Logico-Numerical Max-Strategy Iteration

Peter Schrammel, Pavle Subotic

Logico-Numerical Max-Strategy Iteration*

Peter Schrammel[1], and Pavle Subotic[2,3]

[1] Inria Grenoble – Rhône-Alpes, France
peter.schrammel@inria.fr
[2] Uppsala University, Sweden
[3] University of Sydney, Australia
pavle.subotic@it.uu.se

Abstract. Strategy iteration methods are used for solving fixed point equations. It has been shown that they improve precision in static analysis based on abstract interpretation and template abstract domains, *e.g.* intervals, octagons or template polyhedra. However, they are limited to numerical programs. In this paper, we propose a method for applying max-strategy iteration to logico-numerical programs, *i.e.* programs with numerical and Boolean variables, without explicitly enumerating the Boolean state space. The method is optimal in the sense that it computes the *least fixed point* w.r.t. the abstract domain; in particular, it does not resort to widening. Moreover, we give experimental evidence about the efficiency and precision of the approach.

Keywords: Verification, Static Analysis, Abstract Interpretation, Strategy Iteration.

1 Introduction

This paper deals with the *verification of safety properties* about *logico-numerical* programs, *i.e.*, programs manipulating Boolean and numerical variables. Verification of such properties amounts to checking whether the *reachable state space* is contained in the *invariant* specified by the property.

Classical applications are safety-critical controllers as found in modern transport systems. In such systems the properties to be proved depend on the relationships between the numerical variables of the system. The Boolean state space can be large, especially when analyzing programs written in data-flow languages like Lustre [1] and Scade[4] where the control structure is encoded in Boolean (or finite-type) variables.

Abstract interpretation. The reachability problem is undecidable for this class of programs, so analysis methods are not complete. Abstract interpretation [2] is a classical method with guaranteed termination for the price of an approximate analysis result. The key idea is to over-approximate sets of states

* This work was partly supported by the Inria large-scale initiative Synchronics.
[4] www.esterel-technologies.com

S by elements $S^\sharp$ of an *abstract domain*. A classical abstract domain for numerical invariants in $\wp(\mathbb{R}^n)$ is the domain of convex polyhedra $Pol(\mathbb{R}^n)$ [3]. An approximation $S^\sharp$ of the reachable set S is then computed by iteratively solving the fixed point equation $S = S_0 \cup \mathsf{post}(S)$ *in the abstract domain*. To ensure termination when the abstract domain contains infinitely ascending chains, one applies an extrapolation operator called *widening*, which induces additional approximations.

Strategy iteration. Strategy (or policy) iteration methods [4–9] are a way to solve the above fixed point equation *without* the need for a widening operator. The main idea of these methods is to iteratively approximate the least fixed point of $S = F(S)$ by fixed points of "simpler", more efficiently computable semantic equations $S = F^{(i)}(S)$, induced by so-called strategies, such that a fixed point of F is guaranteed to be found after a finite number of iterations. Depending on whether the least fixed point is approached from above or below, the methods are called min- or max-strategy iteration respectively.

These techniques are applied to template domains, *i.e.*, abstract domains with *a priori* fixed constraints for which constant bounds are determined during the analysis. Linear templates generate so-called *template polyhedra* [10], which subsume classical domains like intervals [11], zones [12] and octagons [13]. However, these methods are restricted to numerical programs.

Handling Boolean variables. The difficulty in dealing with logico-numerical programs is that Boolean and numerical variables tightly interact in their evolution.

The classical method to handle Boolean variables in an abstract-interpretation-based analysis is to *explicitly* unfold the Boolean control structure by *enumerating* the Boolean state space and to analyze the numerical variables on the obtained control flow graph using a numerical abstract domain: however, this raises the problem that the analysis becomes intractable already for small to medium-sized programs because the number of control locations grows exponentially with the number of Boolean variables.

Therefore we have to consider an *implicit*, *i.e.* symbolic, treatment of Booleans:

- A naive approach is to encode Booleans as *integers* $\in \{0, 1\}$. The advantage is that max-strategy iteration can be used "as is" by adding template constraints for those Booleans. Though, such an analysis will yield very rough approximations because commonly used abstract domains can only describe convex sets, whereas Boolean state sets are usually highly non-convex.
- *Logico-numerical abstract domains* aim at abstracting state sets of the form $\wp(\mathbb{B}^p \times \mathbb{R}^n)$. One way of representing such sets stems from composite model checking [14] which combines BDDs and Presburger formulas. Domains combining BDDs and numerical abstract domains like polyhedra have been proposed for example by Jeannet et al [15] and Blanchet et al [16].
- Other (not abstract-interpretation-based) approaches rely on *SAT-modulo-theory* solvers, for example the k-induction-based verification tool KIND [17].

In this paper we follow the approach of using logico-numerical abstract domains.

Contributions. Our contributions can be summarized as follows:

1. We describe a novel method for computing the set of reachable states of a logico-numerical program based on max-strategy iteration that avoids the enumeration of the Boolean state space. The technique interleaves truncated Kleene iterations in a logico-numerical abstract domain and numerical max-strategy iterations. The method is optimal, *i.e.* it computes the *least* fixed point w.r.t. the abstract semantics.
2. We give the results of an experimental evaluation: We show that our logico-numerical max-strategy iteration gains one order of magnitude in terms of efficiency in comparison to the purely numerical approach while being almost as precise. Moreover, these are the first experimental results of applying max-strategy iteration to larger programs.

Organisation of the article. §2 gives an introduction to abstract interpretation with template domains and max-strategy iteration. §3 describes the main contribution, the logico-numerical max-strategy iteration algorithm. §4 presents experimental results, and finally §5 concludes.

2 Preliminaries

Program model. We consider programs modeled as symbolic control flow graphs over a state space Σ:

Definition 1. *A symbolic control flow graph (CFG) is a directed graph* $\langle L, \mathcal{R}, \ell_0 \rangle$ *where*

- L *is a finite set of locations,* $\ell_0 \in L$ *is the initial location, and*
- $(\ell, R, \ell') \in \mathcal{R}$ *define a finite number of arcs from locations* $\ell \in L$ *to* $\ell' \in L$ *with transition relations* $R \subseteq \Sigma^2$.

An execution of a CFG is a possibly infinite sequence

$$(\ell_0, \boldsymbol{s}_0) \to^R (\ell_1, \boldsymbol{s}_1) \to \dots$$

where $\forall k \geq 0$
- $\ell_k \in L, \boldsymbol{s}_k \in \Sigma$
- $(\ell_k, \boldsymbol{s}_k) \to^R (\ell_{k+1}, \boldsymbol{s}_{k+1})$ if $R(\boldsymbol{s}_k, \boldsymbol{s}_{k+1})$

In the case of affine programs, $\Sigma = \mathbb{R}^n$ and the relations $R(\boldsymbol{x}, \boldsymbol{x'})$ are conjunctions of linear inequalities. Fig. 1 in §2.2 shows the CFG of an affine program.

The methods presented in this paper will focus on logico-numerical programs with $\Sigma = \mathbb{B}^p \times \mathbb{R}^n$ with state vectors $\boldsymbol{s} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{x} \end{pmatrix} \in \Sigma$ and relations $R(\boldsymbol{s}, \boldsymbol{s'})$ with affine, numerical subrelations.

Example 1. An example for such a logico-numerical program is the following C program:

```
b1=true; b2=true; x=0;
while(true)
{
  while(x<=19) { x = b1 ? x+1 : x-1; }
  while(x<=99) { x = b2 ? x+1 : x; b2 = !b2; }
  if (x>=100) { b1 = (x<=100); x = x-100; }
}
```

Fig. 4 in §3.2 shows a CFG corresponding to this program. Note that we allow numerical constraints in assignments to Boolean variables.

