Template Polyhedra with a Twist

Sriram Sankaranarayanan[a] and Mohamed Amin Ben Sassi[b]

a. University of Colorado, Boulder, CO, USA,
b. Mediterranean Institute of Technology, Tunis, Tunisia.
firstname.lastname@colorado.edu

Abstract. In this paper, we draw upon connections between bilinear programming and the process of computing (post) fixed points in abstract interpretation. It is well-known that the data flow constraints for numerical domains are expressed in terms of bilinear constraints. Algorithms such as policy and strategy iteration have been proposed for the special case of bilinear constraints that arise from template numerical domains. In particular, policy iteration improves upon a known post-fixed point by alternating between solving for an improved post-fixed point against finding certificates that are used to prove the new fixed point.
In this paper, we draw upon these connections to formulate a policy iteration scheme that changes the template on the fly in order to prove a target reachability property of interest. We show how the change to the template naturally fits inside a policy iteration scheme, and thus propose a policy iteration scheme that updates the template matrices associated with each program location. We demonstrate that the approach is effective over a set of benchmark instances, wherein starting from a simple predefined choice of templates, the approach is able to infer appropriate template directions to prove a property of interest. We also note some key theoretical questions regarding the convergence of the policy iteration scheme with template updates, that remain open at this time.

1 Introduction

In this paper, we study policy iterations for computing inductive invariants of programs using template abstract domains, and present an approach that modifies templates on the fly. In a template abstract domain, we fix the left-hand side expressions of the invariant properties of interest and use abstract interpretation to compute valid right-hand side constants so that the resulting inequalities form an inductive invariant. As such, template domains such as intervals [15], octagons [30, 31], octahedra [11], pentagons [28], linear templates [36], and quadratic templates [2] have been well studied as effective numerical domains for proving safety of runtime assertions in software [24, 6, 5, 18, 29, 39]. Template domains have given rise to specialized approaches such as policy iteration [13, 20] for improving post-fixed points, and strategy iteration for computing the least fixed point [22, 21].

Policy iteration starts from a known post-fixed point, and alternates between finding a "policy" that certifies the current solution versus finding the best solution under the current "policy". This approach was originally proposed by

Costan et al. for the interval domain [13] and generalized to arbitrary templates subsequently [20]. Extensions have been proposed for quadratic templates [2]. On the other hand, strategy iteration approach works in a bottom up fashion starting from the bottom of the lattice and exploiting the "monotonicity" property in the dataflow equations for the template domain [21]. Specifically, the system of data flow equations are linearized around the current solution, and a fixed point of the linearized system is obtained as the next solution.

Our approach here exploits a connection between policy iteration approach and classic bilinear optimization problems. In fact, policy iteration is a variant of the popular alternating coordinate descent that has been used widely in the control systems and optimization communities [23]. Using this connection, we notice that the alternation between solutions and multipliers can be extended to update the templates on the fly, as the iteration proceeds. Significantly, the update to the templates can be made *property-directed* in a simple manner. By combining these observations, we arrive at a policy iteration approach that can start from initial, user-defined templates and update them on the fly. However policy iteration is not guaranteed to converge to a globally optimal solution, which would correspond to the least fixed point solution in the abstract domain. In practice, the technique gets stuck in a local minimum, yielding a suboptimal solution. A result by Helton and Merino on more general biconvex programs suggests that the alternating minimization almost never converges to a local minimum (technically a solution satisfying the KKT conditions) [27]. Adjé et al. demonstrate an approach that computes an optimal solution for systems which are nonexpansive [3]. However, the general applicability of this result is unclear. To circumvent this issue, we work in a property directed fashion, wherein the goal of the approach is to find a suitably strong invariant that is sufficient to prove a property of interest. Such a property can be established with a solution that is not necessarily a least fixed point.

An implementation of the approach and evaluation over a set of small benchmarks shows that the approach of updating the policies on the fly is an effective solution to inferring appropriate templates in a property directed manner.

1.1 Related Work

Colón et al. were the first to discover the connection between linear invariant synthesis problems and bilinear constraints through the use of Farkas lemma in linear programming [12]. These constraints were solved using specialized quantifier elimination techniques, but restricted to small problems [40]. Sankaranarayanan et al. explored the use of heuristic approaches to solve bilinear constraints [35]. These approaches were generalized by Cousot, as instances of *Lagrangian relaxations* [14]. Additionally, Cousot's work uses numerical optimization tools to prove total correctness properties of programs. His approach relies on formulating the constraints as Linear or Bilinear Matrix inequalities (LMI/BMI). However, the use of numerical solvers requires rigorous symbolic verification of the results. Recent experiences reveal surprising pitfalls, including erroneous invariants obtained, even when the error tolerances are quite low [33, 38]. In fact,

one of the advantages of policy iterations lies in the use of exact arithmetic LP solvers to avoid floating point errors. Other approaches to solving the resulting constraints have restricted the multiplier variables to finite domains, enabling linear arithmetic solvers [26].

Template polyhedra and their generalization to support functions have proven useful for constructing reachable sets of linear and nonlinear hybrid systems [34, 25, 19, 8]. The problem of inferring template directions has also been studied in this context. Many heuristics were proposed by Sankaranarayanan et al. in their paper on linear templates, including the use of expressions found in programs, "increasing"/ "decreasing" expressions, and preconditions of already added template expressions [36]. However, none of these are guaranteed to be relevant to the property. Adjé et al. use the idea of Lyapunov-like functions to effectively infer templates that are shown to be effective in proving bounds on variables [1].

The idea of updating templates on the fly was previously proposed by Ben Sassi et al. for analyzing the largest invariant region of a dynamical system [37]. The approach searches for a polytope whose facets are transverse to the flow, failing which, the facet directions are adjusted and tested again. The approach to adjusting facets is based on a local sensitivity analysis to obtain the invariant region around an equilibrium (which facilitates basin of attraction analysis for dynamical systems). Compared to the present work, the differences include the treatment of multiple program locations and transitions, the use of policy iteration, and a property-directed approach that seeks to prove a property rather than find a largest invariant region.

Abraham et al.. propose effective heuristics to guide the choice of directions for constructing reachable sets of linear hybrid systems [9]. Recently, Bogomolov et al. propose a counter-example guided approach for inferring facets of template polyhedra for hybrid systems reachability analysis [7]. The key differences include: (a) we are interested in computing a single polyhedron per location whereas flowpipe construction approaches use a disjunction of polytopes, and (b) we seek to compute time-unbounded invariants, whereas flowpipes are typically time bounded. Another interesting approach by Amato et al. uses principal component analysis (PCA) over concrete states reached by execution traces to design templates [4].

2 Motivating Example

Consider a simple system over two real-valued variables $(x_1, x_2) \in \mathbb{R}^2$, initialized to $(x_1, x_2) \in [-1, 1] \times [-1, 1]$. The system executes the following action

$$\textbf{if } (x_1, x_2) \in [-8, 8]^2 \textbf{ then } \left[\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} := M \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right] \textbf{ else } \left[\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right]$$

wherein $M = \begin{pmatrix} 0.92 & 0.18 \\ 0.18 & 0.92 \end{pmatrix}$. Our goal is to prove that the set $U : \{(x_1, x_2) \mid x_2 - x_1 \geq 2.1\}$ is never reached by any execution of the system. In order to prove the

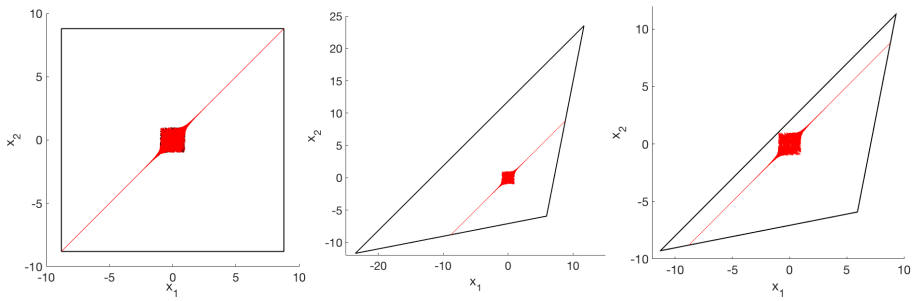

Fig. 1. Invariants synthesized for the three steps of the policy iteration with property directed template modification. The simulation traces are shown in red. Note: each figure is drawn to a different scale.

