Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming

Patrick Cousot

École Normale Supérieure
45 rue d'Ulm, 75230 Paris cedex 05 (France)
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

Abstract. In order to verify semialgebraic programs, we automatize the Floyd/Naur/Hoare proof method. The main task is to automatically infer valid invariants and rank functions.
First we express the program semantics in polynomial form. Then the unknown rank function and invariants are abstracted in parametric form. The implication in the Floyd/Naur/Hoare verification conditions is handled by abstraction into numerical constraints by Lagrangian relaxation. The remaining universal quantification is handled by semidefinite programming relaxation. Finally the parameters are computed using semidefinite programming solvers.
This new approach exploits the recent progress in the numerical resolution of linear or bilinear matrix inequalities by semidefinite programming using efficient polynomial primal/dual interior point methods generalizing those well-known in linear programming to convex optimization.
The framework is applied to invariance and termination proof of sequential, nondeterministic, concurrent, and fair parallel imperative polynomial programs and can easily be extended to other safety and liveness properties.

Keywords: Bilinear matrix inequality (BMI), Convex optimization, Invariance, Lagrangian relaxation, Linear matrix inequality (LMI), Liveness, Parametric abstraction, Polynomial optimization, Proof, Rank function, Safety, S-procedure, Semidefinite programming, Termination precondition, Termination. Program verification.

1 Introduction

Program verification is based on reasonings by induction (e.g. on program steps) which involves the discovery of unknown inductive arguments (e.g. rank functions, invariants) satisfying universally quantified verification conditions. For static analysis the discovery of the inductive arguments must be automated, which consists in solving the constraints provided by the verification conditions. Several methods have been considered: recurrence/difference equation resolution; iteration, possibly with convergence acceleration; or direct methods (such

R. Cousot (Ed.): VMCAI 2005, LNCS 3385, pp. 1–24, 2005.
© Springer-Verlag Berlin Heidelberg 2005

as elimination). All these methods involve some form of simplification of the constraints by abstraction.

In this paper, we explore *parametric abstraction* and direct resolution by *Lagrangian relaxation* into *semidefinite programming*. This is applied to termination (a typical liveness property) of semialgebraic programs. The extension to invariance (a typical safety property) is sketched.

The automatic determination of loop invariant/rank function can be summarized as follows:

1. Establish the relational semantics of the loop body (Sec. 2) (may be strengthened with correctness preconditions (Sec. 2.2), abstract invariants (Sec. 2.3), and/or simplified by relational abstraction (Sec. 2.4));
2. Set up the termination/invariance verification conditions (Sec. 3);
3. Choose a parametric abstraction (Sec. 4). The resolution of the abstract logical verification conditions by first-order quantifier elimination can be considered, but is very often too costly (Sec. 5);
4. Abstract further the abstract logical verification conditions into numerical constraints (Sec. 8) by Lagrangian relaxation (Sec. 6) obtaining Linear Matrix Inequalities for termination (Sec. 6.2) or Bilinear Matrix Inequalities for invariance (Sec. 12);
5. Solve the numerical constraints (Sec. 9) by semidefinite programming (Sec. 7);

After a series of examples (Sec. 10), we consider more complex language features including disjunctions in the loop test and conditionals in the loop body (Sec. 11.1), nested loops (Sec. 11.2), nondeterminism and concurrency (Sec. 11.3), bounded weakly fair parallelism (Sec. 11.4), and semi-algebraic/polynomial programs, for which a further relaxation into a sum of squares is applied (Sec. 11.5). The case of invariance is illustrated in Sec. 12. Potential problems with solvers are discussed in Sec. 13, before concluding (Sec. 14).

2 Relational Semantics of Programs

2.1 Semialgebraic Programs

We consider numerical iterative programs `while` B `do` C `od` where B is a boolean condition and C is an imperative command (assignment, test or loop) on the global program variables. We assume that the operational semantics of the loop is given for an iteration as:

$$[\![B;C]\!](x_0, x) = \bigwedge_{k=1}^{N} \sigma_k(x_0, x) \geq 0 \tag{1}$$

where x_0 is the line vector of values of the n program variables before an iteration, x is the line vector of values of the n program variables after this iteration and the relationship between x_0 and x during a single iteration is expressed as a

conjunction of N real valued positivity constraints with $\sigma_k \in \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$, $k = 1, \ldots, N$ [1]. Algorithmically interesting particular cases are when the constraints $\sigma_k \geq 0$ can be expressed as linear constraints, quadratic forms and polynomial positivity. Equalities $\sigma_k(x_0, x) = 0$ have to be written as $\sigma_k(x_0, x) \geq 0 \wedge -\sigma_k(x_0, x) \geq 0$.

Example 1 (Factorial). The program below computes the greatest factorial less than or equal to a given `N`, if any. The operational semantics of the loop body can be defined by the following constraints:

```
n := 0;
f := 1;
while (f <= N) do
   n := n + 1;
   f := n * f
od
```

$$\begin{aligned} -f_0 + N_0 &\geq 0 \\ n_0 &\geq 0 \\ f_0 - 1 &\geq 0 \\ -n_0 + n - 1 &= 0 \\ -f_0.n + f &= 0 \\ -N_0 + N &= 0 \end{aligned}$$

All constraints are linear but $f - n.f_0 = 0$ which is quadratic (of the form $[x_0\, x]A[x_0\, x]^\top + 2[x_0\, x]\, q + r \geq 0$, where A is symmetric, q is a column vector, r is a constant, and $^\top$ is transposition), and can be written as follows:

$$[n_0\ f_0\ N_0\ n\ f\ N] \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} n_0 \\ f_0 \\ N_0 \\ n \\ f \\ N \end{bmatrix} + 2[n_0 f_0 N_0 n f N] \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{2} \\ 0 \end{bmatrix} + 0 = 0,$$

or equivalently as $[x_0\, x\, 1]M[x_0\, x\, 1]^\top \geq 0$ where M is symmetric, that is:

$$[n_0\ f_0\ N_0\ n\ f\ N\ 1] \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} n_0 \\ f_0 \\ N_0 \\ n \\ f \\ N \\ 1 \end{bmatrix} = 0\ .$$

□

2.2 Establishing Necessary Correctness Preconditions

Iterated Forward/Backward Static Analysis. Program verification may be unsuccessful when the program execution may fail under some circumstances

[1] Any Boolean constraint on numerical variables can be written in that form using an appropriate numerical encoding of the boolean values and embedding of the numerical values into $\mathbb{R}$.

(e.g. non termination, run-time errors). As part of the correctness proof, it is therefore mandatory to establish correctness preconditions excluding such misbehaviors. Such a necessary termination and absence of runtime errors precondition can be discovered automatically by an iterated forward/backward static analysis [7, 12].

Discovering a Termination Precondition by the Auxiliary Termination Counter Method. Termination requirements can be incorporated into the iterated forward/backward static analysis in the form of an *auxiliary termination counter* `k` which is strictly decremented in the loop and is asserted to be zero on loop exit. For relational analyzes, this strengthens the necessary termination precondition.

Example 2. The following example is from [3]. The analyzer uses the polyhedral abstract domain [14] implemented using the *New Polka* library [20].

```
while (x <> y) do
    x := x - 1;
    y := y + 1
od
```

```
{x>=y}
while (x <> y) do
   {x>=y+2}
   x := x - 1;
   {x>=y+1}
   y := y + 1
   {x>=y}
od
{x=y}
```

```
{x=y+2k,x>=y}
while (x <> y) do
   {x=y+2k,x>=y+2}
   k := k - 1;
   x := x - 1;
   y := y + 1
   {x=y+2k,x>=y}
od
{x=y,k=0}
assume (k = 0)
```

Program | Iterated forward/backward static analysis | Iterated forward/backward static analysis with termination counter.

The use of the auxiliary termination counter allows the iterated forward/backward polyhedral analysis to discover the necessary termination condition that the initial difference between the two variables should be even. □

2.3 Strengthening the Relational Semantics

Proofs can be made easier by strengthening the relational semantics through incorporation of known facts about the program runtime behavior. For example an invariant may be needed, in addition to a termination precondition, to prove termination. Such a loop invariant can be determined automatically by a forward static analysis [8] assuming the initial termination precondition.

