Template DBM: A New Weakly Relational Domain for Efficient Memory-Access Validation

Yusen Su[1][0009−0004−8813−0797], Jorge A. Navas[2][0000−0002−0516−1167], and Arie Gurfinkel[1][0000−0002−5964−6792]

[1] Department of Electrical and Computer Engineering, University of Waterloo
[2] Certora Inc.

Abstract. A primary goal of static analysis based on abstract interpretation is to infer invariants to verify programs. Memory safety checks (e.g., proving the absence of out-of-bound accesses) require tracking linear relationships between pointer offsets and object sizes, such as `4*idx + 4 <= sz` for accessing memory. Choosing the right abstract domain is crucial, as each domain captures different kinds of properties. For example, Zones and Octagons limit themselves to unit coefficients and cannot express the invariants required for memory safety. On the other hand, the Polyhedra domain can capture any linear relation but does not scale in real applications. In this paper, as a compromise between expressiveness and efficiency while still covering our target properties, we introduce Template DBM — a new weakly relational numerical domain for expressing Two Variables per Inequality (TVPI) constraints with fixed coefficients. Template DBM supports efficient join, inclusion, and saturation. It strikes the balance of expressiveness and cost between Zones and TVPI. We implemented Template DBM in the Crab library and evaluated it against Zones and Polyhedra domains for memory safety analysis of `aws-c-common` from AWS and `firedancer` from Solana. Our results show that Template DBM maintains its intended level of precision, scales comparably to Zones, and is significantly more efficient than Polyhedra.

Keywords: Static Analysis · Abstract Interpretation · Abstract Domains · Bounds checking · Numerical Domains

1 Introduction

Numerical program analysis powers static analyzers for verification and code optimization across real-world software systems. Although using a precise relational abstract domain ensures accuracy, scalability emerges as a critical challenge when code size grows. For example, Polyhedra [5] domain achieves great precision by encoding arbitrary linear inequalities to capture highly precise program invariants, but its operations incur worst-case time and space complexity exponential in the number of variables, limiting performance. However, when scalability is prioritized, some generality tends to be sacrificed. Existing weakly relational numerical domains [16], such as [20,15,14], make their own sets of restrictions.

```c
1  struct array_list {
2    int size;  // allocated size
3    int len;   // used length
4    int isz;   // item size
5    void *data;
6  };
7
8  int get_at(const struct array_list *l,
          void *val, int idx) {
9    if (l->len > idx) {
10     int ofs = l->isz * idx;
11     void *p = (uint8_t *)l->data + ofs;
12     assert(valid_access(p, l->isz));
13     assert(valid_access(val, l->isz));
14     memcpy(val, p, l->isz);
15     return 0;
16   }
17   return -1;
18 }
```

```c
19 void main(int len, int idx) {
20   if (0 <= idx && idx < len) {
21     struct array_list l;
22     l.len = len;
23     l.isz = 4; // 4 bytes
24     l.size = l.isz * l.len;
25     l.data = malloc(l.size);
26
27     // Assume some code initializes l.data.
28
29     void *item = malloc(sizeof(l.isz));
30     int ret = get_at(&l, item, idx);
31     assert(ret == 0);
32   }
33 }
```

Fig. 1: An example C program.

The TVPI [20] (Two Variables Per Inequality) domain permits any linear inequalities involving up to two variables. It operates in polynomial time, but its inequality set per variable pair can still grow without bound due to arbitrary coefficients. To bound the number of inequalities by the number of variables, one way is to restrict coefficients to unit values, yielding the Unit Two Variables Per Inequality (UTVPI) domains: Zones [14] and Octagons [15] that offer a compromise between precision and cost.

In many cases, program invariants demand more expressiveness than Zones and Octagons provide, but not necessarily the full generality of TVPI. An example in C is shown in Fig. 1. The program takes an `array list` that manages an array with dynamic size but fixed item size (e.g., 4 bytes). Assuming the list is full, copying an array element through function `get_at` requires computing an intermediate pointer `p` with offset `idx * isz` before access. Establishing memory safety (e.g., proving the memory safety check `valid_access` at line 12 stays within buffer `p`) requires that the given domain automatically captures the following invariants [3]:

$$p.\mathit{offset} + 4 \leq p.\mathit{size} \ \wedge \ p.\mathit{offset} = 4 * \mathit{idx} \ \wedge \ p.\mathit{size} = l.\mathit{size} \ \wedge$$
$$0 \leq \mathit{idx} \ \wedge \ \mathit{idx} < l.\mathit{len} \ \wedge \ l.\mathit{size} = 4 * l.\mathit{len} \ \wedge \ l.\mathit{isz} = 4$$

Neither Zones nor Octagons, which merely express Unit Two Variables Per Inequality (UTVPI) constraints, can prove the check.

Although the generality of TVPI and Polyhedra is useful, it does not justify the computational expense. In our experience, proving the memory safety in low-level code requires reasoning about arrays that are traversed using a stride or a

[3] We assume a pointer value with extra information [24]: `offset` is the position of the referred object, `size` is the object size. For brevity, we write *l.len* to mean `l->len` for field access.

step. Often the size of the stride is the size of a memory word (4 or 8 bytes), or the size of a specific structure stored in the array (i.e., a specific `item_size`). In most cases, specializing the domain to deal with a few fixed coefficients, that are heuristically identified from program source code, is sufficient. Restricting the space of coefficients opens a new opportunity for designing an efficient domain that matches Zones in scalability while adding the desired precision.

We aim to extend Zones to handle TVPI constraints while leaving the complexity of the representation and operations intact. Specifically, we introduce Template DBM, a new numerical abstract domain to express Two Variables Per Inequality (TVPI) constraints with fixed coefficients. Each domain element retains the form of constraints $ax - by \leq c$, where x and y are program variables, a, b are fixed integer coefficients, and c is a constant. For example, a property constraint like `p.offset` $= 4 *$ `idx`, originally expressed by two TVPI inequalities, is encoded in UTVPI form as $\pm(p.\mathit{offset} - 4 \cdot \mathit{idx}) \leq 0$, where $4 \cdot \mathit{idx}$ is treated as a ghost variable [4] $4\mathit{idx}$. Therefore, TVPI constraints in UTVPI form, as used by Zones, can capture complex properties like array indexing. We fix the coefficients for representation as a template, so the cost of each operation is inherently bound by the number of variables and the size of the template.

In summary, Template DBM is more precise than Zones but less general than TVPI, since it only supports a subset of TVPI constraints. To validate our contributions, we built Template DBM in Crab [11] and evaluated it in terms of scalability and precision. The evaluation results show that the performance of Template DBM is comparable to that of Zones and the precision measured by the number of memory safety checks validated is close to that of Polyhedra.

The paper is organized as follows. Section 2 covers necessary definitions and operations of the Zones domain. Section 3 introduces a strategy for encoding TVPI constraints in a *difference bound matrix* and provides algorithms for saturation. Section 4 describes the core operations of Template DBM. Section 5 presents the implementation and experimental evaluation. Finally, Section 6 discusses related work.

2 Background

In this section, we discuss the necessary preliminaries for the Zones [14] domain, including key definitions and important operations used in this paper.