A program property we want to prove is for instance the invariant $0 \leq x \leq 100$.

2.1 Abstract interpretation with template polyhedra

Template polyhedra [10] are polyhedra the shape of which is fixed by a so-called template. The domain operations can be performed efficiently with the help of linear programming (LP) solvers.

We will use the following notations: $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$; the operators min, sup, $\leq$, etc are point-wisely lifted to vectors.

Template polyhedra abstract domain. A template polyhedron is generated by a template constraint matrix, or short template, $\mathbf{T} \in \mathbb{R}^{m \times n}$ of which each row contains at least one non-zero entry.

For example, $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ is a template constraint matrix of intervals for a system with a single variable x. It represents the constraints $x \leq d_1 \wedge -x \leq d_2$, *i.e.* $x \in [-d_2, d_1]$.

The set of template polyhedra $\mathcal{P}^{\mathbf{T}}$ generated by $\mathbf{T}$ is $\{X_{\boldsymbol{d}} \mid \boldsymbol{d} \in \overline{\mathbb{R}}^m\}$ with $X_{\boldsymbol{d}} = \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^n, \mathbf{T}\boldsymbol{x} \leq \boldsymbol{d}\}$.

An abstract value is represented by the vector of bounds $\boldsymbol{d}$. The analysis tries to find the smallest values of $\boldsymbol{d}$ representing a fixed point of the semantic equations. $\top$ and $\bot$ are naturally represented by the bound vectors $\boldsymbol{\infty}$ and $-\boldsymbol{\infty}$ respectively.

We state the definitions of the domain operations[5]:

- Concretization: $\gamma^x_{\mathbf{T}}(\boldsymbol{d}) = \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^n, \mathbf{T}\boldsymbol{x} \leq \boldsymbol{d}\},\ \boldsymbol{d} \in \overline{\mathbb{R}}^m$
- Abstraction: $\alpha^x_{\mathbf{T}}(X) = \min\{\boldsymbol{d} \in \overline{\mathbb{R}}^m \mid \gamma^x_{\mathbf{T}}(\boldsymbol{d}) \supseteq X\},\ X \subseteq \mathbb{R}^n$
- Join: $\boldsymbol{d} \sqcup^x_{\mathbf{T}} \boldsymbol{d}' = \big(\max(d_1, d'_1), \ldots, \max(d_m, d'_m)\big)$
- Image: The templates may vary from location to location: let us denote $\mathbf{T}_\ell$ the template in location ℓ and $\boldsymbol{d}_\ell$ the corresponding vector of bounds. Then the post-image by a transition relation R of transition (ℓ, R, ℓ') in a CFG is

[5] The superscript ^{x} is used to distinguish the (numerical) template polyhedra operations from the logico-numerical domain operations that we are going to define in §3.1.

defined as follows:

$$[\![R]\!]^\sharp(\boldsymbol{d}_\ell) = \sup\{\mathbf{T}_{\ell'}\boldsymbol{x}' \mid \mathbf{T}_\ell\boldsymbol{x} \leq \boldsymbol{d}_\ell \ \wedge \ R(\boldsymbol{x},\boldsymbol{x}')\}$$

i.e., given the bounds $\boldsymbol{d}_\ell$ corresponding to the template $\mathbf{T}_\ell$ it returns the bounds corresponding to $\mathbf{T}_{\ell'}$.

Reachability analysis. Let $R^j_{\ell,\ell'}$ denote a transition relation in the set of transitions from ℓ to ℓ'. We will view $\boldsymbol{d}$ alternatively as the concatenated vector of the bound vectors of all locations and the map $L \to \overline{\mathbb{R}}^m$ which assigns a vector of bounds $\boldsymbol{d}_\ell$ to each location ℓ: $\boldsymbol{d}(\ell) = \boldsymbol{d}_\ell$. $\boldsymbol{d}^0 = \lambda\ell. \begin{cases} \infty \text{ for } \ell{=}\ell_0 \\ -\infty \text{ for } \ell{\neq}\ell_0 \end{cases}$ denotes the initial values of the bounds. The set of abstract reachable states represented by the bounds $\boldsymbol{d}$ is computed by:

$$\mathit{lfp}\ \lambda\boldsymbol{d}.\big(\boldsymbol{d}^0 \sqcup^x \lambda\ell'. \bigsqcup_{R^j_{\ell,\ell'}\in\mathcal{R}} [\![R^j_{\ell,\ell'}]\!]^\sharp(\boldsymbol{d}_\ell)\big)$$

2.2 Max-strategy iteration

Max-strategy iteration [7, 8, 18–20] is a method for computing the least solution of a system of equations $\mathcal{M}$ of the form $\boldsymbol{\delta} = \boldsymbol{F}(\boldsymbol{\delta})$, where $\boldsymbol{\delta}$ are the template bounds, and F_i, $0 \leq i \leq n$ is a finite maximum of monotonic and concave operators $\mathbb{R}^n \to \mathbb{R}$; in our case they are affine functions. The max-strategy improvement algorithm for affine programs is guaranteed to compute the least fixed point of $\boldsymbol{F}$, and it has to perform at most exponentially many improvement steps, each of which takes polynomial time.

Semantic equations. The equation system $\mathcal{M}$ is constructed from the abstract semantics of the program's transitions:

$$\text{for each } \ell' \in L : \boldsymbol{\delta}_{\ell'} = \max\Big(\{\boldsymbol{d}^0_{\ell'}\} \cup \{[\![R]\!]^\sharp(\boldsymbol{\delta}_\ell) \mid (\ell, R, \ell') \in \mathcal{R}\}\Big)$$

with $\boldsymbol{d}^0$ and $[\![R]\!]^\sharp(\boldsymbol{d}_\ell)$ as defined above in §2.1.

Note that we use $\boldsymbol{\delta}$ for denoting the vector of bound variables appearing in syntactic expressions, and $\boldsymbol{d}$ for the vector carrying the actual bounds. $\delta_{\ell,i}$ is the bound variable corresponding to the i^{th} line of the template in location ℓ.

Example 2 (Semantic equations). Using the template constraints $\begin{pmatrix} 1 \\ -1 \end{pmatrix} x \leq \begin{pmatrix} d_{\ell,1} \\ d_{\ell,2} \end{pmatrix}$ in locations ℓ, the equation system for location ℓ_1 of the example in Fig. 1 consists of one equation for each template bound variable of which the arguments of the max operator are the initial value $-\infty$ and one expression $[\![R]\!]^\sharp$ for each of the three incoming arcs:

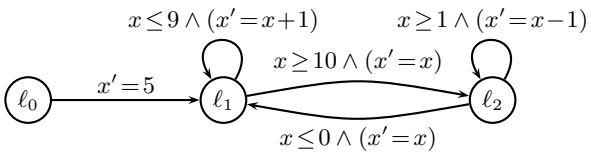

Fig. 1: CFG of an affine program

$$\delta_{1,1} = \max \left\{ \begin{array}{l} -\infty, \\ \sup\{\ x' \mid x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5 \qquad\qquad \}, \\ \sup\{\ x' \mid x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge\ x \leq 9\ \wedge x' = x+1\ \}, \\ \sup\{\ x' \mid x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge\ x \leq 0\ \wedge\ \ x' = x\ \ \} \end{array} \right\}$$

$$\delta_{1,2} = \max \left\{ \begin{array}{l} -\infty, \\ \sup\{\ -x' \mid x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5 \qquad\qquad \}, \\ \sup\{\ -x' \mid x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge\ x \leq 9\ \wedge x' = x+1\ \}, \\ \sup\{\ -x' \mid x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge\ x \leq 0\ \wedge\ \ x' = x\ \ \} \end{array} \right\}$$

Strategies. A *strategy* μ induces a "subsystem" $\boldsymbol{\delta} = \widehat{\boldsymbol{F}}(\boldsymbol{\delta})$ of $\mathcal{M}$ in the sense that exactly one argument $\widehat{F_i}$ of the max operator on the right-hand side of each equation $\delta_i = \max(\ldots, \widehat{F_i}(\boldsymbol{\delta}), \ldots)$ is chosen. Intuitively, this means that a strategy selects exactly one "incoming transition" for each template bound variable in each location ℓ'.

