Donut Domains: Efficient Non-Convex Domains for Abstract Interpretation

Khalil Ghorbal[1], Franjo Ivančić[1], Gogul Balakrishnan[1], Naoto Maeda[2], and Aarti Gupta[1]

[1] NEC Laboratories America, Inc.
[2] NEC Corporation, Kanagawa 211-8666, Japan

Abstract. Program analysis using abstract interpretation has been successfully applied in practice to find runtime bugs or prove software correct. Most abstract domains that are used widely rely on convexity for their scalability. However, the ability to express non-convex properties is sometimes required in order to achieve a precise analysis of some numerical properties. This work combines already known abstract domains in a novel way in order to design new abstract domains that tackle some non-convex invariants. The abstract objects of interest are encoded as a pair of two convex abstract objects: the first abstract object defines an over-approximation of the possible reached values, as is done customarily. The second abstract object under-approximates the set of impossible values within the state-space of the first abstract object. Therefore, the geometrical concretization of our objects is defined by a convex set minus another convex set (or hole). We thus call these domains *donut domains*.

1 Introduction

Efficient program analysis using abstract interpretation [11] typically uses convex domains such as intervals, octagons, zonotopes or polyhedra [10, 12, 14, 17, 26]. However, certain properties of interest require reasoning about non-convex structures. One approach to non-convex reasoning is to utilize powerset domains of elementary convex domains [4, 20, 21]. In general, it has proved to be difficult to provide satisfactory improvements over elementary convex domains with powerset domains while maintaining small enough performance degradation. Furthermore, it would be difficult to maintain enough disjunctions in the powerset depending on the particular non-convex shape being approximated. Note, however, that the recently proposed Boxes domain by Gurfinkel and Chaki [20] can potentially represent exponentially many interval constraints compactly. It utilizes a BDD-like extension to elementary range constraints called LDD [8]. However, we are considering relational domains such as octagons, zonotopes or polyhedra as well.

Additional non-convex domains based on congruence (either linear [19] or trapezoid [25]) analysis have been developed. Such domains catch a congruence relation that variables satisfy and are suitable for the analysis of indexes of arrays for instance. Recent work by Chen et al. considered a polyhedral abstract domain with interval coefficients [9]. This abstract domain has the ability to express certain non-convex invariants. For example, in this domain some multiplications can be evaluated precisely (see Figure 6 for instance). Other interesting non-convex abstract domains were introduced to capture specific invariants such as min-max invariants [2] and quadratic templates [1].

We address a different type of non-convexity commonly occurring in software, which relates to small sub-regions of instability within a normal operating (convex) region of interest. The non-convex region of values that may cause the bug is (under-)approximated using a convex inner region (or hole) that is subtracted from a convex outer region. We call this representation *donut domains*. Our approach relies on the usual operations defined on (convex) sub-domains, except for the need to compute under-approximations in domain $\mathcal{A}_2$. The donut domains can be considered extensions to the work on signed types domain introduced by the authors in [27]. There, we start with a finite set of types, and allow a set-minus operation only from the universal set.

Under-Approximations of Polyhedra. Under-approximations have been utilized for applications such as test vector generation and counterexample generation, by providing *must-reach sets*.

Bemporad et al. introduced the notion of inner approximations of polyhedra using intervals in [6]. In [23], polyhedra are under-approximated for test vector generation of Simulink/Stateflow models using *bounded vertex representation (BVR)*. Goubault and Putot describe a method to compute an under-approximating zonotope [18] using modal intervals [16] for non-linear operations.

In this work, we propose a novel technique to find under-approximations of polyhedra based on a fixed template. We first re-formulate the problem by introducing an auxiliary matrix. This matrix represents the fact that we are looking for an inner polyhedral object of a particular shape. Using this auxiliary matrix re-formulation, we can then use classical convex analysis techniques to characterize an under-approximations of polyhedra.

Motivating Example. Figure 1 highlights a code snippet taken from XTide [3]. The XTide package provides accurate tide and current predictions in a number of formats based on algorithms. Similar patterns may exist in controller-related software to avoid regions of controller or numerical instability.

After the step marked `initializations`, (dx, dy) could be any point in $\mathbb{R}^2$ except the origin $(0,0)$, this particular point is kept and propagated forward as a "hole". After the `if`-statement, the set of reachable values is: $(dy > dx \wedge dy > -dx) \vee (-dy > dx \wedge -dy > -dx)$. The above region is non-convex; therefore, any classical abstract domain will end up at this control point with $\top$ for both variables. Moreover, here, the interpretation of the strict inequality of the test is required to prove that $dx \neq 0$. The `else` case is even harder: in

[3] see www.flaterco.com/xtide

```
static void p_line16_primary (...) {
  double dx, dy, x, y, slope;
  ...                            /* initializations */
  if (dx == 0.0 && dy == 0.0)    /* full-zero-test  */
    return ;
  if (fabs(dy) > fabs(dx)) {     /* fabs-based test */
    slope = dx / dy;             /* division-by-dy  */
    ...
  } else {
    slope = dy / dx;             /* division-by-dx  */
    ...
  }}
```

Fig. 1. Motivating example from XTIDE.

addition to the non-convexity of the set of possible values, one needs to consider the `full-zero-test` together with the negation of $|dy| > |dx|$, to prove that the division by dy is safe. All known non-convex domains fail to prove the safety of such example. Typically, the powerset domain cannot encode the `full-zero-test` which is mandatory here (for the `else` branch).

Content. The rest of this paper is organized as follows: we define, Section 2, a new set of domains called donut domains. Section 3 proposes a novel method to compute polyhedral under-approximations for arbitrary linear templates. Finally, in Section 4, first experiments and promising results are discussed.

2 Donut Abstract Domains

In this section we introduce donut domains, and define the operation on donut domains based on operations in the component domains.

2.1 Lattice Structure

Let $(\mathcal{A}_1, \leq_1, \cup_1, \cap_1, \bot_1, \top_1, \gamma_1)$ and $(\mathcal{A}_2, \leq_2, \cup_2, \cap_2, \bot_2, \top_2, \gamma_2)$ denote two classical numerical abstract domains, where $\leq_\star$, $\cup_\star$,$\cap_\star$,$\bot_\star$,$\top_\star$,$\gamma_\star$ denote the partial order, the join and meet operations, the bottom and top elements and the concretization function of the classical abstract domain for $\star \in \{1, 2\}$, respectively.

