Generation of polynomial inequalities as invariants

Yannick Zakowski

Master research Internship

Internship report

Generation of polynomial inequalities as invariants

Author:
Yannick Zakowski

Supervisor:
David Cachera
Celtique team

Abstract

Embedded software in critical systems rise a need for software analysis, especially for guaranteeing safety properties. In the late seventies, Cousot & Cousot introduced a general framework, called abstract interpretation, dedicated to the conception of particular analyses: static analyses. Among the program properties of interest, discovering algebraic relationships between variables allows for proving the lack of run-time errors. While the inference of linear relationships is efficiently solved, computing polynomial equalities as invariants is still a challenge, *a fortiori* is the inequality counterpart. After a brief overview of existing techniques for invariant inference, this report investigates the viability of a constraint-based method dedicated to the inference of polynomial inequalities as program invariants.

Contents

1 Introduction

During the last decades, embedded software took place in countless industries. Consequently, bugs grew potentially costly, or even life threatening: flight controllers, automotive brakes or even nuclear safety systems constitute critical environments where software has gained a prominent room. Hence, craving for software reliability grew alongside.

Software reliability can be ensured by differents methods. The presumably oldest and currently most used in industry is testing. Nevertheless, such a method can do no more than increasing our confidence in the tested software, while we would like to have access to a formal, mathematical, proof of correctness. Unfortunately, this problem has proven to be undecidable. We therefore tighten our expectation: we seek an analysis which exclusively accepts safe programs, even if this implies some safe programs to be rejected.

In the late seventies, Cousot and Cousot laid the fundation of the domain by introducing a general framework called Abstract Interpretation [14]. This framework formalizes the notion of property and its over approximation, by defining the adequate mathematical structures. Not only does this framework permit one to deeply understand the problem of static analysis, it also cuts up the problem of developing a static analysis into several recurrent subproblems. The community of software static analysis can therefore develop algorithms and ideas reusable in numerous analyses.

We focus in this work on particular properties, namely numerical properties between variables. Both non-relational as well as linear properties are quite well handled by existing analyses. On the other hand, the non-linear case remains a difficult task when it comes to inference.

In this report, we focus on program invariants expressed as polynomial inequalities. While no satisfying solutions exist at the moment, a few trails of extensions to polynomial inequalities as invariants have been suggested in works restricted to lower degrees or easier properties than invariants. We therefore investigated those works, and more specifically a constraint based approach by Cousot [13]: invariance is characterized by constraints, which we progressively relax by weaker ones until reaching a system we are able to efficiently solve.

While this process has originally been developed for an easier task, ranking function inference, its extension to invariant inference has been suggested by Cousot. In this report, our contributions on this subject are the following: we investigated the viability of the extension, both through theoretical insights and experiments; we identified the main challenges as the necessity of getting rid of bilinearity in the constraints and the task of guiding the solver toward semantics relevancy; we suggested trails toward the resolution of these issues.

The remaining of this report is organized as follows. A first section is dedicated to background in invariant inference. We then more specifically

focus on the work we investigated, by exposing the successive relaxations allowing us to abstract the semantics as a set of constraints. Section 4 presents our analysis of the problem, coupled with some background in mathematical programming, as well as the experimental limitations we encountered. Finally, we present a few interesting related works in Section 5, before concluding.

2 Overview of invariant inference

2.1 The static analysis framework

The purpose of a static analysis is to compute a program property without executing it. In order to facilitate this work, Cousot and Cousot introduced in 1977 a mathematical background [14], partially systematizing the conception of the analysis by identifying recurrent subproblem, thus allowing generic solutions. We informally introduce this framework through this section.

2.1.1 Programming model

Analyses we consider act on programs. We therefore need a model to formally represent a generic language. The model used by authors slightly varies from one paper to another, hence we now introduce for this section a model general enough to easily introduce several works here. A more specific model dedicated to our main object of study will be presented in the next section.

A program is modeled as a graph in which nodes represent program points (or *Locations*). Assignments are represented by an edge labeled with the assignment, while conditionals require two different edges starting from the current node, labeled respectively with the test and its negation. Similarly, a loop induces a back-edge in the graph.

2.1.2 An over-approximated world

We consider a program involving n variables, taking their values among the reals; we define a state as a vector of $\mathbb{R}^n$, representing the memory at a given time during execution. For convenience, we will write $\mathbf{x}$ for vectors of reals or variables, while writting x for unique variables or reals.

We would like to extract some information about this program, without executing it. More specifically, we want to safely establish numerical properties over variables which are invariant at a given program point: they hold through any execution.

In order to do so, we would ideally like to compute, given the set of possible initial memory states, the set of possible states at a given program

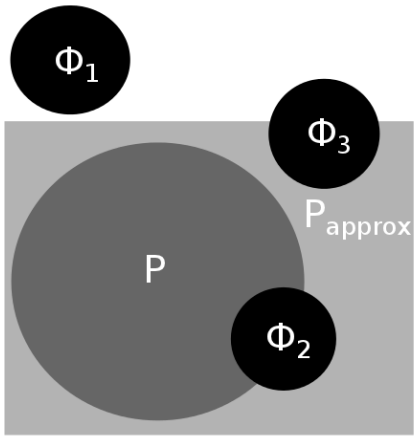


Figure 1: A property P, its approximation P_{approx} and some erroneous sets of states Φ_i. The analysis correctly claims the program to be safe to Φ_1 and unsafe to Φ_2, but erroneously qualifies it as unsafe to Φ_3.

point: this is known as the collecting semantics. On our programming model, defining the collecting semantics consists in following the flow defined by the graph, and iteratively collecting information from each instruction in order to build a function from *Locations* to $\mathcal{P}(\mathbb{R}^n)$. A first challenge arises from conditionals: they appear as two branches in the graph, which reach a common program point. We therefore need to be able to collect the information grabbed from each branch. Furthermore, we have to collect the inductive information induced by a loop: this is solved by computing a fixpoint, *i.e.* a set of states stable for the semantics of the loop body.

Unfortunately, the collecting semantics is known to be not computable. More generally, if we refer to a property as a subset $\mathcal{K} \in \mathbb{R}^n$, static analysis is haunted by Rice's theorem: no non-trivial property on programs semantics is computable. We therefore seek for a weaker analysis through overapproximation. The main idea is to restrict our computations on a fixed subset of properties, referred to as a domain. Since the program semantics, as a subset of $\mathbb{R}^n$, may not directly belong to this domain, we look for a domain element which overapproximates this semantics. The smallest this approximation is, the most precise information we have. The interest of such an analysis is that it may be computable. But as illustrated on Figure 1, it eventually raises false positives. Note however that it can't qualify as safe a program which is not.

Example 1 (The interval domain)**.** *One might be satisfied with the only extrem bounds of the possible values of variables: indeed, it may be a precise*

enough information to prove the absence of memory overflows or out-of-bounds array indices. Such an information can be gathered through the interval domain: the analysis computes, for each program point i and variable x, an interval on $\mathbb{R}$ which gives an over-approximation of all the possible values of x at i.

2.1.3 A world with an adequate behavior: the lattice structure

We informally suggested in the previous subsection to work on a subset of properties, and hope them to be actually computable. Indeed, we need a way to qualify such a subspace. As stated above, we face two intuitive problems when computing our semantics:

- in presence of a conditional, our graph presents a branching which joins farther. Hence we must have a process which, given two elements of our domain, computes one domain element that subsumes them;
- when dealing with a loop, the collecting semantics follows a loop in the graph. We must therefore be able to gather all the information by computing a fixpoint.

The adequate mathematical structure is that of a complete lattice.

Definition 1 (Complete lattice)**.** *A complete lattice is given by* $(A, \sqsubseteq, \bigsqcup)$ *where A is a set supplied with:*

- *a partial order relation* $\sqsubseteq$
- *a least upper bound* $\bigsqcup$*:* $\forall S \subseteq A,\ \bigsqcup S \in A$ *is such that:*
  - $\forall x \in S,\ x \sqsubseteq \bigsqcup S$
  - $\forall x,\ (\forall y \in S,\ y \sqsubseteq x) \Rightarrow \bigsqcup S \sqsubseteq x$

Complete lattices grant any monotonic function with a least fixpoint, defined as $\bigsqcup_{n \in \mathbb{N}} \mathcal{F}^n(\bot)$, where $\bot$ is the least element. Hence, we have an algorithm, known as the *Kleene iteration* (whose termination however is not guaranteed), to compute a fixpoint: starting from $\bot$, iterate $\mathcal{F}$ until stabilization.

Example 2 (The intervals lattice)**.** *The specific subset of properties defined by products of intervals on $\mathbb{R}$ actually possesses the desired structure:*

- $[a, b] \sqsubseteq [c, d]$ if $c \leq a \wedge d \geq b$.
- $\bigsqcup\{[a_i, b_i]\} = [\inf a_i, \sup b_i]$

2.1.4 An abstract world for convenience and efficiency

Theoretically, computing directly on a restricted set of subsets of properties as a domain would be possible. However, it happens to be highly more practical to work on an abstraction of this domain, in order to gain convenience as welle as mathematical properties. Let us highlight the idea through an example.

Example 3. *We consider the interval domain. A domain element is therefore the infinite set of reals contained in an interval. Quite obviously, we cannot represent an element by enumerating every single point in it: we need a compact finite representation. Trivially here, the abstract domain is made of couples in $\bar{\mathbb{R}}$, representing the bounds of the interval. Note that both domains are in bijection: we gained a better representation, but no additional mathematical properties.*

Another abstraction of intervals would be the sign domain: $\mathcal{A} = \{\emptyset, +, -, \pm\}$, that carries information about the sign of variables. An abstract corresponding element would therefore be deduced from the interval considered, while the concretisation of $a \in \mathcal{A}$ would respectively be the intervals $\emptyset, [0, +\infty]$, $[-\infty, 0], [-\infty, +\infty]$. The domains are not in bijection: computations will be easier on the sign domain, but we carry fewer information than on the interval domain.

Note that the notion of abstraction is relative: an interval abstracts a set of reals it contains, but is abstracted by the sign of its elements.

As illustrated through the previous example, it may be highly more convenient to transport ourselves on abstract domains, where two major purposes can be attained. First, the identification of the mathematical properties of the domain, on which the analysis is likely to be funded. Second, properties are infinite sets, hence a compact way to represent them is required: abstract domains fill this need.

The abstract domain can either be bijective with the concrete one, therefore only aiming at representation purpose, or be less precise, enhancing our computational capabilities. Moreover, we need a way to transport our information from the concrete domain to the abstract one, where computations occur. Then, we need to transport the information back in order to obtain the property overapproximation. The correspondence being usually not bijective, we want to lose as few information as possible when moving from a world to another: this intuition is formalized by the notion of Galois connection.

Definition 2 (Galois connection)**.** *Let $(L_1, \sqsubseteq_1, \bigsqcup_1)$ and $(L_2, \sqsubseteq_2, \bigsqcup_2)$ be two complete lattices. A couple of functions $\alpha \in L_1 \to L_2$ and $\gamma \in L_2 \to L_1$ is a Galois connection if :*

$$\forall x_1 \in L_1, x_2 \in L_2, \alpha(x_1) \sqsubseteq_2 x_2 \Leftrightarrow x_1 \sqsubseteq_1 \gamma(x_2)$$

α is referred to as the abstraction function, while γ is the concretization.

2.1.5 The widening operator

As stated above, loops require fixpoint computations. The lattice structure ensures existence of a least fixpoint, equal to $\bigsqcup_{n\in\mathbb{N}} \mathcal{F}^n(\bot)$. But is it always computable, and if so, within a tractable complexity? Intuitively, we may grow too slowly in our lattice and want to accelerate the iteration by performing some kind of jumps in the lattice.

Such intuition can be formalized by an operator such that iterative application of this operator always results in a chain of finite length. However, despite granting us with a computable fix point, it does not always result in the smallest one. Widening operators are therefore a consequent source of imprecision.

Widening operators was the last missing tool. Thanks to this whole framework, designing an abstract analysis is realized by defining an abstract lattice, building a Galois connexion between both the abstract and the concrete lattice, and designing the abstract semantic. On our programming model, the latter is therefore composed of operators on the abstract domain for annotations, both assignments and tests, as well as a process to compute fixpoints, possibly using a widening operator. In the remaining of this section, we present several analyses fitting in this framework, starting with analyses dedicated to linear invariant inference.

2.2 Linear invariants

Inferring polynomial invariants is a difficult task, intuition as well as techniques have therefore first been developed on an easier problem: linear invariants. We present through this section two analyses dedicated to this purpose: first the seminal work of Cousot and Halbwachs [15], and then a more recent one, by Sankaranarayanan [40].

2.2.1 The polyhedral domain

Cousot and Halbwachs designed an analysis in order to generate linear relations between variables which hold at a given program point for any execution [15].

