Formal Verification for Embedded Implementation of Convex Optimization Algorithms

Raphael Cohen, Guillaume Davy, Eric Féron, Pierre-Loïc Garoche

Formal Verification for Embedded Implementation of Convex Optimization Algorithms

Raphael Cohen [*,**] **Guillaume Davy** [**,***] **Eric Feron** [*]
Pierre-Loïc Garoche [**]

[*] *Georgia Institute of Technology, Atlanta, GA, USA*
[**] *Onera – The French Aerospace Lab, Toulouse, FRANCE*
[***] *CNRS-LAAS – Laboratory for Analysis and Architecture of Systems, Toulouse, FRANCE*

Abstract: Advanced embedded algorithms are growing in complexity and length, related to the growth in autonomy, which allows systems to plan paths of their own. However, this promise cannot happen without proper attention to the considerably stronger operational constraints that safety-critical applications must meet. This paper discusses the formal verification for optimization algorithms with a particular emphasis on receding-horizon controllers. Following a brief historical overview, a prototype autocoder for embedded convex optimization algorithms will be discussed. Options for encoding code properties and proofs, and their applicability and limitations will be detailed as well.

1. INTRODUCTION

The need for more safety and better performance is currently pushing the introduction of advanced numerical methods into next generations of cyber-physical systems. While most of the algorithms described in this paper have been known for a long time, their online use within embedded systems is relatively new and opens issues that have to be addressed. Among these methods, we are concerned specifically with numerical optimization algorithms.

Problem: These algorithms solve a constrained optimization problem, defined by an objective function – the cost function – and a set of constraints to be satisfied:

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & f_i(x) \leq b_i \text{ for } i \in [1, m] \end{array} \quad (1)$$

This problem searches for $x \in \mathbb{R}^n$ the optimization variable, minimizing $f_0 \in R^n \to \mathbb{R}$, the objective function, while satisfying constraints $f_i \in \mathbb{R}^n \to \mathbb{R}$, with associated bound b_i. An element $\mathbb{R}^n$ is feasible when it satisfies all the constraints f_i. An optimal point is defined by the element having the smallest cost value among all feasible points. An optimization algorithm computes an exact or approximated estimate of the value of the optimal cost, together with one or more feasible points achieving this value. A subclass of these problems can be efficiently solved: convex problems. In these cases, the functions f_0 and f_i are required to be convex and the optimal solution is unique. We refer the reader to Boyd and Vandenberghe (2004); Nocedal and Wright (2006) for more details on convex optimization.

Context: Recently this formalism has been used with great success for the guidance of safety-critical application. Such applications include autonomous cars Jerez et al. (2014) and reusable rockets Açikmese et al. (2013); Blackmore et al. (2012). The latter case has resulted in a spectacular experiments, including landings of SpaceX's Falcon 9 and BlueOrigin's New Shepard. Thus, powerful algorithms solving optimization problems do exist and are already used online. Although those algorithms have been embedded on board and running on an actual system, they still lack the level of qualification required by civil aircraft or manned rocket flight. computing a solution in a given time.

Algorithms: Let us consider the specific case of Linear Programming problems, a subclass of convex optimization. Multiple algorithm methods are available to solve them. First, the simplex method, which has been used successfully since the 1940s, despite its exponential worst-case complexity. Then the Ellipsoid algorithm and the very popular interior-point methods. These latter both exhibit polynomial complexity, though only interior point methods are considered to be practical for large desktop applications. These two methods are also applicable to more general settings such as quadratic programming (QP), affine constraints and quadratic cost, second order conic programming (SOCP), quadratic constraints and linear cost, or semidefinite programs (SDP).

When optimization algorithms are used offline, the soundness of their implementation and the feasibility of the computed optimizers is not as critical. Solutions could be validated *a posteriori* Roux (2015); Roux et al. (2016). This paper is concerned with online use of such algorithms, providing evidence of the *a priori* validity of the computed solution. This paper focuses on the linear programming paradigm and makes the following contributions:

- high level properties defining a sound optimization algorithm are formalized;
- these properties are expressed directly on the code artifact as annotations;

- the evidence supporting the algorithms properties is expressed at the code level
- the approach is demonstrated on the Ellipsoid algorithm and on an instance of the Interior Point method.

We believe this paper addresses a major certification issue that can considerably influence the use of online optimization methods in safety-critical applications.

Related work. Several authors including Richter et al. (2013), McGovern (2000); McGovern and Feron (1998) have worked on the certification problem of optimization algorithms for their online uses in control, in particular on worst-case execution time issues. In those cases, the authors have chosen to tackle the problem at a high level of abstraction.

Formally verifying high level properties of numerical algorithm implementations was performed on the other context of linear controller stability Feron (2010). This theoretical work was later instantiated concretely, generating code specifications and proving it with respect to the code Herencia-Zapana et al. (2012); Wang et al. (2016a). Some other work develop that approach for weaker properties Araiza-Illan et al. (2015); Pajic et al. (2015).

Our work follows a similar approach with respect to Wang et al. (2016b) in which interior-point method algorithms are annotated with convergence proof elements in order to show the soundness of the imperative implementation. This work however, remains theoretical, expressed in Matlab level, without any proof performed on the actual code.

