Generalizing the Template Polyhedral Domain*

Michael A. Colón[1] and Sriram Sankaranarayanan[2]

1. U.S. Naval Research Laboratory, Washington, DC. colon@itd.nrl.navy.mil
2. University of Colorado, Boulder, CO. srirams@colorado.edu

Abstract. Template polyhedra generalize weakly relational domains by specifying arbitrary fixed linear expressions on the left-hand sides of inequalities and undetermined constants on the right. The domain operations required for analysis over template polyhedra can be computed in polynomial time using linear programming. In this paper, we introduce the generalized template polyhedral domain that extends template polyhedra using fixed left-hand side expressions with bilinear forms involving program variables and unknown parameters to the right. We prove that the domain operations over generalized templates can be defined as the "best possible abstractions" of the corresponding polyhedral domain operations. The resulting analysis can straddle the entire space of linear relation analysis starting from the template domain to the full polyhedral domain.

We show that analysis in the generalized template domain can be performed by dualizing the join, post-condition and widening operations. We also investigate the special case of template polyhedra wherein each bilinear form has at most two parameters. For this domain, we use the special properties of two dimensional polyhedra and techniques from fractional linear programming to derive domain operations that can be implemented in polynomial time over the number of variables in the program and the size of the polyhedra. We present applications of generalized template polyhedra to strengthen previously obtained invariants by converting them into templates. We describe an experimental evaluation of an implementation over several benchmark systems.

1 Introduction

In this paper, we present some generalizations of the template polyhedral domain. Template polyhedral domains [34, 32] were introduced in our earlier work as a generalization of domains such as *intervals* [10], *octagons* [28], *octahedra* [7], *pentagons* [26] and *logahedra* [20]. The characteristic feature of these domains is that assertions are restricted to a form that makes the analysis tractable. For instance, the assertions involved in the octagon domain are of the form $\pm x \pm y \leq c$ for each pair of program variables x and y, along with some unspecified constant

* This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 0953941 and the Office of Naval Research (ONR). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or ONR.

$c \in \mathbb{R}$. The goal of the program analysis is to discover a suitable set of constants so that the resulting assertions are inductive, or equivalently, form a (post-) fixed point under the program's semantics [11].

In this paper, we generalize template polyhedral domains to consider templates with a linear expression on the left-hand side and a bilinear form on the right-hand side that specifies a parameterized linear expression. For instance, our analysis can handle template inequalities of the form $x - y \leq c_1 z + c_2 w + d$, wherein x, y, z, w are program variables and c_1, c_2, d are unknown parameters for which values will be computed by the analysis so that the entire expression forms a program invariant. This generalization can straddle the space of numerical domains from weakly relational domains (bilinear form is an unknown constant) to the full polyhedral domain (bilinear form has all the program variables) [12, 8]. The main contributions of this paper are as follows:

– We generalize template polyhedra to consider the case where each template can be of the form $e \leq f$, wherein e is a linear expression over the program variables and f is a bilinear form over the program variables involving unknown parameters that are to be instantiated by the analysis.
– We prove that the domain operations can be performed *output sensitively* in polynomial time for the special case of two unknown parameters in each template. Our technique uses *fractional linear programming* [5] to simulate Jarvis's march for two dimensional polyhedra [9, 23].
– We describe potential applications of our ideas to improve fixed points computed by other numerical domain analyses. These applications partially address the question of how to select generalized templates and use them in a numerical domain analysis framework.

We evaluate our approach against polyhedral analysis by using generalized templates to improve the fixed points computed by polyhedral analysis on several benchmark programs taken from the literature [3, 34]. We find that the ideas presented in this paper help to improve the fixed point in many of these benchmarks by discovering new relations not implied by the previously computed fixed points

2 Preliminaries

Throughout the paper, let $\mathbb{R}$ represent the set of real numbers. We fix a set of *variables* $X = \{x_1, \ldots, x_n\}$, which often correspond to the variables of the program under study.

Polyhedra: We recall some standard results on polyhedra. A *linear expression* e is of the form $a_1 x_1 + \cdots + a_n x_n + b$, wherein each $a_i \in \mathbb{R}$ and $b \in \mathbb{R}$. The expression is said to be *homogeneous* if $b = 0$. For a linear expression $e = a_1 x_1 + \cdots + a_n x_n + b$, let $\textsc{vars}(e) = \{x_i | a_i \neq 0\}$.

Definition 1 (Linear Assertions). *A* linear inequality *is of the form* $a_1 x_1 + \cdots + a_n x_n + b \leq 0$. *A* linear assertion *is a finite conjunction of linear inequalities.*

Note that the linear inequality $0 \leq 0$ represents the assertion *true*, whereas the inequality $1 \leq 0$ represents *false*. An assertion can be written in matrix form as $A\boldsymbol{x} \leq \boldsymbol{b}$, where A is an $m \times n$ matrix, while $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{b} = (b_1, \ldots, b_m)$ are n- and m-dimensional vectors, respectively. The i^{th} row of the matrix form is an inequality that will be written as $A_i\boldsymbol{x} \leq \boldsymbol{b}_i$. Note that each equality is represented by a pair of inequalities. The set of points in $\mathbb{R}^n$ satisfying a linear assertion $\varphi : A\boldsymbol{x} \leq \boldsymbol{b}$ is denoted $[\![\varphi]\!] : \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$. Such a set is called a *polyhedron*. Given two linear assertions φ_1 and φ_2, we define the entailment relation $\varphi_1 \models \varphi_2$ iff $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$.

The representation of a polyhedron by a linear assertion is known as its *constraint representation*. Alternatively, a polyhedron can be represented explicitly by a finite set of vertices and rays, known as its *generator representation*. There are several well-known algorithms for converting from one representation to the other. Highly engineered implementations of these algorithms such as the Apron [24] and Parma Polyhedral (PPL) [2] libraries implement the conversion between constraint and genrator representations, which is a key primitive for implementing the domain operations required to carry out abstract interpretation over polyhedra [12]. Nevertheless, conversion between the constraint and the generator representations still remains intractable for polyhedra involving a large number of variables and constraints.

Linear Programming: We briefly describe the theory of linear programming. Details may be found in standard textbooks, such as Schrijver [35].

Definition 2 (Linear Programming). *A canonical instance of the* linear programming (LP) problem *is of the form* **min.** e **s.t.** φ*, for a linear assertion* φ *and a linear expression* e*, called the* objective function.

The goal is to determine a solution of φ for which e is minimal. An LP problem can have one of three results: (1) an optimal solution; (2) $-\infty$, i.e, e is unbounded from below in φ; and (3) $+\infty$, i.e, φ has no solutions.

It is well-known that an optimal solution, if it exists, is realized at a vertex of the polyhedron. Therefore, the optimal solution can be found by evaluating e at each of the vertices. Enumerating all the vertices is very inefficient because the number of generators can be exponential in the number of constraints in the worst-case. The popular SIMPLEX algorithm (due to Danzig [35]) employs a sophisticated hill-climbing strategy that converges on an optimal vertex without necessarily enumerating all vertices. In theory, SIMPLEX is worst-case exponential. However, in practice, SIMPLEX is efficient for most problems. Interior point methods and ellipsoidal techniques are guaranteed to solve linear programs in polynomial time [5].

Fractional Linear Programs: Fractional Linear Programs (FLPs) will form an important primitive for computing abstractions and transfer functions of generalized template polyhedra. We describe the basic facts about these in this section. More details about FLPs are available elsewhere [5].

