

A Methodology for Increasing the Efficiency and Coverage of Model Checking and its Application to Aerospace Systems

Orlando Ferrante
United Technologies Research Center

Eelco Scholte
UTC Aerospace Systems

Claudio Pinello, Alberto Ferrari, and Leonardo Mangeruca
United Technologies Research Center

Cong Liu
UTC Aerospace Systems

Christos Sofronis
United Technologies Research Center

ABSTRACT

Formal Methods, and in particular Model Checking, are seeing an increasing use in the Aerospace domain. In recent years, Formal Methods are now commonly used to verify systems and software and its correctness as a way to augment traditional methods relying on simulation and testing. Recent updates to the relevant Aerospace regulations (e.g. DO178C, DO331 and DO333) now have explicit provisions for utilization of models and formal methods. At the system level, Model Checking has seen more limited uses due to the complexity and abstractions needed. In this paper we propose several methods to increase the capability of applying Model Checking to complex Aerospace Systems. An aircraft electrical power system is used to highlight the methodology. Automated model-based methods such as Cone of Influence and Timer Abstractions are described. Results of those simplifications, in combination with traditional Assume-Guarantee approaches will be shown for the Electric Power System application.

CITATION: Ferrante, O., Scholte, E., Pinello, C., Ferrari, A. et al., "A Methodology for Increasing the Efficiency and Coverage of Model Checking and its Application to Aerospace Systems," *SAE Int. J. Aerosp.* 9(1):2016, doi:10.4271/2016-01-2053.

INTRODUCTION

Model Based Design has been extensively applied in the automotive and aerospace domains. It often relies on the use of virtual prototypes (e.g. models) for validation and verification of system requirements. Many of these approaches rely on simulation in order to cover the operational space, through models, Hardware-In-the-Loop and physical testing. Several instances have shown that depending on the nature of the underlying dynamics it is possible to miss certain corner cases. This is particularly true for hybrid systems where the ordering of events can create challenges to define a good set of tests. To cope with these challenges Formal Methods [6, 11] and in particular Model Checking [2] have been used to augment those approaches. [Figure 1](#) shows the overall Model Checking approach where the two main inputs are the system model and the requirement (i.e. specification).

The application of model checking to Cyber Physical Systems (CPS) including physical, control and communication systems, has received much attention, such as the focus on hybrid systems verification [9], as well as the use of probabilistic model checkers (e.g. PRISM [10]).

The work in this paper focuses on the use of model checking for the application to CPS, but using the traditional finite state representations. Rather than attempting to capture the physical dynamics more accurately the approach here focuses on the automated methods to handle large scale industrial problems.

In order to be able to apply Model Checking, the underlying model is typically assumed to be of Finite State [3], and as is detailed later in this paper often requires the modeler to make the appropriate system level abstractions when dealing with Cyber-Physical systems (CPS). In the recent years Model Checking has been applied to aerospace applications, at both the system and software level. The latter application at the software level has shown great benefits and has led

to adoption within the aerospace verification guidelines as an alternate method for creation of verification evidence. Application at the system level has seen less wide adoption due to the challenges involved. Most of these challenges stem from the complexity growth [7] that hampers the efficiency of model checkers, as well as the inherent use of model abstractions of complex dynamics. However, recent advances in both computational capabilities, as well as algorithmic improvements (see the later discussion) have enabled successful applications [1, 4, 5] to more complex systems and can be utilized as a way to detect issues early in the design phase.