In this work, we extend a given abstract domain with an under-approximation operator $\breve{\alpha}$, such that for any concrete object X, we have $\gamma \circ \breve{\alpha}(X) \subseteq X$. An abstract object $X^\sharp_{1\setminus 2}$ of the domain $\mathcal{A}_1 \setminus \mathcal{A}_2$ is defined by a pair of objects $(X^\sharp_1, X^\sharp_2)$, such that $X^\sharp_1 \in \mathcal{A}_1$ and $X^\sharp_2 \in \mathcal{A}_2$. The object $X^\sharp_{1\setminus 2}$ abstracts the set of possible values reached by the variables as follows:

– The object $X^\sharp_1 \in \mathcal{A}_1$ represents an over-approximation of the set of reachable values.

– The object $X_2^\sharp \in \mathcal{A}_2$ represents an under-approximation of the set of unreachable values (usually within $\gamma_1(X_1^\sharp)$).

The concretisation function is defined as follows.

$$\gamma_{1\setminus 2} \stackrel{\text{def}}{=} \gamma_{1\setminus 2}(X_1^\sharp, X_2^\sharp) \stackrel{\text{def}}{=} \gamma_1(X_1^\sharp) \setminus \gamma_2(X_2^\sharp) \ .$$

Figure 2 depicts a concretization of a typical donut object where the domain $\mathcal{A}_1$ is the Affine Sets domain [15] and $\mathcal{A}_2$ is the octagons domain.

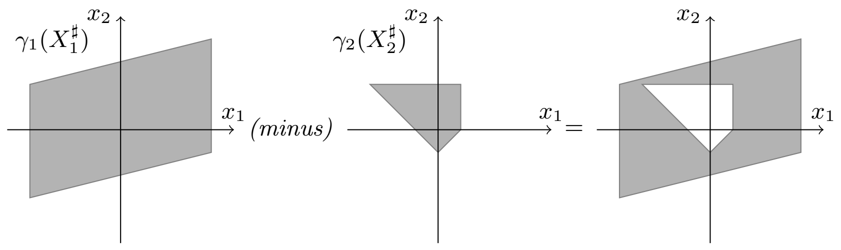

Fig. 2. The concretization of a typical non-convex abstract object

One should keep in mind the implicit set of unreachable values implied by $\gamma_1(X_1^\sharp)$ – namely $\mathbb{R}^p \setminus \gamma_1(X_1^\sharp)$ denoted in the sequel by $\bar{\gamma}_1(X_1^\sharp)$. Indeed, the set of unreachable values is actually $\bar{\gamma}_1(X_1^\sharp) \cup \gamma_2(X_2^\sharp)$. As said earlier, $\gamma_2(X_2^\sharp)$ is a (convex) under-approximation of the set of unreachable values. The fact that the intersection $\gamma_1(X_1^\sharp) \cap \gamma_2(X_2^\sharp)$ is not empty permits to encode a hole inside $\gamma_1(X_1^\sharp)$ (see Figure 2).

Interval Concretisation. The interval concretization of the variable x_k, $1 \leq k \leq p$, denoted by $[x_k]$, is defined by $\pi_k(\gamma_1(X_1^\sharp) \setminus \gamma_2(X_2^\sharp))$, where π_k denotes the orthogonal projection of a given set onto dimension k. Note that $[x_k] \supseteq \pi_k(\gamma_1(X_1^\sharp)) \setminus \pi_k(\gamma_2(X_2^\sharp))$. For instance in $([-2,2] \times [-2,2], [-1,1] \times [-\infty,+\infty])$, we have $[x_2] = [-2,2]$, whereas $[-2,2] \setminus [-\infty,+\infty] = \emptyset$.

We embed $\mathcal{A}_1 \setminus \mathcal{A}_2$ with a binary relation and prove that it is a pre-order.

Definition 1. *Given $X_1^\sharp, Y_1^\sharp \in \mathcal{A}_1$ and $X_2^\sharp, Y_2^\sharp \in \mathcal{A}_2$, we say that $(X_1^\sharp, X_2^\sharp)$ is less than or equal to $(Y_1^\sharp, Y_2^\sharp)$ denoted by $(X_1^\sharp, X_2^\sharp) \leq_{1\setminus 2} (Y_1^\sharp, Y_2^\sharp)$ if and only if $X_1^\sharp \leq_1 Y_1^\sharp$ and*

$$\bar{\gamma}_1(X_1^\sharp) \cup \gamma_2(X_2^\sharp) \supseteq \bar{\gamma}_1(Y_1^\sharp) \cup \gamma_2(Y_2^\sharp) \ . \tag{1}$$

Proposition 1. *The binary relation $\leq_{1\setminus 2}$ is a pre-order over $\mathcal{A}_1 \setminus \mathcal{A}_2$. It defines an equivalence relation $\sim$ defined by $(X_1^\sharp, X_2^\sharp) \leq_{1\setminus 2} (Y_1^\sharp, Y_2^\sharp)$ and $(Y_1^\sharp, Y_2^\sharp) \leq_{1\setminus 2} (X_1^\sharp, X_2^\sharp)$ and characterized by $X_1^\sharp = Y_1^\sharp$ ($X_1^\sharp \leq_1 Y_1^\sharp$ and $Y_1^\sharp \leq_1 X_1^\sharp$), $\gamma_2(X_2^\sharp) \subseteq \gamma_2(Y_2^\sharp) \cup \bar{\gamma}_1(Y_1^\sharp)$ and $\gamma_2(Y_2^\sharp) \subseteq \gamma_2(X_2^\sharp) \cup \bar{\gamma}_1(X_1^\sharp)$. We reuse the symbol $\leq_{1\setminus 2}$ to also denote the partial order quotiented by the equivalence relation $\sim$.*

With respect to $\leq_{1\setminus 2}$, we have

$$(\bot_1, \bot_2) \sim (\bot_1, \top_2) \leq_{1\setminus 2} (\top_1, \top_2) \leq_{1\setminus 2} (\top_1, \bot_2);$$

therefore, we define the bottom and top elements of $\mathcal{A}_1 \setminus \mathcal{A}_2$ by

$$\bot_{1\setminus 2} \stackrel{\text{def}}{=} (\bot_1, -) \qquad\qquad \top_{1\setminus 2} \stackrel{\text{def}}{=} (\top_1, \bot_2) \ .$$

2.2 Decidability of the Order

Despite the non-convexity of $\bar{\gamma}$, the equivalence class introduced in Proposition 1 suggests particular representatives of objects $(X_1^\sharp, X_2^\sharp)$ which are easily comparable. Indeed, $\bar{\gamma}$ is no longer involved when the concretization of the hole $X_2^\sharp$ is included in the concretization of $X_1^\sharp$. Observe moreover, that the definition of the order relation $\leq_{1\setminus 2}$ allows comparing two abstract objects having their holes in two different abstract domains, since only the concretization functions are involved in (1).

