Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 267 (2010) 59–72
www.elsevier.com/locate/entcs

Relational Abstract Domain of Weighted Hexagons

Jędrzej Fulara[1,2], Konrad Durnoga[3], Krzysztof Jakubczyk[1,4] and Aleksy Schubert[1,5]

Institute of Informatics
University of Warsaw
ul. Banacha 2
02-097 Warsaw, Poland

Abstract

We propose a new numerical abstract domain for static analysis by abstract interpretation, the domain of Weighted Hexagons. It is capable of expressing interval constraints and relational invariants of the form $x \leq a \cdot y$, where x and y are variables and a denotes a non-negative constant. This kind of domain is useful in analysis of safety for array accesses when multiplication is used (e.g. in guarding formulæ or in access expressions). We provide all standard abstract domain operations, including widening operator, as well as a graph-based algorithm for checking satisfiability and computing normal form for elements of the domain. All described operations are performed in $O(n^3)$ time. Expressiveness of this domain lies between the Pentagons by Logozzo and Fähndrich and the Two Variables Per Inequality by Simon, King and Howe.

Keywords: Numerical abstract domains, static analysis, abstract interpretation.

1 Introduction

The concrete semantics of a program yields possibly infinite computations, hence answering any non trivial questions may be infeasible. A simpler, yet not fully precise, model can be employed to reason about program properties. Abstract Interpretation [6] is a widely used technique that simplifies the process of computation so that its vital properties can be captured within finite resources (time, space etc.) of a computing machine. This technique has been successfully applied in various fields, including program verification [2], error discovery and debugging [3], specification

[1] This work was partly supported by Polish government grant N N206 493138.
[2] Email: fulara@mimuw.edu.pl
[3] Email: kdr@mimuw.edu.pl
[4] Email: kjk@mimuw.edu.pl
[5] Email: alx@mimuw.edu.pl

doi:10.1016/j.entcs.2010.09.006

```
Require: array input[1...m]
1: {input.len ≤ m}
2: output := array [1...2 * input.len]
3: {output.len ≤ 2 · input.len; ... ; m ≤ 1/2 · output.len}
4: for i := 1 to m do
5:    {i ≤ 1/2 · output.len; ...}
6:    output[2 * i - 1] := input[i]
7:    output[2 * i] := input[i]
8: end for
9: return output
```

Fig. 1. A simple code fragment along with invariants that allow proving correctness of the array accesses in lines 6 and 7.

generation or code optimisation during compilation, including program transformation [5].

The Abstract Interpretation Framework allows one to automatically infer invariants that describe some properties of the analysed program. To apply this framework we need to define an *abstract domain* — a representation of the invariants, and operations on them (union, intersection, widening, satisfiability test, etc.) [8]. In this paper, we present an abstract domain that can express relations between pairs of numerical variables x, y of the form $x \leq a \cdot y$, where a is a non-negative constant, as well as interval constraints $x \in [b, c]$. In a two-dimensional case such constraints describe a polygon with at most 6 edges and angles determined by the coefficients from the inequalities (Figure 2). This motivates the name *Weighted Hexagons*.

Example

Figure 1 presents a simple procedure that duplicates all entries of the given *input* array. Static analysis using the domain of Weighted Hexagons can automatically infer in line 5 an invariant $i \leq \frac{1}{2} \cdot output.len$.

1.1 Related Work

In the past, several numerical abstract domains were developed. The most basic one is the Domain of Intervals [7] that represents invariants of the form $x \in [a, b]$. It is efficient (linear time and memory), but does not handle relations between variables. The domain of Convex Polyhedra [10] is very precise — it represents invariants of the form $\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n \leq c$ where $v_1, \ldots, v_n$ are program variables and $c, \alpha_1, \ldots, \alpha_n$ are numerical constants. However the domain representation and operations are inefficient (exponential in the number of variables), and so not practical for many applications. The domain of Octagons [1,12] stores constraints $\pm x \pm y \leq c$ where x and y are program variables and c is a constant. All domain operations can be performed in $O(n^3)$ time using $O(n^2)$ of memory, where n denotes the number of variables. Using the domain of Pentagons [11] one can represent numerical intervals together with symbolic inequalities between variables of the form $x < y$. The complexity of domain operations is $O(n^2)$, but the authors do not present a satisfiability test (if a domain element describes a system of constraints with empty

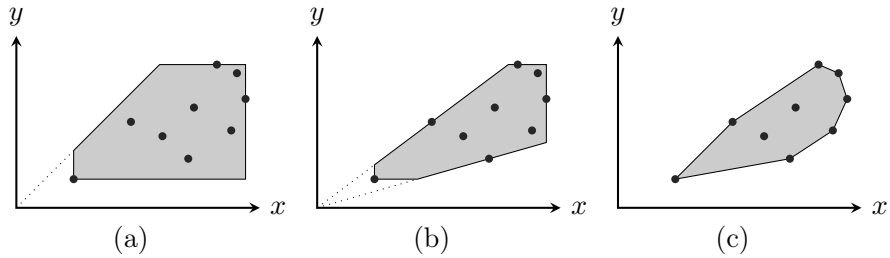


Fig. 2. Pentagon (a), Weighted Hexagon (b), and TVPI (c) for the same set of concrete points.

set of solutions). The Two Variables Per Inequality (TVPI) abstract domain [13] is more powerful (it makes possible to represent $ax+by \leq c$), but is also less efficient — the worst case time complexity is $O(k^2n^3 \log n(\log n + \log k))$, where n denotes the number of variables and k can be as large as the maximal number of inequalities between a pair of variables (which, in case of TVPI, can be arbitrarily large). A comparison of Pentagons, TVPI and Weighted Hexagons can be found in Figure 2.