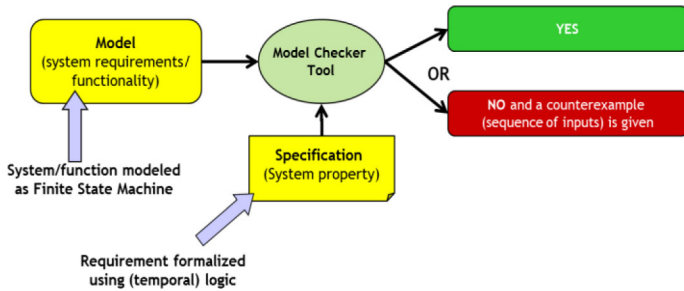


Figure 1. Model Checking

MOTIVATING EXAMPLE: AIRCRAFT ELECTRIC POWER DISTRIBUTION SYSTEM

Electric Power Systems for commercial aircraft are becoming more critical as the overall trend toward more electric aircraft continues. The recent trends have resulted in an increased reliance on electrical power on the aircraft. A system level view of a typical power system is shown in Figure 2. It shows a Single Line Diagram (SLD) that consists of generating sources: {*LGEN*, *RGEN*, *ExtPwr*, *APU*, *Batt*} that represent the generators, the external power coming from a ground cart, the Auxiliary Power Unit, and the battery. In addition it shows the electro-mechanical switches that can be closed to route power to its appropriate loads. Typically a few loads are connected directly onto the system (either AC or DC power), whereas the bulk of the loads would be distributed by a secondary power distribution system (not shown).

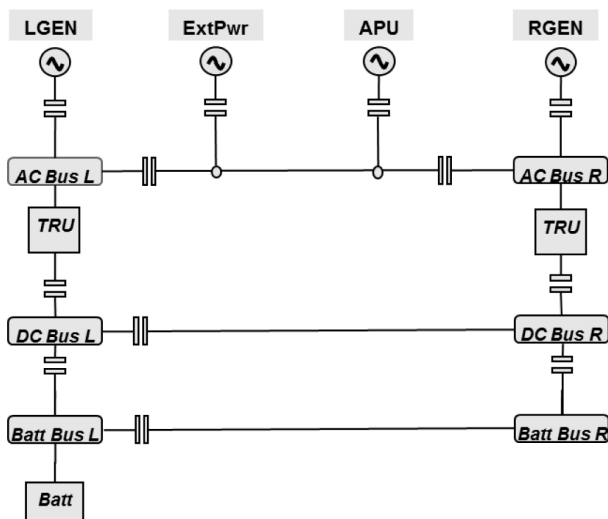


Figure 2. An example power distribution system

The control of the primary power distribution system has to be designed such that switching of power sources happens in a coordinated manner. The control algorithm itself is usually implemented in a distributed set of physical controllers for redundancy reasons. This system can be expressed as a hybrid finite state machine where the continuous dynamics are due to the generators and the electromechanical switches, and the finite state behavior due to the control system.

The objective is to verify the system level behavior and typically system properties. An example is to ensure that the control system will not erroneously connect two variable frequency sources at any point in time. For the subsystem shown in Figure 3 two sources (*S1*, *S2*) are shown and can be connected using two contactors (*C1*, *C2*). The specification can be expressed using for example Linear Temporal Logic (LTL) as:

$$G((ACBUS_powered_by_S1) \& (ACBUS_powered_by_S2))$$

Where *ACBUS powered by S1* is computed based on the discrete states of *S1* and *C1*, and likewise *ACBUS powered by S2* is computed based on the discrete states of *S2* and *C2*.

Similarly properties can be expressed based on other system requirements, such as performance as well as timing constraints. In addition, checking properties that are impossible to verify by testing can be a key reason for application of Formal Methods. The absence of deadlock and livelock conditions is a key concern when designing distributed real-time systems.



Figure 3. An example two generator system

VERIFICATION METHODOLOGY AND TECHNOLOGY

The verification methodology can be summarized in four main parts as shown in Figure 4.

Design Model Definition

During the Design Model definition the control algorithm and the plant models are developed starting from System Requirements. The use of standard modelling guidelines allows for the disciplined use of Simulink/STATEFLOW language constructs. This step can be seen as the “classical” model-based representation of System Requirements.