Proposition 2. *Let $(X_1^\sharp, X_2^\sharp)$ and $(Y_1^\sharp, Y_2^\sharp)$ be two elements of $\mathcal{A}_1 \setminus \mathcal{A}_2$ such that $\gamma_2(X_2^\sharp) \subseteq \gamma_1(X_1^\sharp)$, and $\gamma_2(Y_2^\sharp) \subseteq \gamma_1(Y_1^\sharp)$. Therefore, $(X_1^\sharp, X_2^\sharp) \leq_{1\setminus 2} (Y_1^\sharp, Y_2^\sharp)$ if and only if $X_1^\sharp \leq_1 Y_1^\sharp$ and $\gamma_1(X_1^\sharp) \cap \gamma_2(Y_2^\sharp) \subseteq \gamma_2(X_2^\sharp)$.*

The condition $\gamma_1(X_1^\sharp) \cap \gamma_2(Y_2^\sharp) \subseteq \gamma_2(X_2^\sharp)$, can be checked in the abstract world rather than in the concrete domain up to the use of an expressive enough domain for both $\mathcal{A}_2$ and $\mathcal{A}_1$: for instance a box and an octagon can be seen as special polyhedra and the meet operation of Polyhedra abstract domain can be used.

Let $X_1^{\mathcal{P}}$ denote the abstract representation in Polyhedra domain of the abstract object $X_1^\sharp$, that is $\alpha_{\mathcal{P}}(\gamma_1(X_1^\sharp))$. The general procedure to decide whether $(X_1^\sharp, X_2^\sharp)$ is less than or equal to $(Y_1^\sharp, Y_2^\sharp)$, is as follows:

1. First, we "upgrade" $X_2^\sharp$ and $Y_2^\sharp$ to Polyhedra domain, which gives $(X_1^\sharp, X_2^{\mathcal{P}})$ and $(Y_1^\sharp, Y_2^{\mathcal{P}})$.
2. Then, we derive our particular representatives, namely $(X_1^\sharp, X_1^{\mathcal{P}} \cap_{\mathcal{P}} X_2^{\mathcal{P}})$ for $(X_1^\sharp, X_2^{\mathcal{P}})$ and $(Y_1^\sharp, Y_1^{\mathcal{P}} \cap_{\mathcal{P}} Y_2^{\mathcal{P}})$ for $(Y_1^\sharp, Y_2^{\mathcal{P}})$ ($\cap_{\mathcal{P}}$ being the meet operation in Polyhedra domain).
3. Finally, we use Proposition 2 by checking for the inequalities $X_1^\sharp \leq_1 Y_1^\sharp$ and

$$X_1^{\mathcal{P}} \cap_{\mathcal{P}} Y_1^{\mathcal{P}} \cap_{\mathcal{P}} Y_2^{\mathcal{P}} \leq_{\mathcal{P}} X_1^{\mathcal{P}} \cap_{\mathcal{P}} X_2^{\mathcal{P}} \ .$$

2.3 Meet and join operations

We start with a simple example to clarify the intuition behind the formal definition given later.

Example 1. Consider a one-dimensional donut domain where $\mathcal{A}_1$ and $\mathcal{A}_2$ are Intervals domain. Assume we are interested in computing

$$([0,3],[1,2]) \cup ([1,6],[2,5]) \ .$$

The above join yields the following union of four intervals: $[0,1) \cup (2,3] \cup [1,2) \cup (5,6]$, which can be combined without loss of precision into $[0,2) \cup (2,3] \cup (5,6]$, or equivalently

$$[0,6] \setminus ([2] \cup (3,5]) \ .$$

What the example suggests is that when computing a join of two elements $(X_1^\sharp, X_2^\sharp)$ and $(Y_1^\sharp, Y_2^\sharp)$, we often end up with multiple (not necessarily convex nor connex) holes defined by $(\gamma_2(X_2^\sharp) \cup \bar{\gamma}_1(X_1^\sharp)) \cap (\gamma_2(Y_2^\sharp) \cup \bar{\gamma}_1(Y_1^\sharp))$, which gives by distributing the meet over the join:

$$(\gamma_2(X_2^\sharp) \cap \gamma_2(Y_2^\sharp)) \cup (\gamma_2(X_2^\sharp) \cap \bar{\gamma}_1(Y_1^\sharp)) \cup (\gamma_2(Y_2^\sharp) \cap \bar{\gamma}_1(X_1^\sharp)) \cup (\bar{\gamma}_1(X_1^\sharp) \cap \bar{\gamma}_1(Y_1^\sharp)) \ .$$

An under-approximation of the final element $\bar{\gamma}_1(X_1^\sharp) \cap \bar{\gamma}_1(Y_1^\sharp)$ is implicit since the over-approximation of reachable values is given by $X_1^\sharp \cup_1 Y_1^\sharp$. Thus, only the intersection of the first three sets will be considered (which is sound). In our example, $\bar{\gamma}([1,6]) = [-\infty,1) \cup (6,+\infty]$, and $\bar{\gamma}([0,3]) = [-\infty,0) \cup (3,+\infty]$, this gives $[1,2] \cap [2,5] = [2,2]$ and

$$\begin{aligned} &[1,2] \cap ([-\infty,1) \cup (6,+\infty]) = \emptyset \\ &[2,5] \cap ([-\infty,0) \cup (3,+\infty]) = (3,5] \ . \end{aligned}$$

As said earlier, the intersection $([-\infty,1) \cup (6,+\infty]) \cap ([-\infty,0) \cup (3,+\infty])$ is implicit since it is covered by $\bar{\gamma}_1([0,3] \cup [1,6])$.