Example 3 (Ex. 2 continued). A forward polyhedral analysis [14] yields linear invariants:

```
assume (x=y+2*k) & (x>=y);
while (x <> y) do
    k := k - 1;
    x := x - 1;
    y := y + 1
od
```

Program

```
assume (x=y+2*k) & (x>=y);
{x=y+2k,x>=y}
{loop invariant: x=y+2k}
while (x <> y) do
   {x=y+2k}
   k := k - 1;
   x := x - 1;
   y := y + 1
   {x=y+2k}
od
{k=0,x=y}
```

Forward polyhedral analysis. □

2.4 Abstracting the Relational Semantics

Program verification may be made easier when the big-step operational semantics of the loop body (1) is simplified e.g. through abstraction. To get such a simplified but sound semantics, one can use any relational abstraction. The technique consists in using auxiliary initial variables to denote the values of the program variables at the beginning of the loop iteration (whence satisfying the loop invariant and the loop condition). The invariant at the end of the loop body is then a sound approximation of the relational semantics (1) of the loop body[2]. The polyhedral abstraction [14] will be particularly useful to derive automatically an approximate linear semantics.

Example 4 (Ex. 3 continued). Handling the operator <> (different) by case analysis, we get[3]:

```
assume (x=y+2*k)&(x>=y);
{x=y+2k,x>=y}
assume (x < y);
empty(6)
assume (x0=x)&(y0=y)&(k0=k);
k := k - 1;
x := x - 1;
y := y + 1
empty(6)
```

```
assume (x=y+2*k)&(x>=y);
{x=y+2k,1>=0,x>=y}
assume (x > y);
{x=y+2k,1>=0,x>=y+1}
assume (x0=x)&(y0=y)&(k0=k);
k := k - 1;
x := x - 1;
y := y + 1
{x+2=y+2k0,y=y0+1,x+1=x0,x=y+2k,x+1>=y}
```

□

3 Verification Condition Setup

3.1 Floyd/Naur/Hoare Invariance Proof Method

Given a loop precondition $P(x) \geq 0$ which holds before loop entry, the invariance proof method [16, 19, 25] consists in proving that the invariant $I(x) \geq 0$ is *initial*

[2] The technique was first used in the context of static analysis for context-sensitive interprocedural analysis to compute summaries of recursive procedures [9].

[3] `empty(6)` denotes the empty polyhedron in 6 variables, that is unreachability $\bot$.

(that is to say holds on loop entry) and *inductive* (i.e. remains true after each loop iteration):

$$\forall x \in \mathbb{R}^n : (P(x) \geq 0) \Rightarrow (I(x) \geq 0), \tag{2}$$

$$\forall x_0, x \in \mathbb{R}^n : (I(x_0) \geq 0 \wedge \bigwedge_{k=1}^{N} \sigma_k(x_0, x) \geq 0) \Rightarrow (I(x) \geq 0) \ . \tag{3}$$

3.2 Floyd Rank Function Termination Proof Method

Floyd's method [16] for proving loop termination consists in discovering a rank function $r \in \mathbb{R}^n \longrightarrow W$ of the values of the program variables into a well-founded set $\langle W, \preceq \rangle$ which is strictly decreasing at each iteration of the loop. If the nondeterminism is bounded, one can choose $\langle W, \preceq \rangle = \langle \mathbb{N}, \leq \rangle$. In what follows, we will often use real valued rank functions r which are nonnegative on loop body entry and strictly decrease at each iteration by a positive quantity bounded from below[4]. In such a case, and up to an isomorphism, the rank function r can be embedded into $\mathbb{N}$.

In general a loop terminates for some initial values of the variables only, satisfying some loop termination precondition $P(x) \geq 0$ so that the strict decrementation can be requested for states reachable from this initial condition only, as characterized by an invariant $I(x) \geq 0$ in the sense of [16, 19, 25].

Floyd's verification conditions [16] for proving loop termination become:

$\exists r \in \mathbb{R}^n \longrightarrow \mathbb{R}$, $\exists \delta \in \mathbb{R}$:

$$\forall x_0 \in \mathbb{R}^n : (I(x_0) \geq 0) \Rightarrow (r(x_0) \geq 0), \tag{4}$$

$$\forall x_0, x \in \mathbb{R}^n : (I(x_0) \geq 0 \wedge \bigwedge_{k=1}^{N} \sigma_k(x_0, x) \geq 0) \Rightarrow (r(x_0) - r(x) - \delta \geq 0), \tag{5}$$

$$\delta > 0 \ . \tag{6}$$

Remark 1. We can also choose $\delta = 1$ but it is sometimes more flexible to let its value be computed by the solver (see later Rem. 4). □

Remark 2. As proved in [11, Sec. 9, p. 290], the above choice of I and r not depending upon initial states is incomplete so that, more generally, we may have to use $I(\underline{x}, x')$ and $r(\underline{x}, x')$ where $\underline{x} \in \mathbb{R}^n$ denotes the initial value of the variables before loop entry and $x' \in \mathbb{R}^n$ their current value that is x_0 at the beginning of an iteration of the loop body and x at the end of that same iteration. □

4 Parametric Abstraction

Fixing the form of the unknown invariant $I(x)$ in (2) and (3) or of the rank function r in (4) and (5) in terms of p unknown parameters $a \in \mathbb{R}^p$ to be determined by the analysis is an abstraction [10]. An example is the affine abstraction [21].

[4] To avoid the Zeno phenomenon, that is, a strict decrease by 1, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, ..., which could be infinite.

More generally, a function $f(x)$ can be abstracted in the form $f_a(x)$ where a is a line vector of unknown parameters and x is the line vector of values of the loop variables[5]. For example, the linear case is $f_a(x) = a.x^\top$ and the affine case is $f_a(x) = a.(x\ 1)^\top$. A quadratic choice would be $f_a(x) = x.a.x^\top$ or $f_a(x) = (x\ 1).a.(x\ 1)^\top$ where a is a symmetric matrix of unknown parameters.

After parametric abstraction, it remains to compute the parameters a by solving the verification constraints. For example, the termination verification conditions (4), (5), and (6) become:

$$\exists a \in \mathbb{R}^p : \exists \delta \in \mathbb{R} :$$

$$\forall x_0 \in \mathbb{R}^n : (I(x_0) \geq 0) \Rightarrow (r_a(x_0) \geq 0), \tag{7}$$

$$\forall x_0, x \in \mathbb{R}^n : (I(x_0) \geq 0 \wedge \bigwedge_{k=1}^{N} \sigma_k(x_0, x) \geq 0) \Rightarrow (r_a(x_0) - r_a(x) - \delta \geq 0), \tag{8}$$

$$\delta > 0\ . \tag{9}$$

The resolution of these termination constraints in the case of linear programs and rank functions has been explored by [6], using a reduction of the constraints based on the construction of polars, intersection and projection of polyhedral cones (with limitations, such as that the loop test contains no disjunction and the body contains no test).

5 Solving the Abstract Verification Conditions by First-Order Quantifier Elimination

The Tarski-Seidenberg decision procedure for the first-order theory of real closed fields by quantifier elimination can be used to solve (7), (8), and (9) since it transforms a formula $Q_1 x1 : \ldots Q_n xn : F(x_1, \ldots, x_n)$ (where the Q_i are first-order quantifiers $\forall$, $\exists$ and F is a logical combination of polynomial equations and inequalities in the variables $x_1, \ldots, x_n$) into an equivalent quantifier free formula. However Tarski's method cannot be bound by any tower of exponentials. The cylindrical algebraic decomposition method by Collins [5] has a worst-case time-complexity for real quantifier elimination which is "only" doubly exponential in the number of quantifier blocks. It is implemented in MATHEMATICA® but cannot be expected to scale up to large problems. So we rely on another abstraction method described below.

