

# A New Numerical Abstract Domain Based on Difference-Bound Matrices

Antoine Miné

École Normale Supérieure de Paris, France,  
mine@di.ens.fr,  
<http://www.di.ens.fr/~mine>

**Abstract.** This paper presents a new numerical abstract domain for static analysis by abstract interpretation. This domain allows us to represent invariants of the form  $(x - y \leq c)$  and  $(\pm x \leq c)$ , where  $x$  and  $y$  are variables values and  $c$  is an integer or real constant.

Abstract elements are represented by Difference-Bound Matrices, widely used by model-checkers, but we had to design new operators to meet the needs of abstract interpretation. The result is a complete lattice of infinite height featuring widening, narrowing and common transfer functions.

We focus on giving an efficient  $\mathcal{O}(n^2)$  representation and graph-based  $\mathcal{O}(n^3)$  algorithms—where  $n$  is the number of variables—and claim that this domain always performs more precisely than the well-known interval domain.

To illustrate the precision/cost tradeoff of this domain, we have implemented simple abstract interpreters for toy imperative and parallel languages which allowed us to prove some non-trivial algorithms correct.

## 1 Introduction

Abstract interpretation has proved to be a useful tool for eliminating bugs in software because it allows the design of automatic and sound analyzers for real-life programming languages. While abstract interpretation is a very general framework, we will be interested here only in discovering numerical invariants, that is to say, arithmetic relations that hold between numerical variables in a program. Such invariants are useful for tracking common errors such as division by zero and out-of-bound array access.

In this paper we propose practical algorithms to discover invariants of the form  $(x - y \leq c)$  and  $(\pm x \leq c)$ —where  $x$  and  $y$  are numerical program variables and  $c$  is a numeric constant. Our method works for integers, reals and even rationals.

For the sake of brevity, we will omit proofs of theorems in this paper. The complete proof for all theorems can be found in the author's MS thesis [12].

**Previous and Related Work.** Static analysis has developed approaches to automatically find numerical invariants based on *numerical abstract domains*

representing the form of the invariants we want to find. Famous examples are the lattice of intervals (described in, for instance, Cousot and Cousot’s ISOP’76 paper [4]) and the lattice of polyhedra (described in Cousot and Halbwachs’s POPL’78 paper [8]) which represent respectively invariants of the form ( $v \in [c_1, c_2]$ ) and ( $\alpha_1 v_1 + \dots + \alpha_n v_n \leq c$ ). Whereas the interval analysis is very efficient—linear memory and time cost—but not very precise, the polyhedron analysis is much more precise but has a huge memory cost—exponential in the number of variables.

Invariants of the form ( $x - y \leq c$ ) and ( $\pm x \leq c$ ) are widely used by the *model-checking* community. A special representation, called *Difference-Bound Matrices (DBMs)*, was introduced, as well as many operators in order to model-check *timed automata* (see Yovine’s ES’98 paper [14] and Larsen, Larsson, Pettersson and Yi’s RTSS’97 paper [10]). Unfortunately, most operators are tied to model-checking and are of little interest for static analysis.

**Our Contribution.** This paper presents a new abstract numerical domain based on the DBM representation, together with a full set of new operators and transfer functions adapted to static analysis.

Sections 2 and 3 present a few well-known results about potential constraint sets and introduce briefly the Difference-Bound Matrices. Section 4 presents operators and transfer functions that are new—except for the intersection operator—and adapted to abstract interpretation. In Section 5, we use these operators to build *lattices*, which can be complete under certain conditions. Section 6 shows some practical results we obtained with an example implementation and Section 7 gives some ideas for improvement.

## 2 Difference-Bound Matrices

Let  $\mathcal{V} = \{v_1, \dots, v_n\}$  be a finite set a variables with value in a numerical set  $\mathbb{I}$  (which can be the set  $\mathbb{Z}$  of integers, the set  $\mathbb{Q}$  of rationals or the set  $\mathbb{R}$  of reals).

We focus, in this paper, on the representation of constraints of the form ( $v_j - v_i \leq c$ ), ( $v_i \leq c$ ) and ( $v_i \geq c$ ), where  $v_i, v_j \in \mathcal{V}$  and  $c \in \mathbb{I}$ . By choosing one variable to be always equal to 0, we can represent the above constraints using only *potential constraints*, that is to say, constraints of the form ( $v_j - v_i \leq c$ ). From now, we will choose  $v_2, \dots, v_n$  to be program variables, and  $v_1$  to be the constant 0 so that ( $v_i \leq c$ ) and ( $v_i \geq c$ ) are rewritten ( $v_i - v_1 \leq c$ ) and ( $v_1 - v_i \leq -c$ ). We assume we now work only with potential constraints over the set  $\{v_1, \dots, v_n\}$ .

**Difference-Bound Matrices.** We extend  $\mathbb{I}$  to  $\bar{\mathbb{I}} = \mathbb{I} \cup \{+\infty\}$  by adding the  $+\infty$  element. The standard operations  $\leq, =, +, \min$  and  $\max$  are extended to  $\bar{\mathbb{I}}$  as usual (we will not use operations, such as  $-$  or  $*$ , that may lead to indeterminate forms).

Any set  $C$  of potential constraints over  $\mathcal{V}$  can be represented uniquely by a  $n \times n$  matrix in  $\bar{\mathbb{I}}$ —provided we assume, without loss of generality, that there does not

exist two potential constraints  $(v_j - v_i \leq c)$  in  $C$  with the same left member and different right members. The matrix  $\mathbf{m}$  associated with the potential constraint set  $C$  is called a *Difference-Bound Matrix (DBM)* and is defined as follows:

$$\mathbf{m}_{ij} \triangleq \begin{cases} c & \text{if } (v_j - v_i \leq c) \in C, \\ +\infty & \text{elsewhere .} \end{cases}$$

**Potential Graphs.** A DBM  $\mathbf{m}$  can be seen as the adjacency matrix of a directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{A}, w)$  with edges weighted in  $\mathbb{I}$ .  $\mathcal{V}$  is the set of nodes,  $\mathcal{A} \subseteq \mathcal{V}^2$  is the set of edges and  $w \in \mathcal{A} \mapsto \mathbb{I}$  is the weight function.  $\mathcal{G}$  is defined by:

$$\begin{cases} (v_i, v_j) \notin \mathcal{A} & \text{if } \mathbf{m}_{ij} = +\infty, \\ (v_i, v_j) \in \mathcal{A} \text{ and } w(v_i, v_j) = \mathbf{m}_{ij} & \text{if } \mathbf{m}_{ij} \neq +\infty . \end{cases}$$

We will denote by  $\langle i_1, \dots, i_k \rangle$  a *finite* set of nodes representing a *path* from node  $v_{i_1}$  to node  $v_{i_k}$  in  $\mathcal{G}$ . A *cycle* is a path such that  $i_1 = i_k$ .