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be the variables of the program. Let us consider a linear inequality; it has the following form:

$$\sum_{i\in\{1,\ldots,n\}} a_i x_i \leq b$$

This inequation defines an hyperplane on $\mathbb{R}^n$, which can be seen, as stated above, a particular property. A finite set of linear inequalities defines

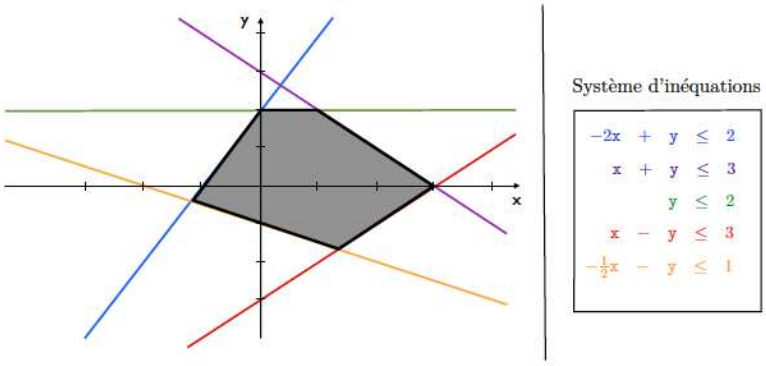
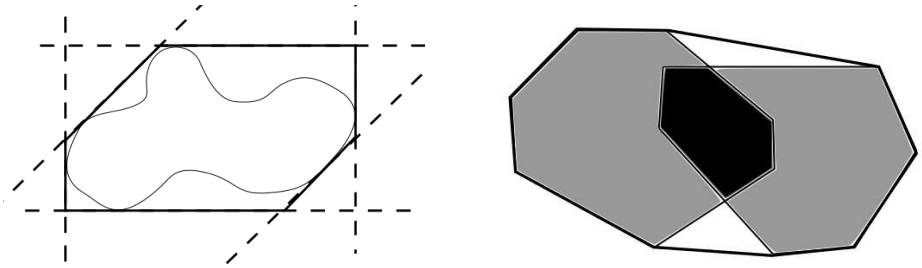


Figure 2: A convex polyhedron, defined both geometrically and algebraically (figure extracted from Jobin's thesis [22])



(a) Approximation of a set by a polyhedron

(b) The meet and join of two polyhedra

Figure 3

an intersection of hyperplanes, which can be seen geometrically as a convex polyhedron, as seen on Figure 2. We therefore want to approximate a property, the collecting semantics of our program, by a convex polyhedron which subsumes that property, as illustrated on Figure 3a. Once more, the smallest the polyhedron, the most precise information we have.

As explained in Section 2, we need a way to represent properties, polyhedra here, in a compact manner. Figure 3a exposes a first solution: a polyhedron can be defined as a set of linear constraints. More precisely, we represent a polyhedron as a matrix A and a vector $\mathbf{b}$, encoding the linear constraints by $A\mathbf{x} \leq b$. A polyhedron can also be interpreted geometrically, being generated by vertices and vectors. Each representation is efficient for specific tasks, both are generally used together in static analysis tools.

The set of convex polyhedra over $\mathbb{R}^n$ can be proved to have a lattice structure, and therefore acts as our abstract domain. Being stable for the meet operation, the definition of this operator is trival. The join is a bit more tricky, since we have to over-approximate the set-theoretic join by its convex hull, as illustrated on Figure 3b.

This context being set, we initially have no constraints on programs, and therefore start from $\mathbb{R}^n$ as an initial polyhedron. We then collect our information thanks to an abstract correspondance for both the assignment and the boolean test. For instance, a linear boolean test adds a constraint to the polyhedron.

The lattice of polyhedra is of infinite height, hence termination of fixpoint computations is not guaranteed: we need a widening operator. The accuracy of the analysis highly depends on its definition and this question has led to numerous researches. Cousot and Halbwachs's initial proposition[15] was to keep only the constraints of one polyhedron which are verified by every generator of the other one. While being safe, this operator gets rid of a lot of information. Hence, the question has been deeply studied and led to numerous enhancements, among which we can cite Bagnara's work [3].

2.2.2 The template-based polyhedral domain

The expressiveness of polyhedra makes them useful for capturing linear relationships between variables. Nevertheless, the worst case complexity of polyhedral computations is exponential. Numerous weaker domains have therefore been designed, looking for a compromise between completeness and computational efficiency. One of particular interest is the template-based polyhedral domain, introduced by Sankaranarayanan [40].

Intuitively, the complexity of the general polyhedral domain lies in the infinity of linear directions to chose from in order to improve the approximation. We therefore restrict ourselves to a finite subset of directions, fixed at each program point.

An abstract element was previously defined by a convex polyhedron, which can be defined as a $m \times n$ matrix A and a vector $\mathbf{b}$, the polyhedron being the set of $\mathbf{x}$ such that $A\mathbf{x} \leq \mathbf{b}$. Restricting the possible directions corresponds to fixing one matrix A for each program point, an abstract element consequently being simply a vector $\mathbf{b}$. This leads to the following Galois connection:

$$\{\text{Set of linear constraints}\} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \{\mathbf{b} \text{ such that } A\mathbf{x} \leq \mathbf{b}\}$$

This illustrates the relative nature of abstraction: the concrete domain here, the sets of linear constraints, is equivalent of the one of convex polyhedra, which was the abstract one in Cousot's analysis.

Despite maintaining a rich expressiveness with respect to linear invariants, operations on such a domain are still worst case polynomial. However, the use of this domain raises a connex problem: the choice, at each program point, of the corresponding template. Sankaranarayanan suggests to use heuristics for this purpose.

2.3 Polynomial equalities as invariants

Inferring linear invariants is an efficiently solved task, scaling up on industrial programs having been realized[1]. Nonetheless, it does not grasp as much information as desired, which makes the task of designing validated program a challenge. Hence, we would like to generate polynomial invariants of any degree.

While what we are ideally looking for, and actually study in the next section, are polynomial inequalities as invariants, *i.e.* relations of the form $\{\mathbf{x} \mid p(\mathbf{x}) \geq 0\}$ where p is a n-variate polynomial, an easier problem consists in seeking invariants as polynomial equalities. Note that one such relation is equivalent to a couple of the former: $\{p(\mathbf{x}) \geq 0\}$ and $\{-p(\mathbf{x}) \geq 0\}$.

We motivate in this section the use of the polynomial ideal domain as an abstract domain for such a problem, before briefly presenting analyses generating polynomial equalities as invariants.

2.3.1 The domain of ideals

Let $\mathcal{S} = \{p_1 = 0, \ldots, p_m = 0\}$ be a set of polynomial equalities. Such a set possesses strong structural properties: $\forall p \in S,\ \forall q \in \mathbb{R}[x_1, \ldots, x_n],\ p.q = 0$ *and* $\forall p, q \in \mathcal{S},\ p + q = 0$.

Such properties of stability for a set define the mathematical notion of ideal.

Definition 3 (Polynomial ideal)**.** *A set $\mathcal{I} \subseteq \mathbb{R}[x_1, \ldots, x_n]$ is an ideal* if*:*

- $0 \in \mathcal{I}$
- $\forall p_1, p_2 \in \mathcal{I},\ p_1 + p_2 \in \mathcal{I}$
- $\forall q \in \mathbb{R}[x_1, \ldots, x_n],\ p.q \in \mathcal{I}$

We note $\langle S \rangle$ the ideal generated by a set of polynomials S.

Hence, if we are able to infer some polynomial invariants, any polynomial in the ideal they generate are of interest. Moreover, the set of ideals has a structure of lattice, which we have seen to be a necessity for defining an analysis: $I \sqcap J = I \cap J$ and $I \sqcup J = \langle I \cup J \rangle$.

Our third concern is having a compact representation of our ideals. The following theorem states that ideals can always be finitely represented, even if containing an infinity of elements.

Theorem 1 (Hilbert)**.** *For any ideal I, there exists $S \subseteq I$, finite, such that $\langle S \rangle = I$.*

[1]For instance, Airbus A380 flight control has been proved run time error free in 2008 with a static analysis tool. See http://www.astree.ens.fr/

Nevertheless, ideal intersection and fixpoints computations require to solve the membership problem: in order to detect stabilization, we have to be able to state if a polynomial p is part of an ideal. While being trivial with mono-variate polynomials, this problem is harder with multi-variates ones as they do not constitute an euclidean ring. We therefore need a substitute to euclidean division: Gröbner bases. A Gröbner basis is a particular set of generators for an ideal which grants us with a process to solve the membership problem. Nevertheless, Gröbner basis computation is known to be doubly exponential with respect to the number of variables.

However, ideals appear as the natural choice of abstract domain when aiming at polynomial equalities inference. In the remaining of this section, we will present the state of the art in this specific sub-domain.

2.3.2 Analyses for polynomial equality inference

While recent, the domain of polynomial equality inference is quite well covered. Notably, Rodríguez-Carbonell and Kapur introduced an analysis [37] complete[2], for a sub-class of programs, whose in particular assignments have to be inversible in a certain sense. The analysis performs on the domain of ideals, but is restricted in practice by the use of Gröbner bases, preventing it to scale well with the number of variables.

Müller-Olm and Seidl proposed a complete backward analysis [32] whose only restriction is that guards have to be polynomial disequalities. While lacking of experiments and complexity result, their analysis is precious for highlighting the interest of backward analysis in order to facilitate abstract assignment computations, as well as for its constrained-based aspect, allowing the systematic derivation of an invariant inference algorithm from a invariance testing algorithm.

Sankaranarayanan *et al.* have proposed in 2004 an analysis dedicated to the inference of polynomial equalites invariant [39]. They do not impose any constraint on programs, but their analysis in turn is not complete. Their major contribution to the domain is the notion of weakening the consecution constraint, allowing to trade completeness with a reduction of the computational cost.

Cachera *et al.* have proposed another analysis [11], using the domain of ideals and extending the works of [32] and [39]. No hypotheses are made on the program structure, but they only compute inductive loop invariants, similarly to Sankaranarayanan's approach. Managing to get rid of any Gröbner bases use, the technique has shown good scalability.

[2] An analysis is said to be complete for a class $\mathcal{I}$ of invariants and a class $\mathcal{P}$ of programs if, executed on a program $p \in \mathcal{P}$, any invariant $i \in \mathcal{I}$ of p is found.

2.4 Polynomial inequalities as invariants

While many progress still have to be done, polynomial equalities as invariants inference seem to have its main methods commonly accepted. On the other hand, polynomial inequalities inference techniques have not shown so far good scalability properties.

In this section, we first describe an approach taking its inspiration from the equality case by looking for an adequate structure similar to the ideal structure; we then study an extension of the template-based domain introduced in Section 2; finally, we analyse the flaws of this method and motivate the study in depth of a third solution.

2.4.1 A similar approach to the equality case: cones of polynomials

The stable properties of polynomial equalities have led to the structure of polynomial ideals. Bagnara *et al.* [4] suggested a similar reasoning in the inequality case.

Given a finite set $\{p_i \geq 0\}$ of polynomial inequalities, one can note that, $\forall \lambda,\ \lambda_i \in \mathbb{R}^+,\ \lambda + \sum_i \lambda_i\ p_i \geq 0$. The set of those polynomials is the so-called *polynomial cone* generated by $\{p_i \geq 0\}$. However, as this structure only contains linear combinations, it does not perfectly entail all consequent invariants: consider the cone generated by $\{x_1, x_1x_2\}$, the invariant $x_2 \geq 0$ is obvious, but not entailed. We therefore need to close the cone with respect to the product operator. Note that ideals also respect this closure property. The analysis on this domain is then implemented on top of the polyhedral domain by considering each variable as an independant variable.

Implementation has shown that the analysis is able to infer non-trivial quadratic invariants. However, the practical results remain quite modest and show no direct hope to scale up.

2.4.2 Extending the template-based domain

A drastically different approach has been proposed by Adjé *et al.* [1], extending the template-based domain described in Section 2. A template is now given by a finite set of polynomials, allowing more complex sets to be constructed, non-convex ones in particular.

Given a template $P = \{p_i\}_{i \in I}$, a concrete element is a set of points delimited by polynomial equalities, named P-sub-level set. Conversely, an abstract element is a function which associates to each p_i a real number v_i, in order to represent the set of points representing $\{\mathbf{x} \mid p_i(\mathbf{x}) \leq v_i, \forall i\}$. We

therefore have the following Galois connection.

$$\{P\text{-sub-level sets}\} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \{\text{functions from } P \text{ to } \bar{\mathbb{R}}\}$$
$$X \mapsto (p \mapsto \underset{\mathbf{x}\in X}{sup}\ p(\mathbf{x}))$$
$$\{\mathbf{x} \in \mathbb{R}^n \mid p(\mathbf{x}) \leq v(p), \forall p \in P\} \hookleftarrow v$$

While being suitable to any degree, this domain leaves us with the task of safely approximizing an optimum of polynomials under polynomial constraints. This problem is known to be NP-hard, leading Adje *et al.* to restrict their analysis to degree two, hence performing the so-called Shor's relaxation.

2.4.3 The problem of template inference

Investigating the extension to higher degrees of the method by Adje *et al.* could have been an interesting way to investigate. Indeed, while being NP-hard, the constraint polynomial optimization problem appears to be deeply related to polynomial invariants: intuitively, no matter which method we follow, we are ultimately trying to bound as precisely as possible a set by a polynomial hypersurface.

Nevertheless, their framework is quite rigid in the sense the templates used in the analysis have to be manually provided. This knowledge therefore needs to be built on human expertise, a situation we would like to prevent as much as possible.

In order to illustrate this flaw, let us introduce a classical approach in stability theory: the concept of Lyapunov function. Indeed, in the domain of stability theory, invariants are specifically of great interest in order to prove that a given system is stable, *i.e.* does not diverge. Such a result is traditionally obtained by exhibiting a Lyapunov function:

Definition 4 (Lyapunov function)**.** *Given a system defined by the evolution law* $\mathbf{xk+1} = f(\mathbf{x_k})$*, a Lyapunov function is a function* $V : \mathbb{R}^n \to \mathbb{R}$ *such that:*

$$V(0) = 0 \wedge \forall \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}, V(\mathbf{x}) > 0 \wedge \lim_{||\mathbf{x}||\to\infty} V(\mathbf{x}) = 0$$
$$\forall \mathbf{x} \in \mathbb{R}^n, V(f(\mathbf{x}) - V(\mathbf{x})) \geq 0$$

Intuitively, such a function, unbounded in all directions, decreases at each step of evolution of the system, therefore proving the stability of the system.