[5] The sets of constraints $f_a(x) \geq 0$ for all a may not be a Moore family for the pointwise ordering, in which case a concretization function γ may be used [13]. For simplicity we make no distinction between the representation of the constraint $f_a(x) \geq 0$ and its value $\gamma(f_a(x) \geq 0) = \{x \in \mathbb{R}^n \mid f_a(x) \geq 0\}$. In consequence, the use of a concretization function will remain implicit.

6 Abstraction of the Verification Conditions into Numerical Constraints by Lagrangian Relaxation

6.1 The Lagrangian Relaxation Method

Let $\mathbb{V}$ be a finite dimensional linear vector space, $N > 0$ and $\forall k \in [0, N] : \sigma_k \in \mathbb{V} \longrightarrow \mathbb{R}$ (not necessarily linear). Let $\mathbb{R}^+ = \{x \geq 0 \mid x \in \mathbb{R}\}$. To prove:

$$\forall x \in \mathbb{V} : \left(\bigwedge_{k=1}^{N} \sigma_k(x) \geq 0 \right) \Rightarrow (\sigma_0(x) \geq 0), \tag{10}$$

the *Lagrangian relaxation* consists in proving that:

$$\exists \lambda \in [1, N] \longrightarrow \mathbb{R}^+ : \forall x \in \mathbb{V} : \sigma_0(x) - \sum_{k=1}^{N} \lambda_k \sigma_k(x) \geq 0, \tag{11}$$

where the λ_k are called *Lagrange coefficients*. The interest of Lagrangian relaxation is that the implication $\Rightarrow$ and conjunction $\bigwedge$ in (10) are eliminated in (11).

The approach is obviously sound, since the hypothesis $\bigwedge_{k=1}^{N} \sigma_k(x) \geq 0$ in (10) and the positivity of the Lagrange coefficients $\lambda \in [1, N] \longrightarrow \mathbb{R}^+$ implies the positivity of $\sum_{k=1}^{N} \lambda_k \sigma_k(x)$. Hence, by the antecedent of (10) and transitivity, $\sigma_0(x) \geq 0$ holds in the consequent of (10).

Observe that equality constraints can be handled in the same way by requesting the corresponding Lagrange coefficients to be reals (as opposed to nonnegative reals for inequality constraints). Indeed for equality constraints $\sigma_k(x) = 0$, we can use (11) for both $\sigma_k(x) \geq 0$ and $-\sigma_k(x) \geq 0$ with respective coefficients $\lambda_k \geq 0$ and $\lambda'_k \geq 0$ so that the terms $\lambda_k \sigma_k(x) + \lambda'_k(-\sigma_k(x))$ can be grouped into a single term $(\lambda_k - \lambda'_k)\sigma_k(x)$ with no sign restriction on $\lambda_k - \lambda'_k$. Since any real is equal to the difference of some nonnegative reals, we can equivalently use a single term $\lambda''_k \sigma_k(x)$ with $\lambda''_k \in \mathbb{R}$.

Lagrangian relaxation is in general incomplete (that is (10) $\not\Rightarrow$ (11), also called *lossy*). However it is complete (also called *lossless*)) in the linear case (by the affine Farkas' lemma) and the linear case with at most two quadratic constraints (by Yakubovitch's S-procedure [33, Th. 1]).

6.2 Lagrangian Relaxation of Floyd Termination Verification Conditions on Rank Functions

Relaxing Floyd's parametric verification conditions (7), (8), and (9), we get:

$$\exists a \in \mathbb{R}^p : \exists \delta \in \mathbb{R} : \exists \mu \in \mathbb{R}^+ : \exists \lambda \in [0, N] \longrightarrow \mathbb{R}^+ :$$

$$\forall x_0 \in \mathbb{R}^n : r_a(x_0) - \mu . I(x_0) \geq 0, \tag{12}$$

$$\forall x_0, x \in \mathbb{R}^n : r_a(x_0) - r_a(x) - \delta - \lambda_0 . I(x_0) - \sum_{k=1}^{N} \lambda_k . \sigma_k(x_0, x) \geq 0, \tag{13}$$

$$\delta > 0 \ . \tag{14}$$

In [29], the constraints $\sigma_k(x, x') \geq 0$ are assumed to be linear in which case the Lagrange coefficients can be eliminated by hand. Then the problem reduces to linear programming (with limitations, such as that the loop test contains no disjunction, the loop body contains no tests and the method cannot identify the cases when the loop does not terminate). We can use semidefinite programming to overcome the linearity limitation.

7 Semidefinite Programming

The *semidefinite programming optimization problem* is to find a solution to the constraints:

$$\begin{cases} \exists x \in \mathbb{R}^m : M(x) \succcurlyeq 0 \\ \text{Minimizing } c^\top x \end{cases}$$

where $c \in \mathbb{R}^m$ is a given real vector, the *linear matrix inequality* (LMI) [2] $M(x) \succcurlyeq 0$ is of the form:

$$M(x) = M_0 + \sum_{k=1}^{m} x_k . M_k$$

with symmetric matrices ($M_k = {M_k}^\top$), and *positive semidefiniteness* is defined as:

$$M(x) \succcurlyeq 0 \quad \stackrel{\Delta}{=} \quad \forall X \in \mathbb{R}^N : X M(x) X^\top \geq 0 \ .$$

The *semidefinite programming feasibility problem* consists in finding a solution to the constraints $M(x) \succcurlyeq 0$. A feasibility problem can be converted into the optimization program $\min\{-y \in \mathbb{R} \mid \bigwedge_{i=1}^{N} M_i(x) - y \succcurlyeq 0\}$.

8 LMI Constraint Setup for Termination

For programs which invariant and operational semantics (1) can be expressed in the form:

$$I(x_0) \wedge [\![B;C]\!](x_0, x) = \bigwedge_{k=1}^{N} (x_0\ x\ 1) M_k (x_0\ x\ 1)^\top \geq 0, \tag{15}$$

the constraints (12), (13), and (14) become LMIs (in the unknown a, μ, δ and the λ_k, $k = 1, \ldots, N$ by parametric abstraction (Sec. 4) of r_a in the form $r_a(x) = (x\ 1) R (x\ 1)^\top$ where R is a real $(n+1) \times (n+1)$-symmetric matrix of unknown parameters).

The conjunction of LMIs $M_1(x) \succcurlyeq 0 \wedge \ldots \wedge M_k(x) \succcurlyeq 0$ can be expressed as a single LMI $\mathrm{diag}(M_1(x), \ldots, M_k(x)) \succcurlyeq 0$ where $\mathrm{diag}(M_1(x), \ldots, M_k(x))$ denotes the block-diagonal matrix with $M_1(x), \ldots, M_k(x)$ on its diagonal.

These LMIs can then be solved by semidefinite programming.

Example 5. To show this, we prove the linear termination of the linear example program below, considered as in general semidefinite form (so that the generalization to (15) is immediate). The semantics of the loop body can be determined by a forward symbolic analysis of the loop body assuming the loop invariant (here the loop condition) and by naming the values of the variables at the beginning of the loop body[6]:

```
while (x >= 1) & (y >= 1) do
  x := x - y
od
```

Program

```
assume (x0 > 0) & (y0 > 0);
{y0>=1,x0>=1}
assume (x = x0) & (y = y0);
{y0=y,x0=x,y0>=1,x0>=1}
x := x - y
{y0=y,x0=x+y,y0>=1,x0>=1}
```

Semantics of the loop body.

