Efficient Strongly Relational Polyhedral Analysis

Sriram Sankaranarayanan[1,3], Michael A. Colón[2], Henny Sipma[3], Zohar Manna[3]

[1] NEC Laboratories America, Princeton, NJ
srirams@nec-labs.com
[2] Center for High Assurance Computer Systems, Naval Research Laboratory,
colon@itd.nrl.navy.mil
[3] Computer Science Department, Stanford University, Stanford, CA 94305-9045
(sipma,zm)@theory.stanford.edu *

Abstract. Polyhedral analysis infers invariant linear equalities and inequalities of imperative programs. However, the exponential complexity of polyhedral operations such as image computation and convex hull limits the applicability of polyhedral analysis. Weakly relational domains such as intervals and octagons address the scalability issue by considering polyhedra whose constraints are drawn from a restricted, user-specified class. On the other hand, these domains rely solely on candidate expressions provided by the user. Therefore, they often fail to produce strong invariants.

We propose a polynomial time approach to strongly relational analysis. We provide efficient implementations of join and post condition operations, achieving a trade off between performance and accuracy. We have implemented a strongly relational polyhedral analyzer for a subset of the C language. Initial experimental results on benchmark examples are encouraging.

1 Introduction

Polyhedral analysis seeks to discover invariant linear equality and inequality relationships among the variables of an imperative program. The computed invariants are used to establish safety properties such as freedom from buffer overflows. The standard approach to polyhedral analysis is through a fixed point iteration in the domain of convex polyhedra [9]. Complexity considerations, however, restrict its application to small systems. Libraries such as NewPolka [13] and PPL [2] have made strides towards addressing some of these tractability issues, but still the approach remains impractical for large systems.

At the heart of this intractability lies the need to repeatedly convert between constraint and generator representations of polyhedra. Efficient analysis techniques work on restricted forms of polyhedra wherein such a conversion can be avoided. *Weakly relational domains* such as octagons [17], intervals [7], octahedra [6] and the TCM domain [18], avoid these conversions by considering

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363 and CCF-0430102, by ARO grant DAAD 19-01-1-0723 and by NAVY/ONR contract N00014-03-1-0939 and the Office of Naval Research.

polyhedra whose constraints are fixed a priori. The abstract domain of Simon et al. [19] considers polyhedra with at most two variables per constraint. Using these syntactic restrictions, the analysis can be carried out efficiently. However, the main drawback of such syntactic restrictions is the inability of the analysis to infer invariants that require expressions of an arbitrary form. Thus, in many cases, such domains may fail to prove the property of interest.

In this paper, we provide an efficient strongly relational polyhedral domain by drawing on ideas from both weak and strong relational analysis. We present alternatives to the join and post condition operations. In particular, we provide a new join algorithm, called *inversion join*, that works in polynomial time in the size of the input polyhedra, as opposed to the exponential space polyhedral join. We make use of linear programming to implement an efficient join and post condition operators, along with efficient inclusion checks and widening operators.

On the other hand, our domain operations are weaker than the conventional polyhedral domain operations, potentially yielding weaker invariants. Using a prototype implementation of our techniques, we have analyzed several sorting and string handling routines for buffer overflows. Our initial results are promising; our analysis performs better than the standard approaches while computing invariants that are sufficiently strong in practice.

Outline. Section 2 discusses the preliminary notions of polyhedra, transition systems and invariants. Section 3 discusses algorithms for domain operations needed for polyhedral analysis. Section 4 discusses the implementation and the results obtained on benchmark examples.

2 Preliminaries

We recall some standard results on polyhedra, followed by a brief description of system models and abstract interpretation. Throughout the paper, let $\mathcal{R}$ represent the set of reals and $\mathcal{R}^+ = \mathcal{R} \cup \{\pm\infty\}$ represent the extended real numbers.

Definition 1 (Linear Assertions). *A* linear expression *e is of the form $a_1x_1 + \cdots + a_nx_n + b$, wherein each $a_i \in \mathcal{R}$ and $b \in \mathcal{R}^+$. The expression is said to be* homogeneous *if $b = 0$. A* linear constraint *is of the form $a_1x_1 + \cdots + a_nx_n + b \bowtie 0$, with $\bowtie \in \{\geq, =\}$. A* linear assertion *is a finite conjunction of linear inequalities.*

Note that the linear inequality $e + \infty \geq 0$ represents the assertion *true*, whereas the inequality $e - \infty \geq 0$ represents *false*. Since each equality $e = 0$ can be represented as a conjunction of two inequalities, an assertion can be written in matrix form as $A\boldsymbol{x} + \boldsymbol{b} \geq \boldsymbol{0}$, where A is a $m \times n$ matrix, while $\boldsymbol{x}$ and $\boldsymbol{b}$ are n and m-dimensional vectors, respectively. The set of points in $\mathcal{R}^n$ satisfying a linear assertion is called a *polyhedron.*

The representation of a polyhedron by a linear assertion is known as its *constraint representation.* Alternatively, a polyhedron may be represented explicitly by a finite set of vertices and rays, known as the *generator representation.* Each representation may be exponentially larger than the other. For instance, the n

dimensional hypercube is represented by $2n$ constraints and 2^n generators. Efficient libraries of conversion algorithms such as the new PolKa [12] and the Parma Polyhedral Library (PPL) [2] have made significant improvements to the size of the polyhedra for which the conversion is possible. Nevertheless, this conversion still remains intractable for large polyhedra involving 100s of variables and constraints.