Definition 3. *A fractional linear program is an optimization problem of the following form:*

$$\textbf{min.}\ \frac{\boldsymbol{a}^T\boldsymbol{x} + a_0}{\boldsymbol{c}^T\boldsymbol{x} + c_0}\ \textbf{s.t.}\ \ A\boldsymbol{x} \le \boldsymbol{b},\ \boldsymbol{c}^T\boldsymbol{x} + c_0 > 0\,. \tag{1}$$

A fractional linear program can be solved by *homogenizing* it to an "equivalent" LP [5]:

$$\textbf{min.}\ \ \boldsymbol{a}^T\boldsymbol{x} + a_0 z\ \textbf{s.t.}\ \ A\boldsymbol{x} \le \boldsymbol{b}z\ \wedge \boldsymbol{c}^T\boldsymbol{x} + c_0 z = 1\ \wedge\ z \ge 0\,. \tag{2}$$

If the LP (2) has an optimal solution $(\boldsymbol{x}^*, z^*)$ such that $z^* > 0$ then $\frac{1}{z^*}\boldsymbol{x}^*$ is an optimal solution to the FLP (1). Furthermore, $\frac{1}{z^*}\boldsymbol{x}^*$ is also a vertex of the polyhedron $A\boldsymbol{x} \le \boldsymbol{b}$. If, on the other hand, $z^* = 0$, the optimal solution to (1) is an extreme ray of the polyhedron $A\boldsymbol{x} \le \boldsymbol{b}$. Finally if LP (2) is infeasible then the FLP (1) is infeasible.

Transition Systems We will briefly define transition systems over real-valued variables as the model of computation used throughout this paper. It is possible to abstract a given program written in a language such as C or Java with arrays, pointers and dynamic memory calls into abstract numerical models. Details of this translation, known as *memory modeling*, are available elsewhere [22, 6, 4]. To ensure simplicity, our presentation here does not include techniques for handling function calls. Nevertheless, function calls can be handled using standard extensions of this approach.

Definition 4 (Transition System). *A* transition system Π *is represented by a tuple* $\langle X, L, \mathcal{T}, \ell_0, \Theta\rangle$*, where*

- $X = \{x_1, \ldots, x_n\}$ *is a set of* variables*. For each variable* $x_i \in X$*, there is an associated* primed variable x_i'*, and the set of primed variables is denoted by* X'*. The variables of* X *are collectively denoted as a vector* $\boldsymbol{x}$*;*
- L *is a finite set of* locations*;*
- $\mathcal{T}$ *is a finite set of* transitions*. Each transition* $\tau \in \mathcal{T}$ *consists of a tuple* $\tau : \langle \ell, m, \rho_\tau\rangle$ *wherein* $\ell, m \in L$ *are the* pre- *and the* post-locations *of the transition,* $\rho_\tau[X, X']$ *is the* transition relation *that relates the current state variables* X *and the next-state variables* X'*;*
- $\ell_0 \in L$ *represents the* initial location*; and*
- Θ *is the* initial condition*, specified as an assertion over* X*. It represents the set of initial values of the program variables.*

A transition system is *linear* iff the variables X range over the domain of real numbers, each transition $\tau \in \mathcal{T}$ has a *linear transition relation* ρ_τ that is represented as a linear assertion over $X \cup X'$, and finally, the initial condition Θ is a linear assertion over X. Further details on transition systems, including their operational semantics and the definition of invariants, are available from standard references [27].

3 Generalized Templates

In this section, we introduce the *generalized template domain.* To begin with, we recall expression templates. A template T is a set of homogeneous linear expressions $\{e_1, \ldots, e_m\}$ over program variables X. The approach of template expressions was formalized in our previous work [34, 32]. Let $X = \{x_1, \ldots, x_n\}$ be the set of program variables and $C = \{c_1, c_2, \ldots, d_1, d_2, \ldots\}$ denote a set of unknown *parameters.*

Definition 5 (Bilinear Form). *A* bilinear form *over X and C is given by an expression of the form*

$$f : \left(\sum_{i \in I} c_i x_i \right) + d, \textit{ wherein } I \subseteq \{1, \ldots, n\}$$

Given the bilinear form f above, let $\textsc{vars}(f) : \{x_i \mid i \in I\}$ *and let* $\textsc{pars}(f) : \{c_i \mid i \in I\} \cup \{d\}$ *denote the parameters involved in f.*

Given a mapping $\mu : C \to \mathbb{R}$, a bilinear form $f : d + \sum_{i \in I} c_i x_i$ can be mapped to the linear expression $f[\mu] : \mu(d) + \sum_{i \in I} \mu(c_i) x_i$

Definition 6 (Generalized Templates). *A* generalized template *G consists of a set of entries: $G = \{(e_1, f_1), \ldots, (e_m, f_m)\}$, where for each $i \in \{1, \ldots, m\}$, e_i is a homogeneous linear expression over X, and f_i is a bilinear form over C and X. The entry (e_j, f_j) represents the linear inequality $e_j \leq f_j$, where the left-hand side is a fixed linear expression and the right-hand side is a parameterized linear expression represented as a bilinear form.*

We assume that two bilinear forms f_i and f_j, for $i \neq j$, cannot share common parameters, i.e., $\textsc{pars}(f_i) \cap \textsc{pars}(f_j) = \emptyset$. *Finally, we assume that for each entry (e_i, f_i) in a generalized template G,* $\textsc{vars}(e_i) \cap \textsc{vars}(f_i) = \emptyset$, *i.e, the left- and right-hand sides share no variables.*

Example 1. Consider program variables $X = \{x_1, x_2, x_3\}$ and parameters $C = \{c_1, c_2, c_3, c_4, d_1, d_2, d_3\}$. An example of a generalized template follows:

$$G : \left\{ \begin{array}{lll} (e_1 : x_1 & , & f_1 : c_1 x_2 + c_2 x_3 + d_1), \\ (e_2 : x_2 & , & f_2 : c_3 x_3 + d_2), \\ (e_3 : x_3 - 2x_1 & , & f_3 : c_4 x_2 + d_3) \end{array} \right\} .$$

Notation: For a bilinear form f_j, will use the variable d_j to denote the constant coefficient and $c_{j,k}$ to denote the parameter for the coefficient of the variable x_k.

Generalized templates define an infinite family of convex polyhedra called *generalized template polyhedra.*

Definition 7 (Generalized Template Polyhedron). *Given a template G, a polyhedron $\varphi : \bigwedge_i \ A_i \boldsymbol{x} \leq \boldsymbol{b}_i$ is a* generalized template polyhedron *(GTP) iff φ is the empty polyhedron, or each inequality $\varphi_i : \ A_i \boldsymbol{x} \leq \boldsymbol{b}_i$ in φ can be cast as the instantiation of some entry $(e_j, f_j) \in G$ in one of the following ways:*

Vertex Instantiation: $\varphi_i : \ e_j \leq f_j[\mu_{ij}]$ *for some map* $\mu_{ij} : \textsc{pars}(f_j) \to \mathbb{R}$ *that instantiates the parameters in* $\textsc{pars}(f_j)$ *to some real values.*
Ray Instantiation: $\varphi_i : \ \mathbf{0} \leq f_j[\mu_{ij}]$ *for some map* $\mu_{ij} : \textsc{pars}(f_j) \to \mathbb{R}$.

The rationale behind ray instantiation will be made clear presently. A *conformance map* γ between a GTP $\varphi : A\boldsymbol{x} \leq \boldsymbol{b}$ and its generalized template G, maps every inequality $\varphi_i : A_i\boldsymbol{x} \leq \boldsymbol{b}_i$ to some entry $\gamma(i) \in G$ so that φ_i may be viewed as a vertex or a ray instantiation of the template entry $\gamma(i)$. If a conformance map exists, then φ is said to *conform* to G.