**$\mathcal{V}$ -Domain and  $\mathcal{V}^0$ -Domain.** We call the  $\mathcal{V}$ -*domain* of a DBM  $\mathbf{m}$  and we denote by  $\mathcal{D}(\mathbf{m})$  the set of points in  $\mathbb{I}^n$  that satisfy all potential constraints:

$$\mathcal{D}(\mathbf{m}) \triangleq \{(x_1, \dots, x_n) \in \mathbb{I}^n \mid \forall i, j, x_j - x_i \leq \mathbf{m}_{ij}\} .$$

Now, remember that the variable  $v_1$  has a special semantics: it is always equal to 0. Thus, it is not the  $\mathcal{V}$ -domain which is of interest, but the  $\mathcal{V}^0$ -*domain* (which is a sort of intersection-projection of the  $\mathcal{V}$ -domain) denoted by  $\mathcal{D}^0(\mathbf{m})$  and defined by:

$$\mathcal{D}^0(\mathbf{m}) \triangleq \{(x_2, \dots, x_n) \in \mathbb{I}^{n-1} \mid (0, x_2, \dots, x_n) \in \mathcal{D}(\mathbf{m})\} .$$

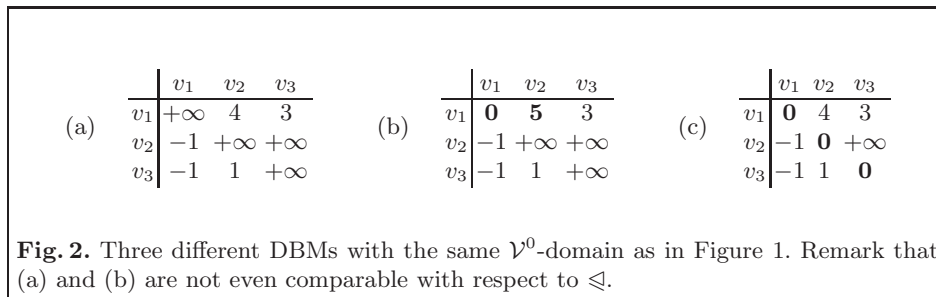
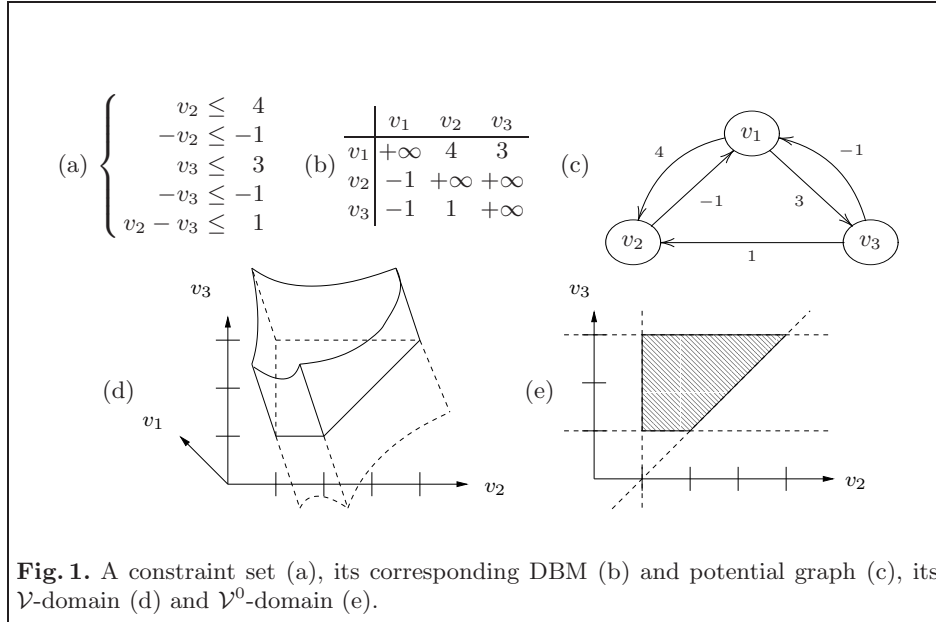
We will call  $\mathcal{V}$ -*domain* and  $\mathcal{V}^0$ -*domain* any subset of  $\mathbb{I}^n$  or  $\mathbb{I}^{n-1}$  which is respectively the  $\mathcal{V}$ -domain or the  $\mathcal{V}^0$ -domain of some DBM. Figure 1 shows an example DBM together with its corresponding potential graph, constraint set,  $\mathcal{V}$ -domain and  $\mathcal{V}^0$ -domain.

**$\trianglelefteq$  Order.** The  $\leq$  order on  $\bar{\mathbb{I}}$  induces a point-wise order  $\trianglelefteq$  on the set of DBMs:

$$\mathbf{m} \trianglelefteq \mathbf{n} \triangleq \forall i, j, \mathbf{m}_{ij} \leq \mathbf{n}_{ij} .$$

This order is *partial*. It is also *complete* if  $\mathbb{I}$  has least-upper bounds, i.e, if  $\mathbb{I}$  is  $\mathbb{R}$  or  $\mathbb{Z}$ , but not  $\mathbb{Q}$ . We will denote by  $=$  the associated equality relation which is simply the matrix equality.

We have  $\mathbf{m} \trianglelefteq \mathbf{n} \implies \mathcal{D}^0(\mathbf{m}) \subseteq \mathcal{D}^0(\mathbf{n})$  but the converse is not true. In particular, we *do not have*  $\mathcal{D}^0(\mathbf{m}) = \mathcal{D}^0(\mathbf{n}) \implies \mathbf{m} = \mathbf{n}$  (see Figure 2 for a counter-example).



### 3 Closure, Emptiness, Inclusion and Equality Tests

We saw in Figure 2 that two different DBMs can represent the same  $\mathcal{V}^0$ -domain. In this section, we show that there exists a normal form for any DBM with a non-empty  $\mathcal{V}^0$ -domain and present an algorithm to find it. The existence and computability of a normal form is very important since it is, as often in abstract representations, the key to equality testing used in fixpoint computation. In the case of DBMs, it will also allow us to carry an analysis of the precision of the operators defined in the next section.

**Emptiness Testing.** We have the following graph-oriented theorem:

**Theorem 1.**

*A DBM has an empty  $\mathcal{V}^0$ -domain if and only if there exists, in its associated potential graph, a cycle with a strictly negative total weight.*  $\square$

Checking for cycles with a strictly negative weight is done using the well-known *Bellman-Ford algorithm* which runs in  $\mathcal{O}(n^3)$ . This algorithm can be found in Cormen, Leiserson and Rivest's classical algorithmics textbook [2, §25.3].