We now formalize the join operator:

$$(X_1^\sharp, X_2^\sharp) \cup_{1\setminus 2} (Y_1^\sharp, Y_2^\sharp) \overset{\text{def}}{=} (X_1^\sharp \cup_1 Y_1^\sharp, (X_1^\sharp, X_2^\sharp) \breve{\cap} (Y_1^\sharp, Y_2^\sharp)),$$

where $\breve{\cap}$ is defined by:

$$(X_1^\sharp, X_2^\sharp) \breve{\cap} (Y_1^\sharp, Y_2^\sharp) \overset{\text{def}}{=} \breve{\alpha}((\gamma_2(X_2^\sharp) \cap \gamma_2(Y_2^\sharp)) \cup (\gamma_2(X_2^\sharp) \cap \bar{\gamma}_1(Y_1^\sharp)) \cup (\gamma_2(Y_2^\sharp) \cap \bar{\gamma}_1(X_1^\sharp))).$$

We may perform heuristic checks to prioritize which hole (if many) to keep, which may also depend on the under-approximation abstraction function $\breve{\alpha}$. For instance we may choose an inner approximation (if working with closed domains) of the hole $(3,5]$ instead of choosing the hole $[2,2]$.

Notice also that we have a straightforward fallback operator $\breve{\cap}_{\text{fb}}$, that involves only $X_2^\sharp$ and $Y_2^\sharp$:

$$X_2^\sharp \breve{\cap}_{\text{fb}} Y_2^\sharp \overset{\text{def}}{=} \breve{\alpha}(\gamma_2(X_2^\sharp) \cap \gamma_2(Y_2^\sharp)) \ .$$

The operator is sound with respect to under-approximation. It focuses only on a particular hole, namely $\gamma_2(X_2^\sharp) \cap \gamma_2(Y_2^\sharp)$, instead of considering all possibilities.

In our current implementation, we use this fallback operator in a smart manner: before computing the meet of both holes, we relax, whenever possible, in a convex way, these holes. This relaxation is performed by removing all constraints that could be removed while preserving $\gamma_1(X_1^\sharp)$. For instance, if the hole is the point $(0,0)$, and the abstraction of $X_1^\sharp$ is given by the conjunction $y \geq x \wedge -y \geq x$, then the hole $(0,0)$ is relaxed to $x \geq 0$ (see Figure 3).

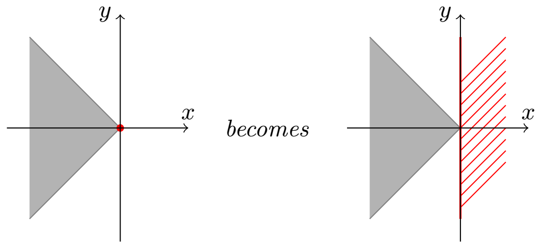

Fig. 3. Relaxing the hole $(0,0)$ (red circle in the left hand side figure) to $x \geq 0$

For the meet operation, we proceed in a similar manner. If the domain $\mathcal{A}_2$ is closed under the meet operation (almost all polyhedra-like abstract domains), it is possible to replace $\breve{\alpha}$ by α, and $\breve{\cap}_{\mathrm{fb}}$ by $\cap_2$. In our example, the fallback operator gives the box $[2,2]$.

The meet operator $\cap_{1\backslash 2}$ is defined in a similar manner:

$$(X_1^\sharp, X_2^\sharp) \cap_{1\backslash 2} (Y_1^\sharp, Y_2^\sharp) \overset{\mathrm{def}}{=} (X_1^\sharp \cap_1 Y_1^\sharp, X_2^\sharp \breve{\cup} Y_2^\sharp)$$
$$\text{where } X_2^\sharp \breve{\cup} Y_2^\sharp \overset{\mathrm{def}}{=} \breve{\alpha}_2(\gamma_2(X_2^\sharp) \cup \gamma_2(Y_2^\sharp)) \ .$$

We deliberately omit $\bar{\gamma}_1(X_1^\sharp) \cup \bar{\gamma}_1(Y_1^\sharp)$ in the above definition of $\breve{\cup}$ because it is implicit from $X_1^\sharp \cap_1 Y_1^\sharp$. If the domain $\mathcal{A}_2$ is closed under the join operation, then $\breve{\cup}$ is exactly equal to $\cup_2$. Very often, however, the join operation leads to an over-approximation. Therefore the detection of an exact join as in [7, 5] is of particular interest. In our current implementation, if $X_2^\sharp$ and $Y_2^\sharp$ overlap, we soundly extend, in a convex way, the non empty intersection. For instance, if $X_2^\sharp = [-2,1] \times [-1,1]$ and $Y_2^\sharp = [-1,2] \times [-2,0]$, the intersection gives the box $[-1,1] \times [-1,0]$, and the extension we compute gives the box $[-2,2] \times [-1,0]$. If, however, the holes are disjoint, we randomly pick up one of them.

Example 2. Consider a 2-dim simple abstract objects. Figure 4 shows a graphical representation of two overlapping objects. The remaining sub-figures highlight some of the pertinent steps with respect to the computation of $\cup_{1\backslash 2}$ and $\cap_{1\backslash 2}$ for such overlapping objects.

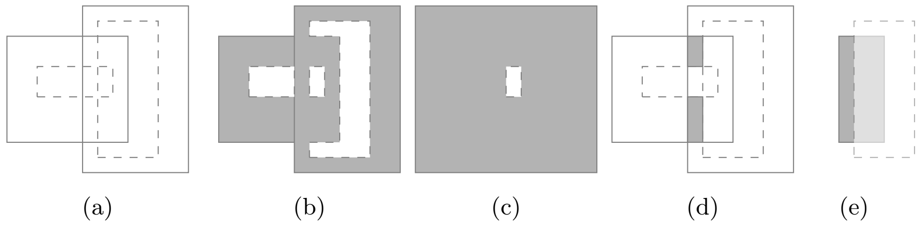
Fig. 4. Illustrating the join and meet operators using interval component domains. The donut holes are highlighted using dashed lines. (a) Two initial abstract objects. (b) The concrete union of the objects. (c) The abstract object representing $\cup_{1\backslash 2}$. (d) The concrete intersection of the objects. (e) The abstract object representing $\cap_{1\backslash 2}$.

