**Formal Methods in Systems Design manuscript No.**
(will be inserted by the editor)

# Template Polyhedra and Bilinear Optimization.

**Jessica Gronski · Mohamed-Amin Ben Sassi · Stephen Becker · Sriram Sankaranarayanan**

Received: January 22, 2018 / Accepted: date

**Abstract** In this paper, we study the template polyhedral abstract domain using connections to bilinear optimization techniques. The connections between abstract interpretation and convex optimization approaches have been studied for nearly a decade now. Specifically, data flow constraints for numerical domains such as polyhedra can be 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 polyhedra wherein the desired invariants conform to a fixed template form. 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 the first part of this paper, we propose 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 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. However, it is well known that policy iteration can end up "stuck" in a saddle point from which future iterations cannot make progress.

In the second part of this paper, we study this problem further by empirically comparing policy iteration with a variety of other approaches for bilinear programming. These approaches adapt well-known algorithms to the special

Becker, Gronski and Sankaranarayanan
University of Colorado, Boulder, USA.
E-mail: firstname.lastname@colorado.edu

Ben Sassi
Mediterranean Institute of Technology, Tunis, Tunisia.
E-mail: mohamed.bensassi@medtech.tn

case of bilinear programs as well as using off-the-shelf tools for nonlinear programming. Our initial experience suggests that policy iteration seems to be the most advantageous approach for problems arising from abstract interpretation, despite the potential problems of getting stuck at a saddle point.

## 1 Introduction

In this paper, we exploit the connections between inferring post-fixed points (inductive invariants) for numerical domains and the process of solving nonlinear constraints to provide a template polyhedral domain that can modify the templates on-the-fly as the analysis progresses. 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 [60]. Template domains generalize a host of popular, "weakly domains" such as intervals [24], octagons [47,48], octahedra [20], pentagons [45], linear templates [60], and quadratic templates [3]. These domains have been well studied and proven to be effective for proving safety of runtime assertions in software [34,13,12,28, 46,67]. Template domains have given rise to specialized approaches such as policy iteration [22,30] for improving post-fixed points, and strategy iteration for computing the least fixed point [32,31].

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 [22] and generalized to arbitrary templates subsequently [30]. Extensions have been proposed for quadratic templates [3]. 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 [31]. 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.

The paper is divided into two parts. In the first part, we exploit the connection between policy iteration approach and classic bilinear optimization problems to design approaches that can vary the template on-the-fly. This is done by adapting policy iteration, which is a variant of the popular alternating coordinate descent that has been used widely in the control systems and optimization communities [33]. 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. 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 in-

ferring appropriate templates in a property directed manner. 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.

In the second part, starting in Section 7, we empirically compare policy iteration approach against other related approaches to local and global optimization problems [15,1,10,68,56,66]. 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) [39]. Adjé et al. demonstrate an approach that computes an optimal solution for systems which are nonexpansive [4]. However, the general applicability of this result is unclear.

Thus, given evidence that policy iteration (or alternating minimization) may not be a good method, we explore alternatives, and run numerical experiments in Section 9, even proposing our own new variant. Our results suggest that, at least on our benchmark problems, alternating minimization is in fact by far the best method, even compared with global optimization solvers. Furthermore, our implementation uses floating point, not exact arithmetic, and still achieves acceptable accuracy. Finally, although we do not focus on runtime, we do note that alternating minimization, in floating point arithmetic, is also one of the fastest solvers. The conclusion is that although alternating minimization can have difficulties with saddle points, because it exploits the specific structure of the problem, it may still be the best choice in many cases.

### 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 [21]. These constraints were solved using specialized quantifier elimination techniques, but restricted to small problems [69]. Sankaranarayanan et al. explored the use of heuristic approaches to solve bilinear constraints [59]. These approaches were generalized by Cousot, as instances of *Lagrangian relaxations* [23]. 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 [55,62]. 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 [38].

Template polyhedra and their generalization to support functions have proven useful for constructing reachable sets of linear and nonlinear hybrid sys-

tems [58,37,29,17]. 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 [60]. 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 [2].

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 [61]. 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 [18]. Recently, Bogomolov et al. propose a counter-example guided approach for inferring facets of template polyhedra for hybrid systems reachability analysis [14]. 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 [5].

## 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 property using a template domain, the user specifies a template matrix [60, 30]:

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

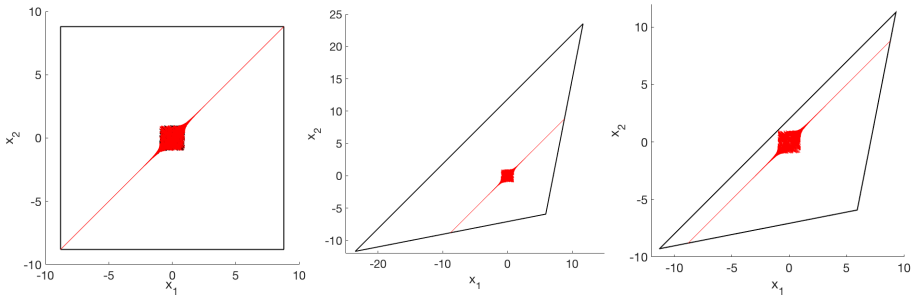


**Fig. 1** Invariants synthesized for the three steps of the policy iteration with property directed template modification. The simulation traces are shown in red. Note: each figure is drawn to a different scale.

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 $T\mathbf{x} \leq \mathbf{c}$ by discovering the unknown constants $\mathbf{c}$ that represent the RHS of the template. For the example shown above, the best possible invariant is obtained as $\mathbf{c} : \begin{pmatrix} 8.8 & 8.8 & 8.8 & 8.8 \end{pmatrix}^T$, 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:

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

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. [30], this approach does not guarantee that the iterates are strictly descending. In fact, the iterates obtained are often incomparable.

## 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 $\mathbf{x}$.

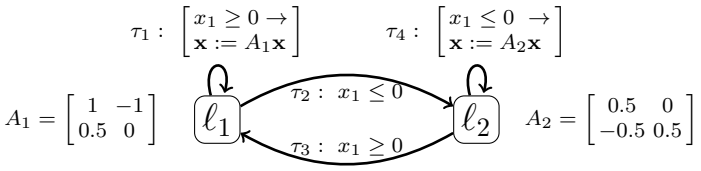


**Fig. 2** Example of a transition system with two variables $x_1, x_2$, two locations $\ell_1, \ell_2$ and four transitions shown as arrows.

**Definition 1 (Transition System)** A (numerical) *transition system* is a tuple $\langle X, \mathcal{L}, \mathcal{T}, \ell_0, \Theta \rangle$, 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 $\tau_i$ 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) $g_i : \mathbb{R}^n \rightarrow \mathbb{R}^n$, an update function, represents the (simultaneous) assignment: $(x_1, \ldots, x_n) := g_i(x_1, \ldots, x_n)$.
4. $\ell_0$ 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 $\langle \ell, \mathbf{x} \rangle$ wherein $\ell \in \mathcal{L}$ is the control location and $\mathbf{x} \in \mathbb{R}^n$ represents a set of valuations for the program variable. Given a transition system, its executions are a finite/infinite sequence of states:

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

such that: (a) $\ell_0$ is the initial location and $\mathbf{x}_0 \in [\![\Theta]\!]$; (b) $\ell_{i-1}, \ell_i$ are the pre/post locations (respectively) of the transition $\tau_i$ for all $i \geq 1$; (c) $\mathbf{x}_{i-1} \in [\![\psi_i]\!]$ for all $i \geq 1$ wherein $\psi_i$ is the guard corresponding to the transition $\tau_i$; and (d) $\mathbf{x}_i = g_i(\mathbf{x}_{i-1})$ for all $i \geq 1$, wherein $g_i$ is the update function for $\tau_i$.

*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 $\mathbf{x} := \mathbf{x}$ is not shown, however. The initial location is $\ell_1$ and the initial condition on $\mathbf{x}$ is $(x_1, x_2) \in [0.5, 1.5] \times [0.5, 1]$.

A state $(\ell, \mathbf{x})$ 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 $e : \mathbf{a}^T\mathbf{x}$ for vector $\mathbf{a} \in \mathbb{R}^n$. A linear inequality is of the form $\mathbf{a}^T\mathbf{x} \leq b$ and a linear assertion is a finite conjunction of linear inequalities $(\mathbf{a}_1^T\mathbf{x} \leq b_1 \ \wedge \ \cdots \ \wedge \ \mathbf{a}_k^T\mathbf{x} \leq b_k)$ conveniently written in matrix form as $A\mathbf{x} \leq \mathbf{b}$.

**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: $g_i(\mathbf{x}) : U_i\mathbf{x} + \mathbf{v_i}$.

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

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

- *Initial Condition:* At the initial location $\ell_0$, 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:

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

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

Let $\eta$ 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 $(\ell, \mathbf{x})$, it follows that $\mathbf{x} \notin [\![\psi]\!]$.*

*Proof* The proof first establishes that for any reachable state $\langle \ell, \mathbf{x} \rangle$ of the system, we have that $\mathbf{x} \in [\![\eta(\ell)]\!]$. In other words, the inductive invariants characterize all reachable states. Therefore, if $\eta(\ell) \ \wedge \ \psi$ is unsatisfiable, then no state can be reachable and satisfy $\psi$.

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

### 3.1 Abstract Interpretation

Abstract interpretation provides a framework for systematically computing inductive assertions using a pre-specified lattice of assertions called an *abstract domain* [26, 25]. The key insight lies in characterizing inductive assertion maps as post-fixed points of a monotone operator over sets of states.

In this section, we briefly sketch the basics of abstract interpretation, and the Kleene iteration using widening to compute post-fixed points.

The *concrete domain* $\Sigma$ is a lattice whose elements are first order assertions over $X$, ordered by entailment $\models$. The logical disjunction $\vee$ is the join operator and conjunction $\wedge$ is the meet operator in this lattice. The bottom element is *false* and the top element is *true*. We define the post condition operation over sets of states and a transition $\tau$.

**Definition 4 (Post-Condition)** Given a set $\psi \in \Sigma$, its post condition with respect to a transition $\tau : \langle \ell_1, \ell_2, \varphi, g \rangle$ is the set of all states reachable from some state in $[\![\psi]\!]$ in one step by executing the transition $\tau$:

$$post(\psi, \tau) : \ (\exists\ \mathbf{y})\ \psi(\mathbf{y})\ \wedge\ \varphi(\mathbf{y})\ \wedge\ \mathbf{x} = g(\mathbf{y})\,.$$

We consider assertion maps $\eta : \mathcal{L} \to \Sigma$ and let $\mathcal{N}$ be the set of all such maps. We lift the $\models$ operator from assertions to maps: $\eta_1 \models \eta_2$ iff for all $\ell \in \mathcal{L}$, $\eta_1(\ell) \models \eta_2(\ell)$. Thus, $\mathcal{N}$ forms a lattice with the lifted $\models$ as the inclusion. Next, we define a monotone operator $\mathcal{F} : \mathcal{N} \to \mathcal{N}$ as

$$\mathcal{F}(\eta)(\ell) : \begin{cases} \Theta\ \vee\ \bigvee_{\tau:\langle m,\ell,\varphi,g\rangle} post(\eta(m), \tau) & \ell = \ell_0 \\ \bigvee_{\tau:\langle m,\ell,\varphi,g\rangle} post(\eta(m), \tau) & \text{otherwise} \end{cases}$$

An assertion map $\eta$ is a post fixed point of $\mathcal{F}$ iff

$$\mathcal{F}(\eta) \models \eta\,.$$

**Theorem 2** *An assertion map is inductive if and only if it is a post fixed point of $\mathcal{F}$.*

*Proof* The proof is available in most textbooks on static analysis [49]. A proof of this statement using the same notation as this section is available in the PhD thesis of Sankaranarayanan [57] (Lemma 3.2).

To compute a post-fixed point, we start with the bottom element of $\mathcal{N}$, an assertion map $\eta_\perp$ such that $\eta_\perp(\ell) = \mathit{false}$ for all $\ell \in \mathcal{L}$. We define the Kleene iteration as the sequence obtained by iterating $\mathcal{F}$ over $\eta_\perp$.

$$\eta^{(i)} : \begin{cases} \mathcal{F}(\eta_{i-1}) & i \geq 1 \\ \eta_\perp & i = 0 \end{cases}.$$

The process is stopped whenever $\eta^{(i+1)} \models \eta^{(i)}$, in which case, we can show that $\eta^{(i+1)} \equiv \eta^{(i)}$ is the *least fixed point*. However, the iteration may go on forever even for simple programs. To make matters worse, each step potentially yields larger and more complex formulas, making the computation of *post*, $\vee$ and $\models$ prohibitively expensive.

