A Polynomial Template Abstract Domain based on Bernstein Polynomials

Pierre Roux[1,2] and Pierre-Loïc Garoche[1]

[1] ONERA – The French Aerospace Lab, Toulouse, FRANCE
[2] ISAE, Toulouse, FRANCE

Abstract. The analysis of reactive systems such as the command control of aircrafts requires the ability to analyze nonlinear systems and synthesize nonlinear invariants. However, in the static analysis state of the art, few analyses are applicable or effective when applied to those systems. We present a template abstract domain that relies on Bernstein polynomials to bound the reachable states of polynomial reactive systems with polynomial invariants.

1 Introduction

The most critical systems have to achieve a high level of quality. A wide category of such systems can be characterized as reactive systems. Such reactive systems are typically designed by control theorists to control a plant — a plane, a UAV or the thrust of an engine for instance — through an endless loop of commands: reading the input sensors, and computing the feedback on the actuators. The design and the V&V process of such systems widely rely on tests even if recent works from academia have been transfered into industry [27]. In [2] Air Force developed its vision of the Technology Horizons 2010-2030 and identified this issue as the second most important grand challenge of the years to come. NASA expressed similar interest through different projects or initiatives.

Static analysis, like abstract interpretation [7] or deductive methods [8,19] try to answer those needs and many publications, e.g. [27,11] address the use of tools to over-approximate the behaviors of those control command systems relying on various abstractions, mainly linear and convex, from intervals to more costly ones, octagons or convex polyhedra.

However the analysis of such systems, even of simplest of those, often requires to deal with nonlinear properties. In fact, even the open loop stable linear PID controllers do not admit in general a linear inductive invariant but a quadratic one. The software itself contains monitor systems or fault detection ones that also embed complex numerical cores, e.g. Kalman filters.

Recent works addressed more specifically those numerical cores of control-command systems [14,23] . They are however restricted to linear systems admitting quadratic invariants, while in general systems are more complex. It is therefore mandatory to leverage those static analysis techniques in order to (1) permit the analysis of higher level properties that cannot be expressed within convex linear values or to (2) allow the analysis of realistic reactive systems.

This paper presents a template abstract domain [26] that relies on Bernstein polynomials to bound the reachable states of polynomial reactive systems with polynomial invariants.

This polynomial representation admits a set of interesting properties for verification purposes. Our proposal describes how to rely on such polynomials to reason over systems expressed with polynomials. We also took care of the floating point implementation issues, providing a safe over-approximation of the reachable states of such systems. This contribution is a first step towards the general analysis of nonlinear systems, for example through polynomialization of system's equations via Taylor or Poisson expansions.

The paper is structured as follows. The remainder of this introduction section presents notations and a toy imperative language on which our latter analyses are applied. Then Section 2 offers a basic overview of Bernstein polynomials and how they can be used for static analysis purposes. Section 3 presents our instantiation of a template domain with polynomials. Finally Sections 4 and 5 compare the current approach with related work and conclude.

1.1 A Toy Imperative Language

Throughout this paper, a classical toy imperative language will be used to illustrate our abstract domain.

Notations Given some $n \in \mathbb{N}^*$, we denote n-tuples $(x_1, \ldots, x_n)$ by bold letters $\boldsymbol{x}$. $\mathbf{0}$ stands for the tuple $(0, \ldots, 0)$. We extend order relation $\leq$ on tuples: $\boldsymbol{x} \leq \boldsymbol{y}$ if for all i between 1 and n, $x_i \leq y_i$. The interval notation $[\mathbf{0}, \mathbf{1}]$ then represents all tuples $\boldsymbol{x}$ satisfying $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$. For $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{i} \in \mathbb{N}^n$, $\boldsymbol{x}^{\boldsymbol{i}}$ is the monomial $\prod_{j=1}^{n} x_j^{i_j}$. Finally $\mathbb{R}[\boldsymbol{x}]$ is the set of polynomials with coefficients in $\mathbb{R}$ over variables $x_1, \ldots, x_n$. Polynomials $p \in \mathbb{R}[\boldsymbol{x}]$ are of the form $p(\boldsymbol{x}) = \sum_{\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{d}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}$ where $\boldsymbol{d} \in \mathbb{N}^n$ is called *degree* of the multivariate polynomial p.

Syntax A program of the language is a statement *stm* in the following grammar:

$$\begin{aligned} \mathit{stm} ::= {} & x \mathbin{:=} \mathit{pol} \mid x \mathbin{:=} ?(r,\, r) \mid \mathit{stm};\ \mathit{stm} \\ & \mid \textbf{if } \mathit{pol} \leq 0 \textbf{ then } \mathit{stm} \textbf{ else } \mathit{stm} \textbf{ fi} \mid \textbf{while } \mathit{pol} \leq 0 \textbf{ do } \mathit{stm} \textbf{ od} \end{aligned}$$

with $x \in \mathbb{V} = \{ x_1, \ldots, x_n \}$ a set of variables and $\mathit{pol} \in \mathbb{R}[\boldsymbol{x}]$ polynomials on those variables. $?(r_1,\, r_2)$ represents a random choice of a real between $r_1 \in \mathbb{R}$ and $r_2 \in \mathbb{R}$ (useful to simulate inputs).

Collecting Semantics Semantics of statements $[\![s]\!](R) \subseteq \mathbb{R}^n$ for a statement s and a set of environments $R \subseteq \mathbb{R}^n$ is defined as:

$$\begin{aligned} & [\![x_i := p]\!](R) = \{(x_1, \ldots, x_{i-1}, p(\boldsymbol{x}), x_{i+1}, \ldots, x_n) \mid \boldsymbol{x} \in R\} \\ & [\![x_i := ?(\mathrm{r}_1, \mathrm{r}_2)]\!](R) = \{(x_1, \ldots, x_{i-1}, r, x_{i+1}, \ldots, x_n) \mid \boldsymbol{x} \in R, r \in \mathbb{R}, r_1 \leq r \leq r_2\} \\ & [\![s_1; s_2]\!](R) = [\![s_2]\!]([\![s_1]\!](R)) \\ & [\![p \bowtie 0]\!](R) = \{\boldsymbol{x} \in R \mid p(\boldsymbol{x}) \bowtie 0\} \text{ for } \bowtie \in \{ >, \geq, <, \leq \} \\ & [\![\text{if p} \leq 0 \text{ then } \mathrm{s}_1 \text{ else } \mathrm{s}_2 \text{ fi}]\!](R) = [\![s_1]\!]([\![p \leq 0]\!](R)) \cup [\![s_2]\!]([\![p > 0]\!](R)) \\ & [\![\text{while p} \leq 0 \text{ do s od}]\!](R) = [\![p > 0]\!](\mathrm{lfp}\,(X \mapsto R \cup X \cup [\![s]\!]([\![p \leq 0]\!](X)))) \end{aligned}$$

with lfp$f$ the least fixpoint of monotone function f on complete lattice $2^{\mathbb{R}^n}$.