2.4 Loop widening

When processing loop elements in abstract interpretation, we may require widening to guarantee termination of the analysis. For donut domains, we extend the widening operations defined on the component abstract domains. We use the pair-wise definition of widening operators ∇. We thus define widening of donut domains as:

$$(X_1^\sharp, X_2^\sharp)\nabla_{1\backslash 2}(Y_1^\sharp, Y_2^\sharp) = (X_1^\sharp \nabla_1 Y_1^\sharp, X_2^\sharp \cap_2 Y_2^\sharp) \ .$$

We use the standard widening operator ∇_1 for abstract domain $\mathcal{A}_1$. Similarly, we use standard meet operator $\cap_2$ of abstract domain $\mathcal{A}_2$ for the inner region, which ensures the soundness of $\nabla_{1\backslash 2}$. The convergence of the first component is guaranteed by the widening operator ∇_1. The convergence of the second component needs however more attention. Note that the simple use of narrowing operator of $\mathcal{A}_2$ is unsound as it may give a donut object which is not an upper bound. To ensure the termination we add a parameter k which will encode the maximal number of allowed iterations. If the donut object does not converge within those k iterations, the hole component is reduced to $\bot_2$. Note that the use of the narrowing operator of $\mathcal{A}_2$ instead of $\cap_2$ does not give in general an upper bound of $(X_1^\sharp, X_2^\sharp)$ and $(Y_1^\sharp, Y_2^\sharp)$.

2.5 Interpretation of Tests

The ability to express holes allows us to better handle a wide range of non-convex tests such as the $\neq$ test or the strict inequality test. We start with classical tests. For $\diamond \in \{=, \leq\}$:

$$[\![x_k \diamond 0]\!]^\sharp(X_1^\sharp, X_2^\sharp) \overset{\text{def}}{=} ([\![x_k \diamond 0]\!]_1^\sharp(X_1^\sharp), [\![x_k \diamond 0]\!]_1^\natural(X_2^\sharp)),$$

where $[\![\cdot]\!]_2^\natural \overset{\text{def}}{=} \breve{\alpha}_2 \circ [\![\cdot]\!]_2$. Such under-approximation is required so that the newly computed (exact) hole can be encoded in $\mathcal{A}_2$. Therefore, if the exact hole fits naturally in $\mathcal{A}_2$ (say we have a linear constraint and $\mathcal{A}_2$ is Polyhedra domain), there is no need to under-approximate ($[\![\cdot]\!]_2^\natural = [\![\cdot]\!]_2^\sharp$). In Section 3, we detail how we compute such an under-approximation, whenever needed. If no algorithm is available for the under-approximation, we keep the object $X_2^\sharp$ unchanged, which is sound.

The non-equality test $\neq$ is defined as follows:

$$[\![x_k \neq 0]\!]^\sharp(X_1^\sharp, X_2^\sharp) \overset{\text{def}}{=} ([\![x_k \neq 0]\!]^\sharp(X_1^\sharp), \breve{\alpha}(\gamma_2(X_2^\sharp) \cup [\![x_k = 0]\!]\top_2)) \enspace .$$

Although $[\![x_k \neq 0]\!]^\sharp(X_1^\sharp)$ is interpreted as the identity function in standard implementations, nothing prevents the use of any available enhancement proposed by the used analyzer. For the hole, we compute the join of the new hole implied by the constraint $x_k \neq 0$ together with the already existing hole $X_2^\sharp$. If holes $\gamma_2(X_2^\sharp)$ and $[\![x_k = 0]\!]\top_2$ do not overlap, we discard $X_2^\sharp$. In fact, very often (as will be seen in experiments), the hole induced by the constraint $x_k \neq 0$ is mandatory in order to prove the safety of subsequent computations.

Finally, our approach offers, for free, an interesting abstraction of the strict inequality tests. A comparison with Not Necessarily Closed domains [3] is planned as future work.

$$[\![x_k < 0]\!]^\sharp(X_1^\sharp, X_2^\sharp) \overset{\text{def}}{=} [\![x_k \neq 0]\!]^\sharp \circ [\![x_k \leq 0]\!]^\sharp(X_1^\sharp, X_2^\sharp) \enspace .$$

2.6 Abstract Assignment

We define in this section the abstraction of the assignment transfer function in $\mathcal{A}_1 \setminus \mathcal{A}_2$. We first give an abstraction of the forget transfer function (non-deterministic assignment) :

$$\begin{aligned}
&[\![x_k \leftarrow ?]\!]_{1\setminus 2}^\sharp(X_1^\sharp, X_2^\sharp) \overset{\text{def}}{=} (Y_1^\sharp, Y_2^\sharp),\\
&\quad \text{where } Y_1^\sharp \overset{\text{def}}{=} [\![x_k \leftarrow ?]\!]_1^\sharp(X_1^\sharp)\\
&\qquad\qquad Y_2^\sharp \overset{\text{def}}{=} \begin{cases} [\![x_k \leftarrow ?]\!]_2^\natural(X_2^\sharp) \ \textbf{if}\ \gamma_1(X_1^\sharp) \cap \gamma_2([\![x_k \leftarrow ?]\!]_2^\natural(X_2^\sharp)) \subseteq \gamma_2(X_2^\sharp) \\ \bot_2 \qquad\qquad\qquad\ \textbf{otherwise} \enspace . \end{cases}
\end{aligned}$$

For $Y_2^\sharp$, we basically check whether applying the forget operator to $X_2^\sharp$ intersects $\gamma_{1\setminus 2}(X_1^\sharp, X_2^\sharp)$, by checking if this newly computed hole is included on the original hole, that is $\gamma_2(X_2^\sharp)$. If yes, $Y_2^\sharp$ is set to $\bot_2$. For instance, forgetting x_2 in $(X_1^\sharp, X_2^\sharp) \overset{\text{def}}{=} ([-2,2] \times [-2,2], [-1,1] \times [-\infty,+\infty])$ gives $([-2,2] \times [-\infty,+\infty], [-1,1] \times [-\infty,+\infty])$: since $[\![x_2 \leftarrow ?]\!]_2^\natural(X_2^\sharp) = [-1,1] \times [-\infty,+\infty]$, $\gamma_1(X_1^\sharp) \cap \gamma_2([\![x_2 \leftarrow ?]\!]_2^\natural(X_2^\sharp)) = [-1,1] \times [-2,2]$ which is included in $\gamma_2(X_2^\sharp)$. Forgetting x_1, however, makes $Y_2^\sharp = \bot_2$.