Structure. The paper is structured as follows: Section 2 presents backgrounds for linear programming and axiomatic semantics using Hoare triples. Sections 3 and 4 focus on the two considered approaches: Interior Point method and Ellipsoid method. Section 5 presents our early proof results on the code artifact, while Section 7 concludes.

2. PRELIMINARIES

In order to support the following analysis, this section introduces the notions and notations used throughout the paper. First, we will discuss Linear Programming (LP) problems. Then we introduce axiomatic semantics and Hoare logic.

2.1 Linear Programming

Linear Programming is a class of optimization problems. A linear programming problem is defined by a matrix A of size $m \times n$, and two vectors b, c of respective size m, n.

Definition 1. Let us consider $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. We will write $P(A, b, c)$ as the linear program:

$$\min_{Ax\geq b} \langle c, x\rangle. \tag{2}$$

Definition 2. Let us consider the problem $P(A, b, c)$ and let us assume that an optimal point exists and is unique. We have the following definitions:

```
/*@ logic LMat mat_add(LMat x0, LMat x1);
  @ axiom getM_add: \forall LMat A, B; getM(
      mat_add(A, B) == getM(A);
  @ axiom getN_add: \forall LMat A, B; getN(
      mat_add(A, B) == getN(A);
  @ axiom get_add:
  @     \forall LMat A, B; (
  @         \forall integer i, j;
  @             mat_get(mat_add(A, B), i, j) ==
      mat_get(A, i, j)
  @               + mat_get(B, i, j)); */
```

Fig. 1. Axiomatization of matrix addition

- $E_f = \{x \mid Ax \geq b\}$ the feasible set of P,
- $f(x) = c^T x = \langle c, x\rangle$ the cost function and
- $x^* = \arg\min_{x\in E_f} f$ the optimal point.

2.2 Axiomatic semantics and Hoare Logic

Semantics of programs express their behavior. For the same program, different means can be used to specify it: (i) a denotational semantics, expressing the program as a mathematical function, (ii) an operational semantics, expressing it as a sequence of basic computations, or (iii) an axiomatic semantics. In the latter case, the semantics can be defined in an incomplete way, as a set of projective statements, ie. observations. This idea was formalized by Floyd (1967) and then Hoare (1969) as a way to specify the expected behavior of a program through pre- and post-condition, or assume-guarantee contracts.

Hoare Logic. A piece of code C is axiomatically described by a pair of formulas (P, Q) such that if P holds before executing C, then Q should be valid after its execution. This pair acts as a contract for the function and (P, C, Q) is called a Hoare triple. In most uses P and Q are expressed in first order formulas over the variables of the program. Depending on the level of precision of these annotations, the behavior can be fully or partially specified. In our case we are interested in specifying, at code level, algorithm specific properties such as the convergence of the analysis or preservation of feasibility for intermediate iterates.

Software frameworks, such as the Frama-C platform, *cf.* Cuoq et al. (2012), provide means to annotate a source code with these contracts, and tools to reason about these formal specifications. For the C language, ACSL, *cf.* Baudin et al. (2016) (ANSI C Specification language) can be used as source comments to specify function contracts, or local annotations such as loop invariants. Statement local annotations act as cuts in proofs and are typically required when analyzing loops.

Linear Algebra-based Specification. ACSL also provides means to enrich underlying logic by defining types, functions, and predicates. In its basic version, ACSL contains no predicate related to linear algebra. It is however possible to introduce such notions, supporting the expression of more advanced properties. Figure 1 presents the formalization of matrix addition.

These definitions are straightforward and fully defined. We also wrote a library for operators used in linear

```
/*@ logic real norm(LMat V, LMat X) =
  @     sqrt(mat_get(mat_mul(
  @          mat_mul(transpose(V), inv(hess(X)))
  @     , V), 0, 0)); */
```

Fig. 2. Hessian norm in ACSL.

optimization, defining $norm, grad, hess,$ LOWER.... Figure 2 presents the definition a Hessian-induced norm, parametrized by the Hessian at point X.

Lemmas can also be defined to support later analyzers to prove annotations.To preserve proof consistency, the Frama-C tool requires to prove these additional lemmas. For example, splitting a complex theorem into several small lemmas usually helps analyzers in proving the whole goal.

3. INTERIOR-POINT METHOD

We focus here on Interior Point methods, that, starting from a feasible solution, solve iteratively a sequence of linear problems, eg. using Newton methods, to reach the optimal solution. This method presented by Karmarkar (1984) and improved by Nesterov and Nemirovski (1988, 1994) is one the most efficient methods to solve linear and convex optimization problems. The presented work directly follows the theory presented in (Nesterov, 2004, $\mathbf{S}_4$). To keep the presentation simple and show the feasibility of our approach, we chose a primal method starting from the analytic center, whereas common methods are primal-dual and start from any point of the domain.
Let us introduce the major concepts before presenting the algorithm.

3.1 Basic concepts: Barrier functions and Central path

Barrier Function. Finding the optimum of a function is usually achieved by computing its derivative and searching for its zeros. In LP, we know that the optimum value is located on the border of the feasible set, which could be problematic to reach numerically. To solve this issue, we add a barrier function F to the cost function. The rationale of the barrier function $F(x)$ is to diverge when x get closer to the border of the feasible set.

In order to guarantee convergence bounds, the barrier function must be a Lipschitz function. In practice, a self-concordant barrier function Nesterov and Nemirovski (1989) satisfies this requirement. The typical choice is a sum of logarithmic functions.