property using a template domain, the user specifies a template matrix [36, 20]:

$$T : \begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}, \quad \begin{array}{l} (* \ 1x_1 + 0x_2 \ *) \\ (* \ -1x_1 + 0x_2 \ *) \\ (* \ 0x_1 + 1x_2 \ *) \\ (* \ 0x_1 - 1x_2 \ *) \end{array}$$

wherein the rows represent the expressions $x_1, -x_1, x_2, -x_2$, respectively. The template domain analysis seeks to find an invariant of the form $T\boldsymbol{x} \leq \boldsymbol{c}$ by discovering the unknown constants $\boldsymbol{c}$ that represent the RHS of the template. For the example shown above, the best possible invariant is obtained as $\boldsymbol{c}$: $\begin{pmatrix} 8.8 & 8.8 & 8.8 & 8.8 \end{pmatrix}^T$, yielding the range $[-8.8, 8.8] \times [-8.8, 8.8]$ for (x_1, x_2). In fact, given our instance on using the template T, this is the best invariant possible (see Fig. 1(a) to verify this).

For this example, the policy iterative scheme presented in this paper is successful in choosing a new template:

$$\hat{T} : \begin{pmatrix} -1 & 1 \\ 1 & -0.1957 \\ 0.1957 & -1 \\ -1 & 1 \end{pmatrix}, \quad \begin{array}{l} (* \ -x_1 + x_2 \ *) \\ (* \ x_1 - 0.1957x_2 \ *) \\ (* \ 0.1957x_1 - x_2 \ *) \\ (* \ -x_1 + x_2 \ *) \end{array}$$

Along with this policy, we compute a tighter invariant shown in Fig. 1**(c)**, that establishes the invariant $x_2 - x_1 \leq 2$, and thus proving U unreachable. We note that (a) the choice of templates is directed by the property, and (b) unlike the original policy iteration approach proposed by Gaubert et al. [20], this approach does not guarantee that the iterates are strictly descending. In fact, the iterates obtained are often incomparable.

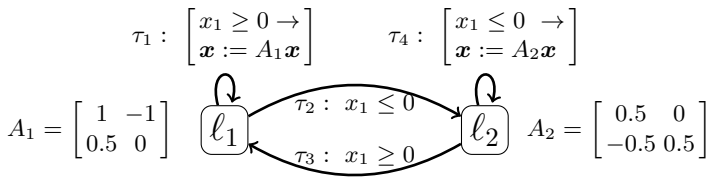

Fig. 2. Example of a transition system with two variables x_1, x_2, two locations ℓ_1, ℓ_2 and four transitions shown as arrows.

3 Preliminaries

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{R}_+ : \mathbb{R} \cup \pm\infty$ denote the extended reals with infinity. We first define the transition system model used throughout this paper. Let X be a set of real-valued variables and $\Pi[X]$ represent a language of assertions over these variables, drawn from a suitable fragment of the first order logic over the reals. For any assertion $\varphi \in \Pi[X]$, we denote its corresponding set of models by $[\![\varphi]\!]$. For convenience, the set of variables X are arranged as a column vector, written as $\boldsymbol{x}$.

Definition 1 (Transition System). *A (numerical)* transition system *is a tuple* $\langle X, \mathcal{L}, \mathcal{T}, \mathcal{I}, \ell_0, \Theta \rangle$*, wherein*

1. $X : \{x_1, \ldots, x_n\}$ *represents a set of* real-valued *program variables,*
2. $\mathcal{L} : \{\ell_1, \ldots, \ell_m\}$ *represents a set of program locations,*
3. $\mathcal{T} : \{\tau_1, \ldots, \tau_k\}$ *represents a set of transitions, wherein each transition* τ_i *is a tuple* $\langle \ell_i, m_i, \psi_i, g_i \rangle$*, wherein*
   (a) $\ell_i, m_i \in \mathcal{L}$ *are the pre and the post locations, respectively.*
   (b) $\psi_i \in \Pi[X]$*, an assertion over* X*, represents the guard of the transition.*
   (c) $g_i : \mathbb{R}^n \to \mathbb{R}^n$*, an update function, represents the (simultaneous) assignment:* $(x_1, \ldots, x_n) := g_i(x_1, \ldots, x_n)$.
4. ℓ_0 *is the initial location, and* $\Theta \in \Pi[X]$ *is an assertion over* X *representing the initial valuations of the program variables.*

A state of the transition system is a tuple $\langle \ell, \boldsymbol{x} \rangle$ wherein $\ell \in L$ is the control location and $\boldsymbol{x} \in \mathbb{R}^n$ represents a set of valuations for the program variable. Given a transition system, its executions are a finite/infinite sequence of states:

$$(\ell_0, \boldsymbol{x}_0) \xrightarrow{\tau_1} (\ell_1, \boldsymbol{x}_1) \xrightarrow{\tau_2} \cdots \xrightarrow{\tau_i} (\ell_i, \boldsymbol{x}_i) \cdots ,$$

such that: (a) ℓ_0 is the initial location and $\boldsymbol{x}_0 \in [\![\Theta]\!]$; (b) ℓ_{i-1}, ℓ_i are the pre/post locations (respectively) of the transition τ_i for all $i \geq 1$; (c) $\boldsymbol{x}_{i-1} \in [\![\psi_i]\!]$ for all $i \geq 1$ wherein ψ_i is the guard corresponding to the transition τ_i; and (d) $\boldsymbol{x}_i = g_i(\boldsymbol{x}_{i-1})$ for all $i \geq 1$, wherein g_i is the update function for τ_i.

Example 1. Figure 2 shows an example of a transition system with $X : \{x_1, x_2\}$, $\mathcal{L} : \{\ell_1, \ell_2\}$ and $\mathcal{T} : \{\tau_1, \tau_2, \tau_3, \tau_4\}$. The guards and updates of the transitions

are as shown in Fig. 2. The identity update $\boldsymbol{x} := \boldsymbol{x}$ is not shown, however. The initial location is ℓ_1 and the initial condition on $\boldsymbol{x}$ is $(x_1, x_2) \in [0.5, 1.5] \times [0.5, 1]$.

A state $(\ell, \boldsymbol{x})$ is reachable if there is an execution that reaches the state.

For this paper, we study *linear transition systems*. A linear expression is of the form $e : \boldsymbol{a}^T\boldsymbol{x}$ for vector $\boldsymbol{a} \in \mathbb{R}^n$. A linear inequality is of the form $\boldsymbol{a}^T\boldsymbol{x} \leq b$ and a linear assertion is a finite conjunction of linear inequalities $(\boldsymbol{a}_1^T\boldsymbol{x} \leq b_1 \ \wedge \ \cdots \ \wedge \ \boldsymbol{a}_k^T\boldsymbol{x} \leq b_k)$ conveniently written in matrix form as $A\boldsymbol{x} \leq \boldsymbol{b}$.

Definition 2 (Linear Transition Systems). *A linear transition system (LTS) is a transition system with the following restrictions:*

1. *The initial conditions and transition guards are all linear assertions over X*
2. *The update function for each transition is an affine function: $g_i(\boldsymbol{x}) : U_i\boldsymbol{x}+\boldsymbol{v_i}$.*

Throughout this paper, we will tackle linear transition systems. An error specification is written as $\langle \ell, \psi \rangle$ for a location ℓ and a linear assertion ψ. The goal is to prove that no reachable state for location ℓ satisfies ψ. I.e, all reachable states $\boldsymbol{x}$ at location ℓ satisfy $\boldsymbol{x} \notin [\![\psi]\!]$. To prove a given specification, we use an *inductive invariant*.