**Closure and Normal Form.** Let  $\mathbf{m}$  be a DBM with a non-empty  $\mathcal{V}^0$ -domain and  $\mathcal{G}$  its associated potential graph. Since  $\mathcal{G}$  has no cycle with a strictly negative weight, we can compute its *shortest path closure*  $\mathcal{G}^*$ , the adjacency matrix of which will be denoted by  $\mathbf{m}^*$  and defined by:

$$\begin{cases} \mathbf{m}_{ii}^* \triangleq 0, \\ \mathbf{m}_{ij}^* \triangleq \min_{\substack{1 \leq N \\ \langle i=i_1, i_2, \dots, i_N=j \rangle}} \sum_{k=1}^{N-1} \mathbf{m}_{i_k i_{k+1}} & \text{if } i \neq j. \end{cases}$$

The idea of closure relies on the fact that, if  $\langle i = i_1, i_2, \dots, i_N = j \rangle$  is a path from  $v_i$  to  $v_j$ , then the constraint  $v_j - v_i \leq \sum_{k=1}^{N-1} \mathbf{m}_{i_k i_{k+1}}$  can be derived from  $\mathbf{m}$  by adding the potential constraints  $v_{i_{k+1}} - v_{i_k} \leq \mathbf{m}_{i_k i_{k+1}}, 1 \leq k \leq N-1$ . This is an *implicit* potential constraint which does not appear directly in the DBM  $\mathbf{m}$ . When computing the closure, we replace each potential constraint  $v_j - v_i \leq \mathbf{m}_{ij}, i \neq j$  in  $\mathbf{m}$  by the tightest implicit constraint we can find, and each diagonal element by 0 (which is indeed the smallest value  $v_i - v_i$  can reach). In Figure 2 for instance, (c) is the closure of both the (a) and (b) DBMs.

**Theorem 2.**

1.  $\mathbf{m}^* = \inf_{\triangleleft} \{ \mathbf{n} \mid \mathcal{D}^0(\mathbf{n}) = \mathcal{D}^0(\mathbf{m}) \}$ .
2.  $\mathcal{D}^0(\mathbf{m})$  saturates  $\mathbf{m}^*$ , that is to say:  
 $\forall i, j, \text{ such that } \mathbf{m}_{ij}^* < +\infty, \exists (x_1 = 0, x_2, \dots, x_n) \in \mathcal{D}(\mathbf{m}), x_j - x_i = \mathbf{m}_{ij}^*.$

$\square$

Theorem 2.1 states that  $\mathbf{m}^*$  is the smallest DBM—with respect to  $\triangleleft$ —that represents a given  $\mathcal{V}^0$ -domain, and thus *the closed form is a normal form*. Theorem 2.2 is a crucial property to prove accuracy of some operators defined in the next section.

Any shortest-path graph algorithm can be used to compute the closure of a DBM. We suggest the straightforward *Floyd-Warshall*, which is described in Cormen, Leiserson and Rivest’s textbook [2, §26.2], and has a  $\mathcal{O}(n^3)$  time cost.

**Equality and Inclusion Testing.** The case where  $\mathbf{m}$  or  $\mathbf{n}$  or both have an empty  $\mathcal{V}^0$ -domain is easy; in all other cases we use the following theorem—which is a consequence of Theorem 2.1:

**Theorem 3.**

1. If  $\mathbf{m}$  and  $\mathbf{n}$  have non-empty  $\mathcal{V}^0$ -domain,  $\mathcal{D}^0(\mathbf{m}) = \mathcal{D}^0(\mathbf{n}) \iff \mathbf{m}^* = \mathbf{n}^*$ .
2. If  $\mathbf{m}$  and  $\mathbf{n}$  have non-empty  $\mathcal{V}^0$ -domain,  $\mathcal{D}^0(\mathbf{m}) \subseteq \mathcal{D}^0(\mathbf{n}) \iff \mathbf{m}^* \triangleleft \mathbf{n}$ .

□

Besides emptiness test and closure, we may need, in order to test equality or inclusion, to compare matrices with respect to the point-wise ordering  $\triangleleft$ . This can be done with a  $\mathcal{O}(n^2)$  time cost.

**Projection.** We define the *projection*  $\pi_{|v_k}(\mathbf{m})$  of a DBM  $\mathbf{m}$  with respect to a variable  $v_k$  to be the interval containing all possible values of  $v \in \mathbb{I}$  such that there exists a point  $(x_2, \dots, x_n)$  in the  $\mathcal{V}^0$ -domain of  $\mathbf{m}$  with  $x_k = v$ :

$$\pi_{|v_k}(\mathbf{m}) \triangleq \{x \in \mathbb{I} \mid \exists(x_2, \dots, x_n) \in \mathcal{D}^0(\mathbf{m}) \text{ such that } x = x_k\} .$$

The following theorem, which is a consequence of the saturation property of the closure, gives an algorithmic way to compute the projection:

**Theorem 4.**

If  $\mathbf{m}$  has a non-empty  $\mathcal{V}^0$ -domain, then  $\pi_{|v_k}(\mathbf{m}) = [-\mathbf{m}_{k1}^*, \mathbf{m}_{1k}^*]$  (interval bounds are included only if finite).

□

## 4 Operators and Transfer Functions

In this section, we define some operators and transfer functions to be used in abstract semantics. Except for the intersection operator, they are new. The operators are basically point-wise extensions of the standard operators defined over the domain of intervals [4].

Most algorithms presented here are either constant time, or point-wise, i.e., quadratic time.

**Intersection.** Let us define the point-wise *intersection DBM*  $\mathbf{m} \wedge \mathbf{n}$  by:

$$(\mathbf{m} \wedge \mathbf{n})_{ij} \triangleq \min(\mathbf{m}_{ij}, \mathbf{n}_{ij}) .$$

We have the following theorem:

**Theorem 5.**

$$\mathcal{D}^0(\mathbf{m} \wedge \mathbf{n}) = \mathcal{D}^0(\mathbf{m}) \cap \mathcal{D}^0(\mathbf{n}). \quad \square$$

stating that the intersection is always exact. However, the resulting DBM is seldom closed, even if the arguments are closed.

**Least Upper Bound.** The set of  $\mathcal{V}^0$ -domains is not stable by union<sup>1</sup> so we introduce here a union operator which over-approximate its result. We define the point-wise *least upper bound DBM*  $\mathbf{m} \vee \mathbf{n}$  by:

$$(\mathbf{m} \vee \mathbf{n})_{ij} \triangleq \max(\mathbf{m}_{ij}, \mathbf{n}_{ij}) .$$

$\mathbf{m} \vee \mathbf{n}$  is indeed the least upper bound with respect to the  $\leq$  order. The following theorem tells us about the effect of this operator on  $\mathcal{V}^0$ -domains:

**Theorem 6.**

1.  $\mathcal{D}^0(\mathbf{m} \vee \mathbf{n}) \supseteq \mathcal{D}^0(\mathbf{m}) \cup \mathcal{D}^0(\mathbf{n})$ .
2. If  $\mathbf{m}$  and  $\mathbf{n}$  have non-empty  $\mathcal{V}^0$ -domains, then

$$(\mathbf{m}^*) \vee (\mathbf{n}^*) = \inf_{\leq} \{ \mathbf{o} \mid \mathcal{D}^0(\mathbf{o}) \supseteq \mathcal{D}^0(\mathbf{m}) \cup \mathcal{D}^0(\mathbf{n}) \}$$

and, as a consequence,  $\mathcal{D}^0((\mathbf{m}^*) \vee (\mathbf{n}^*))$  is the smallest  $\mathcal{V}^0$ -domain (with respect to the  $\subseteq$  ordering) which contains  $\mathcal{D}^0(\mathbf{m}) \cup \mathcal{D}^0(\mathbf{n})$ .