To counter this, abstract interpretation defines an abstract domain which is a lattice $\langle A, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ with inclusion $\sqsubseteq$, join operator $\sqcup$, meet operator $\sqcap$, a bottom element $\bot$ and top element $\top$ wherein each element $a \in A$ is linked to the concrete domain through the concretization function $\gamma(a) \in \Sigma$. Likewise, each assertion $\psi \in \Sigma$ is linked to $A$ through the abstraction function $\alpha(\gamma) \in A$. Together, the pair $\alpha, \gamma$ form a *Galois* connection:

$$(\forall a \in A,\ \varphi \in \Sigma)\ a \sqsubseteq \alpha(\varphi)\ \textit{iff}\ \gamma(a) \models \varphi\,.$$

The abstract post condition operation is defined as $\widehat{post}(a, \tau)$ with a *soundness condition*:

$$(\forall a \in A)\ post(\gamma(a), \tau) \models \gamma(\widehat{post}(a, \tau))\,.$$

Once again, we lift the domain $A$ to a lattice over maps $N : \mathcal{L} \to A$. The abstract operator $\mathcal{G}$ is now defined analogous to the concrete operator $\mathcal{F}$.

$$\mathcal{G}(\widehat{\eta})(\ell) : \begin{cases} \alpha(\Theta) \sqcup \bigsqcup_{\tau:\langle m,\ell,\varphi,g\rangle} \widehat{post}(\widehat{\eta}(m),\tau) & \ell = \ell_0 \\ \bigsqcup_{\tau:\langle m,\ell,\varphi,g\rangle} \widehat{post}(\widehat{\eta}(m),\tau) & \text{otherwise} \end{cases}$$

A map $\widehat{\eta}$ is an abstract fixed point iff $\mathcal{G}(\widehat{\eta}) \sqsubseteq \widehat{\eta}$. The following theorem summarizes the core soundness property of abstract interpretation.

**Theorem 3** *$\widehat{\eta}$ is an abstract post fixed point iff $\gamma \circ \widehat{\eta}$ is an inductive assertion map.*

The proof is available from most expositions of abstract interpretation [49, 26]. For a proof using the notation introduced in this section, we refer the reader to Theorem 3.1 of Sankaranarayanan's PhD thesis [57].

Once again, the abstract Kleene iteration can be applied to compute a post fixed point in the abstract domain.

$$\widehat{\eta}^{(i)} : \begin{cases} \mathcal{G}(\widehat{\eta}_{i-1}) & i \geq 1 \\ \widehat{\eta}_{\perp} & i = 0 \end{cases}.$$

If the lattice $A$ has the *ascending chain condition* property, then the process is guaranteed to converge, yielding an inductive assertion map. Otherwise, the process can still continue for ever. In this case, we use a widening operator to guarantee convergence. Formally, the widening operator $\nabla : A \times A \to A$ has the following properties:

1. $a \sqcup b \sqsubseteq a \nabla b$ for all $a, b \in A$.
2. For any non-decreasing sequence

   $$a_0 \sqsubseteq a_1 \sqsubseteq \cdots$$

   the corresponding widened sequence

   $$b_0 : a_0, b_1 : b_0 \nabla a_1, \cdots b_{i+1} : b_i \nabla a_{i+1} \cdots$$

   always converges in finitely many steps to yield $b_{i+1} \sqsubseteq b_i$.

In practice, widening causes an unacceptable loss in precision that is improved using a narrowing iteration. A narrowing operator $\Delta$ is used to terminate a descending sequence of lattice elements:

$$b_0 \sqsupseteq b_1 \sqsupseteq b_2 \cdots .$$

It satisfies the key property that if $a \sqsupseteq b$ then $a \sqsupseteq (a \Delta b) \sqsupseteq b$, and furthermore, the narrowed descending iteration

$$c_0 : b_0, \; c_1 : (c_0 \Delta b_1), \; c_2 : (c_1 \Delta b_2), \cdots, c_{j+1} : (c_j \Delta b_{j+1}) \cdots,$$

terminates in finitely many steps.

### 3.2 Template Domains

The rest of this paper will focus on the abstract domain of template polyhedra [60]. Let $\mathcal{S} : \langle \mathcal{L}, X, \mathcal{T}, \ell_0, \Theta \rangle$ be a linear transition system. Let $\mathbf{x}$ 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 $T_\ell$. We drop the subscript $\ell$ from the template matrix if the location $\ell$ is clear from the context. A $m \times n$ template $T$ defines a lattice $\mathcal{A}(T)$:

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

In other words, each element of the template abstract domain is a possible valuation $\mathbf{c}$ to the RHS of inequalities $T\mathbf{x} \leq \mathbf{c}$. Note that the entries in $\mathbf{c}$ 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 $\varphi$ over $\mathbf{x}$, its abstraction $\mathbf{c} : \alpha(\varphi)$ is computed as a vector whose $i^{th}$ entry $\mathbf{c}_i$ is the solution to the optimization problem:

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

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

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

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

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 $T\mathbf{x} \leq \mathbf{c}$ is identical to the polyhedron $T\mathbf{x} \leq \mathsf{can}(\mathbf{c})$. A template element $\mathbf{c}$ is *canonical* in $\mathcal{A}(T)$ if and only $\mathbf{c} = \mathsf{can}(\mathbf{c})$.

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

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

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

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

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

Widening and narrowing operators for the template domain are defined by extensions of the standard interval widening operator [60].

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 [30]. Once again we fix a linear transition system $\mathcal{S}$ and assume for simplicity that each location $\ell$ 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 $\varphi : \ A\mathbf{x} \leq \mathbf{b}$ be a linear assertion with $m \times n$ matrix $A$ and $m \times 1$ vector $\mathbf{b}$, $\psi : \ \mathbf{c}^T\mathbf{x} \leq d$ be a given linear inequality.

**Theorem 4 (Farkas Lemma)** *If $\varphi$ is satisfiable, then $\varphi \models \psi$ iff there exists nonnegative multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that*

$$A^T\boldsymbol{\lambda} = \mathbf{c} \ \wedge \ \mathbf{b}^T\boldsymbol{\lambda} \leq d \ \wedge \ \boldsymbol{\lambda} \geq 0\,. \tag{1}$$

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

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

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

A proof can be found in most textbooks that deal with linear optimization [19]. Given a system of constraints $\varphi : A\mathbf{x} \leq \mathbf{b}$, the form in Eq. (1) is often referred to as the dual. Furthermore, the multipliers $\boldsymbol{\lambda}$ are often referred to as as the dual variables or dual multipliers.

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

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

All equalities and inequalities between matrices are interpreted entrywise. Here $\Lambda$ 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 $\Lambda$ contains the multipliers corresponding to the inequality $C_j\mathbf{x} \leq \mathbf{d}_j$. 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 $\ell$ with an unknown vector $\mathbf{c}(\ell) \in \mathcal{A}(T)$ such that the assertion map $\eta(\ell) : T\mathbf{x} \leq \mathbf{c}(\ell)$ is inductive.

We wish to encode the constraints for initiation:

$$\Theta \models T\mathbf{x} \leq \mathbf{c}(\ell_0)\,, \tag{2}$$

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

$$T\mathbf{x} \leq \mathbf{c}(\ell) \ \wedge \ \varphi \ \wedge \ \mathbf{x}' = g(\mathbf{x}) \ \models \ T\mathbf{x}' \leq \mathbf{c}(m)\,. \tag{3}$$

**Initiation:** Let $\Theta : A_0\mathbf{x} \leq \mathbf{b}_0$ be the assertion for the initial condition. Using Farkas' lemma for the entailment in Eq. (2), we obtain the condition:

$$A_0^T\Lambda_0 = T \ \wedge \ \Lambda_0^T\mathbf{b}_0 \leq \mathbf{c}(\ell_0) \ \wedge \ \Lambda_0 \geq 0\,. \tag{4}$$

Here $\Lambda_0$ 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 $\Lambda_0$ are non-negative.

**Consecution:** Let $\tau$ be a transition with guard $A_\tau\mathbf{x} \leq \mathbf{b}_\tau$ and update $g(\mathbf{x}) : U_\tau\mathbf{x}+\mathbf{v}_\tau$. The consecution condition in Eq. (3) can be rewritten through substitution of $\mathbf{x}'$ and arranged as follows:

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

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

$$\begin{array}{c} T^T\Lambda_\tau + A_\tau^T\Gamma_\tau = TU_\tau \\ \Lambda_\tau^T\mathbf{c}(\ell) + \Gamma_\tau^T\mathbf{b}_\tau \leq \mathbf{c}(m) - T\mathbf{v}_\tau \\ \Lambda_\tau, \Gamma_\tau \geq 0 \end{array} \tag{5}$$

Note that Eq. (4) for the initiation yields a system of linear constraints involving $\mathbf{c}(\ell_0)$ and unknown multipliers in $\Lambda_0$. However, the consecution constraints in Eq. (5) for each transition $\tau$ involve the product $\Lambda_\tau^T\mathbf{c}(\ell)$ both of which are unknown. This makes the constraints for consecution fall into a special class called *bilinear constraints*. I.e., for a fixed $\Lambda_\tau$ these constraints are linear in the remaining variables $\mathbf{c}(\ell), \Gamma_\tau$. Similarly, for fixed values of $\mathbf{c}(\ell)$, these constraints are linear in the variables $\Lambda_\tau, \Gamma_\tau$. Figure 3 summarizes the constraints obtained at a glance. Note that the multipliers $\Lambda_\tau$ are called *bilinear multipliers* since they are multiplied with the unknowns $\mathbf{c}(\ell)$ to form

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

**Fig. 3** Bilinear system of constraints at a glance. The constraints are generalized to allow for possibly different templates $T_\ell$ at each location.

the nonlinear terms in the constraints. On the other hand, note that $\Lambda_0, \Gamma_\tau$ variables are not multiplied with other unknowns.

**Connection with Min-Policies:** The original "min-policy" approach of Costan et al. [22] considers data flow equations of the form:

$$\mathbf{c} \geq \min(\mathbf{a}_{i,1}^T \mathbf{c}, \ldots, \mathbf{a}_{i,k}^T \mathbf{c}) \,,\ i = 1, \ldots, M,\ k = 1, \ldots, N \,. \tag{6}$$

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 $\ell$ with template $T$ and unknown template RHS variables $\mathbf{c}$. All transitions are assumed to be self-loops around this location. From eq. (5), a given solution $\mathbf{c}$ satisfies the consecution for transition $\tau$ iff there exist $\Lambda_\tau, \Gamma_\tau$ such that

$$\mathbf{c} \ \geq \ \Lambda_\tau^T \mathbf{c} + \Gamma_\tau^T \mathbf{b}_\tau + T \mathbf{v}_\tau \tag{7}$$

$$T^T \Lambda_\tau + A_\tau^T \Gamma_\tau \ = \ T U_\tau \tag{8}$$

$$\Lambda_\tau, \Gamma_\tau \ \geq 0 \tag{9}$$

Let us define a polyhedron $P(\Lambda_\tau, \Gamma_\tau)$ defined by collecting the constraints in lines (8), (9) above. We may rewrite the constraints equivalently as:

$$\mathbf{c} \ \geq \ \min_{(\Lambda_\tau, \Gamma_\tau) \in P} \left( \Lambda_\tau^T \mathbf{c} + \Gamma_\tau^T \mathbf{b}_\tau + T \mathbf{v}_\tau \right) \tag{10}$$

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. (10) can be equivalently written as a minimization over the finite set of vertices of $P$:

$$\mathbf{c} \ \geq \ \min_{j=1}^{N} \left( \Lambda_j^T \mathbf{c} + \Gamma_j^T \mathbf{b}_\tau + T \mathbf{v}_\tau \right) \tag{11}$$

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 $\mathbf{c}(\ell)$ for each $\ell \in \mathcal{L}$ and solving for the unknown bilinear multipliers $\Lambda_\tau$. Policy iteration starts from a known sound solution $\mathbf{c}^0(\ell)$ 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 $\mathbf{c}^0(\ell) \neq \bot$, for each $\ell \in \mathcal{L}$. If this were the case, then the location $\ell$ 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 $\mathbf{c}$, and (II) *solving for the unknown template RHS* $\mathbf{c}(\ell)$ given $\Lambda_\tau, \Gamma_\tau$ and $\Lambda_0$. Since $\Gamma_\tau$ and $\Lambda_0$ are not involved in any bilinear term, we do not fix them to specific values when solving for $\mathbf{c}(\ell)$.