If φ is a GTP that conforms to the template G, it is possible that (1) a single inequality in φ can be expressed as instantiations of multiple entries in G, and (2) a single entry (e_j, f_j) can be instantiated as multiple inequalities (or no inequalities) in φ. In other words, conformance maps between φ and G, can be many-to-one and non-surjective.

Example 2. Recalling the template G from Example 1, the polyhedron $\varphi_1 \ \wedge \cdots \wedge \ \varphi_4$, below, conforms to G:

$$\begin{array}{l|l} \varphi_1 & x_1 \leq 1 \\ \varphi_2 & x_1 \leq x_3 - 10 \end{array} \qquad \begin{array}{l|l} \varphi_3 & x_2 \qquad\ \ \leq 2x_3 - 10 \\ \varphi_4 & x_3 - 2x_1 \leq 3x_2 + 1 \end{array}$$

The conformance map is given by

$$\gamma : 1 \mapsto (e_1, f_1), 2 \mapsto (e_1, f_1), 3 \mapsto (e_2, f_2), 4 \mapsto (e_3, f_3)\,.$$

On the other hand, the polyhedron $x_1 \leq 1 \ \wedge \ x_1 \geq 5$, does not conform to G, whose definition in this example does not permit lower bounds for the variable x_1. ■

We now describe the abstract domain of generalized templates, consisting of all the generalized template polyhedra (GTP) that conform to the template G. We will denote the set of all GTPs given a template G as Ψ_G and use the standard entailment amongst linear assertions as the ordering amongst GTPs. Theorem 2 shows that the structure $\langle \Psi_G, \models \rangle$, induced by the generalized template G, is a lattice.

Note 1. Using ray instantiations, the empty polyhedron *false* can be viewed as a GTP for any template G. We define the two polyhedra *true* and *false* to belong to Ψ_G for any G, including $G = \emptyset$.

Abstraction We now define the abstraction map α_G that transforms a given convex polyhedron $\varphi : A\boldsymbol{x} \leq \boldsymbol{b}$ into a GTP $\psi_G : \ \alpha_G(\varphi)$. If φ is empty, then the abstraction $\alpha_G(\varphi)$ is defined to be *false*. For the ensuing discussion, we assume that φ is a non-empty polyhedron. Our abstraction map is the *best possible*, yielding the "smallest" GTP in Ψ_G through a process of dualization using Farkas' lemma [8]:

(1) For each $(e_j, f_j) \in G$, we wish to compute the set of all values of the unknown parameters in $\textsc{pars}(f_j)$ so that the entailment $\varphi : A\boldsymbol{x} \leq \boldsymbol{b} \models e_j \leq f_j$ holds. For convenience, we express $e_j \leq f_j$ as $\boldsymbol{c}^T\boldsymbol{x} \leq d$ wherein $\boldsymbol{c}, d$ consist of linear expressions over the unknowns in $\textsc{pars}(f_i)$.

(2) Applying Farkas' lemma to the entailment above yields a dual polyhedron φ_j^D, whose variables include the unknowns in $\textsc{pars}(f_j)$ and some vector of *multipliers* $\boldsymbol{\lambda}$. This dual polyhedron is written as:

$$\begin{gathered} A\boldsymbol{x} \leq \boldsymbol{b} \models \boldsymbol{c}^T\boldsymbol{x} \leq d\,, \text{ holds} \\ \text{iff} \\ \varphi_j^D : \ (\exists\, \boldsymbol{\lambda} \geq 0)\ A^T\boldsymbol{\lambda} = \boldsymbol{c},\ \boldsymbol{b}^T\boldsymbol{\lambda} \leq d. \end{gathered}$$

(3) We eliminate the multipliers $\boldsymbol{\lambda}$ from φ_j^D to obtain ψ_j over variables $\textsc{pars}(f_j)$.

(4) If ψ_j is empty, then the abstraction is defined as the universal polyhedron. Otherwise, each vertex $\boldsymbol{v}_i$ of ψ_j is instantiated to an inequality by *vertex instantiation*: $e_j \leq f_j[\textsc{pars}(f_j) \mapsto \boldsymbol{v}_i]$ Similarly, each ray $\boldsymbol{r}_k$ of ψ_j is instantiated to an inequality by *ray instantiation*: $\boldsymbol{0} \leq f_j[\textsc{pars}(f_j) \mapsto \boldsymbol{r}_k]$.

The overall result is the conjunction of all inequalities obtained by vertex and ray instantiation for all the entries of G. In practice, this result has many redundant constraints. These redundancies can be eliminated using LP solvers [32].

In the general case, the elimination of $\boldsymbol{\lambda}$ in step (3) may be computed exactly using the double description method [31] or Fourier-Motzkin elimination [35]. However, in Section 4 we show techniques where the elimination can be performed in output-sensitive polynomial time when $|\textsc{pars}(f_j)| \leq 2$ (even when $|\boldsymbol{\lambda}|$ is arbitrary) using fractional linear programming. In Section 5, we show how the result of the elimination can be *approximated* soundly using LP solvers.

Theorem 1. *For any polyhedron φ and template G, let $\psi : \alpha_G(\varphi)$ be the abstraction computed using the procedure above. (A) ψ is a GTP conforming to G, i.e, $\psi \in \Psi_G$, (B) $\varphi \models \psi$ and (C) ψ is the best abstraction of φ in the lattice: $(\forall \psi' \in \Psi_G)\ \varphi\ \models\ \psi'$ iff $\psi \models \psi'$.*

Proofs of all the theorems will be provided in an extended version of this paper. As a consequence, the existence of a best abstraction allows us to prove that $\langle \Psi_G, \models \rangle$ is itself a lattice.

Theorem 2. *The structure $\langle \Psi_G, \models \rangle$ forms a lattice, and furthermore, the maps $(\alpha_G, \mathsf{identity})$ form a Galois connection between this lattice and the lattice of polyhedra.*

Example 3 (Abstraction). We illustrate the process of abstraction through an example. Consider the polyhedron φ over two variables x, y:

$$\varphi : x \leq 2\ \wedge\ x \geq 1\ \wedge\ y \leq 2\ \wedge\ y \geq 1\,.$$

We wish to compute $\alpha_G(\varphi)$ for the following template $\left\{ \begin{array}{l} (e_1 :\ x, f_1 :\ c_1 y + d_1) \\ (e_2 :\ y, f_2 :\ c_2 x + d_2) \end{array} \right\}$. We will now illustrate computing the polyhedron $\psi_1[c_1, d_1]$ corresponding to the entry $(e_1 : \; x, f_1 : \; c_1 y + d_1)$. To do so, we wish to find values of c_1, d_1 that satisfy the entailment $\varphi : (x \in [1,2] \;\wedge\; y \in [1,2]) \models x \leq c_1 y + d_1$. Dualizing, using Farkas' lemma, we obtain the following constraints

$$(\exists \lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0) \left[\begin{array}{r} \lambda_1 - \lambda_2 = 1 \\ \lambda_3 - \lambda_4 = -c_1 \\ 2\lambda_1 - \lambda_2 + 2\lambda_3 - \lambda_4 \leq d_1 \end{array}\right] .$$

Eliminating $\lambda_{1,2,3,4}$ yields the polyhedron: $\psi_1 : \; c_1 + d_1 \geq 2 \;\wedge\; 2c_1 + d_1 \geq 2$. The polyhedron ψ_1 has a vertex $\boldsymbol{v}_1 : (0, 2)$ and rays $\boldsymbol{r}_1 : (1, -1)$ and $\boldsymbol{r}_2 : (-\frac{1}{2}, 1)$.