In the context of Adje *et al.*'s method, exhibiting a Lyapunov function provides a suitable template for proving stability on the system. But at this point, the problem has moved to the question of how to infer Lyapunov functions. Cousot proposed a method answering this question in the specific case of ranking function inference [13]. Once again, an extention of

this method to invariant inference and higher degrees raises the question of polynomial optimization. We therefore present Cousot's method before investigating the theoretical and practical viability of its extension.

3 A constraint-based approach through successive relaxations

We have chosen to start from a method mostly synthesized, with a different objective, by Cousot [13]. In opposition to most previous approaches, whose starting point is a chosen domain, the idea here it to pass a quite general characterization of the semantics through successive "sieves", hopefully reaching tractable time and space complexities.

Cousot developed this method specifically aiming at inferring ranking functions. Roughly speaking, a ranking function is a function from the values of program variables into a well-funded set, $\mathbb{N}$ for instance, which is strictly (and fast enough) decreasing at each iteration of the loop, hence proving termination for the given program. Cousot points out at the end of his paper the possibility of an extension of this method to invariant inference. However, this alteration rises additional difficulties, regarding which Cousot seemed quite optimistic, while performing very few actual insight to this problem. Our work has mainly be to explore this idea, trying to validate or invalidate it.

The remaining of this section consists of Cousot's method, mainly through the exposition of the successive relaxations. This is all well known litterature, while we tried to enrich it with discussions about loss of expressivity. The in depth analysis of the resulting system of constraints and our related contributions will be dealt with in Section 4.

3.1 Relational semantics and loop invariant

3.1.1 Programming model

We consider in the remaining of this report a programming model more specific than the one introduced in Section 2.1.1. We consider a simple imperative language, allowing assignments, tests and loops. We suppose our program to have n variables, taking their values in $\mathbb{R}$; a state is therefore an element of $\mathbb{R}^n$. We define the *preconditions* of a loop as an overapproximation of the possible states before entering the loop. Preconditions are defined as algebraic relationship over the variables of the program.

More specifically, we are interested in inferring, given a loop and its preconditions, algebraic relationship for any state reached by the execution of the loop starting from a state satisfying the precondition to hold. Programs

considered therefore have the following shape:

$$\mathbf{x} \leftarrow P \textit{ (Preconditions)}$$
$$\textbf{while } B \textbf{ do } C.$$

where $\leftarrow$ abusively designs here a nondeterministic assignment. B is a boolean test, and C a subprogram, potentially containing tests, loops and assignments itself. Assignments are assumed to be polynomials[3]: any assignment has the form $x_i = p(\mathbf{x})$, where p is polynomial.

3.1.2 Relational semantics

Considering loops, we abstract the semantics of the program in a standard way, as a polynomial relationship between the values of the variables before the execution of the loop body and the ones after the execution. Having restrained our assignments to polynomial functions, we can therefore write, by composition of polynomials, the semantics of the loop body as a conjunction of m polynomial relationships $\bar{\sigma_1}, \ldots, \bar{\sigma_m}$ describing the effect of the loop on each variable. Introducing fresh copies of variables, we represent their values after the loop by $\mathbf{x}$ and before the loop by $\mathbf{x_0}$, we obtain:

$$\bigwedge_{k=1}^{m} \bar{\sigma_k}(\mathbf{x_0}, \mathbf{x}).$$

In presence of a conditional, we have one such relationship for each branch. When the semantics of a loop body is defined only by equality relationships, it can be equivalently expressed as an application σ such that, if $\mathbf{x}$ represents the values of the variables entering the loop, $\sigma(\mathbf{x}) = (\sigma_i(\mathbf{x}))$, with $\sigma_i \in \mathbb{R}[\mathbf{x}]$. This method avoids the introduction of fresh variables, but will induce a rise in the considered degree by substitution inside the invariant, as described in further details in Section 3.1.3. However, we stick in this report to the latter description of the semantics, illustrated on Figure 4.

3.1.3 Characterization of loops invariants

Based on such a description of the semantics, being the least fixpoint of the canonical extension of σ over sets, starting from the set of preconditions, a subclass of invariants can be characterized: loop invariants. A loop invariant is an algebraic relationship which is stable under execution of the loop body. More formally, $P \geq 0$ being a polynomial relationship characterizing the

[3]This apparent restriction has to be mitigated. Despite of the numerous calls to logarithms or trigonometric functions the *Math* library Java allows you, down to a low enough level, any computation is polynomial.

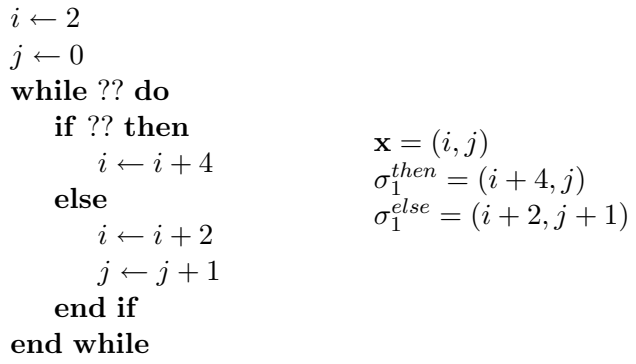


Figure 4: An elementary program and the associated relational semantics.

preconditions, let $I \geq 0$ be a polynomial relationship among variables, $I \geq 0$ is a loop invariant of the program if and only if:

$$P(\mathbf{x}) \geq 0 \Rightarrow I(\mathbf{x}) \geq 0$$
$$\left(I(\mathbf{x_0}) \geq 0 \wedge \bigwedge_{k=1}^{n} \sigma_k(\mathbf{x_0}, \mathbf{x}) \right) \Rightarrow I(\mathbf{x}) \geq 0$$

Equivalently, when the semantics is expressed as polynomial equalities:

$$P(\mathbf{x}) \geq 0 \Rightarrow I(\mathbf{x}) \geq 0 \tag{1}$$
$$I(\mathbf{x}) \geq 0 \Rightarrow I(\sigma(\mathbf{x})) \geq 0 \tag{2}$$

The definition is highly intuitive: we state first that any possible initial state, *i.e.* satisfying the precondition, also satisfies the invariant, and second that the invariant is stable by the semantics of the loop body.

Distinction between invariants and the specific subclass of loop invariants must be understood. In the case of an infinite loop, *while true do...*, we have:

$$[I(\mathbf{x}) \Rightarrow I(\sigma(\mathbf{x})] \wedge [P(\mathbf{x}) \Rightarrow I(\mathbf{x})] \qquad \text{[Loop invariant]}$$
$$[P(\mathbf{x}) \Rightarrow \forall n,\ I(\sigma^n(\mathbf{x})] \qquad \text{[Generic invariant]}$$

Let us consider the example depicted on Figure 5 to illustrate the loss of expressivity induced by the notion of loop invariant. The program simply doubles the norm of the variables if this norm is greater or equal to 1, cuts it in half otherwise. The precondition being the open unit ball, the concrete semantics is exactly this open ball, which therefore is an invariant, and even a loop invariant. However, open sets are quite hard to grasp, reacting badly to meet and join operations. The closed unit ball would therefore be an interesting overapproximation, and hence an invariant. Nevertheless, the closed unit ball is not a loop invariant: the border is not stable by the semantics of the body. More precisely, this program admits no other loop invariant than the concrete collecting semantics, *i.e.* the open unit ball.

```
Precondition: x^2 + y^2 < 1
while true do
    if x^2 + y^2 ≥ 1 then
        x ← 2 · x
        y ← 2 · y
    else
        x ← x/2
        y ← y/2
    end if
end while
```

Figure 5: The closed unit ball is a trivial invariant, but not a loop invariant.

3.2 A bridge to computability: the parametric abstraction

In its current form of characterizing general inductive invariants, our problem is not even computable. Indeed, we implicitely existentially quantify over predicates: we state that there exists a predicate I such that the precondition and the consecution are satisfied. Hence, we have a second order formula, leading to undecidability.

In order to get rid of such a complexity, we realize a parametric abstraction as named by Cousot. Intuitively, we fix *a priori* the shape of the invariants by parameterizing it. For instance, we can restrict ourselves to conics by parameterizing our invariant under the form $I_{a,b,c} = ax^2 + by^2 + c$. The characterization of a parameterized invariant $I_{\mathbf{a}}$ therefore becomes:

$$\begin{aligned} &\exists \mathbf{a} \in \mathbb{R}^p : \\ &\qquad \forall \mathbf{x} \in \mathbb{R}^n : \; P(\mathbf{x}) \geq 0 \Rightarrow I_{\mathbf{a}}(\mathbf{x}) \geq 0 \\ &\qquad \forall \mathbf{x} \in \mathbb{R}^n : \; I_{\mathbf{a}}(\mathbf{x}) \geq 0 \Rightarrow I_{\mathbf{a}}(\sigma(\mathbf{x})) \geq 0 \end{aligned}$$

The loss of expressiveness here is quite easy to grasp: we restrict our inference to invariants of the chosen shape, conics in the example above. However, the quantification is no more over a predicate, but over the parameters, hence relying on first order quantification.

Note that at first sight, this process can present similarities with the notion of template, described previously through the works of Sankaranarayaran and Adje. However, parametric abstraction is far more general: while the idea behind a template is to fix a specific polynomial shape, and optimize how close from the semantics you move it by tweaking exclusively the constant, parametric abstraction theoretically allows you to restrict among ellipses for instance and look for the ideally shaped one.

In spite of the fact that decidability is reached, this restriction is still too loose, calling for a new "sieve".

3.3 Lagrangian relaxation

With this first relaxation to parametric invariants, the inference indeed becomes computable: the Tarski-Seidenberg decision procedure for the first-order theory of real closed fields by quantifier elimination can be used for instance. Anyway, even the cylindrical algebraic decomposition method by Collins [12] is still doubly exponential in the number of quantifier blocks, and is the best method available. There is still a need for further relaxation.

3.3.1 Definition

The Lagrangian relaxation states that a sufficient condition for a relationship of the form:

$$\forall \mathbf{x} \in \mathbb{R}^n : \left(\bigwedge_{k=1}^{m} \sigma_k(\mathbf{x}) \geq 0 \right) \Rightarrow \sigma_0(\mathbf{x}) \geq 0 \tag{3}$$

to hold is given by the following condition:

$$\exists \lambda_i \geq 0 : \ \forall \mathbf{x} \in \mathbb{R}^n, \ \sigma_0(\mathbf{x}) - \sum_{k=1}^{m} \lambda_k \sigma_k(\mathbf{x}) \geq 0. \tag{4}$$

Indeed, assume $\forall k \in \{1, \ldots, m\}, \ \sigma_k(\mathbf{x}) \geq 0$, then $-\sum_{k=1}^{m} \lambda_k \sigma_k(\mathbf{x}) \leq 0$ and (3) $\Rightarrow$ (4) is straightforward.

However, the converse implication does not hold in general. Nevertheless, if all the σ_i are assumed to be linear, the Lagrangian relaxation is complete. Additionally, the relaxation known as the Yakubovitch's *S-procedure* [46, 23], frequently used in system theory to derive stability, happens to be a particular case of Lagrangian relaxation which happens to be complete while admitting one quadratic constraints among the linear ones.

3.3.2 A link between Lagrangian relaxation and duality theory

The Lagrangian relaxation as expressed here takes its origin from the theory of the Lagrangian dual in optimization, described in further details in Section 4.1.1. However, we can notice without further knowledge that (3) is equivalent to $p^* \geq 0$ where p^* is the solution to the following optimization problem:

$$\begin{aligned} &\text{minimize } \sigma_0(\mathbf{x}) \\ &\text{subject to } \bigwedge_{k=1}^{m} \sigma_k(\mathbf{x}) \geq 0 \end{aligned}$$

Hence, defining $L(\lambda) = \inf_{\mathbf{x}} \sigma_0(\mathbf{x}) - \sum_{k=1}^{m} \lambda_k \sigma_k(\mathbf{x})$, and the dual problem:

$$\begin{aligned} &\text{maximize } L(\sigma) \\ &\text{subject to } \lambda \geq 0, \end{aligned}$$

any feasible point of the dual problem is a lower bound on the initial one, hence establishing the sufficient condition. Furthermore, the condition is an equivalence if and only if the duality gap, the difference between the solutions of both problems, is tight (*i.e.* equal to zero). This additional condition is notably achieved in the linear case, but not in the general polynomial context.

3.3.3 Invariant characterization

Back to invariant inference, relaxing the parametric invariance conditions provides the following characterization of invariants:

$$\begin{aligned} \exists \mathbf{a},\, \exists \lambda_P \geq 0,\ \forall \mathbf{x},\ I_{\mathbf{a}}(\mathbf{x}) - \lambda_P P(\mathbf{x}) \geq 0 \\ \exists \lambda \geq 0,\ \forall \mathbf{x},\ I_{\mathbf{a}}(\sigma(\mathbf{x})) - \lambda I_{\mathbf{a}}(\mathbf{x}) \geq 0. \end{aligned}$$

At this point, the constraints are first order quantification over inequality constraints. While this is far from guaranteeing tractability, it constitutes a more reasonable ground to lay on compared to the one we started from. In the following of the report, we investigate both theoretically and experimentally the task of extracting relevant invariants through the use of optimization solvers.

4 Invariant generation as solutions of a set of constraints

The successive relaxations described in the previous section allow for syntactically generating a decidable set of constraints, encoding the so-called precondition and consecution conditions. Any solution of these constraints is an invariant for the considered program. Two questions naturally arise. After those successive relaxations reducing the time complexity, is the derived system efficiently solvable? Furthermore, from an algebraic perspective, the solutions of a system of constraints are not distinguishable. This is not true in our case: we look for semantically meaningful solutions, *i.e.* sets of parameters defining relevant invariants. For instance, if two circles centered at the same point are both invariants of the considered program, the one with the smallest ray is of greater interest. The second question therefore is: how to guide the solver toward solutions which are relevant to us? While both problems are closely related, this explains why we can't restrict ourselves to constraint solving and have to consider the more general setting of constrained optimization.

