Formal Methods for the Analysis of Critical Control Systems Models: Combining Non-Linear and Linear Analyses

Adrien Champion[1,2], Rémi Delmas[1], Michael Dierkes[2], Pierre-Loïc Garoche[1] and Pierre Roux[1,3]

[1] Onera – The French Aerospace Lab
[2] Rockwell collins France
[3] ISAE, University of Toulouse

Abstract. Critical control systems are often built as a combination of a control core with safety mechanisms allowing to recover from failures. For example a PID controller used with triplicated inputs and voting. Typically these systems would be designed at the model level in a synchronous language like Lustre or Simulink, and their code automatically generated from these models. We present a new analysis framework combining the analysis of open-loop stable controllers with safety constructs (redundancy, voters, ...). We introduce the basic analysis approaches: abstract interpretation synthesizing quadratic invariants and backward analysis based on quantifier elimination and convex hull computation synthesizing linear invariants. Then we apply it on a simple but representative example that no other available state-of-the-art technique is able to analyze. This contribution is another step towards early use of formal methods for critical embedded software such as the ones of the aerospace industry.

1 Control-Command Software Focused Analyses to Address V&V and Certification Needs

The aerospace industry is notoriously faced with highly critical issues. The safety of systems should be guaranteed even if the cost of ensuring safety is important. In development costs of the Boeing *777* [8], software accounts for a third of all costs. In this third, 70% consists in verification costs while only 30% are devoted to software development. Other aircraft manufacturers have similar figures.

The software specific certification regulatory document, *ie.* the recently updated DO 178-C, characterizes different levels of criticality from level A - the most critical - to level E - the less critical. Depending on the identified level, verification and validation activities are more or less intensive and therefore costly. This certification document has recently been updated and it also provides a formal methods supplement, identified as RTCA DO 333. This supplement explicitly enables the use of formal methods for critical embedded software.

Among the various systems of an aircraft, and their associated software, one of the most critical is the flight control system of the aircraft. Addressing the issue of verifying such specific software seems to be a pertinent goal: proposing

new ways to validate it could both increase the trust we have in the released software and reduce the cost of V & V by providing more automatic (and exhaustive) analysis means.

These reactive system can be seen as the composition of two parts. The first is the computation core itself, achieving the main objective of the software: controlling the aircraft by receiving inputs from sensors and commanding the aircraft actuators. The second part tries to handle any possible failure of sensors or of the core system. This safety architecture is mainly based on information redundancy and fusion. These two parts are usually designed using a model based approach.

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.

Recent advances of formal methods, as well as the evolution of certification standards enable the deployment of formal methods in the industry to analyze such systems. Formal methods can thus be truly considered as a key technological advantage on the critical systems market. Thanks to long term research efforts, formal methods have matured up to the point they were found, by industrials, to be helpful in dealing with the difficulties arising from highly complex system designs, and enabled system providers to meet the requirements of certification which are to:

- provide evidence of the system safety, and
- master the overall product life-cycle.

Our goal is to support the verification and validation of such systems for all their specification, at the various stages of their development. This paper focus on a representative running example: a controller for a physical device together with inputs triplication and voting. We show how we propose to analyze such systems by composing new-generation formal methods.

The paper is structured as follows: Section 2 recalls the state of the art of formal methods in that context; Section 3 presents the running example in details; Sections 4 and 5 introduce our contribution, two new automatic analyses; Section 6 illustrates the use of these new techniques in combination on the example; and Section 8 presents the tools implementing these techniques.

2 State of the Art of Formal Methods

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 test or simulation methodologies which only cover the system traces identified by a collection of tests scenarios.

We briefly focus here on relevant techniques for the analysis of functional specifications of control systems. These techniques, from early academic work

driven by industry needs, to actual transferred technologies[18], are currently used in the aerospace industry, at different TRLs[4].

2.1 Abstract Interpretation

Abstract interpretation was first proposed in the 70s as a general framework to express static analyses. In practice, it has shown to be very efficient to compute numerical invariants over programs.

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: (1) they characterize the nature of the over-approximation which is performed, (2) 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). A wide literature addresses the definition of abstract domains, and more specifically numerical abstract domains, *eg.* intervals [5], polyhedra [6] or weaker but less costly domains such as zones [13] or octagons [14].

A success story of abstract interpretation is the complete analysis of the flight control systems of the Airbus A380 [12], which has proved that no runtime error (out of bounds memory accesses or arithmetic exceptions) can occur.

One of the domains used to perform this analysis is specifically focused on second order linear filters. This domain is able to precisely over-approximate their set of reachable states. We present in this paper another similar abstraction that outperforms it in terms of expressiveness.

2.2 SMT-Based Verification Approaches

Satisfiability Modulo Theory Satisfiability Modulo Theories solvers are decision procedures for logical theories in which some atoms belong to certain decidable first order theories such as linear real/integer arithmetic, the theory of bit-vectors, the theory of arrays (with read over write axioms), etc. Roughly speaking, these procedures are usually built by extending Boolean satisfiability procedures with a combination of dedicated background theory solvers [20].

SMT-solvers are used as back-end reasoning engines in a wide range of formal verification applications, such as deductive methods, bounded model checking, k-induction, test case generation, etc. Recently, the SMT-lib initiative (`http://www.smtlib.org/`) has gathered major SMT-solver developers around a standardized formula and solver command language. The SMT-lib 2.0 standard introduced specific features easing the implementation of incremental verification approaches like BMC or k-induction.