The constraints $\sigma_k(x_0, x)$ are encoded as $(x_0\ x\ 1)M_k(x_0\ x\ 1)^\top$. For the above example, we have:

```
Mk(:,:,1) =
      0   0   0   0  1/2
      0   0   0   0   0
      0   0   0   0   0
      0   0   0   0   0
     1/2  0   0   0  -1

Mk(:,:,2) =
      0   0   0   0   0
      0   0   0   0  1/2
      0   0   0   0   0
      0   0   0   0   0
      0  1/2  0   0  -1

Mk(:,:,3) =
      0   0   0   0   1/2
      0   0   0   0    0
      0   0   0   0  -1/2
      0   0   0   0  -1/2
     1/2  0- 1/2 -1/2  0

Mk(:,:,4) =
      0   0   0   0   0
      0   0   0   0 -1/2
      0   0   0   0   0
      0   0   0   0  1/2
      0 -1/2  0  1/2  0
```

that is in symbolic form:

$$\begin{aligned}
x_0 - 1 \geq 0 \qquad & (x_0\ y_0\ x\ y\ 1)\mathtt{Mk}(:,:,1)(x_0\ y_0\ x\ y\ 1)^\top \geq 0\\
y_0 - 1 \geq 0 \qquad & (x_0\ y_0\ x\ y\ 1)\mathtt{Mk}(:,:,2)(x_0\ y_0\ x\ y\ 1)^\top \geq 0\\
x_0 - x - y = 0 \qquad & (x_0\ y_0\ x\ y\ 1)\mathtt{Mk}(:,:,3)(x_0\ y_0\ x\ y\ 1)^\top = 0\\
-y_0 + y = 0 \qquad & (x_0\ y_0\ x\ y\ 1)\mathtt{Mk}(:,:,4)(x_0\ y_0\ x\ y\ 1)^\top = 0\ .
\end{aligned}$$

The termination constraints (12), (13), and (14) now become the following LMIs[7]:

[6] As considered in Sec. 2.4, which is different from Rem. 2 where the values of variables were remembered before loop entry.

[7] Notice that if $(x\,1)A(x\,1)^\top \geq 0$ for all x, this is the same as $(y\,t)A(y\,t)^\top \geq 0$ for all y and all $t \neq 0$ (multiply the original inequality by t^2 and call $xt = y$). Since the latter inequality holds true for all x and all $t \neq 0$, by continuity it holds true for all x, t, that is, the original inequality is equivalent to positive semidefiniteness of A (thanks Arkadi Nemirovski for this argument).

```
M0-l0(1,1)*Mk(:,:,1)-
                l0(2,1)*Mk(:,:,2)-l0(3,1)*Mk(:,:,3)-l0(4,1)*Mk(:,:,4)>=0
M0-M_0-delta-
  l(1,1)*Mk(:,:,1)-l(2,1)*Mk(:,:,2)-l(3,1)*Mk(:,:,3)-l(4,1)*Mk(:,:,4)>=0
```

where `>=` is semidefinite positiveness $\succcurlyeq$ in LMI constraints, the `l0(`i`,`j`)` and `l(`i`,`j`)` are the Lagrange coefficients which are requested to be nonnegative for inequality constraints:

```
l0(1,1)>=0    l0(2,1)>=0    l(1,1)>=0    l(2,1)>=0
```

(where `>=` is the real comparison for elementwise constraints), the rank function $r(x) = (x\ 1).R.(x\ 1)^\top$ appears in `M0` $= \begin{bmatrix} R_{1:n,1:n} & 0^{n\times n} & R_{1:n,n+1} \\ 0^{n\times n} & 0^{n\times n} & 0^{n\times 1} \\ R_{n+1,1:n} & 0^{1\times n} & R_{n+1,n+1} \end{bmatrix}$ such that $\forall x : r(x_0) = (x_0\ x\ 1).$`M0`$.(x_0\ x\ 1)^\top$ and in `M_0` $= \begin{bmatrix} 0^{n\times n} & 0^{n\times n+1} \\ 0^{n+1\times n} & R \end{bmatrix}$ such that $\forall x_0 : r(x) = (x_0\ x\ 1).$`M_0`$.(x_0\ x\ 1)^\top$ and `delta` $= \begin{bmatrix} 0^{2n\times 2n} & 0^{2n\times 1} \\ 0^{1\times 2n} & \delta \end{bmatrix}$ so that $\forall x_0, x :$ $(x_0\ x\ 1).$`delta`$.(x_0\ x\ 1)^\top = \delta$. □

Remark 3. An affine (linear by abuse of language) rank function $r_a(x) = a.(x\ 1)^\top$ where $x \in \mathbb{R}^n$ and $a \in \mathbb{R}^n$ can be enforced by choosing $R = \left[\begin{array}{c|c} 0^{n,n} & (\frac{a}{2})_{1:n}^\top \\ \hline (\frac{a}{2})_{1:n} & a_{n:n} \end{array}\right]$. □

9 Solving the Termination LMI Constraints

Following the extension of the interior point method for linear programming to convex cones [27], numerous solvers have been developed for semidefinite programming such as `bnb`[8] [23], `DSDP4` [1], `lmilab` [17], `PenBMI`[9] [22], `Sdplr` [4], `Sdpt3` [32], `SeDuMi` [31], with common interfaces under MATLAB® such as `Yalmip` [23].

Example 6 (Ex. 5 continued). Choosing $\delta = 1$ and a linear rank function as in Rem. 3, we can solve the LMI constraints of Ex. 5 using various solvers under `Yalmip`:

```
r(x,y) = +4.x +2.y -3                                          bnb
r(x,y) = +5.268942e+02.x +4.956309e+02.y -5.270981e+02  CSDP-4.9
r(x,y) = +2.040148e+07.x +2.222757e+07.y +9.096450e+06  DSDP4-4.7
r(x,y) = +2.767658e+11.x +2.265404e+11.y -1.311440e+11  lmilab
r(x,y) = +4.031146e+03.x +3.903684e+03.y +1.401577e+03  lmilab[10]
r(x,y) = +1.042725e+00.x +4.890035e-01.y +1.975391e-01  Sdplr-1.01
r(x,y) = +9.888097e+01.x +1.343247e+02.y -1.725408e+02  Sdpt3-3.02
r(x,y) = +1.291131e+00.x +4.498515e-01.y -1.316373e+00  SeDuMi-1.05
```

[8] For integer semidefinite programming.
[9] To solve bilinear matrix inequalities.

Since different solvers use different resolution strategies, each one may provide a different solution. Moreover, since there are infinitely many different rank functions (e.g. just multiply by or add a positive constant), the solution may not be the one a human being would naturally think of. Indeed, in the above example, any $r(x,y) = ax + by + c$ with $a \geq 1$, $b \geq 0$ and $a + b + c \geq 0$ will do. □

Remark 4. It is also possible to let δ be an unknown parameter with the constraint $\delta > 0$ as in (14). In this case, looking for a linear rank function with `bnb`, we get `r(x,y) = +2.x -2` and $\delta =$ `8.193079e-01`. □

Remark 5. It is possible to check a rank function by fixing R as well as δ and then by checking for the feasibility of the constraints (12), (13), and (14), which returns the Lagrange coefficients. For example to check `r(x,y) = +1.x`, we use R = `[[0,0,1/2]; [0,0,0]; [1/2,0,0]]` and $\delta = 1$ while performing the feasibility check with `bnb`. □

10 Examples

The examples illustrate different kind of ranking functions.

10.1 Examples of Linear Termination of a Linear Loop

Example 7. Choosing $\delta = 1$ and a linear rank function for the naïve Euclidean division:

```
assume (y >= 1);
q := 0; r := x;
while (y <= r) do
   r := r - y;
   q := q + 1
od
```

$$\begin{aligned} y - 1 &\geq 0 \\ q - 1 &\geq 0 \\ r &\geq 0 \end{aligned} \qquad \begin{aligned} -q_0 + q - 1 &= 0 \\ -x_0 + x &= 0 \\ -y_0 + y &= 0 \\ -r_0 + y + r &= 0 \end{aligned}$$

The linear semantics of the loop body (with polyhedral invariant) is provided on the right. Solving the corresponding termination constraints with `bnb`, we automatically get the ranking function `r'(x,y,q,r) = -2.y +2.q +6.r`, which is certainly less intuitive than Floyd's proposal $r'(x,y,q,r) = x - q$ [16] but has the advantage not to depend upon the nonlinear loop invariant $x = r + qy$. □

Example 8 (Ex. 4 continued). For the example Ex. 4 from [3] considered in Sec. 2.2, where the difference `<>` was handled as a disjunction and one case was shown to be impossible, we get:

```
assume (x=y+2*k) & (x>=y);
while (x <> y) do
    k := k - 1;
    x := x - 1;
    y := y + 1
od
```

$$\begin{aligned} x - y + 1 &\geq 0 \\ -2k_0 + x - y + 2 &= 0 \\ -y_0 + y - 1 &= 0 \\ -x_0 + x + 1 &= 0 \\ x - y - 2k &= 0 \end{aligned}$$

[10] With a *feasibility radius* of $\rho =$ `1.0e4`, constraining the solution x to lie in the ball $x^\top x < \rho^2$ where $\rho > 0$.