The assignment could be seen as a sequence of multiple basic, already defined, operations. We distinguish two kind of assignments $x \leftarrow e$, where e is an arithmetic expression: ($\imath$) non-invertible assignments, where the old values of x are lost, such as $x \leftarrow c$, $c \in \mathbb{R}$, and ($\imath\imath$) invertible assignments, such as $x \leftarrow x + y$. For non-invertible assignment, we have:

$$[\![x_k \leftarrow e]\!]_{1\setminus 2}^\sharp \overset{\text{def}}{=} [\![x_k = e]\!]_{1\setminus 2}^\sharp \circ [\![x_k \leftarrow ?]\!]_{1\setminus 2}^\sharp \enspace .$$

Invertible assignments are defined in a similar manner. It augments first the set of variables by a new fresh variable, say v, then enforces the test $v = e$, and finally forgets x and (syntactically) renames v to x. Notice that augmenting the set of variables in $\mathcal{A}_1 \setminus \mathcal{A}_2$ makes the newly added variable, v, unconstrained in both components, $X_1^\sharp$ and $X_2^\sharp$. We can suppose that such a variable v already exists, and used whenever we have an invertible assignment; hence, we obtain:

$$[\![x_k \leftarrow e]\!]^\sharp_{1\setminus 2} \overset{\text{def}}{=} \mathbf{swap}(x_k, v) \ \mathbf{in} \ [\![x_k \leftarrow ?]\!]^\sharp_{1\setminus 2} \circ [\![v = e]\!]^\sharp_{1\setminus 2} \ .$$

3 Template-based Under-Approximations of Polyhedra

In this section we develop a new technique to under-approximate holes obtained after linear tests. Holes obtained after non-linear tests are so far reduced to $\bot_2$, which is sound. We plan to improve this as a future work. Consider for instance the object $([-2, 3] \times [-2, 2], [-1, 1] \times [0, 1])$. Figure 5 depicts the exact evaluation of a linear assignment. If we use boxes to encode holes, we need to compute a box inside the white polytope. In Figure 6, an under-approximation is needed for all convex domains, whereas a non-convex domain such as Interval Polyhedra [9] can express exactly this kind of pattern.

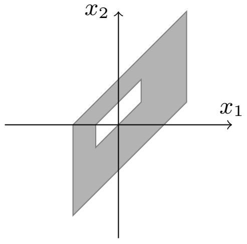

Fig. 5. Evaluation of a linear expression $[\![x_2 \leftarrow x_1 + x_2]\!]^\sharp_{1\setminus 2}$

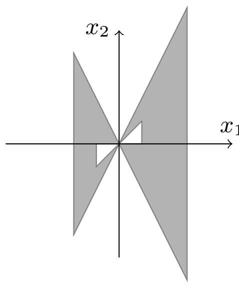

Fig. 6. Evaluation of a non-linear expression $[\![x_2 \leftarrow x_1 \times x_2]\!]^\sharp_{1\setminus 2}$

The problem can be seen as follows: given a polyhedron $\mathcal{P}$, we seek to compute a maximal (in a sense to define) inner polyhedron $\mathcal{T}$ (could be boxes, zones, octagons, linear-templates, etc. depending on $\mathcal{A}_2$), which obeys the template pattern matrix T.

Let $\mathcal{P} = \{x \in \mathbb{R}^p | Ax \leq b\}$ be a non-empty polyhedron, where A is a known $m \times p$ matrix, b a known vector of $\mathbb{R}^m$, and x a vector of $\mathbb{R}^p$. The inner polyhedron $\mathcal{T}$ is expressed in a similar manner: $\mathcal{T} = \{x \in \mathbb{R}^p | Tx \leq c\}$, where T is a known $n \times p$ matrix, and c and x are unknown vectors within $\mathbb{R}^n$ and $\mathbb{R}^p$, respectively. The inclusion $\mathcal{T} \subseteq \mathcal{P}$ holds if and only if

$$\exists c \in \mathbb{R}^n, \text{ such that } \mathcal{T} \text{ is consistent, and } \forall x \in \mathbb{R}^p : Tx \leq c \implies Ax \leq b \ .$$

The consistency of $\mathcal{T}$ (that is the system admits a solution in $\mathbb{R}^p$) discards the trivial (and unwanted) cases where the polyhedron $\mathcal{T}$ is empty. For the non trivial cases, the existence of the vector c and the characterization of the set of its possible values are given by Proposition 3.

Proposition 3. *Let C be the set of c such that $\mathcal{T}$ is consistent. There exists a vector $c \in C$ such that $\mathcal{T} \subseteq \mathcal{P}$ if and only if there exists an $n \times m$ matrix Λ, such that $\lambda_{i,j}$, the elements of the matrix Λ, are non-negative and $\Lambda T = A$. For a given possible Λ, the set $c_\Lambda \subseteq C$ is characterized by*

$$\{c \in \mathbb{R}^n \mid \Lambda c \leq b\} \ .$$

Proof. Let x denote a vector of $\mathbb{R}^p$, and b denote a known vector of $\mathbb{R}^m$. Let A and T be two known matrices with p columns and m and n lines, respectively. Suppose that c is such that $\mathcal{T}$ is consistent. Therefore, we can assume that

$$\langle t_i, x\rangle \leq c_i, 1 \leq i \leq n,$$

where t_i denotes the ith line of the matrix T, is consistent. For a fixed j, $1 \leq j \leq m$, the inequality $\langle a_j, x\rangle \leq b_j$, is then a consequence of the system $Tx \leq c$ if and only if there exist non-negative real numbers $\lambda_{i,j}$, $1 \leq i \leq n$, such that

$$\sum_{i=1}^{n} \lambda_{i,j} t_i = a_j \text{ and } \sum_{i=1}^{n} \lambda_{i,j} c_i \leq b_j \ .$$

The previous claim of the existence of the non-negative $\lambda_{i,j}$ is a generalization of the classical Farkas' Lemma (see for instance [28, Section 22, Theorem 22.3] for a detailed proof). The matrix Λ is then constructed column by column using the elements $\lambda_{i,j}$, $1 \leq i \leq n$ for the jth column. Of course, by construction, such a Λ has non-negative elements, and satisfies $\Lambda T = A$, and $\Lambda c \leq b$.