[4] Technology Readiness Levels as defined by NASA; see http://esto.nasa.gov/files/trl_definitions.pdf

Quantifier Elimination Assuming a first order formula over Boolean, real or integer variables $\exists x, \mathcal{F}(x, y_1, \ldots, y_n)$, where $\mathcal{F}$ is quantifier-free, quantifier elimination allows to generate a new formula

$$\mathcal{G}(y_1, \ldots, y_n) \equiv \exists x, \mathcal{F}(x, y_1, \ldots, y_n)$$

by *eliminating* the quantified variable x from $\mathcal{F}$. Slightly rephrased, quantifier elimination generates a condition $\mathcal{G}$ on variables $y_1, \ldots, y_n$ which, when satisfied, entails the existence of an x such that $\mathcal{F}(x, y_1, \ldots, y_n)$ is also satisfied.

Even though the theory of real closed fields admits quantifier elimination [4,19], general non-linear QE methods have extremely high computational costs, limiting their practical applications. This is why QE for linear fragments of integer and real theories has been a very busy research domain. The most recent advance for linear QE combines state of the art SMT-solving with polyhedral projection [15] for a great performance increase, the general idea of which is given in Algorithm 1.

Algorithm 1 $QE(\mathcal{F}, V)$ QE by Lazy Model Enumeration.

Require: $\mathcal{F}$: a linear arithmetic formula.
Require: V: a collection of variables to eliminate from $\mathcal{F}$.
Ensure: $\mathcal{O}$: a formula in disjunctive normal form such that $\mathcal{O} \equiv \exists \mathcal{V}, \mathcal{F}$
  $\mathcal{O} \leftarrow \bot$
  while *isSatisfiable*$(\mathcal{F} \wedge \neg\mathcal{O})$ **do** ▷ check satisfiability using an SMT solver.
    $M \leftarrow getModel(\mathcal{F} \wedge \neg\mathcal{O})$ ▷ get a model using an SMT solver.
    $P \leftarrow extrapolate(\mathcal{F}, M)$ ▷ extrapolate M, yields a conjunction of literals which entail $\mathcal{F}$.
    $P' \leftarrow project(P, V)$ ▷ polyhedral projection.
    $\mathcal{O} \leftarrow \mathcal{O} \vee P'$
  end while
  return $\mathcal{O}$

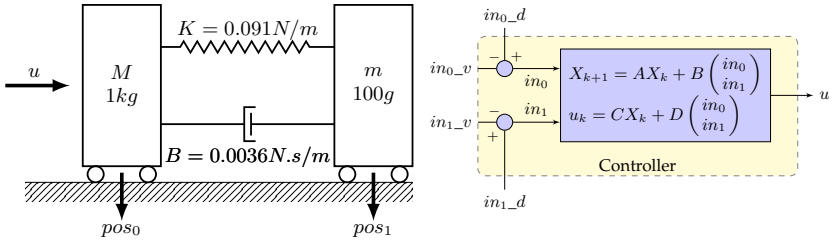
The *extrapolate* function generalizes the model M with respect to $\mathcal{F}$ and produces a conjunction of literals P, *i.e.* a polyhedron in geometric terms, such that $M \models P$ and $P \implies \mathcal{F}$. Polyhedral projection is then used to eliminate variables V from P and obtain another P' characterizing a polyhedron of lower dimension. The formula $\mathcal{O}$ resulting from this procedure can be viewed as a union of polyhedra over reals/integers.

Quantifier elimination enjoys many applications in formal verification: pre-image computation on transition systems, automatic program abstraction and even controller synthesis, to name only a few. Section 5 details our use of QE for pre-image computation and lemma generation.

3 Running Example: Coupled Mass System Linear Controller Behind Triplicated Inputs with Saturations

Throughout this article, we consider the control of the coupled mass system[5] shown in Figure 1a. Such a coupling can be used to model numerous physical phenomena such as vibration propagation patterns in fluids and flexible structures, among others.

[5] This system is extracted from [17].



(a) Coupled mass system.

(b) Controller node.

Fig. 1: Coupled mass system and its linear controller.

3.1 Controller Design

The continuous model for the plant, i.e. the physical system, is as follows:

$$\dot{x}_p = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -K/m & -B/m & K/M & B/M \\ 0 & 0 & 0 & 1 \\ K/M & B/M & -K/M & -B/M \end{bmatrix} x_p + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/M \end{bmatrix} u_p \qquad y_p = \begin{bmatrix} 1\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0 \end{bmatrix} x_p$$

where the output y_p consists of the displacement of mass 1 and mass 2 with respect to their equilibrium position.