**Solving for Multipliers:** Given the values for the current solution $\mathbf{c}^{(i)}(\ell)$ 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* $\mathbf{c}(\ell)$ *at each location by fixed (constant) values.*

*Proof* Proof is by inspection. We run through each inequality and note that the constraints are linear in the variables $\mathbf{c}^{(i)}(\ell)$ and the multipliers $\Lambda_0, \Lambda_\tau$, and $\Gamma_\tau$.

The remaining constraints are linear over $\Lambda_0, \Lambda_\tau$ and $\Gamma_\tau$ for each transition $\tau$, 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 $\mathbf{c}^{(i)}$ are a valid post-fixed point.

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

*Proof* Let us assume that the solutions $\mathbf{c}^{(i)}(\ell)$ yield a post-fixed point over the template polyhedra domain. We now wish to show that the constraints in Fig. 3 are feasible when we replace each $\mathbf{c}(\ell)$ by $\mathbf{c}^{(i)}(\ell)$.

The proof is obtained by combining two facts: (a) because $\mathbf{c}^{(i)}(\ell)$ form a post-fixed point, the initiation condition (Eq. (2)) and consection conditions for each transition $\tau \in \mathcal{T}$ ( Eq. (3) ) for inductive invariance hold. (b) Because these entailments hold, we can apply Farkas' lemma to each of them and derive the existence of multipliers for Eq. (4) and Eq. (5). However, the constraints in Figure 3 are just a conjunction of these constraints. Therefore, we conclude that there exist values of the multipliers that satisfy these constraints when we substitute $\mathbf{c}^{(i)}(\ell)$ for each $\mathbf{c}(\ell)$.

Let $\Lambda_\tau^{(i)}$ be the resulting values of the bilinear multipliers returned by the LP solver when we replace $\mathbf{c} : \mathbf{c}^{(i)}$. These are also called policies [30].

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

**Lemma 3** *If we set $\Lambda_\tau$ for each $\tau$ to constants $\Lambda_\tau^{(i)}$ for the constraints in Figure 3, the resulting problem is linear over $\mathbf{c}(\ell)$ for each $\ell \in \mathcal{L}$ and the linear multipliers $\Gamma_\tau, \Lambda_0$.*

*Proof* Proof is once again by inspection of each constraint in Figure 3.

Once we set $\Lambda_\tau$ 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.*

*Proof* To see this, we note that $\mathbf{c}(\ell) = \mathbf{c}^{(i)}$ 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 $\mathbf{c}^{(i+1)}(\ell)$.

The overall policy iteration scheme alternates between solving for $\mathbf{c}(\ell)$ and solving for $\Lambda_\tau$ 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 $\mathbf{c}^{(i+1)}(\ell) = \mathbf{c}^{(i)}(\ell)$.

## 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 $(\ell, \psi)$ be a error specification at location $\ell$ that we wish to prove unreachable. Our goal is to compute an inductive assertion map $\eta$ such that at location $\ell$, 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 $T\mathbf{x} \leq \mathbf{c}(\ell)$ proves the unreachability of the error specification $\psi : P\mathbf{x} \leq \mathbf{q}$ iff there exist multipliers $\boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0$ s.t.

$$T^\intercal \boldsymbol{\lambda}_s + P^T \boldsymbol{\gamma}_s = \mathbf{0}, \quad \underbrace{\mathbf{c}(\ell)^T \boldsymbol{\lambda}_s + \mathbf{q}^T \boldsymbol{\gamma}_s \leq -1}_{I}, \quad \boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0\,. \tag{12}$$

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 (12) to the objective, as follows:

$$\begin{array}{ll} \min & \mathbf{c}(\ell)^T \boldsymbol{\lambda}_s + \mathbf{q}^T \boldsymbol{\gamma}_s \\ \mathsf{s.t.} & T^T \boldsymbol{\lambda}_s + P^T \boldsymbol{\gamma}_s = \mathbf{0} \\ & \mathbf{1}^T \boldsymbol{\lambda}_s = 1 \text{ (* normalization constraint *)} \\ & \boldsymbol{\lambda}_s, \boldsymbol{\gamma}_s \geq 0 \end{array} \tag{13}$$

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 $\mathbf{c}(\ell)_i < \infty$ then the optimization problem in Eq. (13) is feasible.*

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

*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. (12). 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 $\ell$ 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 $T^{(i+1)} : T^{(i)} + \Delta$, starting with the initial template $T^{(0)} = T$, wherein $\Delta$ is the unknown change in the template. As a result, our analysis explores a series of templates:

$$T^{(0)},\ T^{(1)}, \ldots, T^{(N)}\,.$$

In doing so, we update the constraints to introduce an unknown change $\Delta$. However, allowing arbitrary changes to the template will not work since choosing $\Delta = -T^{(i)}$ 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 $L \leq T^{(i)} + \Delta \leq U$. As a technical condition, we require $T^{(i)} \in [L, U]$, i.e., the option to keep the current template $T^{(i)}$ unchanged is allowed at each step.

Figure 4 shows the bilinear optimization problem

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

obtained when the change in the template variables is also considered. Here note that $\ell$ ranges over all location, $\tau$ over all transitions and finally, $\boldsymbol{\lambda}_s$ pertains to the property to be checked which is assumed to be relevant to a single location in the program. We note that the variables involved in the bilinear terms are once again separated into two sets, represented in different colors for convenience.

| | | |
|---|---|---|
| **Vars** : | $\mathbf{c}(\ell),\ \ell \in \mathcal{L}$ | (* 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** : | $\mathbf{c}(\ell)^T\boldsymbol{\lambda}_s + \mathbf{q}^T\boldsymbol{\gamma}_s$ | |
| **s.t.** | $A_0^T\Lambda_0 = T_{\ell_0}^{(i)} + \Delta_{\ell_0}$ | (* Initiation *) |
| | $\Lambda_0^T\mathbf{b}_0 \leq \mathbf{c}(\ell_0)$ | |
| | $(T_\ell^{(i)} + \Delta_\ell)^T\Lambda_\tau + A_\tau^T\Gamma_\tau = (T_m^{(i)} + \Delta_m)U_\tau$ | (* Consecution $\tau : \langle \ell, m, \varphi, g\rangle$ *) |
| | ${\Lambda_\tau}^T\mathbf{c}(\ell) + {\Gamma_\tau}^T\mathbf{b}_\tau \leq \mathbf{c}(m) - (T_m^{(i)} + \Delta_m)\mathbf{v}_\tau$ | |
| | $(T_\ell^{(i)} + \Delta_\ell)^T\boldsymbol{\lambda}_s + P^T\boldsymbol{\gamma}_s = 0$ | (* Error spec. $\ell,\ \psi : P\mathbf{x} \leq \mathbf{q}$ *) |
| | $\mathbf{1}^T\boldsymbol{\lambda}_s = 1$ | |
| | $\Lambda_0, \Lambda_\tau, \boldsymbol{\lambda}_s, \Gamma_\tau, \boldsymbol{\gamma}_s \geq 0$ | (* Nonnegative multipliers *) |
| | $L_\ell \leq T_\ell^{(i)}\Delta_\ell \leq U_\ell$ | (* Limits on template change *) |

**Fig. 4** Bilinear system of constraints with objective function and template update variables $\Delta_l$. The current template after the $i^{th}$ iteration at location $m \in \mathcal{L}$ is denoted $T_m^{(i)}$.

## 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 $\mathbf{c}^{(0)}$ be an initial value such that $T_\ell^{(0)}\mathbf{x} \leq \mathbf{c}^{(0)}(\ell)$ is inductive. The initial template $T_\ell^{(0)}$ is specified by the user, and furthermore, the initial inductive invariant is assumed to be computed by abstract interpretation. The initial update $\Delta_\ell^{(0)} = 0$ for each location $\ell$.

**Note:** For convenience, we will assume that $(\mathbf{c}^{(0}(\ell))_i \neq \pm\infty$. Indeed, if any entry of $\mathbf{c}_\ell^{(i)}$ is $-\infty$, then the location $\ell$ is deemed unreachable and removed from the transition system. Also, if any entry of $\mathbf{c}_\ell^{(i)}$ is $+\infty$, we will simply remove the corresponding template row from our analysis.

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

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

**Lemma 6** *The following are true:*

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 $(T_\ell^{(i)} + \Delta_\ell^{(i)})\mathbf{x} \leq \mathbf{c}^{(i)}(\ell)$ for $\ell \in \mathcal{L}$, is an inductive assertion map.*
3. *The value of the objective function cannot increase, i.e., for $i > 1$,*

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

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

*Proof* **(1)** First, to see that $\mathcal{M}_i$ is a linear program, we inspect the constraints in Figure 4. Each constraint is linear in the unknowns $(\Lambda_\tau, \boldsymbol{\lambda}_s)$.
**(2)** The proof is identical to that of Lemma 2. Suppose, the map $\eta : \ell \rightarrow (T_\ell^{(i)} + \Delta_\ell^{(i)})\mathbf{x} \leq \mathbf{c}^{(i)}(\ell)$ foreach $\ell \in \mathcal{L}$, forms an inductive assertion map. We note that $\eta$ satisfies the conditions for initiation (4) and consecution (5) with the template $T_\ell^{(i)}$ replaced by $T_\ell^{(i+1)} : T_\ell^{(i)} + \Delta_\ell^{(i)}$. Dualizing the initiations and consecution conditions using Farkas lemma yields the constraints in Fig. 4. As a result, we note that when we substitute values $\Delta_\ell^{(i)}$ for each $\Delta_\ell$ and $\mathbf{c}^{(i)}(\ell)$ for each $\mathbf{c}(\ell)$, the multipliers have a feasible solution provided by Farkas' lemma.
**(3)** We note that $\boldsymbol{\lambda}_s^{(i-1)}, \boldsymbol{\gamma}_s^{(i-1)}$ form (part of a ) feasible solution for $\mathcal{M}_i$ and furthermore, $\boldsymbol{\lambda}_s^{(i)}$ and $\boldsymbol{\gamma}_s^{(i)}$ form part of the optimal solution for the same. Since, we are minimizing the objective, the optimal objective must be less than or equal to any feasible solution.
**(4)** The value of the objective is negative iff $\eta^{(i)}$ proves the specification $(\ell, \psi)$.

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 $\mathbf{c}^{(i+1)}(\ell), \Delta_\ell^{(i+1)}$ for the template variables by solving the problem

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

**Lemma 7** *The following facts hold:*

1. *$\mathcal{C}_{i+1}$ is a linear program over the unknown template variables $\mathbf{c}(\ell), \Delta_\ell$ and unknown linear multipliers $\Gamma_\tau, \boldsymbol{\gamma}_s, \Lambda_0$.*
2. *It is always feasible provided $T^{(i)} \in [L_\ell, U_\ell]$ at each location at each iteration.*
3. *The assertion map $\eta^{(i+1)}$ formed by the solution*

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

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

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

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

*Proof* Proof is entirely analogous to Lemma 6.
**(1)** We note that $\mathcal{C}_i$ is a linear program since each constraint in Figure 4 is linear in the unknowns $(\mathbf{c}(\ell), \Delta_\ell)$.
**(2)** We note that setting $\Delta_\ell = 0, \mathbf{c}(\ell) = \mathbf{c}^{(i)}(\ell)$ yields a feasible solution for $\mathcal{C}_i$.
**(3)** We note that the assertion map $\eta^{(i+1)}$ formed by the solution

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

satisfies the dual of the initiation and consecution constraints due to the existence of the multipliers, and thus by Farkas lemma satisfies the conditions themselves.
**(4)** We note that $\mathbf{c}^{(i)}(\ell), \Delta_\ell = 0$ is already feasible for $\mathcal{C}_i$. Therefore, the objective value achieved by the optimal solution must be less than or equal to that achieved by any feasible solution.
**(5)** This follows directly from Lemma 5.

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 $\mathcal{A}(T_\ell^{(i)} + \Delta_\ell^{(i)},\ \ell \in \mathcal{L})$, 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 [39]. 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.