Given a set of program variables $\mathcal{V} = \{x, y, z, \ldots\}$ with N variables and a set of integer numbers $\mathbb{Z}$ extended by infinity $+\infty$, Zones supports representing a UTVPI constraint system $\mathcal{U}$ over $\mathcal{V}$ of the form: $\{x - y \leq c \mid x, y \in \mathcal{V} \wedge c \in \mathbb{Z} \cup \{+\infty\}\}$. The common data structure to encode UTVPI constraints is a Difference Bound Matrix (DBM), where rows and columns correspond to variables, and each entry represents an UTVPI constraint. For example, the constraint $x - y \leq 3$ is represented in a matrix m with $m_{xy} = 3$[4]. Typically, DBM adds an auxiliary variable v^0 that always takes the value 0 for $\pm x \leq c$. It

[4] Our DBM notation differs slightly from that of [14]: we use $x - y \leq \bar{m}_{x,y}$ for convenience.

follows that the matrix dimensions are $(N+1)\times(N+1)$. For brevity, we suppose the variables are integers[5] and ignore describing constraints with infinite value in the rest of the paper.

We assume readers know the abstract domain operations used for building analysis. A DBM supports join $\sqcup^{DBM}$, meet $\sqcap^{DBM}$, and widening $\triangledown^{DBM}$. It also provides a DbmClosure (saturation) operation to derive all implicit constraints, ensuring that the DBM is *closed*. The idea is to propagate inequalities through a transitive chain: given $x - y \leq a$ and $y - z \leq b$, derive $x - z \leq a + b$. The (full) closure operation runs in cubic time $O(N^3)$. For efficiency, we can derive implicit constraints *incrementally* when a closed DBM adds a few new constraints. IncrementalDbmClosure does this by updating the affected entries in quadratic time $O(N^2)$. The full details of the core DBM operators can be found in [14], and the incremental closure algorithm for Zones is summarized in [2]. Additionally, we assume that the DBM also includes a transfer function such as $m \cup \{x - y \leq c\}$ to add or tighten a set of constraints only.

3 Template DBMs

In this section, we present a new DBM, tDBM (for *template* DBM), that extends the classical DBM for representing TVPI constraints. For any TVPI constraint $ax - by \leq c$ with non-unit coefficients (i.e., $max\{a, b\} > 1$), tDBM introduces extra dimensions for variables with non-unit factors and storing c at the ax, by entry. For example, tDBM associates a dimension for $4idx$ to capture a constraint like $4 * idx - y \leq 0$. These extra dimensions depend on a predefined coefficient template $\mathcal{T}$. We assume $\mathcal{T} = \{a, b, \ldots\}$ with size $|\mathcal{T}| = K$ and always include 1 in $\mathcal{T}$. As usual, we use v^0 with value 0 for $\pm ax \leq c$.

tDBM falls between the system $\mathcal{U}$ and a TVPI constraint system $\mathcal{I} \overset{\text{def}}{=} \{ax - by \leq c \mid x, y \in \mathcal{V} \wedge a, b \in \mathbb{Z}^{\geq 0} \wedge c \in \mathbb{Z}\}$ where a, b are positive coefficients. Specifically, a tDBM represents a constraint system $\mathcal{I}_{\mathcal{T}} \subset \mathcal{I}$ over variables $\mathcal{V}$ and coefficients $\mathcal{T}$ of the form $\{ax - by \leq c \mid x, y \in \mathcal{V} \wedge a, b \in \mathcal{T} \wedge c \in \mathbb{Z}\}$.

Fig. 2 shows how a tDBM represents the TVPI constraints necessary to guarantee that the `memcpy` (i.e., the assertion on line 12) access buffer `p` remains within bounds. The given tDBM extends two extra dimensions for expressing relations for $4idx$ and $4l.len$. To improve readability, we decompose the tDBM $\bar{m} := (m, m^+)$ into two submatrices: a classical sub-DBM, m, that covers dimensions for UTVPI constraints $x - y \leq c$, and an extended sub-DBM, m^+, that handles TVPI constraints with fixed non-unit coefficients $ax - by \leq c$.

The constraint $p.offset - p.size \leq -4$ (denoted as c) is needed to prove the assertion on line 12 is valid. However, this constraint (and all constraints colored in purple) is implicit and can only be inferred from the constraints colored in green. Resolving c requires inferring $c2 : p.offset - l.size \leq -4$, where $c2$ also needs another constraint $c3 : p.offset - 4l.len \leq -4$ to be implied. Section 3.1 shows how to compute those implicit constraints.

[5] DBMs can also be defined over the rationals; however, in this paper, we focus on the integer case exclusively.

$m:$

$$\begin{array}{rr} -idx \leq 0 & -l.len \leq -1 \\ 4 \leq l.isz \leq 4 & idx - l.len \leq -1 \\ p.size - l.size \leq 0 & l.size - p.size \leq 0 \\ p.offset - p.size \leq -4 & p.offset - l.size \leq -4 \end{array}$$

$m^{+}:$

$$\begin{array}{ll} p.offset - 4idx \leq 0 & 4idx - p.offset \leq 0 \\ 4l.len - l.size \leq 0 & p.offset - 4l.len \leq -4 \\ l.size - 4l.len \leq 0 & \end{array}$$

(a)

$m:$

	v^0	$l.isz$	$l.len$	$l.size$	idx	$p.offset$	$p.size$
v^0	$+\infty$	-4	-1	$+\infty$	0	$+\infty$	$+\infty$
$l.isz$	4	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$l.len$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$l.size$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	0
idx	$+\infty$	$+\infty$	-1	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$p.offset$	$+\infty$	$+\infty$	$+\infty$	-4	$+\infty$	$+\infty$	-4
$p.size$	$+\infty$	$+\infty$	$+\infty$	0	$+\infty$	$+\infty$	$+\infty$

$m^{+}:$

	$p.offset$	$4idx$	$4l.len$	$l.size$
$p.offset$	$+\infty$	0	-4	$+\infty$
$4idx$	0	$+\infty$	$+\infty$	$+\infty$
$4l.len$	$+\infty$	$+\infty$	$+\infty$	0
$l.size$	$+\infty$	$+\infty$	0	$+\infty$

(b)

Fig. 2: (a) DBM-related constraints and (b) a tDBM $\bar{m}$.

For the rest of the paper, we use $row(ax)$ and $col(ax)$ to refer to the row and column indexes, respectively, in the $\bar{m}$ associated with variable x and coefficient a. To access the item in $\bar{m}$, we use $\bar{m}_{row(ax)col(by)}$ or simply $\bar{m}_{ax,by}$ to refer to the difference bounds for inequality $ax - by \leq \bar{m}_{ax,by}$. For elements represented for UTVPI constraints, we write $\bar{m}_{x,y}$. For elements of other TVPI constraints, we write $\bar{m}_{ax,by}$, $\bar{m}_{ax,y}$ or $\bar{m}_{x,by}$. To access the difference bound for any TVPI constraint l, we denote it as $\bar{m}_l$.

