Building Confidence on Formal Verification Models

Pierre-Alain Bourdil, Eric Jenn, Silvano Dal Zilio

Building Confidence on Formal Verification Models

Pierre-Alain Bourdil and Eric Jenn
IRT Saint-Exupéry, Toulouse, France

Silvano Dal Zilio
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

I. Introduction

A problem hindering the adoption of formal methods in the industry is how to integrate the models and results used during formal verification with existing processes. Indeed, formal verification is a complex process involving multiple methods, models, level of formality, ... If we want to use formal verification results in an assurance case, it is therefore necessary to build confidence on this process.

The integration of formal methods raises particular problems like, for instance, with the construction of the verification models: a model may not preserve all properties of the system to be verified; it may only cover a subset of these properties; or it may be intractable. In practice, this means that the verification process involves a collection of models whose soundness (with the original system design, but also between each others) shall be justified. Furthermore, formal techniques are usually restricted in terms of the set of properties that can be checked. It is therefore necessary to justify (and trace back) that these restrictions are consistent with the hypotheses made about the system, its application and its environment.

This short abstract gives an overview of a methodology for building *verification arguments*, that is convincing arguments that a system design complies with a set of properties. More details can be found in [2], where we apply our method to a critical function of a small autonomous robot.

II. Verification Arguments and Claims

Our objective is to prove that a property, P, is met by a given system (model), M. In order to be as technology agnostic as possible, we use very broad definitions for these terms. A *model* is an abstraction of reality that gives a complete description of it under some *hypotheses* and for a certain purpose. In our approach, we apply successive abstractions from an initial design model—considered as the less abstract model—until formal verification can be applied. A *property* is a statement about a design intent expressed with respect to a model. The strong relationship between *hypotheses*, *model*, and *properties* is captured as a triplet $\langle H; M; P\rangle$ that must be read as: "the model M satisfies P under the set of hypothesis H". We call this triplet a *claim*, inspired by [5].

In short, a *verification argument* is a hierarchical collection of claims where "rules" are used to draw logical connections between claims. We start with an initial claim that makes as few abstractions and hypotheses as possible on the system design. Unfortunately, the initial formal model is usually not suitable for (automatic) formal verification. To tackle this issue, abstractions are mandatory, and each abstraction shall be captured in a new claim. Our methodology provides guidance to manage the resulting collection of claims, to justify the soundness of each abstraction step, and to collect the verification artifacts: source code, verification results, counter-examples, etc. Thus our approach favors multiple models with incremental abstractions rather than fewer models encompassing lots of abstractions. Indeed it is often easier to define (and justify) an abstraction once the system is well understood [1].

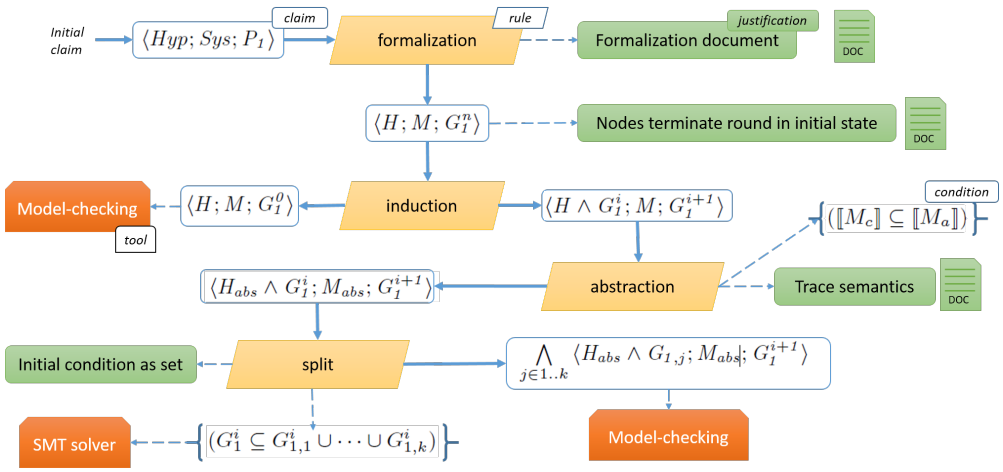
Ultimately—at the leaves of the argument—the validity of a claim should be checked using a verification tool, such as a model-checker. In this case, P may be a temporal-logic formula that needs to be checked on the *composition* of the hypotheses with the model. For instance, with a semantics based on "execution traces"—that associates a set of traces $[\![M]\!]$ to any model M—the claim $\langle H; M; P\rangle$ is valid when $[\![H]\!] \cap [\![M]\!] \subseteq [\![P]\!]$. It is also possible to admit a claim as a *fact* with sufficient justification from a system expert.

In all other cases, when it is not possible to prove a "leaf claim", we propose to apply deduction rules in order to decompose the problem into more manageable sub-claims. This approach to structure an argument using an (inference) tree or a set of deduction rules is quite common, see e.g. [5], [7], [8]. We define a *rule* as a schema linking a conclusion, with a set of premises and a list of conditions required to apply the rule. Premises and conclusion are claims. We also attach a *justification* to each application of a rule to show that its use is sound. While the leaf claims can serve as evidences within an assurance case, the inference tree can be used to build a separate *confidence argument* [4]. We give an example of verification argument (next page) that is a schematic version of the argument at http://projects.laas.fr/fiacre/examples/2016-twirtee/. This page links to the models and the justification reports for our use case.