*Example 2* The following example taken from Adjé et al [4] demonstrates the fundamental difficulties of using policy iteration to calculate a "good" invariant.

```
var x  initially 0 <= x <= 1
while (x <= 100)
    x := (1 - x)
end
```

The program can be written in the formalism of this paper as a single location $\ell$ representing the head of the while loop and a single transition $\tau$ with guard $x \leq 100$ and update $x' = 1 - x$.

Since the program has a single variable, the "best" template possible using linear expressions is equivalent to the interval domain.

Policy iteration does not converge to the best solution $0 \leq x \wedge x \leq 1$, unless it is directly initialized to this invariant. For example, starting from the initial solution $0 \leq x \leq 100$, it converges instead to $-99 \leq x \leq 100$. ■.

By allowing template updates to the process, the problem is worsened in a sense. 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 $T_\ell^{(i)} + \Delta_\ell^{(i)}$ at each step. Depending on how the limits to the template change $L_\ell, U_\ell$ 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 [19]

**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.

## 6 Evaluating Policies with Template Updates

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 $L_\ell, U_\ell$ to the change in template at each location $\ell$. 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

**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. [2]. |
| 6 | (2,35,169) | Grid-based piecewise linearization of Van Der Pol oscillator. |
| 7 | (4,5,54) | A piecewise linear dynamical system. |
| 8 | (5,1,12) | Piecewise linear dynamical system. |
| 9 | (8,1,7) | A platoon of two cars with controller maintaining distance. |

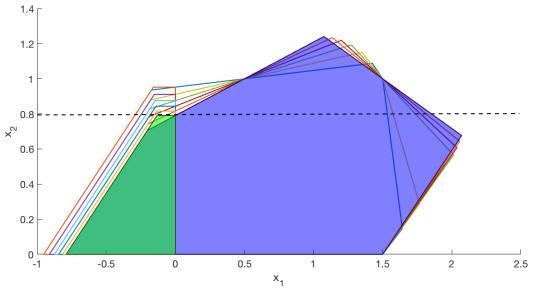


**Fig. 5** Sequence of iterates for benchmark id 2 culminating in the final invariants shown shaded in blue and green. The property $x_2 \geq 0.8$ is shown unreachable at the green location by the final iterate.

$U_\ell(i,j) = z$ for a fixed constant $z > 0$ (taken as 1000 in our experiments). For $T_\ell(i,j) \neq 0$, we allow $\Delta$ to range between $\frac{1}{2}T_\ell(i,j)$ and $2T_\ell(i,j)$ in our experiments.

**Benchmark Examples:** We consider a set of 9 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 and policy iteration with template update. Likewise, the performance for policy iteration without template update is shown in Table 3. Finally, Table 4 shows a comparison with the polyhedral abstract domain using the Parma Polyhedron Library (PPL) [9,8].

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 5 out of the 9 cases, whereas without template update we prove the property in just 2 out of 9. The polyhedral domain proves 3 out of the 9.

**Table 2** Experimental results for policy iteration **with** template update. 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: Succ.**: whether the property was successfully proved, if not, the objective value is reported, |BOP|: size of the bilinear problem (# bilinear template variables, # bilinear mult. variables, # linear mult. variables), # I: # policy iterations - A (*) next to this number indicates that the iteration was stopped due to 5 consecutive steps with same objective value. **TO** indicates time out of 1 hour.

| ID | Init. Templ. | Kleene | | Policy Iteration | | | |
|---|---|---|---|---|---|---|---|
| | Type, $\|T\|$ | Time | Succ. | \|BOP\| | Time | # I | Succ. |
| 1 | Interval, 8 | 0.12 | N(0.2) | (240, 1176, 1249) | 0.55 | 5(*) | N (0.2) |
| 3 | Octagon, 8 | 0.04 | N(0.5) | (24, 72, 161) | 0.05 | 2 | Y |
| 4 | Interval, 4 | 0.02 | N(15.5) | (12, 20, 33) | 0.02 | 2 | Y |
| 5 | Pentagon, 10 | 1.5 | N(2.83) | (40, 410, 681) | 0.3 | 2 | Y |
| 6 | Interval, 4 | 2.5 | N(0.75) | (168, 836, 2033) | 2.9 | 5(*) | N(0.75) |
| 7 | Pentagon, 22 | 3.1 | N(0.2) | (500, 10020, 8332) | 2.4 | 3 | Y |
| 8 | Pentagon, 22 | 2.6 | N(43) | (126, 5313, 6311) | **TO** | - | N |
| 9 | Interval, 16 | 2.2 | N(9500) | (286,4758,9501) | 2.4 | 2 | Y |

**Table 3** Experimental results for policy iteration **without** template update. See Table 2 for the legend.

| ID | Init. Templ. | Kleene | | Policy Iteration | | | |
|---|---|---|---|---|---|---|---|
| | Type, $\|T\|$ | Time | Succ. | \|BOP\| | Time | # I | Succ. |
| 1 | Interval, 8 | 0.12 | N (0.2) | (52,1176, 1249) | 0.19 | 2 | N(0.2) |
| 2 | Octagon, 8 | 0.15 | N (0.2 ) | (16, 264, 353) | 0.1 | 2 | N(0.2) |
| 3 | Octagon, 8 | 0.04 | N(0.5) | (8, 72, 161) | 0.02 | 1 | N(0.5) |
| 4 | Interval, 4 | 0.02 | N(15.5) | (4,20,33) | 0.01 | 1 | N(15.5) |
| 5 | Pentagon, 10 | 1.5 | N(2.83) | (10, 410, 681) | 0.3 | 2 | Y |
| 6 | Interval, 4 | 2.5 | N(0.75) | (140, 836, 2033) | 1.5 | 5(*) | N(0.75) |
| 7 | Pentagon, 22 | 3.1 | N(0.2) | (110, 10020, 8322) | 15.3 | 5(*) | N(0.2) |
| 8 | Pentagon, 22 | 2.6 | N(43) | (22, 5313, 6311) | 16.9 | 5(*) | N(38.8) |
| 9 | Interval, 16 | 2.2 | N(9500) | (16, 4758, 9501) | 2.3 | 2 | Y |

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 $\ell_2$. The final invariant for $\ell_2$ 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.

**Table 4** Experimental results using the polyhedral abstract domain and comparison of outcome against policy iter with template change in column TC (recalled from Tab. 2).

| ID | TIME(S) | SUCC. | TC |
|---|---|---|---|
| 1 | 0.6 | N | N |
| 2 | 0.03 | Y | Y |
| 3 | 0.02 | Y | Y |
| 4 | 0.02 | Y | Y |
| 5 | 0.3 | N | **Y** |
| 6 | 1.7 | N | N |
| 7 | 2.7 | N | **Y** |
| 8 | TO $> 1h$ | N | N |
| 9 | TO $> 1h$ | N | **Y** |

## 7 Bilinear Programming Problems

Thus far, we have studied the problem of abstract interpretation of programs using the template domain. As noted earlier, the data flow constraints characterizing the fixed points of the problem correspond to a class of non-convex optimization problems known as *bilinear programs* (BLP). In this section, we study the basic structure of the bilinear programs, abstracting from the structure of the optimization problem presented in Figure 4.

Our problem of interest is the following generic Bilinear Program (BLP):

$$\begin{array}{ll} \min\limits_{\mathbf{x},\mathbf{y}} & \mathbf{x}^T C_0 \mathbf{y} + \mathbf{a}_0^T \mathbf{x} + \mathbf{b}_0^T \mathbf{y} + \mathbf{c}_0^T \mathbf{z} \\ \text{subject to} & \mathbf{x}^T C_i \mathbf{y} + \mathbf{a}_i^T x + \mathbf{b}_i^T y + \mathbf{c}_i^T \mathbf{z} \leq \mathbf{r}_i \quad i \in \mathcal{I} \\ & \mathbf{x}^T C_j \mathbf{y} + \mathbf{a}_j^T \mathbf{x} + \mathbf{b}_j^T \mathbf{y} + \mathbf{c}_j^T \mathbf{z} = \mathbf{r}_j \quad j \in \mathcal{E} \\ & \mathbf{l}_x \leq \mathbf{x} \leq \mathbf{u}_x,\ \mathbf{l}_y \leq \mathbf{y} \leq \mathbf{u}_y,\ \mathbf{l}_z \leq \mathbf{z} \leq \mathbf{u}_y \end{array} \tag{14}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{z} \in \mathbb{R}^k$ and $\mathbf{l}_x, \mathbf{u}_x, \mathbf{l}_y, \mathbf{u}_y, \mathbf{l}_z, \mathbf{u}_z$ represent lower and upper bounds on $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$, respectively. $\mathcal{I}$ and $\mathcal{E}$ are the inequality and equality index sets for the constraints, respectively.

The bilinear program in (14) is *separable*. By fixing the variables $\mathbf{x}$ to definite values, we obtain a linear program over the remaining variables $(\mathbf{y}, \mathbf{z})$, and likewise, fixing the variables $\mathbf{y}$, we obtain a LP over $(\mathbf{x}, \mathbf{z})$.

**Lemma 8** *The dataflow constraints obtained in Figure 4 form a bilinear program (BLP) with the variables* $\mathbf{x}$ *denoting the template variables* $\Delta_\ell$ *and* $\mathbf{c}(\ell)$ *for* $\ell \in \mathcal{L}$ *and the* $\mathbf{y}$ *denoting the "bilinear multiplier variables"* $\Lambda_\tau$ *and* $\mathbf{z}$ *representing the "linear multiplier variables"* $\boldsymbol{\lambda}_s, \Lambda_0, \Gamma_\tau$.

It is easy to see that the feasible region for a BLP is nonconvex in general. Also, finding if a BLP is feasible is NP-hard.

**Theorem 5** *Checking if there exists* $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathbb{R}^{n+m+k}$ *that satisfy the constraints of a BLP* (14) *is NP-hard.*

*Proof* We reduce from 3-CNF SAT problem that consists of $n$ Boolean variables $p_1, \ldots, p_n$ and $m$ clauses, each clause $C_i$ consisting of a subset of literals $C_i \subseteq \{p_1, \ldots, p_n, \overline{p_1}, \ldots, \overline{p_n}\}$.

We introduce variables $x_i, y_i$ corresponding to each proposition $p_i$, with the goal that $x_i = 1, y_i = 0$ will denote assigning $p_i$ to *true* and $x_i = 0, y_i = 1$ will denote assigning $p_i$ to *false*. To ensure that these are the only possible values, we write the constraints

$$\begin{aligned} x_i y_i &\leq 0, \\ x_i + y_i &= 1, \\ x_i &\in [0,1] \\ y_i &\in [0,1] \end{aligned} \tag{15}$$

Note that the constraint $x_i y_i \leq 0$ is a bilinear inequality. The remaining are linear inequalities. Also, note that the only feasible solutions are $(x_i, y_i) \in \{(0,1),(1,0)\}$.

Next, for each clause $C_j$ involving some literals, we add the constraint

$$\sum_{p_i \in C_j} x_i + \sum_{\overline{p_i} \in C_j} y_i \geq 1\,. \tag{16}$$

Now consider the system of constraints obtained by collecting (15) for $i \in \{1, \ldots, n\}$ and (16) for $j \in \{1, \ldots, m\}$. We can verify that the system forms the constraints for a BLP with $\mathbf{x} : (x_1, \ldots, x_n)^T$, $\mathbf{y} : (y_1, \ldots, y_n)^T$ and $\mathbf{z}$ as the empty vector.

Next, we note that for any satisfying assignment to the original problem, the corresponding assignment to $(x_i, y_i)$ as described above will satisfy the constraints for the BLP. Similarly, we have proven that if the BLP is satisfiable, then for each $(x_i, y_i)$, we can set the corresponding proposition $p_i =$ *true* if $x_i = 1$ and *false* otherwise. Due to constraint (16), each clause has at least one literal that will be true.

This completes the reduction and the proof of NP-hardness.

