Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 288 (2012) 25–36

www.elsevier.com/locate/entcs

Sweeping in Abstract Interpretation

Krzysztof Jakubczyk [1,2]

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

Abstract

In this paper we present how sweeping line techniques, which are very popular in computational geometry, can be adapted for static analysis of computer software by abstract interpretation. We expose how concept of the sweeping line can be used to represent elements of a numerical abstract domain of *boxes*, which is a disjunctive refinement of a well known domain of *intervals* that allows finite number of disjunctions. We provide a detailed description of the representation along with standard domain operations algorithms. Furthermore we introduce very precise widening operator for the domain.
Additionally we show that the presented idea of the representation based on sweeping line technique is a generalisation of the representation that uses Linear Decision Diagrams (LDD), which is one of possible optimisations of our idea. We also show that the presented widening operator is often more precise than the previous one.

Keywords: Numerical Abstract Domains, Static Analysis, Abstract Interpretation, Widening Operator, Sweeping Line

1 Introduction

Several numerical abstract domains for Abstract Interpretation [5] have been proposed. The most popular are: *intervals* [4], *pentagons* [12], *octagons* [14], *two variables per inequality* (TVPI) [16] or *convex polyhedra* [7]. These domains represent conjunctions of some subsets of linear constraints on program variables (e.g. *intervals* represent only constraints between a variable and some constant, *pentagons* also allow strict inequality constraints between two variables). All mentioned numerical domains have one common source of inaccuracy — they represent only convex sets. This means that in most cases when calculating disjunctions of domain elements one has to over-approximate real result. Usually the approximation is defined as the least upper bound in the domain — e.g. for the domain of *octagons* it is the smallest octagon containing the disjuncts.

[1] This work was partly supported by Polish government grant N N206 493138.

[2] Email: kjk@mimuw.edu.pl

doi:10.1016/j.entcs.2012.10.005

In this paper we present an adaptation of the concept of a sweeping line to abstract interpretation. Sweeping line algorithms are very important in computational geometry. They are used to compute all crossings in a set of line segments (Bentley-Ottmann algorithm [2]) or a construction of the Voronoi diagram (Fortune's algorithm [8]). We present a construction that demonstrates how the sweeping line technique can be used to efficiently represent elements and perform operations for a domain that is a disjunctive refinement of the domain of *intervals*. It handles both strict and non-strict inequalities. We also give some ideas for optimisations of the presented base version, which may reduce the size of the representation and the cost of domain operations. Additionally, we introduce a new, very precise, widening operator and present a boundary on the precision of widening operators for the domain.

1.1 Related Work

Abstract interpretation successfully takes advantage of techniques used in various fields. There have been works which employ different graph-based algorithms [9,14]. Also lately quadtrees — a data structure used in computational geometry, was proposed to be adapted for the abstract interpretation [11]. The current paper introduces a new technique to the field of abstract interpretation — a sweeping line technique [2], which is one of the key techniques of the computational geometry. The introduction of the new concept brings new intuitions. We use the concept of a sweeping line to represent elements of the domain of *boxes* which is a disjunctive refinement of the domain of *intervals*.

There have been proposed some ways to build a disjunctive refinement, usually by special strategies of controlling the disjuncts [1,15,13], but most of them do not scale to a large number of disjuncts. Also achieving a satisfactory precision of the widening is hard. Recently, a new implementation of the domain of *boxes* has been proposed [10]. The solution by Gurfinkel and Chaki is based on Linear Decision Diagrams (LDD) and easily scales to large number of disjuncts. Additionally quite high precision of widening was presented. This paper covers the same area as the one by Gurfinkel and Chaki — we propose a different approach to the same domain of *boxes*. Our construction is more generic — the implementation of the domain that uses LDD's can be regarded as an optimisation of the technique presented in this paper. Widening operator introduced in this paper uses thresholds [3] to gain precision. Single application of the widening operator gives more precise result than than the one by Gurfinkel and Chaki.

1.2 Paper Outline

Section 2 introduces the problem we solve. In Section 3 we describe our adaptation of the sweeping line technique. Section 4 presents the representation of elements of the domain of *boxes* and Section 5 outlines the implementation of the domain operators. Section 6 describes a new widening operator for *boxes*. In Section 7 we discuss transfer functions and we conclude in Section 8.