A *Template Constraint Matrix* (TCM) T is a finite set of homogeneous linear expressions over $\boldsymbol{x}$. Given an assertion φ, its expressions induce a TCM T which we shall denote as $\mathsf{Ineqs}(\varphi)$. If φ is represented as $A\boldsymbol{x}+\boldsymbol{b} \geq 0$ then $\mathsf{Ineqs}(\varphi) : A\boldsymbol{x}$.

Linear Programming We briefly describe the theory of linear programming. Details may be found in standard textbooks [5].

Definition 2 (Linear Programming). *A canonical instance of the* linear programming (LP) problem *is of the form*

$$\mathsf{minimize}\ e\ \mathsf{subject\ to}\ \varphi\,,$$

for assertion φ and a linear expression e, called the objective function.

The goal is to determine the solution of φ for which e is minimal. A LP problem can have one of three results: (1) an optimal solution; (2) $-\infty$, i.e, e is unbounded from below in φ; (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 is worst-case exponential in the number of constraints. The popular SIMPLEX algorithm (due to Danzig [10]) employs a sophisticated hill climbing strategy that converges on an optimal vertex without necessarily enumerating all vertices. In theory, the technique is worst-case exponential. The SIMPLEX method is efficient over most problems. Interior point methods such as *Karmarkar's* algorithm and other techniques based on ellipsoidal approximations are guaranteed to solve linear programs in polynomial time. Using an open-source implementation of SIMPLEX such as GLPK [15], massive LP instances involving tens of thousands (10^4 and beyond) of variables and constraints can be solved efficiently.

Programs and Invariants

We assume programs over real valued variables without any function calls. The program is represented by a linear transition system also known as a *control flow graph.*

Definition 3 (Linear Transition Systems). *A* linear transition system *(LTS) $\Pi : \langle L, \mathcal{T}, \ell_0, \Theta\rangle$ over a set of variables V consists of*

- *L: a set of locations (cutpoints);*

- $\mathcal{T}$: *a set of transitions (edges), where each transition* $\tau : \langle \ell_i, \ell_j, \rho_\tau \rangle$ *consists of a pre-location* ℓ_i, *a post-location* ℓ_j, *and a transition relation* ρ_τ, *represented as a linear assertion over* $V \cup V'$, *where* V *denotes the values of the variables in the current state, and* V' *their values in the next state;*
- $\ell_0 \in L$: *the initial location;*
- Θ: *a linear assertion over* V *specifying the initial condition.*

A *run* of a LTS is a sequence $\langle m_0, s_0 \rangle, \langle m_1, s_1 \rangle, \ldots$, with $m_i \in L$ and s_i a valuation of V, also called a *state*, such that

- Initiation: $m_0 = \ell_0$, and $s_0 \models \Theta$
- Consecution: for all $i \geq 0$ there exists a transition $\tau : \langle \ell_j, \ell_k, \rho_\tau \rangle$ such that $m_i = \ell_j$, $m_{i+1} = \ell_k$, and $\langle s_i, s_{i+1} \rangle \models \rho_\tau$.

A state s is *reachable* at location ℓ if $\langle \ell, s \rangle$ appears in some run.

A given linear assertion ψ is a *linear invariant* of a linear transition system (LTS) at a location ℓ iff it is satisfied by every state reachable at ℓ. An *assertion map* associates each location of a LTS to a linear assertion. An assertion map η is *invariant* if $\eta(\ell)$ is an invariant, for each $\ell \in L$. In order to prove a given assertion map invariant, we use the inductive assertions method due to Floyd (see [16]).

Definition 4 (Inductive Assertion Maps). *An assertion map* η *is inductive iff it satisfies the following conditions:*

Initiation: $\Theta \models \eta(\ell_0)$,
Consecution: *For each transition* $\tau : \langle \ell_i, \ell_j, \rho_\tau \rangle$, $(\eta(\ell_i) \ \wedge \ \rho_\tau) \ \models \eta(\ell_j)'$. *Note that* $\eta(\ell_j)'$ *refers to* $\eta(\ell_j)[V|V']$ *with variables in* V *substituted by their corresponding primed variables in* V'.

It is well known that any inductive assertion map is invariant. However, the converse need not be true. The standard technique for proving an assertion invariant is to find an inductive assertion that strengthens it.

Linear Relations Analysis

Linear relation analysis seeks an inductive assertion map for the input program, labeling each location with a linear assertion. Analysis techniques are based on the theory of *Abstract Interpretation* [8] and specialized for linear relations by Cousot and Halbwachs [9]. The technique starts with an initial assertion map, and weakens it iteratively using the *post*, *join* and the *widening* operators. When the iteration converges, the resulting map is guaranteed to be inductive, and hence invariant. Termination is guaranteed by the design of the widening operator.