Finally the semantics of a program *prog* is given by $[\![prog]\!](\mathbb{R}^n)$, i.e. semantics of statement *prog* starting from an unknown memory state.

We can notice that this semantics is given with operations over reals $\mathbb{R}$ whereas an actual program would compute using floating point values. This important issue will not be addressed in this paper and is left as future work.

```
x := 0; y := ?(0, 0.5);
while x ≤ 1 do
    y := y + 0.001 × (18x² − 18x + 3);
    x := x + 0.001;
    if y ≤ 0 then y := 0 else y := y fi od
```

Fig. 1. Example of program.

Example 1. Figure 1 presents an integrator - a basic constriction of typical command control systems - in this language performing a discrete integration of the polynomial $18x^2-18x+3$. On this example, the intervals abstract domain do not allow to bound the variable y, the polyhedra domain gives the bound $y \leq 3.5$[3] whereas the domain developed in this paper will allow to prove that $y \leq 0.833$ (the actual maximum being between 0.7901 and 0.7902).

2 Bernstein Polynomials Optimization

Bernstein polynomials were first introduced in 1912 by Sergei Natanovich Bernstein to constructively prove Weierstrass theorem. However, their use really took off in the early sixties with the work of Paul de Faget de Casteljau and Pierre Étienne Bézier to describe car bodyworks in the French car industry. They are now a fundamental tool in the field of Computer Aided Geometric Design and are also used in other domains, such as global optimization [15,21], for their nice properties when representing polynomials on closed intervals. The interested reader is referred to [13] for a detailed overview.

2.1 Bernstein Polynomials

Definition 1 (Bernstein basis). *Any polynomial* $p \in \mathbb{R}[\boldsymbol{x}]$ *can be written*

$$p(\boldsymbol{x}) = \sum_{\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{d}} b_{p,\boldsymbol{i}} B_{\boldsymbol{d},\boldsymbol{i}}(\boldsymbol{x}) \qquad \text{with} \qquad B_{\boldsymbol{d},\boldsymbol{i}}(\boldsymbol{x}) = \prod_{j=1}^{n} \binom{d_j}{i_j} x_j^{i_j} (1-x_j)^{d_j - i_j}.$$

When considering a polynomial p on the box $[\mathbf{0}, \mathbf{1}]$, the coefficients $b_{p,\boldsymbol{i}}$ of this representation have the interesting property of bounding p.

Property 1. For any polynomial p, for all $\boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]$,

$$\min \{b_{p,\boldsymbol{i}} \mid \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{d}\} \leq p(\boldsymbol{x}) \leq \max \{b_{p,\boldsymbol{i}} \mid \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{d}\}.$$

Moreover, $p(\boldsymbol{i}) = b_{p,\boldsymbol{i}}$ for all $\boldsymbol{i} \in \mathcal{C}_{\boldsymbol{d}}$ where we call *endpoints* the indices in the set $\mathcal{C}_{\boldsymbol{d}} = \{\boldsymbol{i} \in \mathbb{N}^n \mid \forall j, 0 \leq j \leq n \Rightarrow (i_j = 0 \vee i_j = d_j)\}$.

[3] When the nonlinear assignment $y := y + 0.001(18x^2 - 18x + 3)$ is abstracted by $y := y + [-0.018, 0]x + 0.003$ knowing that $0 \leq x \leq 1$.

This property can be generalized to any box $[\boldsymbol{a}, \boldsymbol{b}]$ by considering the polynomial $p'(\boldsymbol{x}) = p\left(\sigma_{[\boldsymbol{a},\boldsymbol{b}]}(\boldsymbol{x})\right)$ with $\sigma_{[\boldsymbol{a},\boldsymbol{b}]}$ mapping each x_i to $a_i + (b_i - a_i)x_i$.

2.2 Optimization Problem

We are targeting the following polynomial global optimization problem:

$$\max\{p(\boldsymbol{x}) \mid q_1(\boldsymbol{x}) \leq b_1 \wedge \ldots \wedge q_k(\boldsymbol{x}) \leq b_k\} \tag{1}$$

where $p, q_1, \ldots, q_k \in \mathbb{R}[\boldsymbol{x}]$ are multivariate polynomials such that for all $i \in \{1, \ldots, n\}$, the polynomials x_i and $-x_i$ are included in the set $\{q_1, \ldots, q_k\}$ and $b_1, \ldots, b_k$ are constant bounds in $\mathbb{R}$. That is the maximum value reached by a polynomial on a feasible set defined by a box and potential polynomial constraints. This can be $-\infty$ if the feasible set is empty and a value in $\mathbb{R}$ otherwise.

Definition 2. *We will later denote formula 1 as* $\mathrm{Opt}(p; q_1, \ldots, q_k)(b_1, \ldots, b_k)$.

Example 2. For instance $\mathrm{Opt}\left(x_1 + x_2; x_1, -x_1, x_2, -x_2, x_1^2 - x_2\right)(1, 0, 1, 0, 0) = 1$ and $\mathrm{Opt}\left(x_1 + x_2; x_1, -x_1, x_2, -x_2, x_1^2 - x_2\right)(1, 0, 1, 0, -2) = -\infty$.