1.2 Paper Outline

Section 2 describes the representation of elements of the abstract domain and defines the join and meet operators and introduces a lattice structure using these operators. Section 3 presents an algorithm for checking satisfiability and computing a normal form of a given element of our domain. In Section 4 we provide a widening operator. The transfer function for the domain is discussed in Section 5. We conclude in Section 6.

2 Problem Definition

We are concerned with determining whether a system of constraints $x \leq a \cdot y$ (where x, y are variables and a is some non-negative constant) is satisfiable or not. More formally: we are given a finite set of variables Var and a set $\mathbb{I}$ chosen to be the field of reals $\mathbb{R}$ or rationals $\mathbb{Q}$ with their natural order.

We consider a finite system $\mathcal{I}$ of inequalities of a form $x \leq a \cdot y$ such that $x, y \in \mathsf{Var}$ and $a \in \mathbb{I}_{\geq 0}$, where by $\mathbb{I}_{\geq 0}$ we mean $\{a' \in \mathbb{I} \mid 0 \leq a'\}$. Note that we put an important limit on a, that is we only allow non-negative coefficients. This makes possible to maintain a nice geometrical interpretation of the constraints. When we ask for satisfiability of such a system $\mathcal{I}$, we in fact inquire whether there exists some valuation $\rho\colon \mathsf{Var} \to \mathbb{I}$ assigning numerical values to variables such that all constraints $\rho(x) \leq a \cdot \rho(y)$ hold over $\mathbb{I}$.

Additionally we want to restrict values of variables to certain intervals, that is to model interval restraints such as $x \in [b, c]$ where $b \in \mathbb{I} \cup \{-\infty\}$ and $c \in \mathbb{I} \cup \{+\infty\}$ (with a premise that $-\infty \leq a' \leq +\infty$ for all $a' \in \mathbb{I}$; below we write just ∞ to indicate $+\infty$). In order to do that we introduce three artificial variables denoted as v^-, v^0 and v^+ that can be assumed to be contained in Var. It should be taken for granted that v^-, v^0, v^+ always admit values -1, 0 and 1, respectively. Everywhere below we

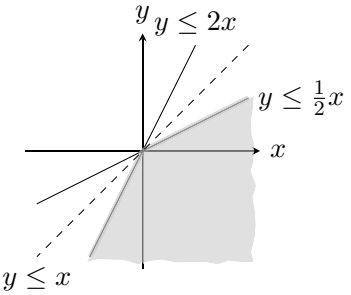


Fig. 3. Region of possible solutions limited by pivotal inequalities $y \leq 2x$ and $y \leq \frac{1}{2}x$.

narrow our attention to valuations $\rho\colon \mathsf{Var} \to^{\bullet} \mathbb{I}$, where $^{\bullet}$ indicates that ρ performs this fixed assignment on v^-, v^0 and v^+. Now that we have defined helper variables, we can express conditions such as $x \in [-5, 3]$ (say when $\mathbb{I} = \mathbb{R}$) as conjunction of inequalities: $v^- \leq \frac{1}{5}x$ and $x \leq 3v^+$. Clearly, only a single inequality is derived in cases where $x \in [b, +\infty]$ (or $x \in [-\infty, c]$). Observe that v^0 is necessary to express inequalities of the form $0 \leq x$.

A system $\mathcal{I}$ may contain some redundant information. We are particularly interested in paired constraints binding the same two variables, e.g. $y \leq 2x$ and $y \leq \frac{1}{2}x$ for $\mathbb{I} = \mathbb{R}$. Unless we have an extra knowledge about x and y (for instance that x and y are limited to positive values) we have to keep track of both inequalities. However, if we additionally had that $y \leq x$ then this constraint would be superfluous as can be seen in Fig. 3. Thus the $y \leq x$ inequality can be safely discarded. This justifies the approach of keeping only the pivotal polar constraints. Namely we establish two complementary functions $s, l\colon \mathsf{Var} \times \mathsf{Var} \to \mathbb{I}_{\geq 0} \cup \{\infty, \mathsf{Nil}\}$ (s stands for *the smallest*, l for *the largest*) defined as

$$\begin{aligned} s(x,y) &\triangleq \min\{a \in \mathbb{I}_{\geq 0} \cup \{\infty\} \mid \text{inequality } x \leq a \cdot y \text{ is in } \mathcal{I}\} \\ l(x,y) &\triangleq \max\{a \in \mathbb{I}_{\geq 0} \cup \{\infty\} \mid \text{inequality } x \leq a \cdot y \text{ is in } \mathcal{I}\} \end{aligned} \tag{1}$$

with an arrangement that $s(x, y)$ and $l(x, y)$ are both equal to the special value Nil in case when $\mathcal{I}$ contains no inequality of the form $x \leq a \cdot y$ (thus $s(x, y) = \mathsf{Nil}$ if and only if $l(x, y) = \mathsf{Nil}$). We extend the standard multiplication in $\mathbb{I}_{\geq 0}$ on ∞ and Nil in the following way:

$$a \cdot \mathsf{Nil} = \mathsf{Nil} \cdot a = \mathsf{Nil} \qquad \infty \cdot a = a \cdot \infty = \infty \text{ iff } a \neq 0 \qquad 0 \cdot \infty = \infty \cdot 0 = 0$$

Using the above settings, we are interested in ascertaining a set of possible solutions γ with respect to s and l:

$$\begin{aligned} \gamma\big((s,l)\big) \triangleq \big\{\rho\colon \mathsf{Var} \to^{\bullet} \mathbb{I} \mid \forall_{x,y \in \mathsf{Var}} \text{ if } s(x,y) \neq \mathsf{Nil} \text{ and } l(x,y) \neq \mathsf{Nil} \\ \text{then } \rho(x) \leq s(x,y) \cdot \rho(y) \text{ and } \rho(x) \leq l(x,y) \cdot \rho(y)\big\} \,. \end{aligned}$$

We refer to these sets as *Weighted Hexagons* throughout this paper.