Example 3 (Strategy). A strategy in the example in Fig. 1 corresponds for instance the following system of equations:

$$\begin{array}{l} \delta_{0,1} = \infty \\ \delta_{0,2} = \infty \\ \delta_{1,1} = \sup\{\ \ \ x' \mid x \leq \delta_{1,1}\ \wedge\ -x \leq \delta_{1,2}\ \wedge\ x \leq 9\ \wedge x' = x+1\ \} \\ \delta_{1,2} = \sup\{\ -x' \mid x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5 \qquad\qquad \} \\ \delta_{2,1} = -\infty \\ \delta_{2,2} = -\infty \end{array}$$

We see that for $\delta_{1,1}$ we have chosen the third and for $\delta_{1,2}$ the second argument of the max operators in the equations of Example 2.

One has to compute the least solution $\mathit{lfp}[\![\mathcal{M}]\!]$ of the system $\mathcal{M}$, where $[\![\mathcal{M}]\!]$ is defined as

$$[\![\mathcal{M}]\!](\boldsymbol{d}) = \max_{\mu \text{ in } \mathcal{M}} [\![\mu]\!](\boldsymbol{d})$$

and with $[\![\mu]\!](\boldsymbol{d}) = [\![\boldsymbol{\delta} = \widehat{\boldsymbol{F}}(\boldsymbol{\delta})]\!](\boldsymbol{d}) = \widehat{\boldsymbol{F}}(\boldsymbol{d})$.

Max-strategy improvement. $\mathit{lfp}[\![\mathcal{M}]\!]$ is computed with the help of the max-strategy improvement algorithm (see Fig. 2) which iteratively improves strategies μ until the least fixed point $\mathit{lfp}[\![\mu]\!]$ of a strategy equals $\mathit{lfp}[\![\mathcal{M}]\!]$.

initial strategy: $\mu := \{\boldsymbol{\delta}_{\ell_0} = \boldsymbol{\infty},\ \boldsymbol{\delta}_\ell = -\boldsymbol{\infty} \text{ for all } \ell \neq \ell_0\}$
initial bounds: $\boldsymbol{d} := \lambda\ell.\boldsymbol{\delta}_\ell \to \begin{cases} \boldsymbol{\infty} & \text{for } \ell = \ell_0 \\ -\boldsymbol{\infty} & \text{for } \ell \neq \ell_0 \end{cases}$
while not $\boldsymbol{d}$ is a solution of $\mathcal{M}$ do
  $\mu := \mathsf{max\_improve}_{\mathcal{M}}(\mu, \boldsymbol{d})$
  $\boldsymbol{d} := \mathit{lfp}[\![\mu]\!]$
done
return $\boldsymbol{d}$

Fig. 2: Max-strategy iteration algorithm

The least fixed point $\mathit{lfp}[\![\mu]\!]$ of a strategy μ can be computed by solving the LP problem with the constraint system

$$\text{for each } \left(\boldsymbol{\delta}_{\ell'} = [\![R]\!]^\sharp(\boldsymbol{\delta}_\ell)\right) \text{ in } \mu : \boldsymbol{d}_{\ell'} \leq \mathbf{T}_{\ell'}\boldsymbol{x}' \ \wedge\ \mathbf{T}_\ell\boldsymbol{x} \leq \boldsymbol{d}_\ell \ \wedge\ R(\boldsymbol{x}, \boldsymbol{x}')$$

(where $\boldsymbol{x}$ and $\boldsymbol{x}'$ are auxiliary variables) and the objective function $\max \sum_i d_i$, *i.e.* the sum of all bounds $\boldsymbol{d}$.

μ' is called an *improvement* of μ w.r.t. $\boldsymbol{d}$, *i.e.* $\mu' = \mathsf{max\_improve}_{\mathcal{M}}(\mu, \boldsymbol{d})$ iff

1. μ' is "at least as good" as μ: $[\![\mu']\!](\boldsymbol{d}) \geq [\![\mu]\!](\boldsymbol{d})$, and
2. μ' is "strictly better for the changed equations": if $(\delta_i = \widehat{F}_i(\boldsymbol{\delta}))$ in μ and $(\delta_i \geq \widehat{F}'_i(\boldsymbol{\delta}))$ in μ' and $\widehat{F}_i \neq \widehat{F}'_i$, then $\widehat{F}'(\boldsymbol{d}) > \widehat{F}(\boldsymbol{d})$.

Example 4 (Max-strategy iteration). We illustrate some steps of the analysis of the example in Fig. 1. Assume the current strategy is:

$$\mu_1 = \left\{ \begin{array}{l} \delta_{0,1} = \infty \\ \delta_{0,2} = \infty \\ \delta_{1,1} = \sup\{\ \ x' \mid x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5\ \ \} \\ \delta_{1,2} = \sup\{\ -x' \mid x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5\ \ \} \\ \delta_{2,1} = -\infty \\ \delta_{2,2} = -\infty \end{array} \right\}$$

and the current template bounds are:

$$\boldsymbol{d}_1 = \left\{ \begin{array}{ll} \delta_{0,1} \to \infty & \delta_{0,2} \to \infty \\ \delta_{1,1} \to 5 & \delta_{1,2} \to -5 \\ \delta_{2,1} \to -\infty & \delta_{2,2} \to -\infty \end{array} \right\}$$

The strategy can only be improved w.r.t. $\delta_{1,1}$:

$$\mu_2 = \left\{ \begin{array}{l} \delta_{0,1} = \infty \\ \delta_{0,2} = \infty \\ \delta_{1,1} = \sup\{\ \ x' \mid x \leq \delta_{1,1} \ \wedge\ -x \leq \delta_{1,2} \ \wedge\ x \leq 9 \ \wedge\ x' = x+1\ \} \\ \delta_{1,2} = \sup\{\ -x' \mid x \leq \delta_{0,1} \wedge -x \leq \delta_{0,2} \wedge x' = 5\ \ \ \ \ \ \ \ \ \ \ \ \} \\ \delta_{2,1} = -\infty \\ \delta_{2,2} = -\infty \end{array} \right\}$$

We compute the new fixed point w.r.t. μ_2:

$$\boldsymbol{d}_2 = lfp[\![\mu_2]\!] = \left\{ \begin{array}{ll} \delta_{0,1} \to \infty & \delta_{0,2} \to \infty \\ \delta_{1,1} \to 10 & \delta_{1,2} \to -5 \\ \delta_{2,1} \to -\infty & \delta_{2,2} \to -\infty \end{array} \right\}$$

In the next step we can improve the strategy w.r.t. $\delta_{2,1}$ and $\delta_{2,2}$, a.s.o.

An improving strategy is selected by testing for each equation whether an argument of its max-operator leads to a greater bound. Since the arguments of the max-operator are required to be monotonic, the bounds are always monotonically increasing and, thus, arguments that have already been selected in previous strategies need not be considered again.