The *post condition* operator takes an assertion φ and a transition τ, and computes the set of states reachable by τ from a state satisfying φ. It can be expressed as

$$post(\varphi, \tau) : \ (\exists V_0)(\varphi(V_0) \ \wedge \ \rho_\tau(V_0, V))$$

Standard polyhedral operations can be used to compute post. However, more efficient strategies for computing post exist when ρ_τ has a special structure. Given assertions $\varphi_{\{1,2\}}$ such that $\varphi_1 \models \varphi_2$, the *standard widening* $\varphi_1 \nabla \varphi_2$ is an assertion φ that contains all the inequalities in φ_1 that are satisfied by φ_2. The details along with key mathematical properties of widening are described in [9, 8], and enhanced versions appear in [12, 4, 1]. As mentioned earlier, the analysis begins with an initial assertion map defined by $\eta_0(\ell_0) = \Theta$, and $\eta_0(\ell) = \mathit{false}$ for $\ell \neq \ell_0$. At each *step*, the map η_i is updated to map η_{i+1} as follows:

$$\eta_{i+1}(\ell) = \eta_i(\ell)\ \langle \textsc{op} \rangle \left[\eta_i(\ell) \bigsqcup_{\tau_j \equiv \langle \ell_j, \ell, \rho \rangle} (\mathit{post}(\eta_i(\ell_j), \tau_j)) \right],$$

where OP is the join ($\sqcup$) operator for a propagation step, and the widening (∇) operator for a widening step. The overall algorithm requires a predefined *iteration strategy*. A typical strategy carries out a fixed number of initial propagation steps, followed by widening steps until termination.

Linear Assertion Domains

Linear relation analysis is performed using a forward propagation wherein polyhedra are used to represent sets of states. Depending on the family of polyhedra considered, such domains are classified as *weakly relational* or *strongly relational*.

Let $T = \{e_1, \ldots, e_m\}$ be a TCM. The weakly relational domain induced by T consists of assertions $\bigwedge_{e_i \in T} e_i + b_i \geq 0$ for $b_i \in \mathcal{R}^+$. TCMs and their induced weakly relational domain are formalized in our earlier work [18]. Given a weakly relational domain defined by a TCM T and a linear transition system Π, we seek an inductive assertion map η such that $\eta(\ell)$ belongs to the domain of T for each location ℓ. Many weakly relational domains have been studied: Intervals, octagons and octahedra are classical examples.

Example 1 (Weakly Relational Analysis). Let X be the set of system variables. The interval domain is defined by the TCM consisting of expressions $T_X = \{\pm x_i \mid x_i \in X\}$. Thus, any polyhedron belonging to the domain is an interval expression of the form $\bigwedge (x_i + a_i \geq 0\ \wedge\ -x_i + b_i \geq 0)$. The goal of interval analysis is to discover the coefficients $a_i, b_i \in \mathcal{R}^+$ representing the bounds for each variable x_i at each location of the program [7].

The octagon domain of Miné subsumes the interval domain by considering additional expressions of the form $\pm x_i \pm x_j$ such that $x_i, x_j \in X$ [17]. The octahedron domain due to Clarisó and Cortadella considers expressions of the form $\sum_i a_i x_i$ such that $a_i \in \{-1, 0, 1\}$ [6].

It is possible to carry out the analysis in any weakly relational domain efficiently [18].

Theorem 1. *Given a TCM T and a linear system Π, all the domain operations for the weakly relational analysis of Π in the domain induced by T can be performed in time polynomial in $|T|$ and $|\Pi|$.*

```
integer x,y where (x = 1 ∧ x ≥ y)
ℓ0 : while true do
      if (x ≥ y) then
         (x, y) := (x + 2, y + 1)
      else
         (x, y) := (x + 2, y + 3)
      end if
   end while
```

Fig. 1. An example program.

Weakly relational domains are appealing since the analysis in these domains is scalable to large systems. On the other hand, the invariants they produce are often imprecise. For instance, even if $e_i + a_i \geq 0$ is invariant for some expression e_i in the TCM, its proof may require an inductive strengthening $e_j + a_j \geq 0$, where e_j is not in the TCM.

A *strongly relational* analysis does not syntactically restrict the polyhedra considered. The polyhedral domain is not restricted in its choice of invariant expressions, and is potentially more precise than a weakly relational domain. The main drawback, however, is the high complexity of the domain operations. Each domain operation requires conversions from the constraint to the generator representation and back. Popular implementations of strongly relational analysis require worst case exponential space due to repeated representation conversions.

Example 2. Consider the system in Figure 1. Interval and octagon domains both discover the invariant $\infty \geq x \geq 1$ at location ℓ_0. A strongly relational analysis such as polyhedral analysis discovers the invariant $x \geq 1 \ \wedge \ 3x - 2y \geq 1$, as does the technique that we present.

3 Domain Operations

The theory of Abstract Interpretation provides a framework for the design of program analyses. A sound program analysis can be designed by constructing an abstract domain with the following domain operations:

Join (union) Given two assertions φ_1, φ_2 in the domain, we seek an assertion φ such that $\varphi_1 \models \varphi$ and $\varphi_2 \models \varphi$. In many domains, it is possible to find the strongest possible φ satisfying this condition. The operation of computing such an assertion is called the *strong join*.

Post Condition Given an assertion φ, and a transition relation ρ_τ, we seek an assertion ψ such that $\varphi[V] \ \wedge \ \rho_\tau[V, V'] \models \psi[V']$. A *strong post condition* operator computes the strongest possible assertion ψ satisfying this condition.