2.1 Lattice Structure

Paired (s,l) constraints exhibit algebraic properties conforming to the lattice structure. We state these properties more precisely in this subsection.

Let $\mathcal{L}$ be a subset of a Cartesian product $\big(\mathsf{Var}\times\mathsf{Var}\to\mathbb{I}_{\geq 0}\cup\{\infty,\mathsf{Nil}\}\big)^2$ composed of exactly those pairs (s,l) for which $\gamma\big((s,l)\big)$ is not empty and for each $x,y\in\mathsf{Var}$ we have either $s(x,y)=l(x,y)=\mathsf{Nil}$ or $s(x,y)\leq l(x,y)$ (where neither $s(x,y)$ nor $l(x,y)$ is equal to Nil). Additionally, we let a special element $\bot$ to be in $\mathcal{L}$. Intuitively, $\bot$ is a common representative for all empty Weighted Hexagons, i.e. (s,l) pairs with $\gamma\big((s,l)\big)=\emptyset$. We can thus make an arrangement that $\gamma(\bot)=\emptyset$.

We define two binary operations: greatest lower bound $\wedge$ (a *meet*) and least upper bound $\vee$ (a *join*) that operate on pairs of elements from $\mathcal{L}$. For this, we begin with two auxiliary order relations $\leq^{\mathsf{Nil}}$ and $\leq_{\mathsf{Nil}}$ on $\mathbb{I}_{\geq 0}\cup\{\infty,\mathsf{Nil}\}$ that extend $\leq$:

$$a\leq^{\mathsf{Nil}}a' \text{ iff } a\leq a' \text{ or } a'=\mathsf{Nil}\qquad a\leq_{\mathsf{Nil}}a' \text{ iff } a\leq a' \text{ or } a=\mathsf{Nil}\,.$$

Obviously, we can also introduce operators (that are denoted as $\min_{\leq^{\mathsf{Nil}}}$ and $\max_{\leq^{\mathsf{Nil}}}$; similarly for $\leq_{\mathsf{Nil}}$) for computing minimum and maximum of an arbitrary, yet finite, number of elements with respect to $\leq^{\mathsf{Nil}}$ and $\leq_{\mathsf{Nil}}$ orders.

Now for $\mathcal{a},\mathcal{b}\in\mathcal{L}$ we have

$$\mathcal{a}\wedge\mathcal{b}\triangleq\begin{cases}(s_{\mathcal{a}\wedge\mathcal{b}},l_{\mathcal{a}\wedge\mathcal{b}}) & \text{if } \mathcal{a}=(s_{\mathcal{a}},l_{\mathcal{a}}) \text{ and } \mathcal{b}=(s_{\mathcal{b}},l_{\mathcal{b}}) \text{ and } \gamma\big((s_{\mathcal{a}\wedge\mathcal{b}},l_{\mathcal{a}\wedge\mathcal{b}})\big)\neq\emptyset\\ \bot & \text{otherwise,}\end{cases}$$

where for all $x,y\in\mathsf{Var}$ functions $s_{\mathcal{a}\wedge\mathcal{b}}$ and $l_{\mathcal{a}\wedge\mathcal{b}}$ are given by

$$\begin{aligned}s_{\mathcal{a}\wedge\mathcal{b}}(x,y)&\triangleq\min\nolimits_{\leq^{\mathsf{Nil}}}\big(s_{\mathcal{a}}(x,y),s_{\mathcal{b}}(x,y)\big) \qquad \text{and}\\ l_{\mathcal{a}\wedge\mathcal{b}}(x,y)&\triangleq\max\nolimits_{\leq_{\mathsf{Nil}}}\big(l_{\mathcal{a}}(x,y),l_{\mathcal{b}}(x,y)\big)\,.\end{aligned}\tag{2}$$

Likewise

$$\mathcal{a}\vee\mathcal{b}\triangleq\begin{cases}(s_{\mathcal{a}\vee\mathcal{b}},l_{\mathcal{a}\vee\mathcal{b}}) & \text{if } \mathcal{a}=(s_{\mathcal{a}},l_{\mathcal{a}}) \text{ and } \mathcal{b}=(s_{\mathcal{b}},l_{\mathcal{b}})\\ \mathcal{a} & \text{if } \mathcal{b}=\bot\\ \mathcal{b} & \text{otherwise,}\end{cases}$$

where for all $x,y\in\mathsf{Var}$

$$\begin{aligned}s_{\mathcal{a}\vee\mathcal{b}}(x,y)&\triangleq\max\nolimits_{\leq^{\mathsf{Nil}}}\big(s_{\mathcal{a}}(x,y),s_{\mathcal{b}}(x,y)\big) \qquad \text{and}\\ l_{\mathcal{a}\vee\mathcal{b}}(x,y)&\triangleq\min\nolimits_{\leq_{\mathsf{Nil}}}\big(l_{\mathcal{a}}(x,y),l_{\mathcal{b}}(x,y)\big)\end{aligned}\tag{3}$$

only in the case where $\max_{\leq^{\mathsf{Nil}}}\big(s_{\mathcal{a}}(x,y),s_{\mathcal{b}}(x,y)\big)\leq\min_{\leq_{\mathsf{Nil}}}\big(l_{\mathcal{a}}(x,y),l_{\mathcal{b}}(x,y)\big)$, while $s_{\mathcal{a}\vee\mathcal{b}}(x,y)=l_{\mathcal{a}\vee\mathcal{b}}(x,y)=\mathsf{Nil}$ otherwise. An intuition, why we put this restriction, is depicted in Fig. 4. The point a from Fig. 4(c) must belong to $\gamma(\mathcal{a}\vee\mathcal{b})$, but does not fulfil the constraint $\min(l_{\mathcal{a}}(y,x),l_{\mathcal{b}}(y,x))$.

