Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)



[Electronic Notes in Theoretical Computer Science 288 (2012) 25–36](http://dx.doi.org/10.1016/j.entcs.2012.10.005)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Sweeping in Abstract Interpretation

Krzysztof Jakubczyk [1](#_bookmark0)*,*[2](#_bookmark0)

*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

# Introduction

Several numerical abstract domains for Abstract Interpretation [[5](#_bookmark24)] have been pro- posed. The most popular are: *intervals* [[4](#_bookmark23)], *pentagons* [[12](#_bookmark31)], *octagons* [[14](#_bookmark32)], *two vari- ables per inequality* (TVPI) [[16](#_bookmark33)] or *convex polyhedra* [[7](#_bookmark26)]. These domains represent conjunctions of some subsets of linear constraints on program variables (e.g. *inter- vals* 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](mailto:kjk@mimuw.edu.pl)

1571-0661 © 2012 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

[doi:10.1016/j.entcs.2012.10.005](http://dx.doi.org/10.1016/j.entcs.2012.10.005)

In this paper we present an adaptation of the concept of a sweeping line to ab- stract 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](#_bookmark21)]) or a construction of the Voronoi diagram (Fortune’s algo- rithm [[8](#_bookmark27)]). 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. *Related Work*

Abstract interpretation successfully takes advantage of techniques used in various fields. There have been works which employ different graph-based algorithms [[9](#_bookmark28),[14](#_bookmark32)]. Also lately quadtrees — a data structure used in computational geometry, was pro- posed to be adapted for the abstract interpretation [[11](#_bookmark30)]. The current paper in- troduces a new technique to the field of abstract interpretation — a sweeping line technique [[2](#_bookmark21)], 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](#_bookmark20),[15](#_bookmark34),[13](#_bookmark35)], 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](#_bookmark29)]. 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](#_bookmark22)] to gain precision. Single application of the widening operator gives more precise result than than the one by Gurfinkel and Chaki.

* 1. *Paper Outline*

Section [2](#_bookmark1) introduces the problem we solve. In Section [3](#_bookmark2) we describe our adaptation of the sweeping line technique. Section [4](#_bookmark4) presents the representation of elements of the domain of *boxes* and Section [5](#_bookmark8) outlines the implementation of the domain operators. Section [6](#_bookmark12) describes a new widening operator for *boxes*. In Section [7](#_bookmark18) we discuss transfer functions and we conclude in Section [8](#_bookmark19).

# 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 I be chosen to be ring of reals R, or rationals Q, or integers Z. The abstract domain of *intervals* is a tuple:

⟨B*n,* ⊆*,* ∅*,* I*n,* à*,* ∩⟩

where B*n* = {B ⊆ I*n*|B is expressed by strict/non-strict interval constraints}, ⊆ is subset ordering, ∅ is an empty set (⊥ of the ordering), à is the least over- approximation of the join of two elements from B*n* and ∩ is an intersection.

We extend this construction to represent finite sets of elements from domain of

*intervals*. The domain of *boxes* is a tuple:

⟨BS*n,* ⊆*,* ∅*,* I*n,* ∪*,* ∩⟩

where BS*n* = {BS ∈ I*n*| there exist B1*,* B2*,...,* B*k* ∈ B*n* such that BS = t*k* B*i*},

*i*=1

⊆ is the subset ordering, ∅ is the empty set that is also ⊥ of the ordering, ∪ is the join of two elements of the domain (join) and ∩ is the intersection (meet). The improvement compared to the domain of *intervals* is that in the domain of *boxes* the operation ∪ does not require approximation. Our goal is to efficiently represent elements of the domain of *boxes* and efficiently perform domain operations on such elements.

# 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.



*z*

2

1

1

1

2

3

*x*

2

3

*y*

(a)

*y*

5

4

3

2

1

0 0 1 2

3 4 5 *x*

(b)

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](#_bookmark3). First, let us focus on the 2-dimensional version (a). We sweep through the space of the variable *y* starting from −∞ to +∞ 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](#_bookmark3)(a), where values of *Sl*1 for *y* ∈ {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](#_bookmark3)(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 = ∅, for *z* = 0 it becomes an area described by a 2-dimensional representation: for *y <* 0 the data is *Sl*1 = ∅, for *y* = 0 it changes to an interval [0*,* 3] and for *y >* 3 the data *Sl*1 becomes ∅. For *z* = 1 the area becomes a little more complicated (a small square is removed) and for *z >* 2 we have *Sl*2 = ∅.