3. If  $\mathbf{m}$  and  $\mathbf{n}$  are closed, then so is  $\mathbf{m} \vee \mathbf{n}$ .

□

Theorem 6.1 states that  $\mathcal{D}^0(\mathbf{m} \vee \mathbf{n})$  is an upper bound in the set of  $\mathcal{V}^0$ -domains with respect to the  $\subseteq$  order. If precision is a concern, we need to find the *least* upper bound in this set. Theorem 6.2—which is a consequence of the saturation property of the closure—states that *we have to close both arguments* before applying the  $\vee$  operator to get this most precise union over-approximation. If one argument has an empty  $\mathcal{V}^0$ -domain, the least upper bound we want is simply the other argument. Emptiness tests and closure add a  $\mathcal{O}(n^3)$  time cost.

<sup>1</sup>  $\mathcal{V}^0$ -domains are always convex, but the union of two  $\mathcal{V}^0$ -domains may not be convex.

**Widening.** When computing the semantics of a program, one often encounters *loops* leading to fixpoint computation involving infinite iteration sequences. In order to compute *in finite time* an upper approximation of a fixpoint, *widening operators* were introduced in P. Cousot's thesis [3, §4.1.2.0.4]. Widening is a sort of union for which every increasing chain is stationary after a finite number of iterations. We define the point-wise widening operator  $\nabla$  by:

$$(\mathbf{m} \nabla \mathbf{n})_{ij} \triangleq \begin{cases} \mathbf{m}_{ij} & \text{if } \mathbf{n}_{ij} \leq \mathbf{m}_{ij}, \\ +\infty & \text{elsewhere.} \end{cases}$$

The following properties prove that  $\nabla$  is indeed a widening:

**Theorem 7.**

1.  $\mathcal{D}^0(\mathbf{m} \nabla \mathbf{n}) \supseteq \mathcal{D}^0(\mathbf{m}) \cup \mathcal{D}^0(\mathbf{n})$ .
2. Finite chain property:  
 $\forall \mathbf{m}$  and  $\forall (\mathbf{n}_i)_{i \in \mathbb{N}}$ , the chain defined by:

$$\begin{cases} \mathbf{x}_0 & \triangleq \mathbf{m}, \\ \mathbf{x}_{i+1} & \triangleq \mathbf{x}_i \nabla \mathbf{n}_i, \end{cases}$$

is increasing for  $\leq$  and ultimately stationary. The limit  $\mathbf{l}$  is such that  $\mathbf{l} \triangleright \mathbf{m}$  and  $\forall i, \mathbf{l} \triangleright \mathbf{n}_i$ .

□

The widening operator has some intriguing interactions with closure. Like the least upper bound, the widening operator gives more precise results if its right argument is closed, so it is rewarding to change  $\mathbf{x}_{i+1} = \mathbf{x}_i \nabla \mathbf{n}_i$  into  $\mathbf{x}_{i+1} = \mathbf{x}_i \nabla (\mathbf{n}_i^*)$ . This is not the case for the first argument: we can have sometimes  $\mathcal{D}^0(\mathbf{m} \nabla \mathbf{n}) \subsetneq \mathcal{D}^0((\mathbf{m}^*) \nabla \mathbf{n})$ . Worse, if we try to force the closure of the first argument by changing  $\mathbf{x}_{i+1} = \mathbf{x}_i \nabla \mathbf{n}_i$  into  $\mathbf{x}_{i+1} = (\mathbf{x}_i \nabla \mathbf{n}_i)^*$ , the finite chain property (Theorem 7.2) *is no longer satisfied*, as illustrated in Figure 3.

Originally [4], Cousot and Cousot defined widening over intervals  $\bar{\nabla}$  by:

$$[a, b] \bar{\nabla} [c, d] \triangleq [e, f],$$

where:

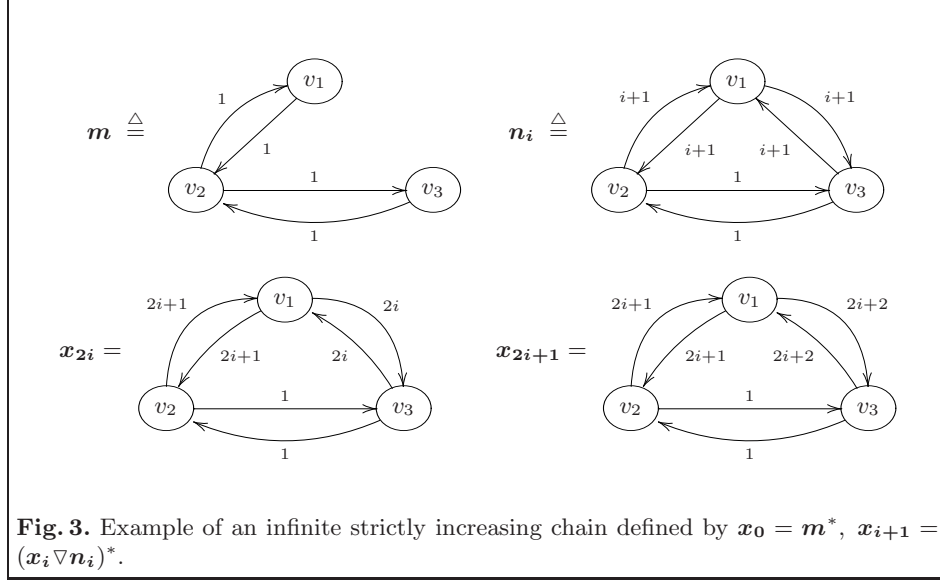
$$e \triangleq \begin{cases} a & \text{if } a \leq c, \\ -\infty & \text{elsewhere,} \end{cases} \quad f \triangleq \begin{cases} b & \text{if } b \geq d, \\ +\infty & \text{elsewhere.} \end{cases}$$

The following theorem proves that the sequence computed by our widening is *always* more precise than with the standard widening over intervals:

**Theorem 8.**

*If we have the following iterating sequence:*

$$\begin{cases} \mathbf{x}_0 & \triangleq \mathbf{m}^*, \\ \mathbf{x}_{k+1} & \triangleq \mathbf{x}_k \nabla (\mathbf{n}_k^*), \end{cases} \quad \begin{cases} [y_0, z_0] & \triangleq \pi_{|v_i}(\mathbf{m}), \\ [y_{k+1}, z_{k+1}] & \triangleq [y_k, z_k] \bar{\nabla} \pi_{|v_i}(\mathbf{n}_k), \end{cases}$$



then the sequence  $(\mathbf{x}_k)_{k \in \mathbb{N}}$  is more precise than the sequence  $([y_k, z_k])_{k \in \mathbb{N}}$  in the following sense:

$$\forall k, \pi_{|v_i}(\mathbf{x}_k) \subseteq [y_k, z_k] .$$

□

Remark that the technique, described in Cousot and Cousot's PLILP'92 paper [7], for improving the precision of the standard widening over intervals  $\overline{\nabla}$  can also be applied to our widening  $\nabla$ . It allows, for instance, deriving a widening that *always* gives better results than a simple sign analysis (which is not the case of  $\nabla$  nor  $\overline{\nabla}$ ). The resulting widening over DBMs will remain more precise than the resulting widening over intervals.