Definition 3. (Log barrier function). Let F be the barrier function associated to problem P. We have:

$$F(x) = \sum_{i=1}^{m} -ln(\langle a_i, x\rangle - b_i) \tag{3}$$

where a_i is the i-th row of A.

Definition 4. (Analytic center). We define the analytic center as the point x_F^* verifying:

$$x_F^* = \arg\min_{x \in E_f} F(x) \tag{4}$$

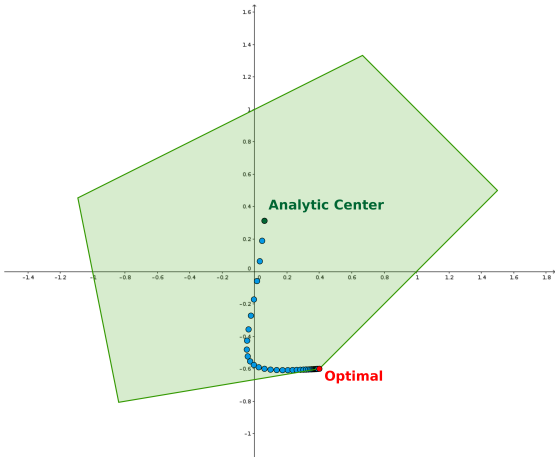


Fig. 3. In blue: Points of the central path for different value of t

Central path. Using both the barrier function and the initial cost function, one can characterize their linear combination.

Definition 5. Let us introduce a new function, $\widetilde{f}$ as:

$$\widetilde{f}(t, x) = tf(x) + F(x) \tag{5}$$

In this function, t is the parameter that weights the relative importance of the initial cost function with respect to the barrier function. When $t = 0$ the original cost function vanishes and minimizing $\widetilde{f}$ leads to the analytic center while when $t \to +\infty$ the cost function is predominant and minimizing $\widetilde{f}$ leads to the optimal point up to ϵ (Figure 3)

Definition 6. (Central path). The central path is defined as the curve:

$$x^*(t) = \arg\min_{x \in E_f} \widetilde{f}(t, x) \text{ for } t \in [0, +\infty) \tag{6}$$

Interior-point methods are part of the path-following optimization methods: starting from the analytic center and following the central path to reach the optimal point, as illustrated in Figure 3.

Approximate Centering Condition (ACC). Since the algorithm is doing numerical computation, the computed points cannot be located exactly on the central path. Thus, we need to introduce an *approximate centering condition*:

Definition 7. (ACC). Let us define $\lambda(t, x)$ as:

$$\lambda(t, x) = \widetilde{f}'(t, x)^T F''(x) \widetilde{f}'(t, x) \tag{7}$$

The approximate centering condition becomes:

$$\lambda(t, x) < \beta^2 \tag{8}$$

Notice that $F''(x)$ induces a norm [1] so λ measures how close to 0 the derivative is.

This definition aims to guarantee that the points are close to the central path so that the last point can be close enough to the optimal:

Theorem 1. (*cf. (Nesterov, 2004,* $\mathbf{S}_4$*)*). If $\lambda(t, x) < \beta^2$ then $\langle c, x\rangle - \langle c, x^*\rangle \leq \frac{\Delta}{t}$ with $\Delta = 1 + \frac{(\beta+1)\beta}{1-\beta}$

[1] By construction, thanks to the use of a self-concordant barrier function.

```
/*@ requires acc(0, MatVar(X,2,1));
  @ ensures dot(c, MatVar(X,2,1))-dot(c,sol(A,b,c
    ))<EPSILON;
  @ ensures A * MatVar(X,2,1) > b; */
void pathfollowing() {
    t = 0;
    /*@ loop invariant ensures acc(t, MatVar(X, 2
        , 1));
        loop invariant A * MatVar(X, 2, 1) > b;
        loop invariant ensures t > LOWER(l); */
    for (unsigned int l = 0;l<NBR;l++)
    {
        compute_pre();
        compute_dt();
        compute_dx();
        t = t + dt;
        for(unsigned int i = 0;i<N;i++)
            X[i] = X[i] + dx[i];
    }
}
```

Fig. 4. Shape of the path following algorithm

While t increases, computed points converge to the optimal. The approximate centering condition is the invariant that we maintain along the iterations of the algorithm.

3.2 Algorithm

Since we are targeting embedded systems, we will only use static variables, a common practice in embedded systems. We also define macros representing the different values of the problems, simplifying the definition of annotations.

Figure 4 presents the main function of the algorithm, computing the next iterate while following the central path. It relies on:

- `acc`, `dot` and `sol`, ACSL self-defined operators representing, the approximate centering condition, the dot product and the solution of $P(A, b, c)$ respectively.
- several constants representing:
  - `EPSILON`: the desired precision
  - `M` x `N`: the size of the matrix
  - `LOWER`, a function associating a lower bound for t at each iteration .
  - `NBR`: the number of iterations required so that $\text{LOWER}(NBR) > \frac{\Delta}{\epsilon}$
- subfunctions `compute_pre`, `compute_dt`, `compute_dx`

In this program, we need to prove that at the end of the loop we ensure :

$$dot(c, x) - dot(c, sol(A, b, c)) < EPSILON.$$

This is proved thanks to a lemma we call `acc_solution` and the main function loop invariants. From the first loop invariant and `acc_solution`, one deduces that $dot(c, x) - dot(c, sol(A, b, c)) < \frac{\Delta}{t}$ and from the second loop invariant, that $t > \frac{\Delta}{\epsilon}$, leading to the expected post-condition.