We try through this section to investigate and reduce the time complexity of the constraint resolution, before considering the problematic of guiding the solver to a semantically relevant solution. Finally, we propose ideas on

yet unsolved problems open for further works. However, we start by giving some notions of optimization that will be useful for our purpose.

4.1 A targeted framework: semidefinite programming

The question of solving or relaxing the current system we are dealing with cannot be answered without some insight into the topic of optimization, also known as mathematical programming. In its most general form, this framework is defined as the optimization of a function $f_0 : \mathbb{R}^n \to \mathbb{R}$, under inequality constraints over functions on the same domain and codomain:

$$\begin{aligned} &\text{minimize} && f_0(\mathbf{x}) \\ &\text{subject to} && f_i(\mathbf{x}) \leq b_i,\ i = 1 \dots m \end{aligned}$$

Where no hypotheses are put neither on the *objective* function, f_0, nor on the f_i, functions denoting the constraints. Note that equalities can be encoded as a couple of opposed inequalities. Naturally, this framework is extremely general, and highly non-computable. For the sake of tractability, special subclasses have therefore been heavily studied.

4.1.1 Linear programming

The most elementary but still interesting optimization problem consists in enforcing all objective and constraint functions to be affine. Such a problem is called a linear program, usually written as:

$$\begin{aligned} &\text{minimize} && \mathbf{c}^T\mathbf{x} + d \\ &\text{subject to} && A\mathbf{x} \succcurlyeq \mathbf{b} \\ & && \mathbf{x} \succcurlyeq 0, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m,\ x \in \mathbb{R}^n$ and $\succcurlyeq$ denotes over vectors the pointwise inequality.

The feasible set of a linear program, *i.e.* the set of $\mathbf{x}$ satisfying the constraints, is a polyhedron, hence naturally a convex set. Solving the problem is quite straightforward: the solution is the farthest possible x following direction $-c$, as illustrated on Figure 6. Dantzig's simplex method for instance permits to efficiently solve in such problems practice.

In addition, the linear case benefits from an invaluable property, namely the duality theorem, that helps in designing optimization solvers. The idea behind duality consists in associating to a so-called *primal* problem of n variables and m constraints a closely related problem of m variables and n constraints. In the linear case, we therefore define the dual as:

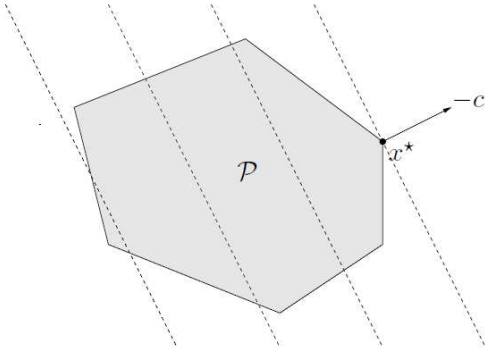


Figure 6: Geometric interpretation of a linear program (from [8]). Dashed lines are objective function's level curves. $\mathbf{x}^*$ is optimal, as far as possible along direction $-\mathbf{c}$.

Primal	Dual
min $\mathbf{c}^T\mathbf{x}$:	max $\mathbf{y}^T\mathbf{b}$:
$A\mathbf{x} \succcurlyeq \mathbf{b}$	$\mathbf{y}^T A \preccurlyeq \mathbf{c}^T$
$\mathbf{x} \succcurlyeq 0$	$\mathbf{y} \preccurlyeq 0$

The duality theorem states the existing link between both primal and dual problems [16].

Theorem 2. *If both the* primal *and the* dual *of a linear program are feasible, then the two optima coincide.*

This very strong result relies on Farkas' lemma, stating when a linear system is satisfiable.

Lemma 1 (Farkas' lemma)**.** *The linear system*

$$\begin{aligned} \textit{subject to} \quad & A\mathbf{x} \succcurlyeq \mathbf{b} \\ & \mathbf{x} \succcurlyeq 0, \end{aligned}$$

is satisfiable if and only if using a positive linear combination of the inequalities, it is possible to derive $-1 \geq 0$, i.e. *there exists* $\lambda_1, \ldots, \lambda_m \in \mathbb{R}^+$ *such that*

$$\sum_{i=1}^{m} \lambda_i \mathbf{a}_i \prec 0 \textit{ and } \sum_{i=1}^{m} \lambda_i b_i > 0,$$

where the a_i *are the lines of* A*.*

The duality theorem therefore offers an alternate problem which can be exploited in order to solve the problem more efficiently. Linear programming is an ideal case, that we can in practice solve exactly in a very efficient way. We will now see how most of the ideas behind this first framework can be extended to the more general case of convex programming.

4.1.2 Convex programming

A particular attention has been given to a subclass of mathematical programs for about one century, namely convex programming [8]. This attention has even been growing in the last decades with the discovery of efficient resolution methods. In a convex optimization problem, the functions $f_0, \ldots, f_m$ are supposed to be convex. Obviously, linear programs are special instances of this framework. While there is in general no analytical formula for the solution of convex optimization problems, convexity is a key to efficiency. Let us give a few insights on this interest of convexity.

First, any locally optimal point of a convex problem is also globally optimal, which highly facilitate the process of optimization. Indeed, an optimization algorithm typically consists in picking up an initial point and iterating the following steps, until a stopping criterion is satisfied:

- determine a descent direction Δx;
- choose a step size t;
- update $x \leftarrow x + t\Delta x$

Hence, algorithms are likely to fall and get stuck in local optima, a problem which does not concern convex optimization.

What's more, convexity provides many optimality critera, which are valuable for determining a relevant descent direction. One illustrative example can be stated if f_0 is additionally assumed to be differentiable: x is optimal if and only if x is feasible and

$$\nabla f_0(x)^T(y - x) \geq 0, \text{ for any } y \text{ feasible},$$

where ∇ denotes the gradient operator. This condition generalizes the idea already present in the linear case: going as far as possible along direction $-c$. Whereas the gradient was constant in the linear case, the condition here more generally states that the opposite of the gradient evaluated at the optimal point defines a supporting hyperplane (*i.e.* tangent and such that the set is included on one side of the hyperplane) to the feasible set, as illustrated on Figure 7.

Such sufficient conditions provide ways to determine the descent direction. The gradient descent method thus suggests to descend in the current opposite gradient direction, as illustrated on Figure 8. The more evolved steepest descent technique takes the direction which minimizes the gradient, intuitively where the slope is the highest, as illustrated on Figure 9. Finally, Newton's method uses a second-order Taylor approximation, assuming f_0 is twice continuously differentiable, to compute a more relevant descent direction based on the Hessian of f_0.

These methods are both quite intuitive and efficient. But they cannot be applied to general convex programs. The best compromise is Newton's

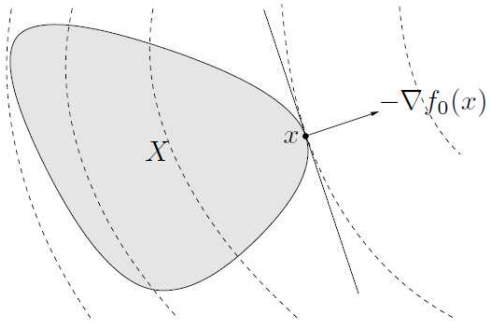


Figure 7: Geometric interpretation of the optimality condition (from [8]). Dashed lines are objective function's level curves. x^* is optimal, as far as possible in the direction $-\nabla f_0$.

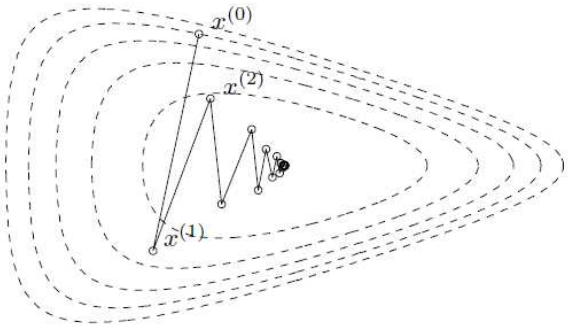


Figure 8: Iterates of the gradient method (from [8]). Dashed lines are objective function's level curves.

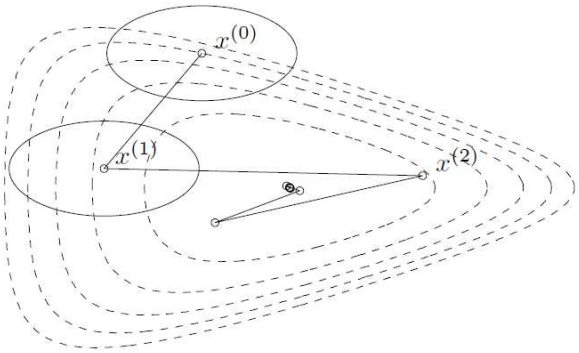


Figure 9: Iterates of the steepest descent method (from [8]). Dashed lines are objective function's level curves.

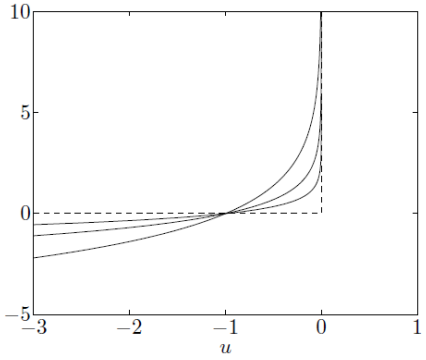


Figure 10: Comparison between the dashed indicator function and several solid logarithmic barrier (from [8]).

method which performs in practice extremely well on problems constrained by equalities exclusively. This method is therefore the basis of the so-called interior-points method for convex optimization. Intuitively, the idea is that an inequality constraint can be replaced by adding to the objective function a function which is null where the constraint is satisfied, and evaluates to infinity otherwise. Unfortunatly, such a function would deprive us from continuity, which we need. We therefore rely on a logarithmic barrier to approximate this indicator function with arbitrary precision, as illustrated on Figure 10. Interior-points method has proven to be in practice of great efficiency.

Convex programming is thus the cornerstone of optimization: the framework is expressive enough to formulate many problems, and most of the ideas developed for more complex problems extends ideas developed in the convex case.

4.1.3 Semidefinite programming

The framework of convex programming admits a significant extension which can be efficiently handled by an extension of the interior-points method: semidefinite programming[4]. Let us first briefly recall a few well known algebraic results.

Proposition 1. *For a real symmetric $n \times n$ matrix A, the following properties are equivalent:*

1. $x^T A x \geq 0$ *for all* $x \in \mathbb{R}^n$.

[4]Actually, inequalities can be generalized with respect to any proper cone (*i.e.* convex, closed, solid and pointed). Semidefinite programming coincides with the cone of positive semidefinite $k \times k$ matrices.

2. *All eigenvalues of* A *are* ≥ 0.

3. $A = U^T U$ *for some* $n \times n$ *matrix* U.

We say a square matrix A is positive semidefinite if A is real symmetric and the equivalent conditions *supra* hold. A linear matrix inequality (**l.m.i.**) is then defined as:

$$F_0 + \sum_{i=1}^{m} x_i F_i \succcurlyeq 0,$$

where over matrices, $A \succcurlyeq B$ means $A - B$ is semidefinite positive.

A semidefinite program is defined as the optimization of a linear function subject to a **l.m.i.** A survey of the theory and applications is provided by Vandenberghe and Boyd [45]. The popularity of this framework has grown constantly due to three circumstances: the amount of domains from which arise naturally semidefinite constraints; the expressiveness of the framework, most of convex programs admitting a semidefinite reformulation; and the quite recent discovery that semidefinite programs are actually not much harder to solve than linear programs.

Indeed, while no practical simplex method is able to handle those constraints, Nesterov and Nemirovsky managed to extend the interior-points method in the late eighties [34]. From a computer scientist perspective, semidefinite programming got notably renowned thanks to Goemans and Williamson [17] which performed a breakthrough on the maximum cut problem by relaxing the problem under the semidefinite framework.

Getting back to invariant inference, we note that semidefinite programming is the framework in which one Adje *et al.* [1] have chosen to fit. In order to do so, they restricted themselves to degree two, allowing them to perform the so-called Shor's relaxation.

In the remaining of the section, we try to analyze our specific problem in the light of the foregoing, and study ways to fall in the semidefinite framework, as well as possible alternatives.

4.2 Polynomial inequalities

While semidefinite programming appears to be the ideal framework, we do not fit straight in it. A first reason is that we deal with constraints as multivariate polynomial inequalities, which are highly non convex and cannot be expressed as a **l.m.i.** Actually, satisfiability of polynomial inequality constraints is known to be a NP-hard problem [33].

Unlike Adje *et al.*, we do not want to put restrictions over the degrees of invariants we infer, and therefore need to relax our polynomial inequality constraints. One well known solution to this issue is the sum of squares relaxation. In the remaining of this section, we study first the relaxation in itself before looking at how the sum of squares constraint reduces to semidefinite programming.

4.2.1 Relaxation as sum of squares

We consider the relation between two properties of a polynomial p:

- being nonnegative, *i.e.* $\forall \mathbf{x},\ p(\mathbf{x}) \geq 0$
- being a sum of squares (**s.o.s.**), *i.e.* $\exists p_1, \ldots,\ p_m \in \mathbb{R}[\mathbf{x}],\ p = \sum_{i=1}^{m} p_i^2$

More precisely, being a **s.o.s.** polynomial obviously implying the nonnegativity, we wonder if the converse implication holds. The study of this connection between the nonnegativity of a polynomial and existence of a way to write it as a sum of squares has a quite long history [29]. This problem goes back to Hilbert, first in a paper in 1888, then by stating a close question, writing a polynomial as a sum of squares of rational functions, as his 17^{th} famous problem.