2.3 Branch and Bound Algorithm

Our goal is to compute an upper bound of the previously defined value (1) within some accuracy ϵ. Let us assume we are considering polynomial p under polynomial constraints $q_1 \leq 0, \ldots, q_k \leq 0$ on the unit box $[\mathbf{0}, \mathbf{1}]$ (this is equivalent to an arbitrary box $[\boldsymbol{a}, \boldsymbol{b}]$ up to an affine transformation). According to Property 1, if there exist a q_j such that $\min\left\{b_{q_j,\boldsymbol{i}} \;\middle|\; \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{d}\right\} > 0$ then the constraint $q_j \leq 0$ is not satisfiable on the box $[\mathbf{0}, \mathbf{1}]$, hence the result $-\infty$. Otherwise $\mathit{bound} := \max\{b_{p,\boldsymbol{i}} \mid \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{d}\}$ is an upper bound of the result. And according to the second part of Property 1, $\mathit{reached} := \max\left\{b_{p,\boldsymbol{i}} \;\middle|\; \boldsymbol{i} \in \mathcal{C}_{\boldsymbol{d}}, \forall j, b_{q_j,\boldsymbol{i}} \leq 0\right\}$ is a lower bound. If $\mathit{bound} - \mathit{reached} \leq \epsilon$ then *bound* is an upper bound with the expected accuracy and we are done, otherwise we subdivide the box $[\mathbf{0}, \mathbf{1}]$ into two smaller boxes and recursively apply the same criteria on them.

Given the $d + 1$ Bernstein coefficients $b_{p,i}$ of a single variable polynomial p of degree d, the *de Casteljau's algorithm* computes the Bernstein coefficients of polynomials p_{left} and p_{right} such that $p_{left}(x) = p(\frac{x}{2})$ and $p_{right}(x) = p(\frac{x+1}{2})$ for all $x \in [0, 1]$. That is, the polynomial p on $[0, 1]$ is *subdivided* into p_{left} on $[0, \frac{1}{2}]$ and p_{right} on $[\frac{1}{2}, 1]$. This is performed in $\Theta(d^2)$ arithmetic operations and can be done on any variable for a multivariate polynomial. The interested reader is referred to [20] for more details.

Using this, algorithms 1 and 2 allow to numerically solve the optimization problem (1) within some accuracy ϵ[4]. The convergence of this procedure is known to be quadratic which means tight bounds can "rapidly" be obtained through it. This algorithm was already presented in [21] along with heuristics to improve its performance.

Algorithm 1 Auxiliary function for max_under_constraints (Algorithm 2). Polynomial p is the objective function whereas *constraints* is a set of polynomial constraints. $\boldsymbol{d}$ is the degree of those polynomials. k and s are respectively current and maximum allowed recursion depth while ϵ is the required accuracy. Finally *reached* is a lower bound of final result used to prune the search tree. The returned value is a pair (r, b) satisfying $r \leq \max\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}] \wedge \forall c \in constraints, c(\boldsymbol{x}) \leq 0\} \leq b$. multipoly is the type of multivariate polynomials.

max_under_constraints_rec(p : multipoly, *constraints* : multipoly set, $\boldsymbol{d}$: int tuple, s : int, k : int, ϵ : real, *reached* : real)
  $this\_reached \leftarrow \max\{b_{p,\boldsymbol{i}} \mid \boldsymbol{i} \in \mathcal{C}_{\boldsymbol{n}}, \forall c \in constraints, b_{c,\boldsymbol{i}} \leq 0\}$
    ▷ Invariant: $this\_reached \leq \max\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}] \wedge \forall c \in constraints, c(\boldsymbol{x}) \leq 0\}$.
  if $\exists c \in constraints, \min\{b_{c,\boldsymbol{i}} \mid \boldsymbol{0} \leq \boldsymbol{i} \leq \boldsymbol{d}\} > 0$ **then**
    $this\_bound \leftarrow -\infty$ ▷ Constraint c is not satisfiable.
  else
    $this\_bound \leftarrow \max\{b_{p,\boldsymbol{i}} \mid \boldsymbol{i} \leq \boldsymbol{n}\}$
  end if
    ▷ Invariant: $\max\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}] \wedge \forall c \in constraints, c(\boldsymbol{x}) \leq 0\} \leq this\_bound$.
  if $k \geq s \vee this\_bound \leq this\_reached + \epsilon \vee this\_bound \leq reached + \epsilon$ **then**
    ▷ If maximum recursion depth or required accuracy is reached or if greater value
    ▷ was already reached on a previously explored part of the domain, stop here.
    return $(this\_reached, this\_bound)$
  end if
  $j \leftarrow$ varsel$(p, \boldsymbol{d}, k)$ ▷ Function varsel returns the index j of one of the ▷ n variables x_j of polynomials p and *constraints*.
  $\{l\}, \{r\} \leftarrow$ subdiv$(\{p\}, j)$ ▷ Function subdiv(*pols*, j) applies
  $cl, cr \leftarrow$ subdiv$(constraints, j)$ ▷ de Casteljau's algorithm to each ▷ polynomial in set *pols* on variable x_j.
  $reached \leftarrow \max(reached, this\_reached)$
  $rl, bl \leftarrow$ max_under_constraints_rec$(l, cl, \boldsymbol{d}, s, k+1, \epsilon, reached)$
  $reached \leftarrow \max(reached, rl)$
  $rr, br \leftarrow$ max_under_constraints_rec$(r, cr, \boldsymbol{d}, s, k+1, \epsilon, reached)$
  return $(\max(rl, rr), \max(bl, br))$

Algorithm 2 max_under_constraints$(p, constraints, \boldsymbol{lbs}, \boldsymbol{ubs}, s, \epsilon)$ computes an upper bound of $\max\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\boldsymbol{lbs}, \boldsymbol{ubs}] \wedge \forall c \in constraints, c(\boldsymbol{x}) \leq 0\}$, algorithm stops when either a bound of accuracy ϵ or recursion depth s is reached.

max_under_constraints(p : multipoly, *constraints* : multipoly set, $\boldsymbol{lbs}$: int tuple, $\boldsymbol{ubs}$: int tuple, s : int, ϵ : real)
  $p, constraints, \boldsymbol{d} \leftarrow$ translate$(p, constraints, \boldsymbol{lbs}, \boldsymbol{ubs})$
    ▷ Function translate first translates polynomials from box $[\boldsymbol{lbs}, \boldsymbol{ubs}]$ to the ▷ unit box $[\boldsymbol{0}, \boldsymbol{1}]$ then to Bernstein basis, it also returns their degree $\boldsymbol{d}$.
  $\_, bound \leftarrow$ max_under_constraints_rec$(p, constraints, \boldsymbol{d}, s, 0, \epsilon, -\infty)$
  return *bound*