3 Logico-Numerical Max-Strategy Iteration

We will present an algorithm which enables the use of max-strategy iteration in a logico-numerical context, *i.e.* programs with a state space $\mathbb{B}^p \times \mathbb{R}^n$.

3.1 Abstract Domain

We consider the logico-numerical abstract domain $A = \wp(\mathbb{B}^p) \times \overline{\mathbb{R}}^m$ which combines Boolean formulas with template polyhedra. An abstract value $(B, \boldsymbol{d})$ consists of the cartesian product of valuations of the Boolean variables B (represented as Boolean formulas using BDDs for example) and the template bounds $\boldsymbol{d}$. We define the domain operations:

- Abstraction: $\alpha_{\mathbf{T}}(S) = \begin{pmatrix} \{\boldsymbol{b} \mid \exists \boldsymbol{x} : (\boldsymbol{b}, \boldsymbol{x}) \in S\} \\ \min\{\boldsymbol{d} \mid (\boldsymbol{b}, \gamma^x_{\mathbf{T}}(\boldsymbol{d})) \in S\} \end{pmatrix}$
- Concretization: $\gamma_{\mathbf{T}}(S^\sharp) = B \wedge \gamma^x_{\mathbf{T}}(\boldsymbol{d})$
- Join: $\begin{pmatrix} B \\ \boldsymbol{d} \end{pmatrix} \sqcup_{\mathbf{T}} \begin{pmatrix} B' \\ \boldsymbol{d}' \end{pmatrix} = \begin{pmatrix} B \vee B' \\ \boldsymbol{d} \sqcup^x_{\mathbf{T}} \boldsymbol{d}' \end{pmatrix}$
- Image: $[\![R_{\ell,\ell'}]\!]^\sharp \begin{pmatrix} B \\ \boldsymbol{d} \end{pmatrix} = \begin{pmatrix} \{\boldsymbol{b}' \mid \mathbf{T}_\ell \boldsymbol{x} \leq \boldsymbol{d} \wedge \boldsymbol{b} \in B \wedge R(\boldsymbol{b}, \boldsymbol{b}', \boldsymbol{x}, \boldsymbol{x}')\} \\ \sup\{\mathbf{T}_{\ell'} \boldsymbol{x}' \mid \mathbf{T}_\ell \boldsymbol{x} \leq \boldsymbol{d} \wedge \boldsymbol{b} \in B \wedge R(\boldsymbol{b}, \boldsymbol{b}', \boldsymbol{x}, \boldsymbol{x}')\} \end{pmatrix}$

$\top$ and $\bot$ are defined as $\begin{pmatrix} \mathsf{tt} \\ \boldsymbol{\infty} \end{pmatrix}$ and $\begin{pmatrix} \mathsf{ff} \\ -\boldsymbol{\infty} \end{pmatrix}$ respectively.

Since we are analyzing a CFG with locations L, we have the overall abstract domain $\Sigma^\sharp = L \to A$. An abstract value $S^\sharp = \lambda\ell.(B_\ell, \boldsymbol{d}_\ell) \in \Sigma^\sharp$ assigns to each location a value of the above logico-numerical domain. Note that the dimension m of $\boldsymbol{d}_\ell$ may depend on ℓ if the templates differ from location to location.

3.2 Algorithm

Our analysis is based on alternating (1) a truncated Kleene iteration over the logico-numerical abstract domain and (2) numerical max-strategy iteration, see Fig. 3.

```
1  S := S^0
2  S' = post(S)
3  while ¬stable(S, S') do
4     while ¬p_stable(S, S') do                    ⎫
5        S := S'                                   ⎪
6        S' = post(S)                              ⎬ phase (1): truncated logico-numerical Kleene iteration
7     done                                         ⎪
8     S := S'                                      ⎭
9     M = generate(S)
10    μ := (δ = d)
11    μ' = max_improve_M(μ, d)
12    while μ' ≠ μ do                              ⎫
13       μ := μ'                                   ⎪
14       d := lfp⟦μ⟧                               ⎬ phase (2): numerical max-strategy iteration
15       μ' = max_improve_M(μ, d)                  ⎪
16    done                                         ⎭
17    S' = post(S)
18 done
19 return S
```

Fig. 3: Logico-numerical max-strategy iteration algorithm

The truncated Kleene iteration (propagation) explores the system until a certain criterion is satisfied; we say that the system *preliminarily stable*. We use the following criterion: we stop Kleene iteration if for all locations the set of reachable Boolean states does not change whatever transition we take. The underlying idea is to discover a subsystem, in which Boolean variables are stable during a presumably larger number of iterations. In such a subsystem numerical variables evolve, while Boolean transitions switch only within the system, *i.e.* they do not "discover" new Boolean states, until numerical conditions are satisfied that make Boolean variables leave the subsystem.

We use max-strategy iteration (phase (2)) to compute the fixed point for the numerical variables for such a subsystem. Then Kleene iteration (phase (1)) continues exploring the next preliminary stable subsystem. The algorithm terminates in a finite number of steps as soon as the Kleene iteration of phase (1) has reached a fixed point.

Formal description. See Fig. 3. Since we only manipulate abstract quantities, we will omit the superscript $^\sharp$ in the sequel in order to improve readability.

For phase (1) we use the definitions:

– Initial abstract value: $S^0 = \lambda\ell.\begin{cases}\top \text{ for } \ell{=}\ell_0\\ \bot \text{ for } \ell{\neq}\ell_0\end{cases}$

– Post-condition: $\mathsf{post}(S) = \lambda\ell'.S(\ell') \sqcup \bigsqcup_\ell [\![R_{\ell,\ell'}]\!](S(\ell))$

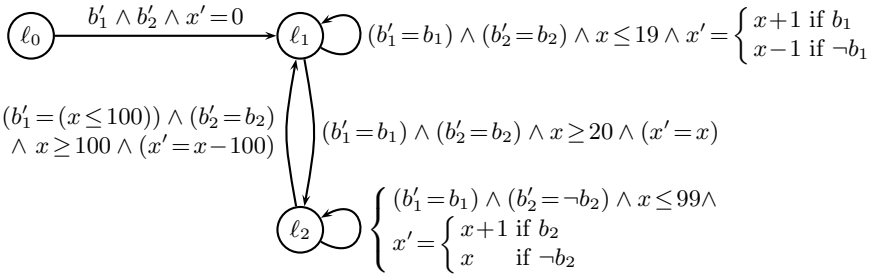

Fig. 4: CFG of the program in Example 1

– Condition for preliminary stability: $\mathsf{p\_stable}(S, S') = (\forall \ell : B_\ell = B'_\ell)$
– Condition for stability (global convergence): $\mathsf{stable}(S, S') = (S = S')$

For phase (2) we define the following:
– The max-strategy improvement operator $\mathsf{max\_improve}_{\mathcal{M}}$ is defined as described in §2.
– The operator $\mathsf{generate}$ dynamically derives the system of equations for the current preliminary stable subsystem: this means that we restrict the system to those transitions which stay within the subsystem defined by the current Boolean states $\lambda\ell.B_\ell$. For this purpose we conjoin the term $\boldsymbol{b} \in B_\ell \wedge \boldsymbol{b'} \in B_{\ell'}$ to the transition relation in the definition below. Strategies may only contain convex constraints: thus, we transform the relation into disjunctive normal form and generate one strategy per disjunct:

$$\mathsf{generate}(S) = \bigcup_{\ell',\ell} \mathsf{decomp\_convex}(\exists \boldsymbol{b}, \boldsymbol{b'} : R_{\ell,\ell'}(\boldsymbol{b}, \boldsymbol{x}, \boldsymbol{b'}, \boldsymbol{x'}) \wedge \boldsymbol{b} \in B_\ell \wedge \boldsymbol{b'} \in B_{\ell'})$$

where $\mathsf{decomp\_convex}(R_{\ell,\ell'}) = \bigcup_j \left(\boldsymbol{\delta}_{\ell'} = [\![R^j_{\ell,\ell'}]\!](\boldsymbol{\delta}_\ell)\right)$ with $R_{\ell,\ell'} = \bigvee_j R^j_{\ell,\ell'}$ and $R^j_{\ell,\ell'}$ convex.