We obtain the inequality $e_1 \leq f_1[\boldsymbol{v}_1] : x \leq 0y + 2$ by instantiating the vertex. Similarly, instantiating using *ray instantiation* yields: $0 \leq f_1[\boldsymbol{r}_1] : 0 \leq y - 1$ and $0 \leq f_1[\boldsymbol{r}_2] : 0 \leq \frac{-1}{2} y + 1$. Considering the second entry (e_2, f_2) yields the inequality $y \leq 2$ through a vertex instantiation and the inequalities $x \geq 1$ and $x \leq 2$ using ray instantiation. Conjoining the results from both entries, we obtain $\alpha_G(\varphi)$, which is equivalent to φ. ■

Why Ray Instantiation? We can now explain the rationale behind ray instantiation in our domain. If ray instantiation is not included then the best abstraction clause in Theorem 1 will no longer hold (see proof in extended version). Example 4, below, serves as a counter-example. Further, $\langle \Psi_G, \models \rangle$ is no longer a lattice since the least upper bound Ψ_G may no longer exist. This can complicate the construction of the abstract domain. Ray instantiations are necessary to capture the rays of the dual polytope obtained in step (3) of the abstraction process. Let us assume that $\boldsymbol{r}$ is a ray of a dual polytope $\psi[\boldsymbol{c}]$ that contains all values of $\boldsymbol{c}$ so that the primal entailment $\varphi \models e_i \leq f_i[\boldsymbol{c}]$ holds. In other words, for any $\boldsymbol{c}_0 \in [\![\psi]\!]$, the inequality $I : e_i \leq f_i[\boldsymbol{c}_0]$ is entailed by φ. If ψ contains a ray $\boldsymbol{r}$ then $\boldsymbol{c}_0 + \gamma \boldsymbol{r} \in [\![\psi]\!]$ for any $\gamma \geq 0$. This models infinitely many inequalities, all of which are consequences of φ:

$$\overline{I}(\gamma) : \;\; e_i \leq f_i[\boldsymbol{c}_0 + \gamma \boldsymbol{r}] = e_i \leq f_i[\boldsymbol{c}_0] + \gamma f_i[\boldsymbol{r}] \, ,$$

for $\gamma \geq 0$. Note that, in general, the inequality $\overline{I}(\gamma_1)$ need not be a consequence of $\overline{I}(\gamma_2)$ for $\gamma_1 \neq \gamma_2$. Including the ray instantiation $\hat{I} : \; \boldsymbol{0} \leq f_i[\boldsymbol{r}]$ allows us to write each inequality $\overline{I}(\gamma)$ as a proper consequence of just two inequalities $I, \hat{I}$, i.e, $\overline{I}(\gamma) : I + \gamma \hat{I}$.

Example 4. If ray instantiation is disallowed, then the abstraction $\alpha_G(\varphi)$ obtained for Example 3 will be $x \leq 2 \wedge y \leq 2$. However, this is no longer the best abstraction possible. For example, each polyhedron $x \leq 2 \wedge y \leq 2 \wedge x \leq \alpha y + 2 - \alpha$, for $\alpha \geq 0$, belongs to Ψ_G and is a better abstraction. However, larger values of α yield strictly stronger abstractions without necessarily having a strongest abstraction. Allowing ray instantiation lets us capture the inequality $0 \leq y - 1$ so that there is once again a best abstraction. ■

Post-Conditions: We now discuss the computation of post-conditions (transfer functions) across a transition $\tau : \langle \ell, m, \rho_\tau[\boldsymbol{x}, \boldsymbol{x}'] \rangle$.

Let φ be a GTP conforming to a template G. Our goal is to compute $post_G(\varphi, \tau)$, given a transition τ. Since $\rho_\tau[\boldsymbol{x}, \boldsymbol{x}']$ is a linear assertion, the best post-condition that we may hope for is given by the abstraction of the post-condition over the polyhedral domain: $\alpha_G(post_{poly}(\varphi, \tau))$. We show that the post-condition $post_G(\varphi, \tau)$ can be computed effectively, without computing $post_{poly}$, by modifying the algorithm for computing α_G.

1. Compute a convex polyhedron $P[\boldsymbol{x}, \boldsymbol{x}'] :\ \varphi \wedge\ \rho_\tau[\boldsymbol{x}, \boldsymbol{x}']$ whose variables range over both the current and the next state variables: $\boldsymbol{x}, \boldsymbol{x}'$. If P is empty, then the result of post-condition is empty.
2. For each entry $(e_j, f_j) \in G$, we consider the entailment $P[\boldsymbol{x}, \boldsymbol{x}'] \models e_j[\boldsymbol{x} \mapsto \boldsymbol{x}'] \leq f_j[\boldsymbol{x} \mapsto \boldsymbol{x}']$.
3. Dualizing, using Farkas' lemma, we compute a polyhedron ψ_j over the parameters in $\textsc{pars}(f_j)$.
4. If ψ_j is non-empty, instantiate the inequalities of the result using the vertices and rays of ψ_j. The result is the conjunction of all the vertex-instantiated and ray-instantiated inequalities.

Theorem 3. *$post_G(\varphi, \tau) = \alpha_G(post_{poly}(\varphi, \tau))$.*

For the case of transition relations that arise from assignments in basic blocks of programs, the post-condition computation can be optimized by substituting the next state variables $\boldsymbol{x}'$ with expressions in terms of $\boldsymbol{x}$ before computing the dualization.

Example 5 (Post-Condition Computation). Recall the template G from Example 3. Let $\varphi :\ x \in [1, 2],\ y \in [1, 2]$. Consider the transition τ with transition relation ρ_τ:

$$\rho_\tau[x, y, x', y'] :\ x' = x + 1 \ \wedge\ y' = y + 1\,.$$

Our goal is to compute $post_G(\varphi, \tau)$. To do so, we first compute $P :\ \varphi \ \wedge\ \rho_\tau$:

$$P[\boldsymbol{x}, \boldsymbol{x}'] :\ 1 \leq x \leq 2 \ \wedge\ 1 \leq y \leq 2 \ \wedge\ x' = x + 1 \ \wedge\ y' = y + 1\,.$$

We consider the entailment $P \models x' \leq c_1 y' + d_1$ arising from the first entry of the template. We may substitute $x + 1$ for x' and $y + 1$ for y' in the entailment yielding:

$$x \in [1, 2] \ \wedge\ y \in [1, 2] \ \models\ x + 1 \leq c_1(y + 1) + d_1 \ =\ x \leq c_1 y + (d_1 + c_1 - 1)\,.$$

After dualizing and eliminating the multipliers, we obtain $\psi_1[c_1, d_1] : 2c_1 + d_1 \geq 3 \wedge 3c_1 + d_1 \geq 3$. This is generated by the vertex $\boldsymbol{v}_1 : (3, 0)$ and rays $\boldsymbol{r}_1 :\ (\frac{1}{2}, -1)$ and $\boldsymbol{r}_2 :\ (\frac{-1}{3}, 1)$, yielding inequalities $x \leq 3,\ y \leq 3,$ and $y \geq 2$ through vertex and ray instantiation. The overall post-condition is $x \in [2, 3], y \in [2, 3]$. ■

For common program assignments of the form $x := x + c$, where c is a constant, the post-condition computation can be optimized so that dualization can be avoided. Similar optimizations are possible for assignments of the form

$x := c$. The post-condition for non-linear transitions can be obtained by abstracting them as (interval) linear assignments or using non-linear programming techniques. We refer the reader to work by Miné for more details [29].

