

Towards Cooperation of Formal Methods for the Analysis of Critical Control Systems

2011-01-2558

Published
10/18/2011

Adrien Champion
ONERA & Rockwell Collins France

Rémi Delmas, Pierre-loïc Garoche and Pierre Roux
ONERA - The French Aerospace Lab

Copyright © 2011 SAE International

doi:[10.4271/2011-01-2558](https://doi.org/10.4271/2011-01-2558)

ABSTRACT

Critical systems are subject to drastic certification constraints (DO178-B for avionic systems, SIL-4 for railway systems, ISO26262 for the automotive domain), which require system providers to produce strong evidence for the correctness, reliability, or performance of their systems. Today, the early use of formal modeling and verification methods is recognized as favorable by the industry.

Formal methods, which started to appear in the 60's, have now reached a maturity level allowing them to be used in an industrial context.

The approach of control system modeling as proposed by the MathWorks with MATLAB Simulink, by Esterel Technologies with the SCADE language or by the academic community with the Lustre language, is extensively used for reactive systems design and often allows the automatic generation of the embedded code. However, despite the existence of a few formal verification tools supporting these languages, few system builders actually rely on formal approaches to demonstrate safety properties of their software products.

Among the different formal proof techniques available for such models, the k-induction approach gives nice results but often needs external help in order to conclude non-trivial proofs and scale up.

Another method, abstract interpretation, is very efficient to discover properties over programs or models but is not well developed at model level to verify user specified properties.

The novelty of our framework is the tight cooperation between the k-induction engine and the abstract interpretation engine on the analysis of safety properties. The cooperation approach consists in using the abstract interpreter as an oracle in order to infer and inject numerical invariants in the k-induction analysis to prevent spurious falsifications of the induction scheme.

This new collaborative approach seems to be a valuable way to overcome identified drawbacks of each technique and ease the scalability of formal methods at model level.

INDUSTRIAL CONTEXT AND CHALLENGES: REACTIVE SYSTEMS AND CERTIFICATION CONSTRAINTS

Critical embedded systems are more than ever present in our everyday life. Any simple device nowadays contains software, and so do transportation systems. Critical embedded systems are the heart of the latter: flight control systems of aircrafts or other safety critical devices, engine control systems for cars, railway control systems, and even satellite control systems. Critical software systems are also found in other fields of the industry, for instance in medical devices like pacemakers.

Most of these devices are reactive systems: contrary to transformational systems, which execution eventually terminates and produces a value, reactive systems sustain a permanent and never ending interaction with their

environment. For synchronous systems, the underlying algorithm (the reactive loop) is the following:

```
Infinite Reactive Loop

do_init();

while true

    read_input();

    compute_reaction();

    write_output();

end while
```

The challenge is to prove that no unsafe reaction (wrong sequence outputs with respect to current or past inputs) can ever be produced by this infinite loop.

INCREASING USE OF CRITICAL SOFTWARE IN AIRBORNE SYSTEMS

Over the last 30 years, an increasing amount of functionality has been implemented in software. In their earliest and simplest form, critical software systems were limited to local actuator control or to purely logical tasks of a system. Nowadays, a typical reactive software tightly combines numerically intensive computations with discrete and temporal logic, and is in charge of advanced fault diagnosis, system reconfiguration, flight mode management, communication, etc. To give an idea of the importance of this evolution, let us review major landmarks in the history of Airbus airliners:

- [A310 (1975-1985)] Use of electronics for the cockpit interface, software implementation of the autopilot and flight management system on two redundant computers;
- [A320 (1985-1990)] Introduction of fly-by-wire technology, flight control system implemented by 9 computers, software implementation of the maintenance system running on 2 computers;
- [A340 (1990-2000)] Generalization of software to: hydraulic and electric power management system, fuel management system, satellite communication system, navigation system, crew and passenger service;
- [A380-A350 (2000-today)] General increase of the part of functions realized in software, new generation of real-time maintenance embedded and diagnosis systems, global on-board flight information system.

Between the A310 (1975) and A380 (2005) programs, the amount of embedded software rose from a few kilo-bytes to

more than a 100 mega-bytes, while the CPU power has been multiplied by a factor 100. This rapid evolution raises technical issues with respect to design and verification processes, as it is now impossible to guarantee by testing alone that the final software actually fulfils its specification, and is safe to operate in all possible conditions.

In addition to safety requirements, the increase in size and complexity of reactive systems is led by strong economic and environmental requirements, which can only be met by using advanced embedded computing. For instance, the A340-600, which gets its fuel efficiency and large passenger capacity from its very long and light structure, can only be controlled thanks to digital computers running advanced control laws that take into account the flexibility of the structure and continuously compensate its oscillations. In the same way, in order to remain maneuverable despite its gigantic dimensions, the A380 was designed to be naturally unstable, and therefore needs a continuous adjustment of its attitude that can only be achieved using digital computers. Engine and fuel management systems also evolved quickly towards more complexity, as again, digital real-time control is the only way to achieve pollution reduction goals set by international regulations. Similar observations can be made about railway systems, automotive systems, etc. So, even if these external factors are often overlooked in the typical argumentation in favor of formal methods, they are nonetheless truly responsible for the increase in systems complexity.