Notice that $(s_{\mathcal{a}\vee\mathcal{b}},l_{\mathcal{a}\vee\mathcal{b}})$ settles a non-contradictory system of inequalities, i.e. $\gamma\big((s_{\mathcal{a}\vee\mathcal{b}},l_{\mathcal{a}\vee\mathcal{b}})\big)\neq\emptyset$. This can be apprehended using the following lemma.

Lemma 2.1 *For all $\mathcal{a},\mathcal{b}\in\mathcal{L}$ we have that:*

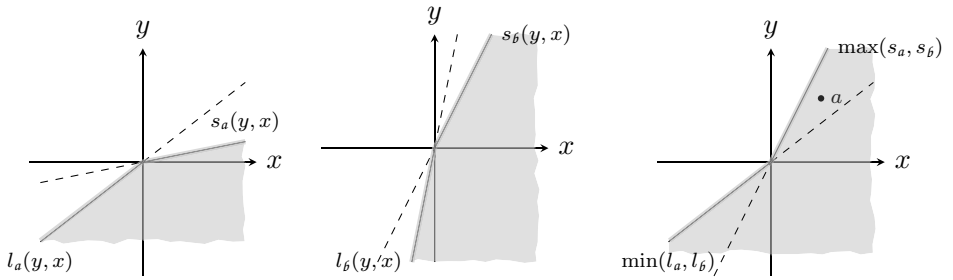


Fig. 4. A hint why $\max(s_a(y,x), s_b(y,x)) \leq \min(l_a(y,x), l_b(y,x))$ is needed in the $\vee$ definition.

(i) $\gamma(a \wedge b) = \gamma(a) \cap \gamma(b)$,

(ii) $\gamma(a \vee b) \supseteq \gamma(a) \cup \gamma(b)$.

Note 1 *Both $\wedge$ and $\vee$ operators can be computed efficiently. This observation may seem to be obvious in case of intersection as the lemma above says that $\wedge$ corresponds directly to this operation. However it is not that evident as in the definition of $\wedge$ there is a check against emptiness of $\gamma\big((s_{a\wedge b}, l_{a\wedge b})\big)$. However the algorithm we present in Section 3 below addresses this issue. It can decide, among other things, whether $(s_{a\wedge b}, l_{a\wedge b})$ represents a satisfiable system of inequalities or not.*

Now that we have listed the above key definitions, we can restate the property mentioned at the beginning of this subsection.

Theorem 2.2 *Set $\mathcal{L}$ forms a lattice under $\wedge$ and $\vee$ operators.*

From the above theorem it follows that $\mathcal{L}$ can be viewed as a poset. In particular, the meet (or join) relation induces a canonical partial ordering on $\mathcal{L}$:

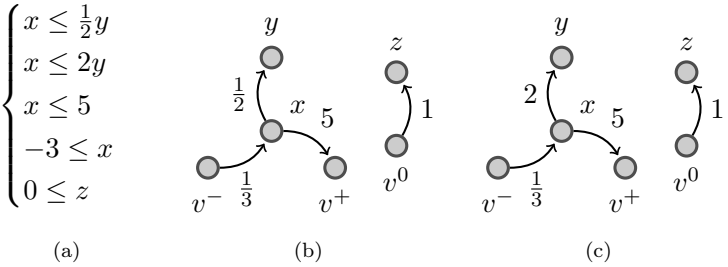
$$a \preceq b \quad \text{iff} \quad a \wedge b = a \qquad \text{for all } a, b \in \mathcal{L}. \tag{4}$$

When $\mathbb{I}$ is chosen as $\mathbb{R}$, then $\mathcal{L}$ with $\wedge$ and $\vee$ form a complete lattice. In this case the concretization function γ uniquely defines an abstraction function $\alpha : \mathcal{P}(\mathsf{Var} \to^{\bullet} \mathbb{I}) \to \mathcal{L}$ as $\alpha(c) = \bigwedge\{a \in \mathcal{L} \mid c \subseteq \gamma(a)\}$ such that γ and α form a Galois connection [9].

2.2 Graph Model

A convenient way to represent a system of inequalities $\mathcal{I}$ is to use graphs. The correspondence between these two is quite straightforward and we favour the graph representation over the set of inequalities model. This allows us to apply a standard graph algorithm that can determine whether $\mathcal{I}$ is satisfiable and, in case it is, we can construct an explicit conforming valuation ρ. We define two weighted digraphs $\mathcal{G}_s = (\mathcal{V}, \mathcal{E}, s)$ and $\mathcal{G}_l = (\mathcal{V}, \mathcal{E}, l)$ with:

- set of vertices $\mathcal{V}$ isomorphic to Var – each vertex is labelled with its own variable name (we can use notions of *vertex* and *variable* interchangeably). We also assume that $\mathcal{V}$ is ordered, i.e. it is isomorphic to $\{1 \ldots |\mathsf{Var}|\}$.



(a) (b) (c)

Fig. 5. A system of inequalities (a) and its graph representation $\mathcal{G}_s$ (b) and $\mathcal{G}_l$ (c).

- sets of directed edges $\mathcal{E}$ such that $(x, y) \in \mathcal{E}$ iff $\mathcal{I}$ contains an inequality of the form $x \le a \cdot y$ for some $a \in \mathbb{I}_{\ge 0}$ (or, which amounts to the same, $s(x, y)$ and $l(x, y)$ are not equal to Nil),
- weight functions s and l given by (1).

A typical pair of graphs $\mathcal{G}_s$ and $\mathcal{G}_l$ is depicted in Fig. 5.

We say that a variable u is *positive* if there exists a path $\langle v^+, v_0, v_1, \ldots, u\rangle$ in $\mathcal{E}$. Similarly u is *negative* if there is a path $\langle u, w_0, w_1, \ldots, v^-\rangle$. For a given weighted digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$ we define *product length* $\pi_{\mathcal{G}}(p)$ of a path $p = \langle v, v_1, v_2, \ldots, v_k, u\rangle$ as $\pi_{\mathcal{G}}(p) = \omega(v, v_1) \cdot \omega(v_1, v_2) \cdots \omega(v_k, u)$.

