880 qualification guidance for verified compilers	W7. Standards incentivise incomplete testing of generators
S7. TQL-1 qualification guidance for model transformers	W8. Established tools not yet considered intensively by in-
S8. TQL-1 certification for DAL-A code generators	dustry
S9. T3 certification of diverse code generators	W9. Effort for validation of (translated) inputs
Opportunities (Sec. 3.3)	Threats (Sec. 3.4)
O1. Motivation from enabling the use of proven components	T1. Size and complexity of code bases, even restricted to
O2. LCF approach with testable single soundness-critical	cores
core	T2. Critical components outside of the core
O3. Guidance in manual proofs as fallback	T3. Missing comprehensive documentation of service history
O4. Provision of guidelines for addressing specific standards	T4. Challenges in explaining indirect approaches
O5. Qualification of model checkers for TQL-5	T5. Questionable necessity of lower levels if qualification
O6. Semantics preserving compilers	T6. Size and complexity of proof objects for replay
O7. Code generation capabilities of proof assistants	T7. Undetected critical flaws in tool chains (e.g. compilers)
O8. Proof-replay with diverse proof assistants	T8. Lack of trust of certification assessors
O9. Improve support for validation of (translated) inputs	T9. Discontinuities in tool maintenance
O10. Expected increase in attention to qualification	
Qualification Requirements from the Standards (Sec. 2)	Our Viewpoint and Suggestions (Sec. 4)
R1. Tool configuration management	V1. Put stronger focus on artefact-based tool qualification
R2. Documentation of tool operational requirements	V2. Foster redundant proof results using diverse tools
R3. Verification of tool operational requirements	V3. Further establish shared proof interchange formats
R4. Validation of tool operational requirements	V4. Disclose qualification cases & communicate them
R5. Tool integration testing with an adequate test suite	V5. Improve/establish tool usage and proof auditing guide-
R6. Comprehensive tool life cycle documentation	lines
R7. Evidence of DAL-A-style verification activities	V6. Further explore mutual proof object certification
	V7. Aim at TQL-5 qualification of established model checkers
	V8. Complement certified compilers with proof certificates
	V9. Define qualification-preserving tool use cases

Table 1: Summary of our SWOT analysis of tool qualification in formal methods applications

- [5] CENELEC. EN 50128:2011 Railway applications Communication, signalling and processing systems Software for railway control and protection systems. 2011.
- [6] ISO/DIS 26262-8. Road vehicles functional safety Part 8: Supporting processes, 2009.
- [7] RTCA SC-205/EUROCAE WG-71. *RTCA DO-330 Software Tool Qualification Considerations*. 1140 Connecticut Avenue, N.W., Suite 1020, Washington, D.C. 20036, December 2011.
- [8] CENELEC. EN 50126-1:2018 Railway Applications The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 1: Generic RAMS Process. 2018.

- [9] Darren D. Cofer, Gerwin Klein, Konrad Slind, and Virginie Wiels. Qualification of formal methods tools (dagstuhl seminar 15182). *Dagstuhl Reports*, 5(4):142–159, 2015.
- [10] Lucas G. Wagner, Darren Cofer, Konrad Slind, Cesare Tinelli, and Alain Mebsout. Formal methods tool qualification. Technical Report NASA/CR-2017-219371, NASA, 2017.
- [11] Zakaria Chihani, Dale Miller, and Fabien Renaud. Foundational proof certificates in first-order logic. In Maria Paola Bonacina, editor, *Automated Deduction*, volume 7898 of *LNCS*, pages 162–177. Springer, 2013.
- [12] Lucas Wagner, Alain Mebsout, Cesare Tinelli, Darren Cofer, and Konrad Slind. Qualification of a model checker for avionics software verification. In NFM, volume 10227 of LNCS, pages 404–419. Springer, Cham, 2017.
- [13] ANSYS Esterel Technologies. SCADE suite KCG 6.4 DO-178C certification kits technical data sheet. Technical report, ANSYS, 2013.