Definition 3 (Inductive Invariant Map). *An inductive invariant map $\eta : \mathcal{L} \to \Pi[X]$ maps each location $\ell \in \mathcal{L}$ to an assertion $\eta(\ell)$ such that the following conditions hold:*

- Initial Condition: *At the initial location ℓ_0, the entailment $\Theta \models \eta(\ell_0)$ holds.*
- Consecution Condition: *For each transition $\tau : \langle \ell_1, \ell_2, \psi_i, g_i \rangle$, the following consecution condition holds:*

$$\eta(\ell_1) \ \wedge \ \psi_i \ \wedge \ \boldsymbol{x}' = g_i(\boldsymbol{x}) \ \models \ \eta(\ell_2)[\boldsymbol{x}'] \, .$$

The condition states that starting from any state $\boldsymbol{x} \in [\![\eta(\ell_1)]\!]$, a single step of the transition τ, if enabled, yields a state $\boldsymbol{x}' \in [\![\eta(\ell_2)]\!]$.

Let η be an inductive assertion map and $\langle \ell, \psi \rangle$ be an error specification.

Theorem 1. *If the conjunction $\eta(\ell) \ \wedge \ \psi$ is unsatisfiable, then for every reachable state $(\ell, \boldsymbol{x})$, it follows that $\boldsymbol{x} \notin [\![\psi]\!]$.*

The problem therefore consists of finding inductive assertion maps that can prove a given error specification.

Abstract interpretation provides a framework for systematically computing inductive assertions using a pre-specified lattice of assertions called an *abstract domain* [17, 16]. The key insight lies in characterizing inductive assertion maps as post-fixed points of a monotone operator over sets of states.

An abstract domain is defined by a lattice $\mathcal{A} : \langle A, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ with inclusion $\sqsubseteq$, join operator $\sqcup$, meet operator $\sqcap$, a bottom element $\bot$ and top element $\top$. Each element $a \in A$ represents a correponding set of states (technically, an element of the concrete domain) through the concretization function $\gamma(a)$, and

likewise, for every set of states S (element of the concrete domain), we define a corresponding abstraction $\alpha(S)$.

The theory of abstract interpretation defines a set of operations including the abstract post condition $b : \widehat{post}(a, \tau)$, that given $a \in A$ and transition τ results in an abstract element $b \in A$ such that $\gamma(b)$ over approximates the reachable states obtained by starting from some state in $\gamma(a)$ and applying the transition τ. Other useful abstract domain operations include $\sqcup$ for merging sets of states, $\sqcap$ for handling conditional branches, $\bot$ for the empty set of states, $\top$ for the universal set of states, $\sqsubseteq$ to test containment between abstract elements and a special operation called widening ∇ that enforces termination. We will omit a detailed presentation of abstract interpretation from this paper. The interested reader may obtain these from standard references [17, 16, 32].

3.1 Template Domains

The rest of this paper will focus on the abstract domain of template polyhedra [36]. Let $\mathcal{S} : \langle \mathcal{L}, X, \mathcal{T}, \ell_0, \Theta \rangle$ be a linear transition system. Let $\boldsymbol{x}$ represent the system variables in X as a vector and $n = |X|$.

A template associates each location $\ell \in \mathcal{L}$ with a $m_\ell \times n$ matrix T_ℓ. We drop the subscript ℓ from the template matrix if the location ℓ is clear from the context. A $m \times n$ template T defines a lattice $\mathcal{A}(T)$:

$$\mathcal{A}(T) : \{\boldsymbol{c} \in \mathbb{R}^m_+\}, \text{ wherein, } \gamma(\boldsymbol{c}) : T\boldsymbol{x} \leq \boldsymbol{c}\,.$$

In other words, each element of the template abstract domain is a possible valuation $\boldsymbol{c}$ to the RHS of inequalities $T\boldsymbol{x} \leq \boldsymbol{c}$. Note that the entries in $\boldsymbol{c}$ can include $\pm\infty$. Naturally, we define the linear inequality $e \leq \infty$ to be synonymous with *true* and $e \leq -\infty$ is synonymous with *false*.

Given an assertion φ over $\boldsymbol{x}$, its abstraction $\boldsymbol{c} : \alpha(\varphi)$ is computed as a vector whose i^{th} entry $\boldsymbol{c}_i$ is the solution to the optimization problem:

$$\boldsymbol{c}_i : \max\ T_i\boldsymbol{x} \text{ s.t. } \varphi(\boldsymbol{x})\,.$$

Since the abstraction is often computed for linear assertions φ, this is a linear programming (LP) problem.

For each template element, its *canonical representative* $\mathsf{can}(\boldsymbol{c})$ is defined as the instantiation $\boldsymbol{d}$, whose i^{th} entry $\boldsymbol{d}_i$ is the solution to the following LP:

$$\boldsymbol{d}_i : \max\ T_i\boldsymbol{x} \text{ s.t. } T\boldsymbol{x} \leq \boldsymbol{c}\,.$$

Note that the solution to an unbounded problem is taken to be $+\infty$ and an infeasible problem to be $-\infty$. Note that the template polyhedron defined by $T\boldsymbol{x} \leq \boldsymbol{c}$ is identical to the polyhedron $T\boldsymbol{x} \leq \mathsf{can}(\boldsymbol{c})$. A template element $\boldsymbol{c}$ is *canonical* in $\mathcal{A}(T)$ if and only $\boldsymbol{c} = \mathsf{can}(\boldsymbol{c})$.

The inclusion operator $\sqsubseteq$ in $\mathcal{A}(T)$ is defined as

$$\boldsymbol{c}_1 \sqsubseteq \boldsymbol{c}_2 \text{ iff } \mathsf{can}(\boldsymbol{c}_1) \leq \mathsf{can}(\boldsymbol{c}_2)\,,$$

wherein $\leq$ operation over vectors compares elements entrywise. The join operator $\boldsymbol{c}_1 \sqcup \boldsymbol{c}_2$ is simply the entrywise maximum $\max(\boldsymbol{c}_1, \boldsymbol{c}_2)$. Likewise, the meet operator is the canonical entry wise minimum.

Let T_ℓ be the template associated with location ℓ and T_m with location m. The abstract post with respect to a transition $\langle \ell, m, \varphi : A\boldsymbol{x} \leq \boldsymbol{b}, g : U\boldsymbol{x} + \boldsymbol{v} \rangle$ is an operator $\widehat{post} : \mathcal{A}(T_\ell) \times \mathcal{T} \ \rightarrow \ \mathcal{A}(T_m)$. Given $\boldsymbol{c} \in \mathcal{A}(T_\ell)$, the result $\boldsymbol{d} : \widehat{post}(\boldsymbol{c}, \tau)$ is a vector wherein $\boldsymbol{d}_i$ is given as the solution to the following LP:

$$\boldsymbol{d}_i : \begin{pmatrix} \max T_{m,i}\boldsymbol{x} & \\ \text{s.t. } T_\ell \boldsymbol{y} & \leq \boldsymbol{c} \\ A\boldsymbol{y} & \leq \boldsymbol{b} \\ \boldsymbol{x} & = U\boldsymbol{y} + \boldsymbol{v} \end{pmatrix}$$

Widening and narrowing operators for the template domain are defined by extensions of the standard interval widening operator [36].

The template domain is a convenient numerical abstract domain that uses linear programming solvers as a primitive for implementing the domain operations. However, a common critique of the template approach is that it requires users to specify the template T. In practice, users default to popular choices such as *intervals*, *octagons* and *pentagons* which avoid repeated calls to LP solvers by using special properties of the constraints in these templates. We proceed by assuming that an initial template has been specified for each location using one of the schemes outlined above. Our approach can change this template as part of the solution scheme.

4 Bilinear Constraints and Policy Iteration

In this section, we consider the data flow equations for template abstract domain, connecting them to a class of nonconvex optimization problems called *bilinear optimization problem* (BOP). We present the *policy iteration* approach, proposed by Gaubert et al. as a technique for solving such bilinear inequalities that alternates between solving linear programs [20]. Once again we fix a linear transition system $\mathcal{S}$ and assume for simplicity that each location ℓ is labeled with the same $m \times n$ matrix T. The approach can be easily extended to the case where the template matrices differ between locations.