2 Problem Definition

The basic version of the domain of *intervals* makes it possible to represent only convex sets. We would like to extend this to represent finite disjunctions of *intervals*. We define this more formally in what follows. Let Var be a set of variables and $\mathbb{I}$ be chosen to be ring of reals $\mathbb{R}$, or rationals $\mathbb{Q}$, or integers $\mathbb{Z}$. The abstract domain of *intervals* is a tuple:

$$\langle \mathbb{B}_n, \subseteq, \emptyset, \mathbb{I}^n, \uplus, \cap \rangle$$

where $\mathbb{B}_n = \{\mathcal{B} \subseteq \mathbb{I}^n | \mathcal{B}$ is expressed by strict/non-strict interval constraints$\}$, $\subseteq$ is subset ordering, $\emptyset$ is an empty set ($\bot$ of the ordering), $\uplus$ is the least over-approximation of the join of two elements from $\mathbb{B}_n$ and $\cap$ is an intersection.

We extend this construction to represent finite sets of elements from domain of *intervals*. The domain of *boxes* is a tuple:

$$\langle \mathbb{BS}_n, \subseteq, \emptyset, \mathbb{I}^n, \cup, \cap \rangle$$

where $\mathbb{BS}_n = \{\mathcal{BS} \in \mathbb{I}^n |$ there exist $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k \in \mathbb{B}_n$ such that $\mathcal{BS} = \bigcup_{i=1}^{k} \mathcal{B}_i\}$, $\subseteq$ is the subset ordering, $\emptyset$ is the empty set that is also $\bot$ of the ordering, $\cup$ is the join of two elements of the domain (join) and $\cap$ is the intersection (meet). The improvement compared to the domain of *intervals* is that in the domain of *boxes* the operation $\cup$ does not require approximation. Our goal is to efficiently represent elements of the domain of *boxes* and efficiently perform domain operations on such elements.

3 Adaptation of the Sweeping Line Technique

The idea of the sweeping line technique is to imagine that a line (usually a vertical one) is swept across the plane, stopping at some points. A data structure, which is associated with the sweeping line, is updated every time the line is stopped. When the line has swept across all points, a result of interest is computed from the data structure.

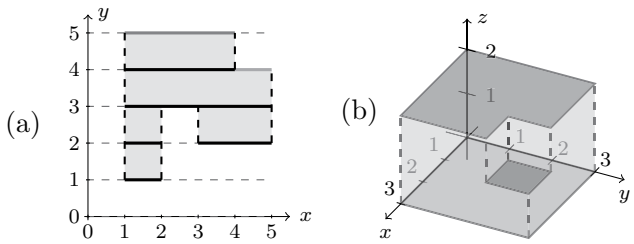

Figure 1. The idea of describing domain elements using the sweeping line technique (a) the two-dimensional version, (b) the three-dimensional one

To demonstrate the idea how to adapt the technique to construct the domain of *boxes*, we describe two and three-dimensional examples presented in Figure 1. First, let us focus on the 2-dimensional version (a). We sweep through the space of the variable y starting from $-\infty$ to $+\infty$ and observe what happens with values

of x. As the data associated with the sweeping line — Sl_1 we store elements of one-dimensional version of the domain (possible values of the variable x). Points at which we stop during the process of sweeping are values of y for which the one-dimensional version of domain changes. This is illustrated in the Figure 1(a), where values of Sl_1 for $y \in \{1, 2, 3, 4\}$ are displayed in black — e.g. for $y < 1$ Sl_1 is empty, for $y = 1$ it changes to interval $[1, 2]$ and for $y > 5$ it becomes empty again.

We proceed analogically for a 3-dimensional version from Figure 1(b). When we sweep though the space of the variable z, the data associated with the sweeping line — Sl_2, is a 2-dimensional version of the domain. Thus for $z < 0$ we have that $Sl_2 = \emptyset$, for $z = 0$ it becomes an area described by a 2-dimensional representation: for $y < 0$ the data is $Sl_1 = \emptyset$, for $y = 0$ it changes to an interval $[0, 3]$ and for $y > 3$ the data Sl_1 becomes $\emptyset$. For $z = 1$ the area becomes a little more complicated (a small square is removed) and for $z > 2$ we have $Sl_2 = \emptyset$.

4 Representation of Domain Elements

In order to manage strict and non-strict inequalities, special points that are encountered during the process of sweeping are described as pairs $(i, b) \in \mathbb{I} \times \mathbb{PM}$ where $\mathbb{PM} = \{\oplus, \ominus\}$. Such pairs describe beginnings of intervals. In case $(i, \oplus)$, the number i is included in the interval and in case $(i, \ominus)$ the value i is excluded from one. Let $\mathbb{P}$ be a set of such pairs, that is:

$$\mathbb{P} = \{(i, b) \in \mathbb{I} \times \mathbb{PM}\}. \tag{1}$$

Additionally, when $\mathbb{I} = \mathbb{Z}$ we add a restriction that $\ominus$ is not used. We introduce an ordering on elements of $\mathbb{P}$, that is $\prec \subseteq \mathbb{P} \times \mathbb{P}$, defined as:

$$(i, b) \prec (i', b') \iff i < i' \text{ or } i = i' \wedge b = \oplus \wedge b' = \ominus \tag{2}$$

and $\preceq$: $\mathbb{P} \times \mathbb{P}$ which is the reflexive closure of $\prec$. Note that these are lexicographical orderings on pairs in case when $\mathbb{PM}$ is ordered as $\oplus < \ominus$. When $\mathbb{I} = \mathbb{Z}$ the ordering is isomorphic to the ordering on first elements of pairs only. In order to describe all beginnings of intervals we extend the set of special points $\mathbb{P}$ with $-\infty$. We define $\mathbb{P}_\infty = \mathbb{P} \cup \{-\infty\}$. Also, we extend the ordering $\prec$ on elements of $\mathbb{P}$ to an ordering on $\mathbb{P}_\infty$ in the natural fashion, so that $\forall_{p \in \mathbb{P}} - \infty \prec p$ and analogically $\preceq$.

We introduce a relation $in \subseteq \mathbb{P}_\infty \times \mathbb{I} \times \mathbb{P}_\infty$ defined as follows:

$$in(p, i, p') \iff p \preceq (i, \oplus) \prec p' \tag{3}$$

It states that $i \in \mathbb{I}$ belongs to the interval described by p and p', where p is the beginning of the interval and p' is the beginning of the next one, e.g. interval $[3, 7)$ is represented by a pair $\langle (3, \oplus), (7, \oplus) \rangle$.

Lemma 4.1 states that any two elements from $\mathbb{P}_\infty$ that represent beginnings of consequent intervals describe a non-empty interval — it contains at least one element from $\mathbb{I}$.

Lemma 4.1 (density) *For all $p, p' \in \mathbb{P}_\infty$ if $p \prec p'$ then there exists $i \in \mathbb{I}$ for which $in(p, i, p')$.*

We use the ordering $\prec$ as the base to construct representation for elements of *boxes*. Let us define an infinite sequence of sets $\mathbb{S}_1, \mathbb{S}_2, \ldots$ as follows:

$$\begin{aligned} \mathbb{S}_0 &= \{\epsilon, \top_0\}, \\ \mathbb{S}_{n+1} &= \{((p_1, v_1), (p_2, v_2), \ldots, (p_m, v_m)) \,|v_1, \ldots, v_m \in \mathbb{S}_n, v_1 \neq \epsilon, \\ &\qquad p_1, \ldots, p_m \in \mathbb{P}_\infty, \forall_{j \in \{1,\ldots,m-1\}} p_j \prec p_{j+1} \wedge v_j \neq v_{j+1}\} \end{aligned} \tag{4}$$

where ϵ is the empty sequence.

First elements of pairs in a sequence $\mathcal{S} \in \mathbb{S}_n$ for $n > 0$ describe special points encountered during the process of sweeping through the n-th dimension. Restriction for $\mathbb{I} = \mathbb{Z}$ that $\ominus$ is not used in x_j is introduced because we want the representation of domain elements to be unique — so that each domain element would have only one possible representation. This can be easily achieved and simplifies domain operations because normalisation operation is not needed.

We define a function $\mathit{find} : \mathbb{I} \times \bigcup_{n>0} \mathbb{S}_n \to \mathbb{N}$ and an auxiliary notation, so that for any $i \in \mathbb{I}$ and $\mathcal{S} = ((p_1, v_1), \ldots (p_m, v_m)) \in \mathbb{S}_n$:

$$\mathit{find}(i, \mathcal{S}) = \begin{cases} 0 & \text{if } (i, \oplus) \prec p_1 \\ k & \text{otherwise} \end{cases} \qquad \mathcal{S}[i] = \begin{cases} \epsilon & \text{if } \mathit{find}(i, \mathcal{S}) = 0 \\ v_{\mathit{find}(i,\mathcal{S})} & \text{otherwise} \end{cases} \tag{5}$$

where $k = \max(j | j \in \{1, \ldots, m\}$ and $p_j \preceq (i, \oplus))$. If $\mathit{find}(i, \mathcal{S}) = k$ and $0 < k < m$ then $in(p_k, i, p_{k+1})$. The function *find* outputs the index of the interval in $\mathcal{S}$ that contains i and $\mathcal{S}[i]$ outputs the value assigned to the interval.