RECENT ADVANCES IN CERTIFICATION REGULATIONS

Certification regulations strictly specify the process to follow to produce safe systems. These rules describe the sequence of steps to follow, the various documents to produce, verification and validation goals, and the means to use to verify and validate the system.

Current certification regulations are domain-specific. The DO178B targets software in civilian aircrafts; it is a world-wide regulation. The ISO26262 covers the safety regulation for the automotive industry. In Europe, the EN50128 addresses railway systems. Space systems also have their dedicated regulations, but their practical application differs.

In particular, these different certification regulations defined ways to specify and validate software. Until recently, the overall approach was mainly based on testing. Depending on the criticality of the target system, tests run on the software were more or less thorough. Usually, a certification authority receives the system manufacturer and evaluates the system development process according to its own regulation. In practice this phase is a negotiation phase between the certification authorities and the aircraft manufacturer for example. The system development process can partially veer out of the recommended standard development process, but a

case has to be made for it, showing evidence that the applied methodologies, while somewhat different, are still adequate and fulfil the original goals, without leading to unsafe systems.

Significant updates of the aforementioned standards have been issued recently or will be issued in the near future: the DO178B will be soon replaced by its successor the DO178C and the ISO26262 was released in 2008. These new versions include the recent advances of research and their transfer into industry. For example they mention explicitly the possibility to rely on formal methods like static analysis to verify software specifications, instead of or in complement to usual test based or simulation based techniques.

So, the traditional process-based approach to certification - whereby a product is accepted as good if the process is arguably good - starts to show its limits today, and the cost of certification represents a major part of the total cost of a system. The soaring of certification costs (and relative efficiency of test based approaches on large and complex systems) has sprung a number of national and international research projects (ITEA ES_PASS in Europe, NSF CMACS in the US, etc), in which the technical feasibility and potential benefits of moving to a product-based certification approach - whereby the quality of a product is evaluated on the product itself rather than on the process that produced it -- was investigated.

Bringing the cost of design and verification down through the use of formal methods can allow more advanced products to be developed, while staying economically competitive. There are a few examples of success stories of formal verification methods (use of the Astrée static analyzer on A380 flight control systems, use of the B-Method on the Paris Meteor metro line, use of formal safety verification with Prover Certifier in place of traditional safety testing for RATP interlocking and communication-based train control systems, etc.), yet there is still a great potential for deployment of formal methods, that can only be reached by enhancing the current state of the art.

STATE OF THE ART OF FORMAL METHODS: AN OVERVIEW

Since the early 60s, researchers have proposed multiple theoretical frameworks to analyze systems and programs. These techniques, based on formal foundations - mainly discrete mathematics, algebra - allow the exhaustive study of all the behaviors of some categories of systems, as opposed to the test or simulation methodologies which only cover the system traces identified by the run tests.

In the following sections, we briefly mention formal frameworks which have been successfully used, at least recently, in the industry.

DEDUCTIVE METHODS

The basic ideas behind deductive methods were introduced in the late 60s by Floyd and Hoare [Flo67, Hoa69]. This approach was later extended by Dijkstra [Dij68] and became known as the weakest precondition technique.

A piece of code is annotated with its specification as a Hoare Triple (precondition, code, postcondition). Then, a set of predicate transformations is applied to transform the postcondition into the weakest precondition (wp) of the code that entails the postcondition after the code execution. It only remains to prove that precondition $\Rightarrow$ wp(code, postcondition) to prove that the postcondition is always satisfied.

This approach is currently used at Airbus [SFF04] as a replacement for some of the unit tests, covering all possible behaviors of the code. Existing tools often target specifications of imperative code. The main issues with imperative code being to handle complex memory models and properties over structures in the heap, manipulated by algorithms.

ABSTRACT INTERPRETATION

The basic principle of this static analysis technique is to automatically compute an over-approximation of the set of all behaviors of the program (i.e. its semantics). This over-approximation is computed thanks to *abstract domains*. The role of abstract domains is twofold:

- they characterize the nature of the over-approximation which is performed,
- they are equipped with a set of functions, the *abstract primitives*, which allow to compute the abstract semantics.

Each abstract domain is associated with a given trade off between precision (depending on the kind of properties that it can infer) and efficiency (related to the computational complexity of its abstract primitives).

Many different abstract domains have been defined in the literature. Each of them is able to infer a particular class of properties over a given set $v_1, \dots, v_d$ of variables involved in the analyzed program. Some abstract domains allow to infer numerical properties. Among the most well known, we find:

- the interval abstract domain [CC77], which infers invariants of the form $a \leq v_i \leq b, a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$ for every variable v_i ,
- the abstract domain of zones [Min01a], which expresses properties of the form $v_i - v_j \leq c$,
- the domain of octagons [Min01b], which provides invariants of the form $\pm v_i \pm v_j \leq c$,