A single TVPI constraint can be represented in a variety of equivalent ways. For example, $2x - y \leq 3$ is equivalent to $4x - 2y \leq 6$, $6x - 3y \leq 9$, etc. To keep as many constraints as possible in tDBM, we normalize each TVPI constraint $ax - by \leq c$ into a *template expressible form* so that its coefficients fit the coefficient template $\mathcal{T}$ (if possible). This is justified by the following inference:

$$\frac{\begin{array}{c} \mathcal{I} \vdash ax - by \leq c \quad d \in \mathbb{Z}^{>1} \quad (d \mid a) \quad (d \mid b) \\ a' = a/d \quad a' \in \mathcal{T} \quad b' = b/d \quad b' \in \mathcal{T} \quad c' = \lceil c/d \rceil \quad c' \in \mathbb{Z} \end{array}}{\mathcal{I}_{\mathcal{T}} \vdash a' \cdot x - b' \cdot y \leq c'} \ \textsc{Scaling}$$

The rule scales the coefficients by a common divisor d to produce an equivalent inequality, where the new coefficients a' and b' are elements of $\mathcal{T}$. For example, for $\mathcal{T} = \{1, 2, 3, 4\}$, the rule scales the constraint $8x - 4y \leq 8$ to $2x - y \leq 2$ (divided by 4). In practice, we keep just one equivalent form, as others like $4x - 2y \leq 4$ are ignored. The purpose of this rule is to convert TVPI constraints into the expressible forms before matrix updates.

Algorithm 1 Nelson-driven [18] tDBM saturation.

```
1: function NelsonTvpiSaturation($\bar{m}$)
2:     for $i \in \{0, \ldots, \lceil lg(N) \rceil - 1\}$ do
3:         for $x, y, z \in \mathcal{V}$ do
4:             for $a, b, d, e \in \mathcal{T}$ do
5:                 $c := \bar{m}_{ax,by}, f := \bar{m}_{dy,ez}$
6:                 if $a'x - b'z \le c' = $ ScaledResultant$(ax - by \le c, dy - ez \le f)$
                   then
7:                     $\bar{m} := \bar{m} \cup \{a'x - b'z \le c'\}$
```

3.1 Saturation

To achieve a full (closed) representation, we *saturate* tDBM by exhaustively deriving all implicit constraints until a fixpoint is reached. We present a saturation algorithm for tDBM based on Fourier-Motzkin variable elimination. Each implicit inequality is deduced following this rule:

$$\frac{\mathcal{I} \vdash ax - by \le c \quad \mathcal{I} \vdash dy - ez \le f \quad g = gcd(b,d)\ \lambda_1 = d/g\ \lambda_2 = b/g}{\mathcal{I} \vdash (\lambda_1 a) \cdot x - (\lambda_2 e) \cdot z \le \lambda_1 c + \lambda_2 f}\ \textsc{Resultant}$$

which eliminates variable y and yields a new inequality. For instance, consider the two TVPI constraints $2x - 3y \le 5$ and $9y - 2z \le 5$. Eliminating y through the rule yields $6x - 2z \le 20$. If this constraint cannot fit the coefficient template (e.g., $\mathcal{T} = \{1, 2, 3, 9\}$), we apply Scaling rule to rewrite it as $3x - z \le 10$.

Algorithm 1 saturates an input tDBM by iteratively applying Resultant to pairs of existing inequalities until the iteration limit $\lceil lg(N) \rceil - 1$ is reached. We cap this bound since by then applying the Resultant guarantees a contradiction witness [18]. The Resultant is extended as ScaledResultant which applies Scaling after to produce an expressible form. The time complexity is $O(K^4 N^3 lg(N))$ when a matrix is dense.

Since the tDBM does not express arbitrary TVPI constraints, the algorithm only introduces implicit inequalities within the available dimensions or tightens the existing ones. A tDBM $\bar{m}$ as *complete* if and only if, for every constraint c over a variable set U that $\bar{m}$ satisfies, the projection of $\bar{m}$ (i.e., restricting constraints) to U still satisfies c.

During the saturation process, however, Algorithm 1 cannot guarantee the result tDBM is complete, since the derived constraints may not be expressible in the tDBM. For example, for $\mathcal{T} = \{1, 2, 3, 4\}$, a tDBM $\bar{m}$ represents $\{2x - 3y \le 6 \ \wedge\ y - 4z \le 8 \ \wedge\ 3z - 2w \le 7\}$. Despite the coefficient template, applying the Resultant rule iteratively derives a new set of constraints $\{x - 6z \le 15, 3y - 8w \le 52, x - 4w \le 29\}$ at the fixpoint. However, to derive $x - 4w \le 29$, tDBM requires representing either $x - 6z \le 15$ or $3y - 8w \le 52$. Thus, $\bar{m}$ is not complete.

Choosing the coefficient template $\mathcal{T}$ is crucial for completeness. For instance, if the template contains coefficients that are powers of two, running the Algorithm 1 guarantees that all implicit constraints are derived and representable.

Theorem 1. *Given a tDBM $\bar{m}$, Algorithm 1 computes a complete tDBM $\bar{m}'$ under the TVPI system $\mathcal{I}_{\mathcal{T}}$ if and only if all implicit constraints are expressible.*

Proof. Since all implicit constraints are expressible, Algorithm 1 strictly follows Lemma 1 in [18]. □

While Algorithm 1 does not promise full completeness, it is sufficient for our purposes. Consider the example shown in Fig. 2, to derive the marked constraint $p.\mathit{offset} - p.\mathit{size} \leq -4$, below are the constraints derived at each iteration:

1. $p.\mathit{offset} - 4\mathit{idx} \leq 0$ and $\mathit{idx} - l.\mathit{len} \leq -1$ derives $p.\mathit{offset} - 4l.\mathit{len} \leq -4$.
2. $p.\mathit{offset} - 4l.\mathit{len} \leq -4$ and $4l.\mathit{len} - l.\mathit{size} \leq 0$ produces $p.\mathit{offset} - l.\mathit{size} \leq -4$.
3. $p.\mathit{offset} - l.\mathit{size} \leq -4$ and $l.\mathit{size} - p.\mathit{size} \leq 0$ gives $p.\mathit{offset} - p.\mathit{size} \leq -4$.

The Resultant rule specializes to the standard DBM closure whenever $b = d$, and this closure is not limited to UTVPI constraints. For example, any pair of constraints such as $2x - 3z \leq 4$ and $3z - y \leq -1$ can also use the DBM closure to derive $2x - y \leq 3$. This gives us the opportunity to reuse that routine and to build a tDBM saturation on top of it. Overall, we iteratively apply Resultant by decomposing it into two steps:

1. $(b \neq d)\ ax - by \leq c \wedge dy - ez \leq f \xrightarrow{\textsc{ScaledResultant}} a'x - b'z \leq c'$
2. $(b = d)\ ax - by \leq c \wedge dy - ez \leq f \xrightarrow{\textsc{DbmClosure}} ax - ez \leq c + f$

Step 1 aligns TVPI constraint coefficients to eliminate y. Step 2 reuses the standard DBM closure. Accordingly, Algorithm 2 shows a tDBM version in which TvpiReduce handles the first step and DbmClosure operates the second. By Theorem 2, algorithm is equivalent to Algorithm 1.