Now we define a relation which states when an element of $\mathbb{I}^n$ belongs to a domain element represented by $\mathcal{S} \in \mathbb{S}_n$. The satisfiability relation $sat_n \subseteq \mathbb{I}^n \times \mathbb{S}_n$ is defined as follows:

$$\begin{cases} sat_0(\boldsymbol{\epsilon}, \mathcal{S}) & \iff \mathcal{S} = \top_0 \\ sat_{n+1}(\langle i_{n+1}, i_n, \ldots, i_1 \rangle, \mathcal{S}) & \iff \mathcal{S}[i_{n+1}] = w \wedge sat_n(\langle i_n, \ldots, i_1 \rangle, w) \end{cases} \tag{6}$$

With the *sat* relation we define subset of $\mathbb{I}^n$ which is represented by $\mathcal{S} \in \mathbb{S}_n$:

$$\gamma(S) = \{\boldsymbol{i} \in \mathbb{I}^n | sat_n(\boldsymbol{i}, \mathcal{S})\} \tag{7}$$

Property 4.1 states that the proposed representation is unique in terms of the *sat* relation. Therefore normalisation is not needed.

Property 4.1 (Uniqueness of representation) *For $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_n$ if $\mathcal{S} \neq \mathcal{S}'$ then there exists $\boldsymbol{i} \in \mathbb{I}^n$ for which only one of $sat_n(\boldsymbol{i}, \mathcal{S})$ and $sat_n(\boldsymbol{i}, \mathcal{S}')$ holds.*

We have proved that elements of the sequence $\mathbb{S}_n$ represent unique subsets of $\mathbb{I}^n$ but we still have not proved that they can be used to represent elements of *boxes*. We proceed for the proof to the next section.

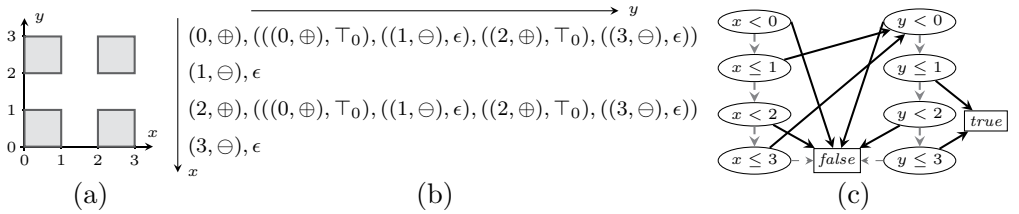

Figure 2. Comparison of the proposed representation (b) and the representation based on LDD's (c) for an example element from the domain of *boxes* (a). In (c) solid black and dashed grey arrows represent *true* and *false* branches, respectively.

Representation of *boxes* based on Linear Decision Diagrams
An LDD is a binary decision diagram with two terminal nodes: *true*, *false* in which non-terminal nodes (decisions) are linear constraints. The representation of *boxes* proposed by Gurfinkel and Chaki uses LDD's with relational interval constraints (both strict and non-strict). A total order on variables $\preceq : \mathsf{Var} \times \mathsf{Var} \to \mathsf{Var}$ is extended to linear constraints:

$$(x_1 \precsim_1 k_1) \preceq (x_2 \precsim_2 k_2) \iff (x_1 \preceq x_2) \vee ((x_1 \preceq_1 k_1) \Rightarrow (x_2 \precsim_2 k_2))$$

where $\precsim_1, \precsim_2 \in \{<, \preceq\}$ and then to nodes:

$$u \preceq v \iff (v \in \{\mathit{true}, \mathit{false}\}) \vee (u \notin \{\mathit{true}, \mathit{false}\} \wedge \mathit{label}(u) \preceq \mathit{label}(v))$$

LDD's that satisfy certain ordering and reduction constraints are canonical representations of propositional formulæ. An example of an LDD for an element of *boxes* is presented in Figure 2(c).

The proposed representation can be considered as a generalisation of the representation based on LDD's. Interval constraints stored in a LDD are sorted by the variable first and then by the entailment of constraints for the variable (see Figure 2(c)). The proposed representation sorts in the same way (see Figure 2(b)) — each special point corresponds to a constraint (node) in the LDD. The main difference is that the LDD is optimised so that there are no duplicate nodes — in Figure 2(b) sequences for $x = (0, \oplus)$ and $x = (2, \oplus)$ are the same thus in the corresponding LDD black arrows from $x \leq 1$ and $x \leq 3$ lead to the same node. The LDD-based representation is one of the optimisations that can be applied. Possibly some graph algorithms or compression techniques used in graphical applications could be employed here.