Join Join over the polyhedral domain is computed by the convex hull operation. For the case of GTPs, join is obtained by first computing the convex hull and then abstracting it using α_G. However, this can be more expensive than computing the polyhedral convex hull. We provide an alternate and less expensive scheme that is once again based on Farkas' lemma. This join will be effectively computable in output-sensitive polynomial time when $|\textsc{pars}(f_j)| \leq 2$ for each template entry.

Let φ_1 and φ_2 be two GTPs conforming to a template G. We wish to compute the join $\varphi_1 \sqcup_G \varphi_2$ that is the least upper bound of φ_1 and φ_2 in Ψ_G.

1. For each entry $(e_j, f_j) \in G$, we encode the entailments

$$\varphi_1 \models e_j \leq f_j \text{ and } \varphi_2 \models e_j \leq f_j \,.$$

2. Let $\psi_1[\textsc{pars}(f_j)]$ and $\psi_2[\textsc{pars}(f_j)]$ be the polyhedra that result from encoding the entailments using Farkas' lemma and eliminating the multipliers for φ_1 and φ_2, respectively.
3. Let $\psi : \psi_1 \wedge \psi_2$ be the intersection. ψ captures all the linear inequalities that (a) fit the template $e_j \leq f_j$ and (b) are entailed by φ_1 as well as φ_2.
4. If ψ is non-empty, compute the generators of ψ and instantiate by vertex and ray instantiation.

The conjunction of all the inequalities so obtained for all the entries in G forms the resulting join.

Theorem 4. *For GTPs φ_1, φ_2 corresponding to template G, $\varphi : \varphi_1 \sqcup_G \varphi_2$ is the least upper bound in Ψ_G of φ_1 and φ_2.*

Widening Given φ_1 and φ_2, the standard widening $\varphi_1 \nabla \varphi_2$ drops those inequalities of φ_1 that are not entailed by φ_2. The result is naturally guaranteed to be in Ψ_G provided φ_1 and φ_2 are. Furthermore, the entailment for each inequality can be checked using linear programming [32]. Termination of this scheme is immediate since we drop constraints. *Mutually redundant* constraints used in the standard polyhedral widening can also be considered for the widening [18].

Note 2. Care must be taken not to apply the abstraction operator on the result φ of the widening. It is possible to construct examples wherein widening may drop a redundant constraint in φ_1 that can be restored by the application of the α_G operator [28].

Inclusion Checking Since our lattice uses the same ordering as polyhedra, inclusion checking can be performed in polynomial time using LP solvers.

Complexity Thus far, we have presented the GTP domain. The complexity of various operations may vary depending on the representation chosen for GTPs. We assume that each GTP is represented as a set of constraints.

The abstraction of convex polyhedra, computation of join and computation of post-condition require the elimination of multipliers $\boldsymbol{\lambda}$ from the assertion obtained after dualization using Farkas' lemma. The number of multipliers $\boldsymbol{\lambda}$ equals the number of constraints in the polyhedron being abstracted. This number is of the same order as the number of variables in the program plus the number of constraints in the polyhedron. It still remains open if the elimination of the $\boldsymbol{\lambda}$ multipliers can be performed efficiently by avoiding the exponential blow-up that is common for general polyhedra. We present two techniques to avoid this problem:

1. We present algorithms for the special case when each template entry has no more than two parameters. In this case, our algorithms run in time that is polynomial in the sizes of the inputs and the outputs.
2. The problem of sound approximations of the results of the elimination that can be computed in polynomial time will be tackled in Section 5.

4 Two Parameters Per Template Entry

In this section, we present efficient algorithms when each template entry has at most two parameters. Following the discussion on complexity in the previous section, we directly enumerate the generators of the projection of a dual form $P[c, d, \boldsymbol{\lambda}]$ onto c and d using repeated LP and FLP instances. The efficiency of the remaining operations over GTPs follow from the fact that they can all be reduced to computing α_G.

Abstraction: Our goal is to present an efficient algorithm for α_G when each entry of G has at most two unknown parameters. Specifically, let G be a template with entries of the form (e_i, f_i), wherein e_i is an arbitrary linear expression over the program variables and f_i is of the form $c_i x_j + d_i$, for some variable x_j and unknown parameters c_i, d_i.

Let $\varphi[x_1, \ldots, x_n]$ be a polyhedron over the program variables. If φ is empty, then the abstraction is empty. We assume for the remainder of this section that φ is a non-empty polyhedron. We compute $\alpha_G(\varphi)$ by dualizing the entailment below, for each entry (e_i, f_i):

$$\varphi[x_1, \ldots, x_n] \ \models \ e_i \leq c_i x_j + d_i \, . \tag{3}$$

Using Farkas' lemma, we obtain the dual form

$$\psi_i[c_i, d_i] : \ (\exists\, \boldsymbol{\lambda} \geq 0) \ P[c_i, d_i, \boldsymbol{\lambda}] \tag{4}$$

We present a technique that computes the vertices and rays of ψ_i using a series of fractional linear programs without explicitly eliminating $\boldsymbol{\lambda}$. We first note the following useful fact:

Lemma 1. *The vector $(0, 1)$ is a ray of ψ_i. Furthermore, since φ is non-empty, the vector $(0, -1)$ cannot be a ray of ψ_i.*

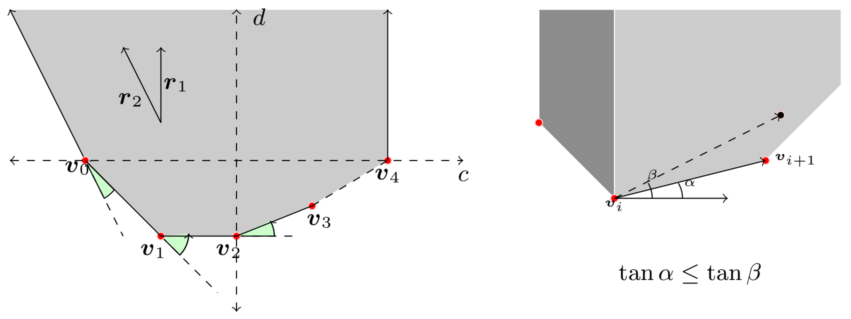

Fig. 1. Finding vertices and extreme rays of a 2D polygon by adapting Jarvis's march. The solid line connects vertices discovered so far. Each edge is discovered by means of an angle sweep starting from the previously discovered vertex.

4.1 Projecting to Two Dimensions Efficiently

In this section, we consider a polyhedron $P[c, d, \boldsymbol{\lambda}]$ as in (4). Our goal is to compute the vertices and rays of $\psi : (\exists\boldsymbol{\lambda})\ P$ efficiently. If P is empty, then the resulting projection is also the empty polyhedron. For the remainder of this section, P is assumed to be non-empty.

The overall technique for computing the vertices of ψ simulates Jarvis's march for computing the convex hull of a set of points in two dimensions [23, 9]. In this case, however, the polyhedron P is specified implicitly as the projection of a higher dimensional polyhedron. Figure 1 illustrates the idea behind this algorithm. The algorithm starts with some initial point guaranteed to be on the boundary of the polyhedron. Given a vertex $\boldsymbol{v}_i$ (or the initial point), the next vertex or extreme ray is computed in two steps:

1. Sweep a ray in a clockwise or counterclockwise direction from the current vertex until it touches the polyhedron. This ray defines an edge of the projection. Our algorithm performs the ray sweep by solving a suitably formulated FLP.
2. Find the farthest point in the polyhedron along this edge. If a farthest point can be found, then it is a vertex of the projection. Otherwise, the edge is an extreme ray. This step is formulated as a linear program in our algorithm.

We now discuss Steps 1 and 2 in detail.