Theorem 2. *Given a tDBM $\bar{m}$,*

$$\textsc{NelsonTvpiSaturation}(\bar{m}) \equiv \textsc{DbmTvpiSaturation}(\bar{m})$$

Proof. For any input tDBM $\bar{m}$, Algorithm 2 repeatedly invokes TvpiReduce (with coefficient alignment) and DbmClosure (no alignment). This is identical to applying the Resultant rule in Algorithm 1. Even the new derived constraint can be used directly during iteration i, with the guarantee of the loop upper bound $\lceil lg(N) \rceil - 1$, ensuring that all implicit constraints are ultimately captured in the matrix, regardless of the order of steps (earlier or later). □

As the above example shows, the first implicit constraint $p.\mathit{offset} - 4l.\mathit{len} \leq -4$ is derived from TvpiReduce, and the other two from DbmClosure.

Our purpose is extending DBM to support TVPI constraints using a small coefficient template. Experiments in Section 5 show that using three active coefficients suffices, and TVPI constraints with non-unit coefficients remain few compared with UTVPI constraints. Thus, running TvpiReduce is cheap, and DbmClosure runs nearly as efficiently when only UTVPI constraints are present.

Algorithm 2 A DBM closure based saturation for tDBM.

```
function TvpiReduce($\bar{m}$)
    for $x, y, z \in \mathcal{V}$ do
        for $a, b, d, e \in \mathcal{T} \wedge b \neq d$ do
            $c := \bar{m}_{ax,by}, f := \bar{m}_{dy,ez}$
            if $a'x - b'z \leq c' =$ ScaledResultant$(ax - by \leq c, dy - ez \leq f)$ then
                $\bar{m} := \bar{m} \cup \{a'x - b'z \leq c'\}$

function DbmTvpiSaturation($\bar{m}$)
    for $i \in \{0, \ldots, \lceil lg(N) \rceil - 1\}$ do
        TvpiReduce($\bar{m}$)
        DbmClosure($\bar{m}$)
```

3.2 Incremental Saturation

Existing abstract domains for DBM apply an incremental closure [10,17,2,3] to restore DBM in closed form after each assignment or constraint strengthening, making this procedure the dominant use. By updating only the affected entries in the DBM, the algorithm runs more efficiently than the full saturation (closure) algorithm. In this section, we present a worklist-based procedure that incrementally applies the Resultant rule to propagate new constraints.

Algorithm 3 gives the pseudocode for incremental saturation. It adds the new constraint $ax - by \leq c$ into $\bar{m}$. Once the input constraint is tighter, the algorithm first collects all existing constraints involving y (in positive occurrence) and applies the Resultant rule to eliminate it, as shown in the purple box, then repeats for x (orange box). Each elimination step produces new constraints that contain only one of the two variables, x or y; we store them in the worklists W_x and W_y, respectively. We present a graph representation (shown in Fig. 3a) that illustrates and highlights the edges added to each list in their corresponding colors. Next, we reuse those derived constraints to completely eliminate either y or x (green box). As a result, none of the new constraints include y and x. Specifically, we derive constraints between z and w through the transitive chain $\{z, x\}, \{x, y\}, \{y, w\}$ (see the green edge in Fig. 3a).

To illustrate how Algorithm 3 works, given a set of constraints with no implicit ones, represented by a tDBM over the coefficient set $\mathcal{T} = \{1, 4\}$:

$$\begin{array}{ccc} idx - len \leq -1 & size - 4len \leq 0 & 4len - size \leq 0 \\ 4idx - 4len \leq -4 & 4idx - size \leq -4 & \end{array}$$

Suppose we add $ofs - 4idx \leq 0$ and perform incremental saturation. The algorithm finds all inequalities involving idx and ofs: $idx - len \leq -1$, $4idx - 4len \leq -4$, and $4idx - size \leq -4$. Applying the Resultant rule to eliminate idx yields $ofs - 4len <= -4$ and $ofs - size \leq -4$. With no existing constraints involving ofs, no further constraints can be derived, and the tDBM is once again closed.

Lemma 1. *Suppose a tDBM $\bar{m}$ is closed under system $\mathcal{I}_{\mathcal{T}}$, adding a new constraint $l = ax - by \leq c$ by Algorithm 3 yields $\bar{m}'$ which is satisfiable if and only if:*

Algorithm 3 Incremental saturation for tDBM.

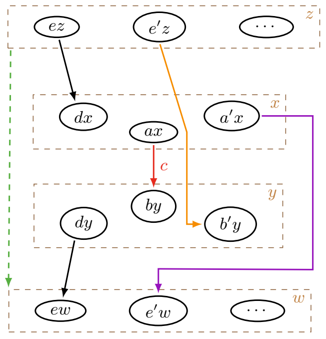
```
1:  function TvpiIncrementalSaturation($\bar{m}, ax - by \le c$)
2:      $\bar{m} := \bar{m} \cup \{ax - by \le c\}$
3:      $W_x := \{\}, W_y := \{\}$
4:      for $d, e, w \in \mathcal{T} \times \mathcal{T} \times \mathcal{V}$ do                ▷ successors related to $y$
5:          if $a'x - e'w \le c' = $ ScaledResultant($ax - by \le c, dy - ew \le \bar{m}_{dy,ew}$)
            then
6:              $\bar{m} := \bar{m} \cup \{a'x - e'w \le c'\}$
7:              $W_x := W_x \cup \{a'x - e'w \le c'\}$
8:      for $d, e, z \in \mathcal{T} \times \mathcal{T} \times \mathcal{V}$ do                ▷ predecessors related to $x$
9:          if $e'z - b'y \le c' = $ ScaledResultant($ez - dx \le \bar{m}_{ez,dx}, ax - by \le c$) then
10:             $\bar{m} := \bar{m} \cup \{e'z - b'y \le c'\}$
11:             $W_y := W_y \cup \{e'z - b'y \le c'\}$
12:     for $ez - by \le c \in W_y$ do
13:         for $d, g, w \in \mathcal{T} \times \mathcal{T} \times \mathcal{V}$ do             ▷ successors related to $y$
14:             if $e'z - g'w \le c' = $ ScaledResultant($ez - by \le c, dy - gw \le \bar{m}_{dy,gw}$)
                then
15:                 $\bar{m} := \bar{m} \cup \{e'z - g'w \le c'\}$
16:     for $ax - ew \le c \in W_x$ do
17:         for $d, g, z \in \mathcal{T} \times \mathcal{T} \times \mathcal{V}$ do             ▷ predecessors related to $x$
18:             if $g'z - e'w \le c' = $ ScaledResultant($gz - dx \le \bar{m}_{gz,dx}, ax - ew \le c$)
                then
19:                 $\bar{m} := \bar{m} \cup \{g'z - e'w \le c'\}$
```

1. $\bar{m}'_{ax,by} \le c$
2. $\exists o \in \bar{m}, dx - ew \le f :=$ ScaledResultant$(l, o), \bar{m}'_{dx,ew} \le f$
3. $\exists o \in \bar{m}, dz - ey \le f :=$ ScaledResultant$(o, l), \bar{m}'_{dz,ey} \le f$
4. *for any constraint* $dz - ew \le f$ *derived through the transitivity chain:* $\{z, x\}, l, \{y, w\}$, $\bar{m}'_{dz,ew} \le f$

Theorem 3. *Given a tDBM $\bar{m}$ and a new constraint $ax - by \le c$, Algorithm 3 computes a complete tDBM $\bar{m}'$ under the TVPI system $\mathcal{I}_\mathcal{T}$ if and only if all implicit constraints are expressible.*

Proof. Since all implicit constraints are expressible, Algorithm 3 strictly follows Lemma 1. □

The time complexity is $O(K^4N^2)$. The correctness of Algorithm 3 is guaranteed by the Lemma 1. Due to the limitation of the coefficient template, running Algorithm 3 guarantees completeness if all implicit constraints are derived and representable in the tDBM. After incremental saturation, contradictions can be detected by identifying negative cycles (Section 4).