5 Domain Operations

In this section we describe an implementation of exact $\cup$, $\cap$ and $\subseteq$ operations on elements of sequences $\mathbb{S}_0, \mathbb{S}_1, \ldots$. We use these algorithms to prove the correspondence between elements in our representation and the domain of *boxes*, i.e. that the representation using the sweeping line technique can be used to describe elements of the domain.

```
1  def Fix(S):
2      v_prev ← ϵ
3      for (p, v) in S:
4          if v ≠ v_prev:
5              v_prev ← v
6              yield (x, v)
```

```
1  def Op_aux(S, S', n, ♦):
2      if n = 0: yield S♦S', return
3      for (p, p') in segm(spec(S) ∪ spec(S')):
4          yield (p, Op(S[p], S'[p], n − 1, ♦)
5  def Op(S, S', n, ♦):
6      return Fix(Op_aux(S, S', n, ♦))
```

Figure 3. Python-like language implementation of the ♦-extension

We define a function $spec_L : \bigcup_{n>0} \mathbb{S}_n \to 2^{\mathbb{P}_\infty}$ which outputs a set of local special points for a given sequence as follows:

$$spec_L(\mathcal{S}) = \{p_1, \ldots, p_k\} \text{ where } \mathcal{S} = ((p_1, v_1), \ldots, (p_k, v_k)) \,. \tag{8}$$

We say that a sequence $segm(X) = ((-\infty, p_1), (p_1, p_2), \ldots, (p_{k-1}, +\infty))$ for $X \subseteq \mathbb{P}_\infty$ is a segmentation by X if $X \setminus \{-\infty\} = \{p_1, \ldots, p_{k-1}\}$ and for all $i \in \{1, \ldots, k-2\}$ it holds that $p_i \prec p_{i+1}$. Naturally for a set X there exists only one segmentation by X.

Let $\blacklozenge : \mathbb{S}_0 \times \mathbb{S}_0 \to \mathbb{S}_0$ be some operator defined for $\mathbb{S}_0$. We extend the operator to $\mathcal{S}_n$ for any $n \geq 0$ by an inductive construction. Let $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_n$ and $segm(spec_L(\mathcal{S}) \cup spec_L(\mathcal{S}')) = ((p_0, p_1), (p_1, p_2), \ldots, (p_{k-1}, p_k))$. We define ♦-extension $\blacklozenge : \mathbb{S}_n \times \mathbb{S}_n \to \mathbb{S}_n$ as $\mathcal{S} \blacklozenge \mathcal{S}' = \mathcal{R}$ for $n > 0$, such that:

$$\mathcal{R}[p_i] = \mathcal{S}[p_i] \blacklozenge \mathcal{S}'[p_i] \tag{9}$$

where $i \in \{0, \ldots, k-1\}$. When $\mathcal{R}$ is then normalised so that $\mathcal{R}[p_i] \neq \mathcal{R}[p_{i+1}]$, such definition yields exactly one element of $\mathbb{S}_n$. A natural implementation of ♦-extension is presented in Figure 3. It has time complexity $O(||\mathcal{S}|| \cdot ||\mathcal{S}'||)$ for $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_n$, where $||\mathcal{S}||$ for $\mathcal{S} \in \mathbb{S}_n$, $n > 0$ denotes the size of $\mathcal{S}$ (the sum of lengths of all sequences that appear in $\mathcal{S}$).

Lemma 5.1 (Domain operations) *Exact domain operations can be defined as ♦-extensions of the operations for $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_0$:*

- ***join*** *by an extension of $\mathcal{S} \blacklozenge \mathcal{S}' = \top_0 \iff \mathcal{S} = \top_0$ or $\mathcal{S}' = \top_0$;*
- ***meet*** *by an extension of $\mathcal{S} \blacklozenge \mathcal{S}' = \top_0 \iff \mathcal{S} = \mathcal{S}' = \top_0$;*
- ***inclusion*** *for $\subseteq$ we first compute extension of $\mathcal{S} \blacklozenge \mathcal{S}' = \top_0 \iff \mathcal{S} = \top_0$ and $\mathcal{S}' = \epsilon$. For $\mathcal{S}_n, \mathcal{S}'_n \in \mathbb{S}_n$ it holds that $\mathcal{S}_n \subseteq \mathcal{S}'_n \iff \mathcal{S}_n \blacklozenge \mathcal{S}'_n = \epsilon$.*

Lemma 5.1 states how domain operations can be defined by ♦-extension.

Theorem 5.2 (Implementation of *boxes*) *There is a one-to-one correspondence between elements of $\mathbb{BS}_n$ and elements of $\mathbb{S}_n$, for any $\boldsymbol{i} \in \mathbb{I}^n$:*

$$\boldsymbol{i} \in \mathcal{BS} \iff sat(\boldsymbol{i}, \mathcal{S}) \tag{10}$$

Theorem 5.2 states that the introduced representation of elements of domain of *boxes* along with the operations from Lemma 5.1 can be used to represent the domain of *boxes*.