Finding an Edge Using FLP: Let $\boldsymbol{v} : (a, b)$ be some vertex of ψ. Consider the following optimization problem:

$$\textbf{min.}\ \frac{d-b}{c-a}\ \textbf{s.t.}\ \ c > a\ \wedge\ P[c, d, \boldsymbol{\lambda}] \tag{5}$$

In effect, the optimization yields a *direction of least slope* joining $(c, d) \in [\![\psi]\!]$ and (a, b), where $c > a$. The optimization problem above is a *fractional linear program* [5] (also Cf. Section 2).

Lemma 2. *(a) If the FLP* (5) *is infeasible, then there is no point to the right of* (a, b) *in* $[\![\psi]\!]$. *(b) For non-empty* φ*, the FLP cannot be unbounded. (c) An optimal solution to the FLP yields the direction of least slope, as required by Jarvis's march.*

Finding the Farthest Point Along an Edge: Once a direction with slope θ is found starting from a vertex (a, b), we wish to find a vertex along this direction. This vertex can be found by solving the following LP:

$$\mathbf{max}.\ \ c - a\ \mathbf{s.t.}\ \ c - a \geq 0\ \wedge\ d = b + \theta(c - a)\ \wedge\ P[c, d, \boldsymbol{\lambda}] \tag{6}$$

In effect, we seek the maximum offset along the direction θ starting from (a, b) such that the end point (c, d) remains in ψ. The LP above cannot be infeasible (since $(c, d) = (a, b)$ is a feasible point). If the LP above is unbounded then we obtain an extreme ray of ψ. Otherwise, we obtain an end-point that is a vertex of ψ. The overall algorithm is similar to Jarvis's march as presented in standard textbooks (Cf. Cormen, Leiserson and Rivest, Ch. 35.3 [9] or R.A. Jarvis's original paper [23]). Rather than perform a "clockwise" and a "counterclockwise" march, we make use of the fact that $(0, 1)$ is a ray of ψ and perform an equivalent "leftward" and a "rightward" march.

1. We start by finding some point on the boundary of ψ. Let $(a, b, \boldsymbol{\lambda})$ be some feasible point in P. We solve the problem $\mathbf{min}.\ \ d\ \mathbf{s.t.}\ \ c = a\ \wedge\ P[c, d, \boldsymbol{\lambda}]$. The point $\boldsymbol{v}_0$ can be treated as a starting point that is guaranteed to lie on the boundary of ψ. Note, however, that $\boldsymbol{v}_0$ is not necessarilty a vertex of ψ. Since φ is assumed non-empty, the LP above cannot be unbounded.
2. We describe the rightward march. Each vertex $\boldsymbol{v}_{i+1}$ is discovered after computing $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_i$. Let $\boldsymbol{v}_i : (a_i, b_i)$ be the last vertex discovered.
   (a) Obtain least slope by solving the FLP problem in (5).
   (b) If the FLP is infeasible, then stop. $\boldsymbol{v}_i$ is the right-most vertex and the vector $(0, 1)$ an extreme ray.
   (c) Solve the LP in (6). An optimal solution yields a vertex $\boldsymbol{v}_{i+1}$. Otherwise, if the LP is unbounded, we obtain an extreme ray of ψ and $\boldsymbol{v}_i$ is the right-most vertex.
3. The leftward march can be performed using the reflection $T : c' \mapsto -c, d' \mapsto d$, carried out as a substitution in the polyhedron P. The resulting generators are reflected back.

Each generator discovered by our algorithm requires solving FLP and LP instances. Furthermore, the initial point can itself be found by solving at most two LP instances. Since LPs and FLPs can be solved efficiently, we have presented an efficient, *output-sensitive* algorithm for performing the abstraction. This algorithm, in turn, can be used as a primitive to compute the join and the post-condition efficiently, as well.

Note 3. It must be mentioned that an efficient (output-sensitive) algorithm for each abstract domain operation does not necessarily imply that the overall analysis itself can also be performed efficiently in terms of program size and template size. For instance, it is conceivable that the intermediate polyhedra and their coefficients can grow arbitrarily large during the course of the analysis by repeated applications of joins and post-conditions. This is a possibility for most strongly-relational domains, including the two variables per inequality domain proposed by Simon et al. [37]. Nevertheless, in practice, we find that a careful implementation can control this blow-up effectively by means of widening.

5 Efficient Approximations

In Section 4, we discussed the special case in which each bilinear form in a template has at most two parameters. We presented an efficient algorithm for computing the abstraction operation. However, if $|\textsc{pars}(f_j)| > 2$, we are forced to resort to expensive elimination techniques using Fourier-Motzkin elimination [35] or the double description method [31]. An alternative is to perform the elimination approximately to obtain $\psi[\boldsymbol{c}, d]$ such that

$$\psi[\boldsymbol{c}, d] \ \models \ (\exists\, \boldsymbol{\lambda})\ P[\boldsymbol{c}, d, \boldsymbol{\lambda}]\,.$$

Note that, since we are working with the dual representation, we seek to *under-approximate* the result of the elimination.

We consider two methods for deriving an under-approximation of ψ_j by computing some of its generators. The under-approximations presented here are similar to those used in earlier work by Kanade *et al.* involving one of the authors of this paper [25].

Finding Vertices By Linear Programming: We simply solve k linear programs, each yielding a vertex, by choosing some fixed or random objective $f_i[\boldsymbol{c}, d]$ and solving $\mathbf{min.}\ \ f_i\ \mathbf{s.t.}\ \ P[\boldsymbol{c}, d, \boldsymbol{\lambda}],\ \boldsymbol{\lambda} \geq 0$. The solution yields a vertex (or a ray) of the feasible region $(\boldsymbol{c}, d, \boldsymbol{\lambda})$. By projecting $\boldsymbol{\lambda}$ out, we obtain a point inside ψ_j or a ray of ψ_j, alternatively.

Finding Vertices By Ray-Shooting: Choose a point $(\boldsymbol{c}_0, d_0)$ known to be inside ψ_j (using a linear programming). Let $\boldsymbol{r}$ be some direction. We seek a point of the form $(\boldsymbol{c}_0, d_0) + \gamma r$ for some $\gamma \in \mathbb{R}$ by solving the following LP:

$$\mathbf{max.}\ \ \gamma\ \mathbf{s.t.}\ \ ((\boldsymbol{c}_0, d_0) + \gamma \boldsymbol{r}) \in [\![P[\boldsymbol{c}, d, \boldsymbol{\lambda}] \ \wedge\ \boldsymbol{\lambda} \geq 0]\!]\,.$$

Any solution yields a point inside ψ_j by projection. If the LP is unbounded, we conclude that $\boldsymbol{r}$ is a ray of ψ_j.

We refer the reader to Kanade et al. [25] for more details on how under-approximation techniques described above can be used to control the size of the polyhedral representation.

6 Applications

In this section, we present two applications of the ideas presented thus far. In doing so, we seek to answer the following question: *how do we derive interesting* *templates without having the user of the analysis tool specify them*? Note that our previous work provides heuristics for guessing an initial set of templates in a property directed manner and enriching this set based on pre-condition computations [34].

Improving Fixed-Points Template polyhedra can be used to improve post-fixed points that are computed using other known techniques for numerical domains. For instance, suppose we perform a polyhedral analysis for some program and compute invariants $\varphi : e_1 \leq 0, \ \ldots, \ e_m \leq 0$ for linear expressions $e_1, \ldots, e_m$. A local fixed point improvement template could specify entries of the form (e_i, f_i), wherein $e_i \leq 0$ is an inequality involved in the polyhedral invariant and f_i is a bilinear form involving the program variables that do not appear in e_i. Improvement of fixed points can be attempted under more restrictive settings where the inequalities $e_i \leq 0$ are obtained from a less expensive analysis.