Actually, Hilbert already knew that the equivalence stands in the univariate case. It is even true if we restrict ourselves either to degree two, or to degree four and two variables. But in any other case, and especially as soon as we have two variables and if we don't want to put restrictive hypotheses over degrees, there exists nonnegative polynomials which admit no formulation as a **s.o.s.** polynomial. The first constructive example has been exhibited by Motzkin in 1967 [31]:

$$s(x, y) = 1 - 3x^2y^2 + x^2y^4 + x^4y^2$$

Relaxing the positivity constraint by a **s.o.s.** constraint therefore strictly simplifies our problem. However, estimating the loss of expressivity is as much tricky than crucial. A few results are known: on one hand, Blekherman derived bounds for the volumes of the bases of both the cones of nonnegatives polynomials and **s.o.s.** ones, showing that in a sense, "there are significantly more nonnegative polynomials than sums of squares" [6]; on the other hand, Berg *et al.* established a denseness result [5], in the sense of a specific norm, of the cone of **s.o.s.** polynomial in the space of nonnegatives ones. From our perspective, those results appears highly technical, making it especially difficult to rely to a notion of semantical expressiveness. Two remarks can nevertheless be made: from a far more pragmatical point of view, experiments tend to show that many polynomials practically raised in various domain do accept a **s.o.s.** decomposition, allowing actual experimental use of the relaxation; from a perspective of a long term work, we tried to develop, and believe many more would be required, an expertise on the overall domains of real algebraic geometra as well as optimization in order to be able to identify the relevant techniques, as well as their precise drawbacks.

The dual issue to expressiveness raised by the **s.o.s.** relaxation is the one of time complexity. Fortunately, this problem has found a far more satisfactory answer.

4.2.2 Reformulation as a semidefinite program

Hence, **s.o.s.** constraints relaxes the positivity constraints while maintaining an experimentally interesting, despite hard to predict, amount of expressiveness. Moreover, the time complexity of **s.o.s.** testing recently encountered a satisfactory solution through several independant work, notably by Lasserre and by Parrilo. The main result, enforcing high interest in this relaxation, is the reformulation of the **s.o.s.** constraints in the semidefinite framework.

The method developed by Parrilo [36, 35] is built upon the computation of a **s.o.s.** decomposition thanks to the Gram matrix method. Indeed, given a polynomial p of degree $2d$, we write it in the basis of all monomials of degree less or equal to d:

$$Z = (\prod_{\sum_j r_{i,j} \leq n} x_j^{r_{i,j}}) : \ p = Z^t P Z,$$

where P is a constant matrix. For instance, considering the bivariate polynomial

$$F(x,y) = 2x^4 + 2x^3y - x^2y^2 + 5y^4 = \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}^T \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \end{bmatrix}.$$

Such a decomposition is not unique, but p is a **s.o.s.** polynomial if and only if there exists one such P which is semidefinite positive. The existence of a **s.o.s.** decomposition of a polynomial of n variable and degree d can therefore be checked by solving a semidefinite program of size $\binom{n+d}{d} \times \binom{n+d}{d}$.

Upon this result, Parrilo developed a procedure allowing the search for bounded degree solutions to the more general problem consisting of solving a set of polynomial inequalities. Furthermore, while this process checks for the emptyness of a set of constraints, it can be extended to compute a global lower bound of a polynomial p by maximizing a parameter γ such that $p(x) - \gamma$ is a **s.o.s.** polynomial, hopefully returning a precise underapproximation.

This procedure has been implemented, coupled with numerous optimization, especially linked to sparsity, by Parrilo in the **SOSTOOLS**[5] free **Matlab** toolbox.

Major breakthroughs in the domain have independently been developed by Lasserre, forging a link with the dual theory of moments. He has built both a process constructing a family of **s.o.s.** polynomials converging to a given polynomial [26], and a sequence of semidefinite problems whose sequence of solution converges to the solution of the initial problem bearing a nonnegativity constraint [24, 25] (see [41] for a more didactical survey of these results). Unfortunately, no implementations are available, and results

[5] http://www.cds.caltech.edu/sostools/

on the speed of convergence are still lacking at this time, outside of specific cases. Nevertheless, we believe that these works study might lead to enhancements.

4.3 Yalmip

Aiming at some experimental validation of this constraint-based method for invariant inference, we had to experiment various solvers over examples. To this end, the **Matlab** optimization application programming interface **YALMIP**[6], developed and maintained by Löfberg, has been of invaluable help. It offers a simple common language to describe mathematical programs of any nature, includes transparent speed optimization processes, and interface numerous solvers dedicated to each specific class of problem. In particular, **SOSTOOL** is even directly plugged inside of it [28].

4.4 Bilinear parametrization

So far, we saw that ideally, the semidefinite framework is the suitable one. We therefore aim at relaxing our generic polynomial inequalities constraints into **s.o.s.** ones as those have been discovered to be reformulable as semidefinite programs during the last decade. However, troubles remain. As stated sooner, semidefinite programming deals with **l.m.i.** Nevertheless, Lagrangian relaxation reformulates the consecution condition as

$$I_{\mathbf{a}}(\sigma(\mathbf{x})) - \lambda * I_{\mathbf{a}}(\mathbf{x}),$$

where $I_{\mathbf{a}}(\mathbf{x})$ is parametrized. We therefore, due to the Lagrangian multipliers λ, have a product between two parameters, moving us from the well-behaved world of linear matrix inequalities to the much wider one of bilinear matrix inequalities (**b.m.i.**).

4.4.1 An insight into the b.m.i. problem

A **b.m.i.** is of the form:

$$F(x,y) = F_0 + \sum_{i=1}^{m} x_i F_i + \sum_{j=1}^{n} y_j G_j + \sum_{i=1}^{m}\sum_{j=1}^{n} x_i y_j H_{ij} \succcurlyeq 0.$$

The bilinear term gets rid of convexity which was inherent to **l.m.i.**, allowing such a constraint to describe nonconvex sets and therefore suggesting the much greater complexity of this problem. Another way to have an insight into this complexity lies in the expressivity of this framework: any general polynomial inequality can be written as a **b.m.i.** [44]. It is therefore an NP-hard problem [7].

[6] http://users.isy.liu.se/johanl/yalmip/

However, recent progress has been performed on the way of practical solving of **b.m.i.** problems. We can in particular cite the works of Quoc Tran Dinh, Michiels and Diehl [42, 43], appearing to be promising, but which we could not test due to lack of available implementation. However, further development of these works should be followed.

The second recent breakthrough is due to Henrion, Kočvara and Stingl [20, 21]. They outline the fact that a bilinear problem can still contain hidden convexity properties and therefore be "simple" to solve, or can be numerically efficiently solved through relaxations such as in the case of the **s.o.s.** decomposition by Parrilo, or the dual theory of moments by Lasserre. Coupled with the observation that ill cases of convex problems can be hard to solve, they suggest that the convex/non-convex opposition is too simple, and propose an algorithm for non-linear programming. To give the general idea, the method is based on the use of a penalty/barrier function, similarly to the logarithmic one exploited for the interior-points method. However, technical details are at this point out of reach for us. Note finally that this algorithm cannot pretend at global optimization, but only local one. Nevertheless, this is by no mean eliminatory, a non optimal invariant still potentially bringing us relevant information.

From our point of view, the huge benefit of this work is the existency of an implementation, as a Matlab solver called *PENbmi*[7]. *PENbmi* appears to be the only one bilinear solver currently available, and has therefore been the one we used for experimentations.

We stated we did not manage to go into the technical details of the method. This actually constitutes a real practical difficulty since the algorithm can require high degree of expertise to exploit it at its best. Indeed, reformulation of the problem or choice of an appropriate initial point can be crucial for its efficiency.

4.4.2 Experiments

As stated above, even using the **s.o.s.** relaxation, our problem remains part of the **b.m.i.** framework, due to the product between Lagrangian multipliers and parameters of the invariant. Coupled with the asymmetry of relevancy between solutions, this peculiarity makes the treatment of invariant inference inherently harder than the one of ranking function inference as treated by Cousot [13]. Cousot directly underlined this point in his paper but seemed reasonably optimistic in the efficiency of *PENbmi*, exhibiting two examples. We therefore started experimenting thanks to this solver.

Let us first consider the following *Yalmip* program (omitting variable declarations and printing commands for clarity) from [13]. This program encodes the search for linear invariants, under the form $a.i + b.j + c \geq 0$, of the program described on Figure 4, p. 16:

[7] http://www.penopt.com/penbmi.html

```
(1)  F = [a*2+c>=0];
(2)  F = F + [sos(a*(i+4)+b*j+c-m*(a*i+b*j+c)), m>=0];
(3)  F = F + [sos(a*(i+2)+b*(j+1)+c-l*(a*i+b*j+c)), l>=0];
(4)  sol = solvesos(F, [], sdpsettings('solver','penbmi'), [a;b;c;m;l]);
```

Each line translates to:

(1) $P(x) \Rightarrow I(x)$, simplified as $I(2,0) \geq 0$

(2) $\exists m \geq 0,\ I(\sigma(x)) - m.I(x)$ is a **s.o.s.** (first branch)

(3) $\exists l \geq 0,\ I(\sigma(x)) - l.I(x)$ is a **s.o.s.** (second branch)

(4) Resolution using the *PENbmi* solver

Depending on the version of *PENbmi* used, we either get the same invariant as Cousot did:

$$2.14678.10^{-12}i - 3.12793.10^{-10}j + 0.486712 \geq 0,$$

or a totally different one:

$$1.59925.10^{-4}i + 4.85516.10^{-4}j + 2.22665.10^{5} \geq 0$$

This is not a surprise, as this simply highlights the lack of control we have at this point on the resulting invariant, only its correction being guaranteed. However, a more disturbing point might be noted: which relevance can we give to a result precise at 10^{-12} given by a numerical tool? Especially, if we refer to the *solvesos* documentation[8], terms smaller than 10^{-6} are get rid off as being most likely due to numerical inaccuracy. It is therefore unclear whether the invariant exhibited by Cousot is not simply the trivial relationship $0.497812 \geq 0$.

Anyway, the situation gets even worse on other examples. Reproducing the second example proposed by Cousot [13, page 21], we apparently get the exact same invariant as he did, but along with the following message from *PENbmi*:

```
+2.11831e-05*x -2.11831e-05*q*y -2.11831e-05*r >= 0
Stopped by iterations counter. Result may be wrong.
PENBMI failed.
```

The provided invariant therefore is totally irrelevant. And this does not constitute an isolated case. Considering the following elementary example, also taken from [13], but treated for invariant inference instead of ranking function inference:

Under this form, the solver successively returns an invariant, semantically irrelevant. However, thinking of guiding the solver, one first simple

[8] http://users.isy.liu.se/johanl/yalmip/pmwiki.php?n=Commands.Solvesos

$j \leftarrow 2;\ k \leftarrow 0$
while ?? **do**
  if ?? **then**
    $j \leftarrow j + 4$
  else
    $j \leftarrow j + 2$
    $k \leftarrow k + 1$
  end if
end while

```
I = a*j+b*k+c;
F = [2*a+c>=0];
F = [F, sos(a*(j+4)+b*k+c-l1*I), l1>=0];
F = [F, sos(a*(j+2)+b*(k+1)+c-l2*I), l2>=0];
sol = solvesos(F, [], ..
  ..sdpsettings('solver','penbmi'), param);
```

idea could be to normalize the result by enforcing $c = 1$, either as an additionnal constraint or directly in the parametric abstraction. However, this simple change, intuitively more likely to simplify the problem by suppressing a variable, now raises a *PENbmi* error:

```
Stopped by linesearch failure. Result may be wrong.
PENBMI failed.
```

Multiplying the experiments, trying to deal with examples from [1] or setup elementary examples, both the previously described errors arised in the majority of the cases, in an impredictable way. As a conclusion, *PENbmi* has shown to be, without extensive expertise allowing one to cleverly reformulate the problem, unable to deal even with problems as small as two variables and five parameters, with degree 2.

The conclusion of this first stream of experiments is quite unexpectedly one-sided: as things stand, bilinear solvers do not handle our problem under its current form. It is not excluded that, considered how restricted is the bilinearity of our problem, a clever reformulation could allow the use of *PENbmi*. However, this would require far more expertise in the specific field of optimization under **b.m.i.** that we got. We therefore decided to seek for a heuristic that aims at getting rid of bilinearity.

4.5 Necessity of a heuristic

Experiments tend to show tools dealing with bilinear matrix inequalities cannot currently handle our issues. Nevertheless, the bilinearity we are dealing with is quite restricted, and only present in the consecution condition:

$$I_{\mathbf{a}}(\sigma(\mathbf{x}) - \lambda * I_{\mathbf{a}}(\mathbf{x})$$

Fixing the value of λ, coupled with the **s.o.s.** relaxation, would therefore be sufficient to get down to semidefinite programming. Hence we seek for a heuristic aiming at finding an interesting value for this Lagrangian multiplier. The problematic need nevertheless to be precised. Indeed, to each

while *true* **do**
  $x \leftarrow -y$
  $y \leftarrow x$
end while

Consecution:
$ay^2 + bx^2 - 1 - \lambda(ax^2 + by^2 - 1)$ is a **s.o.s.**

Figure 11: A simple rotation and the induced consecution.

positive λ is associated a set of admissible $\mathbf{a}$, hence of inferable invariants. We would therefore like to be able to approximate this set for a given λ. Note that despite the restricted bilinearity we are dealing with, a precise answer to this question seems to be highly unlikely to be found, *PENbmi* appearing to be quite unable to deal with our constraints.