It is discretized with a period $T = 0.4s$ with a zero order hold, yielding:

$$x_{p,k+1} = \begin{bmatrix} 0.9285 & 0.3876 & 0.0715 & 0.0124 \\ -0.3516 & 0.9146 & 0.3516 & 0.0854 \\ 0.0071 & 0.0012 & 0.9929 & 0.3988 \\ 0.0352 & 0.0085 & -0.0352 & -0.9915 \end{bmatrix} x_{p,k} + \begin{bmatrix} 0.0013 \\ 0.0124 \\ 0.0799 \\ 0.3988 \end{bmatrix} u_{p,k} \quad y_{p,k} = \begin{bmatrix} 1\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0 \end{bmatrix} x_{p,k}$$

Also assuming the presence of some sensor noise, the control theorist designs the following so called *linear quadratic Gaussian regulator*:

$$\begin{cases} x_{k+1} = Ax_{c,k} + By_{c,k} \\ u_{c,k} = Cx_{c,k} + Dy_{c,k} \end{cases}$$

with:

$$A = \begin{bmatrix} 0.6229 & 0.3871 & -0.117 & 0.0102 \\ -0.3363 & 0.9103 & -0.4062 & 0.06486 \\ 0.1134 & -0.02646 & -1.065 & 0.2669 \\ 0.2877 & -0.1298 & -1.333 & 0.3331 \end{bmatrix} \quad B = \begin{bmatrix} 0.3063 & 0.1866 \\ -0.009808 & 0.7406 \\ -0.07102 & 1.947 \\ -0.07649 & 0.7439 \end{bmatrix}$$

$$C = \begin{bmatrix} -0.1896 & -0.346 & 6.709 & -1.651 \end{bmatrix} \qquad D = \begin{bmatrix} 0.6311 & -8.098 \end{bmatrix}$$

The controller is parameterized by the positions of the two masses. Figure 1b illustrates the architecture of this linear controller, in which in_0_d and in_1_d correspond to the desired positions while in_0 and in_1 (denoted $y_{c,k}$ in the equations) are the differences between measured and desired positions.

This classic design uses a Kalman state estimator combined with a linear quadratic regulator. Typical high level properties guaranteed by this design include that the variance of the estimation error is minimized and convergence

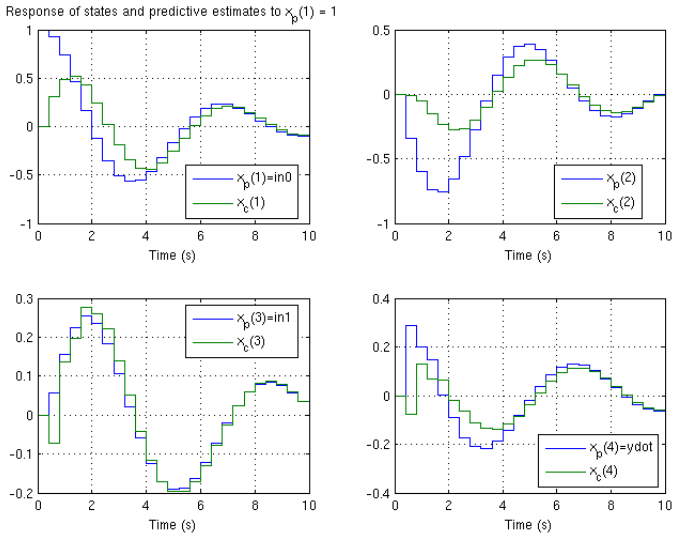


Fig. 2: Simulation of the controlled system. $x_{p,i}$ correspond to actual plant states and $x_{c,i}$ to associated estimates in the controller. Simulation starts with an initial position of 1 for m_1.

to zero in the absence of noise, as shown in Figure 2, obtained by simulation in the absence of noise and under $x_{p,0} = \begin{bmatrix}1\ 0\ 0\ 0\end{bmatrix}^{\mathrm{T}}$, $x_{c,0} = \begin{bmatrix}0\ 0\ 0\ 0\end{bmatrix}^{\mathrm{T}}$.

In the rest of this article, we focus on the simpler, yet essential bounded-input, bounded-output (BIBO) stability of the controller, as it is a high level property that can be established by inspecting the controller code alone. At the system level of description, inspecting the modulus (.84, .84, .0063 and .73) of the eigenvalues of the controller state matrix A is enough to guarantee that property. However, more elaborate techniques are required at code level since neither these eigenvalues nor their link with BIBO stability are readily available.

3.2 Safety Architecture

In real world systems, the controller will not be directly embedded on the target platform but rather used in conjunction with a safety architecture used to obtain a fault tolerant system.

In our case, we choose to rely on representative yet simple building blocks: validators and voters. We assume the sensors could be faulty and provide an erroneous data. Either the data could be out of a legal value range or it can vary within the legal range. First, a set of validators allows to correct the signals by saturating them to keep them within the legal range. Then, the triplication of each sensor allows to tolerate a faulty sensor. We rely on the triplex presented in [7].

This fault-tolerant system is depicted in Figure 3. To guard against execution hardware errors (CPU or RAM), one could pursue the experiment by triplicating the controller core and vote on its outputs, or rely on the command/monitor

(COM/MON) architecture. An even more fault-tolerant and diagnosable system would also implement error detection flows with some alarm logic combining information about detected errors. However, in this paper we focus on numerical properties of control systems.

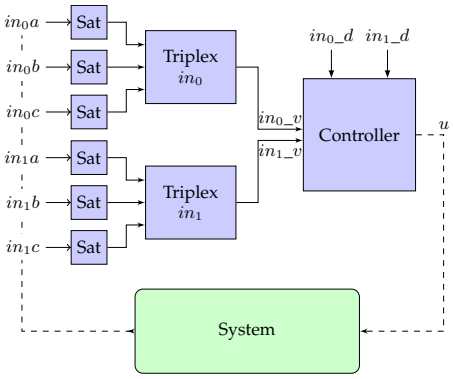


Fig. 3: Full controller.