We will make use of Farkas' lemma, a standard result in linear programming. Let $\varphi : \ A\boldsymbol{x} \leq \boldsymbol{b}$ be a linear assertion with $m \times n$ matrix A and $m \times 1$ vector $\boldsymbol{b}$, $\psi : \ \boldsymbol{c}^T\boldsymbol{x} \leq d$ be a given linear inequality.

Theorem 2 (Farkas Lemma). *If φ is satisfiable, then $\varphi \models \psi$ iff there exists nonnegative multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that*

$$A^T\boldsymbol{\lambda} = \boldsymbol{c} \ \wedge \ \boldsymbol{b}^T\boldsymbol{\lambda} \leq d \ \wedge \ \boldsymbol{\lambda} \geq 0\,.$$

Furthermore, φ is unsatisfiable if and only if there exists multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that

$$A^T\boldsymbol{\lambda} = \boldsymbol{0} \ \wedge \ \boldsymbol{b}^T\boldsymbol{\lambda} \leq -1 \ \wedge \ \boldsymbol{\lambda} \geq 0\,.$$

The constraints can be seen as encoding the entailment $\varphi \models \boldsymbol{0}^T\boldsymbol{x} \leq -1$.

Note that Farkas lemma handles the entailment of a single linear inequality. However, for a polyhedron $C\boldsymbol{x} \leq \boldsymbol{d}$, we may encode the entailment $A\boldsymbol{x} \leq \boldsymbol{b} \models C\boldsymbol{x} \leq \boldsymbol{d}$ as a series of single inequality entailments: $A\boldsymbol{x} \leq \boldsymbol{b} \models C_j\boldsymbol{x} \leq \boldsymbol{d}_j$ for each row j of $C, \boldsymbol{d}$. The resulting constraints can be collectively written as:

$$A^T \Lambda = C,\ \Lambda^T \boldsymbol{b} \leq \boldsymbol{d},\ \Lambda \geq 0\,.$$

All equalities and inequalities between matrices are interpreted entrywise. Here Λ is a matrix with as many rows as A and as many columns as the number of rows in C. The j^{th} column of Λ contains the multipliers corresponding to the inequality $C_j\boldsymbol{x} \leq \boldsymbol{d}_j$. This notation will be used throughout the rest of the paper.

Using Farkas' lemma, we may now derive a system of constraints corresponding to the *data flow equations* for the template domain. Let T be a $m \times n$ template matrix. We associate each location ℓ with an unknown vector $\boldsymbol{c}(\ell) \in \mathcal{A}(T)$ such that the assertion map $\eta(\ell) :\ T\boldsymbol{x} \leq \boldsymbol{c}(\ell)$ is inductive.

We wish to encode the constraints for initiation:

$$\Theta \models T\boldsymbol{x} \leq \boldsymbol{c}(\ell_0)\,, \tag{1}$$

and for each transition $\tau : \langle \ell, m, \varphi, g \rangle$, we wish to model consecution:

$$T\boldsymbol{x} \leq \boldsymbol{c}(\ell)\ \wedge\ \varphi\ \wedge\ \boldsymbol{x}' = g(\boldsymbol{x})\ \models\ T\boldsymbol{x}' \leq \boldsymbol{c}(m)\,. \tag{2}$$

Initiation: Let $\Theta : A_0\boldsymbol{x} \leq \boldsymbol{b}_0$ be the assertion for the initial condition. Using Farkas' lemma for the entailment in Eq. (1), we obtain the condition:

$$A_0^T \Lambda_0 = T\ \wedge\ \Lambda_0^T \boldsymbol{b}_0 \leq \boldsymbol{c}(\ell_0)\ \wedge\ \Lambda_0 \geq 0\,. \tag{3}$$

Here Λ_0 is a $k \times m$ matrix wherein k is the number of rows in A_0 and m is the number of rows in T. We write $\Lambda_0 \geq 0$ to indicate that all entries in Λ_0 are non-negative.

Consecution: Let τ be a transition with guard $A_\tau\boldsymbol{x} \leq \boldsymbol{b}_\tau$ and update $g(\boldsymbol{x}) : U_\tau\boldsymbol{x} + \boldsymbol{v}_\tau$. The consecution condition in Eq. (2) can be rewritten through substitution of $\boldsymbol{x}'$ and arranged as follows:

$$\begin{array}{rl|l} \Lambda_\tau \rightarrow & T\boldsymbol{x} & \leq \boldsymbol{c}(\ell) \\ \Gamma_\tau \rightarrow & A_\tau\boldsymbol{x} & \leq \boldsymbol{b}_\tau \\ \hline \models & TU_\tau\boldsymbol{x} & \leq \boldsymbol{c}(m) - T\boldsymbol{v}_\tau \end{array}$$

The notation above shows the constraints and the associated dual multipliers with each block of constraints. Furthermore, we have substituted $\boldsymbol{x}' = U_\tau\boldsymbol{x} + \boldsymbol{v}_\tau$. This is dualized using Farkas' lemma to yield the following constraints:

$$\begin{gathered} T^T\Lambda_\tau + A_\tau^T\Gamma_\tau = TU_\tau \\ \Lambda_\tau^T\boldsymbol{c}(\ell) + \Gamma_\tau^T\boldsymbol{b}_\tau \leq \boldsymbol{c}(m) - T\boldsymbol{v}_\tau \\ \Lambda_\tau, \Gamma_\tau \geq 0 \end{gathered} \tag{4}$$

TemplateVars : $\boldsymbol{c}(\ell),\ \ell \in \mathcal{L}$		
BilinearMults : $\Lambda_\tau,\ \tau \in \mathcal{T}$		
LinearMults : $\Lambda_0, \Gamma_\tau,\ \tau \in \mathcal{T}$		
Constraints :	$A_0^T \Lambda_0 = T_{\ell_0}$	(* Initiation *)
	$\Lambda_0^T \boldsymbol{b}_0 \leq \boldsymbol{c}(\ell_0)$	
	$T_l^T \Lambda_\tau + A_\tau^T \Gamma_\tau \ = \ T_m U_\tau$	(* Consecution $\tau : \langle l, m, \varphi, g \rangle$ *)
	$\Lambda_\tau^T \boldsymbol{c}(\ell) + \Gamma_\tau^T \boldsymbol{b}_\tau \ \leq \ \boldsymbol{c}(m) - T_m \boldsymbol{v}_\tau$	
	$\Lambda_0, \Lambda_\tau, \Gamma_\tau \geq 0$	(* Nonnegative multipliers *)

Fig. 3. Bilinear system of constraints at a glance. The constraints are generalized to allow for possibly different templates T_ℓ at each location.

Note that Eq. (3) for the initiation yields a system of linear constraints involving $\boldsymbol{c}(\ell_0)$ and unknown multipliers in Λ_0. However, the consecution constraints in Eq. (4) for each transition τ involve the product $\Lambda_\tau^T \boldsymbol{c}(\ell)$ both of which are unknown. This makes the constraints for consecution fall into a special class called *bilinear constraints*. I.e., for a fixed Λ_τ these constraints are linear in the remaining variables $\boldsymbol{c}(\ell), \Gamma_\tau$. Similarly, for fixed values of $\boldsymbol{c}(\ell)$, these constraints are linear in the variables $\Lambda_\tau, \Gamma_\tau$. Figure 3 summarizes the constraints obtained at a glance.