Variable Packing Packing of variables is a commonly used tactic to tackle the high complexity of domain operations in numerical domains [4, 22]. An alternative to packing would involve using generalized templates by specifying expressions of the form $0 \leq c_1x_1 + \cdots + c_kx_k$ as the template at a particular location where a variable pack of $\{x_1, \ldots, x_k\}$ is to be employed. This allows us to seamlessly reason about multiple packs at different program locations and integrate the reasoning about packs during the course of the analysis.

7 Implementation and Experiments

In this section, we describe our experimental evaluation of several of the ideas presented in this paper on some benchmarks.

Implementation The generalized template polyhedral domain presented in this paper has been implemented using the Parma Polyhedral Library (PPL) for manipulating polyhedra [2]. Our implementation supports general templates with arbitrarily many RHS parameters using generator enumeration capabilities of PPL. The special case of templates with at most two parameters has been implemented using the LP solver available in PPL. The analysis engine uses a naive iteration with delayed widening followed by the narrowing of post-fixed points, for a bounded number of iterations.

Note 4. In this paper, we have described an algorithm for performing abstraction using polyhedral projection (elimination of multiplier variables for dualization). Our implementation uses a different technique based directly on generator enumeration: to compute $\alpha(\varphi)$ we simply compute the generators of φ and derive the dual constraints for each template entry from the generators of φ. This has been done since PPL has an efficiently engineered generator enumeration algorithm. Therefore, our implementation performs a single vertex enumeration up-front and re-uses the result for each template entry, as opposed to eliminating multipliers for each template entry.

Template Construction The overall template construction is based on a pre-computed fixed point. For our experiments, we use the polyhedral domain

Table 1. Experimental evaluation of the GTP domain on some benchmark examples using various methods for forming templates. All timings are in seconds. #tpl: # of templates, #entries: average/min/max number of entries per template, # FP Impr: # of templates that improve the fixed point, T_{poly}: polyhedral analysis time, T_{flp}: average time per template with FLP, T_{ve}: average time per template with vertex enum.

Name	#var	#trs	M1						
			#tpl	#entries		#fp	T_{poly}	T_{flp}	T_{ve}
				avg	(min,max)	imp		avg	avg
lifo	8	10	42	1.0	1 , 1	6	0.02	0.02	0.00
tick3i	8	9	78	1.0	1 , 1	0	0.03	0.02	0.01
pool	9	6	64	1.1	1 , 2	8	0.01	0.03	0.01
ttp	9	20	968	1.4	1 , 2	64	0.06	0.02	0.01
cars2	10	7	149	1.1	1 , 2	4	13.5	0.06	0.01
req-gr	11	8	81	1.3	1 , 2	25	0.01	0.06	0.02
cprot	12	8	100	1.3	1 , 2	27	0.02	0.07	0.02
CSM	13	8	131	1.4	1 , 2	27	0.02	0.06	0.02
cprod	18	14	233	1.2	1 , 2	38	0.06	0.13	0.05
incdec	32	28	854	1.1	1 , 2	103	1	0.41	0.23
mesh2x2	32	32	940	1.1	1 , 2	212	1	0.51	0.29
bigjava	45	37	1856	1.4	1 , 2	129	239	0.85	0.48

Name	#tpl	M2					M3			
		#entries		T_{flp}	T_{ve}	#fp	#entries		T	#fp
		avg	(min,max)	avg	avg	impr	avg	(min,max)	avg	impr
lifo	7	6.0	6 , 6	0.6	0.0	6	1.0	1 , 1	0.0	0
tick3i	14	5.6	3 , 7	0.1	0.0	0	1.0	1 , 1	0.0	0
pool	10	7.1	4 , 8	0.5	0.0	8	1.2	1 , 2	0.0	3
ttp	140	9.5	5 , 16	0.4	0.0	32	1.4	1 , 2	0.0	2
cars2	18	9.3	7 , 18	1.4	0.2	3	1.1	1 , 2	0.1	1
req-gr	9	11.3	10 , 16	1.5	0.2	9	1.3	1 , 2	0.0	8
cprot	10	13.4	11 , 20	2.0	0.3	10	1.4	1 , 2	0.0	9
CSM	12	15.3	7 , 22	2.5	0.5	12	1.4	1 , 2	0.0	7
cprod	16	18.1	12 , 32	7.4	1.5	15	1.3	1 , 2	0.1	13
incdec	29	32.7	16 , 60	43.6	87.8	29	1.1	1 , 2	1.1	7
bigjava	44	60.4	34 , 86	67.1	50.4	42	1.4	1 , 2	1.7	8

to generate these fixed points. Next, using each inequality $e \leq 0$ in the result computed by the polyhedral domain, we compare three methods for forming templates: (**M1**) Create *multiple templates* per invariant inequality, each template consisting of a single entry of the form $e \leq c_j x_j + d_j$, for each $x_j \not\in \textsc{vars}(e)$, (**M2**) Create *one template* per inequality, each template consisting of multiple entries of the form $e \leq c_j x_j + d_j$, for each $x_j \not\in \textsc{vars}(e)$, and finally, (**M3**) Create *one template* per inequality, consisting of a single entry of the form $e \leq \sum_j c_j x_j + d$ for all variables $x_j \not\in \textsc{vars}(e)$. For equalities $e = 0$, we consider entries of the form $e \leq c_j x_j + d_j$ and $e \geq c'_j x_j + d'_j$, modifying the methods above appropriately.

Furthermore, for methods **M1** and **M2**, we compare the implementation using vertex enumeration against the fractional LP implementation for the key primitive of abstraction, which also underlies the implementation of the join and post-condition operations. Since the fractional LP algorithm produces the same result as the vertex enumeration, we did not observe any difference in the generated invariants. We compare the running times for these techniques.

Results Table 1 reports on the time taken for the initial polyhedral analysis run, the number of templates, avg/min/max of the number of entries per template and the number of analysis runs that discovered a stronger invariant. The benchmarks have been employed in our previous work [34] as well as other tools [3]. They range in size from small (≤ 10 program variables) to large (≥ 30 program variables). The initial widening delay was set to 4 for each run of the polyhedral analysis.

We note that our domain can be used to infer invariants that are not consequences of the polyhedral domain invariants. This validates the usefulness of generalized templates for improving fixed points. The results using method **M2** show that analysis using generalized templates can be expensive when the number of entries in each template is large. However, method **M2** also provides a fixed point improvement for almost all the templates that were run, whereas in method **M1**, roughly 10% of the runs resulted in a fixed point improvement. Even though method **M3** does not allow us to use fractional linear programming, it presents a good tradeoff between running time and the possibility of fixed point improvement.

The comparison between running times for the fractional LP approach and the basic approach using vertex enumeration yields a surprising result: on most larger benchmarks, the fractional LP approach (which is polynomial time, in theory) is *slower* than the vertex enumeration. A closer investigation reveals that each vertex enumeration pass yields very few (~ 10) vertices and extreme rays. Therefore, the well-engineered generator enumeration implementation in PPL invoked once, though exponential in worst-case, can outperform our technique which requires repeated calls to LP solvers. On the other hand, the constraints in the dual form $P[c, d, \boldsymbol{\lambda}]$ obtained by Farkas' Lemma are common to all the LP and FLP solver invocations during a run of our algorithm. Incremental solvers used in SMT solvers such as Z3 [13], could potentially be adapted to speed our algorithm up.

Finally, the comparison here has been carried out against invariants obtained by polyhedral analysis. In the future, we wish to evaluate our domain against other types of invariant generation tools such as VS3 [38] and InvGen [17].