Remark 1. Since the bounds $\boldsymbol{d}$ are monotonically increasing, we use the constant strategy $\boldsymbol{\delta} = \boldsymbol{d}$ (where $\boldsymbol{d}$ are the previously obtained bounds) as initial strategy for phase (2) (see line 10 in Fig. 3). This prevents the numerical max-strategy improvement from restarting with $\boldsymbol{\delta} = -\boldsymbol{\infty}$ each time.

We illustrate this algorithm by applying it to the CFG in Fig. 4:

Example 5. We use the template constraint matrix $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ which corresponds to an interval analysis. In order to make the presentation more concise, we will write states $(\ell \to \begin{pmatrix} B \\ \boldsymbol{d} \end{pmatrix}) \in \Sigma$ as $\ell \to \begin{pmatrix} \varphi(b_1, b_2) \\ [-\delta_{\ell,2}, \delta_{\ell,1}] \end{pmatrix}$ where φ is a Boolean formula.

The initial state in ℓ_0 is $\begin{pmatrix} \mathsf{tt} \\ [-\infty,\infty] \end{pmatrix}$. We start exploring the system by taking transition (ℓ_0, R, ℓ_1): $\ell_1 \to \begin{pmatrix} b_1 \wedge b_2 \\ [0,0] \end{pmatrix}$. We continue propagating through (ℓ_1, R, ℓ_1): $\ell_1 \to \begin{pmatrix} b_1 \wedge b_2 \\ [0,1] \end{pmatrix}$. Now, we have reached preliminary stability because none of the transitions makes us discover new Boolean states in the next iteration ((ℓ_0, R, ℓ_1) and (ℓ_1, R, ℓ_1) yield nothing new w.r.t. Boolean states and the other transitions are infeasible, *i.e.*, they give $\bot$). Hence, we go ahead to phase (2) and extract the numerical equation system for each (ℓ, R, ℓ'). *E.g.* for (ℓ_1, R, ℓ_1), we compute:

$$\exists \boldsymbol{b}, \boldsymbol{b'} : R \wedge b_1 \wedge b_2 \wedge b_1' \wedge b_2' = (x' = x{+}1 \wedge x \leq 19)$$

which gives us the partial equations:

$$\begin{aligned} \delta_{1,1} &= \sup\{ \quad x' \mid x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge x' = x{+}1 \wedge x \leq 19 \} \\ \delta_{1,2} &= \sup\{ -x' \mid x \leq \delta_{1,1} \wedge -x \leq \delta_{1,2} \wedge x' = x{+}1 \wedge x \leq 19 \} \end{aligned}$$

which have to be completed by the other incoming arcs of ℓ_1. We start the max-strategy iteration with the strategy corresponding to the values obtained in phase (1):

$$\mu = \{\boldsymbol{\delta}_0 = \boldsymbol{\infty}, \qquad \delta_{1,1} = 1, \qquad \delta_{1,2} = 0, \qquad \boldsymbol{\delta}_2 = \boldsymbol{-\infty}\}$$

We observe that we can improve this strategy w.r.t. $\delta_{1,1}$:

$$\mu' = \left\{ \begin{array}{l} \boldsymbol{\delta}_1 = \boldsymbol{\infty} \\ \delta_{1,1} = \sup\{x' \mid x \leq \delta_{1,1} \ \wedge \ -x \leq \delta_{1,2} \ \wedge \ x' = x{+}1 \ \wedge \ x \leq 19\}, \quad \delta_{1,2} = 0 \\ \boldsymbol{\delta}_2 = \boldsymbol{-\infty} \end{array} \right\}$$

The max-strategy iteration terminates with: $\ell_1 \to \begin{pmatrix} b_1 \wedge b_2 \\ [0,20] \end{pmatrix}$.

We continue propagating (phase (1)): By (ℓ_1, R, ℓ_2) we get $\ell_2 \to \begin{pmatrix} b_1 \wedge b_2 \\ [20,20] \end{pmatrix}$; then (ℓ_2, R, ℓ_2) results in $\begin{pmatrix} b_1 \wedge \neg b_2 \\ [21,21] \end{pmatrix}$; by joining these values we get $\ell_2 \to \begin{pmatrix} b_1 \\ [20,21] \end{pmatrix}$. Taking (ℓ_2, R, ℓ_2) a second time does not change the Boolean state: $\ell_2 \to \begin{pmatrix} b_1 \\ [20,22] \end{pmatrix}$. Taking any other transition does not discover new Boolean states either, thus, we move on to phase (2) and compute the numerical equation system w.r.t. the current Boolean state: For example for (ℓ_2, R, ℓ_2), we compute

$$(\exists \boldsymbol{b}, \boldsymbol{b'} : R \wedge b_1 \wedge b_1') = \big((x' = x{+}1 \vee x' = x) \wedge x \leq 99\big)$$

which results in the partial equations

$$\begin{aligned} \delta_{2,1} &= \max \left\{ \begin{array}{l} \sup\{ \ x' \mid x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x' = x{+}1 \wedge x \leq 99 \ \}, \\ \sup\{ \ x' \mid x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x' = x \quad \wedge x \leq 99 \ \} \end{array} \right\} \\ \delta_{2,2} &= \max \left\{ \begin{array}{l} \sup\{ -x' \mid x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x' = x{+}1 \wedge x \leq 99 \ \}, \\ \sup\{ -x' \mid x \leq \delta_{2,1} \wedge -x \leq \delta_{2,2} \wedge x' = x \quad \wedge x \leq 99 \ \} \end{array} \right\} \end{aligned}$$

which have to be completed by the other incoming arcs of ℓ_2. The only possible improvement w.r.t to the current state is w.r.t. $\delta_{2,1}$; phase (2) results in $\ell_2 \to \begin{pmatrix} b_1 \\ [20,100] \end{pmatrix}$.

We continue with phase (1), which filters the above value through (ℓ_2, R, ℓ_1) augmenting the abstract value in ℓ_1 to $\begin{pmatrix} b_1 \\ [0,20] \end{pmatrix}$. Then, *none* of the transitions makes the reachable state sets increase (neither Boolean nor numerical), hence we have reached the global fixed point:

$$\ell_0 \to \top, \quad \ell_1 \to \begin{pmatrix} b_1 \\ [0,20] \end{pmatrix}, \quad \ell_2 \to \begin{pmatrix} b_1 \\ [20,100] \end{pmatrix}$$

Note that a logico-numerical analysis in the same domain with widening and descending iterations yields no information about this example: $S = \lambda\ell.\top$.

3.3 Properties

Theorem 1 (Termination). *The logico-numerical max-strategy algorithm terminates after a finite number of iterations.*

Proof. Termination follows from these observations:
(a) Phase (1) only propagates as long as new Boolean states are discovered; the number of Boolean states is finite.
(b) Max-strategy iteration is called at most once for each subset of Boolean states. The number of subsets of Boolean states is finite.
(c) There is a unique system of numerical equations (built by generate) for each subset of Boolean states.
(d) Max-strategy iteration terminates after a finite number of improvement steps, because there is a finite number of strategies and each strategy is visited at most once [8].
(e) Max-strategy iteration returns the unique least fixed point w.r.t. the given system of equations [8].