Connection with Min-Policies: The original "min-policy" approach of Costan et al. [13] considers data flow equations of the form:

$$\boldsymbol{c} \geq \min(\boldsymbol{a}_{i,1}^T \boldsymbol{c}, \ldots, \boldsymbol{a}_{i,k}^T \boldsymbol{c})\,,\ i = 1, \ldots, M,\ k = 1, \ldots, N\,. \tag{5}$$

We will demonstrate that the equations shown in Figure 3 can be equivalently expressed in this form. For simplicity, we consider the case for a single location ℓ with template T and unknown template RHS variables $\boldsymbol{c}$. All transitions are assumed to be self-loops around this location. From eq. (4), a given solution $\boldsymbol{c}$ satisfies the consecution for transition τ iff there exist $\Lambda_\tau, \Gamma_\tau$ such that

$$\boldsymbol{c} \ \geq \ \Lambda_\tau^T \boldsymbol{c} + \Gamma_\tau^T \boldsymbol{b}_\tau + T \boldsymbol{v}_\tau \tag{6}$$

$$T^T \Lambda_\tau + A_\tau^T \Gamma_\tau \ = \ T U_\tau \tag{7}$$

$$\Lambda_\tau, \Gamma_\tau \ \geq 0 \tag{8}$$

Let us define a polyhedron $P(\Lambda_\tau, \Gamma_\tau)$ defined by collecting the constraints in lines (7), (8) above. We may rewrite the constraints equivalently as:

$$\boldsymbol{c} \ \geq \ \min_{(\Lambda_\tau, \Gamma_\tau) \in P} \left(\Lambda_\tau^T \boldsymbol{c} + \Gamma_\tau^T \boldsymbol{b}_\tau + T \boldsymbol{v}_\tau \right) \tag{9}$$

Note that P is a polyhedron. Let us assume that it is defined by N vertices:

$$(\Lambda_1, \Gamma_1)\,,\ \ldots,\ (\Lambda_N, \Gamma_N)\,.$$

The min in eq. (9) can be equivalently written as a minimization over the finite set of vertices of P:

$$\boldsymbol{c} \;\geq\; \min_{j=1}^{N} \left(\Lambda_j^T \boldsymbol{c} + \Gamma_j^T \boldsymbol{b}_\tau + T \boldsymbol{v}_\tau \right) \tag{10}$$

We note that this form arises from the specific structure of the data flow equations for the template abstract domain. In particular, not all bilinear constraints satisfy this property.

4.1 Policy Iteration

We now describe policy iteration as an alternation between solving for unknown $\boldsymbol{c}(\ell)$ for each $\ell \in \mathcal{L}$ and solving for the unknown bilinear multipliers Λ_τ. Policy iteration starts from a known sound solution $\boldsymbol{c}^0(\ell)$ and successively improves the solution to obtain better solutions (smaller in the lattice) until no further improvements can be obtained. The initial solution may be obtained by using Kleene iteration with widening. For simplicity, we will assume that $\boldsymbol{c}^0(\ell) \neq \bot$, for each $\ell \in \mathcal{L}$. If this were the case, then the location ℓ is unreachable, and can be removed from the system.

The overall scheme alternates between (I) *solving for the unknown multipliers* $\Lambda_\tau, \Gamma_\tau, \Lambda_0$ given a fixed value of $\boldsymbol{c}$, and (II) *solving for the unknown template RHS* $\boldsymbol{c}(\ell)$ given $\Lambda_\tau, \Gamma_\tau$ and Λ_0. Since Γ_τ and Λ_0 are not involved in any bilinear term, we do not fix them to specific values when solving for $\boldsymbol{c}(\ell)$.

Solving for Multipliers: Given the values for the current solution $\boldsymbol{c}^{(i)}(\ell)$ at each location, we simply plug in these values and solve the system in Figure 3.

Lemma 1. *The constraints shown in Fig. 3 become linear if we replace* $\boldsymbol{c}(\ell)$ *at each location by fixed (constant) values.*

The remaining constraints are linear over Λ_0, Λ_τ and Γ_τ for each transition τ, and can be thus solved using a LP solver. The following lemma guarantees that the constraints will always yield a feasible solution provided the values $\boldsymbol{c}^{(i)}$ are a valid post-fixed point.

Lemma 2. *If the solution* $\boldsymbol{c}^{(i)}(\ell)$ *for each* $\ell \in \mathcal{L}$ *is a post-fixed point, the constraints in the Fig. 3 are feasible for the remaining multipliers, when* $\boldsymbol{c}(\ell)$ *is replaced by* $\boldsymbol{c}^{(i)}(\ell)$.

Let $\Lambda_\tau^{(i)}$ be the resulting values of the bilinear multipliers returned by the LP solver when we replace $\boldsymbol{c} : \boldsymbol{c}^{(i)}$. These are also called policies [20].

Solving for Template RHS: Next, let us assume that the variables Λ_τ for each transition are set to constants $\Lambda_\tau^{(i)}$.

Lemma 3. *If we set* Λ_τ *for each* τ *to constants* $\Lambda_\tau^{(i)}$ *for the constraints in Figure 3, the resulting problem is linear over* $\boldsymbol{c}(\ell)$ *for each* $\ell \in \mathcal{L}$ *and the linear multipliers* Γ_τ, Λ_0.

Once we set Λ_τ to specific values, the resulting system is once again a linear program. Let us call this problem $\mathcal{C}_i$.

Lemma 4. *The LP $\mathcal{C}_i$ is always feasible.*

To see this, we note that $\boldsymbol{c}(\ell) = \boldsymbol{c}^{(i)}$ is already a solution to this LP due to how the values of $\Lambda_\tau^{(i)}$ were obtained in the first place. We call the resulting values $\boldsymbol{c}^{(i+1)}(\ell)$.

The overall policy iteration scheme alternates between solving for $\boldsymbol{c}(\ell)$ and solving for Λ_τ variables. Gaubert et al. show that the number of policies needed is finite (but large), and thus the process is guaranteed to yield a stable solution such that $\boldsymbol{c}^{(i+1)}(\ell) = \boldsymbol{c}^{(i)}(\ell)$.

5 Policies with Template Update

In this section, we extend policy iteration process to achieve two goals simultaneously: (a) be goal-directed towards a specific property and (b) allow the template T at each location to be updated.

Let (ℓ, ψ) be a error specification at location ℓ that we wish to prove unreachable. Our goal is to compute an inductive assertion map η such that at location ℓ, the conjunction $\eta(\ell) \wedge \psi$ is unsatisfiable. Once again, we will first assume for the sake of exposition that the same template matrix T is used at each location.

Using Farkas' lemma, the invariant $T\boldsymbol{x} \leq \boldsymbol{c}(\ell)$ proves the unreachability of the error specification $\psi : P\boldsymbol{x} \leq \boldsymbol{q}$ iff there exist multipliers $\boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0$ s.t.

$$T^\intercal \boldsymbol{\lambda}_s + P^T \boldsymbol{\gamma}_s = \boldsymbol{0}, \quad \underbrace{\boldsymbol{c}(\ell)^T \boldsymbol{\lambda}_s + \boldsymbol{q}^T \boldsymbol{\gamma}_s \leq -1}_{I}, \quad \boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0\,. \tag{11}$$

However, if the invariant fails to prove the property, we will be unable to find suitable multipliers $\boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0$. Since, our procedure will involve intermediate solutions that do not satisfy the property, we will consider the following optimization-based formulation by moving the inequality labeled "I" in (11) to the objective, as follows:

$$\begin{array}{ll} \min & \boldsymbol{c}(\ell)^T \boldsymbol{\lambda}_s + \boldsymbol{q}^T \boldsymbol{\gamma}_s \\ \mathsf{s.t.} & T^T \boldsymbol{\lambda}_s + P^T \boldsymbol{\gamma}_s = \boldsymbol{0} \\ & \boldsymbol{1}^T \boldsymbol{\lambda}_s = 1 \text{ (* normalization constraint *)} \\ & \boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0 \end{array} \tag{12}$$

Note that we have added a *normalization constraint* requiring that the sum of the multipliers $\boldsymbol{\lambda}_s$ equal 1. Without such a constraint, the problem always has a trivial solution 0 by setting all the multipliers $(\boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s)$ to 0, which is undesirable for the policy iteration scheme to be discussed subsequently.