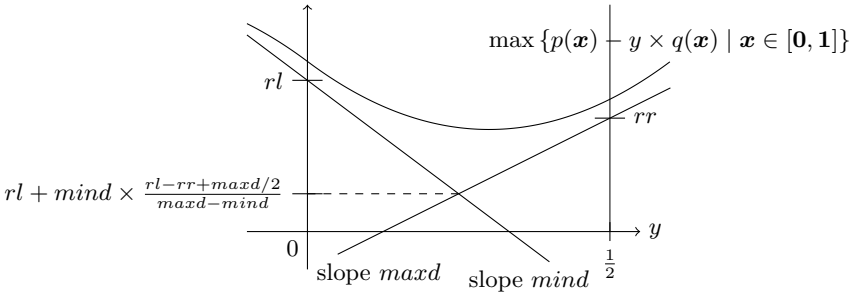


Fig. 2. Illustration of Property 3.

Algorithm 3 Auxiliary function for max_relaxation (Algorithm 4). pyq is the polynomial $p - yq$ with p the objective function and q the constraint. $\boldsymbol{d}$ is the degree of this polynomial. k and s are respectively current and maximum recursion depth. ϵ is the accuracy within which upper bounds are computed for each value of y tested. rl and rr are lower bound of the maximum of pyq for respective values 0 and 1 of y. $mind$ and $maxd$ are bounds on $\frac{d(pyq)}{dy}$. The result is an upper bound of pyq for some value of $y \in (0, 1)$.

max_relaxation_rec(pyq : multipoly, $\boldsymbol{d}$: int tuple, s : int, k : int, ϵ : real, rl : real, rr : real, $mind$: real, $maxd$: real)
  $l, r \leftarrow$ subdiv(pyq, y)
    ▷ First get a bound for $y = \frac{1}{2}$.
  $this\_reached, this\_bound \leftarrow$ max_under_constraints_rec($r[y \leftarrow 0], \emptyset, \boldsymbol{d}, s, 0, \epsilon, -\infty$)
    ▷ Then look for a smaller bound with $y \in \left(0, \frac{1}{2}\right)$.
  if $k + 1 < s \ \wedge \ rl + mind \times \frac{rl - this\_reached + maxd/2}{maxd - mind} < bound - \epsilon$ **then**
    $bl \leftarrow$ max_relaxation_rec($l, \boldsymbol{d}, s, k + 1, \epsilon, rl, this\_reached, mind/2, maxd/2$)
    if $bl < this\_bound$ **then**
      $this\_bound \leftarrow bl$
    end if
  end if
    ▷ Finally look for a smaller bound with $y \in \left(\frac{1}{2}, 1\right)$.
  if $k + 1 < s \ \wedge \ rr - maxd \times \frac{rr - this\_reached - mind/2}{maxd - mind} < bound - \epsilon$ **then**
    $br \leftarrow$ max_relaxation_rec($r, \boldsymbol{d}, s, k + 1, \epsilon, this\_reached, rr, mind/2, maxd/2$)
    if $br < this\_bound$ **then**
      $this\_bound \leftarrow br$
    end if
  end if
  return $this\_bound$

Algorithm 4 max_relaxation($p, q, \boldsymbol{lbs}, \boldsymbol{ubs}, s, relaxdomain, \epsilon$) computes an upper bound of $\max\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\boldsymbol{lbs}, \boldsymbol{ubs}] \wedge q(\boldsymbol{x}) \leq 0\}$. To do this, it looks for a non negative real $y0$ such that the maximum of $p - y0 \times q$ is the smallest possible. An upper bound of $p - y0 \times c$ is returned. s determines the number of values tried for $y0$ ($2^s + 1$ value equally spaced between 0 and $relaxdomain$, including 0 and $relaxdomain$) whereas ϵ determines the precision with which the bound is computed for each try for $y0$.

max_relaxation(p : multipoly, q : multipoly, $\boldsymbol{lbs}$: int tuple, $\boldsymbol{ubs}$: int tuple, s : int, $relaxdomain$: real, ϵ : real)
  $mind \leftarrow$ max_under_constraints$(q, \emptyset, \boldsymbol{lbs}, \boldsymbol{ubs}, s, \epsilon)$
  $maxd \leftarrow$ max_under_constraints$(-q, \emptyset, \boldsymbol{lbs}, \boldsymbol{ubs}, s, \epsilon)$
  $pyq \leftarrow p - y \times q$ ▷ y is a fresh variable
  $pyq, \_, \boldsymbol{d} \leftarrow$ translate$(pyq, \emptyset, (0, \boldsymbol{lbs}), (relaxdomain, \boldsymbol{ubs}))$ ▷ $y \in [0, relaxdomain]$
  $rl, bl \leftarrow$ max_under_constraints_rec$(pyq[y \leftarrow 0], \emptyset, \boldsymbol{d}, s, \epsilon, -\infty)$
  $rr, br \leftarrow$ max_under_constraints_rec$(pyq[y \leftarrow 1], \emptyset, \boldsymbol{d}, s, \epsilon, -\infty)$
  if $bl < br$ **then**
    $bound \leftarrow bl$
  else
    $bound \leftarrow br$
  end if
  if $mind < 0 \wedge maxd > 0$ **then**
    $this\_bound \leftarrow$ max_relaxation_rec$(pyq, \mathbf{d}, s, 0, \epsilon, rl, rr, mind, maxd)$
  end if
  return min$(bound, this_bound)$

2.4 Lagrangian Relaxation

Property 2 (Relaxation scheme). For any polynomials p and q, any box $[\boldsymbol{a}, \boldsymbol{b}]$ and any real c, if there exist a real $y \geq 0$ such that:

$$\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}], p(\boldsymbol{x}) - y \times q(\boldsymbol{x}) \leq c$$

then:

$$\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}], q(\boldsymbol{x}) \leq 0 \Rightarrow p(\boldsymbol{x}) \leq c.$$

The reverse direction does not hold but if for some real y we bound the polynomial $p - y \times q$ on the box $[\boldsymbol{a}, \boldsymbol{b}]$, this bound also holds for $\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}] \wedge q(\boldsymbol{x}) \leq 0\}$. By trying various values for y, we can then obtain a bound on the constrained problem while only computing bounds of polynomials without constraints.