6 Widening Operator

We use a variation of the classical definition of the widening operator (see [1] or [6]) where the second argument is greater or equal to the first one. The main idea behind the construction of the proposed operator is to compute before the widening sequence a set of special points (thresholds) for each variable: $spec_\nabla : \mathsf{Var} \to 2^{\mathbb{P}\infty}$, where $spec_\nabla(v)$ for $v \in \mathsf{Var}$ is finite. Then, if a refinement is needed, we take into consideration these points.

The construction of the widening is recursive. When we compute $\mathcal{R} = \mathcal{S} \nabla \mathcal{S}'$ for $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_n$, two segmentations are prepared: sg and sg'. The first one is a segmentation by a set $spec_L(\mathcal{S}) \cup spec_\nabla(v_n)$ and the second one is by the set extended by $spec_L(\mathcal{S}')$. The second segmentation is more precise than the first one — some segments from sg may be split to smaller ones in sg'. The result of the widening is created by calculating a widening separately for each segment in sg'. If it happens that some segment (p, p') from sg' is inside a segment from sg and $\mathcal{S}[p] = \mathcal{S}'[p]$ then a refinement is needed. Otherwise we would get an infinite strictly increasing sequence, as for a sequence of intervals: $[0,1], [0,2], [0,3], \ldots$. If the segment (p, p') has a right side neighbour in sg' which is included in the same segment in sg then as a refinement we choose the value of this neighbour. Otherwise we choose the value of the left neighbour.

Let $\nabla : \mathbb{S}_0 \times \mathbb{S}_0 \to \mathbb{S}_0$ such that $\mathcal{S}_0 \nabla \mathcal{S}'_0 = \mathcal{S}_0 \cup \mathcal{S}'_0$ for $\mathcal{S}_0, \mathcal{S}'_0 \in \mathbb{S}_0$. We extend the operator to $\mathbb{S}_n$ for any $n > 0$ by an inductive construction. Let $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_n$, $sg = segm(spec_\nabla(v) \cup spec_L(\mathcal{S}))$, $sg' = segm(spec_\nabla(v) \cup spec_L(\mathcal{S}) \cup spec_L(\mathcal{S}')) = (p_0, p_1), (p_1, p_2), \ldots, (p_{k-1}, p_k)$. We define ∇-extension, $\nabla : \mathbb{S}_{n+1} \times \mathbb{S}_{n+1} \to \mathbb{S}_{n+1}$ as $\mathcal{S} \nabla \mathcal{S}' = \mathcal{R}$, such that $\mathcal{R} = \mathcal{S}'$ if $\mathcal{S} = \epsilon$ and otherwise:

$$\mathcal{R}[p_i] = \begin{cases} \mathcal{S}[p_i] \nabla_n \mathcal{S}'[p_i] & \text{if } \mathcal{S}[p_i] \neq \mathcal{S}'[p_i] \text{ or } sg'(p_i) = sg(p_i) \\ \mathcal{S}[p_i] \nabla_n \mathcal{S}'[p_{i+1}] & \text{if } i < k \text{ and } sg'(p_{i+1}) \subseteq sg(p_i) \\ \mathcal{S}[p_i] \nabla_n \mathcal{S}'[p_{i-1}] & \text{if } 0 < i \text{ and } sg'(p_{i-1}) \subseteq sg(p_i) \end{cases} \tag{11}$$

where $i \in \{0, \ldots, k-1\}$. In the widening, when a refinement is done, a segment from $\mathcal{S}$ is replaced by finitely many segments for which value is strictly greater than the original one.

Initial special points $spec_\nabla$ are computed once at the beginning of the whole widening sequence. There are many tactics for building $spec_\nabla$. These are a few examples:

- take global special points from the first element of the sequence: $spec_\nabla = spec_G(\mathcal{S}_0)$, where $spec_G : \mathbb{S}_n \to \mathsf{Var} \to 2^{\mathbb{P}\infty}$ is a function defined as follows:

$$spec_G(\mathcal{S})(v) = \begin{cases} spec_L(\mathcal{S}) & \text{if } v = v_m, \\ \bigcup_{j \in \{1,\ldots,k\}} spec_G(v, s_j) & \text{otherwise} \end{cases}$$

  where $\mathcal{S} = ((p_1, s_1), \ldots, (p_k, s_k))$.
- the set of initial special points can be updated finitely many times during the

process of widening, for example for first k iterations, after i-th iteration the set of special points for variable v is updated $spec_\nabla(v) \leftarrow spec_\nabla(v) \cup spec_G(\mathcal{S}_i)$

- the set of special points can be based on the source code of the analysed software, e.g. it may contain all constants that appear in the code.