Widening Widening ensures the termination of the fixed point iteration.

Additionally, inclusion tests between assertions are important for detecting the termination of an iteration. Feasibility tests and redundancy elimination are also frequently used to speed up the analysis. We present several different join and post condition operations, each achieving a different trade off between efficiency and precision.

Join

Given two linear assertions φ_1 and φ_2 over a vector $\boldsymbol{x}$ of system variables, we seek a linear assertion φ, such that both $\varphi_1 \models \varphi$ and $\varphi_2 \models \varphi$.

Strong Join. The strong join seeks the strongest assertion φ (denoted $\varphi_1 \sqcup_s \varphi_2$) subsuming both φ_1 and φ_2. In the domain of convex polyhedra, this is known as the *polyhedral convex hull* and is obtained by computing the generator representations of φ_1 and φ_2. The set of generators of φ is the union of those of φ_1 and φ_2. This representation is then converted back into the constraint representation. Due to the repeated representation conversions, the strong join is worst-case exponential space in the size of the input assertions.

Example 3. Consider the assertions

$$\begin{aligned}\varphi_1 &: x - y \leq 5 \wedge y + x \leq 10 \wedge -10 \leq \ x \leq 5\\ \varphi_2 &: x - y \leq 9 \wedge y + x \leq 5 \ \ \wedge -9 \leq \ x \leq 6\end{aligned}$$

Their strong join $\varphi_1 \sqcup_s \varphi_2$, generated by the union of their vertices, is

$$\varphi : \ 6x + y \leq 35 \ \wedge \ y + 3x + 45 \geq 0 \ \wedge \ x - y \leq 9 \ \wedge \ x + y \leq 10 \ \wedge \ -10 \leq x \leq 6\,.$$

Weak Join. The weak join operation is inspired by the join used in weakly relational domains.

Definition 5 (Weak Join). *The weak join of two polyhedra φ_1, φ_2 is computed as follows:*
1. Let TCM $T = \mathsf{Ineqs}\,(\varphi_1) \cup \mathsf{Ineqs}\,(\varphi_2)$ be the set of inequality expressions that occur in either of $\varphi_{\{1,2\}}$. Recall that each equality in φ_1 or φ_2 is represented by two inequalities in T.
2. For each expression e_i in T, we compute the values a_i and b_i using linear programming, as follows:

$$\begin{aligned}a_i &= \mathsf{minimize}\ e_i\ \mathsf{subject\ to}\ \varphi_1\\ b_i &= \mathsf{minimize}\ e_i\ \mathsf{subject\ to}\ \varphi_2\end{aligned}$$

It follows that $\varphi_1 \models (e_i \geq a_i)$ and $\varphi_2 \models (e_i \geq b_i)$.
3. Let $c_i = \min(a_i, b_i)$. Therefore, both $\varphi_1, \varphi_2 \models (e_i \geq \min(a_i, b_i) \equiv c_i)$.
The weak join $\varphi_1 \sqcup_w \varphi_2$ is given by the assertion $\bigwedge_{e_i \in T}\ e_i \geq c_i$.

The advantage of the weak join is its efficiency: it can be computed using LP queries, where both the number of such queries and the size of each individual query is polynomial in the input size. On the other hand, the weak join does not discover any new relations. It is weaker than the strong join, as shown by the argument above (and strictly so, as shown by the following Example).

Example 4. Consider the assertions φ_1, φ_2 from Example 3 above. The TCM T and the a_i, b_i values are shown in the table below:

$T:$

#	*Relation*		$a_i(\varphi_1)$	$b_i(\varphi_2)$
1	$y - x$	$\geq$	-5	-9
2	$-y - x$	$\geq$	-10	-5
3	x	$\geq$	-10	-9
4	$-x$	$\geq$	-5	-6

The weak join is given by

$$\varphi_w: \ (y - x \geq -9 \ \wedge \ -y - x \geq -10 \ \wedge \ x \geq -10 \ \wedge \ -x \geq -6)\,.$$

This result is strictly weaker than the strong join computed in Example 3.

Restricted Joins. The weak join shown above is more efficient than the strong join. However, this efficiency comes at the cost of precision. We therefore seek efficient alternatives to strong and weak join. The k-restricted join (denoted $\sqcup_k$) improves upon the weak join as follows:

1. Choose a subset of inequalities from φ_1, φ_2, each of cardinality at most k. Let ψ_1 and ψ_2 be the assertions formed by the chosen inequalities. In general, ψ_1, ψ_2 may contain different sets of inequalities, even different cardinalities. Note that $\varphi_i \models \psi_i$ for $i = 1, 2$.
2. Compute the strong join $\psi_1 \sqcup_s \psi_2$ in *isolation.* Conjoin the results with the weak join $\varphi_1 \sqcup_w \varphi_2$.
3. Repeat step 1 for a different set of choices of $\psi_{\{1,2\}}$, while conjoining each such join to the weak join.

Since $\varphi_i \models \psi_i$, for $i = 1, 2$, it follows by the monotonicity of the strong join operation that $\varphi_1 \sqcup_s \varphi_2 \ \models \psi_1 \sqcup_s \psi_2$. Thus $\varphi_1 \sqcup_s \varphi_2 \models \varphi_1 \sqcup_k \varphi_2$ for each $k \geq 0$.