The following property will allow to safely ignore some intervals while looking for appropriate values for relaxation variable y.

Property 3. For any polynomials p and q, if there exist $mind, maxd, rl, rr \in \mathbb{R}$ such that:

- $\forall \boldsymbol{x} \in [\mathbf{0}, \mathbf{1}],\ mind \leq -q(\boldsymbol{x}) \leq maxd$;
- $\max\{p(\boldsymbol{x}) \mid \boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]\} \geq rl$ and $\max\left\{p(\boldsymbol{x}) - \frac{1}{2} \times q(\boldsymbol{x}) \;\middle|\; \boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]\right\} \geq rr$;

[4] Provided a big enough bound s on recursion depth.

then, for all $y \in \left[0, \frac{1}{2}\right]$:

$$\max\{p(\boldsymbol{x}) - y \times q(\boldsymbol{x}) \mid \boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]\} \geq rl + mind \times \frac{rl - rr + maxd/2}{maxd - mind}.$$

Proof. The result follows from the fact that $\frac{d(p-yq)}{dy} = -q$ is bounded by $mind$ and $maxd$.

This property, illustrated on Figure 2, allows one to prune branches while looking for a value of y leading to an interesting bound as done in Algorithms 3 and 4. This can be generalized to more than one constraint by adding as many relaxation variables y as constraints.

To the extent of authors knowledge, this algorithm is new. It is an alternative to the one of the previous Section 2.3 which appears to be more efficient in cases when the objective function is close to the optimal value on a large part of the edge of the feasible space. Those cases can indeed require a high number of subdivisions to test the satisfiability of a constraint which can be avoided by the relaxation. Moreover, they can typically appear when analyzing reactive systems where a single iteration of the system induces relatively small changes. However, unlike the previous algorithm, it cannot guarantee the accuracy of the upper bound it returns.

3 Polynomial Template Abstract Domain

This section presents an abstract domain working on polynomial templates. The definition of the domain, the abstract operators and possible widenings are rather standard for template domains [26]. The main interest lies in the possibility to precisely handle polynomial templates and assignments.

3.1 Lattice Structure

Given a set $P = \{ p_1, \ldots, p_k \}$ of $k \in \mathbb{N}$ polynomials over $n \in \mathbb{N}$ variables, the set $\mathcal{T}_P$ of tuples $\boldsymbol{b} \in \overline{\mathbb{R}}^k$ forms a complete lattice when ordered by $\sqsubseteq^\sharp$, the pointwise extension of the usual order on $\overline{\mathbb{R}} = \mathbb{R} \cup \{ -\infty, +\infty \}$ (i.e. $(b_1, \ldots, b_k) \sqsubseteq^\sharp (b'_1, \ldots, b'_k)$ if: $\forall i, 1 \leq i \leq k \Rightarrow b_i \leq b'_i$). The infimums are then $\top = (+\infty, \ldots, +\infty)$ and $\bot = (-\infty, \ldots, -\infty)$ and the least upper bound $\sqcup^\sharp$ and greatest lower bound $\sqcap^\sharp$ are the pointwise extensions of respectively max and min on $\overline{\mathbb{R}}$. This complete lattice defines an abstraction of $2^{\mathbb{R}^n}$ thanks to the concretization function $\gamma : \mathcal{T}_P \to 2^{\mathbb{R}^n}$ such that:

$$\gamma(\boldsymbol{b}) = \{\boldsymbol{x} \in \mathbb{R}^n \mid p_1(\boldsymbol{x}) \leq b_1 \wedge \ldots \wedge p_k(\boldsymbol{x}) \leq b_k\}.$$

Example 3. Figure 3 presents an abstract value in $\mathcal{T}_P$.

As can be noticed on Figure 3, the concretization of a value in $\mathcal{T}_P$ is a subset of $\mathbb{R}^n$ that is not necessarily convex. This is an uncommon feature since most numerical abstract domains are convex (there however exist a few non convex numerical abstract domains, for instance tropical polyhedra [3] which are convex in tropical algebra but not in classical algebra).

$$\begin{aligned} x &\leq 1.001 \\ -x &\leq 0 \\ y &\leq 0.833 \\ -y &\leq 0 \\ y - 6x^3 + 9x^2 - 3.2x &\leq 0.5 \end{aligned}$$

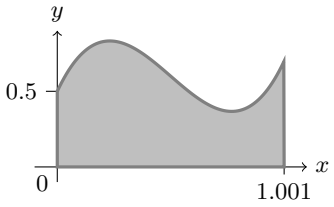


Fig. 3. Example of abstract value $(1, 0, 0.833, 0, 0.5) \in \mathcal{T}_P$ with $P = \{ x, -x, y, -y, y - 6x^3 + 9x^2 - 3.2x \}$. This value is an invariant at loop head for the program of Figure 1.

3.2 Abstract Operators

To be able to perform abstract interpretation analyses with our new domain on programs defined in Section 1.1, we now need to define abstract operators for guards, assignments and random assignments.

Guards We define the abstract semantic of a guard $r(\boldsymbol{x}) \leq 0$ on $\boldsymbol{b} \in \mathcal{T}_P$ with r a n variable polynomial in $\mathbb{R}[\boldsymbol{x}]$ as:

$$[\![r(\boldsymbol{x}) \leq 0]\!]^\sharp (\boldsymbol{b}) = \boldsymbol{b'} \qquad \text{with} \qquad b'_i = \mathrm{Opt}\,(p_i; p_1, \ldots, p_k, r)\,(b_1, \ldots, b_k, 0)\,.$$

Example 4. Still using set of templates $P = \{ x, -x, y, -y, y - 6x^3 + 9x^2 - 3.2x \}$, $[\![x - 1 \leq 0]\!]^\sharp (10, 0, 10, 0, 0.5) = (1, 0, 0.833, 0, 0.5)$. This is illustrated on (k) and (l) in Figure 4.