3 Satisfiability Testing and Normal Form

It may happen that multiple systems of inequalities have the same set of solutions. It is desirable to define a transformation that computes their normal form. This is indeed the aim of the Algorithm 1 we present below. It also solves another problem — the emptiness test. The Algorithm 1 allows determining whether a given system of inequalities is satisfiable.

The algorithm, for $\mathfrak{a} = (s, l)$ and corresponding $\mathcal{G}_s, \mathcal{G}_l$ as above, computes a variant of the transitive closure of the given inequalities. It is based on the well-known Floyd-Warshall algorithm [4] for computing shortest paths between all pairs of vertices in a graph. If the set of solutions $\gamma\big((s, l)\big)$ for the given input graphs is empty, the algorithm returns False. Otherwise it computes a normal form $\mathfrak{a}^* \triangleq (s^*, l^*)$.

Algorithm 1 *Execute the following steps:*

(i) *Add to (s, l) reflexive information $x \le x$, trivial constraints $v^- \le 0 \cdot x$, $v^0 \le 0 \cdot x$ and $x \le \infty \cdot v^+$ and relations between v^-, v^0 and v^+. To achieve this, we define*

For k from 1 to $|\mathsf{Var}|$ we define:

$$\begin{cases} (s_0, l_0)(x,y) & = (s_{in}, l_{in})(x,y) \\ (s_k, l_k)(x,y) & = (\min_{\leq \mathsf{Nil}}(s_{k-1}(x,y), s_{k-1}(x,v_k) \cdot s_{k-1}(v_k,y)), \\ & \quad \max_{\leq \mathsf{Nil}}(l_{k-1}(x,y), l_{k-1}(x,v_k) \cdot l_{k-1}(v_k,y))) \ . \end{cases}$$

Now, $(s_{out}, l_{out})(x,y) = (s_{|\mathsf{Var}|}, l_{|\mathsf{Var}|})(x,y)$.

Fig. 6. Modified Floyd-Warshall ($\mathcal{FW}$) algorithm for finding paths with smallest and greatest product lengths. The pair (s_{out}, l_{out}) denotes the computed output for the input (s_{in}, l_{in}).

(s', l') *as:*

$$(s',l')(x,y)\,[6] = \begin{cases} \big(0,\ \max_{\leq \mathsf{Nil}}(0, l(x,y))\big) & \text{if } x \in \{v^-, v^0\},\ x \neq y \\ \big(\min_{\leq \mathsf{Nil}}(\infty, s(x,y)),\ \infty\big) & \text{if } y = v^+,\ x \notin \{v^-, v^0, v^+\} \\ (1,1) & \text{if } x = y,\ x \notin \{v^-, v^0, v^+\}, \\ & s(x,y) = l(x,y) = \mathsf{Nil} \\ (0,1) & \text{if } x = y = v^- \\ (0,\infty) & \text{if } x = v^0,\ y \in \{v^0, v^+\} \\ & \text{or } y = v^0,\ x \in \{v^-, v^0\} \\ (1,\infty) & \text{if } x = y = v^+ \\ (s,l)(x,y) & \text{otherwise.} \end{cases}$$

(ii) *Find all positive variables. To do this, traverse one of the graphs ($\mathcal{G}_s$ or $\mathcal{G}_l$), using a standard* breadth-first search *algorithm, starting from v^+. Mark each visited node as positive. If v^0 is reached, return* False.

(iii) *Find all negative variables: traverse $\mathcal{G}_s$ or $\mathcal{G}_l$, but with reversed edges, starting from v^-. If v^0 or a node that was previously marked as positive is reached, return* False. *Mark each successfully visited node as negative.*

(iv) *For each pair of vertices, find paths with the smallest and greatest product lengths. We use here a modification of the Floyd-Warshall algorithm ($\mathcal{FW}$) presented in Fig. 6 applied to (s', l'). Let $(s'', l'') = \mathcal{FW}(s', l')$.*

(v) *Find cycles c and $\tilde{c}$ with product length $\pi_{\mathcal{G}_s}(c) < 1$ and $\pi_{\mathcal{G}_l}(\tilde{c}) > 1$: let $\mathcal{X} = \{v \in \mathsf{Var} \mid s''(v,v) < 1\}$ and $\mathcal{Y} = \{v \in \mathsf{Var} \mid l''(v,v) > 1\}$. If $\mathcal{X}$ contains positive variables or $\mathcal{Y}$ contains negative ones, return* False.

(vi) *For $x \in \mathcal{X}$ assign $s''(x,x) = 0$. For $y \in \mathcal{Y}$ assign $l''(y,y) = \infty$. For x such that $l''(x,y) = 0$ for some y, let $s''(x,v^0) = 0$ and let $l''(x,v^0) = \infty$. For x such that $l''(x,y) \neq 0$ and $s''(x,y) = 0$ for some y, let $s''(x,v^0) = 0$ and let $l''(x,v^0) = \infty$.*

(vii) *Apply the algorithm from Fig. 6 to (s'', l'') and denote the output as (s^*, l^*). If there exist $s^*(v^+, v) = 0$ (which corresponds to an $1 \leq 0 \cdot v$) or $l^*(v, v^-) = \infty$*

[6] We use a notation $(s,l)(x,y) \triangleq \big(s(x,y), l(x,y)\big)$.

(that represents $v \leq -\infty$), return False. *Otherwise return* True.