With `bnb`, the proposed rank function is `r(x,y,k) = +4.k`, proving that the necessary termination precondition automatically determined by the auxiliary termination counter method of Sec. 2.2 is also sufficient. □

10.2 Example of Quadratic Termination of a Linear Loop

Example 9. Let us consider the program below which oddly simulates `for i = n downto 1 do for j = n downto 1 do skip end end`. The termination precondition has been automatically determined by iterated forward/backward polyhedral analysis. The loop invariant has been automatically determined by a forward polyhedral analysis, assuming the termination precondition. The analysis of the loop body involves a partitioning according to the test `(j > 0)`, as later explained in Sec. 11.1. For each case, the polyhedral approximation of the semantics of the loop body (where initially `(n0 = n) & (i0 = i) & (j0 = j)`) is given on the right:

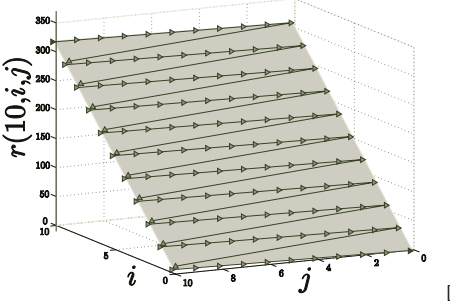
```
assume (n >= 0);
i := n; j := n;
while (i <> 0) do
  assume ((j>=0) & (i>=0) &
          (n>=i) & (n>=j));
  if (j > 0) then
    j := j - 1
  else
    j := n; i := i - 1
  fi
od
```

Case $(j_0 > 0)$:	Case $(j_0 \leq 0)$:
$n - i \geq 0$	$i \geq 0$
$i - 1 \geq 0$	$-i + j - 1 \geq 0$
$j \geq 0$	$-n_0 + j = 0$
$n - j - 1 \geq 0$	$j_0 = 0$
$-j_0 + j + 1 = 0$	$-i_0 + i + 1 = 0$
$-i_0 + i = 0$	$-n + j = 0$
$-n_0 + n = 0$	

Choosing $\delta = 1$ and a quadratic rank function, the resolution of the LMI constraints given in next Sec. 11.1 by `Sdplr-1.01` (with feasibility radius of `1.0e+3`) yield the solution:

```
r(n,i,j) = +7.024176e-04.n^2 +4.394909e-05.n.i -2.809222e-03.n.j...
           +1.533829e-02.n +1.569773e-03.i^2 +7.077127e-05.i.j  ...
           +3.093629e+01.i -7.021870e-04.j^2 +9.940151e-01.j    ...
           +4.237694e+00 .
```

Successive values of `r(n,i,j)` during program execution are plotted above for `n` = 10 on loop entry. They strictly decrease along the inclined plane.



Nested loops are better handled by induction on the nesting level, as shown in Sec. 11.2. □

10.3 Example of Linear Termination of a Quadratic Loop

Example 10. The following program computes the least factorial strictly greater than a given integer `N`:

```
n := 0; f := 1;
while (f <= N) do
   n := n + 1; f := n * f
od
```

$$-f_0 + N_0 \geq 0 \qquad -n_0 + n - 1 = 0$$
$$n_0 \geq 0 \qquad -f_0.n + f = 0$$
$$f_0 - 1 \geq 0 \qquad -N_0 + N = 0$$

The non-linear semantics of the loop body (with polyhedral invariant) is provided on the right. It has only one quadratic constraint, a case when the Lagrangian relaxation is complete. The ranking function found by `SeDuMi-1.05` (with feasibility radius of `1.0e+3`) is `r(n,f,N) = -9.993455e-01.n +4.346533e-04.f +2.689218e+02.N +8.744670e+02`. □

11 Extension to More Complex Language Features

11.1 Disjunctions in the Loop Test and Conditionals in the Loop Body

Disjunctions in the loop test and/or conditionals within the loop body can be analyzed by partitioning along the values of the boolean expressions [10, Sec. 10.2]. Equivalently, a case analysis of the boolean expressions yields an operational semantics of the loop body of the form:

$$[\![B;C]\!](x,x') = \bigvee_{j=1}^{M} \bigwedge_{k=1}^{N_j} \sigma_{jk}(x,x') \geq 0 \ . \tag{16}$$

Whichever alternative is chosen, the rank function must strictly decrease while remaining nonnegative. Hence, we just have to consider the conjunction of all terminating constraints for each of the possible alternatives. We have already seen Ex. 9. Here is another one.

Example 11. For the program below:

```
while (x < y) do
  if (i >= 0) then
     x := x+i+1
  else
     y := y+i
  fi
od
```

Case $(x_0 < y_0)$:

$$-x_0 + y_0 - 1 \geq 0$$
$$i_0 \geq 0$$
$$-i_0 - x_0 + x - 1 = 0$$
$$-y_0 + y = 0$$
$$-i_0 + i = 0$$

Case $(x_0 \geq y_0)$:

$$-x_0 + y_0 - 1 \geq 0$$
$$-i_0 - 1 \geq 0$$
$$-i_0 - y_0 + y = 0$$
$$-x_0 + x = 0$$
$$-i_0 + i = 0$$

the cases are listed on the right[11]. The termination constraints are given below (the `P(`j`).Mk(:,:,`k`)` corresponding to the k-th constraint in the j-th case, the corresponding Lagrange coefficients being `l0(`j`).v(`k`,`j`)` for the nonnegativity and `l(`j`).v(`k`,`j`)` for decrementation by at least $\delta = 1$. The matrices `M0` and `M_0` encapsulate the matrix `R` of the ranking function $r(x) = (x\ 1).\mathtt{R}.(x\ 1)^\top$ while $(x\ 1).\mathtt{delta}.(x\ 1) = \delta$ as explained in Sec. 8):

[11] Since the alternatives are considered on loop entry, a backward analysis may in general have to be used if some variable involved in a test of the loop body is modified in the body before that test.

```
M0-l0(1).v(1,1)*P(1).Mk(:,:,1)-l0(1).v(2,1)*P(1).Mk(:,:,2)-...
   l0(1).v(3,1)*P(1).Mk(:,:,3)-l0(1).v(4,1)*P(1).Mk(:,:,4)-...
   l0(1).v(5,1)*P(1).Mk(:,:,5) >= 0
M0-M_0-delta-l(1).v(1,1)*P(1).Mk(:,:,1)-l(1).v(2,1)*P(1).Mk(:,:,2)-...
   l(1).v(3,1)*P(1).Mk(:,:,3)-l(1).v(4,1)*P(1).Mk(:,:,4)-...
   l(1).v(5,1)*P(1).Mk(:,:,5) >= 0
l0(1).v(1,1) >= 0
l0(1).v(2,1) >= 0
l(1).v(1,1) >= 0
l(1).v(2,1) >= 0
M0-l0(2).v(1,1)*P(2).Mk(:,:,1)-l0(2).v(2,1)*P(2).Mk(:,:,2)-...
   l0(2).v(3,1)*P(2).Mk(:,:,3)-l0(2).v(4,1)*P(2).Mk(:,:,4)-...
   l0(2).v(5,1)*P(2).Mk(:,:,5) >= 0
M0-M_0-delta-l(2).v(1,1)*P(2).Mk(:,:,1)-l(2).v(2,1)*P(2).Mk(:,:,2)-...
   l(2).v(3,1)*P(2).Mk(:,:,3)-l(2).v(4,1)*P(2).Mk(:,:,4)-...
   l(2).v(5,1)*P(2).Mk(:,:,5) >= 0
l0(2).v(1,1) >= 0
l0(2).v(2,1) >= 0
l(2).v(1,1) >= 0
l(2).v(2,1) >= 0
```

Solving these LMI and elementwise constraints with `bnb`, we get `r(i,x,y) = -4.x +4.y`, that is essentially $\mathtt{y} - \mathtt{x}$, which corresponds to the intuition. □

11.2 Nested Loops

In the case of nested loops, the loops are handled one at a time, starting from the inner ones.