(a)

$$\begin{aligned}
\textbf{Bottom: } & \bar{m} = \bot^{tDBM} && \Leftrightarrow \;\; \bar{m} = \bot^{DBM} \\
\textbf{Top: } & \bar{m} = \top^{tDBM} && \Leftrightarrow \;\; \bar{m} = \top^{DBM} \\
\textbf{Join: } & \sqcup^{tDBM} && \stackrel{\text{def}}{=} \bar{m}_1 \sqcup^{DBM} \bar{m}_2 \\
\textbf{Meet: } & \sqcap^{tDBM} && \stackrel{\text{def}}{=} \bar{m}_1 \sqcap^{DBM} \bar{m}_2 \\
\textbf{Order: } & \sqsubseteq^{tDBM} && \stackrel{\text{def}}{=} \bar{m}_1 \sqsubseteq^{DBM} \bar{m}_2 \\
\textbf{Widening: } & \triangledown^{tDBM} && \stackrel{\text{def}}{=} \bar{m}_1 \triangledown^{DBM} \bar{m}_2
\end{aligned}$$

(b)

Fig. 3: (a) Graph view of the tDBM update after adding $ax - by \le c$ with new edges highlighted and the green dashed edge marking all implicit constraints between z and w; (b) The lattice operations of Template DBM.

4 Template DBM Abstract Domain

Template DBM is a new weakly relational domain focused on inferring and checking program properties expressible as TVPI constraints. It is based on tDBM, where each constraint $ax - by \le c$ corresponds to a specific entry in the matrix.

The concretization function γ maps a tDBM $\bar{m}$ to a set of all possible values assigned for variables that satisfy all potential constraints in $\bar{m}$. Formally:

$$\gamma(\bar{m}) \triangleq \{(s_1, \ldots, s_n) \in \mathbb{Z}^n \mid \forall i, j \in [1..n], a, b \in \mathcal{T}, a \cdot s_i - b \cdot s_j \le \bar{m}_{av_i bv_j}\}$$

s_i (s_j) represents a value for a variable v_i (v_j).

A tDBM $\bar{m}$ is unsatisfiable (empty) when repeatedly applying the Resultant rule yields a contradiction. For example, a tDBM $\bar{m}$ with coefficient template $\mathcal{T} = \{1, 2, 3, 4, 5\}$:

$$2y - x \le 3 \qquad 5x - 3z \le -22 \qquad 2z - 5p \le -2 \qquad 3p - 4y \le 0$$

It is unsatisfiable because an implicit inequality $3p - 2x \le 6$ contradicts another implicit one, $2x - 3p \le -10$, following:

1. applying Resultant rule for $3p - 4y \le 0$ and $2y - x \le 3$ derives $3p - 2x \le 6$.
2. from $5x - 3z \le -22$ and $2z - 5p \le -2$, Resultant derives $2x - 3p \le -10$.

As Nelson [18] showed that iterating the Resultant rule through saturation Algorithm 2 exposes a contradiction. Thus, the unsatisfiable check boils down to inspecting a saturated tDBM $\bar{m}$: $\exists\, x, y \in \mathcal{V},\; a, b \in \mathcal{T} :\; \bar{m}_{ax,by} + \bar{m}_{by,ax} < 0$.

A tDBM is bottom (resp. top) if and only if its DBM is bottom (resp. top). For other domain operations, we leverage DBM operations for efficiency. All are defined in Fig. 3b. Each operation performs element-wise matrix updates and runs in quadratic time $O(K^2N^2)$ at worst. All domain operations are safe approximations, but not the best (except for meet). For example, $\sqcup^{tDBM}$ combines two tDBMs by taking the element-wise maximum of their entries. However, a more precise approximation is obtained by finding the extreme points of the convex hull and reconstructing the TVPI constraints accordingly. Consider a join of two abstract states s_1 and s_2:

$$s1 : -i \leq 0 \wedge i \leq 9 \wedge -c \leq 10 \wedge c \leq -1 \qquad s2 : i = 10 \wedge c = 0$$

The convex hull join computes the result state as $-i \leq 0 \wedge i \leq 10 \wedge -c \leq 10 \wedge c \leq 0 \wedge 10c - i \leq -10 \wedge 10i - c \leq 100$. In contrast, $s1 \sqcup^{tDBM} s2$ as $-i \leq 0 \wedge i \leq 10 \wedge -c \leq 10 \wedge c \leq 0$. While join can be designed using the convex hull algorithm, our design is simpler and takes operations from DBM directly.

The primitive operations during analysis are the addition or removal of variables. For an assignment $x := e$, we define the transfer function (see Algorithm 4) that handles two cases. When e is not a linear expression, we over-approximate its value by its interval $[-e^-, e^+]$; otherwise, we approximate it more precisely by using interval information to iteratively drop variables on e until the remaining constraint follows the TVPI form. We first approximate assignment in UTVPI form, since our tDBM natively represents them. Next, we attempt to convert the assignment to TVPI form. If e involves any unbounded variable, we conservatively approximate its value as $\top$. Finally, we insert each new constraint with incremental saturation to maintain closure. Although more precise approximations exist, this simple approach is effective for analyzing programs such as Fig. 1. As an example, let us consider the assignment `ofs = l->isz * idx` at line 10 with a pre-abstract state:

$$l.isz \leq 4 \ \wedge \ -l.isz \leq -4 \ \wedge \ -idx \leq 0 \ \wedge \ \cdots$$

Although the expression `l->isz * idx` is non-linear, we know from the pre-state that `l->isz` is $l.isz \leq 4 \wedge -l.isz \leq -4$. It is safe to rewrite that expression as `4 * idx` and then invoke the Algorithm 4. The assignment thereby is abstracting as two TVPI constraints, $\pm(ofs - 4idx) \leq 0$, and one UTVPI constraint, $idx - ofs \leq 0$, since idx has a known lower bound. After incremental saturation completed, the resulting state remains closed.

For variable removal, we denote $\exists x.\bar{m}$ for eliminating all constraints on x from the tDBM $\bar{m}$. In practice, this simply means dropping all rows and columns for x (including ghost variables). To preserve precision, saturating $\bar{m}$ is required before existential quantification.

5 Implementation and Experimental Evaluation

We have implemented Template DBM[6] in the Crab library [11]. We reuse the implementation from [10] as the underlying DBM which is tailored to make use of a direct graph $\bar{m} := \langle V, E \rangle$. Nodes V correspond to the combination of variables and coefficients $\mathcal{V} \times \mathcal{T}$ and each constraint $ax - by \leq c$ is represented as

[6] Available at `https://github.com/LinerSu/crab/tree/tvpi_dbm`

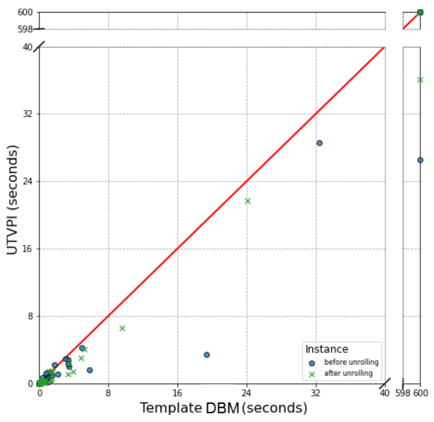
Algorithm 4 Transfer function for assignment.