Let φ_1, φ_2 have at most m constraints. The k-restricted join requires vertex enumeration for $O(\binom{m}{k}^2)$ polyhedra with at most k constraints. As such, this join is efficient only if k is a small constant. We shall now provide an efficient $O(m^2)$ algorithm based on $\sqcup_2$, to improve the weak join.

Inversion Join. The inversion join is based on the 2-restricted join. Let T be the TCM and a_i, b_i be the values computed for the weak join as in Definition 5. Consider pairs of expressions $e_i, e_j \in T$ yielding the assertions

$$\begin{array}{l} \psi_1 : e_i \geq a_i \wedge e_j \geq a_j \\ \psi_2 : e_i \geq b_i \wedge e_j \geq b_j \end{array}$$

We use the structure of the assertions ψ_1, ψ_2 to perform their strong join analytically. The key notion is that of an *inversion.*

Definition 6 (Inversion). *Expressions $e_i, e_j \in T$ and corresponding coefficients a_i, a_j, b_i, b_j form an* inversion *iff the following conditions hold:*

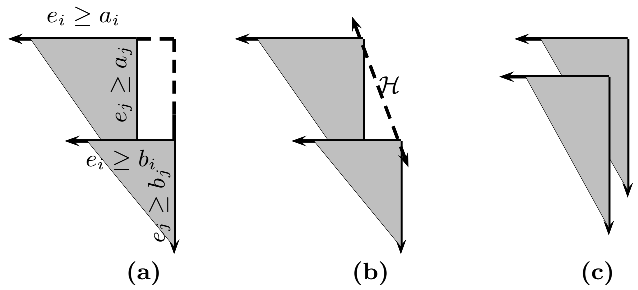

Fig. 2. (a) $a_i > b_i,\ a_j < b_j$, (b) Weak join is strictly weaker than strong join, (c) $a_i > b_i,\ a_j > b_j$: Weak join is the same as strong join.

1. *$a_i, a_j, b_i, b_j \in \mathcal{R}$, i.e, none of them is $\pm\infty$.*
2. *$e_i \neq \lambda e_j$ for $\lambda \in \mathcal{R}$, i.e, e_i, e_j are linearly independent.*
3. *$a_i < b_i$ and $b_j < a_j$ (or vice-versa).*

Example 5. Consider two "wedges" $\psi_1 :\ e_i \geq a_i \ \wedge\ e_j \geq a_j$ and $\psi_2 :\ e_i \geq b_i \ \wedge\ e_j \geq b_j$. Depending on the values of a_i, a_j, b_i, b_j, two cases arise as depicted in Figures 2(a,b,c). Figures 2(a,b) form an inversion. When this happens, the weak join (a) is strictly weaker than the strong join (b). Figure 2(c) does not form an inversion. The weak and strong joins coincide in this case.

Therefore, a strong join of polyhedra that form an inversion gives rise to a half space that is not discovered by the weak join. We now derive this "missing half-space" $\mathcal{H}$ analytically.

The half space subsumes both ψ_1 and ψ_2. A half-space that is a consequence of $\psi_1 :\ e_i \geq a_i \ \wedge\ e_j \geq a_j$ is of the form $\mathcal{H} :\ e_i + \lambda_{ij} e_j \geq a_i + \lambda_{ij} a_j$, for some $\lambda_{ij} \geq 0$. Similarly for ψ_2, we obtain $\mathcal{H} :\ e_i + \lambda_{ij} e_j \geq b_i + \lambda_{ij} b_j$. Equating coefficients, yields the equation $a_i + \lambda_{ij} a_j = b_i + \lambda_{ij} b_j$. The required value of λ_{ij} is

$$\lambda_{ij} = \frac{a_i - b_i}{b_j - a_j} \,.$$

Note that requiring $\lambda_{ij} > 0$ yields $a_i < b_i$ and $b_j < a_j$. Therefore, ψ_1, ψ_2 contain a non trivial common half-space iff they form an inversion.

Definition 7 (Inversion Join). *Given φ_1, φ_2 the inversion join $\varphi_1 \sqcup_{inv} \varphi_2$ is computed as follows:*

1. *Compute the TCM $T = \mathsf{Ineqs}(\varphi_1) \cup \mathsf{Ineqs}(\varphi_2)$.*
2. *For each $e_i \in T$ compute a_i, b_i as defined in Definition 5 using linear programming. At this point $\varphi_1 \models e_i \geq a_i$ and $\varphi_2 \models e_i \geq b_i$. Let $\varphi_w = \varphi_1 \sqcup_w \varphi_2$ be the weak join.*
3. *For each pair e_i, e_j, consider the expression $e_i + \lambda_{ij} e_j \geq a_i + \lambda_{ij} a_j$, with λ_{ij} as defined above.*
4. *The inversion join is the conjunction of φ_w and all the inversion expressions generated in Step 3. Optionally, simplify the result by removing redundant inequalities.*

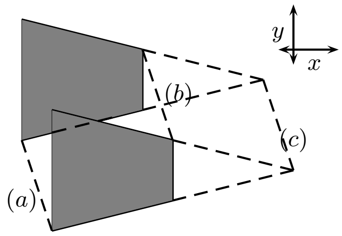

Fig. 3. Inversion join over two polyhedra (a), (b) and (c) are the newly discovered relations.