Property 4 (soundness). This abstract operator is sound with respect to the concrete semantics of guards: for all polynomial $r \in \mathbb{R}[\boldsymbol{x}]$ and all $\boldsymbol{b} \in \mathcal{T}_P$, $[\![r \leq 0]\!] (\gamma(\boldsymbol{b})) \subseteq \gamma \left([\![r \leq 0]\!]^\sharp (\boldsymbol{b}) \right)$.

Assignments We define the abstract semantic of an assignment $x_i := r$ on $\boldsymbol{b} \in \mathcal{T}_P$ with r a n variable polynomial in $\mathbb{R}[\boldsymbol{x}]$ as:

$$[\![x_i := r]\!]^\sharp (\boldsymbol{b}) = \boldsymbol{b'} \qquad \text{with} \qquad b'_j = \mathrm{Opt}\,(p_j[x_i \leftarrow r(\boldsymbol{x})]; p_1, \ldots, p_k)\,(\boldsymbol{b})\,.$$

Example 5. $[\![y := 0]\!]^\sharp (1.001, -0.001, 0, 0.002, 0.133) = (1.001, -0.001, 0, 0, 0.133)$. This is illustrated on (n) and (o) in Figure 4.

We can notice that this definition can (and in practice will) increase the degree of objective polynomials of the optimization problems to solve. This has an impact on the cost of solving those problems. This cost is then dependent on the composition of degrees of polynomials of the set of templates and polynomials appearing in the analyzed program.

Property 5 (soundness). This abstract operator is sound with respect to the concrete semantics of assignments: for all variables x_i, for all polynomial $r \in \mathbb{R}[\boldsymbol{x}]$ and all $\boldsymbol{b} \in \mathcal{T}_P$, $[\![x := r]\!] (\gamma(\boldsymbol{b})) \subseteq \gamma \left([\![x := r]\!]^\sharp (\boldsymbol{b}) \right)$.

Random assignments We define the abstract semantic of a random assignment $x_i := ?(r_1, r_2)$ on $\boldsymbol{b} \in \mathcal{T}_P$ with $r_1, r_2 \in \mathbb{R}$ as:

$$[\![x_i := ?(r_1, r_2)]\!]^\sharp(\boldsymbol{b}) = \rho(\boldsymbol{b'}) \qquad \text{with} \qquad b'_i = \begin{cases} r2 & \text{if } p_i = x_i \\ -r1 & \text{if } p_i = -x_i \\ +\infty & \text{otherwise, if } x_i \text{ appears in } p_i \\ b_i & \text{otherwise.} \end{cases}$$

where $\rho : \mathcal{T}_P \to \mathcal{T}_P$ is defined as: $\rho(\boldsymbol{b}) = \boldsymbol{b'}$ with $b'_i = \mathrm{Opt}(p_i; p_1, \ldots, p_k)(\boldsymbol{b})$.

Example 6. $[\![y := ?(0, 0.5)]\!]^\sharp(0, 0, +\infty, +\infty, +\infty) = (0, 0, 0.5, 0, 0.5)$. This is illustrated on (b) and (c) in Figure 4.

Property 6 (soundness). This abstract operator is sound with respect to the concrete semantics of assignments: for all variables x_i, for all $r_1, r_2 \in \mathbb{R}$ and all $\boldsymbol{b} \in \mathcal{T}_P$, $[\![x := ?(r_1, r_2)]\!](\gamma(\boldsymbol{b})) \subseteq \gamma\left([\![x := ?(r_1, r_2)]\!]^\sharp(\boldsymbol{b})\right)$.

3.3 Widening

The domain $\mathcal{T}_P$ has infinite ascending chains. A widening is then required to enforce termination of the analyses.

Assuming any widening ∇_s on lattice $\overline{\mathbb{R}}$ equipped with the usual order, its pointwise extension gives a widening on $\mathcal{T}_P$.

However, since the implementation with Bernstein polynomials definitely jumps to $\top$ as soon as one of the bounds b_i with p_i of the form x_j or $-x_j$ is infinite, a widening not jumping directly to $+\infty$, such as a widening with thresholds, is actually required to get any practical result.

3.4 Implementation considerations

Optimization problems $\mathrm{Opt}(p; q_1, \ldots, q_k)(\boldsymbol{b})$ can be solved (within some accuracy ϵ) with algorithms of Sections 2.3 or 2.4. However, those algorithms require each variable x_i appearing in polynomials $p, q_1, \ldots, q_k$ to be bounded. That is, polynomials x_i and $-x_i$ have to be among $q_1, \ldots, q_k$ and the bounds associated to them in tuple $\boldsymbol{b}$ must be in $\mathbb{R}$ (i.e. neither $-\infty$ nor $+\infty$). In case one of those bounds is $-\infty$, we just returns $-\infty$ which is the exact result. But in case $+\infty$, we have to conservatively overapproximate the result either by acting as the identity function in the case of guards[5] or by returning $+\infty$ in the case of assignments.

Parameter ϵ unfortunately plays a key role in the trade off between precision and cost of the analysis. A large value enables a faster analysis but may result in a far less precise result if for instance we miss a fixpoint whereas a small value leads to more precise result but induces a more costly analysis. In contrast, parameters s and *relaxdomain* of algorithm of Section 2.4, defining the domain in which

[5] Since guards do not modify any variable, the identity function is always a sound, although very coarse, overapproximation.

the relaxation value is looked for, play a less critical role since this domain is rapidly cut down, preventing a large domain to induce much overhead.

Last but not least the choice of arithmetic for all computations performed by algorithms of Section 2 is critical for soundness of the analyses and can have an important impact on performances. Floating point arithmetic is definitely the fastest but is not an option since it doesn't guarantee soundness of the result. Rational arithmetic with arbitrary precision integers is sound but expensive. A workaround could be to compute potentially unsound results with floating point and check the final result with a rational implementation. There even exists such an implementation fully proved in the theorem prover PVS [20][6]. But finally, we found out that a good compromise is to use floating point interval arithmetic [24]. This is only about two times slower than a pure floating point – unsound – implementation. Given the good numerical stability properties of Bernstein polynomials [13, §6], this works very well in practice.