- [14] Daniel Kästner, Xavier Leroy, Sandrine Blazy, Bernhard Schommer, Michael Schmidt, and Christian Ferdinand. Closing the Gap – The Formally Verified Optimizing Compiler CompCert. In SSS'17: Safety-critical Systems Symposium 2017, Developments in System Safety Engineering, pages 163–180, Bristol, United Kingdom, February 2017. CreateSpace.
- [15] Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. *J. Autom. Reason.*, 61(1-4):333–365, 2018.
- [16] Jörg Brauer, Jan Peleska, and Uwe Schulze. Efficient and trustworthy tool qualification for model-based testing tools. In *Testing Software and Systems*, volume 7641 of *LNPSE*, pages 8–23. Springer, Berlin Heidelberg, 2012.
- [17] Mario Gleirscher, Lukas Plecher, and Jan Peleska. Sound development of supervisors. Working paper, U Bremen, 2022.
- [18] Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric, and Zachary Tatlock. QED at large: A survey of engineering of formally verified software. *CoRR*, abs/2003.06458, 2020.
- [19] Simon Wimmer. Munta: A verified model checker for timed automata. In FORMATS, volume 11750 of LNTCS, pages 236–243. Springer, 2019.
- [20] Yves Bertot and Pierre Castéran. Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
- [21] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. Isabelle/HOL A Proof Assistant for Higher-Order Logic, volume 2283 of Lecture Notes in Computer Science. Springer, 2002.
- [22] André Platzer. KeYmaera X tutorial. Technical report, Computer Science Department, Carnegie Mellon University, 2022.
- [23] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Automated Deduction - CADE-25*, volume 9195 of *LNCS*, pages 378–388. Springer, Cham, 2015.
- [24] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In Automated Deduction (CADE-11), volume 607 of LNCS, pages 748–752. Springer, 1992.
- [25] Thomas Gibson-Robinson, Philip J. Armstrong, Alexandre Boulgakov, and A. W. Roscoe. FDR3: a parallel refinement checker for CSP. Int. J. Softw. Tools Technol. Transf., 18(2):149–167, 2016.
- [26] Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. The KIND 2 model checker. In Computer Aided Verification, pages 510–517. Springer, 2016.
- [27] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *Computer Aided Verification*, volume 8559 of *LNCS*, pages 334–342. Springer, 2014.
- [28] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification (CAV)*, 23rd Int. Conf., volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- [29] Michael Leuschel and Michael Butler. ProB: A model checker for b. In *Formal Methods*, pages 855–874. Springer, Berlin Heidelberg, 2003.
- [30] Pierre Deransart, AbdelAli Ed-Dbali, and Laurent Cervoni. *Prolog the standard: reference manual.* Springer, 1996.
- [31] Gerard J. Holzmann. The model checker SPIN. IEEE Trans. Software Eng., 23(5):279–295, 1997.

- [32] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on UPPAAL. In *SFM*, volume 3185 of *LNCS*, pages 200–236, Berlin, Heidelberg, 2004. Springer.
- [33] John Barnes. High Integrity Software. Addison Wesley, London, 2003.
- [34] SAE/ARP-4754. Certification considerations for highly-integrated or complex aircraft systems. Standard, Society of Automotive Engineers (SAE) / Aerospace Recommended Practice (ARP), 1996.
- [35] Nigel Piercy and William Giles. Making SWOT analysis work. *Marketing Intelligence & Planning*, 7(5/6):5–7, 1989.
- [36] Hubert Garavel, Maurice H. ter Beek, and Jaco van de Pol. The 2020 expert survey on formal methods. In *Formal Methods for Industrial Critical Systems*, pages 3–69. Springer, 2020.
- [37] Mario Gleirscher and Diego Marmsoler. Formal methods in dependable systems engineering: A survey of professionals from Europe and North America. *Empirical Software Engineering*, 25(6):4473–4546, 2020. Presented at the ESEC/FSE'21 journal first track.
- [38] Stefan Berghofer and Tobias Nipkow. Proof terms for simply typed higher order logic. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher Order Logics*, 13th International Conference, TPHOLs 2000, Portland, Oregon, USA, August 14-18, 2000, Proceedings, volume 1869 of LNCS, pages 38–52. Springer, 2000.