Example 6. Figure 3 shows the result of an inversion join over two input polyhedra φ_1, φ_2 used in Example 3. Example 4 shows the TCM T and the a_i, b_i values. There are three inversions

#	Expressions	Subsuming Half-Space
(a)	$\langle 1, 3 \rangle$	$y + 3x + 45 \geq 0$
(b)	$\langle 2, 4 \rangle$	$-y - 6x + 35 \geq 0$
(c)	$\langle 1, 2 \rangle$	$y - 9x + 65 \geq 0$

The "expressions" column in the table above refers to expressions by their row numbers in the table of Example 4. From Figure 3, note that (c) is redundant. Therefore the result of the join may require redundancy elimination (algorithm provided later in this section). This result is equivalent to the result of the strong join in Example 3.

Theorem 2. *Let φ_1, φ_2 be two polyhedra. It follows that*

$$\varphi_1 \sqcup_s \varphi_2 \ \models\ \varphi_1 \sqcup_{inv} \varphi_2 \ \models\ \varphi_1 \sqcup_w \varphi_2 \,.$$

The inversion join requires as many LP queries as the weak join and additional $O(m^2 n)$ arithmetic operations to compute inversions, where m is the number of inequalities in T and n, the dimensionality.

Note. The descriptions of the weak and inversion join treat each equality as two inequalities. The resulting join could be made more precise if additionally, the equality join defined by Karr's analysis [14] is computed and conjoined to the result. This can be achieved in time that is polynomial in the number of equalities.

Post Condition

The post condition computes the image of an assertion φ under a transition relation of the form $\xi \ \wedge \ \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{b}$. This is equivalent to the image of $\varphi \wedge \xi$ under the affine transformation $\boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{b}$. If the matrix A is invertible, then this image is easily computed by substituting $\boldsymbol{x} = A^{-1}(\boldsymbol{x}' - \boldsymbol{b})$ [9]. On the other hand, it is frequently the case that A is not invertible. We present three alternatives, the strong, weak and restricted post conditions.

Strong Post. The strong post is computed by first enumerating the generators of $\varphi \wedge \xi$. Each generator is transformed under the operation $A\boldsymbol{x} + \boldsymbol{b}$. The resulting polyhedron is generated by these images. Conversion back to the constraint representation completes the computation.

Weak Post. Weak post requires a TCM T' labeling the post location of the transition. Alternatively, this TCM may be derived from $\mathsf{Ineqs}\,(\eta(\ell'))$ where $\eta(\ell')$ labels the post-location. Given the existence of such a TCM, we may use the post operation defined for TCMs [18] to compute the weak post.

Note. The post condition computation for equalities can be performed separately using the image operation defined for Karr's analysis. This can be added to the result, thus strengthening the weak post.

k-Restricted Post The k-restricted post condition improves upon the weak post by using the monotonicity of the strong post operation (see [8]) similar to the k-restricted join algorithm. Therefore, considering a subset of up to k inequalities ψ, we may compute the strong post of ψ and add the result conjunctively to the weak post. The results improve upon the precision of weak post. As is the case for join, it is possible to treat the cases for $k = 1, 2$ efficiently.

Example 7. Consider the polyhedron $\varphi :\ x - y \geq 0 \ \wedge\ x \leq 0 \ \wedge\ x + y + 3 \leq 0$ and the transformation $x := x + 3, y := 0$. Consider the TCM T $= \{x - y, x + y, y - x, -x - y\}$. The weak post of φ w.r.t T is computed by finding bounds for each expression. For instance the bound for $x - y$ is discovered by solving:

$$\mathsf{minimize}\ x' - y'\ \mathsf{s.t.}\ \varphi \ \wedge\ x' = x + 3 \ \wedge\ y' = 0$$

The overall weak post is obtained by solving 4 LPs, one for each element of T,

$$\varphi_w :\ 3 \geq x - y \geq 1.5 \ \wedge\ 3 \geq x + y \geq 1.5\,.$$

This is strictly weaker than the strong post $\varphi_s :\ 3 \geq x \geq 1.5 \ \wedge\ y = 0$. The 1-restricted post computes the post condition of each half-space in φ separately. This yields the result $y = 0$ for all the three half-spaces. Conjoining the 1-restricted post with the weak post yields the same result as the strong post in this example.

Note. The projection operation, an important primitive for interprocedural analysis, can be implemented along the same lines as the post condition operation, yielding the strong, weak and restricted projection operations.

Feasibility, Inclusion Check and Redundancy Elimination

There exist polynomial time algorithms that are efficient in practice for checking feasibility of a polyhedron and inclusion between two polyhedra.

Feasibility. The SIMPLEX method can be used to check feasibility of a given linear inequality assertion φ. In practice, we solve the optimization problem $\mathsf{minimize}\ 0\ \mathsf{subject\ to}\ \varphi$. An answer of $+\infty$ indicates the infeasibility of φ.