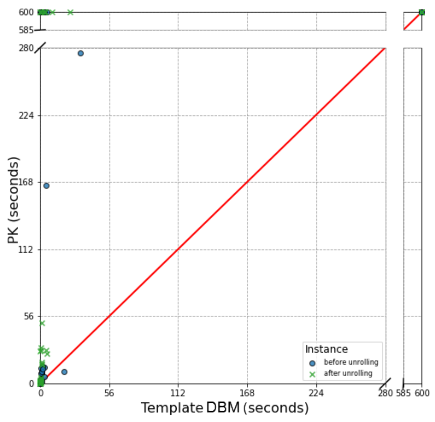
```
1: function TvpiAssign($\bar{m}$, $[\![x := e]\!]$)
2:     $T := \{\}$
3:     if $e = a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n + c \wedge \forall i \in [1..n] : a_i \in \mathbb{Z}^{\geq 0}$ then
4:         for $i \in [1..n]$ do
5:             $e_{1i} := \sum_{j \neq i} a_j \cdot x_j + (a_i - 1) \cdot x_i + c$          ▷ dropping $x_i$
6:             $T := T \cup \{y - x_i \leq e^{+}_{1i}; x_i - y \leq e^{-}_{1i}\}$
7:             if $a_i \in \mathcal{T}$ then
8:                 $e_{ai} := \sum_{j \neq i} a_j \cdot x_j + c$          ▷ dropping $a_i \cdot x_i$
9:                 $T := T \cup \{y - a_i x_i \leq e^{+}_{ai}; a_i x_i - y \leq e^{-}_{ai}\}$
10:    else
11:        $T := T \cup \{x \leq e^{+}, -x \leq -e^{-}\}$
12:    for $t \in T$ do
13:        $\bar{m}$ := TvpiIncrementalSaturation($\bar{m}, t$)
```

a directed edge $ax \xrightarrow{c} by$. The graph representation avoids the $O(K^2N^2)$ space of a matrix, since inferred constraints are often quite sparse during analysis [9], especially after widening in loop-invariant computation [21]. Besides, the graph representation can efficiently perform the domain operations. For example, the join can be implemented by merging two graphs and taking the maximum weight for each edge. The inclusion check $\bar{m}_1 \sqsubseteq^{tDBM} \bar{m}_2$ can also be done by checking if all edges in $\bar{m}_2$ can be entailed by edges in $\bar{m}_1$, which can be done in linear time w.r.t the number of edges (inequalities) $|E|$. The implementation for the remaining domain operations follows algorithms discussed in Section 4.

For efficiency, we implement incremental saturation instead of full saturation (Algorithm 2) for analysis. It performs local updates on each new assignment or assumption for low amortized cost. Our implementation of Algorithm 3 is based on DBM incremental closure. This special version splits into two phases: first, it runs Algorithm 3 where the Resultant rule only applies to cases requiring coefficient alignment; then it invokes IncrementalDbmClosure with previously derived constraints to finish saturation. While this version misses constraints since implied constraints from phase two can be used at once, later experiments show that this version preserves good precision and performance.

In the paper, we demonstrate how Template DBM infers constraints to check for buffer overflows in Fig. 1. This example is from a case study in [23], where an abstract interpreter preprocesses the program to prove and remove memory safety checks before a bounded model checker completes verification. These checks as assertions guard each memory access, and the interpreter proves them both before and after loop unrolling. We reuse this study to evaluate Template DBM performance and precision in proving memory safety, while the details of how the interpreter works are outside the scope here. We also omit discussion on interpreter effectiveness, as it has already been discussed in [23].





(a) (b)

Fig. 4: (a) Zones vs. Template DBM and (b) Polyhedra (PK) vs. Template DBM.

The benchmark suites originate from two open-source production codebases: `aws-c-sdk` and `firedancer`, with a total of 139 benchmarks[7]. Note that benchmarks drawn from `aws-c-sdk` include the `aws-c-common` component. These suites cover programs from simple arithmetic computations to complex data structure manipulations. The benchmarks also include loops to evaluate domain operation performance and precision in proving assertions before and after loop unrolling. The experiment here measures how the interpreter using different numerical domains performs as the program size varies.

We compare Template DBM with the Zones (UTVPI) and the Polyhedra (linear inequality) domains. Because Template DBM and Zones use the same DBM implementation, Zones serves as the baseline for precision and performance comparison. We choose Polyhedra instead of the original TVPI [20] because our experimental results (as shown later) indicate that Template DBM achieves nearly the same precision as Polyhedra on most benchmarks. Besides, the TVPI implementation[8] is unmaintained. Having a direct side-by-side evaluation is challenging. For Polyhedra, we use the implementation from the ELINA library [22].

For consistency, we choose the same configurations for running all domains. Template DBM employs a predefined coefficient template $\{1, 2, 3, 4, 5, 8, 10, 16, 24, 32, 40\}$ with 11 heuristic numbers. Each task is given a timeout of 600 seconds. All experimental results are collected from a machine with an Intel Xeon E5-2680 @2.50GHz, with 256 GB RAM. The artifact and results are available at `https://doi.org/10.5281/zenodo.16075045`.

[7] Available at `https://github.com/LinerSu/TVPI-Domain-Benchmarks`
[8] The original implementation is at `https://github.com/axel-simon/tvpi`.

suite	category	Before loop unroll				After loop unroll			
		Total	Zones	tDBM	PK	Total	Zones	tDBM	PK
`aws-c-sdk`	array_list	24	75%	83%	83%	62	61%	65%	65%
	hash_table	498	83%	85%	85%	2171	54%	58%	58%
	others	1689	67%	67%	67%	2853	56%	59%	59%
`firedancer`	tango	33	36%	85%	100%	151	17%	85%	100%
	util	106	62%	62%	62%	195	75%	75%	87%
	others	270	24%	24%	11%	305	95%	95%	86%
	total	2620	65%	66%	65%	5737	57%	62%	62%

Table 1: Precision across Zones, Template DBM (tDBM), and Polyhedra (PK).

All performance results for analyzing each program before and after loop unrolling are shown in Fig. 4. Zones timed out on 5 before unrolling, and on 2 after. Template DBM has 1 more pre-unrolling timeout case than Zones. Polyhedra timed out 8 more times than Zones before unrolling and 1 more after. As shown in Fig. 4a, Template DBM runs slower but remains within a similar range to Zones after excluding timeouts. Before loop unrolling, Zones averages 0.2s (SD = 0.6) and Template DBM 0.4s (SD = 1.9); after loop unrolling, Zones takes 0.1s (SD = 0.5) and Template DBM 0.2s (SD = 0.7). We achieve similar running times because it extends dimensions to support TVPI constraints. Since not all variables require extra dimensions, the size of Template DBM remains comparable to Zones in most cases. However, as the graph shows one additional timeout and one major slowdown (i.e., the 19 seconds to complete), we diagnosed these and conclude that heavily dimensioned matrices harm operation speed. This limitation can be addressed in future work by implementing a more efficient join algorithm and involving a geometric approach to remove redundant constraints, such as the $filter$ operation introduced from the original TVPI work [20]. In Fig. 4b, Polyhedra does not scale well, with 2.5s (SD = 15.0) average analysis time before unrolling and 2.2s (SD = 7.4) after unrolling. Overall, Template DBM has performance comparable to Zones and is faster than Polyhedra.