**Narrowing.** *Narrowing operators* were introduced in P. Cousot's thesis [3, §4.1.2.0.11] in order to restore, in a finite time, some information that may have been lost by widening applications. We define here a point-wise narrowing operator  $\Delta$  by:

$$(\mathbf{m} \Delta \mathbf{n})_{ij} \triangleq \begin{cases} \mathbf{n}_{ij} & \text{if } \mathbf{m}_{ij} = +\infty, \\ \mathbf{m}_{ij} & \text{elsewhere .} \end{cases}$$

The following properties prove that  $\Delta$  is indeed a narrowing:

**Theorem 9.**

1. If  $\mathcal{D}^0(\mathbf{n}) \subseteq \mathcal{D}^0(\mathbf{m})$ , then  $\mathcal{D}^0(\mathbf{n}) \subseteq \mathcal{D}^0(\mathbf{m} \Delta \mathbf{n}) \subseteq \mathcal{D}^0(\mathbf{m})$ .
2. Finite decreasing chain property:

$\forall \mathbf{m}$  and for any chain  $(\mathbf{n}_i)_{i \in \mathbb{N}}$  decreasing for  $\triangleleft$ , the chain defined by:

$$\begin{cases} \mathbf{x}_0 & \triangleq \mathbf{m}, \\ \mathbf{x}_{i+1} & \triangleq \mathbf{x}_i \triangle \mathbf{n}_i, \end{cases}$$

is decreasing and ultimately stationary. □

Given a sequence  $(\mathbf{n}_k)_{k \in \mathbb{N}}$  such that the chain  $(\mathcal{D}^0(\mathbf{n}_k))_{k \in \mathbb{N}}$  is decreasing for the  $\subseteq$  partial order (but not  $(\mathbf{n}_k)_{k \in \mathbb{N}}$  for the  $\triangleleft$  partial order), one way to ensure the best accuracy as well as the finiteness of the chain  $(\mathbf{x}_k)_{k \in \mathbb{N}}$  is to force the closure of the right argument by changing  $\mathbf{x}_{i+1} = \mathbf{x}_i \triangle \mathbf{n}_i$  into  $\mathbf{x}_{i+1} = \mathbf{x}_i \triangle (\mathbf{n}_i^*)$ . Unlike widening, forcing all elements in the chain to be closed with  $\mathbf{x}_{i+1} = (\mathbf{x}_i \triangle \mathbf{n}_i)^*$  poses no problem.

**Forget.** Given a DBM  $\mathbf{m}$  and a variable  $v_k$ , the *forget operator*  $\mathbf{m}_{\setminus v_k}$  computes a DBM where all informations about  $v_k$  are lost. It is the opposite of the projection operator  $\pi_{|v_k}$ . We define this operator by:

$$(\mathbf{m}_{\setminus v_k})_{ij} \triangleq \begin{cases} \min(\mathbf{m}_{ij}, \mathbf{m}_{ik} + \mathbf{m}_{kj}) & \text{if } i \neq k \text{ and } j \neq k, \\ 0 & \text{if } i = j = k, \\ +\infty & \text{elsewhere .} \end{cases}$$

The  $\mathcal{V}^0$ -domain of  $\mathbf{m}_{\setminus v_k}$  is obtained by projecting  $\mathcal{D}^0(\mathbf{m})$  on the subspace orthogonal to  $\mathbb{I}\vec{v}_k$ , and then extruding the result in the direction of  $\vec{v}_k$ :

**Theorem 10.**

$$\mathcal{D}^0(\mathbf{m}_{\setminus v_k}) = \{(x_2, \dots, x_n) \in \mathbb{I}^{n-1} \mid \exists x \in \mathbb{I}, (x_2, \dots, x_{k-1}, x, x_{k+1}, \dots, x_n) \in \mathcal{D}^0(\mathbf{m})\}. \quad \square$$

**Guard.** Given an arithmetic equality or inequality  $g$  over  $\{v_2, \dots, v_n\}$ —which we call a *guard*—and a DBM  $\mathbf{m}$ , the *guard transfer function* tries to find a new DBM  $\mathbf{m}_{(g)}$  the  $\mathcal{V}^0$ -domain of which is  $\{s \in \mathcal{D}^0(\mathbf{m}) \mid s \text{ satisfies } g\}$ . Since this is, in general, impossible, we will only try to have:

**Theorem 11.**

$$\mathcal{D}^0(\mathbf{m}_{(g)}) \supseteq \{s \in \mathcal{D}^0(\mathbf{m}) \mid s \text{ satisfies } g\}. \quad \square$$

Here is an example definition:

**Definition 12.**

1. If  $g = (v_{j_0} - v_{i_0} \leq c)$  with  $i_0 \neq j_0$ , then:

$$(\mathbf{m}_{(v_{j_0} - v_{i_0} \leq c)})_{ij} \triangleq \begin{cases} \min(\mathbf{m}_{ij}, c) & \text{if } i = i_0 \text{ and } j = j_0, \\ \mathbf{m}_{ij} & \text{elsewhere .} \end{cases}$$

The cases  $g = (v_{j_0} \leq c)$  and  $g = (-v_{i_0} \leq c)$  are settled by choosing respectively  $i_0 = 1$  and  $j_0 = 1$ .

2. If  $g = (v_{j_0} - v_{i_0} = c)$  with  $i_0 \neq j_0$ , then:

$$\mathbf{m}_{(v_{j_0} - v_{i_0} = c)} \triangleq (\mathbf{m}_{(v_{j_0} - v_{i_0} \leq c)})_{(v_{i_0} - v_{j_0} \leq -c)} .$$

The case  $g = (v_{j_0} = c)$  is a special case where  $i_0 = 1$ .

3. In all other cases, we simply choose:

$$\mathbf{m}_{(g)} \triangleq \mathbf{m} .$$

□

In all but the last—general—cases, the guard transfer function is exact.

**Assignment.** An *assignment*  $v_k \leftarrow e(v_2, \dots, v_n)$  is defined by a variable  $v_k$  and an arithmetic expression  $e$  over  $\{v_2, \dots, v_n\}$ .