- [39] Makarius Wenzel. The Isabelle/Isar reference manual. https://isabelle.in.tum.de/dist/doc/isar-ref.pdf, December 2021.
- [40] Wai Wong. Validation of HOL proofs by proof checking. Formal Methods Syst. Des., 14(2):193–212, 1999.
- [41] Tobias Nipkow and Simon Roßkopf. Isabelle's metalogic: Formalization and proof checker. In André Platzer and Geoff Sutcliffe, editors, Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of LNCS, pages 93–110. Springer, 2021.
- [42] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. Proc. ACM Program. Lang., 4(POPL):8:1–8:28, 2020.
- [43] Oskar Abrahamsson. A verified proof checker for higher-order logic. J. Log. Algebraic Methods Program., 112:100530, 2020.
- [44] ANSSI-INRIA. Requirements on the use of Coq in the context of Common Criteria evaluations. J. Log. Algebraic Methods Program., 2021.
- [45] John Harrison. Towards self-verification of HOL light. In Ulrich Furbach and Natarajan Shankar, editors, Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, volume 4130 of LNCS, pages 177–191. Springer, 2006.
- [46] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. Self-formalisation of higher-order logic semantics, soundness, and a verified implementation. J. Autom. Reason., 56(3):221–259, 2016.
- [47] Oskar Abrahamsson, Magnus O. Myreen, Ramana Kumar, and Thomas Sewell. Candle: A verified implementation of HOL light. In June Andronick and Leonardo de Moura, editors, 13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel, volume 237 of LIPIcs, pages 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [48] Magnus O. Myreen. The CakeML project's quest for ever stronger correctness theorems. In Liron Cohen and Cezary Kaliszyk, editors, *Interactive Theorem Proving (ITP), 12th Int Conf*, volume 193 of *LIPIcs*, pages 1:1– 1:10. Schloss Dagstuhl - Leibniz-Zentrum f
 ür Informatik, 2021.
- [49] Jens Bendisposto, Sebastian Krings, and Michael Leuschel. Who watches the watchers: Validating the ProB validation tool. In C. Dubois, D. Giannakopoulou, and D. Mry, editors, *F-IDE*, volume 149 of *EPTCS*, pages 16–29, 2014.
- [50] ClearSy. Atelier B: User and Reference Manuals. Aix-en-Provence, France, 2009.
- [51] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. SMT proof checking using a logical framework. *Form Method Syst Des*, 42(1):91–118, 2012.
- [52] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, pages 171–177. Springer, 2011.
- [53] Alain Mebsout and Cesare Tinelli. Proof certificates for SMT-based model checkers for infinite-state systems. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 117–124. IEEE, 2016.
- [54] Simon Wimmer and Joshua von Mutius. Verified certification of reachability checking for timed automata. In *TACAS*, volume 12078 of *LNCS*, pages 425–443. Springer, 2020.

- [55] Alberto Griggio, Marco Roveri, and Stefano Tonetta. Certifying proofs for SAT-based model checking. *FMSD*, 57(2):178–210, 2021.
- [56] Simon Wimmer, Frédéric Herbreteau, and Jaco van de Pol. Certifying emptiness of timed Büchi automata. In *FORMATS*, volume 12288 of *LNCS*, pages 58–75. Springer, 2020.
- [57] Johannes Hölzl and Tobias Nipkow. Verifying pCTL model checking. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *LNTCS*, pages 347–361, Berlin, Heidelberg, 2012. Springer.
- [58] Simon Wimmer and Peter Lammich. Verified model checking of timed automata. In *TACAS*, volume 10805 of *LNTCS*, pages 61–78. Springer, 2018.
- [59] Alexander Schimpf, Stephan Merz, and Jan-Georg Smaus. Construction of Büchi automata for LTL model checking verified in isabelle/hol. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOL*, volume 5674 of *LNTCS*, pages 424–439, Berlin, Heidelberg, 2009. Springer.