Precision is evaluated in terms of how many assertions are successfully proved in each domain. The compared results before and after loop unrolling are shown in Table 1. We keep assertions proved before loop unrolling instead of discharging and prevent reproving them by adding assumptions. This approach explains why the total assertions grow dramatically after unrolling.

In the table, regardless of loop unrolling stage, the number of assertions proved by Template DBM lies between Zones and Polyhedra. We are more precise than Zones because proving assertions requires TVPI constraints, which Zones cannot express. For instance, in array_list, hash_table, and tango categories, most assertion checks require constraints like $\mathit{offset} * 4 - \mathit{size} \leq 0$ with offset as pointer offset and size as object size, thus driving significantly different assertion rates across domains. Compared to the "others" category from `aws-c-sdk`, where almost all domains prove the same number of assertions since most checks only require UTVPI constraints to prove. An important observation is that analyzing each benchmark uses fewer than three coefficients from the template. Consequently, the number of non-unit coefficient TVPI constraints grows sparsely yet remains sufficient to verify most assertions.

On the other hand, Template DBM and Polyhedra solve assertions at similar rates across most benchmarks. There are 8 benchmarks where Polyhedra solves extra 9 assertions before and 59 after unrolling. Among these 8 cases, Polyhedra covers more general linear inequalities, such as $x+y+z \leq 10$, that neither Zones nor Template DBM support. Template DBM also fails to prove some assertions due to its limited support for the assignment transfer function. The implementation only handles the form $x := e$ and not scaled assignments such as $24 * x := 24 * e$, though it can be notably improved. However, these assertions represent only a small fraction of all benchmarks and assertions.

In the "others" category from `firedancer`, Polyhedra proves fewer assertions than either Zones or Template DBM. It fails on 79 assertions on 4 cases before loop unrolling and 52 across 4 cases after unrolling. ELINA library logs report coefficient-overflow and vector-product exceptions [9] during these analyses, which cause imprecision. To isolate the ELINA flaw, we use the APRON [13] and PPL [1] Polyhedra as back-ends to verify these cases; either they got the same assertion rate as Template DBM or timed out. We therefore conclude that Polyhedra can, in theory, prove these assertions but, as Table 1 shows, it fails due to limitations in the ELINA implementation.

Overall, our experiment demonstrates that Template DBM offers greater scalability than Polyhedra while providing higher precision than Zones.

6 Related Work

We have already seen some abstract domains close to our work in Section 1. This section explores their deeper connections and examines alternative approaches.

The TVPI domain, originally from [20], represents arbitrary inequalities of the form $ax + by \leq c$ with $a, b, c \in \mathbb{Q}$. Our work restricts this to $ax - by \leq c$ where $a, b \in \mathcal{T}$ (a predefined coefficient template) and $c \in \mathbb{Z}$, matching the DBM structure for difference bounds. One way to extend our tDBM is to introduce dimensions ax^+ and ax^- (where $ax^+ = -ax^-$), as in the Octagons domain, to represent more general TVPI constraints $\pm ax \pm by \leq c$. However, this dimensional increase may cause a blow-up and thus degrade performance.

Our work instead prioritizes scalability, which relies on the underlying DBM operations. Template DBM performs saturation to expose all implicit constraints, directly applying the standard DBM operations without altering its structure. In contrast, the original work treats constraints as a geometric polyhedron, leading to very different design choices for domain operations.

Our work aims to limit the form of TVPI constraints by fixing the coefficient template. A similar approach has been applied in other abstract domains. Logahedra [12] is a TVPI-based domain which restricts coefficients to powers of

[9] Issue report: `https://github.com/eth-sri/ELINA/issues/39`

two. The work also introduces a bounded version, which limits the exponent and thus represents a finite set of inequalities. This version preserves cubic time complexity for core operations like completion and join. Unlike Logahedra, our domain allows custom coefficients, offering a more flexible configuration since array strides are not always powers of two [10]. Template Polyhedra [19] domain fixes linear expressions ahead of time by a predefined template and tracks linear inequalities only for those expressions. As a result, each abstract operation runs in polynomial time relative to the number of template expressions. However, the template must be chosen heuristically at each program location, and each post-condition operation invokes a linear programming (LP) solver to compute the tightest bound for each template expression. Our approach requires only one coefficient template to restrict the TVPI form and computes post abstract states efficiently using the incremental saturation algorithm we propose whenever a new assignment or assumption is added.

The Weighted Hexagon [8] domain captures invariants of the form $x \in [-a, b] \wedge x \leq a \cdot y$ where $a, b \in \mathbb{I}^{\geq 0}$ with $\mathbb{I}$ representing reals or rationals. It features at most six edges per pair of variables and provides a transitive closure algorithm in cubic time. However, it is less expressive because $x \leq a \cdot y$ relations are limited to only the maximal and minimal slopes, and the domain cannot represent constraints with constant offsets. For example, $x - 2y \leq -3$ can only be over-approximated as $x \leq 2y$. The Stripes [7] domain expresses linear inequalities of the form $x - a \cdot (y[+z]) \geq b$ with $a, b \in \mathbb{Z}$. It serves only as a subdomain for the symbolic representation of such inequalities and for propagating information back and forth between other subdomains. It is built in Clousot [6] and serves a similar purpose, but with a different trade-off between precision and efficiency, and with different expectations of surrounding domains. For example, Stripes assumes that equalities are maintained by some other domain along it. This makes direct empirical comparison with Template DBM difficult since they are not easily implemented within the same system.

7 Conclusion

We introduce a new weakly numerical abstract domain, Template DBM, representing inequalities of the form $ax - by \leq c$, between pairs of variables x and y, where a and b come from a predefined coefficient template and c is an integer constant. This work shows how to use a DBM to represent domain elements. We provide algorithms for full saturation, incremental saturation, and lattice operations for the domain. In terms of precision and performance, Template DBM lies between the Zones and Polyhedra domains. Our experiments demonstrate that its runtime is comparable to Zones, while the number of assertion checks it solves for memory-safety verification is close to that of Polyhedra.

[10] In the hash_table category, we capture TVPI constraints requiring a coefficient of 24, which is the allocation size of a C structure without alignment.