3.3 Need for Formal Specification

The specification of the obtained system should now be precised. We rely on axiomatic semantics with Hoare triples to associate a function contract to each building block of the system. For this running example, we can identify the following contracts:

- the system should ensure that the output on data flow u does not exceed the capability of the moving mass m_1 hardware actuator. For example, if the maximal capability is $200N$, we should ensure that $|u| \leq 200$;
- the controller itself has specific system-level properties as described above. In this paper we restrict on the BIBO property;
- the triplex voter contract, presented in [7], bounds the output of the voter according to bounds on the input. It is also a BIBO property, the formalisation and verification of which is detailed in Section 5.1;
- the Sat nodes correspond to the validator nodes evocated above for the replicated sensors, a typical specification would be that $lb \leq o \leq ub$ where o is the output flow, lb and ub are lower and upper saturation bounds.

These requirements should be satisfied by the blocks and their later implementations, and each of them should be formally expressed in a logic language format that can be parsed and processed by available tools.

This running example is simple but representative of control command software, and its analysis is meaningful since most of these properties are not currently analyzable at model or code level with existing formal tools.

4 BIBO (Bounded Input Bounded Output) for the Control Core: the Need for Non-Linear Invariants

We focus here on the controller node and its BIBO property as characterized in Section 3.1. Assuming a bound on the inputs in (y_k in the equation), we want to bound x in the system $x_{k+1} = Ax_k + By_k$ and therefore the output u_k.

Abstract interpretation is a good way to compute numerical abstraction of reachable states, therefore computing bounds on reachable values. However the precision of the obtained results highly depends on the abstraction used (intervals, polyhedra, octagons, ...). When we assume a bound of $|y_{0k}| \leq 3.6$

and $|y_{1_k}| \leq 3.6$, trying to analyze this with intervals gives for the first few iterates (rounding figures to one digit after the point):

x_0	x_1	x_2	x_3
$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
$[-1.8, 1.8]$	$[-2.7, 2.7]$	$[-7.3, 7.3]$	$[-3.0, 3.0]$
$[-4.8, 4.8]$	$[-8.9, 8.9]$	$[-16.1, 16.1]$	$[-14.5, 14.5]$
$[-10.2, 10.2]$	$[-19.9, 19.9]$	$[-29.0, 29.0]$	$[-31.7, 31.7]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

This does not converge. Such linear system, so called *open-loop stable* (*ie.* admitting this BIBO property), usually do not admit linear inductive invariants. A possible approach to prove such stability is to rely on Lyapunov stability theory, stated as the existence of a positive definite[6] matrix P such that

$$P - A^T P A \succ 0$$

Finding any such P gives an invariant $x^T P x \leq x_0^T P x_0$ for the sequence $x_{k+1} = Ax_k$. When assuming bounds on the inputs y_k, it is possible to compute a scalar λ such that the set of values of $x_{k+1} = Ax_k + By_k$ could be bound by the relation $x^T P x \leq \lambda$.

We developed an approach combining the synthesis of an appropriate quadratic template P [16] for a given linear system with a static analysis engine based on policy iteration to compute the appropriate λ.

Given a finite set $\{ t_1, \ldots, t_n \}$ of expressions on program variables $\mathbb{V}$, the template domain $\mathcal{T}$ is defined as $\overline{\mathbb{R}}^n = (\mathbb{R} \cup \{ -\infty, +\infty \})^n$ and the meaning of an abstract value $(b_1, \ldots, b_n) \in \mathcal{T}$ is the set of environments $\gamma_{\mathcal{T}}(b_1, \ldots, b_n) = \{\rho \in (\mathbb{V} \to \mathbb{R}) \mid [\![t_1]\!](\rho) \leq b_1, \ldots, [\![t_n]\!](\rho) \leq b_n\}$. In other words, the abstract value $(b_1, \ldots, b_n)$ represents all the environments satisfying all the constraints $t_i \leq b_i$. In our case, in addition to templates allowing to bound each variable, a quadratic template will be added for each linear system of the program.

Example 1. We then have five templates on the running example: (1) $t_1 := 0.098\, x_3^2 - 0.224\, x_3\, x_2 + 0.040\, x_3\, x_1 - 0.026\, x_3\, x_0 + 0.141\, x_2^2 - 0.053\, x_2\, x_1 + 0.030\, x_2\, x_0 + 0.024\, x_1^2 - 0.017\, x_1\, x_0 + 0.019\, x_0^2$ (2) $t_2 := x_0^2$ (3) $t_3 := x_1^2$ (4) $t_4 := x_2^2$ (5) $t_5 := x_5^2$.

Policy iterations [1,10,11] give then a direct methods to compute a precise over-approximation of the collecting semantics of a program, given a set of such templates. It intends to achieve better precision than the usual widening/narrowing approach of abstract interpretation Kleene iterations by computing a numerical solution of the problem.

While Kleene iterations iterate locally through each construct of the program, policy iterations require a global view of the analyzed program. For that purpose, the whole program is first translated into a system of equations which is later solved.

A first step in deriving these equations from the program is to build its control flow graph.

[6] A matrix M is positive definite (noted $M \succ 0$) if $x^T P x > 0$ for all $x \neq 0$.