We will now detail the content and contracts of the three functions used by the main one: `compute_dt`, `compute_dx` and `compute_pre`.

Function `compute_pre` computes the gradient $F'(X)$ and the hessian $F''(X)$ of F at point X according to the Equs. 9 and 10.

```
/*@ requires acc(t, MatVar(X,2,1));
  @ ensures MatVar(H,2,2) == hess(MatVar(X,2,1));
  @ ensures dt > t*0.125;
  @ ensures dt = (5/36)/norm(c,MatVar(X,2,1));
  @ */
void compute_dt();
```

Fig. 5. Contract on `compute_dt`

$$(G)_j(X) = \sum_{i=0}^{m-1} -\frac{a_{i,j}}{\langle a_i, X\rangle - b_i} \tag{9}$$

$$(H)_{j,k}(X) = \sum_{i=0}^{m-1} \frac{a_{i,j}a_{i,k}}{\langle a_i, X\rangle - b_i} \tag{10}$$

Function `compute_dt` computes the update of dt according to Eq. 11. Following Nesterov (2004), we have

$$dt = \frac{5}{36} \cdot \frac{1}{c^T \times H \times c} \tag{11}$$

We can deduce that $dt > 0.125t$. From this, we can lower bound t by a geometric progression: $\text{LOWER}(n) = t_0 \times (1.125)^n$. This allow us to determine the fixed number of iterations required to reach a given level of precision.

Function `compute_dx` updates X using a Newtonian iteration on the derivative of F:

$$dx = H^{-1}(tc + G) \tag{12}$$

Notice that t in this formula is the one already updated $(t + dt)$. We expect from this update that the approximate centering condition remains valid.

4. THE ELLIPSOID METHOD

The other method that has attracted our attention is the Ellipsoid Method. Despite its relative efficiency with respect to Interior Point methods, the Ellipsoid method benefits from concrete proof elements and could be considered viable option for critical embedded systems where safety is more important than performance. Before recalling the main steps of the algorithm, the needed elements will be presented.

4.1 Preliminaries

Ellipsoids in $\mathbb{R}^n$. An ellipsoid can be characterized by a positive definite matrix. A symmetric matrix P of $\mathbb{R}^{n\times n}$ is positive definite ($\mathcal{S}_n^+$) when

$$x^T P x > 0 \ , \ \forall x \neq 0 \in \mathbb{R}^n$$

Furthermore, this definition implies that all matrices P in $\mathcal{S}_n^+$ are invertible.

Definition 8. (Ellipsoid Sets). Let $x \in \mathbb{R}^n$ and $P \in \mathbb{R}^{n\times n}$ positive-definite, we denote the Ellipsoid $E(P, x)$ the set :

$$E(P, x) = \{z \in \mathbb{R}^n : (z - x)^T P^{-1}(z - x) \leq 1\}$$

Definition 9. (Euclidean ball). Let $n \in \mathbb{N}$ we denote V_n the unit Euclidean ball in $\mathbb{R}^n$. $\text{Vol}(V_n)$ denotes its volume.

In addition, we define $B(x, R)$ as the ball of radius R centered on x.

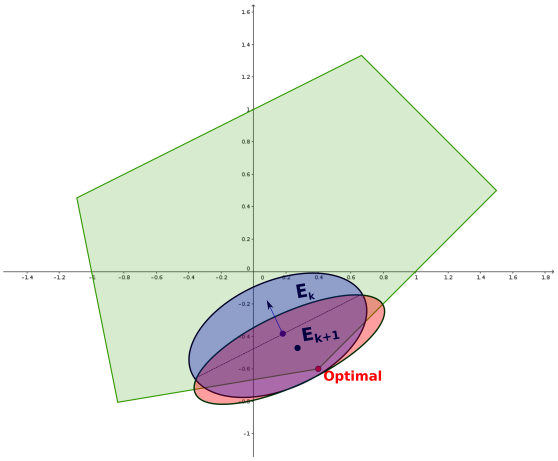


Fig. 7. Ellipsoid Cut for an iteration applied to LP

Definition 10. (Volume of ellipsoids). Let $E(P, x)$ be an ellipsoid set in $\mathbb{R}^n$. We denote by $Vol(E)$ its volume defined as

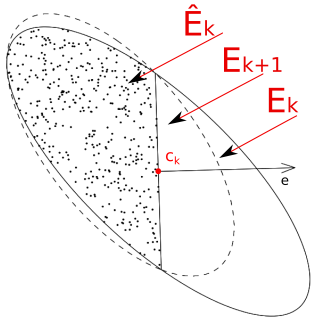
$$\text{Vol}(E(P,x)) = \text{Vol}(E) = \sqrt{det(P)} \cdot \text{Vol}(V_n)$$

4.2 Algorithm

Let us now recall the main steps of the algorithm detailed in Bland et al. (1981); Grötschel et al. (1981); Khachiyan (1980); Boyd and Barratt (1991).