Thus, as soon as all reachable Boolean states have been discovered and the associated numerical fixed point has been computed, the overall fixed point has been reached and the algorithm terminates.

Theorem 2 (Soundness). *The logico-numerical max-strategy algorithm computes a fixed point of* $\lambda S.S_0 \sqcup [\![R]\!]^\sharp(S)$.

Proof. Let us denote
- $F = \lambda S.S_0 \sqcup [\![R]\!]^\sharp(S)$.
- $\lambda S.(lfp^B\ F)(S)$ the truncated Kleene iteration phase (1) (lines 4 to 7 in Fig. 3), *i.e.* $\lambda S.\big($while $B \neq B'$ do $S := S'; S' = F(S)$ end; return $S'\big)$.

- $\lambda(B, \boldsymbol{d}).(B, (\mathit{lfp}^X\ F^X)(\boldsymbol{d}))$ the max-strategy iteration in phase (2) (lines 9 to 16 in Fig. 3),
  i.e. $\lambda S.\big(\ \mathcal{M} = \mathsf{generate}(S); \mu := (\boldsymbol{\delta} = \boldsymbol{d}); \mu' = \mathsf{max\_improve}_{\mathcal{M}}(\mu, \boldsymbol{d});$
  while $\mu' \neq \mu$ do $\mu := \mu'; \boldsymbol{d} := \mathit{lfp}[\![\mu]\!]; \mu' = \mathsf{max\_improve}_{\mathcal{M}}(\mu, \boldsymbol{d})$ done;
  return $S\big)$.

Then, we can write the whole algorithm as $\mathit{lfp}((id^B, \mathit{lfp}^X\ F^X) \circ (\mathit{lfp}^B\ F) \circ F)$.

Since $\mathit{lfp}^B\ F$ and $(id^B, \mathit{lfp}^X\ F^X)$ are both extensive, we can under-approximate them by $id = \lambda S.S$. Hence, we conclude from

$$\mathit{lfp}\ F \sqsubseteq \mathit{lfp}(\underbrace{id}_{id \sqsubseteq (id^B, \mathit{lfp}^X F^X)} \circ \underbrace{id}_{id \sqsubseteq (\mathit{lfp}^B F)} \circ F)$$

that our algorithm computes an over-approximation of the least fixed point, *i.e.* it is sound.

Theorem 3 (Optimality). *The logico-numerical max-strategy algorithm computes the* least *fixed point of* $\lambda S.S_0 \sqcup [\![R]\!]^\sharp(S)$.

Proof. Additionally to Thm. 2, we have to show that
$$\mathit{lfp}((id^B, \mathit{lfp}^X\ F^X) \circ (\mathit{lfp}^B\ F) \circ F) \sqsubseteq \mathit{lfp}\ F.$$
(a) Phase (1) computes a certain number of iterations $F^k(\bot) = (\mathit{lfp}^B\ F) \circ F(\bot)$ taking into account the whole transition system. We trivially have $F^k(\bot) \sqsubseteq \mathit{lfp}\ F(\bot)$.
(b) Phase (2) $(id^B, \mathit{lfp}^X\ F^X)$ iterates over the transitions of a subsystem. It is known [8] that it computes the least fixed point w.r.t. this subsystem. Hence, the result of phase (2) cannot go beyond the fixed point of the whole system: $(id^B, \mathit{lfp}^X\ F^X) \circ F^k(\bot) \sqsubseteq \mathit{lfp}\ F$.
(c) We can repeat arguments (a) and (b) for the outer loop:
$\ldots \circ (id^B, \mathit{lfp}^X\ F^X) \circ F^{k_2} \circ (id^B, \mathit{lfp}^X\ F^X) \circ F^{k_1}(\bot) \sqsubseteq \mathit{lfp}\ F$
where k_n is the number of iterations of the n^{th} phase (1).
Thus, we have $((id^B, \mathit{lfp}^X\ F^X) \circ F^{k_n})^n(\bot) \sqsubseteq \mathit{lfp}\ F$ for $n \geq 0$.

Hence, we conclude from $\mathit{lfp}((id^B, \mathit{lfp}^X\ F^X) \circ (\mathit{lfp}^B\ F) \circ F) \sqsubseteq \mathit{lfp}\ F$ and Thm. 2 that our algorithm computes the least fixed point, *i.e.* it is optimal.

3.4 Application to data-flow programs

For our experiments in §4, we used Lustre programs, *i.e.* synchronous data-flow programs. For this reason we will briefly explain how to apply our algorithm to such programs.

Lustre programs can be reduced to a symbolic transition system $\begin{cases} \mathcal{I}(\boldsymbol{s}) \\ \mathcal{A}(\boldsymbol{s}, \boldsymbol{i}) \rightarrow \boldsymbol{s}' = \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{i}) \end{cases}$ where $\boldsymbol{s} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{x} \end{pmatrix}$ and $\boldsymbol{i} = \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\xi} \end{pmatrix}$ are the vectors of (Boolean and numerical) state and input variables, $\mathcal{I}(\boldsymbol{s})$ is the initial condition, $\mathcal{A}(\boldsymbol{s}, \boldsymbol{i})$ is an *assertion* constraining input variables depending on state variables, and $\boldsymbol{f}$ is the vector of transition functions.

State space partitioning is used to obtain a CFG in which each equivalence class of the partition corresponds to a location.

The transition relations are constructed by

$$R_{\ell,\ell'} = \exists \boldsymbol{\beta} : \left\{ \begin{array}{l} \boldsymbol{x}' = \boldsymbol{f}^x(\boldsymbol{b}, \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi}) \\ \boldsymbol{b}' = \boldsymbol{f}^b(\boldsymbol{b}, \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi}) \end{array} \right\} \wedge \psi_\ell(\boldsymbol{x}, \boldsymbol{b}) \wedge \psi_{\ell'}(\boldsymbol{x}', \boldsymbol{b}') \wedge \mathcal{A}(\boldsymbol{b}, \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi})$$

where $f_i^x = \left\{ \begin{array}{l} \cdots \\ a_{ij}(\boldsymbol{x}, \boldsymbol{\xi}) \text{ if } \phi_{ij}(\boldsymbol{b}, \boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{\xi}) \\ \cdots \end{array} \right.$, and ψ_ℓ are the definitions of the partitions (locations).

Boolean input variables are quantified existentially. Numerical inputs appear as auxiliary variables (*i.e.*, variables without associated bounds) in the max-strategy iteration, hence, they are treated without modification of the algorithms.

3.5 Discussion

An important observation is that, since the overall abstract domain is of the form $L \to \wp(\mathbb{B}^p) \times \overline{\mathbb{R}}^m$, the choice of the CFG has two effects on the performance: first, it determines the set of representable abstract properties, and second, it influences the approximations made in the generation of the numerical equation system for the max-strategy iteration phase (2), because there is only one template polyhedron per location.

Generalization. The structure of the algorithm we presented is quite general. In particular, it does not depend on the method used to compute the numerical least fixed point in phase (2). We conjecture that the algorithm makes every method, that is able to compute the least fixed point of a numerical system by ascending iterations, compute the least fixed point of a logico-numerical system.

For example, we suppose that our algorithm can be used without any modification with the variant of max-strategy iteration for quadratic programs and quadratic templates proposed in [19].

If a method computes the fixed point by descending iterations, as for example min-strategy iteration [4, 5], our algorithm can still be used, but requires a small adaptation because the abstract value computed in phase (1), which is an under-approximation of the least fixed point, cannot be used to initialize phase (2), which requires an over-approximation: hence, line 10 in Fig. 3 must be replaced by guessing appropriate initial bounds and an initial strategy for phase (2). This makes the algorithm less elegant and the analysis, probably, less efficient.