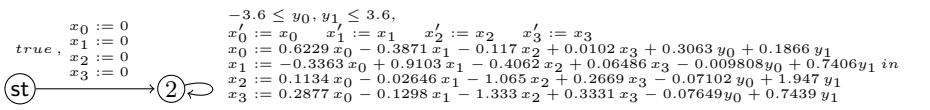


Fig. 4: Control flow graph for our running example.

Example 2. Figure 4 represents the control flow graph for our running example. Vertex "st" corresponds to the starting point of the program and vertex "2" to the head of the loop. The edge between "st" and "2" reflects the four assignments before the loop and the looping edge on vertex "2" represents the loop body.

It is worth noting that, unlike the usual notion of control flow graph with vertices between each single instruction of the program, whole sequences of instructions are used to label the edges, with graph vertices only for starting point and loop heads of the program. This will both improve the precision of the analysis and decrease its cost by avoiding useless intermediate abstractions.

A system of equations is then defined with a variable $b_{i,j}$ for each vertex i of the graph and each template t_j.

Example 3. Here is the system of equations for our running example:

$$\begin{cases} b_{1,1} = +\infty \quad b_{1,2} = +\infty \quad b_{1,3} = +\infty \quad b_{1,4} = +\infty \quad b_{1,5} = +\infty \\ b_{2,1} = \max\{0 \mid \mathrm{be}(1)\} \vee \max\{a(t_1) \mid -3.6 \leq in \leq 3.6 \wedge \mathrm{be}(2)\} \\ b_{2,2} = \max\{0 \mid \mathrm{be}(1)\} \vee \max\{a(t_2) \mid -3.6 \leq in \leq 3.6 \wedge \mathrm{be}(2)\} \\ b_{2,3} = \max\{0 \mid \mathrm{be}(1)\} \vee \max\{a(t_3) \mid -3.6 \leq in \leq 3.6 \wedge \mathrm{be}(2)\} \\ b_{2,4} = \max\{0 \mid \mathrm{be}(1)\} \vee \max\{a(t_3) \mid -3.6 \leq in \leq 3.6 \wedge \mathrm{be}(2)\} \\ b_{2,5} = \max\{0 \mid \mathrm{be}(1)\} \vee \max\{a(t_4) \mid -3.6 \leq in \leq 3.6 \wedge \mathrm{be}(2)\} \end{cases}$$

where $\mathrm{be}(i)$ is a shorthand for $t_1 \leq b_{i,1} \wedge t_2 \leq b_{i,2} \wedge t_3 \leq b_{i,3} \wedge t_4 \leq b_{i,4} \wedge t_5 \leq b_{i,5}$ and $a(t)$ is the template t in which variable x_0 to x_3 are replaced by their image by the linear combination $Ax_k + By_k$. The usual maximum on $\overline{\mathbb{R}}$ is denoted $\vee$.

Each $b_{i,j}$ bounds the template t_j at program point i and is defined in one equation as a maximum over as many terms as incoming edges in i. More precisely, each edge between two vertices v and v' translates to a term in each equation $b_{v',j}$ on the pattern: $\max\left\{a(t_j) \;\middle|\; c \wedge \bigwedge_j t_j \leq b_{v,j}\right\}$ where c and a are respectively the constraints and the assignments associated to this edge. This expresses the maximum value the template t_j can reach in destination vertex v' when applying the assignments a on values satisfying both the constraints c of the edge and the constraints $t_j \leq b_{v,j}$ of the initial vertex v.

Our tool computes these steps automatically when provided with the set of variables that participate in a BIBO linear controller. It rebuilds the control flow graph and characterizes linear relations, synthesizes the associated quadratic templates and performs the policy iterations computation to obtain the bounds.

On the running example, we obtain $0.098\,x_3^2 - 0.224\,x_3\,x_2 + 0.040\,x_3\,x_1 - 0.026\,x_3\,x_0 + 0.141\,x_2^2 - 0.053\,x_2\,x_1 + 0.030\,x_2\,x_0 + 0.024\,x_1^2 - 0.017\,x_1\,x_0 + 0.019\,x_0^2 \leq 14.259$ resulting in following bounds on variables

$$|x_0| \leq 25.423 \quad |x_1| \leq 26.844 \quad |x_2| \leq 33.612 \quad |x_3| \leq 37.164 \quad |u| \leq 194.499.$$

Figure 5 depicts a cut of this invariant along plane $x_2 = x_3 = 0$. The ellipsoid shape corresponds to the bound on the quadratic polynomial while vertical cuts on the x_0 axis correspond to the bound on $|x_0|$.

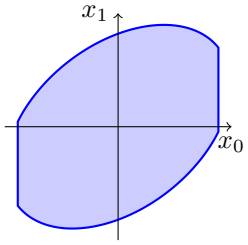


Fig. 5: Resulting quadratic invariant projected on variables x_0 and x_1.

Indeed, the largest value we were able to reach by random simulation for variable u is 190.019. The actual maximum reachable value may be higher but it is worth noting that our bound 194.499 is less than 2.4% larger. Other linear abstractions perform poorly on such reactive systems, or on typical aircraft controllers. Our solution is automatic when provided with the set of variables of the considered linear system.