Inclusion Check. As a primitive, consider the problem of checking whether a given inequality $e \geq 0$ is entailed by φ, posing the LP: $\mathsf{minimize}\ e\ \mathsf{subject\ to}\ \varphi$. If the optimal solution is a, it follows from the definition of a LP problem that $\varphi \models e \geq a$. Thus subsumption holds iff $a \geq 0$. In order to decide if $\varphi \models A\boldsymbol{x} + \boldsymbol{b} \geq 0$, we decide if the entailment holds for each half-space $A_i\boldsymbol{x} + b_i \geq 0$.

Redundancy Elimination (Simplification). Each inequality is checked for subsumption by the remaining inequalities using the inclusion check primitive.

Widening

The standard widening of Cousot and Halbwachs may be implemented efficiently using linear programming. Let φ_1, φ_2 be two polyhedra such that $\varphi_1 \models \varphi_2$.

Let us assume that φ_1, φ_2 are both satisfiable. We seek to drop any constraint $e_i \geq 0$ in φ_1 that is not a consequence of φ_2. This can be achieved by the inclusion test primitive described above.

Definition 8 (Standard Widening). *The standard widening of two polyhedra $\varphi_1 \models \varphi_2$, denoted $\varphi = \varphi_1 \nabla \varphi_2$ is computed as follows,*

1. *Check satisfiability of φ_1, φ_2. If either one is unsatisfiable, widening reduces to their join.*
2. *Otherwise, for each $e_i \in \mathsf{Ineqs}(\varphi_1)$, compute $b_i = \mathsf{minimize}\ e_i\ \mathsf{subject\ to}\ \varphi_2$. If $b_i < 0$ then drop the constraint $e_i \geq 0$ from φ_1.*

This operator is identical to the widening defined by Cousot and Halbwachs [9]. The operator may be improved by additionally computing the join of the equalities in both polyhedra. The work of Bagnara et al. [1] presents several approaches to improving the precision of widening operators.

4 Performance

We have implemented many of the ideas in this paper in the form of an abstract domain library written in OCAML. Our library uses GLPK [15] to solve LP queries, and PPL [2] to convert between the constraint and generator representations of polyhedra. Such conversions are used to implement the strong join and post condition. Communication between the different libraries is implemented using Unix pipes. As a result, the communication overhead is significant for small examples.

Choosing Domain Operations. We have provided several options for the join and the post condition operations. In practice, one can envision many strategies for choosing among these operations. Our implementation chooses between the strong and the weak versions based on the sizes of the input polyhedra. Strong post condition and joins are used for smaller polyhedra (40 variables+constraints). On the other hand, the inversion join is used for polyhedra with roughly 100s of variables+constraints, while the weak versions are used for larger polyhedra.

Name (#vars)	#trans	Strong+Weak		Purely Strong		±
		time	mem	time	mem	
REQ-GRANT(11)	8	3.14	5.7	0.1	4.1	+
CSM(13)	8	6.21	5.9	0.1	4.2	≠
C-PJAVA(18)	14	11.2	6.0	0.1	4.1	≠
MULTIPOOL(18)	21	10.0	6.0	2.1	9.2	+
INCDEC(32)	28	39.12	6.8	8.7	10.4	≠
MESH2X2(32)	32	33.8	6.4	18.53	66.2	≠
BIGJAVA(44)	37	46.9	7.2	256.2	55.3	≠
MESH3X2(52)	54	122	8.1	$> 1h+$	$> 800+$	+

Table 1. Performance on Benchmark Examples. All times are in seconds and memory utilization in Mbs.

We observe empirically that the use of strong operations does not improve the result once the widening phase is started. Therefore, we resort to weak join and post condition for the widening phase of the analysis.

4.1 Benchmark Examples

We supplied our library to generate invariants for a few benchmark system models drawn from related projects such as FAST [3] and our previous work [18]. Table 1 shows the complexity of each system in terms of number of variables (#vars) along with the performance of our technique of mixed strong, weak and inversion domain operations as compared with the purely strong join/post operations implemented directly in C++ using the PPL library. We compare the running time and memory utilization of both implementations. Results were measured on an Intel Xeon II processor with 1GB RAM. The last column compares the invariants generated. A "+" indicates that our technique discovers strictly stronger invariants whereas a "≠" denotes that the invariants are incomparable.

Also, for small polyhedra, strong operations frequently outperform weak domain operations in terms of time. However, their memory consumption seems asymptotically exponential. Therefore, weak domain operations yield a drastic performance improvement when the size of the benchmark examples increases beyond the physical memory capacity of the system. Comparing the invariants generated, it is interesting to note that the invariants produced by both techniques are, for the most part, incomparable. While inversion join is weaker than strong join, the non-monotonicity of the widening operation and its dependence on the syntactic representation of the polyhedra cause the two versions to compute different invariants.