### 7.1 Policy Iteration: Alternating Minimization

We have already observed that if we could fix $\mathbf{x}$ to concrete values in the BLP (14), we obtain an LP in terms of the remaining variables $(\mathbf{y}, \mathbf{z})$ and likewise for $\mathbf{y}$. A simple strategy is to perform alternating minimization wherein, we find an initial feasible solution $(\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$ and improve the current solution $(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$ at each iteration by carrying out two steps: (a) Fixing $\mathbf{x} : \mathbf{x}_i$, solve the resulting LP to obtain $(\mathbf{y}_{i+1}, \mathbf{z})$; and (b) fixing $\mathbf{y} : \mathbf{y}_{i+1}$, solve the resulting LP to obtain $(\mathbf{x}_{i+1}, \mathbf{z}_{i+1})$. Combining the two steps, we obtain the next solution $(\mathbf{x}_{i+1}, \mathbf{y}_{i+1})$. We continue this process until no more improvements are possible. There are two basic questions posed by this algorithm: (a) does it terminate? and (b) if it terminates, then how does the result compare with the optimal solution to the problem?

First, it is easy to formulate an example that illustrates non-termination.

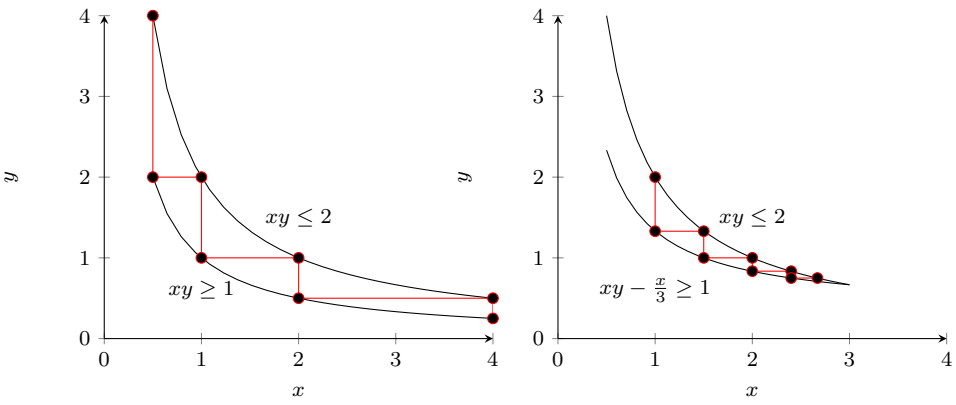


**Fig. 6** Examples showing nontermination of alternating minimization.

*Example 3 (Nontermination of Alternating Minimization)* Consider the problem below illustrated in Figure 6 (left).

$$\begin{array}{ll} \max\ x - y & \\ \quad xy & \geq 1 \\ \quad xy & \leq 2 \\ \quad x, y & \geq 0 \end{array} \tag{17}$$

Starting with the initial solution $(x_0 = 1, y_0 = 1)$, we note that the sequence of iterates obtained using alternating minimization are

$$(x_1 = 2, y_1 = \frac{1}{2}), (x_2 = 4, y_2 = \frac{1}{4}), \cdots, (x_j = 2^j, y_j = 2^{-j}) \cdots$$

The process continues forever without termination with each LP producing a bounded result although the original BLP is unbounded.

In fact, alternating minimization may fail to terminate even if the feasible region of the BLP is bounded.

*Example 4 (Nontermination of Alternating Minimization: Bounded Case)* Consider the problem below illustrated in Figure 6 (right).

$$\begin{array}{ll} \max\ x - y & \\ \quad xy & \leq 2 \\ \quad x(y - \frac{1}{3}) & \geq 1 \\ \quad x, y & \geq 0 \end{array} \tag{18}$$

The reader may verify that starting with $(x_0 = 1, y_0 = 1)$, the approach iterates to obtain $(x_{i+1} = \frac{6x_i}{3+x_i},\ y_{i+1} = \frac{3+x_i}{3x_i})$. As $i \to \infty$, the process converges to $x^* = 3, y^* = \frac{2}{3}$, which can be verified as the optimal solution to the problem by graphing the constraints.

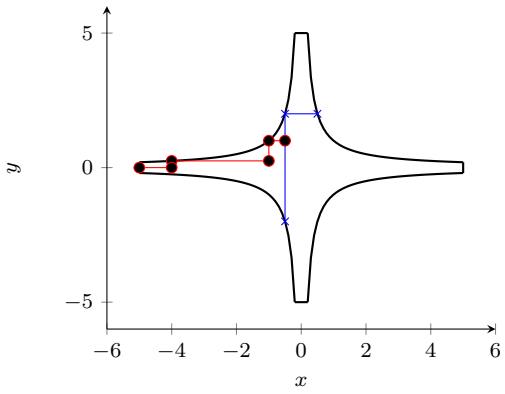


**Fig. 7** BLP with saddle point at $(-1, 1)$. The red path shows how an interior point solver can solve the problem whereas the simplex approach shown in blue can be stuck in a saddle point.

Finally, we illustrate so-called saddle points, wherein the process converges $(x_{i+1}, y_{i+1}) = (x_i, y_i)$, but the solution is not a KKT point (a local optimum) of the original BLP.

*Example 5 (Saddle Points)* Consider the problem instance shown below illustrated in Figure 7.

$$\begin{aligned} \min_{x,y} \quad & x \\ \text{subject to} \quad & xy \leq 1 \\ & xy \geq -1 \\ & -5 \leq x \leq 5 \\ & -5 \leq y \leq 5 \end{aligned} \tag{19}$$

The point $(-1, 1)$ is a saddle point of the problem. Fixing $x = -1$ yields the optimization problem

$$\min\ 0 \ \text{s.t.}\ -1 \leq y \leq 1\,, \tag{20}$$

which can yield either $y = -1$ or $y = 1$ if a Simplex-based LP solver is used. Assuming that the solver results in $y = 1$, we obtain no further change in the values $(x, y)$. If $y = -1$ were chosen by the solver, instead, the very next step yields the desired saddle point. The point $(-1, 1)$ is not a local optimum for the problem. For instance the descent direction $(\Delta x, \Delta y) : (-1, 1)$ can maintain feasibility while providing a step that decreases the objective value. Unfortunately, alternating minimization is unable to find the descent direction since it is restricted to finding directions with $\Delta x = 0$ or $\Delta y = 0$. Unfortunately, such directions do not exist at the saddle point.

We also note that if a simplex solver were not used, it is theoretically possible that an "interior point" solver can choose $y = 0$ as the solution to

the LP (20). Subsequently, we can solve for $x$ and obtain a globally optimal solution.

In the context of saddle point, the key problem is that of moving the iteration past the saddle point to continue improving the solution? This problem has remained mostly open thus far. We summarize some of the difficulties encountered. First, we note that alternating minimization is a form of *coordinate descent* wherein the solution improvement direction for the problem in (14), wherein starting from a current solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ that is feasible, we seek a new solution $(\mathbf{x}+\Delta_x, \mathbf{y}+\Delta_y, \mathbf{z}+\Delta_z)$, such that (a) $(\mathbf{x}+\Delta_x, \mathbf{y}+\Delta_y, \mathbf{z}+\Delta_z)$ is feasible and

$$f(\mathbf{x}+\Delta_x, \mathbf{y}+\Delta_y, \mathbf{z}+\Delta_z) < f(\mathbf{x}, \mathbf{y}, \mathbf{z}),$$

wherein $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) : \mathbf{x}^T C_0 \mathbf{y} + \mathbf{a}_0^T \mathbf{x} + \mathbf{b}_0^T \mathbf{y} + \mathbf{c}_0^T \mathbf{z}$ is the objective function shown for the problem (14). However, it is immediate that the search for such feasible directions $(\Delta_x, \Delta_y, \Delta_z)$ requires solving a bilinear problem with a similar structure as the original problem (14) in the first place. Thus, linearity is forced through coordinate descent wherein either $\Delta_x = 0$ or $\Delta_y = 0$. Thus, the problem of "escaping" from local saddle points is an open problem that limits alternating minimization approaches.

### 7.2 Bilinear Problems and QCQPs

There exists a natural encoding of BLPs to Quadratically-Constrained Quadratic Programs (QCQPs) of a higher dimensional space. That is, any BLP can be reformulated as the QCQP and numerous approaches to solving QCQPs can be applied to BLPs, as well. Specifically, we can encode (14) as the QCQP

$$\begin{array}{lll} \min\limits_{\tilde{\mathbf{x}}} & \tilde{\mathbf{x}}^T \widetilde{C}_0 \tilde{\mathbf{x}} + \tilde{\mathbf{a}}_0^T \tilde{\mathbf{x}} & \\ \text{subject to} & \tilde{\mathbf{x}}^T \widetilde{C}_i \tilde{\mathbf{x}} + \tilde{\mathbf{a}}_i^T \tilde{\mathbf{x}} \leq \mathbf{r}_i & i \in \mathcal{I} \\ & \tilde{\mathbf{x}}^T \widetilde{C}_i \tilde{\mathbf{x}} + \tilde{\mathbf{a}}_i^T \tilde{\mathbf{x}} = \mathbf{r}_i & i \in \mathcal{E} \\ & \tilde{\mathbf{l}} \leq \tilde{\mathbf{x}} \leq \tilde{\mathbf{u}} & \end{array} \tag{21}$$

where $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{pmatrix} \in \mathbb{R}^{m+n}, \widetilde{C}_0 = \frac{1}{2} \begin{pmatrix} 0 & C_0 & 0 \\ C_0^T & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, $\tilde{\mathbf{a}}_0 = \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{b}_0 \\ \mathbf{c}_0 \end{pmatrix}$, $\tilde{\mathbf{l}} = \begin{pmatrix} \mathbf{l}_x \\ \mathbf{l}_y \\ \mathbf{l}_z \end{pmatrix}$, $\tilde{\mathbf{u}} = \begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix}$, and $\forall i \in \mathcal{I} \cup \mathcal{E}$, $\widetilde{C}_i = \frac{1}{2} \begin{pmatrix} 0 & C_i & 0 \\ C_i^T & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, $\tilde{\mathbf{a}}_i = \begin{pmatrix} \mathbf{a}_i \\ \mathbf{b}_i \\ \mathbf{c}_i \end{pmatrix}$. Note that in the QCQP, the matrices $\widetilde{C}_0$ and $\widetilde{C}_i$ are symmetric, whereas the matrices $C_0$ and $C_i$ in the BLP may not even be square.

Conversely, for any QCQP there exists an encoding into a BLP. We can see this by replacing all terms of the form $\tilde{\mathbf{x}}^T \widetilde{C}_i \tilde{\mathbf{x}}$ with $\tilde{\mathbf{x}}^T \widetilde{C}_i \tilde{\mathbf{y}}$ and embedding the constraint $\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$ into the problem. However, the resulting BLP has equality

constraints $\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$. Although it seemingly maintains the "separability" property, it is immediately clear that solving it via policy iteration (also referred to as "alternating minimization" in this section) is useless; that is, if we fix $\tilde{\mathbf{x}}$ and solve for $\tilde{\mathbf{y}}$, the constraint $\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$ means the method stagnates.

Rewriting the BLP as a QCQP in order to solve it numerically is not always a good idea, since the new QCQP is much larger, but it motivates us to consider some particular QCQP methods and their application to BLPs. Notice that there is no other requirement on $\widetilde{C}_i$ other than symmetry.

If we further assume that each $\widetilde{C}_i$ is positive semidefinite for $i \in \{0\} \cup \mathcal{I}$, and $\widetilde{C}_i = 0$ for $i \in \mathcal{E}$, then the problem is convex. This is generally not the case for problems arising from program analysis, and therefore, we do not assume our QCQPs are convex.

Because neither the general QCQPs nor BLPs are convex, most nonlinear solvers can only produce a stationary point, which may be either a local minimizer (and possibly a global minimizer) or a saddle point. While ideally one wants a global rather than local minimizer, a secondary goal is to find a local minimizer rather than a saddle point. We focus on this second problem, and note that it has been the subject of much recent attention in machine learning, where it is sometimes argued that all local minimizers are "good enough" and thus one only needs to avoid saddle points [27].

Various relaxations for solving nonconvex QPs have been proposed including lift-and-project cutting plane algorithms, reformulation-linearization techniques (RLT), semi-definite programming (SDP) relaxations, matrix-cut algorithms, and semi-infinite linear programs [7,44,53,70,64,52].