Remark 1. In the following, we denote E_k, x_k and P_k, the ellipsoid, the corresponding center and positive-definite matrix, respectively computed by the algorithm at the $k-th$ iteration.

Algorithm. We start the algorithm with an ellipsoid containing the feasible set E_f, and therefore the optimal point x^*. We iterate by transforming the current ellipsoid E_k into a smaller volume ellipsoid E_{k+1} that also contains x^*. Given an ellipsoid E_k of center x_k, we find a hyperplane containing x_k that cuts E_k in half, such that one half is known not to contain x^*. Finding such a hyperplane is called the *oracle separation* step, *cf.* Ben-Tal and Nemirovski (2004). In our LP setting, this step is not computationally expensive. Then, we define the ellipsoid E_{k+1} by the minimal volume ellipsoid containing the half ellipsoid $\hat{E}_k$ that is known to contain x^*. In addition to that, we can compute an upper bound γ of the ratio of $\text{Vol}(E_{k+1})$ to $\text{Vol}(E_k)$. The Figures 6 and 7 illustrates such ellipsoids cuts.



e representing the vector normal to the chosen cutting hyperplane from oracle separation.

Fig. 6. Ellipsoid Cut for an iteration

Ellipsoid transformation. From the oracle separation step, a separating hyperplane, e, that cuts E_k in half with the guarantee that x^* is localized in $\hat{E}_k$ has been computed. The following step is the *Ellipsoid transformation*. Using this hyperplane e, one can update the Ellipsoid E_k to its next iterate E_{k+1} according to equations 13 and 14.

$$x_{k+1} = x_k - \frac{1}{n+1} P_k \tilde{e} \tag{13}$$

and

$$P_{k+1} = \frac{n^2}{n^2-1}(P_k - \frac{2}{n+1} P_k \tilde{e}\tilde{e}^T P_k) \tag{14}$$

with: $\tilde{e} = \frac{e}{e^T P_k e}$.

One can now characterize the upper bound of the ratio of $\text{Vol}(E_{k+1})$ to $\text{Vol}(E_k)$.

Property 1. (Reduction ratio). Let $k \geq 0$ be an iteration of the algorithm. We have

$$Vol(E_{k+1}) \leq \exp(\frac{-1}{2 \cdot (n+1)}) \cdot Vol(E_k)$$

Hypotheses. We recall that we assumed the matrix A and the vector b are such that the feasible set E_f is bounded and not empty. Additionally, further information about E_f is needed. Indeed, in order to know the number of steps required for the algorithm to return an ϵ-optimal solution, two scalars are needed:

- a radius R such that E_f is included in $B(\mathbf{0_n}, R)$
- another scalar r such that there exist a center c_1 such that $B(c_1, r)$ is included in E_f.

The main result can be stated as:

Theorem 2. Let us assume that E_f is bounded, not empty and such that R, r are known. Then, for all $\epsilon > 0$, $c \in \mathbb{R}^n$ there exists a N such that the algorithm, using N iterations, will return $\hat{x}$, such that

$$f(\hat{x}) \leq f(x^*) + \epsilon \text{ and } \hat{x} \in E_f \text{ } (\epsilon \text{ optimal})$$

4.3 Algorithm's Structure and Annotations

Figure 8 presents an annotated version of the implementation. Annotations combine ellipsoidal constraints `inEllipsoid` and volume related properties to express the progress of the algorithm along iterates.

5. TOWARD PROVING ANNOTATION

FramaC supports the analysis of C code with ACSL annotations through multiple analyzers. The tool WP projects both the ACSL specification – the Hoare triples – and the source code in a predicate encoding on which external tools can be applied.

Each function contract, or statement assertion, becomes a goal, a proof objective. This goal can be solved either in an automatic fashion by SMT solvers such as Alt-Ergo, *cf.* Conchon et al. (2012), Z3, *cf.* de Moura and Bjørner (2008), or thanks to manual proofs supported by proof assistants such as Coq, *cf.* The Coq development team (2012).

```
/*@ requires inEllipsoid(Ellipsoid(
    P_minus,x_minus),sol(A,b,c));
  @ ensures dot(c,MatVar(X,2,1))-dot(c,sol(A,b,c)
      ) <= EPSILON;
  @ ensures A * MatVar(X,2,1) <=  b; */
void ellipsoidMethod() {
  long double P_minus[N*N],x_minus[N];
  long double x_best[N],grad[N];
  int i,Nit;
  initializationEllipsoid(P_minus,x_minus,N,R);
  affectationVector(x_best, x_minus, N);
    /*@ loop invariant Vol(Ellipsoid(
        P_minus,x_minus)) <= gamma^i * R^N ;
      @ loop invariant inEllipsoid(Ellipsoid(
          P_minus,x_minus),sol(A,b,c)) ;
      @ loop invariant isMoreOptimal(
          x_best,x_minus) ; */
  for (i = 0; i < NBR; ++i) {
    updateBest(x_best, x_minus);
    getGrad(x_minus, grad);
    updateEllipsoid(P_minus, x_minus, grad);
  }
  affectationVector(x, x_best, N);
}
```