Theorem 6.1 (Widening) *The ∇-extension is a proper widening operator.*

Comparison with the widening operator based on LDD's

The proposed widening operator is similar to ∇_{ldd} — the widening operator based on LDD's. Construction of the widening ∇_{ldd} expressed in terms of the presented sweeping line implementation of the domain of *boxes* differs by segmentations used in (11). It does not take advantage of $spec_\nabla$ thus $sg_{ldd} = segm(spec_L(\mathcal{S}))$ and $sg'_{ldd} = segm(spec_L(\mathcal{S}) \cup spec_L(\mathcal{S}'))$. The consequence is stated by Theorem 6.2 — the widening operator proposed in this paper is more precise (in single step) than ∇_{ldd}.

Theorem 6.2 *For any $n \geq 0$ and $\mathcal{S}, \mathcal{S}' \in \mathbb{S}_n$ it holds that:*

$$\mathcal{S} \nabla \mathcal{S}' \subseteq \mathcal{S} \nabla_{ldd} \mathcal{S}' \tag{12}$$

.

For the comparison of ∇ and ∇_{ldd} consider an example from Figure 4. Values of the variable y for the block $x \in [0, 1]$ need a refinement. The widening ∇_{ldd} refines the interval for y to $[0, +\infty)$. In the example $2 \in spec_\nabla(y)$ thus we choose $[0, 2]$ as the refined interval for y. The sequence of refinements for values of y in the block stabilises because the set $spec_\nabla(y)$ is finite.

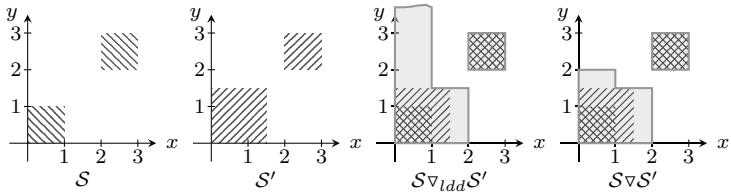

Figure 4. Comparison of widening operators for $spec_\nabla(y) = spec_G(\mathcal{S})(y) = \{0, 1, 2, 3\}$.

Theorem 6.3 states what accuracy of the widening operator can be expected. By the introduction of special points $spec_\nabla$ to use in the widening sequence we try to get closer to the result that would not depend on the variable ordering. The result of the proposed widening in the example from Figure 4 is pretty close to the intersection of widenings based on LDD's for both variable orderings.

Theorem 6.3 *For $n > 1$ there is no widening operator for the domain of* boxes *which is both more precise than the one based on LDD's and which does not depend on the variable ordering.*

Proof *(Outline)* We prove this by showing a 2-dimensional strictly increasing sequence and a divergent sequence constructed by an intersection of two possible

widenings based on LDD's (for the two variable orderings). Both sequences are presented in Figure 5. The first element x_0 consists of two squares. Next elements are constructed by adding a rectangle next to the upper left corner and lower right corner segments. Every new rectangle is $1/2$ times thinner than the previous one and also longer by it's width. Additionally, starting from the element x_2, in the upper right corner a square is added. It gets bigger in next elements. All three polygons converge to the point where two dashed lines intersect. The sequence in Figure 5(b) illustrates an intersection of the widenings based on LDD's for both possible variable orderings. The sequence also converges to the center point but it is strictly increasing. □

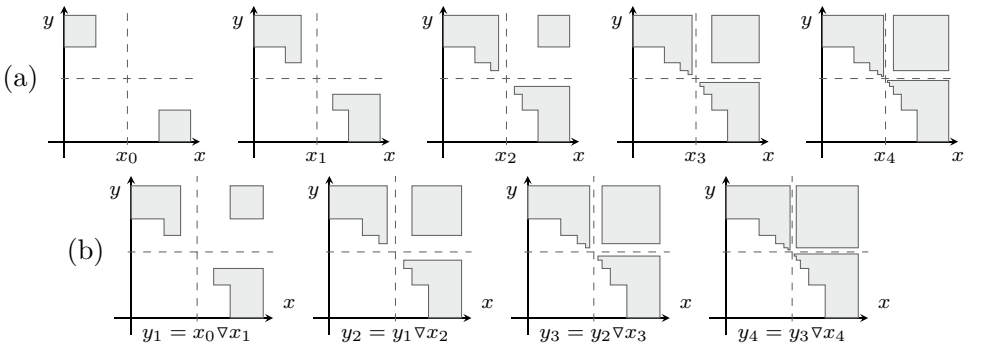

Figure 5. Divergent widening (a) Sequence for widening $x_1, x_2, x_3, \ldots$, (b) intersection of possible results of widening $y_1 = x_1 \triangledown x_2, y_2 = y_2 \triangledown x_3, \ldots$