In order to develop some intuition about it, we first focus on degree two and study some examples. Visualization is easier in this restricted case, since these invariants define conics. As a first illustration, let us consider a parametric invariant $I_{a,b} = ax^2 + by^2 - 1$, and a loop body executing a simple rotation of $\pi/2$. The program and the associated consecution constraint are given on Figure 11.

Note that the quarter of the plane in which parameters (a, b) lie defines the type of conic in the variables (x, y) we obtain as an invariant of the program: an ellipse if $a > 0,\ b > 0$, the whole plane if $a < 0,\ b < 0$, and a hyperbola otherwise. Hyperbolae are not invariant by rotation, thus cannot be invariant for our program. Neither can ellipses, except for the particular case of circles, defined by the constraint $a = b$. The plane is naturally always invariant, but gives us no information on the semantics of our program.

This last remark highlights an hidden difficulty to our problem. At first sight, we could assume that finding a λ whose associated feasible set is non-empty would be sufficient in order to get relevant information about the semantics of the program. In particular, one could try to exploit the optimization description of the Lagrangian relaxation, as described in Section 3.3.2, trying to enforce Slater's conditions[9] for instance. However, this example shows how insufficient this idea is[10].

Indeed, elementary calculations lead to the following result about the feasibility of the consecution constraint, as subsumed on Figure 12:

- *if* $\lambda \in [0, 1[$, $a \leq b/\lambda \wedge a \geq \lambda b$, hence only invariants describing the whole plane are available, for a decreasing set of admissible values of a and b.

[9] Given a convex programming problem, Slater's condition states that if a strictly feasible point exists (all constraints are strictly satisfied at this point), then the duality gap is tight.

[10] Additionaly, note that the way we encode the constraints also is relevant. For instance, assuming our preconditions are the unit square, encoding it as four linear inequalities or two quadratic inequalities do not lead to the same sets of solutions.

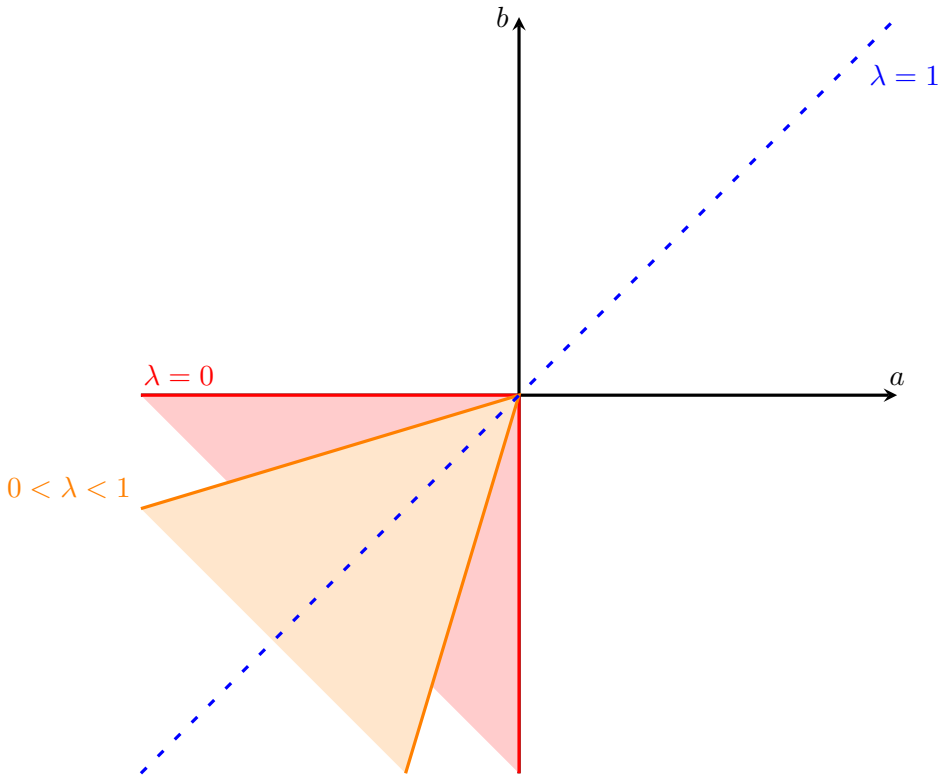


Figure 12: Admissible parameters set for the rotation, depending on the value of the Lagrangian multiplier.

- *if* $\lambda = 1$, $a = b$, hence both the whole plane and any circle are admissible invariants.
- *if* $\lambda > 1$, the consecution admits no solution.

A few things might be noticed. We normalized our equation by setting the constant to 1. This guaranties λ to lie in $[0, 1]$ by setting $x = y = 0$. We also note that, as suggested above, while the system admits solutions for any $\lambda \in [0, 1]$, $\lambda = 1$ is the only possibility which leads to semantically relevant solutions.

While these observations can be made by hand on such an elementary example, it quickly gets harder for less elementary ones. We therefore tried to formalize the idea by taking a close look to the **s.o.s.** decomposition process. As described in section 4.2.2, the algorithm consists in writing p in the monomial basis, $p = Z^t P Z$, and imposing P to be semidefinite positive. One characterization of the semidefinite positiveness of a matrix is the nonnegativity of each of its principal minors:

Definition 5 (Principal minor)**.** *Let M be a square matrix of size $n \times n$.*

We define the principal minors of degree $k < n$ of M as the determinants of any submatrix of M obtained by deleting k rows and k columns with the same indexes.

In the example of the rotation, the matrix

$$P = \begin{pmatrix} b - \lambda \cdot a & 0 & 0 \\ 0 & a - \lambda \cdot b & 0 \\ 0 & 0 & \lambda - 1 \end{pmatrix}$$

is so sparse that the condition is even equivalent to the nonnegativity of its principal minors of first order. While this won't be true in the general case, it could be an interesting necessary condition to use as, at least, a starting point for a heuristic.

We therefore experiment this idea on a bit more elaborate example, a filter taken from [1], as shown on Figure 13.

while *true* **do**	**Consecution:**
$x \leftarrow \frac{3}{4} \cdot x - \frac{1}{4} \cdot y$	$a(\frac{3}{4}x - \frac{1}{4}y)^2 + bx^2 - 1 - \lambda(ax^2 + by^2 - 1)$ is a
$y \leftarrow x$	**s.o.s.**
end while	

Figure 13: A linear filter from [1] and the induced consecution.

We deal in this case with a matrix

$$P = \begin{pmatrix} (\frac{9}{16} - \lambda) \cdot a + b & -\frac{3}{64} \cdot a & 0 \\ -\frac{3}{64} \cdot a & \frac{1}{64} \cdot a - \lambda \cdot b & 0 \\ 0 & 0 & (1 - \lambda \cdot c) \end{pmatrix}$$

Only imposing at this point the positivity of first order principal minors, we obtain the situation depicted on Figure 14.

Once again, we see a transitory value of λ, defined by the colinearity of both hyperplanes delimiting admissible semi-planes, below which no ellipse is admissible, but above which some are. It therefore appeared as an interesting first guess for λ, starting point of an eventual exploration for various invariants. Anyway, the comment above must be balanced by the fact we took conditions only from first degree principal minors. Experimentally, on this example, we observed it was actually necessary to raise λ of roughly 0.2, hence 20% of the size of the domain we search, in order to actually find an ellipse. While it is hard to be categorical, due to the successive layers of relaxation, it is most likely to be a consequence of higher order principal minors. Such an imprecision on a quite elementary example makes this idea of heuristic arguably uninteresting compared to a simple dichotomy one.

However, either through dichotomy or manual tweaking based on our idea of heuristic, we quite manage in practice to get rid of the bilinearity. The second question therefore rises: how can we guide the solver toward semantically relevant solutions?

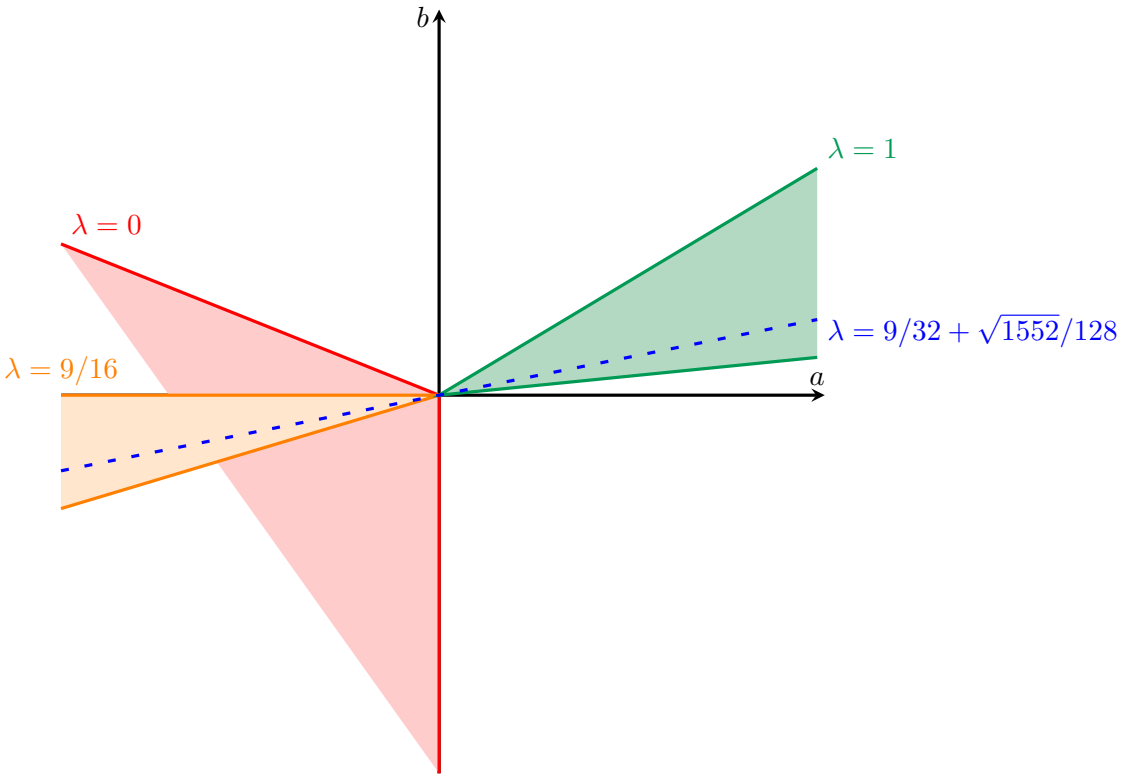


Figure 14: Admissible set for the filter depending on the value of the Lagrangian multiplier.

4.6 Guiding the solver

While any solution to the set of constraints is a perfectly correct invariant of the considered program, we would like to find semantically relevant invariants, approximating with decent precision the concrete semantics. In particular, the solver is more than likely to find trivial invariants, such as $0 \leq 1$ or $x^2 + y^2 \geq 0$, providing strictly no information about the program.

We therefore need to guide the solver, and we can use the objective function to perform this task. In the particular case of degree two, one could want, for instance, to use the objective function in order to minimize the area of the obtained ellipse. Provided we write the ellipse in the form $ax^2 + by^2 \leq 1$, the area is $\frac{\pi}{ab}$. But such an objective cannot be expressed as a semidefinite programming problem.

Indeed, semidefinite programming only admits linear objectives. However, some non-linear objective functions can be translated into linear ones: consider for instance the following objective:

$$\frac{(c^T x)^2}{d^T x}.$$

By introducing a new variable t, we can equivalently minimize t by imposing a new constraint bounded by t:

$$\frac{(c^T x)^2}{d^T x} \leq t \Leftrightarrow \begin{pmatrix} t & c^T x \\ c^T x & d^T x \end{pmatrix} \succcurlyeq 0.$$

But in the case of a simple product between two parameters, one such process would to the best of our knowledge inevitably induce a **b.m.i.** Nevertheless, the overall idea seems to be interesting. Considering the example on Figure 11, with the Lagrangian multiplier λ set to 1, the only invariants which can be obtained are circles. Hence minimizing the parameter $-a$ would lead the solver to the smallest circle, centered at the origin, containing the preconditions, as illustrated on Figure 15.

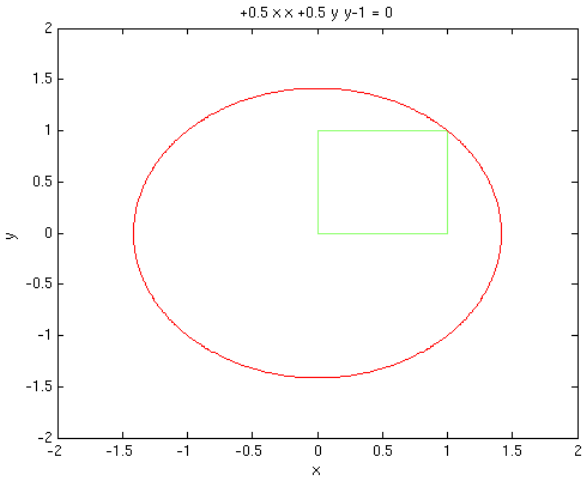


Figure 15: Results obtained for the rotation. The precondition are plot in green and the inferred invariant in red.

Naturally, this case is minimalistic. We therefore experimented on harder examples, such as the oscillator from [1] described on Figure 16. Being unable to minimize the area, we nevertheless seek optimal ellipses in a sense: successively minimizing $-a$ and $-b$ also provides interesting invariants, minimizing each axis of the ellipse. Invariants being sound for the intersection operator, we obtain this way a more precise invariant.

The result of the process can be found on Figure 17. We can note that the intersection of both our guided invariants, in blue and magenta, is strictly better than the one obtained by Adje *et al.*, in black.