5 BIBO Stability for the Triplex Voter: Linear Analysis

5.1 Triplex Voting Verification Challenges

The Triplex node used in the presented controller design implements redundancy management for three sensor input values InA, InB, InC, aiming at providing sensor fault tolerance. This voter does not compute an average value, but uses the $\mathbf{median}(x, y, z)$ function, which returns the median of its arguments(for instance z when $y \leq z \leq x$). The values considered for voting are *equalized* by substracting equalization values from the inputs. The role of the equalization values is to compensate offset errors of the sensors, assuming that the middle value gives the most accurate measurement. The recursive equations in Figure 6 describe the behavior of the voter. The $\mathbf{sat}_c(x)$ function saturates its argument such that $|\mathbf{sat}_c(x)| \leq c$.

initialization

$$
\begin{array}{lcl}
EqualizationA_0 & = & 0.0 \\
EqualizationB_0 & = & 0.0 \\
EqualizationC_0 & = & 0.0
\end{array}
$$

transition $\forall k \in \mathbb{N}$

$$
\begin{array}{lcl}
EqualizedA_k & = & InA_k - EqualizationA_k \\
EqualizedB_k & = & InB_k - EqualizationB_k \\
EqualizedC_k & = & InC_k - EqualizationC_k \\
SatA_k & = & \mathbf{sat}_{0.5}(EqualizedA_k - Output_k) \\
SatB_k & = & \mathbf{sat}_{0.5}(EqualizedB_k - Output_k) \\
SatC_k & = & \mathbf{sat}_{0.5}(EqualizedC_k - Output_k) \\
EqualizationA_{k+1} & = & EqualizationA_k + 0.05 * (SatA_k - SatCentering_k)) \\
EqualizationB_{k+1} & = & EqualizationB_k + 0.05 * (SatB_k - SatCentering_k)) \\
EqualizationC_{k+1} & = & EqualizationC_k + 0.05 * (SatC_k - SatCentering_k)) \\
Centering_k & = & \mathbf{median}(EqualizationA_k, EqualizationB_k, EqualizationC_k) \\
SatCentering_k & = & \mathbf{sat}_{0.25}(Centering_k) \\
Output_k & = & \mathbf{median}(EqualizedA_k, EqualizedB_k, EqualizedC_k)
\end{array}
$$

Fig. 6: Triplex voter equations.

Just as with the controller core, we are interested in proving the BIBO stability of the voter, that is in ensuring that the voter output cannot grow indefinitely as long as its inputs remain within a certain range. This property is needed for the overall design validation since the BIBO stability of the controller core makes assumptions on the voter outputs. Here, the difficulty of automating the voter proof lies in inferring the right auxiliary inductive invariant on the *internal* equalization values $EqualizationA$, $EqualizationB$, $EqualizationC$, whereas the main proof obligation and assumptions are expressed only on the voter's *interface* variables InA, InB, InC and $Output$. Equation (1) shows a refined, yet still not inductive $BIBO$ property including the equalization values.

$$BIBO(a) \equiv \forall k \in \mathbb{N},\ |InA_k| \leq a \wedge |InB_k| \leq a \wedge |InC_k| \leq a \implies |Output_k| \leq 3a \wedge \\ |EqualizationA_k| \leq 2a \wedge |EqualizationB_k| \leq 2a \wedge |EqualizationC_k| \leq 2a \tag{1}$$

In [7], the author managed to manually identify octagonal regions enclosing the equalization values (a relational invariant over the sums of equalization values), and prove them using a k-induction tool. The k-induction technique, as it can only prove user-specified properties and not infer new properties by itself, is of no help in this situation. Discovering an appropriate invariant using abstract interpretation is theoretically possible. However, the combination of the non-linear $\mathbf{median}$ and $\mathbf{sat_c}$ functions with the filter-like calculations over equalization values is such that abstract interpretation cannot infer the desired invariants without specifying numerous domain partitioning directives, which must be guessed by the human user, a task impossible to achieve in practice.

5.2 HullQe: A Technique for Property Directed Invariant Generation

The difficulties posed by systems such as the triplex voter lead us to design a new invariant generation technique named HullQe, based on a backwards state space traversal and polyhedral computations [3].

Given a triplet $\langle\langle I, T\rangle, P\rangle$, where $\langle I, T\rangle$ is transition system specified by an initial state predicate I, and a transition predicate T, and P is a proof objective, HullQe works as follows.

A backwards state space traversal allows to discover successive fringes of *gray states*. Gray states are states which satisfy P but from which a states violating P can be reached in one or more T-transitions. When only numerical state variables are considered, such fringes of gray states can be thought of as unions of polyhedra over state variables. The quantifier elimination procedure of Algorithm 1 is iterated as follows to discover new gray states:

$$G_1(s) \equiv QE(s', P(s) \wedge T(s, s') \wedge \neg P(s')) \\ G_n(s) \equiv QE(s', P(s) \wedge T(s, s') \wedge G_{n-1}(s'))$$

At each iteration n HullQe first checks whether $I(s) \wedge \bigvee_{i\in[1,n]} G_i(s)$ is satisfiable, which would indicate that there exists a path to $\neg P(s)$ from the initial states. HullQe also checks whether $\bigvee_{i\in[1,n-1]} G_i(s) \implies G_n(s)$ is valid, which would indicate a fixed point has been reached in the backwards exploration.