7 Transfer Functions

In this section we show a few examples of a transfer function for the presented implementation of *boxes*. To apply a guard $g_{in} = p_a \preceq v_k \prec p_b$ where $p_a, p_b \in \mathbb{P}$ to $\mathcal{S} \in \mathbb{S}_n$, we intersect all sequences for the variable v_k with $((p_a, \top_{k-1}), (p_b, \epsilon))$. To apply the assignment $v_k \leftarrow a$ for $a \in \mathbb{I}$, every sequence for the variable v_k is replaced by $(((a, \oplus), \mathcal{S}'), ((a, \ominus), \epsilon))$ where $\mathcal{S}' \in \mathbb{S}_{k-1}$ is computed as join of values (second elements) in the original sequence. Shift operation $v_k \leftarrow v_k + a$ for some $a \in \mathbb{I}$ comes down to shift by a first elements in all sequences for v_k. An assignment $v_k \leftarrow a \cdot v_k$ is a bit tricky. For $a < 0$ we first reverse sequences for v_k (remembering they have to belong to $\mathbb{S}_k$) and then perform the multiplication. In case $\mathbb{I} = \mathbb{Z}$ and $a \notin \mathbb{Z}$ we might have to additionally fix the result.

Transfer function becomes more complicated for the case $v_k \leftarrow v_k + a \cdot v_l$ where $k \neq l$ and $a \neq 0$. If $l < k$ then for sequences that represent variable v_k we already have an interval for v_l set. All we have to do is to update each interval in these sequences. Some intervals may overlap thus we have to fix the sequence. When $k < l$ then for a sequence for variable v_k we do not have set limitations on v_l yet. We have to go down to sequences for v_l, compute value for v_k and propagate this up to the sequence for v_k.

8 Conclusion

In this paper we have introduced a sweeping line technique to the abstract interpretation. We have used it to create a representation for the domain of *boxes*. Our construction generalises the construction of the domain that uses LDD's. Additionally we have proved how far we can go with the accuracy of the widening operator and proposed a more precise widening operator.

We plan to check how the new widening operator behaves in practise and investigate what are the best values of the initial set of special points. We would also like to think of optimisations of the representation. We might try to introduce approximate versions of presented operators, e.g. which limit the length of sequences.

Acknowledgement

We thank Aleksy Schubert for numerous insightful discussions and his comments on the paper.

References

[1] R. Bagnara, P. Hill, and E. Zaffanella. Widening operators for powerset domains. *International Journal on Software Tools for Technology Transfer (STTT)*, 9:413–414, 2007. 10.1007/s10009-007-0029-y.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, 28:643–647, September 1979.

[3] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, chapter Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software, pages 85–108. Lecture Notes in Computer Science 2566. Springer-Verlag, 2002.

[4] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[6] P. Cousot and R. Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In *Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming*, pages 269–295, London, UK, 1992. Springer-Verlag.

[7] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[8] S. Fortune. A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on Computational geometry*, SCG '86, pages 313–322, New York, NY, USA, 1986. ACM.

[9] J. Fulara, K. Durnoga, K. Jakubczyk, and A. Schubert. Relational abstract domain of weighted hexagons. *Electron. Notes Theor. Comput. Sci.*, 267:59–72, October 2010.

[10] A. Gurfinkel and S. Chaki. Boxes: a symbolic abstract domain of boxes. In *Proceedings of the 17th international conference on Static analysis*, SAS'10, pages 287–303, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] J. M. Howe, A. King, and C. Lawrence-Jones. Quadtrees as an abstract domain. *Electron. Notes Theor. Comput. Sci.*, 267:89–100, October 2010.

[12] F. Logozzo and M. Fähndrich. Pentagons: a weakly relational abstract domain for the efficient validation of array accesses. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 184–188, New York, NY, USA, 2008. ACM.

[13] L. Mauborgne and X. Rival. Trace partitioning in abstract interpretation based static analyzers. In *In ESOP*, pages 5–20. Springer, 2005.

[14] A. Miné. The octagon abstract domain. *Higher Order Symbol. Comput.*, 19:31–100, March 2006.

[15] S. Sankaranarayanan, F. Ivancic, I. Shlyakhter, and A. Gupta. Static analysis in disjunctive numerical domains. In *SAS*, volume 4134 of *LNCS*, pages 3–17. Springer, 2006.

[16] A. Simon, 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*, pages 71–89, Berlin, Heidelberg, 2003. Springer-Verlag.