**Convex Relaxation Techniques:** Relaxations for QCQPs based on SDPs or RLTs lift the original problem into a higher dimension by implementing a nonlinear change of variables that puts all the nonconvexity into a single constraint, and then drops the nonconvex constraint in order to relax the original problem to a convex program. Both the SDP and RLT relaxations are invariant to an invertible affine transformation of the original variables. One of the commonly used change of variables is $\mathbf{x} \to \mathbf{x}\mathbf{x}^T = X$, lifting the problem from $\mathbf{x} \in \mathbb{R}^n$ to $X \in \mathbb{S}^n_+$, and replacing quadratic terms with matrix inner products using the trace operator: $\mathbf{x}^T C \mathbf{x} = \mathrm{Tr}(\mathbf{x}^T C \mathbf{x}) = \mathrm{Tr}(C\mathbf{x}\mathbf{x}^T) = C \cdot X$.

$$\begin{aligned} \min_{\tilde{\mathbf{x}}, X} \quad & \widetilde{C}_0 \cdot X + \tilde{\mathbf{a}}_0^T \tilde{\mathbf{x}} \\ \text{subject to} \quad & \widetilde{C}_i \cdot X + \tilde{\mathbf{a}}_i^T \tilde{\mathbf{x}} \le \mathbf{r}_i \quad i \in \mathcal{I} \\ & C_i \cdot X + \tilde{\mathbf{a}}_i^T \tilde{\mathbf{x}} = \mathbf{r}_i \quad i \in \mathcal{E} \\ & X = \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T \\ & \tilde{\mathbf{l}} \le \tilde{\mathbf{x}} \le \tilde{\mathbf{u}} \end{aligned} \tag{22}$$

Along with the constraints $X = \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$, this reformulation does not change the problem, and is still nonconvex, though all terms except the $X = \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$ constraint are linear. To get a relaxation, the equality constraint $X = \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$ is relaxed to $X \succeq \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$ where $\succeq$ is the Loewner ordering where $Y \succeq 0$ means

$Y$ is a symmetric positive semi-definite matrix. This new inequality defines a convex set, and the relaxation is a convex program and in particular a SDP.

The reformulation-linearization technique follows the same procedure of employing auxiliary variables in place of the product terms, $\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_j = X_{ij}$, but rather than imposing a positive semidefinite constraint, RLTs add the product of bound-factors $(\tilde{\mathbf{u}}_i - \tilde{\mathbf{x}}_i)(\tilde{\mathbf{x}}_j - \tilde{\mathbf{l}}_j) \geq 0$ for any or all pairs of variables $\tilde{\mathbf{x}}_i,\ i \in \{1, \ldots, n\}$, where $\tilde{\mathbf{u}}_i$ and $\tilde{\mathbf{l}}_j$ denote the *finite* upper and lower bounds on variables $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_j$, respectively. Furthermore, RLT constraints can include the products of equality constraints with monomials $\mathit{\Pi}_j\tilde{\mathbf{x}}_j$ and inequality constraints with themselves as well as the bound-factors mentioned above. While these extra constraints may be redundant, after a round of reformulation and linearization they lead to a tighter convex relaxation.

Previous work on combining RLTs and SDP relaxations to remove a substantial portion of the feasible region was explored in [6] with provable results. Sherali et al. [64] investigated the effects of enhancing the RLT formulations with semi-definite cuts on QPs without quadratic constraints. They verify empirically that combining certain aspects from each formulation leads to better results. Other convex relaxations for specific problem structures, including Second-Order Conic Programs, have been proposed to speed up computation while preserving accuracy [41,63].

In addition to SDP and RLT relaxations, McCormick Envelopes provide a convex relaxation for bilinear programs commonly used to solve Mixed Integer NonLinear Programs (MINLPs). They are designed to guarantee convexity while keeping the bounds on the original problem sufficiently tight. The method behind McCormick Envelopes is to replace distinct products of variables, i.e. bilinear terms, with auxiliary variables and append bound constraints which form convex over- and under-estimators of the bilinear terms. The essence of the method is similar to that of the reformulation-linearization technique; however, the goal of McCormick Envelopes is to replace all bilinear terms with appropriately bounded auxiliary variables whereas the goal of RLTs is to append additional, potentially redundant, constraints in order to minimize the set of candidate solutions to the relaxed problem.

In the context of proving program properties, relaxation-based approaches are less useful since they focus on proving lower bounds for the optimal value of a minimization problem. Therefore, in the setting of this paper, it is possible to use relaxation methods to establish that no linear invariant involving a fixed number of conjunctions can prove a property. However, unless the relaxation is exact, these methods do not directly find invariants to establish a property since the solution obtained by the relaxation may not be feasible.

### 7.3 Augmented Lagrangian Approaches

Augmented Lagrangian methods are a promising approach to solving constrained optimization problems through a series of related unconstrained problems [50]. The constraints are incorporated into the objective function via a

quadratic penalty plus a pseudo-Lagrangian term. By increasing the effect of the penalty parameter, the sequence of iterates tend to a local solution of the original constrained problem.

We now present a method to solve BLPs using the Augmented Lagrangian framework. Augmented Lagrangian methods naturally handle equality constraints. To handle inequality constraints, there are several variants, and we follow the variant that converts inequality constraints into equality constraints via a slack variable $s$, but keeps all box constraints (including those on the slack) explicit. We reformulate (14) as follows

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y},\mathbf{s}} \quad & \underbrace{\mathbf{x}^T C_0 \mathbf{y} + \mathbf{a}_0^T \mathbf{x} + \mathbf{b}_0^T \mathbf{y}}_{f(\mathbf{x},\mathbf{y})} \\ \text{subject to} \quad & \underbrace{\mathbf{x}^T C_i \mathbf{y} + \mathbf{a}_i^T \mathbf{x} + \mathbf{b}_i^T \mathbf{y} - r_i}_{c_i(\mathbf{x},\mathbf{y})} + s_i = 0 \quad i \in \mathcal{I} \cup \mathcal{E} \\ & \mathbf{l}_x \leq \mathbf{x} \leq \mathbf{u}_x, \quad \mathbf{l}_y \leq \mathbf{y} \leq \mathbf{u}_y, \\ & s_i \geq 0 \; \forall i \in \mathcal{I}, \quad s_i = 0 \; \forall i \in \mathcal{E} \end{aligned} \tag{23}$$

For convenience, the $\mathbf{z}$ variables are assumed to be nonexistent in our exposition. However, the approach described can be readily extended to include linear $\mathbf{z}$ variables, as well. We will denote the box constraints concisely as $(\mathbf{x},\mathbf{y},\mathbf{s}) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{S}$. Since $c_i(\mathbf{x},\mathbf{y}) + s_i = 0$ for any feasibly points, we can see that for any $\mu > 0$, (23) is equivalent to

$$\begin{aligned} \min_{(\mathbf{x},\mathbf{y},\mathbf{s}) \in \mathcal{X}\times\mathcal{Y}\times\mathcal{S}} \quad & f(\mathbf{x},\mathbf{y},\mathbf{z}) + \frac{\mu}{2} \sum_{i \in \mathcal{E}\cup\mathcal{I}} \left(c_i(\mathbf{x},\mathbf{y},\mathbf{z}) + s_i\right)^2 \\ \text{subject to} \quad & c_i(\mathbf{x},\mathbf{y},\mathbf{z}) + s_i = 0 \quad i \in \mathcal{I} \cup \mathcal{E} \end{aligned} \tag{24}$$

The Augmented Lagrangian method then applies subgradient ascent with respect to the dual variable $\boldsymbol{\lambda}$ on the Lagriangian of (24). Starting at some $\boldsymbol{\lambda}_0 \in \mathbb{R}^{|\mathcal{I}\cup\mathcal{E}|}$, and defining the augmented Lagrangian $\mathcal{L}_A$ as

$$\mathcal{L}_A(\mathbf{x},\mathbf{y},\mathbf{s};\boldsymbol{\lambda}) \stackrel{\text{def}}{=} f(\mathbf{x},\mathbf{y}) - \sum_{i \in \mathcal{E}\cup\mathcal{I}} \lambda_i \cdot \left(c_i(\mathbf{x},\mathbf{y}) + s_i\right) + \frac{\mu}{2} \sum_{i \in \mathcal{E}\cup\mathcal{I}} \left(c_i(\mathbf{x},\mathbf{y}) + s_i\right)^2$$

the method iterates for $k = 1, 2, \ldots$,

$$(\mathbf{x}^k, \mathbf{y}^k, \mathbf{s}^k) \leftarrow \operatorname*{argmin}_{(\mathbf{x},\mathbf{y},\mathbf{s}) \in \mathcal{X}\times\mathcal{Y}\times\mathcal{S}} \mathcal{L}_A(\mathbf{x},\mathbf{y},\mathbf{s};\boldsymbol{\lambda}^k) \tag{25}$$

$$\boldsymbol{\lambda}_i^{k+1} \leftarrow \boldsymbol{\lambda}_i^k - \mu \left(c_i(\mathbf{x}^k,\mathbf{y}^k) + s_i^k\right) \tag{26}$$

While the Augmented Lagriangian method has been applied generically to non-convex non-linear programs, as in the `PENNON` package [42], here we can specialize it to take advantage of the structure of BLP problems. Note that (25) can be written as

$$\min_{\mathbf{x}\in\mathcal{X}} \underbrace{\left( \min_{(\mathbf{y},\mathbf{s}) \in \mathcal{Y}\times\mathcal{S}} \mathcal{L}_A(\mathbf{x},\mathbf{y},\mathbf{s};\boldsymbol{\lambda}^k) \right)}_{\varphi(\mathbf{x})} \tag{27}$$

and evaluating $\varphi(\mathbf{x})$—that is, finding the minimizing points $(y_x, s_x)$—requires solving a *convex* quadratic program, which can be done efficiently (our implementation uses the `MATLAB` software `CVX` and `Gurobi`). Furthermore, $\nabla\varphi$ can be calculated using the following lemma. This lemma is similar to Danskin's Theorem [11] but the proof is distinct since we minimize rather than maximize, and do not assume compact constraints.

**Lemma 9** *Assume* $(\mathbf{y}_x, \mathbf{s}_x)$ *are the unique minimizers of* $\mathcal{L}_A(\mathbf{x}, \mathbf{y}, \mathbf{s}; \boldsymbol{\lambda}^k)$ *over* $(\mathbf{y}, \mathbf{s}) \in \mathcal{Y} \times \mathcal{S}$ *and* $\mathbf{l}_x, \mathbf{u}_x, \mathbf{l}_y, \mathbf{u}_y$ *are real-valued (i.e, they do not have* $\pm\infty$ *entries) . Then* $\nabla\varphi(\mathbf{x}) = \nabla_x \mathcal{L}_A(\mathbf{x}, \mathbf{y}_x, \mathbf{s}_x; \boldsymbol{\lambda}^k)$.

*Proof* The main tool is Thm. 10.58 in [54]. Let $\varphi(\mathbf{x}) = \min_{(\mathbf{y},\mathbf{s})} \mathcal{L}_A(\mathbf{x}, \mathbf{y}, \mathbf{s}; \boldsymbol{\lambda}^k)$ and $\Phi(\mathbf{x}) = \operatorname{argmin}_{(\mathbf{y},\mathbf{s})} \mathcal{L}_A(\mathbf{x}, \mathbf{y}, \mathbf{s}; \boldsymbol{\lambda}^k)$ denote the optimal value and the optimal solution set, respectively. Since $\mathcal{L}_A(\mathbf{x}, \mathbf{y}, \mathbf{s}; \boldsymbol{\lambda}^k)$ is continuous, proper, and level-set bounded in $(\mathbf{y}, \mathbf{s})$ uniformly locally in $\mathbf{x}$, $\varphi(\mathbf{x})$ is proper, and continuous and for each $\mathbf{x} \in \operatorname{dom}(\varphi)$ the set of minimizers $\Phi(\mathbf{x}) = \{(\mathbf{y}_x, \mathbf{s}_x)\}$ is nonempty and compact. Moreover, there exists a compact neighborhood $B$ of $\mathbf{x}$ such that the image $\mathbf{x}$ under $\Phi$ is a compact set for all $\mathbf{x} \in B$. For each $(\mathbf{y}, \mathbf{s}) \in \Phi(B)$ the function $\mathcal{L}_{A(\mathbf{y},\mathbf{s})} := \mathcal{L}_A(\cdot, \mathbf{y}, \mathbf{s})$ is continuously differentiable on the interior of $B$. Let $\mathcal{O} := \operatorname{int}(B)$. Then $\varphi(\mathbf{x}) = \min_{(\mathbf{y},\mathbf{s})\in\mathcal{Y}\times\mathcal{S}} \mathcal{L}_{A(\mathbf{y},\mathbf{s})}(\mathbf{x})$ is continuously differentiable on $\mathcal{O}$.