In order to discover new invariants, HullQe performs a thorough exploration of all possible partitionnings of the gray states discovered so far. The following sets of new polyhedra are computed until no new polyhedra can be found or merged. This computation takes place after each pre-image calculation and is initialised with all pre-images found so far ($\bigvee_i G_i$):

$$\begin{array}{lllll}
H_1 = \{\ p & , & p \in \bigvee_i G_i & & \} \\
H_2 = \{\ hull(p,p')\ , & p \in H_1 & , & p' \in H_1 & \} \\
H_3 = \{\ hull(p,p')\ , & p \in H_2 & , & p' \in H_1 \cup H_2 & \} \\
\vdots \quad \vdots & \vdots & & \vdots & \\
H_n = \{\ hull(p,p')\ , & p \in H_{n-1} & , & p' \in \bigcup_{i\in[1,n-1]} H_i & \}
\end{array}$$

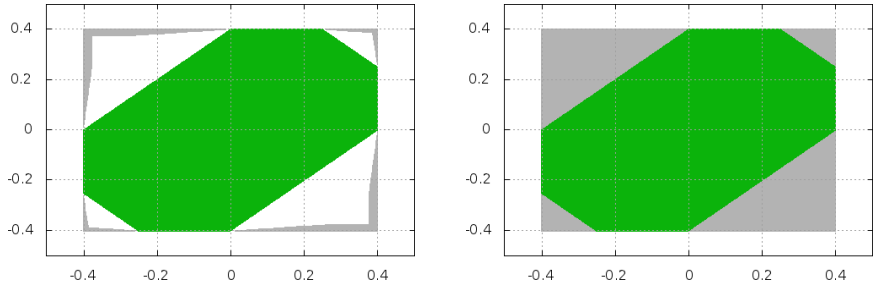
In the above definitions, the *hull* function, which computes a convex hull of two polyhedra, can be used in either *exact mode*, whereby a hull will be returned only if it does not contain more points than the two source polyhedra, or in *inexact mode*, by accepting hulls which contain strictly more points than the two given polyhedra, thus performing abstraction by accepting more states than already present in the root polyhedra. When using the inexact mode, it is possible to accept inexact hulls only for polyhedra with a non-empty intersection. This allows to overapproximate clusters of gray states while avoiding to merge completely disconnected zones of the gray state space. In order to limit the combinatorial explosion, for each hull the algorithm keeps track of the base polyhedra it is derived from, and mathematical properties of convex hulls as well as dynamic programming techniques are used to skip uninteresting combinations of polyhedra, which would necessarily result in already existing hulls, or for which the exact hull computation would fail.

Each polyhedra in $H_1, \ldots, H_n$ is viewed as a conjunction of linear relations over state variables delimiting portions of the gray state space. If P is invariant, none of the gray states should be reachable, so the negations of these relations are then taken as proof obligations and analyzed using k-induction in conjunction with P. Many of the generated relations will be non-invariant or even falsifiable, but sometimes some of them can successfully strengthen the original proof obligation.

Let us illustrate the HullQe technique on a two-input voter, for which the $BIBO$ proof objective is that:

$$\begin{aligned}
BIBO(a) \equiv & \forall k \in \mathbb{N}, |InA_k| \leq a \wedge |InB_k| \leq a \implies \\
& |Output_k| \leq 3a \wedge |EqualizationA_k| \leq 2a \wedge |EqualizationB_k| \leq 2a
\end{aligned}$$

The Figure 7a shows a plot of the state space for state variables $EqualizationA$ and $EqualizationB$ of the two-input voter and the proof objective $BIBO(0.2)$. In green, we show the octagonal invariants of [7], in gray, we show the gray states polyhedra obtained after two iterations of the pre-image computation. The Figure 7b shows the result of HullQe's inexact hull computation modulo non-empty intersection on the two-input voter. We can see that the gray regions obtained by HullQe match exactly the octagonal invariant of [7], their negation then exactly delimits the octagonal region enclosing the equalization values.



(a) Octagonal invariant and gray states. (b) HullQe inexact hull computation.

Fig. 7: Two-input voter

The following results are obtained when analyzing $BIBO(1.2)$ (cf Equation 1) on the triplex voter: the first pre-image of the negated proof objective contain 23 distinct polyhedra, the HullQe lemma generation algorithm creates 41 potential lemmas, out of which 32 are found to be 1-inductive and allow to strengthen the proof objective. Out of all the generated lemmas, the following two suffice to prove the $BIBO(1.2)$ proof objective:

$$-29/10 \leq EqualizationA_k + EqualizationB_k + EqualizationC_k \quad (2)$$

$$EqualizationA_k + EqualizationB_k + EqualizationC_k \leq 29/10 \quad (3)$$

So, we see that, given a non-inductive and non-relational proof objective on the input, output and internal variables of the voter, HullQe is able to discover an inductive polyhedra on internal state variables, which renders the $BIBO$ proof objective inductive.

6 Composing Proofs

Using the two previous techniques, implemented in our prototypes, it is now possible to perform an analysis of the complete system presented in Figure 3. When run in parallel, the analyzers communicate their results to each other. We only assume the following knowledge:

- The **Triplex** node description is fitted with the contract presented in Equation 1, page 11;
- The **Controller** node is known to be a linear open stable controller; its internal variables are automatically considered for analysis with our quadratic templates analysis.

We would like to guarantee that the system does not diverge whatever the inputs are. This contract could either be implicit, *ie.* no overflow, or specific, *eg.* the output should satisfy a given constraint $|u| \leq 200(N)$.