Given a DBM  $\mathbf{m}$  representing all possible values that can take the variables set  $\{v_2, \dots, v_n\}$  at a program point, we look for a DBM, denoted by  $\mathbf{m}_{(v_k \leftarrow e)}$ , representing the possible values of the same variables set after the assignment  $v_k \leftarrow e$ . This is not possible in the general case, so the *assignment transfer function* will only try to find an upper approximation of this set:

**Theorem 13.**

$$\mathcal{D}^0(\mathbf{m}_{(v_k \leftarrow e)}) \supseteq \{(x_2, \dots, x_{k-1}, e(x_2, \dots, x_n), x_{k+1}, \dots, x_n) \mid (x_2, \dots, x_n) \in \mathcal{D}^0(\mathbf{m})\} .$$

□

For instance, we can use the following definition for  $\mathbf{m}_{(v_{i_0} \leftarrow e)}$ :

**Definition 14.**

1. If  $e = v_{i_0} + c$ , then:

$$(\mathbf{m}_{(v_{i_0} \leftarrow v_{i_0} + c)})_{ij} \triangleq \begin{cases} \mathbf{m}_{ij} - c & \text{if } i = i_0, j \neq j_0, \\ \mathbf{m}_{ij} + c & \text{if } i \neq i_0, j = j_0, \\ \mathbf{m}_{ij} & \text{elsewhere .} \end{cases}$$

2. If  $e = v_{j_0} + c$  with  $i_0 \neq j_0$ , then we use the forget operator and the guard transfer function:

$$\mathbf{m}_{(v_{i_0} \leftarrow v_{j_0} + c)} \triangleq ((\mathbf{m} \setminus v_{i_0})_{(v_{i_0} - v_{j_0} \leq c)})_{(v_{j_0} - v_{i_0} \leq -c)} .$$

The case  $e = c$  is a special case where we choose  $j_0 = 1$ .

3. In all other cases, we use a standard interval arithmetic to find an interval  $[-e^-, e^+]$ ,  $e^+, e^- \in \mathbb{I}$  such that

$$[-e^-, e^+] \supseteq e(\pi_{v_2}(\mathbf{m}), \dots, \pi_{v_n}(\mathbf{m}))$$

and then we define:

$$(\mathbf{m}_{(v_{i_0} \leftarrow e)})_{ij} \triangleq \begin{cases} e^+ & \text{if } i = 1 \text{ and } j = i_0, \\ e^- & \text{if } j = 1 \text{ and } i = i_0, \\ (\mathbf{m} \setminus v_{i_0})_{ij} & \text{elsewhere .} \end{cases}$$

□

In all but the last—general—cases, the assignment transfer function is exact.

**Comparison with the Abstract Domain of Intervals.** Most of the time, the precision of numerical abstract domains can only be compared experimentally on example programs (see Section 6 for such an example). However, we claim that the DBM domain *always* performs better than the domain of intervals.

To legitimize this assertion, we compare informally the effect of all abstract operations in the DBM and in the interval domains. Thanks to Theorems 5 and 6.2, and Definitions 12 and 14, the intersection and union abstract operators and the guard and assignment transfer functions are more precise than their interval counterpart. Thanks to Theorem 8, approximate fixpoint computation with our widening  $\nabla$  is always more accurate than with the standard widening over intervals  $\bar{\nabla}$  and one could prove easily that each iteration with our narrowing is more precise than with the standard narrowing over intervals. This means that *any* abstract semantics based on the operators and transfer functions we defined is *always* more precise than the corresponding interval-based abstract semantics.

## 5 Lattice Structures

In this section, we design two lattice structures: one on the set of DBMs and one on the set of closed DBMs. The first one is useful to analyze fixpoint transfer between abstract and concrete semantics and the second one allows us to design a meaning function—or even a Galois Connection—linking the set of abstract  $\mathcal{V}^0$ -domains to the concrete lattice  $\mathcal{P}(\{v_2, \dots, v_n\} \mapsto \mathbb{I})$ , following the abstract interpretation framework described in Cousot and Cousot’s POPL’79 paper [5].

**DBM Lattice.** The set  $\mathcal{M}$  of DBMs, together with the *order relation*  $\leq$  and the point-wise *least upper bound*  $\vee$  and *greatest lower bound*  $\wedge$ , is almost a lattice. It only needs a *least element*  $\perp$ , so we extend  $\leq$ ,  $\vee$  and  $\wedge$  to  $\mathcal{M}_\perp = \mathcal{M} \cup \{\perp\}$  in an obvious way to get  $\sqsubseteq$ ,  $\sqcup$  and  $\sqcap$ . The *greatest element*  $\top$  is the DBM with all its coefficients equal to  $+\infty$ .

**Theorem 15.**

1.  $(\mathcal{M}_\perp, \sqsubseteq, \sqcap, \sqcup, \perp, \top)$  is a lattice.
2. This lattice is complete if  $(\mathbb{I}, \leq)$  is complete ( $\mathbb{Z}$  or  $\mathbb{R}$ , but not  $\mathbb{Q}$ ).

□

There are, however, two problems with this lattice. First, we cannot easily assimilate this lattice to a sub-lattice of  $\mathcal{P}(\{v_2, \dots, v_n\} \mapsto \mathbb{I})$  as two different DBMs can have the same  $\mathcal{V}^0$ -domain. Then, the least upper bound operator  $\sqcup$  is not the most precise upper approximation of the union of two  $\mathcal{V}^0$ -domains because we do not force the arguments to be closed.

**Closed DBM Lattice.** To overcome these difficulties, we build another lattice based on closed DBMs. First, consider the set  $\mathcal{M}_\perp^*$  of closed DBMs  $\mathcal{M}^*$  with a *least element*  $\perp^*$  added. Now, we define a *greatest element*  $\top^*$ , a *partial order relation*  $\sqsubseteq^*$ , a *least upper bound*  $\sqcup^*$  and a *greatest lower bound*  $\sqcap^*$  in  $\mathcal{M}_\perp^*$  by:

$$\top^*_{ij} \triangleq \begin{cases} 0 & \text{if } i = j, \\ +\infty & \text{elsewhere .} \end{cases}$$

$$\mathbf{m} \sqsubseteq^* \mathbf{n} \triangleq \begin{cases} \text{either } \mathbf{m} = \perp^*, \\ \text{or } \mathbf{m} \neq \perp^*, \mathbf{n} \neq \perp^* \text{ and } \mathbf{m} \leq \mathbf{n} . \end{cases}$$

$$\mathbf{m} \sqcup^* \mathbf{n} \triangleq \begin{cases} \mathbf{m} & \text{if } \mathbf{n} = \perp^*, \\ \mathbf{n} & \text{if } \mathbf{m} = \perp^*, \\ \mathbf{m} \vee \mathbf{n} & \text{elsewhere .} \end{cases}$$

$$\mathbf{m} \sqcap^* \mathbf{n} \triangleq \begin{cases} \perp^* & \text{if } \mathbf{m} = \perp^* \text{ or } \mathbf{n} = \perp^* \text{ or } \mathcal{D}^0(\mathbf{m} \wedge \mathbf{n}) = \emptyset, \\ (\mathbf{m} \wedge \mathbf{n})^* & \text{elsewhere .} \end{cases}$$