Analysis of πVC Programs. We applied our abstract domain library to analyze a subset of the C language called πVC, consisting of imperative programs over integers with function calls. The language features dynamically allocated arrays,

Description	Size		(Weak+Strong)		(Strong)		Proves
	#LOC	#fns	time(sec)	mem(Mb)	time(sec)	mem(Mb)	Property
binary-search (*)	27	2	0.48	7.8	0.4	7.5	$\surd$
insertionsort	37	1	2.9	7.9	2	7.8	$\surd$
heapsort	75	5	26.2	9.8	23.0	9.6	$\surd$
quicksort (*)	106	4	2m	13.2	overflow		$\surd$
Knuth-Morris-Pratt	110	4	9.4	8.6	7.9	8.6	4
Boyer-Moore	106	3	33.7	10.4	28.8	10.8	12
fixwrites(*)	270	10	4.2m	26.5	> 75m	> 75M	28

Table 2. Performance of invariant generator for benchmark programs

records and recursive function calls while excluding pointers. Parameters are passed by value and global variables are disallowed. The language incorporates invariant annotations by the user that are verified by the compiler using a background decision procedure. Our analysis results in sound annotations that aid the verifying compiler in checks for runtime safety such as freedom from overflows, and with optional user supplied assertions, help prove functional correctness.

Our analyzer is inter-procedural, using summaries to handle function calls in a context sensitive manner. Our abstraction process models arrays in terms of their allocated sizes while treating their contents as unknowns. Integer operations such as multiplication, division and modulo are modeled conservatively so that soundness is maintained. The presence of recursive function calls requires that termination be ensured by limiting the number of summary instances per function and by widening on the summary preconditions.

Table 2 shows the performance on implementations of standard sorting algorithms, string search algorithms and a part of the `web2C` code for converting Pascal-style writes into C-style printf functions, originally verified by Dor et al. [11]. The columns in Table 2 show the size of each program in lines of code and number of functions. An asterisk (*) identifies programs containing recursive functions. We place a check mark ($\surd$) in the "proves property" column if the resulting annotations themselves prove all array accesses and additional user provided assertions. Otherwise, the number of unproven accesses/assertions is indicated. Our analyzer proves a vast majority ($\geq 90\%$) of the assertions valid, without any user interaction. Indirect array accesses such as $a[b[i]]$ are a major reason for the false positives. We are looking into more sophisticated abstractions to handle such accesses. The invariants generated by both the versions are similar for small programs, even though weak domain operations were clearly used during the analysis. The difference in performance is clearer as the size of the program increases. Our interface to the PPL library represents coefficients using long integers. This led to an overflow error while analyzing quicksort.

In conclusion, we have designed and implemented efficient domain operations and applied our technique to verify interesting benchmark examples. We hope to extend our analyzer to handle essential features such as pointers and arrays.

Acknowledgments. Many thanks to Mr. Aaron Bradley for implementing the πVC front end, the reviewers for their incisive comments and to the developers of the PPL [2] and GLPK [15] libraries.

References

1. Bagnara, R., Hill, P. M., Ricci, E., and Zaffanella, E. Precise widening operators for convex polyhedra. In *Static Analysis Symposium* (2003), vol. 2694 of *LNCS*, Springer–Verlag, pp. 337–354.
2. Bagnara, R., Ricci, E., Zaffanella, E., and Hill, P. M. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *SAS* (2002), vol. 2477 of *LNCS*, Springer–Verlag, pp. 213–229.
3. Bardin, S., Finkel, A., Leroux, J., and Petrucci, L. FAST: Fast accelereation of symbolic transition systems. In *Computer-aided Verification* (July 2003), vol. 2725 of *LNCS*, Springer–Verlag.
4. Besson, F., Jensen, T., and Talpin, J.-P. Polyhedral analysis of synchronous languages. In *Static Analysis Symposium* (1999), vol. 1694 of *LNCS*, pp. 51–69.
5. Chvátal, V. *Linear Programming.* Freeman, 1983.
6. Clarisó, R., and Cortadella, J. The octahedron abstract domain. In *Static Analysis Symposium* (2004), vol. 3148 of *LNCS*, Springer–Verlag, pp. 312–327.
7. Cousot, P., and Cousot, R. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming* (1976), Dunod, Paris, France, pp. 106–130.
8. Cousot, P., and 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* (1977), pp. 238–252.
9. Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among the variables of a program. In *ACM POPL* (Jan. 1978), pp. 84–97.
10. Dantzig, G. B. *Programming in Linear Structures.* USAF, 1948.
11. Dor, N., Rodeh, M., and Sagiv, M. CSSV: Towards a realistic tool for statically detecting all buffer overflows in C. In *Proc. PLDI'03* (2003), ACM Press.
12. Halbwachs, N., Proy, Y., and Roumanoff, P. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design 11* (1997), 157–185.
13. Jeannet, B. The convex polyhedra library New Polka. Available online from http://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html.
14. Karr, M. Affine relationships among variables of a program. *Acta Inf. 6* (1976), 133–151.
15. Makhorin, A. The GNU Linear Programming Kit (GLPK), 2000. Available online from http://www.gnu.org/software/glpk/glpk.html.
16. Manna, Z. *Mathematical Theory of Computation.* McGraw-Hill, 1974.
17. Miné, A. A new numerical abstract domain based on difference-bound matrices. In *PADO II* (May 2001), vol. 2053 of *LNCS*, Springer–Verlag, pp. 155–172.
18. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Scalable analysis of linear systems using mathematical programming. In *Verification, Model-Checking and Abstract-Interpretation (VMCAI 2005)* (January 2005), vol. 3385 of *LNCS*.
19. Simon, A., King, A., and Howe, J. M. Two variables per linear inequality as an abstract domain. In *LOPSTR* (2003), vol. 2664 of *Lecture Notes in Computer Science*, Springer, pp. 71–89.