The generated normal form contains for each pair of variables the most restrictive constraints that are fulfilled by each solution of the input system. In the step i of the Algorithm 1 we add some trivial constraints that must be included in the normal form, but cannot be found while computing the standard transitive closure (performed in step iv) — they are not a consequence of the original inequalities. The steps ii and iii allow us to find variables the value of which must be positive (or negative). In the step iv we propagate the constraints, using the Floyd-Warshall algorithm. In step v we check if there exists a positive variable x and an inequality $x \leq a \cdot x$ where $a < 1$ or a negative y and an inequality $y \leq b \cdot y$ where $b > 1$. Such inequalities cannot be satisfied. As for the step vi, if there is a constraint $x \leq a \cdot x$ where $a < 1$ and x is not positive, it can be iterated arbitrarily many times: $x \leq a \cdot x \leq a^2 \cdot x \leq \ldots \leq a^k \cdot x \leq \ldots$. The path with smallest product length between x and x is not well defined. To avoid this problem, we replace in s'' the original inequality $x \leq a \cdot x$ with $x \leq 0 \cdot x$. For a similar reason we replace in l'' inequalities of the form $y \leq b \cdot y$ with $y \leq \infty \cdot y$ if $b > 1$. Finally we propagate the modifications, using the Floyd-Warshall algorithm once more in step vii.

The theorems stated below express the main properties of the Algorithm 1. We start with showing that it can be used as a satisfiability test.

Theorem 3.1 (Satisfiability Test) *$\mathcal{I}$ is satisfiable if and only if the Algorithm 1 returns* True *for the corresponding pair of functions (s, l).*

Proof. If the algorithm returns True, we can construct a valuation $\rho\colon \mathsf{Var} \to^{\bullet} \mathbb{I}$ that satisfies $\mathcal{I}$ in the following way:

- if u is positive then $\rho(u) = \frac{1}{s^*(v^+,u)}$ (note that $s^*(v^+, u) > 0$),
- if u is negative then $\rho(u) = -l^*(u, v^-)$ (note that $l^*(u, v^-) < \infty$),
- $\rho(u) = 0$ in other cases.

Each inequality $x \leq ay$ from the original system $\mathcal{I}$ matches one of the following cases:

- x, y are positive. Then $\rho(x) = 1/s^*(v^+, x)$ and $\rho(y) = 1/s^*(v^+, y)$. Because $s^*(v^+, y)$ denotes the smallest product length of all paths between v^+ and y, we have that
$$s^*(v^+, y) \leq a \cdot s^*(v^+, x) \; .$$
Both $s^*(v^+, x) > 0$ and $s^*(v^+, y) > 0$ (because x, y are positive). Dividing both sides by $s^*(v^+, x) \cdot s^*(v^+, y)$ we get:
$$\frac{1}{s^*(v^+, x)} \leq a \cdot \frac{1}{s^*(v^+, y)} \; .$$
Hence ρ satisfies the inequality $x \leq a \cdot y$, that is $\rho(x) \leq a \cdot \rho(y)$.
- x, y are both negative. Hence $\rho(x) = -l^*(x, v^-)$ and $\rho(y) = -l^*(y, v^-)$. Because $l^*(x, v^-)$ denotes the greatest product length of all paths between x and v^-, it

holds that

$$a \cdot l^*(y, v^-) \leq l^*(x, v^-) \ .$$

Multiplying both sides by -1 we get

$$-l^*(x, v^-) \leq -a \cdot l^*(y, v^-) \ .$$

and therefore the valuation ρ satisfies $x \leq a \cdot y$.

- In other cases x is not positive and y is not negative. Hence $\rho(x) \leq 0$ and $\rho(y) \geq 0$. All coefficients in all inequalities are nonnegative and so valuation ρ satisfies $x \leq a \cdot y$.

A standard case analysis can be applied to show that if the algorithm returns False then $\gamma\big((s,l)\big) = \emptyset$. □

The following theorem shows that if the algorithm returns True, then the computed output (s^*, l^*) is equivalent to the input (s, l).

Theorem 3.2 (Correctness) *The computed output (s^*, l^*) has the same set of solutions as the input (s, l): $\rho \in \gamma\big((s,l)\big)$ iff $\rho \in \gamma\big((s^*, l^*)\big)$.*

Proof. We show that no solution is lost and no new one is introduced. No solution is lost since the computed new weight of an edge should be obeyed anyway due to the transitivity of inequalities. No solution is introduced since all the inequalities in the result are not less tight than the original ones. □

Now we can show that the output of the Algorithm 1 is indeed a normal form of all domain elements that have the same set of solutions.

Theorem 3.3 (Normal Form) *If (s, l) is satisfiable then*

$$\inf_{\preceq}\{c \in \mathcal{L} \mid \gamma(c) = \gamma\big((s,l)\big)\}$$

is well defined and equal to (s^*, l^*).

Proof. The pair (s^*, l^*) is in $A = \{c \in \mathcal{L} \mid \gamma(c) = \gamma\big((s,l)\big)\}$ by Theorem 3.2. We then take any element of A and show that each of its elements after the normalisation with Algorithm 1 is still above (s^*, l^*). □

A join of two Weighted Hexagons does not need to be a Weighted Hexagon. Our join operator ($\vee$) computes only some over-approximation of the union of Weighted Hexagons described by its arguments (see Lemma 2.1). The following theorem shows that normalizing both arguments results in the smallest possible Weighted Hexagon:

Theorem 3.4 (Best Approximation) *Normal forms a^* and b^* computed by the Algorithm 1 can be used to find the best over-approximation of $\gamma(a) \cup \gamma(b)$:*

$$\gamma(a^* \vee b^*) = \inf_{\subseteq}\{\gamma(c) \mid \gamma(c) \supseteq \gamma(a^*) \cup \gamma(c) \supseteq \gamma(b^*)\} \ .$$

Proof. We prove first the property

$$\gamma(a^* \vee b^*) = \inf_{\subseteq}\{\gamma(c \vee d) \mid \gamma(a^*) \subseteq \gamma(c) \text{ and } \gamma(b^*) \subseteq \gamma(d)\} .$$