Level-set bounded in $(\mathbf{y}, \mathbf{s})$ uniformly locally in $\mathbf{x}$ means that for each $\mathbf{x} \in \mathcal{X}$ and $\alpha \in \mathbb{R}$ there is a neighborhood $V$ of $\mathbf{x}$ and bounded set $B \subset \mathcal{Y} \times \mathcal{S}$ such that the level-sets $\{(\mathbf{y}, \mathbf{s}) | \mathcal{L}_A(\mathbf{x}, \mathbf{y}, \mathbf{s}; \boldsymbol{\lambda}^k) \leq \alpha\} \subset B$ for all $\mathbf{x} \in V$, that is, the level-sets are bounded. The uniform level boundedness assumption allows us to find a neighborhood $V$ of $\mathbf{x}$ satisfying the criteria for $\varphi(\mathbf{x})$ to be lower semi-continuous which is a weaker statement than continuity, but sufficient for the original proof of Thm. 10.58 in [54].

Given a means to calculate $\nabla\varphi(\mathbf{x})$, we can minimize $\varphi$ over $\mathbf{x} \in \mathcal{X}$ using gradient-based method such as L-BFGS-B [16]. Our motivation for pursuing gradient-based approaches, rather than alternating minimization approaches, is due to a very recent body of literature showing that gradient descent is (under certain conditions) "unlikely" to be trapped at saddle points. In particular, Jin et al [40] show that small infrequent random perturbations are sufficient to prevent gradient descent from converging to a saddle point, while [43] shows that random initializations prevent convergence to a saddle point. This formulation allows for bilinear optimization where $\mathbf{x}$ and $\mathbf{y}$ can be treated separately without the disjoint treatment of alternating minimization.

## 8 Solvers Used for Empirical Comparisons

In this section, we review a set of solvers beyond alternating minimization that are used as a basis of comparison over the benchmark problems discussed in Section 6. A brief description of each tool is provided. We used our own implementations of alternating minimizations and augmented Lagrangian methods

| Solver | Solver Type |
|---|---|
| Alternating Minimization | Local |
| FMINCON | Local |
| BONMIN | Global |
| COUENNE | Global |
| IPOPT | Local |
| BARON | Global |
| Augmented Lagrangian | Local |

**Table 5** Solvers used in the numerical experiments.

as described subsequently. All the solvers are implemented using floating point arithmetic and thus the soundness of the results obtained from these solvers in the context of abstract interpretation is an issue that is not addressed here. However, if a solution approach is deemed promising, then approaches to surmount the issues caused by floating point arithmetic will be well motivated for future work.

### 8.1 Alternating Minimization

The implementation described in section 6 used an exact arithmetic LP solver. Here we examined two efficient floating point LP solvers for alternating minimization. One approach used the dual simplex solver implemented inside the Gurobi tool. Another approach used a floating point primal dual interior point method using the Sedumi solver.

### 8.2 FMINCON

`fmincon` is a built-in `MATLAB` nonlinear programming solver. It attempts to return feasible stationary points to problems of the form

$$\begin{aligned} \min_{x} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & c(\mathbf{x}) \leq 0 \\ & c_{\text{eq}}(\mathbf{x}) = 0 \\ & A_i\mathbf{x} \leq \mathbf{b}_i \\ & A_e\mathbf{x} = \mathbf{b}_e \\ & \mathbf{l}_x \leq \mathbf{x} \leq \mathbf{u}_x \end{aligned}$$

where both the objective and constraints can be nonlinear functions. The method makes no attempt to find a global optimal point. In order to apply `fmincon` to (14), we reformulate the problem into (21) as mentioned above. Since the problems we will be comparing are sparse, with a few examples being large scale, we use the solver's primal-dual interior-point algorithm and supply the gradient function. Other supplied algorithms include Trust Region Reflective, Active Set, and Sequential Quadratic Programming; however, Active Set

and SQP are not large scale algorithms and the problems considered do not meet the required conditions to apply the Trust Region Reflective algorithm.

### 8.3 BONMIN

`BONMIN` (**B**asic **O**pen-source **N**onlinear **M**ixed **IN**teger programming) is an open source software for solving mixed-integer non-linear programs (MINLPs) of the form

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 \quad i = 1 \dots m \\ & \mathbf{l}_x \leq x \leq \mathbf{u}_x \\ & \mathbf{x}_i \in \mathbb{Z}, \quad i \in I \subseteq \{1, \dots, n\} \\ & \mathbf{x}_i \in \mathbb{R}, \quad i \notin I \end{array} \tag{28}$$

where $f$ and $g$ are assumed to be twice continuously differentiable. The software is distributed by the Computational Infrastructure for Operations Research (COIN-OR). `BONMIN` is equipped with several nonlinear programming algorithms based on branch-and-bound, branch-and-cut, and outer-approximation techniques. The branch-and-bound approach considers polyhedral subsets of the state space and solves a continuous relaxation of the problem containing only a subset of the variables in search for an improvement on the lower bound of the approximation. The branch-and-cut technique uses the same approach to infer an optimal set of candidate solutions but further refines the estimate of the lower bound by implementing nonlinear lift-and-project cutting plane methods to tighten the corresponding relaxation. `BONMIN` uses COIN-ORs NLP interior-point solver `IPOPT`, which we introduce below, for its branch-and-bound algorithm. The outer-approximation method is a technique for recasting the original MINLP into an equivalent, and relaxed linear representation by removing any nonlinearities from the objective and adding their linearized variants to the constraint set. It is an iterative procedure that appends newly computed bound constraints found at each iteration to tighten the relaxation. It implements `IPOPT` to solve each NLP and COIN-ORs branch-and-cut algorithm `Cbc` to solve each MILP. The final algorithm offered by `BONMIN` is a hybrid outer-approximation based branch-and-cut algorithm. If either or both $f$ and $g$ are nonconvex then the algorithms are only meant to be heuristics [15]. Thus there is no guarantee of a global optimal point, but unlike `fmincon` which returns as soon as it has found an approximate stationary point, the `BONMIN` solver attempts to find a global optimal point. For all numerical comparisons in Section 9 we implemented each of `BONMIN`s four solvers and achieved similar results; therefore, the results reported are the results of the hybrid solver.

### 8.4 COUENNE

`COUENNE` (**C**onvex **O**ver and **U**nder **EN**velopes for **N**onlinear **E**stimation) is an open source software primarily developed by Pietro Belotti to solve MINLPs of the form (28), where $f$ and $g_i$ are possibly nonconvex [1]. The software is also distributed by COIN-OR. `COUENNE` is a RLT-based spatial branch-and-bound algorithm, meaning that the nonconvex terms are replaced with their convex envelopes and branching can occur on either integer or continuous variables. It ensures global optimal solutions of convex MINLPs and aims to find global optima of nonconvex MINLPS. `COUENNE` is built on top of `BONMIN` and utilizes many of the same options and solvers. More generally, it is a branch and cut algorithm that implements linearization, branching, MINLP heuristics to find feasible solutions, and bound reduction techniques [10]. We emphasize that the software attempts to find a global solution, and thus is naturally slower than methods like `fmincon`.

### 8.5 IPOPT

`IPOPT` (**I**nterior **P**oint **Opt**imizer) is an open source software package for large-scale nonlinear optimization that is distributed by COIN-OR. `IPOPT` supports the same general framework as `COUENNE`; however, it further requires that $f$ and all $g_i$ be sufficiently smooth (at least continuously differentiable). Moreover, `IPOPT` aims to find local solutions to (28), like `fmincon`, as opposed to global solutions like `COUENNE`. The `IPOPT` algorithm implements a primal-dual interior-point method that uses line searches based on filter methods. Filter methods offer an alternative to merit functions in that iterates are accepted if they either improve the objective function or improve the constraint violation as opposed to a combination of the two [68].

### 8.6 BARON

`BARON` (**B**ranch-**A**nd-**R**educe **O**ptimization **N**avigator ) is a global optimization software for solving MINLPs that was developed by Nikolaos Sahinidis at the University of Illinois-Urbana Champaign. It is a branch-and-bound algorithm that has the ability to employ other solvers to solve appropriate subproblems. Given a BLP, `BARON` generates McCormick Envelopes to relax and provide a lower bound for the original problem. Further measures are then taken to improve the accuracy of the computed variables by transforming the relaxed problem into an SDP. `BARON` also employs linear approximations, including outer-approximations, to speed up the computation of the relaxed problem. While this is only a high-level description that is by no means complete, it provides a brief explanation of the general methods employed by `BARON` for solving BLPs. `BARON` requires that all nonlinear variables and expressions be bounded from above and below to guarantee global optimality. If bounds are not provided, it utilizes appropriate algorithms to infer bounds [56, 66].

| Solver | Solver Type | Method(s) Employed for Experimental Results |
|---|---|---|
| Alt Min - GUROBI | Local | Dual-Simplex |
| Alt Min - SeDuMi | Local | Primal-Dual IP |
| FMINCON | Local | Primal-Dual IP |
| BONMIN | Global | Hybrid OA/BnC |
| COUENNE | Global | RLT-based Spatial BnB |
| IPOPT | Local | Primal-Dual IP |
| BARON | Global | McCormick Envelope/SDP |
| Augmented Lagrangian | Local | Gradient-Based/Dual-Simplex |

**Table 6** Solvers used in the numerical experiments.

## 9 Experimental Results

In this section, we apply the algorithms mentioned in the previous section to a series of benchmarks that include Example 2 discussed previously and benchmarks 2-5 in Table 1. Benchmarks 1 and 6-9 are not included in the numerical comparisons since the algorithms implemented do not scale well with the size of the problems. All algorithms were implemented in `MATLAB R2016b`. The COIN-OR source code and binaries provided for `COUENNE` and `IPOPT` have interfaces in `AMPL`, which can be interfaced through `MATLAB`. For the alternating minimization procedure we used the `MATLAB` software `CVX` [35,36] with the simplex solver `Gurobi` [51] and the interior-point solver `Sedumi` [65]. For each instance, we report the optimal objective reported by the solver along with the *feasibility gaps* that are given by the largest absolute violation of equality constraints and largest inequality constraint violation to deduce feasibility of the resulting solution.

**Dominant Eigenvector: A Toy Problem.**

As a toy example used to test the various approaches, we begin by computing the dominant eigenvector of a positive semidefinite operator, $C \in \mathbb{S}^4_+$. Computing the dominant eigenvector of a positive semidefinite operator can be phrased as the following optimization program:

$$\begin{array}{ll} \max\limits_{x} & x^T C x \\ \text{subject to} & ||x||_2 \leq 1, \end{array} \tag{29}$$

or equivalently,

$$\begin{array}{ll} \min\limits_{x} & -x^T C y \\ \text{subject to} & ||y||_2 \leq 1 \\ & x = y \end{array} \tag{30}$$

where $||\cdot||_2$ is the $\ell_2$-norm of a vector, $||x||_2 = \sqrt{\sum_i x_i^2}$. By imposing the additional constraint $x = y$ we can convert the quadratic objective of (29) to the bilinear one in (30). While this form does not match the general form of a BLP we can still employ the solvers listed above to get an idea of the

| Method | Status | Normalized Residual |
|---|---|---|
| FMINCON | Solved | 9.750e-09 |
| BONMIN | Solved | 4.999e-09 |
| COUENNE | Solved | 4.000e-03 |
| IPOPT | Solved | 3.156e-10 |
| BARON | Solved | 2.270e-05 |
| Aug Lag | Solved | 5.100e-05 |

**Table 7** Results of dominant eigenvector program (30).

| Method | Obj Value | $\max\{\lvert c_{\text{eq}}(x)\rvert\}$ | $\max\{c_{\text{ineq}}(x)\}$ | Status |
|---|---|---|---|---|
| Alt Min Gurobi | **1.000e+00** | **0.000e+00** | **0.000e+00** | Solved |
| Alt Min Sedumi | 1.970e+02 | **0.000e+00** | 1.421e-14 | Solved |
| FMINCON | **1.000e+00** | **0.000e+00** | **-1.603e-08** | Solved |
| BONMIN | **1.000e+00** | 2.675e-08 | 1.020e-06 | Solved |
| COUENNE | **1.000e+00** | 8.963e-09 | 5.108e-07 | Solved |
| IPOPT | **1.000e+00** | 1.749e-08 | 1.011e-06 | Solved |
| BARON | **1.000e+00** | 5.845e-09 | 7.437e-09 | Solved |
| Aug Lag | 2.75e+02 | 9.925e-07 | **-8.074e-08** | Timed Out |

**Table 8** Results on Example 2, m=2, n=6, NumCons=4.

expected solver accuracy. For the Augmented Lagrangian formulation we remove the slack variable $s$ and replace the LP solve via `CVX` with the closed-form projection

$$y_x = \begin{cases} \hat{x}, & \text{for } ||\hat{x}||_2 \leq 1 \\ \frac{\hat{x}}{||\hat{x}||_2}, & \text{otherwise} \end{cases} \tag{31}$$

where $\hat{x} = -\frac{1}{\mu}(Cx^k + \lambda^k) + x^k$. Table 7 summarizes the performance of various solvers n terms of the final solution status and the normalized residuals. The table does not include alternating minimization, since every feasible point is also a saddle point for this problem.