On the other hand, if such a matrix exists, and the set $\{c \in \mathbb{R}^n \mid \Lambda c \leq b\}$ is not empty, we have by the fact that Λ has non-negative elements

$$Tx \leq c \implies \Lambda T x \leq \Lambda c \ .$$

Therefore, $\Lambda T = A$ and $\Lambda c \leq b$, gives $Ax \leq b$. □

On the Consistency of $Tx \leq c$. It not obvious in general, given a matrix T, to characterize the set of c such that $\mathcal{T}$ is consistent. However, given a vector c, we can efficiently check whether the system is consistent or not using its dual form and a LP solver. Indeed, the system $Tx \leq c$ is inconsistent if and only if there exists a non negative vector $\lambda \in \mathbb{R}^n$ such that $T^t\lambda = 0$ and $\langle \lambda, c\rangle < 0$, where T^t denotes the transpose of T. Therefore, given a vector c, if the objective value of the following problem

$$\begin{aligned} \min \quad & \langle \lambda, c\rangle \\ \text{s.t.} \quad & T^t\lambda = 0 \ . \end{aligned} \tag{2}$$

is non negative, the system is consistent. Observe that, for simple patterns such as boxes, the characterization of the set of c that makes the system consistent is immediate.

Computing Λ. The matrix Λ is built column by column. Let us denote by $\lambda_{-,j} \in \mathbb{R}^n$ the jth column of Λ, by $a_j \in \mathbb{R}^p$, $1 \leq j \leq m$, the jth line of A, by $b_j \in \mathbb{R}$ the jth component of b, and by $t_i \in \mathbb{R}^p$, $1 \leq i \leq m$, the ith line of T. The vector $\lambda_{-,j}$ satisfies $\sum_{i=1}^n \lambda_{i,j} t_i = a_j$. To each feasible $\lambda_{-,j}$ corresponds a pattern

$$\mathcal{P}_{\lambda_{-,j}} \overset{\text{def}}{=} \{x \in \mathbb{R}^p \mid \bigwedge_{\lambda_{i,j}>0} \langle t_i, x\rangle \leq 0\},$$

which is included in the affine subspace $\mathcal{P}_j \overset{\text{def}}{=} \{x \in \mathbb{R}^p \mid \langle a_j, x\rangle \leq 0\}$. The maximal pattern (with respect to set inclusion) corresponds to $\bar{\lambda}$ defined as the solution of the following linear program.

$$\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{n} \lambda_{i,j} \|t_i\| \\ \text{s.t.} & \begin{array}{l} \sum_{i=1}^n \lambda_{i,j} t_i = a_j \\ \forall 0 \leq i \leq n, \lambda_{i,j} \geq 0 \end{array} . \end{array} \tag{3}$$

Therefore, computing Λ needs solving p instances of the LP (3).

Computing c. We have already established (Proposition 3) that the vector c verifies $\Lambda c \leq b$. Since Λ is known, any feasible c (that is such that $\Lambda c \leq b$) that makes the system $Tx \leq c$ consistent (the objective value of the LP (2) is non negative) gives an under-approximation of $\mathcal{P}$ that respects our initial template T. Of course, it is immediate to see that the set of c that lies on the boundaries of the feasible region (that is by making $\Lambda c = b$) gives, in general, a "better" under-approximation than the strict feasible solutions since the saturation makes some of the facets of the inner pattern ($\mathcal{T}$) included in those of the under-approximated polyhedron $\mathcal{P}$. Moreover, in some cases, the saturation gives a unique consistent solution for c. For instance, when we under-approximate a shape $\mathcal{P}$ which respects already the pattern $\mathcal{T}$, c is uniquely determined and gives actually b using our technique. In other words, under-approximating an octagon (for instance) with an octagonal pattern gives exactly the first octagon.

4 Implementation

We have implemented donut domains on the top of Apron library [22]. The domains $\mathcal{A}_1$ and $\mathcal{A}_2$ are parameters of the analysis and can be specified by the user among already existing Apron domains. The current version uses an enhanced implementation of the set-theoretic operators, mainly based on already existing routines of the underlying abstract domains, as described earlier and relies on $\breve{\cup}_{fb}$ and $\breve{\cap}_{fb}$ as fallback operators. This very simple approach allows to build the donut domain without an additional effort on the top of already existing domains. As Table 1 shows, our approach permits to catch almost all division-by-zero false positives that classical domains (even non-convex) fail to prove. The WCfS column indicates the weakest condition that we need to infer to prove the

	WCfS	boxes (hole)	false alarms
`motiv`(if)	$dy \neq 0$	$dy = 0$	0
`motiv`(else)	$dx \neq 0$	$dx = 0$	0
`gpc`	$den \neq 0$	$den \in [-0.1, 0.1]$	0
`goc`	$d \neq 0$	$d \in [-0.09, 0.09]$	0
`x2`	$Dx \neq 0$	$Dx = 0$	0
`xcor`	$usemax \neq 0$	$usemax \in [1, 10]$	1

Table 1. Division-by-zero analysis results

safety of the program. Whenever the negation of this condition is verified by (included in) the donut hole, the program is proved to be safe. The third column shows the inferred donut holes when using a non-relational domain (boxes) to encode holes. The analyzed examples [4] use mainly the absolute value function to avoid the division by zero (widely used technique).The `motiv` example is the motivating example with its two branches. The `gpc` code is extracted from the Generic Polygon Clipper project. The examples `xcor`, `goc` and `x2` are extracted from a geometric object contact detection library. Observe that the use of boxes is sufficient to eliminate almost all false alarms here. In the last example, among the two possible holes, namely $usemax \in [1, 10]$ and $usemax \in \{0\}$, we choose by default the one created immediately after the test ($usemax > 10$ or $usemax < 1$). Here the safety property can not be proved with this hole and relies on an earlier (disjoint) hole created by a former test, namely $usemax \in \{0\}$. We could also choose systematically (as a heuristic) the hole that contains "zero", which is sufficient here to discard the remaining false alarm. Such a property-driven hole behaviour would be an interesting direction for future research.

The proof of the motivating example is really challenging as it requires to handle both the hole that comes from the full-zero-test, together with strict inequalities tests and the over-approximation that comes from the join operation. Our technique that consists of relaxing the hole in a convex way before using the fallback operator works here and is able to prove that in both branches the division is safe. In `goc` example, we can see one interesting ability of donuts domain: when we compute a convex join of two non-overlapping objects, the hole in between is directly captured which permits a better precision. Finally, example `x2` needs a precise interpretation of strict inequalities.