References

1. Bagnara, R., Hill, P.M., Zaffanella, E.: The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comput. Program. **72**(1-2), 3–21 (2008). `https://doi.org/10.1016/J.SCICO.2007.08.001`, `https://doi.org/10.1016/j.scico.2007.08.001`
2. Ballou, K., Sherman, E.: Incremental transitive closure for zonal abstract domain. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13260, pp. 800–808. Springer (2022). `https://doi.org/10.1007/978-3-031-06773-0_43`, `https://doi.org/10.1007/978-3-031-06773-0_43`
3. Chawdhary, A., Robbins, E., King, A.: Incrementally closing octagons. Formal Methods Syst. Des. **54**(2), 232–277 (2019). `https://doi.org/10.1007/S10703-017-0314-7`, `https://doi.org/10.1007/s10703-017-0314-7`
4. Chevalier, M., Feret, J.: Sharing ghost variables in a collection of abstract domains. In: Beyer, D., Zufferey, D. (eds.) Verification, Model Checking, and Abstract Interpretation - 21st International Conference, VMCAI 2020, New Orleans, LA, USA, January 16-21, 2020, Proceedings. Lecture Notes in Computer Science, vol. 11990, pp. 158–179. Springer (2020). `https://doi.org/10.1007/978-3-030-39322-9_8`, `https://doi.org/10.1007/978-3-030-39322-9_8`
5. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Aho, A.V., Zilles, S.N., Szymanski, T.G. (eds.) Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978. pp. 84–96. ACM Press (1978). `https://doi.org/10.1145/512760.512770`, `https://doi.org/10.1145/512760.512770`
6. Fähndrich, M., Logozzo, F.: Static contract checking with abstract interpretation. In: Beckert, B., Marché, C. (eds.) Formal Verification of Object-Oriented Software - International Conference, FoVeOOS 2010, Paris, France, June 28-30, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6528, pp. 10–30. Springer (2010). `https://doi.org/10.1007/978-3-642-18070-5_2`, `https://doi.org/10.1007/978-3-642-18070-5_2`
7. Ferrara, P., Logozzo, F., Fähndrich, M.: Safer unsafe code for .net. In: Harris, G.E. (ed.) Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, October 19-23, 2008, Nashville, TN, USA. pp. 329–346. ACM (2008). `https://doi.org/10.1145/1449764.1449791`, `https://doi.org/10.1145/1449764.1449791`
8. Fulara, J., Durnoga, K., Jakubczyk, K., Schubert, A.: Relational abstract domain of weighted hexagons. Electron. Notes Theor. Comput. Sci. **267**(1), 59–72 (Oct 2010). `https://doi.org/10.1016/j.entcs.2010.09.006`, `https://doi.org/10.1016/j.entcs.2010.09.006`
9. Gange, G., Ma, Z., Navas, J.A., Schachte, P., Søndergaard, H., Stuckey, P.J.: A fresh look at zones and octagons. ACM Trans. Program. Lang. Syst. **43**(3), 11:1–11:51 (2021). `https://doi.org/10.1145/3457885`, `https://doi.org/10.1145/3457885`
10. Gange, G., Navas, J.A., Schachte, P., Søndergaard, H., Stuckey, P.J.: Exploiting sparsity in difference-bound matrices. In: Rival, X. (ed.) Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9837, pp. 189–211.

Springer (2016). https://doi.org/10.1007/978-3-662-53413-7_10, https://doi.org/10.1007/978-3-662-53413-7_10

11. Gurfinkel, A., Navas, J.A.: Abstract interpretation of LLVM with a region-based memory model. In: Bloem, R., Dimitrova, R., Fan, C., Sharygina, N. (eds.) Software Verification - 13th International Conference, VSTTE 2021, New Haven, CT, USA, October 18-19, 2021, and 14th International Workshop, NSV 2021, Los Angeles, CA, USA, July 18-19, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13124, pp. 122–144. Springer (2021). https://doi.org/10.1007/978-3-030-95561-8_8, https://doi.org/10.1007/978-3-030-95561-8_8
12. Howe, J.M., King, A.: Logahedra: A new weakly relational domain. In: Liu, Z., Ravn, A.P. (eds.) Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5799, pp. 306–320. Springer (2009). https://doi.org/10.1007/978-3-642-04761-9_23, https://doi.org/10.1007/978-3-642-04761-9_23
13. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5643, pp. 661–667. Springer (2009). https://doi.org/10.1007/978-3-642-02658-4_52, https://doi.org/10.1007/978-3-642-02658-4_52
14. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (eds.) Programs as Data Objects, Second Symposium, PADO 2001, Aarhus, Denmark, May 21-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2053, pp. 155–172. Springer (2001). https://doi.org/10.1007/3-540-44978-7_10, https://doi.org/10.1007/3-540-44978-7_10
15. Miné, A.: The octagon abstract domain. In: Burd, E., Aiken, P., Koschke, R. (eds.) Proceedings of the Eighth Working Conference on Reverse Engineering, WCRE'01, Stuttgart, Germany, October 2-5, 2001. p. 310. IEEE Computer Society (2001). https://doi.org/10.1109/WCRE.2001.957836, https://doi.org/10.1109/WCRE.2001.957836
16. Miné, A.: Weakly Relational Numerical Abstract Domains. (Domaines numériques abstraits faiblement relationnels). Ph.D. thesis, École Polytechnique, Palaiseau, France (2004), https://tel.archives-ouvertes.fr/tel-00136630
17. Miné, A.: The octagon abstract domain. High. Order Symb. Comput. **19**(1), 31–100 (2006). https://doi.org/10.1007/s10990-006-8609-1, https://doi.org/10.1007/s10990-006-8609-1
18. Nelson, C.G.: An $n^{log n}$ algorithm for the two-variable-per-constraint linear programming satisfiability problem. Tech. rep., Stanford University, Stanford, CA, USA (1978)
19. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Cousot, R. (ed.) Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3385, pp. 25–41. Springer (2005). https://doi.org/10.1007/978-3-540-30579-8_2, https://doi.org/10.1007/978-3-540-30579-8_2
20. Simon, A., King, A., Howe, J.M.: Two variables per linear inequality as an abstract domain. In: Leuschel, M. (ed.) Logic Based Program Synthesis and Tranformation, 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17-20,2002, Revised Selected Papers. Lecture Notes in Computer Science, vol. 2664,

pp. 71–89. Springer (2002). `https://doi.org/10.1007/3-540-45013-0_7`, `https://doi.org/10.1007/3-540-45013-0_7`

21. Singh, G., Püschel, M., Vechev, M.T.: Making numerical program analysis fast. In: Grove, D., Blackburn, S.M. (eds.) Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015. pp. 303–313. ACM (2015). `https://doi.org/10.1145/2737924.2738000`, `https://doi.org/10.1145/2737924.2738000`
22. Singh, G., Püschel, M., Vechev, M.T.: Fast polyhedra abstract domain. In: Castagna, G., Gordon, A.D. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017. pp. 46–59. ACM (2017). `https://doi.org/10.1145/3009837.3009885`, `https://doi.org/10.1145/3009837.3009885`
23. Su, Y., Navas, J.A., Gurfinkel, A., Garcia-Contreras, I.: Automatic inference of relational object invariants. In: Krishna, S., Sankaranarayanan, S., Trivedi, A. (eds.) Verification, Model Checking, and Abstract Interpretation - 26th International Conference, VMCAI 2025, Denver, CO, USA, January 20-21, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15529, pp. 214–236. Springer (2025). `https://doi.org/10.1007/978-3-031-82700-6_10`, `https://doi.org/10.1007/978-3-031-82700-6_10`
24. Zhou, J., Criswell, J., Hicks, M.: Fat pointers for temporal memory safety of C. Proc. ACM Program. Lang. **7**(OOPSLA1), 316–347 (2023). `https://doi.org/10.1145/3586038`, `https://doi.org/10.1145/3586038`