*Example 6* Table 8 applies these approaches to the relatively small problem obtained from the program shown in Example 2. The objective in this example is set to $c_1 - c_2$, wherein the invariant that we seek has the form $x_1 \leq c_1 \wedge x_1 \geq c_2$. Starting with the initial solution $c_1 = 100, c_2 = 0$, alternating minimization using an interior point solver and 5 other approaches compute the best solution $c_1 - c_2 = 1$.

Tables 8, 9, 10, 11, and 12 compare the performance of various algorithms over benchmark IDs 2-5 from Table 1. In each table, the highlighted entries indicate "desirable results" in terms of a negative solution and corresponding feasibility gaps of $10^{-7}$ or less. Due to the different types of solvers and the use of Matlab interfaces, we will not compare the time taken by each approach. The key conclusion is that alternating minimization approach is by far the best in terms of solving these benchmarks with acceptable feasibility gaps. This is surprising given the propensity of this approach to get "stuck"

| Method | Obj Value | $\max\{\lvert c_{\text{eq}}(x)\rvert\}$ | $\max\{c_{\text{ineq}}(x)\}$ | Status |
|---|---|---|---|---|
| Alt Min Gurobi | **-3.175e-02** | **3.980e-11** | **4.818e-14** | Solved |
| Alt Min Sedumi | **-9.344e-01** | 1.632e+01 | 7.915e-01 | Infeasible |
| FMINCON | -7.064e+06 | 7.502e-02 | 2.984e-01 | Timed Out |
| BONMIN | -1.022e+21 | NA | NA | Failed |
| COUENNE | NA | NA | NA | Timed Out |
| IPOPT | -3.265e+12 | 1.962e+08 | 5.258e+07 | Timed Out |
| BARON | -2.375e+07 | 4.475e+09 | 4.096e+13 | Timed Out |
| Aug Lag | 3.950e+02 | 1.140e-01 | 1.432e-01 | Stagnant |

**Table 9** Benchmark # 2 from Table 1, m=48, n=264, p=353, NumCons=123.

| Method | Obj Value | $\max\{\lvert c_{\text{eq}}(x)\rvert\}$ | $\max\{c_{\text{ineq}}(x)\}$ | Status |
|---|---|---|---|---|
| Alt Min Gurobi | **-2.066e-01** | **1.137e-13** | **4.441e-16** | Solved |
| Alt Min Sedumi | **-2.286e-01** | **2.220e-09** | **5.754e-09** | Solved |
| FMINCON | 8.386e+01 | **2.536e-11** | **-2.560e-05** | Timed Out |
| BONMIN | -6.298e+08 | NA | NA | Failed |
| COUENNE | NA | NA | NA | Timed Out |
| IPOPT | -3.805e+08 | 1.087e+08 | 1.547e+11 | Timed Out |
| BARON | -1.102e+13 | 3.148e+12 | 1.786e+11 | Timed Out |
| Aug Lag | 3.310e+00 | 1.029e-05 | 5.331e-06 | Stagnant |

**Table 10** Benchmark # 3 from Table 1, m=24, n=72, p=161, NumCons=51.

| Method | Obj Value | $\max\{\lvert c_{\text{eq}}(x)\rvert\}$ | $\max\{c_{\text{ineq}}(x)\}$ | Status |
|---|---|---|---|---|
| Alt Min Gurobi | **-1.000e-01** | **1.138e-14** | 1.213e-13 | Solved |
| Alt Min Sedumi | **-1.000e-01** | **2.064e-13** | **-5.174e-14** | Solved |
| FMINCON | 1.208e+00 | 2.211e-02 | 7.211e-02 | Infeasible |
| BONMIN | -5.641e+09 | NA | NA | Failed |
| COUENNE | NA | NA | NA | Timed Out |
| IPOPT | 3.741e+10 | 2.799e+10 | 5.144e+12 | Timed Out |
| BARON | -7.834e+13 | 1.454e+11 | 7.509e+15 | Infeasible |
| Aug Lag | 2.000e+00 | 7.380e-07 | 5.306e-07 | Solved |

**Table 11** Benchmark # 4 from Table 1, m=12, n=20, p=33, NumCons=27.

in a saddle point. Furthermore, floating point alternating minimization approaches achieve acceptable feasibility gaps and the approach itself lends to exact arithmetic implementation as shown in Section 6.

## 10 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

| Method | Obj Value | $\max\{\lvert c_{\text{eq}}(x)\rvert\}$ | $\max\{c_{\text{ineq}}(x)\}$ | Status |
|---|---|---|---|---|
| Alt Min Gurobi | **-2.417e-01** | **5.994e-13** | **1.742e-11** | Solved |
| Alt Min Sedumi | **-2.417e-01** | **8.652e-10** | **2.261e-09** | Solved |
| FMINCON | 2.546e+02 | **3.553e-15** | **-9.184e-04** | Infeasible |
| BONMIN | -6.687e+17 | NA | NA | Failed |
| COUENNE | NA | NA | NA | Timed Out |
| IPOPT | -1.764e+13 | 5.408e+09 | 2.230e+10 | Timed Out |
| BARON | 1e+51 | NA | NA | Failed |
| Aug Lag | 1.847e+01 | 1.859e-05 | 1.198e-06 | Stagnant |

**Table 12** Benchmark # 5 from Table 1, m=40, n=410, p=681, NumCons=204.

iteration approaches that can incorporate the template update process [32]. Our previous work on invariant set computation for polynomial differential equations mentioned earlier, already contains clues to such an approach [61]. 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 award numbers SHF 1527075. All opinions expressed are those of the authors, and not necessarily of the NSF.

## References

1. Couenne, a solver for nonconvex minlp problems (2016), `https://www.coin-or.org/Couenne/`
2. 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)
3. 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)
4. 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)
5. Amato, G., Parton, M., Scozzari, F.: Deriving Numerical Abstract Domains via Principal Component Analysis, pp. 134–150. Springer (2010)
6. Anstreicher, K.M.: Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. Journal of Global Optimization 43(2), 471–484 (2009)
7. Audet, C., Hansen, P., Jaumard, B., Savard, G.: A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. Mathematical Programming 87(1), 131–152 (2000)
8. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Quaderno 457, Dipartimento di Matematica, Università di Parma, Italy (2006)
9. Bagnara, R., Ricci, E., Zaffanella, E., Hill, P.M.: Possibly not closed convex polyhedra and the Parma Polyhedra Library. In: Static Analysis Symposium. vol. 2477, pp. 213–229 (2002)

10. Belotti, P.: Couenne: a users manual. Tech. rep., Technical report, Lehigh University (2009)
11. Bertsekas, D.P., Nedić, A., Ozdaglar, A.E.: Convex Analysis and Optimization. Athena Scientific (2003)
12. 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)
13. 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)
14. Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.: Counterexample-guided refinement of template polyhedra (2017), to Appear
15. Bonami, P., Biegler, L.T., Conn, A.R., Cornuéjols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., et al.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization 5(2), 186–204 (2008)
16. Byrd, R.H., Lu, P., Nocedal, J.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Stat. Comp. 16(5), 1190–1208 (1995)
17. 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)
18. 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)
19. Chvátal, V.: Linear Programming. Freeman (1983)
20. Clariso, R., Cortadella, J.: The octahedron abstract domain. Science of Computer Programming 64(1), 115 – 139 (2007)
21. Colón, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: CAV. vol. 2725, pp. 420–433 (July 2003)
22. 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)
23. 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)
24. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proc. ISOP'76. pp. 106–130. Dunod, Paris, France (1976)
25. 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)
26. 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)
27. Dauphin, Y.N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., Bengio, Y.: Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In: Advances in neural information processing systems. pp. 2933–2941 (2014)
28. 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)
29. 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)
30. 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)

31. 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)
32. 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)
33. 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)
34. 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)
35. Grant, M., Boyd, S.: Graph implementations for nonsmooth convex programs. Recent advances in learning and control pp. 95–110 (2008)
36. Grant, M., Boyd, S., Ye, Y.: Cvx: Matlab software for disciplined convex programming (2008)
37. Guernic, C.L., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Analysis: Hybrid Systems 4(2), 250 – 262 (2010)
38. Gulwani, S., Srivastava, S., Venkatesan, R.: Program analysis as constraint solving. In: PLDI. pp. 281–292. ACM (2008)
39. Helton, J., Merino, O.: Coordinate optimization for bi-convex matrix inequalities. In: IEEE Conf. on Decision & Control(CDC). p. 36093613 (1997)
40. Jin, C., Ge, R., Netrapalli, P., Kakade, S.M., Jordan, M.I.: How to escape saddle points efficiently. In: ICML (2017)
41. Kim, S., Kojima, M.: Second order cone programming relaxation of nonconvex quadratic optimization problems. Optimization methods and software 15(3-4), 201–224 (2001)
42. Kočvara, M., Stingl, M.: PENNON: A code for convex nonlinear and semidefinite programming. Optimization methods and software 18(3), 317–333 (2003)
43. Lee, J.D., Simchowitz, M., Jordan, M.I., Recht, B.: Gradient descent only converges to minimizers. In: Conference on Learning Theory. pp. 1246–1257 (2016)
44. Linderoth, J.: A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. Mathematical Programming 103(2), 251–282 (2005)
45. 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)
46. Mathworks Inc.: PolySpace design verifier, cf. `http://www.mathworks.com/products/polyspace/` viewed April 2017
47. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: PADO II. vol. 2053, pp. 155–172 (May 2001)
48. Miné, A.: The octagon abstract domain. In: AST 2001 in WCRE 2001. pp. 310–319. IEEE, IEEE CS Press (October 2001)
49. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis (1999)
50. Nocedal, J., Wright, S.: Numerical Optimization. Springer, 2nd edn. (2006)
51. Optimization, G.: Inc.,gurobi optimizer reference manual, 2017. URL: http://www.gurobi. com (2017)
52. Park, J., Boyd, S.: General heuristics for nonconvex quadratically constrained quadratic programming. arXiv preprint arXiv:1703.07870 (2017)
53. Qualizza, A., Belotti, P., Margot, F.: Linear programming relaxations of quadratically constrained quadratic programs. Mixed Integer Nonlinear Programming pp. 407–426 (2012)
54. Rockafellar, R., Wets, R.: Variational Analysis. Springer Berlin Heidelberg (2009)
55. 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)
56. Sahinidis, N.V.: BARON 17.8.9: Global Optimization of Mixed-Integer Nonlinear Programs, *User's Manual* (2017)
57. Sankaranarayanan, S.: Mathematical Analysis of Programs. Ph.D. thesis, Stanford University (7 2005)
58. 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)

59. 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)
60. 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)
61. 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)
62. 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)
63. Sherali, H.D., Dalkiran, E., Liberti, L.: Reduced rlt representations for nonconvex polynomial programming problems. Journal of Global Optimization 52(3), 447–469 (2012)
64. Sherali, H.D., Fraticelli, B.M.: Enhancing rlt relaxations via a new class of semidefinite cuts. Journal of Global Optimization 22(1), 233–261 (2002)
65. Sturm, J.F.: Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. Optimization methods and software 11(1-4), 625–653 (1999)
66. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Mathematical Programming 103, 225–249 (2005)
67. Venet, A., Brat, G.P.: Precise and efficient static array bound checking for large embedded C programs. In: PLDI. pp. 231–242. ACM (2004)
68. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical programming 106(1), 25–57 (2006)
69. 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)
70. Zheng, X.J., Sun, X.L., Li, D.: Convex relaxations for nonconvex quadratically constrained quadratic programming: matrix cone decomposition and polyhedral approximation. Mathematical programming 129(2), 301–329 (2011)