Then standard considerations give the required result. □

4 Widening Operator

When a loop is encountered in a control flow graph of the analysed program, one may need to compute a fixpoint to find the corresponding abstract state. When the abstract domain used in the analysis is of an infinite height this may lead to infinite iteration process. To avoid this problem, *widening operator* was introduced [6]. It is used to compute an over-approximation of the fixpoint in a finite number of steps. Thus we need to define the widening operator for the Weighted Hexagons. Intuitively, if for a pair of variables x and y the second argument of the widening contains a weaker constraint than the first one, then the result does not contain a constraint for this pair. We define the widening operator $\nabla \colon \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ as:

$$a \nabla b \triangleq \begin{cases} (s_{a\nabla b}, l_{a\nabla b}) & \text{if } a = (s_a, l_a) \text{ and } b = (s_b, l_b) \\ a & \text{if } b = \bot, \\ b & \text{otherwise,} \end{cases}$$

where for any $x, y \in \mathsf{Var}$:

- if $s_b(x,y) \leq^{\mathsf{Nil}} s_a(x,y)$ and $l_a(x,y) \leq_{\mathsf{Nil}} l_b(x,y)$ then:

$$s_{a\nabla b}(x,y) = s_a(x,y) \qquad \text{and} \qquad l_{a\nabla b}(x,y) = l_a(x,y) .$$

- $s_{a\nabla b}(x,y) = l_{a\nabla b}(x,y) = \mathsf{Nil}$ otherwise.

We have to show that ∇ defined above is in fact a widening operator.

Theorem 4.1 *Our operator ∇ meets the definition of a widening operator:*

(i) *for all $a, b \in \mathcal{L}$ $\quad a \vee b \preceq a \nabla b$,*

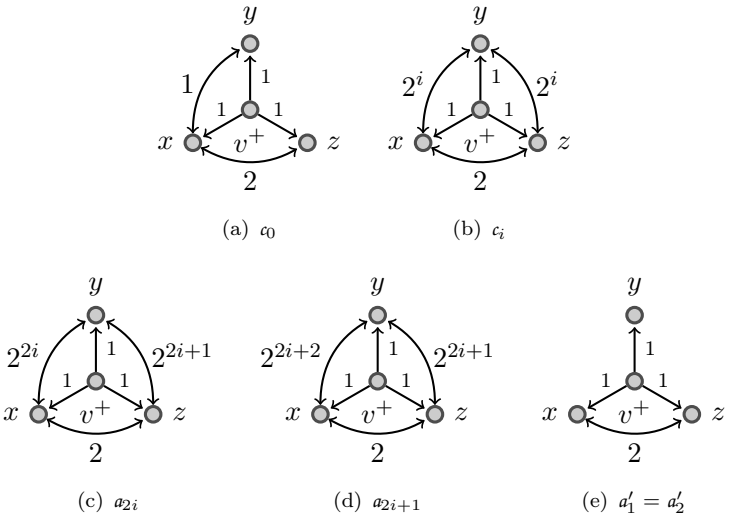
(ii) *for every infinite sequence of abstract states $c_0, c_1, \ldots$, sequence $a_0, a_1, \ldots$ defined as:*

$$\begin{cases} a_0 = c_0 \\ a_i = a_{i-1} \nabla c_i \quad \text{for } i > 0 \end{cases}$$

is not strictly increasing.

Proof. Direct examination of the required properties. □

Note that one should not normalize the result of the widening (replace $(s_{a\nabla b}, l_{a\nabla b})$ with $(s_{a\nabla b}, l_{a\nabla b})^*$). This may result in a potentially infinite increasing sequence $a_0, a_1, \ldots$. Figure 7 presents a sequence $c_0, c_1, \ldots$ that generates infinite increasing sequence $a_0, a_1, \ldots$. For simplicity, only a part of the $\mathcal{G}_s$ graph is presented (the variables v^- and v^0 are not shown). Note that using the widening operator without normalization, the sequence a' stabilises after the first step (Figure 7(e)).



(a) c_0 (b) c_i (c) a_{2i} (d) a_{2i+1} (e) $a'_1 = a'_2$

Fig. 7. Normalization of the result of widening may give a strictly increasing infinite sequence.

5 Transfer Function

To apply an abstract domain to program analysis, one needs to specify how an abstract state is altered by each type of instruction. The transfer function must over-approximate the concrete semantics [6]. We show a few examples of how to model a test in the Weighted Hexagons. Let $\mathcal{a}$ represent a set of states before a test e. We compute an over-approximation $\mathcal{b}$ of the set after the test:

$$\{\rho \in \gamma(\mathcal{a}) \mid \rho \text{ satisfies } e\} \subseteq \gamma(\mathcal{b}) \ .$$

For $e = v_1 \leq c * v_2$ we compute $\mathcal{b}$ as:

$$\mathcal{b} = \begin{cases} \bot & \text{if } \mathcal{a} = \bot \text{ or } \gamma(\mathcal{b}') = \emptyset \\ \mathcal{b}' & \text{otherwise,} \end{cases}$$

where $\mathcal{b}'$ is defined as:

$$\mathcal{b}'(x,y) = \begin{cases} \big(s_{\mathcal{a}}(x,y), l_{\mathcal{a}}(x,y)\big) & \text{if } (x,y) \neq (v_1, v_2) \\ \big(\min_{\leq \mathsf{Nil}}(s_{\mathcal{a}}(x,y), c), \max_{\leq \mathsf{Nil}}(l_{\mathcal{a}}(x,y), c)\big) & \text{otherwise.} \end{cases}$$

Let us now consider an assignment $w \leftarrow w + k$ for $k > 0$. Figure 8 shows the result of an assignment in a two-dimensional case. Note that the transfer function introduces some loss of precision — the computed Weighted Hexagon over-approximates the exact result of the operation (represented in Figure 8(b) as a dashed shape).