Fig. 8. Shape of the Main Ellipsoid Method Algorithm

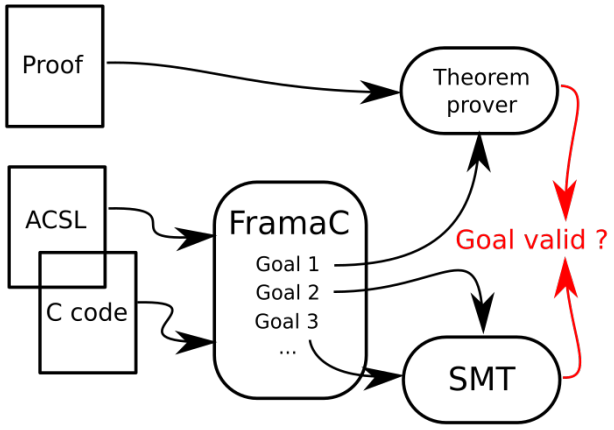


Fig. 9. Automatic Verification Framework

5.1 Low level matrix operation

We used this toolchain to prove that the annotations are valid. The goal is to have as many annotations as possible proven by SMT solvers since a Coq proof requires additional work. One of the difficulties in Coq is to address low level C manipulations such as matrix operations. The memory model axiomatization makes the manipulation of these encodings difficult while the properties are not specifically hard to prove. To avoid this, all matrix operations in C are associated to an ACSL contract, stating the computation performed in linear algebra instead of arrays and pointers. Fig. 10 presents an example of these annotations while Sec. 8 describes the (simplified) generated goal. SMT solvers easily discard this kind of proof objectives.

5.2 Lemmas hierarchy

Once these matrix operations have been abstracted by their ACSL axiomatization, all that remains is to prove the regular annotation.

```
/*@ ensures MatVar(X, 2, 1) == \old(
  @ mat_add(MatVar(X, 2, 1),
  @ MatVar(dx, 2, 1))); */
{
    X[0] = X[0] + dx[0];
    X[1] = X[1] + dx[1];
}
```

Fig. 10. Simple example of matrix operation specification

For each postcondition Q, we find what precondition $P_1, ..P_k$ is required to prove Q. We also extract all variables $v_1, ..v_l$ appearing in Q and $P_1, ..P_k$. With all this information we write an ACSL lemma stating : $\forall v_1, ..v_l, P_1 \Rightarrow P_2 \Rightarrow \ ... \ \Rightarrow P_k \Rightarrow Q$.

Assuming this lemma an SMT will prove easily the postcondition. This permits to remove all C code aspect, to have pure math results to proof.

To ease the proof of this lemmas, one can build intermediary lemmas. This lemmas premits to split the proof into small pieces. Then each pieces can be proved by a SMT solver. We represented as a dependance graph in Fig. 5.2 the current status of the proof. This depency graph is used to tell the SMT which lemmas is useless to prove a given lemmas. This permits to remove a lot of lemmas from the context of the SMT simplifying their task.

We considered some results as Axioms(Blue colored in the figure). Deciding if a results should be an axiom is difficult. For now we decided that all math results we can find in (Nesterov, 2004, $\mathbf{S}_4$) should be axioms. This permits to be clear on what we admited or not. Moreover results in Nesterov are usually quite general but remain feasible for SMT.

Maybe at some point, SMT wont be powerfull enough to prove the lemmas. If this happens we can write the proof by hand using a proof assistant as Coq. An other possibility would be to use machine learning to build the proofs, see e.g. Kaliszyk and Urban (2015).

6. FLOATING-POINT CONSIDERATIONS

Another issue concerns the necessary rounding on the variables that happens due to the floating point representation of the variable and the impact of this numerical imprecision on the validity of the algorithm implementation.

In the following, we denote by $\mathbb{F}$ the set of floating point numbers. Since the sum of two floats is not necessarily a float, each computation will generate noise. One can still bound such numerical errors using the so-called standard model Rump (2006); Roux et al. (2012). This analysis can either be performed symbolically, analyzing the compuation performed and summing up the accumulated errors. Another approach, eg. implemented in the tool Fluctuat Delmas et al. (2009), will rely on affine or interval arithmetics to represent such acucmulated errors of the standard models and bound the overall error obtained.

Problem Formulation: Ellipsoid method In the Ellipsoid method algorithm, we iterate the center point and matrix x, P such that the ellipsoid $\text{Ell}(x, P)$ has a decreasing volume and includes the optimal point.