Lemma 5. *Suppose $T_i = -P_j$ for row i of matrix T, row j of matrix P, and $\boldsymbol{c}(\ell)_i < \infty$ then the optimization problem in Eq. (12) is feasible.*

Furthermore, its objective value is strictly negative iff $T\boldsymbol{x} \leq \boldsymbol{c}(\ell)$ proves the specification $(\ell, \psi : P\boldsymbol{x} \leq \boldsymbol{q})$.

Vars :	$\boldsymbol{c}(\ell),\ \ell \in \mathcal{L}$	(* Template RHS *)
	$\Delta_\ell,\ \ell \in \mathcal{L}$	(* Template update *)
	$\Lambda_\tau,\ \tau \in \mathcal{T}$	(* Bilinear mult.*)
	$\boldsymbol{\lambda}_s$	(* Error Spec.*)
	$\Lambda_0, \Gamma_\tau,\ \tau \in \mathcal{T}$	(* Linear Mults. *)
	$\boldsymbol{\gamma}_s$	(* Error Spec.*)
min :	$\boldsymbol{\lambda}_s{}^T \boldsymbol{c}(\ell) + \boldsymbol{\gamma}_s^T \boldsymbol{q}$	
s.t.	$A_0^T \Lambda_0 = T_{\ell_0} + \Delta_{\ell_0}$	(* Initiation *)
	$\Lambda_0^T \boldsymbol{b}_0 \leq \boldsymbol{c}(\ell_0)$	
	$(T_l + \Delta_l)^T \Lambda_\tau + A_\tau^T \Gamma_\tau \ = \ (T_m + \Delta_m) U_\tau$	(* Consecution $\tau : \langle l, m, \varphi, g \rangle$ *)
	$\Lambda_\tau{}^T \boldsymbol{c}(\ell) + \Gamma_\tau^T \boldsymbol{b}_\tau \ \leq \ \boldsymbol{c}(m) - (T_m + \Delta_m) \boldsymbol{v}_\tau$	
	$(T_\ell + \Delta_\ell)^T \boldsymbol{\lambda}_s + P^T \boldsymbol{\gamma}_s = 0$	(* Error spec. $\psi : P\boldsymbol{x} \leq \boldsymbol{q}$ *)
	$\Lambda_0, \Lambda_\tau, \boldsymbol{\lambda}_s, \Gamma_\tau, \boldsymbol{\gamma}_s \geq 0$	(* Nonnegative multipliers *)
	$L_l \leq \Delta_l \leq U_l$	(* Limits on template change *)

Fig. 4. Bilinear system of constraints with objective function and template update variables Δ_l.

Proof. Given that $T_i = -P_j$, we then choose $\boldsymbol{\lambda}_s(i) = 1$ and the rest of entries to zero. Likewise, $\boldsymbol{\gamma}_s(j) = 1$ and the remaining entries of $\boldsymbol{\gamma}_s$ are set to 0. We can now verify that this will satisfy the constraints, thus providing a feasible solution.

Note that if we find a solution $(\boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s)$ such that the objective value is $\epsilon < 0$, then $(\frac{\boldsymbol{\lambda}_s}{|\epsilon|}, \frac{\boldsymbol{\gamma}_s}{|\epsilon|})$ satisfy the constraints in Eq. (11). The rest follows from Farkas' lemma.

Thus, we will use the optimization formulation as an objective function that measures how "far away" the current solution at ℓ is from proving the property of interest.

5.1 Updating Templates

Next, we allow the template T to change at each step to a new template $T + \Delta$, wherein Δ is the unknown change in the template. In doing so, we update the constraints to introduce an unknown change Δ. However, allowing arbitrary changes to the template will not work since choosing $\Delta = -T$ immediately makes the template trivial, and not useful for our purposes. Therefore, we specify upper and lower limits to the change in the template. These limits can be set using different strategies that we will explore in the experimental evaluation section. Let L be the lower limit and U be the upper limit so that $L \leq \Delta \leq U$. As a technical condition, we require $0 \in [L, U]$, i.e., the option to keep T unchanged is allowed.

Figure 4 shows the bilinear optimization problem

$$\mathcal{B}\left((\boldsymbol{c}(\ell), \Delta_\ell), (\Lambda_\tau, \boldsymbol{\lambda}_s)\right) ,$$

obtained when the change in the template variables is also considered. We note that the variables involved in the bilinear terms are once again separated into two sets, represented in different colors for convenience.

5.2 Template Updates and Policy Iteration

We now update the policy iteration process to consider the change in templates, as shown in Fig. 4. Let $\boldsymbol{c}^{(0)}$ be an initial value such that $T\boldsymbol{x} \leq \boldsymbol{c}^{(0)}(\ell)$ is inductive. The initial update $\Delta_\ell^{(0)} = 0$ for each location ℓ.

Multiplier Update: At each iteration i, the multiplier update uses $\boldsymbol{c}^{(i)}, \Delta^{(i)}$ to obtain values of $\Lambda_\tau^{(i)}, \boldsymbol{\lambda}_s^{(i)}$. Formally, we consider the problem

$$\mathcal{M}_i : \ \mathcal{B}\left((\boldsymbol{c}^{(i)}(\ell), \Delta_\ell^{(i)}), (\Lambda_\tau, \boldsymbol{\lambda}_s)\right)$$

Lemma 6. *1. $\mathcal{M}_i$ is a linear program over unknown multipliers $\Lambda_\tau, \boldsymbol{\lambda}_s, \Gamma_\tau, \boldsymbol{\gamma}_s, \Lambda_0$.*

2. It is feasible iff the map $\eta^{(i)}$ formed by the assertions $(T_\ell + \Delta_\ell^{(i)})\boldsymbol{x} \leq \boldsymbol{c}^{(i)}(\ell)$ for $\ell \in \mathcal{L}$, is an inductive assertion map.

3. The value of the objective function cannot increase, i.e., for $i > 1$,

$$\boldsymbol{c}^{(i)}(\ell)^T \boldsymbol{\lambda}_s^{(i)} + \boldsymbol{q}^T \boldsymbol{\gamma}_s^{(i)} \leq \boldsymbol{c}^{(i)}(\ell)^T \boldsymbol{\lambda}_s^{(i-1)} + \boldsymbol{q}^T \boldsymbol{\gamma}_s^{(i-1)} .$$

4. The value of the objective is negative iff $\eta^{(i)}$ proves the specification (ℓ, ψ).

The result of multiplier update yields values for the variables $(\Lambda_\tau, \boldsymbol{\lambda}_s)$: $(\Lambda_\tau^{(i)}, \boldsymbol{\lambda}_s^{(i)})$.

Template Update: Given the current values $(\Lambda_\tau^{(i)}, \boldsymbol{\lambda}_s^{(i)})$ for the multipliers, we derive new values $\boldsymbol{c}^{(i+1)}(\ell), \Delta_\ell^{(i+1)}$ for the template variables by solving the problem

$$\mathcal{C}_{i+1} : \ \mathcal{B}\left((\boldsymbol{c}(\ell), \Delta_\ell), (\Lambda_\tau^{(i)}, \boldsymbol{\lambda}_s^{(i)})\right) .$$

Lemma 7. *1. $\mathcal{C}_{i+1}$ is a linear program over the unknown template variables $\boldsymbol{c}(\ell), \Delta_\ell$ and unknown linear multipliers $\Gamma_\tau, \boldsymbol{\gamma}_s, \Lambda_0$.*

2. It is always feasible provided $0 \in [L_\ell, U_\ell]$ at each location.

3. The assertion map $\eta^{(i+1)}$ formed by the solution

$$(T_\ell + \Delta_\ell^{(i+1)})\boldsymbol{x} \leq \boldsymbol{c}^{(i+1)}(\ell) \text{ for } \ell \in \mathcal{L} ,$$

is inductive.