- [60] Daniel Kästner, Jörg Barrho, Ulrich Wünsche, Marc Schlickling, Bernhard Schommer, Michael Schmidt, Christian Ferdinand, Xavier Leroy, and Sandrine Blazy. CompCert: Practical Experience on Integrating and Qualifying a Formally Verified Optimizing Compiler. In *ERTS2 2018 - 9th European Congress Embedded Real-Time Software and Systems*, pages 1–9, Toulouse, France, January 2018. 3AF, SEE, SIE.
- [61] S. Tucker Taft, Elie Richa, and Andres Toom. Building trust in a model-based automatic code generator. *ACM SIGAda Ada Letters*, 36(2):54–57, 2017.
- [62] Jean-Louis Colaço. An overview of scade, a synchronous language for safety-critical software (keynote). In REBLS 2020: 7th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems, Virtual Event, USA, November 16, 2020, page 1. ACM, 2020.
- [63] Michael Leuschel. Private communication, September 2022.
- [64] Shenghao Yuan, Frédéric Besson, Jean-Pierre Talpin, Samuel Hym, Koen Zandberg, and Emmanuel Baccelli. End-to-end mechanized proof of an eBPF virtual machine for micro-controllers. In *Computer Aided Verification*, pages 293–316. Springer, 2022.
- [65] George C. Necula and Peter Lee. The design and implementation of a certifying compiler. ACM SIGPLAN Notices, 33(5):333–344, 1998.
- [66] Mark Miles Adams. Proof auditing formalised mathematics. J. Formaliz. Reason., 9(1):3–32, 2016.
- [67] David Monniaux and Sylvain Boulmé. The trusted computing base of the CompCert verified compiler. In Ilya Sergey, editor, Programming Languages and Systems - 31st European Symposium on Programming (ESOP), held as part of ETAPS, volume 13240 of LNCS, pages 204–233. Springer, 2022.
- [68] Mark Lawford. Stupid tool tricks for smart model based design. In Sandrine Blazy and Marsha Chechik, editors, Verified Software. Theories, Tools, and Experiments - 8th International Conference, VSTTE 2016, Toronto, ON, Canada, July 17-18, 2016, Revised Selected Papers, volume 9971 of LNCS, pages 1–7, 2016.
- [69] Olivier Savary Bélanger, Matthew Z Weaver, and Andrew W Appel. Certified code generation from CPS to C. *Proc. ACM*, 2019.
- [70] Lars Hupel and Tobias Nipkow. A verified compiler from Isabelle/HOL to CakeML. In Amal Ahmed, editor, *Programming Languages and Systems*, volume 10801 of *LNCS*, pages 999–1026. Springer, 2018.
- [71] Darren Cofer. You keep using that word. ACM SIGLOG News, 2(4):17–25, October 2015.
- [72] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: formal verification of an OS kernel. In Jeanna Neefe Matthews and Thomas E. Anderson, editors, *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009*, pages 207–220. ACM, 2009.
- [73] Gerwin Klein, June Andronick, Matthew Fernandez, Ihor Kuz, Toby Murray, and Gernot Heiser. Formally verified software in the real world. *Commun. ACM*, 61(10):68–77, 2018.
- [74] Steven H. VanderLeest. Is formal proof of seL4 sufficient for avionics security? *IEEE Aerospace and Electronic Systems Magazine*, 33(2):16–21, 2018.
- [75] Yves Bertot, Maxime Dénés, Vincent Laporte, Arnaud Fontaine, and Thomas Letan. The use of Coq for Common Criteria evaluations. In *CoqPL 2020 The Sixth International Workshop on Coq for Programming Languages*, pages 1–3, New Orleans, United States, 2020.
- [76] Abderrahmane Brahmi, David Delmas, Mohamed Habib Essoussi, Famantanantsoa Randimbivololona, Abdellatif Atki, and Thomas Marie. Formalise to automate: deployment of a safe and cost-efficient process for avionics software. In 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), pages 1–11, Toulouse, France, January 2018.

- [77] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, volume 8044 of *LNCS*, pages 463–478. Springer, 2013.
- [78] Joe Hurd. The opentheory standard theory library. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings, volume 6617 of LNCS, pages 177–191. Springer, 2011.