Example 12 (Manna's original bubble sort). For the bubble sort example below (taken literally from [24, p. 191]), the necessary termination precondition $\mathtt{N} \geq 0$ is automatically determined by the iterated forward/backward method of Sec. 2.2. A further automatic forward reachability analysis starting from this termination precondition yields loop invariants:

```
assume (N >= 0);
n := N;
i := n;
loop invariant: {N=n,i>=0,n>=i}
while (i <> 0 ) do
  j := 0;
  loop invariant: {N=n,j>=0,i>=j,i>=1,N>=i}
  while (j <> i) do
    j := j + 1
  od;
  i := i - 1
od
```

The result of this global analysis is used to determine the semantics of the inner loop body as given by its forward analysis:

```
assume ((N=n) & (j>=0) & (i>=j) & (i>=1) & (N>=i));
assume (j <> i);
assume ((N0=N) & (n0=n) & (i0=i) & (j0=j));
j := j + 1
{j=j0+1,i=i0,N=n0,N=N0,N=n,j>=1,N>=i,j<=i}
```

The termination of the inner loop is then proved by solving the corresponding termination constraints as shown in Sec. 8. The `bnb` solver yields the rank function `r(N,n,i,j) = +2.n +4.i -4.j -4`.

Next, the semantics of the outer loop body is given by its forward polyhedral analysis:

```
assume ((N=n) & (i>=0) & (n>=i));
assume (i <> 0 );
assume ((N0=N) & (n0=n) & (i0=i) & (j0=j));
j := 0;
while (j <> i) do
      j := j + 1
od;
i := i - 1
{i+1=j,i+1=i0,N=n0,N=N0,N=n,N>=i+1,i>=0}
```

The termination of the outer loop is then proved by solving the corresponding termination constraints as shown in Sec. 8. With `bnb`, we get the rank function `r(N,n,i,j) = +2.n +4.i -3`. □

In case the program graph is irreducible, the program has to be considered as a whole, with different ranking functions attached to cutpoints (the choice of which may not be unique).

11.3 Nondeterminism and Concurrency

Nondeterministic semantics are similar to (16) in Sec. 11.1. Nondeterminism can be used to handle concurrency by nondeterministic interleaving.

Example 13. The following concurrent program (where atomic actions are square bracketed) does terminate without any fairness hypothesis. If one process is never activated then the other process will terminate and so the remaining one will then be activated.

```
[|                                  while (x+2 < y) do
   while [x+2 < y] do                  if (??) then
      [x := x + 1]                       x := x + 1
   od                                  else if (??) then
||                                       y := y - 1
   while [x+2 < y] do                  else
      [y := y - 1]                       x := x + 1;
   od                                    y := y - 1
|]                                     fi fi
                                    od
```

By nondeterministic interleaving, the program is equivalent to the nondeterministic one on its right. The conditionals in the loop body can be handled as explained in Sec. 11.1. An even simpler solution is to consider an abstract interpretation of the semantics of the loop body through a polyhedral approximation (the resulting constraints are given on the right):

```
assume (x+2 < y);
assume ((x0 = x) & (y0 = y));
if (??) then
  x := x + 1
else if (??) then
  y := y - 1
else
  x := x + 1;
  y := y - 1
fi fi
{y+1>=y0,x<=x0+1,x+y0>=y+x0+1,x0+3<=y0}
```

$$-y_0 + y + 1 \geq 0$$
$$x_0 - x + 1 \geq 0$$
$$-x_0 + y_0 + x - y - 1 \geq 0$$
$$-x_0 + y_0 - 3 \geq 0$$

Establishing the termination constraints as explained in Sec. 8, and solving with `bnb`, we get the following termination function `r(x,y) = -4.x +4.y -9`. □

11.4 Bounded Weakly Fair Parallelism

One way of handling fair parallelism is to consider nondeterministic interleaving with a scheduler to ensure bounded weak fairness.

Example 14. The following weakly fair parallel program (where atomic actions are bracketed):

```
[[ while [(x > 0) | (y > 0) do       || while [(x > 0) | (y > 0) do
      x := x - 1]                          y := y - 1]
   od                                   od ]]
```

does not terminate when `x` and `y` are initially positive and any one of the processes is never activated. Because of the bounded fairness hypothesis, the parallel program is semantically equivalent to the following nondeterministic program:

```
assume (m >= 1);
t := ?;
assume (0 <= t & t <= 1);
s := ?;
assume ((1 <= s) & (s <= m));
while ((x > 0) | (y > 0)) do
  if (t = 1) then
     x := x - 1
  else
     y := y - 1
  fi;
  if (s = 1) then
    if (t = 1) then
       t := 0
    else
       t := 1
    fi;
    s := ?;
    assume ((1 <= s) & (s <= m))
  else
     s := s - 1
  fi
od
```

The nondeterministic program incorporates an explicit scheduler of the two parallel processes where the turn `t` indicates which process is running and `s` indicates

the number of atomic steps remaining to run before activating another process. The nondeterminism is bounded by `m` which ensures the existence of an integer-valued rank function. Notice however that, as found in practice, although the nondeterminism is known to be bounded, it is not known of how much (`m` can take any positive integer value, including very large ones). The theoretical notion of weak fairness corresponds to the case when $\mathtt{m} \to \infty$ that is unbounded nondeterminism, which may require ordinal-valued rank functions. In practice one can use lexicographic orderings on $\mathbb{N}$.

A forward analysis of the program determines the loop invariant {`t<=1,s<=m, s>=1,t>=0`}. The disjunction in the loop test is handled by partitioning, see Sec. 11.1. There are two cases for the loop test `(x > 0)`, or `(y > 0)`. In each case, the loop body is partitioned according to the value of `s` which, according to the invariant determined by the forward polyhedral analysis is either `(s = 1)` or `(s > 1)`. The case $(\mathtt{x} > 0 \wedge \mathtt{s} > 1)$ is illustrated below (`empty(10)` stands for $\bot$, that is unreachability):

```
assume (t <= 1) & (s <= m) & (s >= 1) & (t >= 0);
assume (x > 0);
assume (s = 1);
assume ((x0 =x) & (y0 = y) & (t0 = t) & (s0 = s) & (m0 = m));
if (t = 1) then x := x - 1 else y := y - 1 fi;
if (s = 1) then
  if (t = 1) then
     t := 0
  else
     t := 1
  fi;
  s := ?;
  assume ((1 <= s) & (s <= m))
  {empty(10)}
else
  s := s - 1
fi
{m=m0,s+1=s0,t=t0,t+y0=y+1,t+x=x0,s+1<=m,t<=1,t>=0,t+x>=1,s>=1}
```

The other three forward analyses are similar and yield the following affine operational semantics for each of the alternatives:

$\mathtt{x} > 0 \wedge \mathtt{s} > 1$ {`m=m0,s+1=s0,t=t0,t+y0=y+1,t+x=x0,s+1<=m,t<=1,t>=0,t<=y,s>=1`}
$\mathtt{x} > 0 \wedge \mathtt{s} = 1$ {`m=m0,s+1=s0,t=t0,t+y0=y+1,t+x=x0,s+1<=m,t<=1,t>=0,t<=y,s>=1`}
$\mathtt{y} > 0 \wedge \mathtt{s} > 1$ {`m=m0,s+1=s0,t=t0,t+y0=y+1,t+x=x0,s+1<=m,t<=1,t>=0,t<=y,s>=1`}
$\mathtt{y} > 0 \wedge \mathtt{s} = 1$ {`m=m0,s0=1,t+t0=1,t+y=y0,t+x0=x+1,t<=1,s<=m,s>=1,t>=0,t+y>=1`}