We recall that – up to our knowledge – none of these specifications is provable by any academic nor commercial tool available except ours. Let us describe the sequence of analyses needed to achieve the global proof:

1. Using abstract interpretation, **Sat** node outputs are bounded to their saturation value of 1.2;
2. These bounds enable the instanciation of the **Triplex** contracts. As described in Section 5, taking the instanciated contracts as proof objectives, HullQE with k-induction generates the missing lemmas and proves the contracts, effectively bounding the output of both voters by 3.6. The analysis is fully automatic once the proof objective is given.
3. Once the inputs of the controller node are bounded (*ie.* voter outputs), the analysis of Section 4 is enabled: the control flow graph is computed, the quadratic template is synthetized and the policy iteration is performed, bounding the internal variables and the output u of the controller: $|u| \leq 194.499$. The analysis is fully automatic and only requires the list of internal variables of the controller as well as bounds on the inputs.

Finally the global contract can be checked. The quadratic invariant synthesized by abstract interpretation satisfies the required bound of $200N$.

7 Tools

The presented analyses are currently implemented in two different tools, freely available.

Stuff [2] is a Lustre analysis framework that combines a k-induction engine with the HullQE technique presented in Section 5.2. It is about 30k loc of Scala. It uses the Actor-oriented programming approach to use the multiple CPU cores of the machine. It must be provided with the Lustre code as weel as a set of proof objectives to analyse.

SMT-AI [9] is an abstract interpreter that targets Lustre models. It computes the over-approximation of reachable values. It relies on the APRON library for linear domains and also embed the analysis presented in Section 4. It is written mainly in Ocaml for about 20k loc, numerical computation are performed with the CSDP and OCaml-GLPK libraries and Scilab.

8 Concrete Outcomes and Future Works

We proposed a combination of formal methods to achieve the verification of formal functional specifications of control-command systems. This framework of analysis is highly generic and automatic. The two presented analyses are able either to compute precise information about reachable states or to validate functional contracts; in both cases without needing to interact with the user.

The kind of systems targeted by this work – linear controllers stable in open-loop analysis, with a safety architecture based on redundancy and voters – are not analyzable in general at code or model level by any other tool we are aware of.

The objective of the approach is to enable the analysis of realistic aerospace critical software such as guidance and control-command systems, enabling the analysis of system-level specifications (stability, fault tolerance) on the actual implementation.

Future works will focus on a larger set of properties, *eg*. performance properties, and additional fault-tolerance constructs. We also consider studying more complex controllers, for example the combination of linear controllers switched according on the aircraft flight mode.

References

1. A. Adjé, S. Gaubert, and É. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In *ESOP*, 2010.
2. A. Champion and R. Delmas. Stuff: Stuff is the ultimate formal framework. https://cavale.enseeiht.fr/redmine/projects/stuff.
3. A. Champion, R. Delmas, and M. Dierkes. Generating property-directed potential invariants by backward analysis. In *FTSCS*, pages 22–38, 2012.
4. G. E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183. Springer, 1975.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM, 1977.
6. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–97. ACM Press, 1978.
7. Michael Dierkes. Formal analysis of a triplex sensor voter in an industrial context. In *FMICS*, pages 102–116, 2011.
8. E. Feron, G. Brat, P-L Garoche, P. Manolios, and M. Pantel. Formal methods for areospace applications. FMCAD 2012 tutorial.
9. P.-L. Garoche and P. Roux. SMT-AI: SMT abstract interpreter. https://cavale.enseeiht.fr/redmine/projects/smt-ai.
10. T. Gawlitza and H. Seidl. Computing relaxed abstract semantics w.r.t. quadratic zones precisely. In *SAS*, 2010.
11. T. Gawlitza, H. Seidl, A. Adjé, S. Gaubert, and E. Goubault. Abstract interpretation meets convex optimization. *J. Symb. Comput.*, 47(12), 2012.
12. Daniel Kästner, Stephan Wilhelm, Stefana Nenova, Patrick Cousot, Radhia Cousot, Jérôme Feret, Antoine Miné, Laurent Mauborgne, and Xavier Rival. Astrée: Proving the absence of runtime errors. In *ERTSS*, 2010.
13. A. Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO II*, volume 2053 of *LNCS*, pages 155–172. Springer-Verlag, 2001.
14. A. Miné. The octagon abstract domain. In *AST (satt. of WCRE)*, IEEE, pages 310–319, 2001.
15. D. Monniaux. Quantifier elimination by lazy model enumeration. In *CAV*, pages 585–599, 2010.
16. P. Roux, R. Jobredeaux, P.-L. Garoche, and E. Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*. ACM, 2012.
17. D. Rowell. Dicrete time observers and lqg control. MIT, Dpt. of Mechanical Engineering – 2.151 Advanced System Dynamics and Control – `http://web.mit.edu/2.151/www/Handouts/Kalman.pdf`, 2004.
18. J. Souyris and D. Favre-Félix. Proof of properties in avionics. In *Building the Information Society*, volume 156, pages 527–535. Springer, 2004.
19. A. Tarski. A decision method for elementary algebra and geometry: Prepared for publication with the assistance of j.c.c. mckinsey. Technical report, RAND Corporation, 1951.
20. C. Tinelli. Foundations of satisfiability modulo theories. In *WoLLIC*, page 58, 2010.