4. The value of the objective function cannot increase, i.e., for $i \geq 0$,

$$\boldsymbol{c}^{(i+1)}(\ell)^T \boldsymbol{\lambda}_s^{(i)} + \boldsymbol{q}^T \boldsymbol{\gamma}_s^{(i+1)} \leq \boldsymbol{c}^{(i)}(\ell)^T \boldsymbol{\lambda}_s^{(i)} + \boldsymbol{q}^T \boldsymbol{\gamma}_s^{(i)} .$$

5. The value of the objective function $\boldsymbol{c}^{(i+1)}(\ell)^T \boldsymbol{\lambda}_s^{(i)} + \boldsymbol{q}^T \boldsymbol{\gamma}_s^{(i+1)}$ is negative iff the $\eta^{(i+1)}$ proves the property.

The overall scheme alternates between updating the multipliers and the template variables, until no more changes can occur. We also observe that starting from a valid inductive invariant, the solutions obtained during the policy iteration continue to remain inductive or post-fixed points. However, they are post-fixed points over the lattice $\mathcal{A}(T_\ell + \Delta_\ell^{(i)},\ \ell \in \mathcal{L})$, which is different from the original lattice. As observed already in the motivating example (section 2), these invariants can be mutually incomparable. However, we show that at each step, the value of the objective function measuring progress towards proving the specification cannot increase.

5.3 Discussion

We now focus on issues such as convergence and the complexity of each step.

Convergence: In general, the known results about the convergence of alternating minimization schemes for bilinear optimization problems indicate that the process *seldom* converges to a global optimal value [27]. Often, these iterations get "stuck" in a local *saddle point*, from which no further progress is possible. Nevertheless, our goal here is not to converge to a global optimum but to a *good enough* solution whose objective function value is strictly negative, thus proving the property of interest.

By allowing template updates to the process, it is no longer clear that the process will necessarily converge (even if it converges to a saddle point) in finitely many steps. It is entirely possible that the value of the objective function remains unchanged but the process produces a new template $T_\ell + \Delta_\ell^{(i)}$ at each step. Depending on how the limits to the template change L_ℓ, U_ℓ are specified, this process may produce a fresh new template at each step.

Nevertheless, we note that the lack of convergence does not pose a serious hurdle to an application of template update to policy iteration. It is possible to iterate while each step provides at least $\epsilon > 0$ decrease in the value of the objective function, and stop otherwise.

Complexity: At each step, we solve a linear programming problem. For a transition system with n variables, $|\mathcal{L}|$ locations, $|\mathcal{T}|$ transitions, k template rows at each step, the size of each LP in terms of number of variables + constraints is $\mathcal{O}\left(|\mathcal{L}|kn + |\mathcal{T}|k^2\right)$. Although this is polynomial, the process can be prohibitively expensive for large programs. In our future work, we wish to exploit the block structure of these constraints in order to allow us to solve the LPs using standard approaches such as Benders or Danzig-Wolfe decomposition techniques [10]

Collecting Invariants: Finally, we note that each step yields an invariant map $\eta^{(i)}$ that is not necessarily comparable to the invariant obtained in the next step $\eta^{(i+1)}$. However, we note that the finite conjunction

$$\eta^{(0)}\ \wedge\ \cdots\ \wedge\ \eta^{(N)},$$

over all the iterations of this process can be a stronger invariant than each of them. This is already demonstrated by the motivating example in Section 2.

Table 1. Description of the benchmarks used and the sizes in terms of (# variables, # locations, # transitions)

ID	Size	Remark
1	(4,2,2)	Switched linear system with 4 state variables.
2	(2,2,4)	Example in Fig. 2.
3	(2,1,1)	Linear System with 1 location and transition.
4	(2,1,1)	Motivating example from Section 2.
5	(3,1,4)	Adjé et al. [1].
6	(2,35,169)	Grid-based piecewise linearization of Van Der Pol oscillator.

6 Experimental Evaluation

We present a preliminary experimental evaluation of the ideas presented thus far using a prototype implementation.

Prototype Implementation: A prototype implementation was developed in Python, using the exact arithmetic LP solver QSOptEx. The QSOptEx solver provides a fast and convenient interface to an optimized Simplex implementation in exact arithmetic. Our implementation allows the specification of a transition system and supports a few additional features on top of those presented in the paper including location invariants. We also support the option to specify different templates at various program locations. During the template update, our approach considers independent updates to the template at each location.

Specifying Template Changes: We consider a simple approach to specifying the limits L_ℓ, U_ℓ to the change in template at each location ℓ. First, the option for $\Delta_\ell = 0$ must be allowed, secondly, $\Delta_\ell = -T$ must be disallowed. For each $T_\ell(i,j) = 0$, we specify corresponding limits $L_\ell(i,j) = -z$ and $U_\ell(i,j) = z$ for a fixed constant $z > 0$ (taken as 1000 in our experiments). For $T_\ell(i,j) \neq 0$, we allow Δ to range between $\frac{1}{2}T_\ell(i,j)$ and $2T_\ell(i,j)$ in our experiments.

Benchmark Examples: We consider a small set of benchmark examples that are illustrative of applications that we encounter in the verification of discrete-time affine hybrid systems. Table 1 briefly describes each benchmark example.

Experimental Comparison: Table 2 shows the comparison between abstract interpretation using Kleene iteration, policy iteration without template update and with template update for the 6 benchmarks. The table reports the objective value of the initial solution obtained after the Kleene iteration using widening/narrowing terminates. A non-negative value of the objective function indicates the failure to prove the property. Overall, we see that policy iteration with template update is *effective* in these benchmarks in proving properties in 4 out of the 6 cases, whereas without template update we prove the property in just 1 out of 6. It is interesting that whenever the approaches manage to reduce the objective value of the initial solution, they end up proving the property. Further experiments are needed to clarify whether this represents an artifact of the benchmarks chosen.

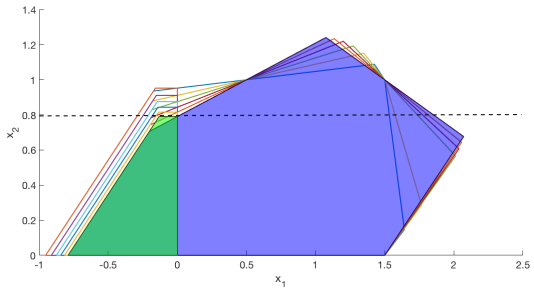

Fig. 5. Sequence of iterates for benchmark id 2 culminating in the final invariants shown shaded in blue and green. The property $x_2 \geq 0.8$ is shown unreachable at the green location by the final iterate.

Figure 5 shows the sequence of iterates at the two locations for the transition system shown in Fig. 2 corresponding to benchmark number 2. The goal is to establish the unreachability of $x_2 \geq 0.8$ at location ℓ_2. The final invariant for ℓ_2 is shown in green, proving the specification.

Thus, we provide preliminary evidence that the bilinear approach is effective in cases where Kleene or policy iteration fail. At the same time, we notice that the size of the bilinear problem, though polynomial in the original transition system and template size, is often large with thousands of variables. However, the problems are sparse with each constraint involving just a tiny fraction of these variables. This points out the need for simplification techniques and approaches to solving bilinear problems that exploit this sparsity to make the approach more efficient.

7 Conclusions

To conclude, we exploit the connection between template domains and bilinear constraints. In doing so, we show that policy iteration allows the template directions to be updated on the fly in a property directed fashion. We present preliminary evidence that such an approach can be effective, though many challenges remain. Our future work will focus on techniques to make progress when the policy iteration is stuck in a local saddle point, *without sacrificing the soundness of the approach*. In this context, we are investigating strategy iteration approaches that can incorporate the template update process [22]. Our previous work on invariant set computation for polynomial differential equations mentioned earlier, already contains clues to such an approach [37]. As mentioned earlier, exploiting the sparsity of constraints to provide a more scalable solver is also another fruitful future direction.

Acknowledgments

The authors gratefully acknowledge the anonymous reviewers for their valuable comments and suggestions. This work was funded in part by NSF under

Table 2. Experimental results including a comparison between policy iteration without template update and with template updated (shaded rows). All experiments were run on a Macbook Air laptop with 1.8 GHz Intel processor, 8GB RAM running OSX10.12. All timings are in seconds. **Legend: T. Upd:** Template updated at each iteration? **Proved?**: whether the property was proved, if not, the objective value is reported, |BOP|: size of the bilinear problem (# bilinear template variables, # bilinear mult. variables, # linear mult. variables), # Iter: # policy iterations - A (*) next to this number indicates that the iteration was stopped due to 5 consecutive steps with same objective value.