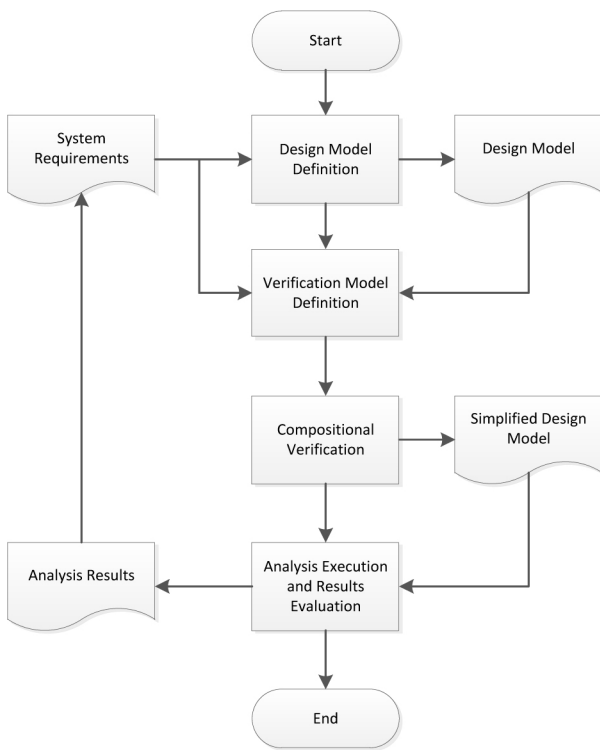


Figure 4. Methodology Overview

Verification Model Definition

During this phase additional modelling artifacts are defined for the subsequent application of the formal analyses. These modeling artifacts represent the requirements or properties against which the model is analyzed or additional constraints that should be provided to the formal engine. The output of this activity is the formal problem to be analyzed.

Compositional Verification

During this phase the formal problem is *decomposed* using both manual and automatic techniques. The objective of this phase is to produce a set of formal models that are *simpler* from a verification complexity point of view and that can be used to prove the correctness of the original model composing the proofs obtained using the derived model.

Analysis Execution and Results Evaluation

During this phase the formal analysis is performed and results are evaluated. In case a property is found false, the produced counterexample is analyzed and possible change requests to the System Requirements associated to the model and the property are performed.

In the next sections we will provide a more detailed view of the described steps as well as the technology chain used to perform the analyses.

VERIFICATION MODEL DEFINITION

The input of the verification model definition phase is the Design Model (Figure 6) developed during the previous phase. It contains both the control algorithm model and the physical plant model. The main activities performed in this phase are:

- The property formalization during which the property to be formalized is encoded using a formal language.
- The development of an environment model and property assumptions that are required to perform the verification step efficiently
- The design and plant model abstractions which are required when the original design model contains constructs that are unsupported by the formal engine, or cannot be processed due to model size or complexity.

In the next sub-section an overview of each activity is provided.

Property Formalization

During this step the property to be verified against the system is formalized using formal language. The choice of the formal language to be used for the formalization impacts the effectiveness of the subsequent simplification and analysis phase. Among the different formal languages that have been used we report three approaches currently adopted.

Linear Time Logic Formulae

Often the application of Formal Methods is used to check key properties of distributed control systems and absence (i.e. negative requirements) of deadlock and livelock conditions. Temporal Logic formulae are adopted for the formalization of liveness requirements. In the context of verification for distributed control systems they have been used for the analysis of deadlock freedom and livelock freedom by encoding these conditions as liveness LTL properties.

Finite State Monitors

A *monitor* (also known as *observer*) is a finite-state machine that takes as inputs both inputs and outputs signals of the System Under Analysis and that has as output a Boolean value. At every time step the monitor check the value of System inputs and outputs and assigns to its output a TRUE value if they satisfy the property it represents, otherwise it signals a property violation assigning a FALSE value. Monitors are commonly used in both formal and run-time verification. Each monitor encodes an *invariant* which is a Boolean condition that must always be true and they can be used to represent both safety properties and bounded-liveness properties that can be reduced to (possibly non-deterministic) finite state machines.

Specification Patterns

The definition of properties using LTL formulae or finite state monitors can be time consuming and error prone. In addition, since natural language requirements are inherently ambiguous, the formalization of the same property from different engineers may lead to different encodings of the formal properties. The use of patterns mitigates both aspects. A pattern is a reusable formalized library block that represents a natural language sentence containing free inputs called parameters. Each pattern encapsulates an interpretation of the natural language sentence.

Example

As an example let us consider the following Textual Pattern provided with the Formal Specs Verifier Block-based Constraint Pattern Library [18, 19]:

Once [C] goes [V], it remains [V] for [T]

The parameters of the pattern are the Boolean condition [C] the Boolean value [V] and the time interval [T]. This pattern is encoded as an LTL specification as follows:

$$G[(C = V) \rightarrow G_T(C = V)]$$

where G_T is shorthand for *globally for a number of steps* and it is implemented as a Simulink monitor block exposing at the interface the three inputs related to the three conditions and as output a Boolean output representing the satisfaction of the pattern.