Under-Approximation. We have implemented our technique of Section 3 using GLPK [24] solver. Some experiments, obtained for randomly generated polyhedra with octagonal template, are shown in Figure 7. Although all shown polyhedra are bounded, our technique works perfectly well for unbounded shapes. The rate of volume, $\frac{vol\mathcal{T}}{vol\mathcal{P}}$, is used as a metric for the quality of the under-approximation (shown near each pattern in Figure 7). All obtained octagons are maximal with respect to set inclusion. It is not clear which choice among many

[4] www.nec-labs.com/research/system/systems_SAV-website/benchmarks.php. The C files are the real source code, while the SPL files extracts the hard piece of code that leads to false alarms, and with which we feed our proof of concept implementation.

(see the left graph), is the best. Indeed, such a choice depends on the future computations and the properties one would like to prove.

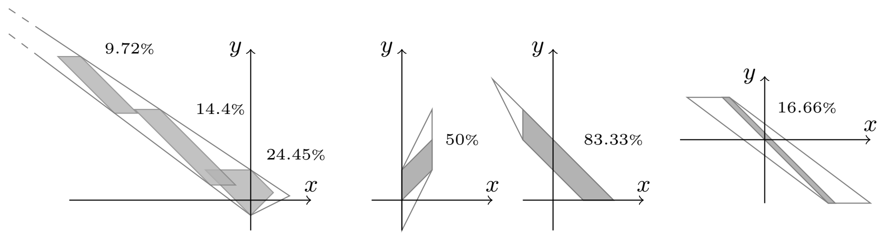

Fig. 7. Under-approximation of randomly generated polyhedra with octagons

5 Conclusions and Future Work

The donut domains can be viewed as an effort to make some Boolean structure in the underlying concrete space visible at the level of abstract domains as a "set-minus" operator. This allows optimization of the related abstract operators (such as meet and join) to take full advantage of its semantics in terms of excluded states. While the powerset domains allow an arbitrary Boolean combination, and automata-based abstractions [13] using may- and must-transitions offer full generality in verification, this comes at significant cost. In practice, the full expressiveness may not be needed. We exploit the set-minus operator, which is quite versatile in capturing many problems of interest - division by zero, instability regions in numeric computations, sets excluded by contracts in a modular setting, etc.

In the future, we wish to expand the experiments performed using donut domains. Furthermore, other non-convexity issues may be addressed by trying to combine the work on LDDs with insights gained here to allow handling many holes in an efficient manner.

Acknowledgment. The authors would like to thank Sriram Sankaranarayanan, Enea Zaffanella and anonymous reviewers for their valuable comments on an earlier draft of this work.

References

1. A. Adjé, S. Gaubert, and E. Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In *ESOP*, pages 23–42, 2010.
2. X. Allamigeon, S. Gaubert, and E. Goubault. Inferring Min and Max Invariants Using Max-plus Polyhedra. In *SAS'08*, LNCS, pages 189–204, 2008.
3. R. Bagnara, P. M. Hill, and E. Zaffanella. Not necessarily closed convex polyhedra and the double description method. *Form. Asp. Comput.*, pages 222–257, 2005.
4. R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *STTT*, 8(4-5):449–466, 2006.

5. R. Bagnara, P. M. Hill, and E. Zaffanella. Exact join detection for convex polyhedra and other numerical abstractions. *Comput. Geom.*, 43(5):453–473, 2010.
6. A. Bemporad, C. Filippi, and F. D. Torrisi. Inner and outer approximations of polytopes using boxes. *Comput. Geom.*, 27(2):151–178, 2004.
7. A. Bemporad, K. Fukuda, and F. D. Torrisi. Convexity recognition of the union of polyhedra. *Comput. Geom.*, 18(3):141–154, 2001.
8. S. Chaki, A. Gurfinkel, and O. Strichman. Decision diagrams for linear arithmetic. In *FMCAD*, pages 53–60. IEEE, 2009.
9. L. Chen, A. Miné, J. Wang, and P. Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In *SAS'09*, LNCS, pages 309–325, 2009.
10. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In 2^{nd} *Intl. Symp. on Programming*, pages 106–130. Dunod, France, 1976.
11. P. Cousot and R. Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM, 1977.
12. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *POPL*, pages 84–97. ACM, Jan. 1978.
13. D. Dams and K. S. Namjoshi. Automata as abstractions. In *VMCAI*, pages 216–232, 2005.
14. K. Ghorbal, E. Goubault, and S. Putot. The zonotope abstract domain taylor1+. In *CAV*, volume 5643 of *LNCS*, pages 627–633. Springer, 2009.
15. K. Ghorbal, E. Goubault, and S. Putot. A logical product approach to zonotope intersection. In *CAV'10*, LNCS, Edinburgh, UK, 2010. Springer.
16. A. Goldsztejn, D. Daney, M. Rueher, and P. Taillibert. Modal intervals revisited: a mean-value extension to generalized intervals. In *QCP*, 2005.
17. E. Goubault and S. Putot. Static analysis of numerical algorithms. In *SAS*, volume 4134 of *LNCS*, pages 18–34. Springer, 2006.
18. E. Goubault and S. Putot. Under-approximations of computations in real numbers based on generalized affine arithmetic. In *SAS*, pages 137–152, 2007.
19. P. Granger. Static analysis of linear congruence equalities among variables of a program. In *TAPSOFT, Vol.1*, pages 169–192, 1991.
20. A. Gurfinkel and S. Chaki. Boxes: A symbolic abstract domain of boxes. In *SAS*, volume 6337 of *LNCS*, pages 287–303. Springer, 2010.
21. N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *SAS*, pages 223–237, 1994.
22. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
23. A. Kanade, R. Alur, F. Ivančić, S. Ramesh, S. Sankaranarayanan, and K. Shashidhar. Generating and analyzing symbolic traces of Simulink/Stateflow models. In *CAV*, volume 5643 of *LNCS*, pages 430–445. Springer, 2009.
24. A. Makhorin. The GNU Linear Programming Kit (GLPK), 2000. Available online from http://www.gnu.org/software/glpk/glpk.html.
25. F. Masdupuy. Array abstractions using semantic analysis of trapezoid congruences. In *ICS*, pages 226–235, 1992.
26. A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001.
27. P. Prabhu, N. Maeda, G. Balakrishnan, F. Ivančić, and A. Gupta. Interprocedural exception analysis for C++. In *ECOOP'11*, 2011.
28. R. T. Rockafellar. *Convex Analysis.* Princeton University Press, 1970.