Logico-numerical max-strategy iteration using a power domain. The algorithm is also rather generic w.r.t. the kind of logico-numerical abstract domain we use. For example, we could consider the logico-numerical power domain $\mathbb{B}^p \to \wp(\mathbb{R}^n)$ where $\wp(\mathbb{R}^n)$ is abstracted by any domain that is supported by strategy iteration. Then the overall domain for our method is $L \to \mathbb{B}^p \to \overline{\mathbb{R}}^m$. This domain implicitly dynamically partitions each location into sub-locations corresponding to Boolean valuations sharing a common numerical abstract value[6].

[6] This sharing can be implemented with the help of MTBDDs where the numerical abstract values are stored in the leaves [21].

The construction of the equation system (generate in our algorithm, Fig. 3) must take into account these partitions.

This domain is more precise than the product domain described in §3, however, the drawback is that the number of partitions might explode if only few Boolean valuations share a common numerical abstract value.

Comparison with logico-numerical min-strategy iteration. The power domain $\mathbb{B}^p \to \overline{\mathbb{R}}^m$ is also used by Sotin et al [22] who propose an approach to analyzing logico-numerical programs using min-strategies. In accordance with the form of the abstract domain, they consider logico-numerical strategies $\mathbb{B}^p \to \Pi$ (where Π is the set of min-strategies), which dynamically associates the numerical min-strategies to the reachable Boolean states during analysis. They start with an initial logico-numerical strategy $P^{(0)} = \lambda \boldsymbol{b}.\pi^{(0)}$ with a chosen numerical min-strategy $\pi^{(0)}$ and compute a fixed point using logico-numerical Kleene iteration with widening and descending iterations. Then they iteratively improve the min-strategies in $P^{(i)}$ and recompute the fixed point.

This approach does not integrate well with mathematical programming because the only known method for computing the fixed point of a logico-numerical strategy is logico-numerical Kleene iteration (with widening). Hence, in contrast to our approach, there is no guarantee to compute the least fixed point.

Comparison with abstract acceleration. Numerous methods have been developed to alleviate the problem of bad extrapolations due to widening, *e.g.* abstract acceleration [23–25], a method for computing the transitive closure of numerical loops. These methods are able to accelerate some cases of self-loops and cycles with certain types of affine transformations, and they rely on widening in the general case. However, due to the use of general convex polyhedra, they are able to "discover" complex invariant constraints.

In contrast, max-strategy iteration is able to "accelerate" globally the whole transition system regardless of the graph structure or type of affine transformation, and it effectively computes the least fixed point. However, this is only possible on the simpler domain of template polyhedra.

Although the use of template polyhedra is a restriction, this kind of (static) approximation is much more predictable than the (dynamic) approximations made by widening.

Remark 2. Guided static analysis [26] is a framework for analyzing monotonically increasing subsystems, which makes it possible to reduce the impact of widening by applying descending iterations "in the middle" of an analysis. Our algorithm proceeds in a similar fashion – although for different technical reasons – by applying max-strategy iteration on monotonically increasing subsystems.

4 Experimental Evaluation

We implemented the algorithm in our reactive system verification tool ReaVer[7], which is based on the logico-numerical abstract domain library BddApron [21]

[7] `http://pop-art.inrialpes.fr/people/schramme/reaver/`

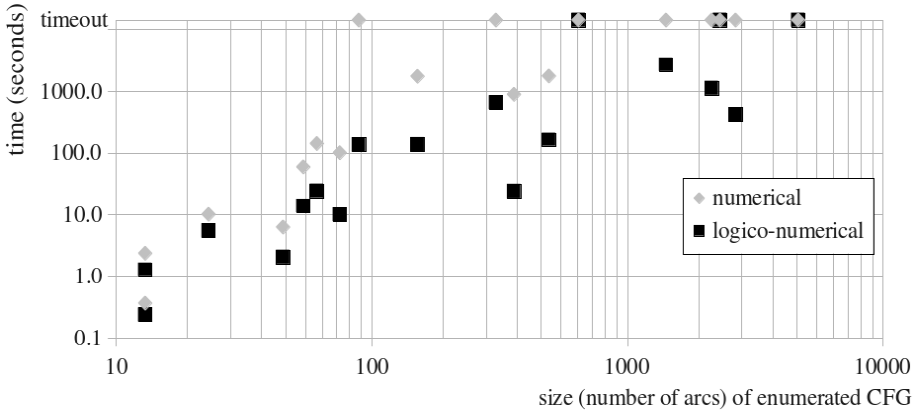

Fig. 5: Scalability of *logico-numerical* max-strategy iteration in comparison with *numerical* max-strategy iteration on the enumerated CFG, using octagonal constraints. The timeout was set to 3600 seconds. Note the logarithmic scales.

and an improved version of the max-strategy iteration solver of Gawlitza et al [27]. Since template polyhedra are not yet implemented in the Apron library [28], we emulated template polyhedra operations in phase (1) with the help of general polyhedra, which certainly impaired the performance – nonetheless we obtained encouraging results.

Benchmarks. We took 18 benchmarks[8] used in [25]. These are high-level simulation models (programmed in Lustre) of manufacturing systems which consist of building blocks like sources, buffers, machines and routers that synchronize via handshakes (for this reason there are many Boolean variables). The properties to be verified like maximal throughput time depend on numerical variables. These programs have up to a few hundred lines of code, 27 Boolean and 7 numerical variables, which produce enumerated CFGs of up to 650 locations and 5000 transitions after simplification by Boolean reachability. The focus of the experiments was on comparing the precision of the inferred invariants rather than proving properties.

Results. We performed experiments with octagonal constraints ($\pm x_i$, $x_i \pm x_j$) in order to evaluate efficiency and precision. We compared max-strategy iteration on the enumerated CFGs (MSI) with logico-numerical max-strategy iteration (LNMSI) on CFGs obtained by the static partitioning method by Boolean states implying the same numerical transitions ("numerical modes") from [25]. The resulting CFGs are on average five times smaller than the enumerated CFGs for the medium-sized benchmarks.

– LNMSI scales clearly better than MSI (see Fig. 5): our method was on average 9 times faster – for those benchmarks where both methods terminated before

[8] The benchmarks can be downloaded from:
`http://pop-art.inrialpes.fr/people/schramme/maxstrat/`

the timeout: MSI hit the timeout in 8 out of 18 cases (versus 3 for our method). The gain in efficiency grows with increasing benchmark sizes.
- The precision is almost preserved: only 0.38% (!) of the bounds were worse but still finite, and 0.16% were lost. This precision loss did not impact the number of proved properties. Due to the better scalability we were even able to prove 3 more benchmarks (10 as opposed to 7).
- We compared the precision of LNMSI with octagonal constraints with a logico-numerical analysis with octagons using the standard approach with widening ($N=2$) and 2 descending iterations on the same CFG. On average 18% of the bounds of our invariants were strictly better than those computed using the standard analysis. In 2 cases these improvements made the difference to prove the property. However, the standard analysis was 19 times faster on average.

Furthermore, we experimented with different templates and CFG sizes:
- The average gain in speed increases with the template size: 3.3 for interval analysis ($\pm x_i$), 5 for zones ($\pm x_i$, $x_i - x_j$) and 9 for octagons (for those benchmarks which did not run into timeouts).
- The precision of LNMSI depends on the CFG size: the general trend is "the bigger the more precise", but the results are less clear: CFGs of the same size seem to have very different *quality* w.r.t. precision. Partitioning methods which find *good* partitions matter!
- A smaller CFG does not automatically mean faster analysis: the fact that a smaller graph means more complicated logico-numerical transition functions and more numerical strategies per location outweighs the advantage of dealing with less locations.
- It is interesting that LNMSI scales also better on the enumerated CFG: it seems to be advantageous to start with a small system with few strategies, iteratively increase the system, and finally, when computing the numerical fixed point of the full system, most of the strategies are already known not to improve the bounds, and thus max-strategy iteration converges faster.