It is also interesting to notice that those branch and bound algorithms should be rather easy to parallelize.

3.5 Example

Example 7. Figure 4 displays an analysis on the running example (Figure 1), still with $P = \{ x, -x, y, -y, y - 6x^3 + 9x^2 - 3.2x \}$ and with a widening with thresholds $\{ 10, 100, 1000 \}$. A descending iteration from the fixpoint obtained then gives the invariant displayed on Figure 3 at loop head.

On this example, the template polynomial $y - 6x^3 + 9x^2 - 3.2x$ allows one to bound the variable y by 0.833 while an analysis with the polyhedra domain yields $y \leq 3.5$. This is rather tight since the actual maximal value can be proven by hand to be between 0.788 and 0.792.

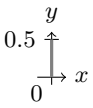
It is interesting to notice that consecutive assignments are considered together (Figures 4(e) and 4(m)). This is an easy way to improve both the precision of the analysis (by avoiding intermediate abstraction, hence loss of precision[7]) and its cost (by avoiding redundant computations). An alternative solution would have been to use the mechanism proposed in [26] to compute local templates in a similar way to a weakest precondition calculus [12], but this offers only the precision and not the cost improvement of previous solution.

This analysis takes 1.8s with the algorithm of Section 2.3 and 0.2s with the alternative algorithm of Section 2.4, both implemented with floating point interval arithmetic and running on an Intel Core2 @ 2.66GHz.

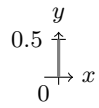
[6] The implementation presented in this paper only allows one to check the results of alternative algorithm of Section 2.4 but, according to personal communication with the authors, they more recently implemented a more general version of the algorithm of Section 2.3, allowing to also check the results of this algorithm.

[7] See for instance [16] for a more detailed discussion on this point.

(a) $\top$

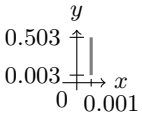


(b) $[\![x := 0]\!]^\sharp(\text{a}) = (0, 0, +\infty, +\infty, +\infty)$



(c) $[\![y := ?(0, 0.5)]\!]^\sharp(\text{b}) = (0, 0, 0.5, 0, 0.5)$

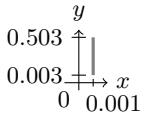
(d) $[\![x - 1 \leq 0]\!]^\sharp(\text{c}) = (0, 0, 0.5, 0, 0.5)$



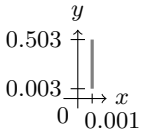
(e) $[\![y := y + 0.001(18x^2 - 18x + 3); x := x + 0.001]\!]^\sharp(\text{d}) = (0.001, -0.001, 0.503, -0.003, 0.5)$

(f) $[\![y \leq 0]\!]^\sharp(\text{e}) = \bot$

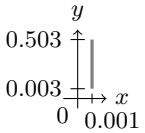
(g) $[\![y := 0]\!]^\sharp(\text{f}) = \bot$



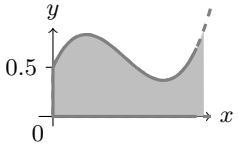
(h) $[\![y \geq 0]\!]^\sharp(\text{e}) =$ (e)



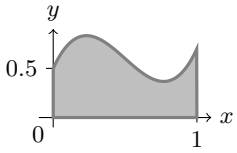
(i) $[\![y := y]\!]^\sharp(\text{h}) =$ (e)



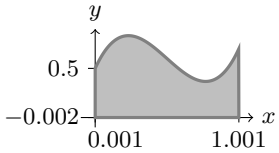
(j) (g) $\sqcup^\sharp$ (i) = (e)



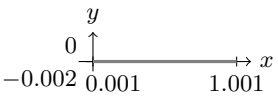
(k) (c) ∇ (j) $= (10, 0, 10, 0, 0.5)$



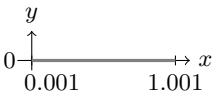
(l) $[\![x - 1 \leq 0]\!]^\sharp(\text{k}) = (1, 0, 0.833, 0, 0.5)$



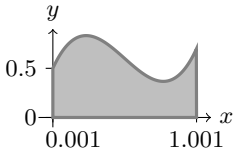
(m) $[\![y := y + 0.001(18x^2 - 18x + 3); x := x + 0.001]\!]^\sharp(\text{l}) = (1.001, -0.001, 0.833, 0.002, 0.5)$



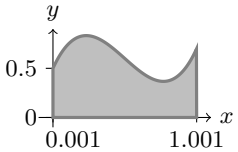
(n) $[\![y \leq 0]\!]^\sharp(\text{m}) = (1.001, -0.001, 0, 0.002, 0.133)$



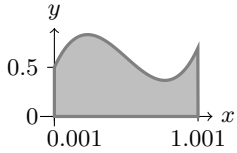
(o) $[\![y := 0]\!]^\sharp(\text{n}) = (1.001, -0.001, 0, 0, 0.133)$



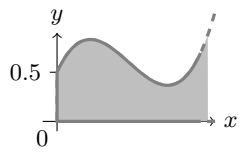
(p) $[\![y \geq 0]\!]^\sharp(\text{m}) = (1.001, -0.001, 0.833, 0, 0.5)$



(q) $[\![y := y]\!]^\sharp(\text{p}) =$ (p)



(r) (o) $\sqcup^\sharp$ (q) = (p)



(s) (k) ∇ (r) = (k) (fixpoint reached!)

Fig. 4. Example of analysis on the running example (Figure 1).

4 Related Work

Very few works address the analysis of nonlinear properties; and, to the authors' knowledge, none of them previously addressed the computation of bounds for nonlinear systems.

The use of numerical optimization procedures in template domains was introduced by Sankaranarayanan, Sipma and Manna [26]. They used linear programming to implement template polyhedra but already mentioned the possibility to use semidefinite programming to infer nonlinear invariants.

Feret [14] already generates quadratic invariants but use of semidefinite programming for general quadratic template domains was developed by Adjé, Gaubert and Goubault [1] and Gawlitza and Seidl [17]. The use of semidefinite programming is definitely more efficient than branch and bound algorithms with Bernstein polynomials on this common subclass of quadratic invariants. Moreover, policy iterations techniques [1,17] avoid the loss of precision due to the widening. However, this technique does not deal with other properties than quadratic properties for linear systems.