The whole transfer rule for an assignment is long and technical. For clarity and brevity we present here only the case when the added constant k is positive. The result $\mathcal{b}$ of an assignment, given a non-empty input $\mathcal{a}$, is computed as follows:

- for $x, y \neq w$: $\mathcal{b}(x,y) = (s^*_{\mathcal{a}}(x,y), l^*_{\mathcal{a}}(x,y))$

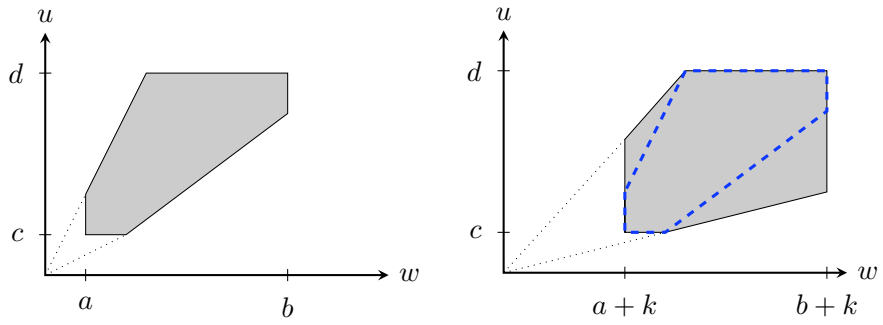


Fig. 8. Weighted Hexagon before (a) and after (b) assignment $w \leftarrow w + k$. The dashed shape represents the exact result of the assignment.

- $\mathcal{b}(w,y) = \begin{cases} (s^*_a(w,y) + k \cdot s^*_a(v^+,y), \infty) & \text{if } s^*_a(v^+,w) \neq \mathsf{Nil},\ s^*_a(w,y) \neq \mathsf{Nil} \\ (0, l^*_a(w,y) + \frac{k}{l^*_a(y,v^-)}) & \text{if } l^*_a(y,v^-) \neq \mathsf{Nil},\ l^*_a(w,v^-) \neq \mathsf{Nil}, \\ & \quad l^*_a(w,v^-) - k > 0 \\ (\mathsf{Nil}, \mathsf{Nil}) & \text{otherwise,} \end{cases}$
- $\mathcal{b}(x,w) = (s_{\mathcal{b}}(x,w), l_{\mathcal{b}}(x,w))$, where:
  - $s_{\mathcal{b}}(x,w) = \begin{cases} \frac{s^*_a(x,w) \cdot s^*_a(x,v^+)}{s^*_a(x,v^+) + k \cdot s^*_a(x,w)} & \text{if } s^*_a(x,w) \neq \mathsf{Nil},\ s^*_a(x,v^+) \neq \mathsf{Nil} \\ s^*_a(x,w) & \text{otherwise,} \end{cases}$
  - $l_{\mathcal{b}}(x,w) = \begin{cases} \frac{l^*_a(x,w)}{1 - k \cdot l^*_a(v^-,w)} & \text{if } 0 < l^*_a(v^-,w) < \infty,\ 1 - k \cdot l^*_a(v^-,w) > 0 \\ l^*_a(x,w) & \text{otherwise.} \end{cases}$

6 Conclusion

We have described a new numerical abstract domain that represents inequalities of the form $x \leq a \cdot y$, between pairs of variables x, y with a being a non-negative constant, as well as interval constraints $x \in [b,c]$, where b and c are arbitrary constants. It is an abstract domain that handles multiplication in a simple fashion. Let n denote the number of program variables. An abstract state can be represented with $O(n^2)$ worst case memory cost. The most time-consuming operations are the normalisation and satisfiability test, which use the Floyd-Warshall algorithm. Their worst case time complexity is $O(n^3)$. The domain of Weighted Hexagons lies between Pentagons and TVPI in terms of expressiveness and efficiency.

We plan to investigate how to extend the Weighted Hexagons to handle strict inequalities and work within the ring of integers. We are also working on an implementation of an analyser for the Java language that uses our domain.

References

[1] Balasundaram, V. and K. Kennedy, *A technique for summarizing data access and its use in parallelism enhancing transformations*, SIGPLAN Not. **24** (1989), pp. 41–53.

[2] Blanchet, B., P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival, *A static analyzer for large safety-critical software*, in: *PLDI '03* (2003), pp. 196–207.

[3] Bourdoncle, F., *Abstract debugging of higher-order imperative languages*, in: *PLDI '93* (1993), pp. 46–55.

[4] Cormen, T. H., C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms," MIT Press, 1990.

[5] Cousot, P., *Design of syntactic program transformations by abstract interpretation of semantic transformations*, in: *Proceedings of the 17th International Conference on Logic Programming* (2001), pp. 4–5.

[6] Cousot, P. and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in: *POPL '77* (1977), pp. 238–252.

[7] Cousot, P. and R. Cousot, *Static determination of dynamic properties of recursive procedures*, in: E. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA* (1977), pp. 237–277.

[8] Cousot, P. and R. Cousot, *Systematic design of program analysis frameworks*, in: *POPL '79* (1979), pp. 269–282.

[9] Cousot, P. and R. Cousot, *Abstract interpretation and application to logic programs*, J. Log. Program. **13** (1992), pp. 103–179.

[10] Cousot, P. and N. Halbwachs, *Automatic discovery of linear restraints among variables of a program*, in: *POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages* (1978), pp. 84–96.

[11] Logozzo, F. and M. Fähndrich, *Pentagons: a weakly relational abstract domain for the efficient validation of array accesses*, in: *SAC '08* (2008), pp. 184–188.

[12] Miné, A., *The octagon abstract domain*, Higher Order Symbolic Computation **19** (2006), pp. 31–100.

[13] Simon, A., A. King and J. M. Howe, *Two variables per linear inequality as an abstract domain*, in: *LOPSTR'02: Proceedings of the 12th international conference on Logic based program synthesis and transformation* (2003), pp. 71–89.