The LMI termination constraints can then be established, as explained in Sec. 11.1. Solving with `SeDuMi-1.05` (with a feasibility radius of `1.0e+4`), we get the quadratic rank function:

```
r(x,y,m,s,t) = +8.078228e-06.x^2 +8.889797e-10.x.y +2.061102e-10.x.m
   +2.360326e-11.x.s +2.763786e-09.x.t +9.998548e-01.x +9.770849e-07.y^2
   +7.219411e-07.y.m -1.091400e-07.y.s -2.098975e-06.y.t +6.158628e+02.y
```

```
+4.044804e-06.m^2 -2.266154e-08.m.s +1.794800e-06.m.t +4.524134e-04.m
+7.994478e-06.s^2 -1.899723e-08.s.t -3.197335e-05.s +2.450149e-06.t^2
+3.556544e-04.t +9.696939e+03 .
```

□

11.5 Semi-algebraic/Polynomial Programs

The termination constraints (12), (13), and (14) for semi-algebraic/polynomial programs lead to polynomial inequalities. A necessary condition for $\forall x : p(x) \geq 0$ is that the degree $m = 2d$ of p be even. A sufficient condition for nonnegativity of $p(x)$ is that $p(x) \geq q(x)$ where $q(x)$ is a sum of squares (SOS) of other polynomials $q(x) = \sum_i r_i^2(x)$ for some $r_i(x) \in \mathtt{R}[x]$ of degree d [26]. However the condition is not necessary.

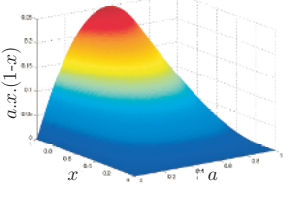
The Gram matrix method [28] consists in fixing a priori the form of the base polynomials $r_i(x)$ in the sum of squares and in assuming that $q(x) = z(x)^\top Q z(x)$ where $z(x)$ is the vector of $N = \binom{n+d}{d}$ monomials of the monomial basis $B_{d,n}$ in any total monomial order (for example $z(x) == [1, x_1, \ldots, x_n, x_1^2, x_1x_2, \ldots, x_n^d]$) and Q is a symmetric positive definite matrix of reals. Since $Q \succcurlyeq 0$, Q has a Cholesky decomposition L which is an upper triangular matrix L such that $Q{=}L^\top L$. It follows that $q(x) = z(x)^\top Q z(x) = z(x)^\top L^\top L z(x) = (Lz(x))^\top Lz(x) = [L_{i,:} \cdot z(x)]^\top [L_{i,:} \cdot z(x)] = \sum_i (L_{i,:} \cdot z(x))^2$ (where $\cdot$ is the vector dot product $x \cdot y = \sum_i x_i y_i$), proving that $q(x)$ is a sum of squares.

Finally, $z(x)^\top z(x)$ contains all monomials in x appearing in $p(x)$ and so $\forall x : p(x) - q(x) \geq 0$ can be expressed in the form $\forall x : z(x)^\top M z(x) \geq 0$ where M is a square symmetric matrix depending upon the coefficients of $p(x)$ and the unknowns in Q. By letting X be $z(x)$, the problem can be relaxed into the feasibility of $\forall X : X^\top M X$ which can be expressed as a semidefinite problem. If the problem is feasible, then the solution provides the value of Q whence a proof that $p(x)$ is positive.

The method is implemented by SOSTOOL [30] under MATLAB®.

Example 15 (Logistic map). The deterministic logistic map $f(x) = ax(1-x)$ with bifurcation parameter a such that $0 \leq a < 1$ has a sink at 0 and every initial condition between 0 and 1 is attracted to this sink. So the following program (where $z > 0$ is implemented as $z \geq \epsilon$ with a small ϵ) terminates.

```
eps = 1.0e-10;
while (0<=a) & (a<=1-eps)
   & (eps<=x) & (x<=1) do
     x := a*x*(1-x)
od
```



The MATLAB® program below establishes the termination conditions with Lagrangian relaxation (12), (13), and (14):

```
pvar a x0 x1 c0 d0 e0 l1 l2 l3 l4 l5 m1 m2 m3 m4 m5;
eps=1.0e-10;
iv = [a;x0;x1];
uv = [c0;d0;l1;l2;l3;l4;l5;m1;m2; m3;m4;m5];
pb = sosprogram(iv,uv);
pb = sosineq(pb,l1); pb = sosineq(pb,l2);
pb = sosineq(pb,l3); pb = sosineq(pb,l4);
pb = sosineq(pb,c0*x0+d0-l1*a-l2*(1-eps-a)-l3*(x0-eps)-l4*(1-x0)...
               -l5*(x1-a*x0*(1-x0)));
pb = sosineq(pb,m1); pb = sosineq(pb,m2);
pb = sosineq(pb,m3); pb = sosineq(pb,m4);
pb = sosineq(pb,c0*x0-c0*x1-eps^2-m1*a-m2*(1-eps-a)-m3*(x0-eps)...
               -m4*(1-x0)-m5*(x1-a*x0*(1-x0)));
spb = sossolve(pb);
c = sosgetsol(spb,c0); d = sosgetsol(spb,d0);
disp(sprintf('r(x) = %i.x + %i', double(c),double(d)));
```

These polynomial constraints are relaxed by SOSTOOLS v2.00 into a semidefinite program which is then solved by `SeDuMi-1.05`. The result is:

`r(x) = 1.222356e-13.x + 1.406392e+00` . □

12 Invariance

In the same way, loop invariants can be generated automatically by parametric abstraction (Sec. 4) and resolution of the Lagrangian relaxation (Sec. 6.1) of Floyd's invariance verification conditions (2) and (3). We get:

$$\exists a \in \mathbb{R}^p : \exists \mu \in \mathbb{R}^+ : \exists \lambda \in [0, N] \longrightarrow \mathbb{R}^+ :$$

$$\forall x \in \mathbb{R}^n : I_a(x) - \mu.P(x) \geq 0, \tag{17}$$

$$\forall x_0, x \in \mathbb{R}^n : I_a(x) - \lambda_0.I_a(x_0) - \sum_{k=1}^{N} \lambda_k.\sigma_k(x_0, x) \geq 0 \ . \tag{18}$$

There is an additional difficulty however since the appearance of the parametric abstraction of the invariant on the left of the implication in (3) yields, by Lagrangian relaxation, to the term $\lambda_0.I_a(x_0)$ in (18), which is bilinear in λ_0 and a. For programs which operational semantics has the form $[\![B;C]\!](x_0, x) = \bigwedge_{k=1}^{N} (x_0\ x\ 1) M_k (x_0\ x\ 1)^\top \geq 0$, constraint (18) is a bilinear matrix inequality (BMI), which can be solved by BMI solvers, which first appeared only recently, such as `PenBMI` [22] and `bmibnb` [23]. Contrary to iterative methods (at least when the ascending chain condition is satisfied), the invariant need not be the strongest one.

Example 16. This is illustrated by the following example from [14] where the invariant is obtained by forward polyhedral analysis:

```
i := 2;              clear all; yalmip('clear');
j := 0;              [iv,v] = variables('i','j');
while (??) do        p = parameters('a','b','c', 'm', 'l');
  {j>=0,i>=2j+2}     F = set(a*2+c>=0);
  if (??) then       F = F + set(sos(a*(i+4)+b*j-m*(a*i+b*j+c)));
    i := i + 4       F = F + set(m>=0);
  else               F = F + set(sos(a*(i+2)+b*(j+1)+c-l*(a*i+b*j+c)));
    i := i + 2;      F = F + set(l>=0);
    j := j + 1       sol = solvesos(F,[],sdpsettings('solver','penbmi'),p);
  fi                 disp(sprintf('%+g*i %+g*j %+g >= 0',double(a),...
od;                                                 double(b),double(c)));
```

Solving the given constraints with `Yalmip` yields the solution:

$$\texttt{+2.14678e-12*i -3.12793e-10*j +0.486712 >= 0},$$

which is not the strongest possible one. However, satisfiability is easily checked by setting $\texttt{a} = 1$, $\texttt{b} = -2$ and $\texttt{c} = -2$. □

However one can imagine other methods to discover the parameters (e.g. random interpretation [18]). Then a proof of invariance can be given by semidefinite programming relaxation.