We say that the application of a rule, a *rule instance*, is sound when its conclusion is valid whenever its premises are. We propose some predefined rules to support this process. A first example, (split), can be used to decompose a proof goal into two sub-claims with stronger hypotheses. This rule has a condition of application, which states that the constraints in H are covered by either H_1 or H_2 (the exact meaning of $H \subseteq H_1 \cup H_2$ depends on the choice of semantics for H).

$$\text{(split)}\ \frac{(H \subseteq H_1 \cup H_2) \quad \langle H_1; M; P\rangle \quad \langle H_2; M; P\rangle}{\langle H; M; P\rangle}$$

This rule is useful to decompose a set of initial conditions in two smaller subsets. The justification for rule (split) shall explain how to interpret the hypotheses in H as sets. It should



also constraints P. Indeed, while (split) is sound for safety properties, it is not necessarily so for liveness properties. The condition can itself be checked using other abstractions or a verification tool, for instance a SMT solver.

Another example, (abstraction), illustrates the use of abstractions during the verification process. In this rule, we consider two different models of the system—an abstract (M_a) and a concrete (M_c) one—both equipped with a trace semantics. This rule states that we can check properties on the abstract model if any execution trace of M_c is also a trace of M_a. In our use case, we use (abstraction) to check properties on a timed system by proving it on a simplified model, where timing constraints are omitted. In this particular case, we need to justify that properties in P depends only on the order of the events and not on their date. We can use many kinds of abstractions, some are purely automatic, like: predicate abstraction; Counter-Example Guided Abstraction Refinement; symmetry; ... while others are hand-crafted, such as cut-point, counter-abstraction, or data independence.

By construction, the verification objective (the root claim) is valid when all the rules instances in the argument are sound. We can back the soundness of rules like (split) and (abstraction) by reasoning on the semantics of the models. This is not always the case and this is one of the motivation for adding a justification to every rule instance. Next, we give two examples of "evidence-based rules", (scope) and (formalization), which are rules whose soundness is supported only by subject matter expertise.

$$\text{(scope)}\ \frac{\langle H_s; M_s; P \rangle}{\langle H; M; P \rangle} \qquad \text{(formalize)}\ \frac{\langle H; M; P \rangle}{\langle Hyp; Sys; Prop \rangle}$$

In general, only a subset of the design models and requirements need to be considered for a given verification objective. This is achieved by rule (scope), where M_s, H_s are obtained by simplifying M, H with respect to the property P. The soundness of this rule relies only on its justification, e.g. to explain why some parts of the design documents have no influence on the verification of P.

Another instance where formal reasoning is not enough to simplify a claim is related to the steps where we move from an informal to a formal model. Rule (formalization) states that H, M, P are formal interpretation of the corresponding design models; $Hyp, Sys, Prop$. This rule uses informal refinement steps, where High Level Requirements (HLR) are those given in the design models and Low Level Requirements (LLR) are formal model elements. The formal model is then produced from LLR. This is a classical way to support confidence in software correctness [10]. Evidence that LLR meet their HLR is given as the justification.

To conclude, we propose a methodology to build a structured verification argument in which claims are proved valid using "derivation rules", together with a rationale stating why a given rule can be rightfully applied. We have applied this methodology to prove a critical function of an autonomous rover that has been used as a test-bench for various engineering activities [2], [3]. For future work, we plan to provide patterns and tools to automate, or at least simplify, the management of claims and the selection of the appropriate verification tools. We are currently using Certware [9] to integrate the verification argument with the safety case of our case study. We also use the Evidential Tool Bus [6] to implement the rules and to check claims validity.

References

[1] A. Datta and V. Singhal, *Formal verification of a public-domain DDR2 controller design,* in *21st Int. Conference on VLSI Design*. IEEE, 2008.

[2] P.-A. Bourdil, S. Dal Zilio, E. Jenn, *Integrating Model Checking in an Industrial Verification Process,* Research Report LAAS 16115, 2016.

[3] M. Clabaut, N. Ge, N. Breton, E. Jenn, R. Delmas, and Y. Fonteneau, *Industrial grade model checking,* in *ERTSS*, 2016.

[4] R. Hawkins, T. Kelly, J. Knight, P. Graydon, *A new approach to creating clear safety arguments*, Advances in Systems Safety, 2011.

[5] J. Rushby, *On the interpretation of assurance case arguments,* in *2nd Int. Workshop on Argument for Agreement and Assurance (AAA)*, 2015.

[6] S. Cruanes, G. Hamon, S. Owre, and N. Shankar, *Tool integration with the evidential tool bus,* in *Proc. of VMCAI*. Springer, 2013.

[7] E. Denney and G. Pai, *Evidence arguments for using formal methods in software certification,* in Proc. of ISSREW, 2013.

[8] J. M. Rushby, *An evidential tool bus,* in *ICFEM*, LNCS vol. 3785, 2005.

[9] M. R. Barry, *CertWare: A workbench for safety case production and analysis,* in *IEEE Aerospace Conference*, 2011.

[10] C. M. Holloway, *Explicate'78: Uncovering the Implicit Assurance Case in DO-178C* in *SCSC Annual symposium*, 2015.