Thanks to Theorem 2.1, every non-empty  $\mathcal{V}^0$ -domain has a unique representation in  $\mathcal{M}^*$ ;  $\perp^*$  is the representation for the empty set. We build a *meaning function*  $\gamma$  which is an extension of  $\mathcal{D}^0(\cdot)$  to  $\mathcal{M}_\perp^*$ :

$$\gamma(\mathbf{m}) \triangleq \begin{cases} \emptyset & \text{if } \mathbf{m} = \perp^*, \\ \mathcal{D}^0(\mathbf{m}) & \text{elsewhere .} \end{cases}$$

**Theorem 16.**

1.  $(\mathcal{M}_\perp^*, \sqsubseteq^*, \sqcap^*, \sqcup^*, \perp^*, \top^*)$  is a lattice and  $\gamma$  is one-to-one.
2. If  $(\mathbb{I}, \leq)$  is complete, this lattice is complete and  $\gamma$  is meet-preserving:  
 $\gamma(\sqcap^* X) = \bigcap \{\gamma(x) \mid x \in X\}$ . We can—according to Cousot and Cousot [6, Prop. 7]—build a canonical Galois Insertion:

$$\mathcal{P}(\{v_2, \dots, v_n\} \mapsto \mathbb{I}) \xleftarrow{\gamma} \xrightarrow{\alpha} \mathcal{M}_\perp^*$$

where the abstraction function  $\alpha$  is defined by:

$$\alpha(D) = \sqcap^* \{ m \in \mathcal{M}_\perp^* \mid D \subseteq \gamma(m) \}.$$

□

The  $\mathcal{M}_\perp^*$  lattice features a nice meaning function and a precise union approximation; thus, it is tempting to force all our operators and transfer functions to live in  $\mathcal{M}_\perp^*$  by forcing closure on their result. However, we saw this does not work for widening, so fixpoint computation *must* be performed in the  $\mathcal{M}_\perp$  lattice.

## 6 Results

The algorithms on DBMs presented here have been implemented in OCaml and used to perform forward analysis on toy—yet Turing-equivalent—imperative and parallel languages with only numerical variables and no procedure.

We present here neither the concrete and abstract semantics, nor the actual forward analysis algorithm used for our analyzers. They follow exactly the abstract interpretation scheme described in Cousot and Cousot’s POPL’79 paper [5] and Bourdoncle’s FMFA’93 paper [1] and are detailed in the author’s MS thesis [12]. Theorems 1, 3, 5, 6, 11 and 13 prove that all the operators and transfer functions we defined are indeed abstractions on the domain of DBMs of the usual operators and transfer functions on the concrete domain  $\mathcal{P}(\{v_2, \dots, v_n\} \mapsto \mathbb{I})$ , which, as shown by Cousot and Cousot [5], is sufficient to prove soundness for analyses.

**Imperative Programs.** Our toy forward analyzer for imperative language follows almost exactly the analyzer described in Cousot and Halbwachs’s POPL’78 paper [8], except that the abstract domain of polyhedra has been replaced by our DBM-based domain. We tested our analyzer on the well-known Bubble Sort and Heap Sort algorithms and managed to prove automatically that they do not produce out-of-bound error while accessing array elements. Although we did not find as many invariants as Cousot and Halbwachs for these two examples, it was sufficient to prove the correctness. We do not detail these common examples here for the sake of brevity.

**Parallel Programs.** Our toy analyzer for parallel language allows analyzing a fixed set of processes running concurrently and communicating through global variables. We use the well-known *nondeterministic interleaving* method in order to analyze all possible control flows. In this context, we managed to prove automatically that the Bakery algorithm, introduced in 1974 by Lamport [9], for synchronizing two parallel processes never lets the two processes be at the same time in their critical sections. We now detail this example.

**The Bakery Algorithm.** After the initialization of two global shared variables  $y_1$  and  $y_2$ , two processes  $p_1$  and  $p_2$  are spawned. They synchronize through the variables  $y_1$  and  $y_2$ , representing the priority of  $p_1$  and  $p_2$ , so that only one process at a time can enter its *critical section* (Figure 4).

Our analyzer for parallel processes is fed with the initialization code ( $y_1 = 0$ ;  $y_2 = 0$ ) and the control flow graphs for  $p_1$  and  $p_2$  (Figure 5). Each control graph is a set of control point nodes and some edges labeled with either an *action* performed when the edge is taken (the assignment  $y_1 \leftarrow y_2 + 1$ , for example) or a *guard* imposing a condition for taking the edge (the test  $y_1 \neq 0$ , for example).

The analyzer then computes the nondeterministic interleaving of  $p_1$  and  $p_2$  which is the product control flow graph. Then, it computes iteratively the abstract invariants holding at each product control point. It outputs the invariants shown in Figure 6.

The state  $(2, c)$  is never reached, which means that  $p_1$  and  $p_2$  cannot be at the same time in their critical section. This proves the correctness of the Bakery algorithm. Remark that our analyzer also discovered some non-obvious invariants, such as  $y_1 = y_2 + 1$  holding in the  $(1, c)$  state.

$y1 = 0; y2 = 0;$

(p1)

**while true do**

$y1 = y2 + 1;$

**while  $y2 \neq 0$  and  $y1 > y2$  do done;**

*--- critical section ---*

$y1 = 0;$

**done**

(p2)

**while true do**

$y2 = y1 + 1;$

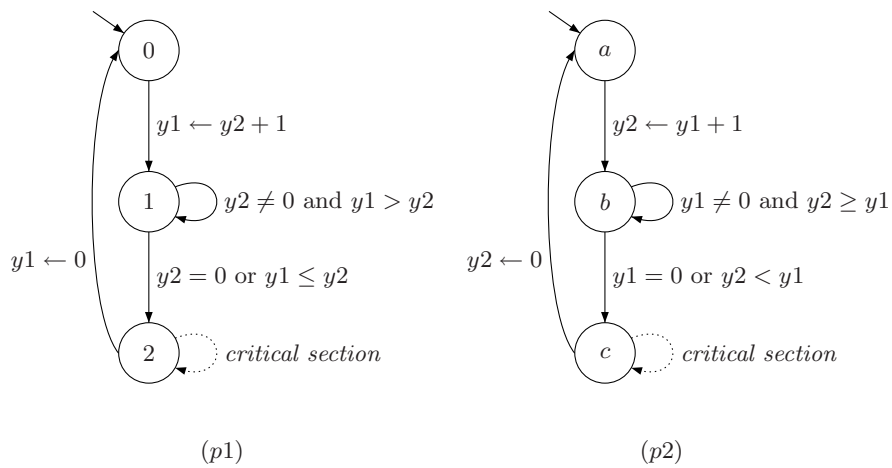
**while  $y1 \neq 0$  and  $y2 \geq y1$  do done;**