A next step would have been to manage to rotate and translate the ellipses. While the computations get more complicated, making it hard to visualize what we can optimize, a few results have been obtained concerning

$h == 0.01$
while *true* **do**
  $y \leftarrow (1-h)y - hx$
  $x \leftarrow x + hy$
end while

Parametric abstraction:

$$I_{a,b,c} = -ax^2 - by^2 - cxy + 1$$

Figure 16: Implementation of an oscillator from [1]

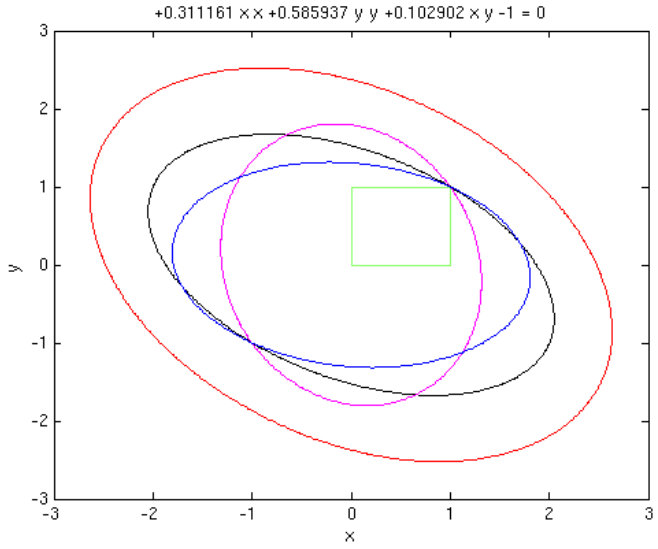


Figure 17: Results obtained for the oscillator. Are respectively plotted: preconditions in green, Adje *et al.*'s result in black, no guiding in red, minimizing $-a$ in magenta and minimizing $-b$ in blue.

rotations. However, we got interrupted in this task by the discovery of a recently published paper.

4.7 A generic ellipsoid abstract domain for linear time invariants systems

Restricting the range of our analysis, we ended up on a problem already solved by Roux *et al.* [38]: a generic ellipsoid abstract domain has been designed, using highly similar, while more polished, techniques than the ones we ended up dealing with. We expose this work in the remaining of this section.

4.7.1 Studied systems

The authors consider linear control-command based critical systems: a loop models the acquisiton of new input values via sensors, the update of internal state variables and the generation of new outputs. More specifically, the model is an infinite loop whose body can be expressed under the form $\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k$ where $\mathbf{u}_k$ is non-deterministic, such that $||\mathbf{u}_k|| \leq 1$.

Such a system is usually proven stable by exhibiting a so-called Lyapunov function, as defined in section 2.4.2. In particular, such a function V can be searched as a quadratic form, hence $V(\mathbf{x}) = \mathbf{x}^T P\mathbf{x}$. The Lyapunov characterization then is equivalent to:

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

This formulation of the problem is one of the keys of their success: they encode into the constraints the specific research of bounded ellipsoids. We will come back to this point in greater details in Section 4.8.

A Lyapunov function consists in the exhibition of a particular bounded invariant. Nevertheless, as once again we are interested in more than stability, we would like to compute a semantically interesting one. Two steps are therefore required: finding a well suited shape, and then optimizing the ratio.

4.7.2 A relevant shape of the ellipsoid

Any matrix P such that $A^T PA - P \prec 0$ will lead to a proof of stability, but additional constraints can enforce semantical relevance. Three of them are proposed by Roux *et al.*:

- **Minimizing condition number**. This measurement expresses how close from an hypersphere we are. Intuitively, this could be desirable since it kind of ensures that no dimension is "too much penalized". This condition is encoded, in a semidefinite program, by minimizing a fresh variable r under the additional constraint $I \preccurlyeq P \preccurlyeq rI$.

- **Preserving the shape.** We are only looking at loop invariants, hence shapes which are not too much changed by the body semantics might be relevant. This idea is encoded by minimizing a fresh variable r under the additional constraint $A^T PA - rP \preccurlyeq 0$. Note that the process rises bilinearity which the authors simply solve by dichotomy.

- **All in one**. In order to also take into account B, the authors seek the smallest possible ellipsoid P such that

$$\forall \mathbf{x}, \forall \mathbf{u},\ ||\mathbf{u}||_\infty \leq 1 \Rightarrow \mathbf{x}^T P\mathbf{x} \leq 1 \Rightarrow (A\mathbf{x} + B\mathbf{u})^T P(A\mathbf{x} + B\mathbf{u}) \leq 1$$

This condition is relaxed through the S-procedure, a particular case of the Lagrangian relaxation which happens to be complete, as exposed in Section 3.3.1. Once again, dichotomy is used to solve the bilinearity issue.

4.7.3 Finding a good stable ratio

P being a solution of the Lyapunov inequality, a stable ratio is guaranteed to exist. Anyway, we look for the smallest possible. The set of admissible ones are exactly the fixpoints of the functional mapping λ_k to the maximum of λ_k and the least λ_{k+1} such that

$$\forall \mathbf{x}_k, \mathbf{u}_k,\ ||\mathbf{u}_k||_\infty \leq 1 \Rightarrow \mathbf{x}_k^T P_k \leq \lambda_k \Rightarrow \mathbf{x}_{k+1}^T P \mathbf{x}_{k+1} \leq \lambda_{k+1}$$

Each iteration may be encoded in the semidefinite framework. Convergence is classically ensured thanks to a widening operator.

As their work is quite subsuming the idea we were following, we rather tried to look for a generalization.

4.8 An attempt to generalization

The quadratic case, to which we had restricted ourselves as a starting point, thus seems to be decently addressed by [38]. We therefore once again would like to deal with higher degrees. While their analysis attested we were going in the good way, they addressed more efficiently a few problematic points, bringing ideas from which we could get some inspiration. First, a simple dichotomy seems to be viable in order to deal with bilinearity. More significantly, they were able to constrain the solver to look exclusively for ellipsoids, avoiding any trivial invariant. This idea came from a previous work by Feron *et al.* [2] which exhibited the following property: the set defined by

$$\mathcal{E}(P) = \{\mathbf{x} \in \mathbb{R}^n : \begin{pmatrix} 1 & \mathbf{x}^t \\ \mathbf{x} & P \end{pmatrix} \geq 0\}$$

where P is positive semi-definite defines an ellipsoid, possibly flat in some dimensions.

Hence, extending such a property to higher degrees could ensure potentially interesting invariants, before considering some more precise guiding. More formally, we want a sufficient condition on the coefficients of a polynomial p for the set $SA_p = \{\mathbf{x}|\ p(\mathbf{x}) \leq 0\}$ to be bounded. Furthermore, we naturally need to be able to express this condition in the semidefinite programming framework.

A first few remarks might help us to build some intuition. For instance, if p is definite positive, *i.e.*

$$\forall \mathbf{x},\ p(\mathbf{x}) > 0$$

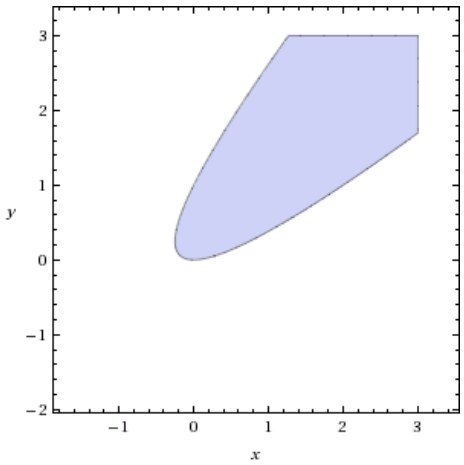


Figure 18: The set $\{(x, y) \mid (x - y)^2 - y \leq 0\}$ is unbounded: enforcing the top degree part to be semidefinite positive is insufficient.

then the set SA_p is empty. Another idea comes naturally in mind, especially since we know how to relax it under our framework: letting p be semidefinite positive, *i.e.*:

$$\forall \mathbf{x},\ p(\mathbf{x}) \geq 0.$$

However, this constraint reduces SA_p to the set of the zeros of p. We therefore would in particular restrict ourselves to sets with empty interiors, which semantically would make very little sense. Indeed, invariants we could still look for would be more than likely to be handled by polynomial equalities. This idea therefore is not suitable.

A better idea comes from considering the top degree part of p, name it q. q is an homogeneous polynomial of degree $k = deg(p)$. Unfortunately, q semidefinite positive is still not suitable. Take for instance the following polynomial:

$$p(x, y) = (x - y)^2 - y.$$

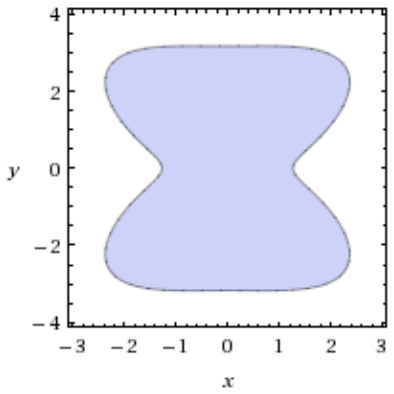
$q = x^2 - 2xy + y^2$ is semidefinite positive as a **s.o.s.** However, if we take a look at SA_p as plotted on Figure 18, the set is far from bounded, containing a whole parabola.

In contrast, if q is definite positive then SA_p is bounded. Indeed, we first prove the following lemma:

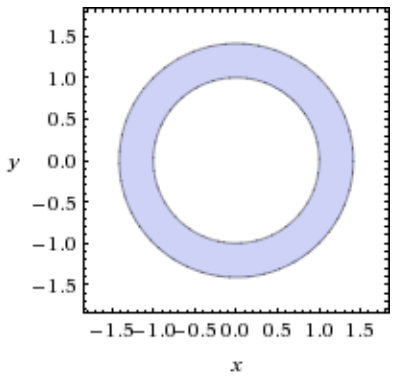
Lemma 2. *Let p be homogeneous, $\forall \mathbf{x},\ ||\mathbf{x}|| = 1,\ p(\mathbf{x}) > 0$ implies $\forall \mathbf{x},\ p(\mathbf{x}) > 0$.*

Proof. Let $\mathbf{x} \neq 0$, as p is homogenous of degree l,

$$p(\mathbf{x}) = p\left(||\mathbf{x}|| \cdot \frac{\mathbf{x}}{||\mathbf{x}||}\right) = ||\mathbf{x}||^l \cdot p\left(\frac{\mathbf{x}}{||\mathbf{x}||}\right) > 0$$



(a) $x^4 + y^4 - x^2 - 10y^2 - 1 \leq 0$



(b) $x^4+y^4+2x^2y^2-3x^2-3y^2+2 \leq 0$

Figure 19

□

We now can prove the theorem:

Property 1. *Let p be a polynomial and q its top degree part. If q is definite positive, then SA_p is bounded.*

Proof. Let $m = \inf_{\mathcal{S}(0,1)} q$. As q is continuous and $\mathcal{S}(0,1)$ is a compact space, m is reached, hence $m > 0$.

We therefore have, $\forall \mathbf{x},\ q(\mathbf{x}) = ||\mathbf{x}||^l \cdot q(\frac{\mathbf{x}}{||\mathbf{x}||}) \geq ||\mathbf{x}||^l \cdot m$. Hence, writing $p = q + r$, as $deg(r) < deg(q) = l$, $r(\mathbf{x})/q(\mathbf{x}) = o(1)$ and the result follows. □

We believe this characterization might be of great help. However, definite positiveness can't be direcly expressed as semidefinite constraints. A relaxation we can think of is imposing q to be a positive linear combination of squares of monomials. The result is obvious if all the x_i^r are included, and slightly relaxed but still interesting by allowing hyperplanes, defined by $x_i = 0$, otherwise.

While this relaxation might seem unreasonably strong, it still allows interesting non-convex sets (actually, even non-connex), which could be semantically relevant. See Figure 19 for examples.

A finer relaxation would be to state that "q is almost nonnegative", which can be formalized by the condition

$$\exists \epsilon > 0,\ q(\mathbf{x}) - \epsilon||\mathbf{x}||^l \text{ is a } \textbf{s.o.s.}$$

We could therefore fix for instance $\epsilon = 10^{-2}$ and iterate by dividing ϵ by 10 in case of failure. The process is guaranteed to terminate [27].

Nevertheless, characterizing a class of bounded sets is not sufficient: we still need to be able to minimize, in a sense, the volume of a set. This task appears at the moment to be quite hard from an algebraic point of view. A more reasonable solution would be to first seek for a good shape thanks to the solver and then minimize the volume by numerical methods.

However, those ideas remain to be explored through further works.

5 Related work

5.1 Strenghtening

Through this report, we mainly addessed the purpose of guiding a solver toward an invariant as good as possible. However, assuming this first step is realized, iterating the process in order to find successive invariants of different nature would be of great interest. Intuitively, we would first prove the inclusion of the semantics inside an ellipse, then exclude a part of the ellipse by proving the semantics to be on one side of an hyperplane, and so on continue to refine our result.

This idea has been developed by Bradley and Manna [9, 10] by introducing the notion of *strenthening*. They enhance the conditions of precondition and consecution with an additional constraint encoding the existency of a memory state which is rejected with the new invariant being sought, despise being included in the abstract semantics computed so far. Let us illustrate the idea on the example depicted on Figure 4.4.2:

$$
\begin{aligned}
& I[j,k] = a + bj + ck & \\
& I[2,0] \geq 0 & (precondition) \\
\wedge\ & \chi_{i-1} \wedge I \geq 0 \Rightarrow I[j+4,k] \geq 0 & (consecution_1) \\
\wedge\ & \chi_{i-1} \wedge I \geq 0 \Rightarrow I[j+2,k+1] \geq 0 & (consecution_2) \\
\wedge\ & \exists j,k,\ \chi_{i-1} \wedge I < 0 & (strenghtening),
\end{aligned}
$$

where χ_{i-1} is the current knowledge when starting iteration at step i, setting $\chi_0 = true$. Then if ϕ_i is the solution computed, $\chi_i = \phi_i \wedge \chi_{i-1}$.