Sankaranarayanan, Sipma and Manna [25] and Rodriguez-Carbonell and Kapur [22] use Gröbner bases to generate invariant polynomial equalities, hence a disjoint class of properties compared to the one considered in this paper. Cachera, Jensen, Jobin and Kirchner [5] offer an alternative – more efficient – algorithm not using Gröbner bases but still targeting equalities.

Bagnara, Rodriguez-Carbonell and Zaffanella [4] generate polynomial invariants. They only applied the technique to quadratic invariants but seem confident that some improvements could leverage it to at least cubic invariants. The huge advantage of this work compared to the one presented in this paper is that it is fully automatic, not requiring polynomial templates to be given prior to any analysis. However, the techniques are of very different nature and it is expectable that each one is able to generate invariants which won't be found by the other.

This work is not the first one to make use of Bernstein polynomials in the scope of verification. Clauss and Tchoupaeva [6] already offered to use Bernstein polynomials to get linear approximations of polynomials on some interval for compilation purpose.

Finally Dang and Salinas [9] and Dang and Testylier [10] also used them for hybrid system analysis. A major difference with the present proposal is that, since they have to perform a very high number of iterations, they cannot afford branch and bound algorithms. That is why templates are limited to linear templates handled with some smart and efficient method.

5 Conclusion and Perspectives

This paper proposed a polynomial template domain relying on Bernstein polynomials. To the best of authors' knowledge it is the first template abstract domain able to deal with arbitrary polynomial systems admitting polynomial invariants.

Existing works [1,17,18] already proposed to rely on template domains to compute polynomial invariants, using a combination of policy iterations with semidefinite programming, but they were restricted to quadratic invariants on linear programs.

In the author's view, the main weakness of the current paper is the need of templates to perform the analysis. This weakness is however shared by all existing works (including the ones above) addressing nonlinear properties with abstract domains. In [23] we proposed a technique to synthesize meaningful quadratic templates. An identified future work would be to adapt this approach to general polynomial systems.

On the efficiency/precision issue, it would be very interesting to replace Kleene iterations by policy iterations. However, this does not look very tractable. An alternative solution could be to stick to Kleene iterations but to analyze code "en bloc". Some specific classes of polynomials could also be handled differently. For instance in the particular case of convex polynomials, convex optimization would certainly be more efficient than branch and bound algorithms with Bernstein polynomials.

To conclude, this work is a first contribution using Bernstein polynomials within an abstract domain. Lot of directions are opened to increase the efficiency of this proposal or ease its application. An exciting future work enabled by this precise analysis of polynomial systems is the analysis of more general nonlinear functions that could be abstracted by polynomials through a Taylor or Poisson expansion. This is our next target.

References

1. Assalé Adjé, Stéphane Gaubert, and Éric Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In *ESOP*, 2010.
2. United States Air Force Chief Scientist (AF/ST). Report on technology horizons - a vision for air force science & technology during 2010–2030, af/st-tr-10-01-pr. Technical report, Air Force, 2010.
3. Xavier Allamigeon, Stephane Gaubert, and Eric Goubault. Inferring min and max invariants using max-plus polyhedra. In *SAS*, 2008.
4. Roberto Bagnara, Enric Rodríguez-Carbonell, and Enea Zaffanella. Generation of basic semi-algebraic invariants using convex polyhedra. In *SAS*, 2005.
5. David Cachera, Thomas P. Jensen, Arnaud Jobin, and Florent Kirchner. Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases. In Antoine Miné and David Schmidt, editors, *SAS*, volume 7460 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2012.
6. Philippe Clauss and Irina Tchoupaeva. A symbolic approach to Bernstein expansion for program analysis and optimization. In *CC*, 2004.
7. Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Why does Astrée scale up? *Formal Methods in System Design*, 35(3):229–264, 2009.
8. Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-C - a software analysis perspective. In *SEFM*, 2012.

9. Thao Dang and David Salinas. Image computation for polynomial dynamical systems using the Bernstein expansion. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 219–232. Springer, 2009.
10. Thao Dang and Romain Testylier. Reachability analysis using the Bernstein expansion over polyhedra. In *Numerical Software Verification*, 2012.
11. Michael Dierkes. Formal analysis of a triplex sensor voter in an industrial context. In *FMICS*, 2011.
12. Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, 1975.
13. Rida T. Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379 – 419, 2012.
14. Jérôme Feret. Static analysis of digital filters. In *ESOP*, number 2986, 2004.
15. Jürgen Garloff, Christian Jansson, and Andrew P Smith. Lower bound functions for polynomials. *Journal of Computational and Applied Mathematics*, 157(1):207 – 225, 2003.
16. Thomas Martin Gawlitza and David Monniaux. Improving strategies via SMT solving. In *ESOP*, 2011.
17. Thomas Martin Gawlitza and Helmut Seidl. Computing relaxed abstract semantics w.r.t. quadratic zones precisely. In *SAS*, 2010.
18. Thomas Martin Gawlitza, Helmut Seidl, Assalé Adjé, Stéphane Gaubert, and Eric Goubault. Abstract interpretation meets convex optimization. *J. Symb. Comput.*, 47(12), 2012.
19. Temesghen Kahsai and Cesare Tinelli. Pkind: A parallel k-induction based model checker. In *PDMC*, 2011.
20. César Muñoz and Anthony Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 2012.
21. P.S.V. Nataraj and M. Arounassalame. Constrained global optimization of multivariate polynomials using Bernstein branch and prune algorithm. *Journal of Global Optimization*, 49:185–212, 2011.
22. Enric Rodríguez-Carbonell and Deepak Kapur. Automatic generation of polynomial loop invariants: Algebraic foundations. In *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, ISSAC, 2004.
23. Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*. ACM, 2012.
24. Siegfried M. Rump. Verification methods: rigorous results using floating-point arithmetic. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 3–4, New York, NY, USA, 2010. ACM.
25. Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Non-linear loop invariant generation using Gröbner bases. POPL, 2004.
26. Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, 2005.
27. Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. Formal verification of avionics software products. In *FM*, 2009.