- [79] Thibault Gauthier and Cezary Kaliszyk. Aligning concepts across proof assistant libraries. J. Symb. Comput., 90:89–123, 2019.
- [80] Chantal Keller and Benjamin Werner. Importing HOL light into Coq. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July* 11-14, 2010. Proceedings, volume 6172 of LNCS, pages 307–322. Springer, 2010.
- [81] Marie Farrell, Matt Luckcuck, Oisín Sheridan, and Rosemary Monahan. FRETting about requirements: Formalised requirements for an aircraft engine controller. In *REFSQ*, pages 96–111. Springer, 2022.
- [82] Simon Foster, Yakoub Nemouchi, Mario Gleirscher, Ran Wei, and Tim Kelly. Integration of formal proof into unified assurance cases with Isabelle/SACM. *Form Asp Comput*, 2021.
- [83] Hana Chockler, Orna Kupferman, and Moshe Vardi. Coverage metrics for formal verification. Int. J. Softw. Tools Technol. Trans., 8(4):373–386, 2006.
- [84] Achim D. Brucker and Burkhart Wolff. Isabelle/DOF: Design and implementation. In *SEFM*, volume 11724 of *LNTCS*, pages 275–292. Springer, 2019.
- [85] Freek Wiedijk. Pollack-inconsistency. Electron. Notes Theor. Comput. Sci., 285:85–100, 2012.
- [86] Mark Adams. HOL zero's solutions for pollack-inconsistency. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, volume 9807 of *LNCS*, pages 20–35. Springer, 2016.
- [87] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in C compilers. In Programming language design and implementation (PLDI), 32nd ACM SIGPLAN Conf, pages 283–294. ACM Press, 2011.
- [88] Henk Barendregt and Freek Wiedijk. The challenge of computer mathematics. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1835):2351–2375, September 2005.
- [89] Amir Pnueli, Ofer Strichman, and Michael Siegel. The code validation tool CVT: automatic verification of a compilation process. *Int. J. Softw. Tools Technol. Transf.*, 2(2):192–201, 1998.
- [90] Amir Pnueli, Ofer Strichman, and Michael Siegel. Translation validation: From SIGNAL to C. In Ernst-Rüdiger Olderog and Bernhard Steffen, editors, *Correct System Design, Recent Insight and Advances, (to Hans Langmaack on the occasion of his retirement from his professorship at the University of Kiel)*, volume 1710 of *Lecture Notes in Computer Science*, pages 231–255. Springer, 1999.
- [91] Mario Gleirscher, Jaco van de Pol, and James Woodcock. A manifesto for applicable formal methods. Working paper, University of Bremen and Aarhus University and University of York, 2021.
- [92] Alessandro Fantechi and Stefania Gnesi. On the adoption of model checking in safety-related software industry. In *SAFECOMP*, volume 6894 of *LNPSE*, pages 383–396. Springer, Berlin Heidelberg, 2011.
- [93] Camille Parillaud, Yoann Fonteneau, and Fabien Belmonte. Interlocking formal verification at alstom signalling. In Simon Collart Dutilleul, Thierry Lecomte, and Alexander B. Romanovsky, editors, *Reliability, Safety, and Security of Railway Systems*, volume 11495 of *Lecture Notes in Computer Science*, pages 215–225. Springer, 2019.
- [94] Geoff Sutcliffe. The TPTP problem library and associated infrastructure from CNF to th0, TPTP v6.4.0. J. *Autom. Reason.*, 59(4):483–502, 2017.
- [95] Dirk Beyer, Matthias Dangl, Daniel Dietsch, and Matthias Heizmann. Correctness witnesses: Exchanging verification results between verifiers. In *Foundations of Software Engineering (FSE), 24th ACM SIGSOFT Int Symp*, pages 326–337. ACM, 2016.
- [96] Mohamad Ibrahim and Umut Durak. State of the art in software tool qualification with DO-330: A survey. In S. Götz, L. Linsbauer, I. Schaefer, and A. Wortmann, editors, *Software Engineering (SE) Satellite Events*, volume 2814 of *LNI*, *CEUR Workshop Proceedings*, pages 1–22. 2021.