While this idea of strenghtening is appealing, it brings a huge drawback under the form of an existential constraint. Indeed, this constraint is to the best of our knowledge conflicting with the Lagrangian relaxation, which constitutes the centrepiece in our chain of successive "sieves". The authors therefore proposed an iterative, randomized algorithm:

1. Randomly pick a point $s \in \chi_{i-1}$.
2. Solve $I[p,s] < 0 \wedge dual(\psi_i(p))$ for p. If no solution is found, go back to step 1.

3. Optimize the solution (hopefully avoiding to refine at each step the same shape). More precisely, taking $I = p_0 + J[p, x]$:

$$\textbf{minimize } p_0$$
$$\textbf{subject to } p_0 + J[q, x] \geq 0 \wedge dual(\psi_i[p_0, q_1, \ldots, q_k])$$

4. $\chi_i = \chi_{i-1} \wedge I[q] \geq 0$

While we believe that the concept of strengthening is extremely relevant and would deserve further investigations, their current result seems far from satisfying. Indeed, the author address the bilinearity issue in a quite unsatisfactory way, basically human choice, and their iterative algorithm lays on a raw randomized process coming with no guarantee of termination.

5.2 Program analysis as SAT constraints solving

Gulwani, Srivastava and Venkatesan developed a program analysis based on constraint solving [19] presenting many similarities with Cousot's approach. However, they consider more specifically programs annoted with linear assertions, and try to verify the program. They also derive both weakest preconditions and strongest postconditions.

Their method, as in the one we investigated, generates second order constraints based on a relational semantics. However, the generation of these constraints are studied more accurately, based on the notion of *cut-set*. A *cut-set* is defined as a set of nodes such that for any cycle in the control flow graph, the intersection is non-empty. An easy way to generate such a set is by taking the node at the loop headers. Based on this *cut-set*, the author syntactically reduce the verification problem to a set of constraints. Intuitively, the idea is to generate a constraint for each path in the graph defined by the *cut-set*, and take their conjunction. Those constraints are backward collected from the *assume* and *assert* statements, using substitution to deal with assignment.

The choice of the *cut-set* influences the time complexity of the resulting system of constraints. However, as with Cousot's work, the authors relax those second-order constraints by performing the parametric abstraction, followed by the Lagrangian relaxation, reduced here to the Farkas' Lemma due to the restriction to linear assertions.

Having reached this point, they choose to rely on SAT solvers rather than mathematical programming solvers. They therefore convert the first-order quantifier-free formulae to SAT formulae by encoding it as boolean operations over bit-vectors.

Their method extends to linear invariant inference for interprocedural analysis, as well as weakest precondition and strongest postcondition inference. Benefiting from the high efficiency of existing off-the-shelf SAT

solvers, benchmarks seem to prove their technique promising. Nevertheless, possibility to scale up to richer class of invariants, and especially non-linear polynomials, is unclear. Note finally that their study of the importance of a relevant choice of *cut-sets* would probably also concern methods based on mathematical programming.

5.3 Fixpoints of abstract semantics as solutions of mathematical programs

Goubault *et al.* recently provided a different tribute to mathematical programming in the domain of static analysis [18]. For convenience, they consider the classic abstract domain of intervals, defining on this domain an abstract semantics for an elementary iterative language. Given a program, the result of the analysis therefore overapproximates the concrete semantics by providing bounds for every variable of the program, result which is defined as a least fixpoint over the lattice of intervals.

Their contribution lies in the definition of a compiler generating from the language to mathematical programming. The resulting mathematical program is proved to have a unique solution which coincide with the least fixed point of the semantics over intervals.

While this first work is restricted to a quite elementary abstract domain, the process highlights once more how powerful mathematical programming can be in static analysis. We expect with great interest extensions to richer domains, as well as further enrichments of the process.

5.4 Optimal abstraction on real-valued programs

Monniaux proposed an interesting insight to the issue of semantics decidability [30]. The author considers programs using real variables without loops and abstracts these programs using formulas in the theory of real closed fiels. This abstraction is proved to be exact.

Then, defining an abstract domain similar to the polynomial template-based domain used by Adje *et al.* [1], Monniaux provides an algorithm computing an optimal abstraction of the semantics of the program with respect to this domain. This establishes, in this sense, a decidability result over the semantics of real valued, loop-free, programs.

However, this algorithm makes use of quantifier elimination in the theory of real closed fields. The time complexity of the best known algorithm for this operation is a tower of exponentials. The process therefore is obviously not tractable at the moment.

6 Conclusion

With the development of embedded software in critical devices, the urge for safety analysis has increased. One specific tool desired for this purpose is invariants, *i.e.* algebraic relationships between variables of the program. In this study, we have been more specifically interested in polynomial inequalities as invariants.

This report first presents an overview of existing methods on this topic. While no general techniques are effective at this time for polynomial inequalities, some restrictions already provide tractable approaches, and the recent progress in the field of mathematical programming gives hope for some generalizations.

The work we present here specifically pays attention to the generalization to invariant inference of the method developed by Cousot [13] in the context of ranking function inference. We therefore have presented the successive relaxations we considered, sieving the relational semantics up to a grain big enough to be tractable.

This process generates a set of constraints whose resolution provides us with an invariant. We thus gave insights on mathematical programming before investigating the class of mathematical problems we are facing. Two difficulties being identified, polynomial positivity and bilinearity, we addressed them both through an additional relaxation, namely **s.o.s.** formulation, and experiments on the *PENbmi* solver.

Concluding to the impracticability of *PENbmi*, we tried to restrict ourself to a subclass of invariants in order to conceive a heuristic. While obtaining promising initial results, a recent work by Roux *et al.* [38] addressing the problem was discovered. Restricting themselves to ellipsoids, they followed a similar process as ours, enhanced in particular with a better characterization of their domain, allowing an efficient guidance of the solver. We therefore went on seeking for a generalization of their work, looking for a domain generalizing the set of ellipsoids. This task first led us to looking for a characterization of polynomials p such that the set $\{\mathbf{x} \mid p(\mathbf{x}) \leq 0\}$ is bounded.

Numerous further works could be carried on. First, additional expertise on polynomial positivity and bilinear optimization could allow one to mitigate our conclusion of non-viability of bilinear solvers dependent techniques, as well as allow a better characterization of bounded sets in the framework of semidefinite programming. Furthermore, characterizing interesting sets would still leave us with the task of finding the "smallest" such set possible. Additionally, assuming the resolution of such sets of constraints would be mastered, the key issue therefore would be the one of strenghening: how could we enforce the generation of new solutions, offering an intersection of invariants?

Nevertheless, while appearing not as reliable as algebraic methods at this point, we believe numerical methods on constrained-based abstractions

of the semantics might be a promising way.

Acknowledgments

I am grateful to Michel Coste, Jérémie Le Borgne, Marie-Françoise Roy and Tristan Vaccon for kindly answering to our questions.

I would like to deeply thanks Thomas Jensen for having warmly welcomed me for the second time in the Celtique team, as well as all the members of the team. In particular, I thank Arnaud Jobin for the numerous fruitful discussions.

As appropriate, my last words go to David Cachera for having once more dedicated so much time and kindness to my supervision!

References

[1] Assalé Adjé, Stéphane Gaubert, and Eric Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. *Logical Methods in Computer Science*, 2012.

[2] Fernando Alegre, Eric Feron, and Santosh Pande. Using ellipsoidal domains to analyze control systems software. *CoRR*, abs/0909.1977, 2009.

[3] Roberto Bagnara, Patricia M. Hill, Elisa Ricci, and Enea Zaffanella. Precise widening operators for convex polyhedra. *Sci. Comput. Program.*, 58(1-2):28–56, October 2005.

[4] Roberto Bagnara, Enric Rodríguez-Carbonell, and Enea Zaffanella. Generation of basic semi-algebraic invariants using convex polyhedra. In *SAS*, pages 19–34, 2005.

[5] Christian Berg, JensPeterReus Christensen, and Paul Ressel. Positive definite functions on abelian semigroups. *Mathematische Annalen*, 223(3):253–274, 1976.

[6] Grigoriy Blekherman. There are significantly more nonegative polynomials than sums of squares. *Israel Journal of Mathematics*, 153(1):355–380, 2006.

[7] Vincent Blondel and John N. Tsitsiklis. NP-hardness of some linear control design problems. *SIAM J. Control Optim.*, 35(6):2118–2127, November 1997.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[9] Aaron R. Bradley and Zohar Manna. Verification constraint problems with strengthening. In *Proceedings of the Third international conference on Theoretical Aspects of Computing*, ICTAC'06, pages 35–49, Berlin, Heidelberg, 2006. Springer-Verlag.

[10] Aaron R. Bradley and Zohar Manna. Property-directed incremental invariant generation. *Form. Asp. Comput.*, 20(4-5):379–405, June 2008.

[11] David Cachera, Thomas P. Jensen, Arnaud Jobin, and Florent Kirchner. Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases. In *SAS*, pages 58–74, 2012.

[12] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, September 1991.

[13] Patrick Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *Proceedings of the 6th international conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'05, pages 1–24, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[15] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Symposium on Principles of Programming Languages*. ACM Press, New York, NY, 1978.

[16] G.B. Dantzig and M.N. Thapa. *Linear Programming: 2: Theory and Extensions*. Linear Programming 2: Theory and Extensions. Springer, 2003.

[17] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995.

[18] Eric Goubault, Stéphane Le Roux, Jeremy Leconte, Leo Liberti, and Fabrizio Marinelli. Static analysis by abstract interpretation: A mathematical programming approach. *Electron. Notes Theor. Comput. Sci.*, 267(1):73–87, October 2010.

[19] Sumit Gulwani, Saurabh Srivastava, and Ramarathnam Venkatesan. Program analysis as constraint solving. *SIGPLAN Not.*, 43(6):281–292, June 2008.

[20] D. Henrion, M. Kocvara, and M. Stingl. Solving simultaneous stabilization BMI problems with PENNON. Technical report, LAAS-Toulouse, 2003.

[21] D. Henrion, J. Lofberg, M. Kocvara, and M. Stingl. Solving polynomial static output feedback problems with PENbmi. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 7581–7586, 2005.

[22] Arnaud Jobin. *Dioïdes et idéaux de polynômes en analyse statique.* PhD thesis, ENS Cachan - Bretagne, 2012.

[23] Ulf T. Jönson. A lecture on the S-procedure, 2001.

[24] Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[25] Jean B. Lasserre. Semidefinite programming vs. LP relaxations for polynomial programming. *Math. Oper. Res.*, 27(2):347–360, 2002.

[26] Jean B. Lasserre. A sum of squares approximation of nonnegative polynomials. *SIAM Journal on Optimization*, 16(3):751–765, 2006.

[27] Monique Laurent. Sums of squares, moment matrices and optimization over polynomials, 2008.

[28] J. Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, 54(5):1007–1011, 2009.

[29] M. Marshall. *Positive polynomials and sums of squares.* Mathematical surveys and monographs, v. 146. Amer Mathematical Society, 2008.

[30] David Monniaux. Optimal abstraction on real-valued programs. In *Proceedings of the 14th international conference on Static Analysis*, SAS'07, pages 104–120, Berlin, Heidelberg, 2007. Springer-Verlag.

[31] T.S. Motzkin. The arithmetic-geometric inequality. In O. Shisha ed., editor, *Inequalities*, pages 205–224. New York: Academic Press, 1967.

[32] Markus Müller-Olm and Helmut Seidl. Computing polynomial program invariants. *Inf. Process. Lett.*, 91(5):233–244, 2004.

[33] K.G. Murty and S.N. Kabadi. *Some NP-complete problems in quadratic and nonlinear programming.* Mathematical programming, 1987.

[34] I.U.E. Nesterov, Y. Nesterov, and A. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming.* SIAM studies in applied and numerical mathematics: Society for Industrial and Applied Mathematics. Society for Industrial and Applied Mathematics, 1987.

[35] Pablo A. Parrilo. Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization. Technical report, California Institute of Technology, Pasadena, 2000.

[36] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96:293–320, 2003.

[37] Enric Rodríguez-Carbonell and Deepak Kapur. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Sci. Comput. Program.*, 64(1):54–75, 2007.

[38] Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 105–114, New York, NY, USA, 2012. ACM.

[39] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Non-linear loop invariant generation using gröbner bases. In *POPL*, 2004.

[40] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, pages 25–41, 2005.

[41] Markus Schweighofer. Optimization of polynomials on compact semialgebraic sets. *SIAM J. on Optimization*, 15(3):805–825, March 2005.

[42] Dinh Quoc Tran, Suat Gumussoy, Wim Michiels, and Moritz Diehl. Combining convex-concave decompositions and linearization approaches for solving BMIs, with application to static output feedback. *IEEE Trans. Automat. Contr.*, 57(6):1377–1390, 2012.

[43] Dinh Quoc Tran, Wim Michiels, Sebastien Gros, and Moritz Diehl. An inner convex approximation algorithm for BMI optimization and applications in control. In *CDC*, pages 3576–3581, 2012.

[44] J. G. Vanantwerp and R. D. Braatz. A tutorial on linear and bilinear matrix inequalities. *Journal of Process Control*, 10(4):363–385, August 2000.

[45] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Rev.*, 38(1):49–95, March 1996.

[46] V. A. Yakubovich. S-procedure in nonlinear control theory. *Vestnik Leningrad University*, 1971.