ID	Initial Template	Kleene		Policy Iteration				
	Type, $\|T\|$	Time	Proved?	T. Upd.	|BOP|	Time	# Iter	Proved
1	Pentagon, 26	0.37	N (0.2)	N	(52,1176, 1249)	0.5	2	N(0.2)
				Y	(240, 1176, 1249)	18.2	5(*)	N (0.2)
2	Octagon, 8	0.15	N (0.2)	N	(16, 264, 353)	0.1	2	N(0.2)
				Y	(48, 264, 353)	0.4	6	Y
3	Octagon, 8	0.04	N(0.5)	N	(8, 72, 161)	0.02	1	N(0.5)
				Y	(24, 72, 161)	0.05	2	Y
4	Interval, 4	0.02	N(15.5)	N	(4,20,33)	0.01	1	N(15.5)
				Y	(12, 20, 33)	0.02	2	Y
5	Pentagon, 10	1.5	N(2.83)	N	(10, 410, 681)	0.3	2	Y
				Y	(40, 410, 681)	0.3	2	Y
6	Interval, 4	2.5	N(0.75)	N	(140, 836, 2033)	1.5	5(*)	N(0.75)
				Y	(168, 836, 2033)	2.9	5(*)	N(0.75)

award numbers SHF 1527075. All opinions expressed are those of the authors, and not necessarily of the NSF.

References

1. Adjé, A., Garoche, P.: Automatic synthesis of piecewise linear quadratic invariants for programs. In: Verification, Model Checking, and Abstract Interpretation (VMCAI). pp. 99–116 (2015)
2. Adjé, A., Gaubert, S., Goubault, E.: Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. Logical Methods in Computer Science 8(1) (2012)
3. Adjé, A., Gaubert, S., Goubault, E.: Computing the smallest fixed point of order-preserving nonexpansive mappings arising in positive stochastic games and static analysis of programs. Journal of Mathematical Analysis and Applications 410(1), 227 – 240 (2014)
4. Amato, G., Parton, M., Scozzari, F.: Deriving Numerical Abstract Domains via Principal Component Analysis, pp. 134–150. Springer (2010)
5. 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: Prog. Lang. Design & Implementation. pp. 196–207. ACM Press (2003)
6. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: Design and implementation of a special-purpose static program

analyzer for safety-critical real-time embedded software (invited chapter). In: In The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones. LNCS, vol. 2566, pp. 85–108. Springer (2005)
7. Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.: Counterexample-guided refinement of template polyhedra (2017), to Appear
8. Chen, X., Abraham, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: Real Time Systems Symposium (RTSS). pp. 183–192. IEEE Press (2012)
9. Chen, X., Erika, Á.: Choice of directions for the approximation of reachable sets for hybrid systems. In: EUROCAST'11. pp. 535–542. Springer-Verlag, Berlin, Heidelberg (2012)
10. Chvátal, V.: Linear Programming. Freeman (1983)
11. Clariso, R., Cortadella, J.: The octahedron abstract domain. Science of Computer Programming 64(1), 115 – 139 (2007)
12. Colón, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: CAV. vol. 2725, pp. 420–433 (July 2003)
13. 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 (CAV). Lecture Notes in Computer Science, vol. 3576, pp. 462–475. Springer (2005)
14. Cousot, P.: Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In: VMCAI. Lecture Notes in Computer Science, vol. 3385, pp. 1–24. Springer (2005)
15. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proc. ISOP'76. pp. 106–130. Dunod, Paris, France (1976)
16. Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to Abstract interpretation, invited paper. In: PLILP '92. LNCS, vol. 631, pp. 269–295. springer (1992)
17. Cousot, P., Cousot, R.: Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM Principles of Programming Languages. pp. 238–252 (1977)
18. Delmas, D., Souyris, J.: Astrée: from research to industry. In: Proc. 14th International Static Analysis Symposium, SAS 2007. LNCS, vol. 4634, pp. 437–451. Springer, Berlin (2007)
19. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Proc. CAV'11. vol. 6806, pp. 379–395 (2011)
20. Gaubert, S., Goubault, E., Taly, A., Zennou, S.: Static analysis by policy iteration on relational domains. In: European Symposium on Programming. Lecture Notes in Computer Science, vol. 4421, pp. 237–252. Springer (2007)
21. Gawlitza, T., Seidl, H.: Precise fixpoint computation through strategy iteration. In: European Symposium on Programming (ESOP). Lecture Notes in Computer Science, vol. 4421, pp. 300–315. Springer (2007)
22. Gawlitza, T.M., Seidl, H.: Solving systems of rational equations through strategy iteration. ACM Trans. Program. Lang. Syst. 33(3), 11:1–11:48 (2011)
23. Ghaoui, L.E., Balakrishnan, V.: Synthesis of fixed-structure controllers via numerical optimization. In: Proceedings of the 33rd Conference on Decision and Control(CDC). IEEE (1994)
24. Goubault, E., Putot, S., Baufreton, P., Gassino, J.: Static analysis of the accuracy in control systems: Principles and experiments. In: FMICS. LNCS, vol. 4916, pp. 3–20. Springer (2008)

25. Guernic, C.L., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Analysis: Hybrid Systems 4(2), 250 – 262 (2010)
26. Gulwani, S., Srivastava, S., Venkatesan, R.: Program analysis as constraint solving. In: PLDI. pp. 281–292. ACM (2008)
27. Helton, J., Merino, O.: Coordinate optimization for bi-convex matrix inequalities. In: IEEE Conf. on Decision & Control(CDC). p. 36093613 (1997)
28. Logozzo, F., Fähndrich, M.: Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. In: Symposium on Applied Computing. pp. 184–188. SAC '08, ACM, New York, NY, USA (2008)
29. Mathworks Inc.: PolySpace design verifier, cf. http://www.mathworks.com/products/polyspace/ viewed April 2017
30. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: PADO II. vol. 2053, pp. 155–172 (May 2001)
31. Miné, A.: The octagon abstract domain. In: AST 2001 in WCRE 2001. pp. 310–319. IEEE, IEEE CS Press (October 2001)
32. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis (1999)
33. Roux, P., Voronin, Y.L., Sankaranarayanan, S.: Validating numerical semidefinite programming solvers for polynomial invariants. In: Static Analysis Symposium (SAS). Lecture Notes in Computer Science, vol. 9837, pp. 424–446. Springer (2016)
34. Sankaranarayanan, S., Dang, T., Ivančić, F.: Symbolic model checking of hybrid systems using template polyhedra. In: TACAS. LNCS, vol. 4963, pp. 188–202. Springer (2008)
35. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constraint-based linear-relations analysis. In: Static Analysis Symposium (SAS 2004). vol. 3148, pp. 53–69 (August 2004)
36. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Verification, Model-Checking and Abstract-Interpretation (VMCAI 2005). vol. 3385 (January 2005)
37. Sassi, M.A.B., Girard, A., Sankaranarayanan, S.: Iterative computation of polyhedral invariants sets for polynomial dynamical systems. In: IEEE Conference on Decision and Control (CDC). pp. 6348–6353. IEEE Press (2014)
38. Sassi, M.A.B., Sankaranarayanan, S., Chen, X., Ábraham, E.: Linear relaxations of polynomial positivity for polynomial lyapunov function synthesis. IMA Journal of Mathematical Control and Information 33, 723–756 (2016)
39. Venet, A., Brat, G.P.: Precise and efficient static array bound checking for large embedded C programs. In: PLDI. pp. 231–242. ACM (2004)
40. Weispfenning, V.: Quantifier elimination for real algebra—the quadratic case and beyond. In: Applied Algebra and Error-Correcting Codes (AAECC) 8. pp. 85–101 (1997)