8 Related Work

Polyhedral analysis was first introduced by Cousot & Halbwachs [12] using the abstract interpretation framework [11]. Further work has realized the usefulness of domains based on convex polyhedra to the analysis of hybrid systems [19], control software [4], and systems software [36, 21, 22, 14]. However, these advances have avoided the direct use of the original polyhedral domain due to the high cost of domain operations such as convex hulls and post-conditions [35]. In spite of impressive advances made by polyhedra libraries such as PPL, the combinatorial explosion in the number of generators cannot be overcome, in general [2]. However, it is also well-known that these operations can be performed at a lower polynomial cost (in terms of the problem size and the output size) for two dimensional polyhedra [9, 35]. This fact has been used to design fast program analysis techniques to compute invariants that involve at most two program variables [37]. Further restricting the coefficients to units (± 1), yields the weakly-relational octagon domain wherein the process of reasoning about the constraints can be reduced to graph operations [28].

The restriction to at most two variables for each invariant and unit coefficients has been observed to be quite useful for proving absence of runtime errors [21]. On the other hand, current evidence suggests that invariants involving three or more variables are often crucial for proving safety of string operations and user-defined assertions [33].

Template polyhedral domains attempt to avoid the exponential complexity of polyhedral domain operations by fixing templates that represent the form of the desired invariants [34]. Recently, there have been many advances in our understanding of template polyhedral domains. Techniques such as policy iteration and strategy iteration have exploited ideas from max-plus algebra to show that the least fixed point in weakly-relational domains is computable [15, 16]. These techniques have been extended to handle templates with non-linear inequalities using ideas from convex programming [1, 16]. The possibility that some of these results may generalize to the domain presented in this paper remains open.

The problem of modular template polyhedral analysis has been studied by Monniaux [30], wherein summaries for templates are derived by dualization using Farkas' lemma followed by quantifier elimination in the theory of linear arithmetic. This work has also extended the template domain to handle floating point semantics for control software. Srivastava and Gulwani [38] present a verification technique using templates that can specify a richer class of templates involving quantifiers and various Boolean combinations of predicates. However, their approach for computing transformers works by restricting the coefficients to a finite set and "*bit blasting*" using SAT solvers. The logahedron domain uses a similar restriction [20]. Invariants with unit coefficients are quite useful for reasoning about runtime properties of low-level code. However, it is as yet unclear if verifying functional properties in domains such as control systems will necessitate invariants with arbitrary coefficients.

9 Conclusion

We have presented a generalization of the template domain with arbitrary bilinear forms, along with efficient algorithms for computing the domain operations when the template has at most two parameters per entry. Our experimental results demonstrate that our techniques can be applied to strengthen the invariants computed by other numerical domains. In the future, we plan to investigate techniques such as policy and strategy iteration for analysis over generalized templates. Extensions to non-linear templates and the handling of floating point semantics are also of great interest.

References

1. A. Adjé, S. Gaubert, and E. Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In *ESOP*, volume 6012 of *LNCS*, pages 23–42. Springer, 2010.
2. R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Static Analysis Symposium*, volume 2477 of *LNCS*, pages 213–229. Springer, 2002.
3. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: fast accelereation of symbolic transition systems. In *CAV*, volume 2725 of *LNCS*, pages 118–121. Springer, 2003.
4. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. 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*, volume 2566 of *LNCS*, pages 85–108. Springer, 2005.
5. S. Boyd and S. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004. Online http://www.stanford.edu/ boyd/cvxbook.html.
6. S. Chatterjee, S. K. Lahiri, S. Qadeer, and Z. Rakamaric. A reachability predicate for analyzing low-level software. In *TACAS*, volume 4424 of *LNCS*, pages 19–33. Springer, 2007.
7. R. Clarisó and J. Cortadella. The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139, January 2007.
8. M. A. Colón. *Deductive Techniques for Program Analysis.* PhD thesis, Stanford University, 2003.
9. T. Corman, C. F. Leiserson, and R. Rivest. *Introduction to Algorithms.* McGraw Hill, 1990.
10. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. ISOP'76*, pages 106–130. Dunod, Paris, France, 1976.
11. P. Cousot and R. Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
12. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *POPL'78*, pages 84–97, Jan. 1978.
13. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
14. P. Ferrara, F. Logozzo, and M. Fähndrich. Safer unsafe code for .net. In *OOPSLA*, pages 329–346. ACM, 2008.
15. S. Gaubert, E. Goubault, A. Taly, and S. Zennou. Static analysis by policy iteration on relational domains. In *ESOP*, volume 4421 of *LNCS*, pages 237–252. Springer, 2007.
16. T. Gawlitza and H. Seidl. Precise fixpoint computation through strategy iteration. In *ESOP*, volume 4421 of *LNCS*, pages 300–315. Springer, 2007.
17. A. Gupta and A. Rybalchenko. InvGen: An efficient invariant generator. In *CAV*, volume 5643 of *LNCS*, pages 634–640, 2009.

18. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
19. T. A. Henzinger and P. Ho. HyTech: The Cornell hybrid technology tool. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 265–293. Springer, 1995.
20. J. Howe and A. King. Logahedra: A new weakly relational domain. In *ATVA'09*, volume 5799 of *LNCS*, pages 306–320. 2009.
21. F. Ivančić, S. Sankaranarayanan, I. Shlyakhter, and A. Gupta. Buffer overflow analysis using environment refinement, 2009. Draft (2009).
22. F. Ivančić, I. Shlyakhter, A. Gupta, and M. K. Ganai. Model checking C programs using F-SOFT. In *ICCD*, pages 297–308. IEEE Computer Society, 2005.
23. R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18 – 21, 1973.
24. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
25. A. Kanade, R. Alur, F. Ivančić, S. Ramesh, S. Sankaranarayanan, and K. Sashidhar. Generating and analyzing symbolic traces of Simulink/Stateflow models. In *Computer-aided Verification (CAV)*, volume 5643 of *LNCS*, pages 430–445, 2009.
26. F. Logozzo and M. Fähndrich. Pentagons: A weakly relational abstract domain for the efficient validation of array accesses. *Sci. Comp. Prog.*, 75(9):796–807, 2010.
27. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, New York, 1995.
28. A. Miné. A new numerical abstract domain based on difference-bound matrices. In *PADO II*, volume 2053 of *LNCS*, pages 155–172. Springer, May 2001.
29. A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI'06*, volume 3855 of *LNCS*, pages 348–363. Springer, 2006.
30. D. Monniaux. Automatic modular abstractions for template numerical constraints. *Logical Methods in Computer Science*, 6(3), 2010.
31. T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In *Contributions to the theory of games*, volume 2 of *Annals of Mathematics Studies*, pages 51–73. Princeton University Press, 1953.
32. S. Sankaranarayanan, M. A. Colón, H. Sipma, and Z. Manna. Efficient strongly relational polyhedral analysis. In *VMCAI'06*, volume 3855/2006 of *LNCS*, pages 111–125, 2006.
33. S. Sankaranarayanan, F. Ivančić, and A. Gupta. Program analysis using symbolic ranges. In *SAS*, volume 4634 of *LNCS*, pages 366–383, 2007.
34. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, volume 3385 of *LNCS*, January 2005.
35. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.
36. A. Simon. *Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities*. Springer, 2008.
37. A. Simon, A. King, and J. M. Howe. Two variables per linear inequality as an abstract domain. In *LOPSTR*, volume 2664 of *LNCS*, pages 71–89. Springer, 2003.
38. S. Srivastava and S. Gulwani. Program verification using templates over predicate abstraction. In *PLDI '09*, pages 223–234, New York, NY, USA, 2009. ACM.