We also experimented with LNMSI using the logico-numerical power domain discussed in §3.5, which performed on our CFGs still 6 to 7 times faster on average and with a 100% preservation of bounds compared to MSI.

5 Conclusions

We presented *logico-numerical max-strategy iteration*, a solution to the intricate problem of combining numerical max-strategy iteration with techniques that are able to deal with Boolean variables implicitly and therefore allow to trade precision for efficiency.

In contrast to the previous attempt of Sotin et al [22] of extending strategy iteration to logico-numerical programs, which relies on widening operators to converge, our method enables the use of mathematical programming and hence, it indeed computes the *best logico-numerical invariant* w.r.t. the chosen abstract domain.

The effectiveness of our method depends on two factors:

(1) The choice of the templates: in our experiments we used mainly octagonal constraints, but we could have used methods (*e.g.* [10]) for inferring template constraints.
(2) The considered CFG (either of the imperative program, or the one obtained by partitioning in the case of data-flow programs) which determines the abstract domain: the partitioning method by "numerical modes" turned out to be surprisingly effective: compared to the solution based on an enumeration of the reachable Boolean states, the obtained CFGs were 5 times smaller on average, still the precision loss was negligible, *i.e.* almost zero, and we gained at least one order of magnitude w.r.t. efficiency.

Furthermore, this paper delivers the first experimental results of applying *numerical* max-strategy iteration to larger programs: on the one hand max-strategy iteration is guaranteed to compute more precise invariants than standard techniques in the same domain, on the other hand our implementation is not (yet) able to compete with standard techniques w.r.t. efficiency.

Perspectives. Our algorithm is quite generic w.r.t. the numerical analysis method and logico-numerical abstract domain. Though, in order to tackle efficiency issues evoked above, it would be interesting to design a more integrated logico-numerical max-strategy solver. This would enable to share more information between subsequent calls to the max-strategy iteration, *e.g.* to avoid retesting of strategies that will definitely not lead to an improvement. Beyond that, we could more extensively use SMT-solvers. For instance, checking whether a strategy is an improvement is currently done after having constructed the numerical equation system; it would be beneficial to find the improving strategies already on the logico-numerical level.

Moreover, we plan to apply our method to the analysis of logico-numerical hybrid automata [29] by extending the hybrid max-strategy iteration method of Dang and Gawlitza [30, 27].

Acknowledgements. We thank Thomas M. Gawlitza, Bertrand Jeannet, Philipp Rümmer and the anonymous reviewers for their valuable remarks on the draft of this paper.

References

1. Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J.A.: LUSTRE: a declarative language for real-time programming. In: Principles of Programming Languages, ACM (1987) 178–188
2. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Principles of Programming Languages. (1977) 238–252
3. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Principles of Programming Languages, ACM (1978) 84–97
4. Costan, A., Gaubert, S., Goubault, E., Martel, M., Putot, S.: A policy iteration algorithm for computing fixed points in static analysis of programs. In: Computer-Aided Verification. Volume 3576 of LNCS., Springer (2005) 462–475

5. Gaubert, S., Goubault, E., Taly, A., Zennou, S.: Static analysis by policy iteration on relational domains. In: European Symposium on Programming. Volume 4421 of LNCS., Springer (2007) 237–252
6. Adjé, A., Gaubert, S., Goubault, E.: Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In: European Symposium on Programming. Volume 6012 of LNCS., Springer (2010) 23–42
7. Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: European Symposium on Programming. Volume 4421 of LNCS., Springer (2007) 300–315
8. Gawlitza, T., Seidl, H.: Precise relational invariants through strategy iteration. In: Computer Science Logic. Volume 4646 of LNCS., Springer (2007) 23–40
9. Gawlitza, T., Seidl, H., Adjé, A., Gaubert, S., Goubault, E.: Abstract interpretation meets convex optimization. Journal of Symbolic Computation **47** (2012) 1416–1446
10. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Verification, Model Checking, and Abstract Interpretation. Volume 3385 of LNCS., Springer (2005) 25–41
11. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming, Dunod (1976) 106–130
12. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Programs as Data Objects. Volume 2053 of LNCS., Springer (2001) 155–172
13. Miné, A.: The octagon abstract domain. In: Working Conference on Reverse Engineering, IEEE (2001) 310–319
14. Bultan, T., Gerber, R., Pugh, W.: Symbolic model checking of infinite state systems using Presburger arithmetic. In: Computer-Aided Verification. Volume 1254 of LNCS., Springer (1997) 400–411
15. Jeannet, B., Halbwachs, N., Raymond, P.: Dynamic partitioning in analyses of numerical properties. In: Static Analysis Symposium. Volume 1694 of LNCS., Springer (1999) 39–50
16. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Programming Language Design and Implementation, ACM (2003) 196–207
17. Hagen, G., Tinelli, C.: Scaling up the formal verification of lustre programs with smt-based techniques. In: Formal Methods in Computer-Aided Design, IEEE (2008) 1–9
18. Gawlitza, T., Seidl, H.: Precise interval analysis vs. parity games. In: Formal Methods. Volume 5014 of LNCS., Springer (2008) 342–357
19. Gawlitza, T.M., Seidl, H.: Computing relaxed abstract semantics w.r.t. quadratic zones precisely. In: Static Analysis Symposium. Volume 6337 of LNCS., Springer (2010) 271–286
20. Gawlitza, T.M., Seidl, H.: Solving systems of rational equations through strategy iteration. Transactions on Programming Languages and Systems **33** (2011) 11
21. Jeannet, B.: BDDAPRON: A logico-numerical abstract domain library. `http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/` (2009)
22. Sotin, P., Jeannet, B., Védrine, F., Goubault, E.: Policy iteration within logico-numerical abstract domains. In: Automated Technology for Verification and Analysis. Volume 6996 of LNCS., Springer (2011) 290–305
23. Gonnord, L., Halbwachs, N.: Combining widening and acceleration in linear relation analysis. In: Static Analysis Symposium. Volume 4134 of LNCS., Springer (2006) 144–160

24. Schrammel, P., Jeannet, B.: Applying abstract acceleration to (co-)reachability analysis of reactive programs. Journal of Symbolic Computation **47** (2012) 1512–1532
25. Schrammel, P., Jeannet, B.: Logico-numerical abstract acceleration and application to the verification of data-flow programs. In: Static Analysis Symposium. Volume 6887 of LNCS., Springer (2011) 233–248
26. Gopan, D., Reps, T.W.: Guided static analysis. In: Static Analysis Symposium. Volume 4634 of LNCS., Springer (2007) 349–365
27. Dang, T., Gawlitza, T.M.: Discretizing affine hybrid automata with uncertainty. In: Automated Technology for Verification and Analysis. LNCS, Springer (2011) 473–481
28. Jeannet, B., Miné, A.: APRON: A library of numerical abstract domains for static analysis. In: Computer-Aided Verification. Volume 5643 of LNCS., Springer (2009) 661–667
29. Schrammel, P., Jeannet, B.: From hybrid data-flow languages to hybrid automata: A complete translation. In: Hybrid Systems: Computation and Control, ACM (2012) 167–176
30. Dang, T., Gawlitza, T.M.: Template-based unbouded time verification of affine hybrid automata. In: Asian Symposium on Programming Languages and Systems. LNCS, Springer (2011) 34–49