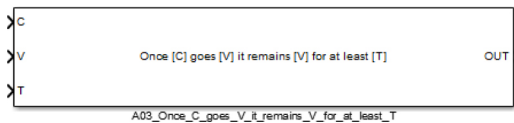


Figure 5. Example of patterns library block

Patterns libraries can be developed for specific domains representing modeling assets that reduce the time required for the properties formalization and verification. In addition the use of patterns allows the use of complex temporal logic expressions without requiring the knowledge of temporal logic.

Assumptions Identification

A key aspect of a verification analysis is the definition of the *assumptions* under which the property to be evaluated should be verified. Assumptions represent “enabling conditions” that the system should satisfy in order to evaluate the property. Even if from a formalization point of view assumptions can be encoded as part of the property, the identification of an explicit assumption identification step allows for better identifying such enabling conditions mitigating the risk of limited or incomplete property formalization.

Environment Model

In addition to the formalized property and related assumptions, during this phase the *environment model* is identified and formalized. In the context of verification, the *environment* is defined as the collection of external actors that interact with the System. These actors are both physical - such as the pilot that can interact using the

switches - and functional - such as the occurrences of faults that leads to a specific failure in the plant. The environment model contains a formal representation of both requirements and domain knowledge related to what is external to the controlled system. From a methodology applicability perspective, the identification of such component is fundamental to avoid the generation of *spurious* counter-examples that are corner cases that violates the properties but shows unreasonable or unlikely scenarios such as the simultaneous occurrence of two inputs or opposite faults. It is important to stress that the environment represents a different set of constraints w.r.t. property’s assumptions: the latter are property-specific condition and belongs to the single property verification “knowledge domain” while the former are system-wide constraints that need to be taken into account by all properties hence representing a *global* set of constraints. The combination of these two “levels” of knowledge - model and property - is a key part of the verification methodology pruning unnecessary behaviors and focusing the search only to relevant behaviors sub-sets.

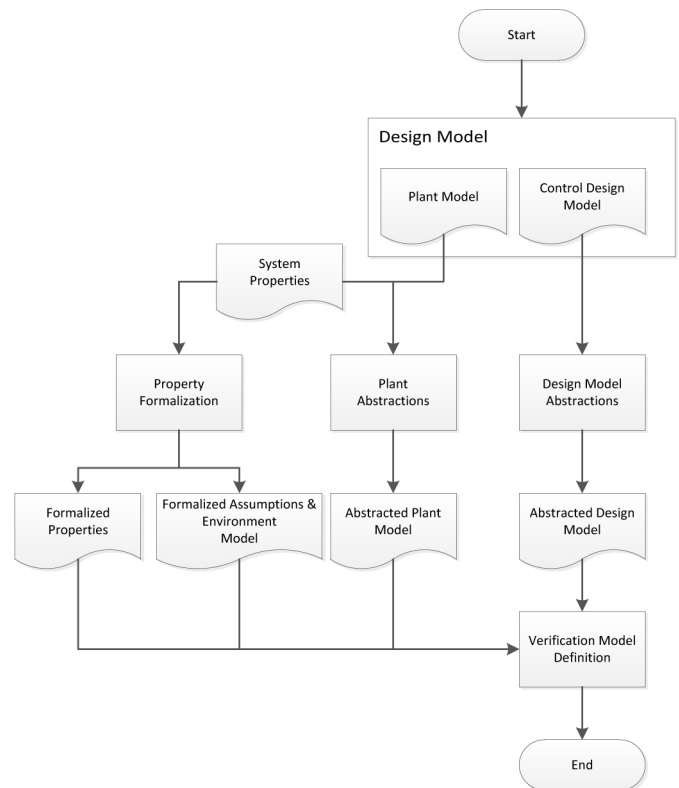


Figure 6. Verification model definition

Abstractions

Classical Model-Based Design methodology places emphasis on the use of models for simulation and automatic code generation. Detailed control and plant models represent a key aspect of such techniques since the effectiveness and usefulness of the results strictly depend on the quality and fidelity of the models. This is true also for the application of formal analysis in the context of Model-Based Design with some additional complexities. Formal engines, due to the exhaustive nature of the search they perform, are inherently limited in terms of supported modeling constructs and model size. In order to successfully overcome such limitations, several abstractions should