*--- critical section ---*

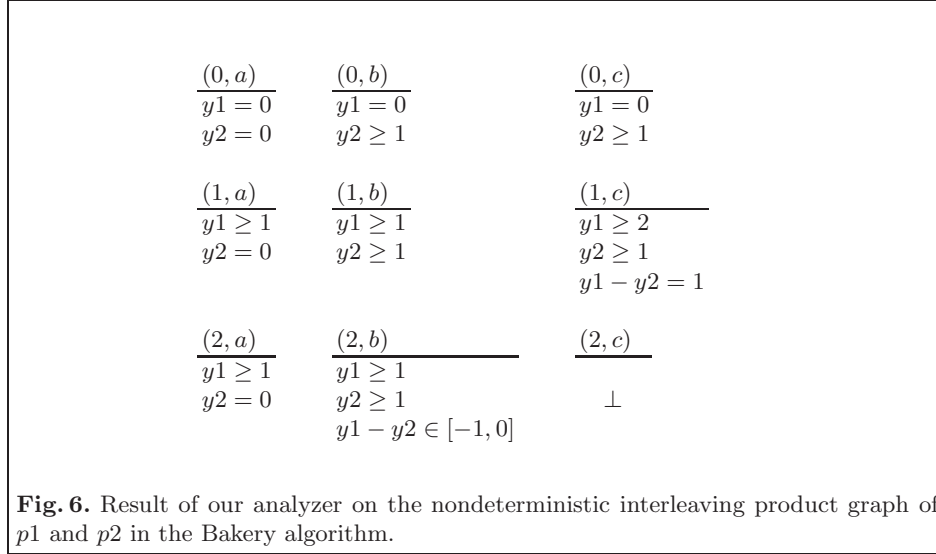
$y2 = 0;$

**done**

**Fig. 4.** Pseudo-code for the Bakery algorithm.



**Fig. 5.** Control flow graphs of processes  $p1$  and  $p2$  in the Bakery algorithm.



## 7 Extensions and Future Work

**Precision improvement.** In our analysis, we only find a coarse set of the invariants held in a program since finding *all* invariants of the form  $(x - y \leq c)$  and  $(\pm x \leq c)$  for all programs is non-computable. Possible losses of precision have three causes: non-exact union, widening in loops and non-exact assignment and guard transfer functions.

We made crude approximations in the last—general—case of Definitions 12 and 14 and there is room for improving assignment and guard transfer functions, even though exactness is impossible. When the DBM lattices are complete, there exists most precise transfer functions such that Theorems 11 and 13 hold, however these functions may be difficult to compute.

**Finite Union of  $\mathcal{V}^0$ -domains.** One can imagine to represent finite unions of  $\mathcal{V}^0$ -domains, using a finite set of DBMs instead of a single one as abstract state. This allows an exact union operator but it may lead to memory and time cost explosion as abstract states contain more and more DBMs, so one may need from time to time to replace a set of DBMs by their union approximation.

The model-checker community has also developed specific structures to represent finite unions of  $\mathcal{V}$ -domains, that are less costly than sets. *Clock-Difference Diagrams* (introduced in 1999 by Larsen, Weise, Yi and Pearson [11]) and *Difference Decision Diagrams* (introduced in Møller, Lichtenberg, Andersen and Hulgaard’s CSL’99 paper [13]) are tree-based structures made compact thanks to the sharing of isomorphic sub-trees; however existence of normal forms for such structures is only a conjecture at the time of writing and only local or

path reduction algorithms exist. One can imagine adapting such structures to abstract interpretation the way we adapted DBM in this paper.

**Space and Time Cost Improvement.** Space is often a big concern in abstract interpretation. The DBM representation we proposed in this paper has a fixed  $\mathcal{O}(n^2)$  memory cost—where  $n$  is the number of variables in the program. In the actual implementation, we decided to use the graph representation—or hollow matrix—which stores only edges with a finite weight and observed a great space gain as most DBMs we use have many  $+\infty$ . Most algorithms are also faster on hollow matrices and we chose to use the more complex, but more efficient, *Johnson* shortest-path closure algorithm—described in Cormen, Leiserson and Rivest’s textbook [2, §26.3]—instead of the *Floyd-Warshall* algorithm.

Larsen, Larsson, Pettersson and Yi’s RTSS’97 paper [10] presents a *minimal form algorithm* which finds a DBM with the fewest finite edges representing a given  $\mathcal{V}^0$ -domain. This minimal form could be useful for memory-efficient storing, but cannot be used for direct computation with algorithms requiring closed DBMs.

**Representation Improvement.** The invariants we manipulate are, in term of precision and complexity, between interval and polyhedron analysis. It is interesting to look for domains allowing the representation of more forms of invariants than DBMs in order to increase the granularity of numerical domains. We are currently working on an improvement of DBMs that allows us to represent, with a small time and space complexity overhead, invariants of the form  $(\pm x \pm y \leq c)$ .

## 8 Conclusion

We presented in this paper a new numerical abstract domain inspired from the well-known domain of intervals and the Difference-Bound Matrices. This domain allows us to manipulate invariants of the form  $(x - y \leq c)$ ,  $(x \leq c)$  and  $(x \geq c)$  with a  $\mathcal{O}(n^2)$  worst case memory cost per abstract state and  $\mathcal{O}(n^3)$  worst case time cost per abstract operation (where  $n$  is the number of variables in the program).

Our approach made it possible for us to prove the correctness of some non-trivial algorithms beyond the scope of interval analysis, for a much smaller cost than polyhedron analysis. We also proved that this analysis always gives better results than interval analysis, for a slightly greater cost.

**Acknowledgments.** I am grateful to J. Feret, C. Hymans, D. Monniaux, P. Cousot, O. Danvy and the anonymous referees for their helpful comments and suggestions.

## References

- [1] F. Bourdoncle. Efficient chaotic iteration strategies with widenings. In *FMPA '93*, number 735 in LNCS, pages 128–141. Springer-Verlag, 1993.
- [2] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
- [3] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse d'état ès sciences mathématiques, Université scientifique et médicale de Grenoble, France, 1978.
- [4] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. of the 2<sup>d</sup> Int. Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
- [5] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *ACM POPL'79*, pages 269–282. ACM Press, 1979.
- [6] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2-3):103–179, 1992.
- [7] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In *PLILP'92*, number 631 in LNCS, pages 269–295. Springer-Verlag, August 1992.
- [8] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM POPL'78*, pages 84–97. ACM Press, 1978.
- [9] L. Lamport. A new solution of dijkstra's concurrent programming problem. *Communications of the ACM*, 8(17):453–455, August 1974.
- [10] K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *IEEE RTSS'97*, pages 14–24. IEEE CS Press, December 1997.
- [11] K. Larsen, C. Weise, W. Yi, and J. Pearson. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, October 1999.
- [12] A. Miné. Representation of two-variable difference or sum constraint set and application to automatic program analysis. Master's thesis, ENS-DI, Paris, France, 2000. [http://www.eleves.ens.fr:8080/home/mine/stage\\_dea/](http://www.eleves.ens.fr:8080/home/mine/stage_dea/).
- [13] J. Møller, J. Lichtenberg, R. Andersen, H., and H. Hulgaard. Difference decision diagrams. In *CSL'99*, volume 1683 of LNCS, pages 111–125. Springer-Verlag, September 1999.
- [14] S. Yovine. Model-checking timed automata. In *Embedded Systems*, number 1494 in LNCS, pages 114–152. Springer-Verlag, October 1998.