In program verification, the assertions to be proved yields additional constraints which can be useful in the resolution.

Example 17 (Ex. 7 continued). In the Euclidean division of Ex. 7 from [16], we have to prove the postcondition `(x=qy+r)&(r<y)`. For a parametric invariant `I=a*x+b*q*y+c*r`, the constraints are the following:

```
clear all; yalmip('clear');
[iv,v] = variables('x','y','q','r');
p = parameters('a','b','c','m1','m2','m3','m4','m5','m6','l0','l1',...
               'l2','l3','l4','l5','l6','n1','n2');
I0=a*x0+b*q0*y0+c*r0; I=a*x+b*q*y+c*r;
F = set(sos(I0-m1*(y0-1)-m2*q0-m3*(r0-x0)));
F = F + set(m1>=0);
F = F + set(sos(I-l0*I0-l1*(y0-1)-l2*(r0-y0)-l3*(r-r0+y0)-l4*(q-q0-1)...
                -l5*(x-x0)-l6*(y-y0)));
F = F + set(l0>=0) + set(l1>=0) + set(l2>=0);
P=x-q*y-r;
F = F + set(sos(P-n1*I-n2*(r-y+1)));
F = F + set(n2>=0);
[sol,m,B] = solvesos(F,m1,sdpsettings('solver','penbmi'),p)
disp(sprintf('%+g*x %+g*q*y %+g*r >= 0',double(a),double(b),double(c)));
```

Solving with `Yalmip`, we get:

$$\texttt{+2.11831e-05*x -2.11831e-05*q*y -2.11831e-05*r >= 0}\ .$$

Then, in the other direction (where `I0`, `I`, `P` are respectively replaced by `-I0`, `-I`, `-P`), we get the loop invariant:

$$\texttt{+0.000167275*x -0.000167275*q*y -0.000167275*r >= 0}\ .$$

By normalization of the coefficients and antisymmetry, the total correctness proof is finished. □

13 Potential Problems with Solvers

13.1 Constraint Resolution Failure

The resolution of the termination constraints will definitely fail when the program does not terminate. The same way, the invariance constraints may be infeasible. However, infeasibility of the constraints does not mean "non termination" or "non invariance" but simply failure. First the parametric abstraction of Sec. 4 may be too coarse (so that e.g. a quadratic or even polynomial invariant/rank function may have to be considered instead of a linear one). Second, the solver may have failed (e.g. due to numerical difficulties when handling equalities) but may succeed with some help (e.g. by adding a shift [23]).

13.2 Numerical Computations

LMI/BMI solvers perform numerical computations with rounding errors, shifts, etc. It follows that the parameters of the parametric abstraction are subject to numerical errors and so the logical rigor of the proof may be questionable. Obviously the use of integer solvers or the concordant conclusions of several different solvers will be conclusive, at least from an experimental point of view, anyway more rigorous than mere tests.

Obviously, the hard point is to discover a candidate for the rank function or invariant and it is much less difficult, when it is known, to re-check for satisfaction (e.g. by static analysis or a proof assistant).

14 Conclusion

The resolution of systems of fixpoint (in)equations involving linear, semidefinite, and even polynomial numerical constraints by parametric abstraction and Lagrangian relaxation appears promising thanks to the spectacular progress in semidefinite programming and LMI/BMI solvers this last decade.

The approach seems naturally useful for termination since one is essentially interested in the existence of a rank function, even if it looks "unnatural". This is true of all inevitability/liveness properties for which generalization presents no fundamental problem.

The situation looks different for invariance since unnatural solutions may look less acceptable in the context of static analysis. However, the situation is different for correctness proofs, where the nature of the invariants has no importance provided the proof can be done. A difficulty is nevertheless to establish the form of the parametric abstraction, since the most general form would be costly at higher degrees.

To conclude, beyond numerical programs, parametric abstraction remains to be explored in other non-numerical contexts, such as symbolic computations.

Acknowledgements

We thank É. Féron who introduced us to LMIs [15], D. Henrion for his suggestions in formulating quantified polynomial positiveness as LMIs in Sec. 11.5, J. Löfberg for his help and fixes for YALMIP [23], B. Blanchet, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival for their comments.

References

1. S. Benson and Y. Ye. DSDP4: A software package implementing the dual-scaling algorithm for semidefinite programming. Technical Report ANL/MCS-TM-255, Argonne National Laboratory, 2002.
2. S. Boyd, L. E. Ghaoui, É. Féron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.
3. J. Brauburger and J. Giesl. Approximating the domains of functional and imperative programs. *Sci. Comput. Programming*, 35(1):113–136, 1999.
4. S. Burer and R. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (series B)*, 95(2):329–357, 2003.
5. G. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12:299–328, 1991.
6. M. Colón and H. Sipma. Synthesis of linear ranking functions. *Proc. 7th Int. Conf. TACAS '2001*, LNCS 2031, pp. 67–81. Springer.
7. P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Univ. scient. et méd. de Grenoble, 1978.
8. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th POPL*, pp. 238–252, 1977. ACM Press.
9. P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews*, pp. 237–277. North-Holland, 1977.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th POPL*, pp. 269–282, 1979. ACM Press.
11. P. Cousot and R. Cousot. 'À la Floyd' induction principles for proving inevitability properties of programs. In *Algebraic Methods in Semantics*, ch. 8, pp. 277–312. Cambridge U. Press, 1985.
12. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs[12]. *J. Logic Programming*, 13(2–3):103–179, 1992.
13. P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, 1992.
14. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. *5th POPL*, pp. 84–97, 1978. ACM Press.
15. É. Féron. Abstraction mechanisms across the board: A short introduction. *Workshop on Robustness, Abstractions and Computations*. Philadelphia, 18 Mar. 2004.

[12] The editor of J. Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`.

16. R. Floyd. Assigning meaning to programs. *Proc. Symposium in Applied Mathematics*, vol. 19, pp. 19–32. AMS, 1967.
17. P. Gahinet, A. Nemirovski, A. Laub, and M. Chilali. LMI Control Toolbox for use with MATLAB®, user's guide. 1995.
18. S. Gulwani and G. Necula. Discovering affine equalities using random interpretation. *30th POPL*, pp. 74–84, 2003. ACM Press.
19. C. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580, 1969.
20. B. Jeannet. New Polka. `http://www.irisa.fr/prive/bjeannet/newpolka.html`.
21. M. Karr. Affine relationships among variables of a program. *Acta Informat.*, 6:133–151, 1976.
22. M. Kočvara and M. Stingl. PENBMI *User's Guide (Version 1.1)*, 2004.
23. J. Löfberg. YALMIP. `http://control.ee.ethz.ch/~joloef/yalmip.msql`.
24. Z. Manna. *Mathematical theory of computation*. McGraw Hill, 1974.
25. P. Naur. Proofs of algorithms by general snapshots. *BIT*, 6:310–316, 1966.
26. Y. Nesterov. Squared functional systems and optimization problems. In *High Performance Optimization*, pp. 405–440. Kluwer Acad. Pub., 2000.
27. Y. Nesterov and A. Nemirovskii. Polynomial barrier methods in convex programming. *Èkonom. i Mat. Metody*, 24(6):1084–1091, 1988.
28. P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003.
29. A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. *Proc. 5th Int. Conf. VMCAI 2004*, pp. 239–251. LNCS 2937, Springer.
30. S. Prajna, A. Papachristodoulou, P. Seiler, and P. Parrilo. SOSTOOLS*: Sum of squares optimization toolbox for* MATLAB, 2004.
31. J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.
32. K. Toh, M. Todd, and R. Tütüncü. SDPT3–a MATLAB software package for semidefinite programming. *Optimization Methods and Software*, 11:545–581, 1999.
33. V. Yakubovich. Nonconvex optimization problem: The infinite-horizon linear-quadratic control problem with quadratic constraints. *Systems Control Lett.*, 19:13–22, 1992.