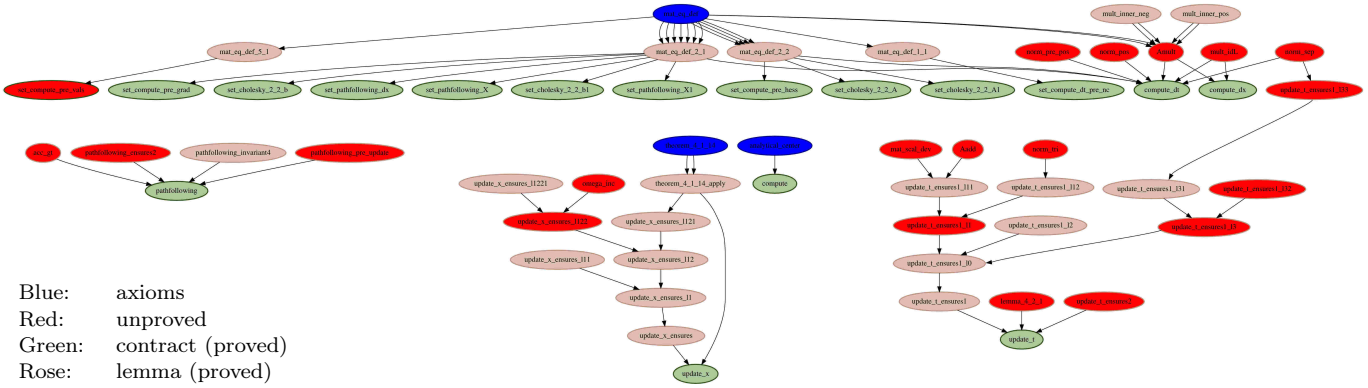


Fig. 11. Lemma proof hierarchy

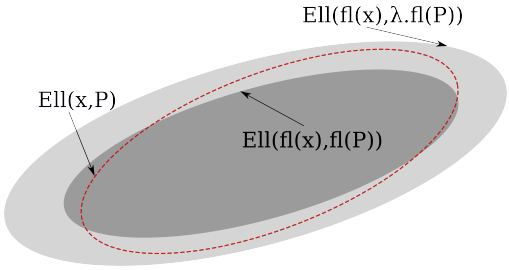


Fig. 12. Ellipsoid Widening

The operations being done in $\mathbb{F}$ and not $\mathbb{R}$, we have no guarantee that the optimal point lies within the ellipsoid at each iteration (see figure 12). Therefore, this last condition being an important aspect of the program's semantics, the errors due to floating point rounding could have a direct impact on its stability.

Let $\mathrm{fl}(x)$ and $\mathrm{fl}(P)$ be, respectively, the floating point values of x and P computed and used in the algorithm. x and P being the values in $\mathbb{R}$ obtained if the computations were not rounded. The objective here is to find a $\lambda > 1$ such that:

$$\mathrm{Ell}(x, P) \subset \mathrm{Ell}(\mathrm{fl}(x), \lambda \cdot \mathrm{fl}(P))$$

By doing that, we widen the ellipsoid $\mathrm{Ell}(x, P)$, as illustrated on Fig. 12, to take into account the rounding. The convergence issue becomes a problem when reaching the optimal value, ie. the need to prove the decreasing volume of the ellipsoids. Thus, the widening coefficient λ cannot be too big. This last constraint is currenlty under investigation.

7. CONCLUSION

In this article, annotations for numerical algorithms solving linear programming problems were proposed. We focused on a primal Interior Point algorithm and on the Ellipsoid method. The Interior Point algorithm is fully automated annotated while annotations for the Ellipsoid method are still manual. Annotations can be partitioned in two levels:

- Low-level annotation specifying matrix operations. This separates the handling of memory array accesses from math reasoning, easing the automatic proof of simpler annotations by SMT solvers.
- Math reasoning annotations specify the high level properties of the algorithms: convergence, preservation of feasibility, ϵ-optimality of the solution, etc. These annotations are proven either by SMT solvers or with Coq. The finalization of these proofs is still a work in progress.

Our perspectives include the application of this approach to more general settings, eg. generic linear programming problems. We plan to couple code generation with annotations and proof synthesis, providing embedded code together with its functional soundness proof, at the code level. This would include the generation of the corresponding Coq proofs.

Our approach is also compatible with more general convex optimization problems. Finally, we intend to address floating point semantics issues while performing the proofs. This would guarantee a completely sound and bug-free implementation.

ACKNOWLEDGEMENTS

This work was partially supported by projects ANR ASTRID VORACE and NSF CPS SORTIES under grant 1446758. The authors would also like to deeply thank Didier Henrion for his participation to this work.

REFERENCES

Açikmese, B., Carson, J.M., and Blackmore, L. (2013). Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Trans. Contr. Sys. Techn.*, 21(6), 2104–2113.

Araiza-Illan, D., Eder, K., and Richards, A. (2015). Verification of control systems implemented in simulink with assertion checks and theorem proving: A case study. 2670–2675.

Baudin, P., Filliâtre, J.C., Marché, C., Monate, B., Moy, Y., and Prevosto, V. (2016). ACSL: ANSI/ISO C Specification Language. version 1.11.

Ben-Tal, A. and Nemirovski, A. (2004). Lecture notes, optimizattion i-ii, convex analysis, non-linear programming theory, non-linear programming algorithms.'.

Blackmore, L., Açikmese, B., and Carson, J.M. (2012). Lossless convexification of control constraints for a class of nonlinear optimal control problems. *Systems & Control Letters*, 61(8), 863–870.

Bland, R.G., Goldfarb, D., and Todd, M.J. (1981). The ellipsoid method: A survey. *Operations research*, 29(6), 1039–1091.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Boyd, S.P. and Barratt, C.H. (1991). *Linear controller design: limits of performance*. Prentice Hall.

Conchon, S., Contejean, E., and Iguernelala, M. (2012). Canonized rewriting and ground ac completion modulo shostak theories : Design and implementation. *Logical Methods in Computer Science*, 8(3).

Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., and Yakobowski, B. (2012). Frama-c: a software analysis perspective. SEFM'12, 233–247. Springer.

de Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *TACAS*, 337–340.

Delmas, D., Goubault, E., Putot, S., Souyris, J., Tekkal, K., and Védrine, F. (2009). *Towards an Industrial Use of FLUCTUAT on Safety-Critical Avionics Software*, 53–69. Springer.

Feron, E. (2010). From control systems to control software. *Control Systems, IEEE*, 30(6), 50 –71.

Floyd, R.W. (1967). Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19, 19–32.

Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2), 169–197.

Herencia-Zapana, H., Jobredeaux, R., Owre, S., Garoche, P.L., Feron, E., Perez, G., and Ascariz, P. (2012). Pvs linear algebra libraries for verification of control software algorithms in c/acsl. In *NFM'12*, volume 7226 of *LNCS*, 147–161. Springer.

Hoare, C.A.R. (1969). An axiomatic basis for computer programming. *Commun. ACM*, 12, 576–580.

Jerez, J.L., Goulart, P.J., Richter, S., Constantinides, G.A., Kerrigan, E.C., and Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Trans. Automat. Contr.*, 59(12), 3238–3251.

Kaliszyk, C. and Urban, J. (2015). Learning-assisted theorem proving with millions of lemmas. *J. Symb. Comput.*, 69, 109–128.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4), 373–395.

Khachiyan, L.G. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1), 53–72.

The Coq development team (2012). *The Coq proof assistant reference manual*. LogiCal Project. Version 8.4.

McGovern, L.K. (2000). *Computational Analysis of Real-Time Convex Optimization for Control Systems*. Ph.D. thesis, Massachussetts Institute of Technology.

McGovern, L.K. and Feron, E. (1998). Requirements and hard computational bounds for real-time optimization in safety-critical control systems. In *CDC'98*, volume 3, 3366–3371.

Nesterov, Y. (2004). *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ.

Nesterov, Y. and Nemirovski, A. (1988). A general approach to the design of optimal methods for smooth convex functions minimization. *Ekonomika i Matem. Metody*, 24, 509–517.

Nesterov, Y. and Nemirovski, A. (1989). *Self-Concordant functions and polynomial time methods in convex programming*. Materialy po matematicheskomu obespecheniiu EVM. USSR Academy of Sciences, Central Economic & Mat hematic I nstitute.

Nesterov, Y. and Nemirovski, A. (1994). *Interior-point Polynomial Algorithms in Convex Programming*, volume 13 of *Studies in Applied Mathematics*. Society for Industrial and Applied Mathematics.

Nocedal, J. and Wright, S.J. (2006). *Numerical Optimization*. Springer, 2nd edition.

Pajic, M., Park, J., Lee, I., Pappas, G., and Sokolsky, O. (2015). Automatic verification of linear controller software. 217–226.

Richter, S., Jones, C.N., and Morari, M. (2013). Certification aspects of the fast gradient method for solving the dual of parametric convex programs. *Mathematical Methods of Operations Research*, 77(3), 305–321.

Roux, P. (2015). Formal proofs of rounding error bounds. *Journal of Automated Reasoning*, 1–22.

Roux, P., Jobredeaux, R., Garoche, P.L., and Féron, E. (2012). A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC'12*, 105–114. ACM.

Roux, P., Voronin, Y.L., and Sankaranarayanan, S. (2016). Validating numerical semidefinite programming solvers for polynomial invariants. In Springer (ed.), *SAS'16*, LNCS.

Rump, S. (2006). Verification of positive definiteness. *BIT Numerical Mathematics*, 46(2), 433–452.

Wang, T., Jobredeaux, R., Herencia-Zapana, H., Garoche, P.L., Dieumegard, A., Feron, E., and Pantel, M. (2016a). *From Design to Implementation: An Automated, Credible Autocoding Chain for Control Systems*, volume 460 of *LNCIS*, 137–180. Springer.

Wang, T., Jobredeaux, R., Pantel, M., Garoche, P.L., Feron, E., and Henrion, D. (2016b). Credible autocoding of convex optimization algorithms. *Optimization and Engineering*. To appear.

8. APPENDIX

Example of a generated goal for basic matrix operation:

```
goal set_X_post:
  let a = shift_A2_float64(global(G_X_1018), 0) : addr in
  let a_1 = shift_float64(a, 0) : addr in
  let a_2 = shift_float64(a, 1) : addr in
  let a_3 = shift_A2_float64(global(G_dx_1019), 0) : addr
     in let a_4 = shift_float64(a_3, 0) : addr in
  let a_5 = shift_float64(a_3, 1) : addr in
  forall t : (addr,real) farray.
  let r = t[a_1] : real in
  let r_1 = t[a_2] : real in
  let r_2 = t[a_4] : real in
  let r_3 = t[a_5] : real in
  let a_6 = L_mat_add(L_MatVar(t, a, 2, 1), L_MatVar(t, a_3
     , 2, 1)) : A_LMat in
  let a_7 = L_MatVar(t[a_1 <- r + r_2][a_2 <- r_1 + r_3],
     a, 2, 1) : A_LMat in
  is_float64(r) → is_float64(r_1) →
  is_float64(r_2) → is_float64(r_3) →
  (1 = L_getN(a_6)) → (2 = L_getM(a_6)) →
  (1 = L_getN(a_7)) → (2 = L_getM(a_7)) →
  (L_mat_get(a_6, 0, 0) = L_mat_get(a_7, 0, 0)) →
  (L_mat_get(a_6, 1, 0) = L_mat_get(a_7, 1, 0)) →  (a_6 =
     a_7)
```