# Representation of Domain Elements

In order to manage strict and non-strict inequalities, special points that are encoun- tered during the process of sweeping are described as pairs (*i, b*) ∈ I × PM where PM = {⊕*,* g}. Such pairs describe beginnings of intervals. In case (*i,* ⊕), the num- ber *i* is included in the interval and in case (*i,* g) the value *i* is excluded from one. Let P be a set of such pairs, that is:

P = {(*i, b*) ∈ I × PM}*.* (1)

Additionally, when I = Z we add a restriction that g is not used. We introduce an ordering on elements of P, that is ≺⊆ P × P, defined as:

(*i, b*) ≺ (*i*'*, b*') ⇐⇒ *i < i*' or *i* = *i*' ∧ *b* = ⊕∧ *b*' = g (2)

and ≤: P × P which is the reflexive closure of ≺. Note that these are lexicographical orderings on pairs in case when PM is ordered as ⊕ *<* g. When I = 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 P with −∞. We define P∞ = P ∪ {−∞}. Also, we extend the ordering ≺ on elements of P to an ordering on P∞ in the natural fashion, so that ∀*p*∈P −∞ ≺ *p* and analogically ≤.

We introduce a relation *in* ⊆ P∞ × I × P∞ defined as follows:

*in*(*p, i, p*') ⇐⇒ *p* ≤ (*i,* ⊕) ≺ *p*' (3)

It states that *i* ∈ 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 ⟨(3*,* ⊕)*,* (7*,* ⊕)⟩.

Lemma [4.1](#_bookmark5) states that any two elements from P∞ that represent beginnings of consequent intervals describe a non-empty interval — it contains at least one element from I.

**Lemma 4.1 (density)** *For all p, p*' ∈ P∞ *if p* ≺ *p*' *then there exists i* ∈ I *for which in*(*p, i, p*')*.*

We use the ordering ≺ as the base to construct representation for elements of

*boxes*. Let us define an infinite sequence of sets S1*,* S2*,...* as follows:

S0 = {*ϵ,* 𝖳0}*,*

S*n*+1 = {((*p*1*, v*1)*,* (*p*2*, v*2)*,...,* (*pm, vm*)) |*v*1*,..., vm* ∈ S*n, v*1 ≠ *ϵ,*

*p*1*,..., pm* ∈ P∞*,* ∀*j*∈{1*,...,m*−1}*pj* ≺ *pj*+1 ∧ *vj* ≠ *vj*+1}

where *ϵ* is the empty sequence.

(4)

First elements of pairs in a sequence S ∈ S*n* for *n >* 0 describe special points encountered during the process of sweeping through the *n*-th dimension. Restriction for I = Z that g is not used in *xj* 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.

t

We define a function *ﬁnd* : I × *n>*0 S*n* → N and an auxiliary notation, so that for any *i* ∈ I and S = ((*p*1*, v*1)*,...* (*pm, vm*)) ∈ S*n*:

*ﬁnd*( )= 0 if (*i,* ⊕) ≺ *p*1

*i,* S

*k* otherwise

[ ]= *ϵ* if *ﬁnd*(*i,* S)=0

*vfind*(*i,*S) otherwise

S *i*

(5)

where *k* = max(*j*|*j* ∈ {1*,..., m*} and *pj* ≤ (*i,* ⊕)). If *ﬁnd*(*i,* S)= *k* and 0 *< k < m* then *in*(*pk, i, pk*+1). The function *ﬁnd* outputs the index of the interval in S that contains *i* and S[*i*] outputs the value assigned to the interval.

Now we define a relation which states when an element of I*n* belongs to a domain element represented by S ∈ S*n*. The satisfiability relation *satn* ⊆ I*n* × S*n* is defined as follows:

*sat*0(***s****,* S) ⇐⇒ S = 𝖳0

*satn*+1(⟨*in*+1*, in,..., i*1⟩*,* S) ⇐⇒ S[*in*+1]= *w* ∧ *satn*(⟨*in,..., i*1⟩*, w*) With the *sat* relation we define subset of I*n* which is represented by S ∈ S*n*:

(6)

*γ*(*S*)= {***i*** ∈ I*n*|*satn*(***i****,* S)} (7)

Property [4.1](#_bookmark6) 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* S*,* S' ∈ S*n if* S ≠ S' *then there exists* ***i*** ∈ I*n for which only one of satn*(***i****,* S) *and satn*(***i****,* S') *holds.*

We have proved that elements of the sequence S*n* represent unique subsets of 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.

*y*

3

2

1

0 0 1

2 3

(a)

*y*

(0*,* ⊕)*,* (((0*,* ⊕)*,* 𝖳0)*,* ((1*,* g)*, є*)*,* ((2*,* ⊕)*,* 𝖳0)*,* ((3*,* g)*, є*))

*x*

(1*,* g)*,є*

(2*,* ⊕)*,* (((0*,* ⊕)*,* 𝖳0)*,* ((1*,* g)*, є*)*,* ((2*,* ⊕)*,* 𝖳0)*,* ((3*,* g)*, є*))

(3*,* g)*,є*

*x*

(b)

(c)

*x <* 0

*y <* 0

*x ≤* 1

*y ≤* 1

*x <* 2

*y <* 2

*x ≤* 3

*y ≤* 3

*false*

*true*

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 ≤ : Var × Var → Var is extended to linear constraints:

(*x*1 Ç1 *k*1) ≤ (*x*2 Ç2 *k*2) ⇐⇒ (*x*1 ≤ *x*2) 𝖵 ((*x*1 ≤1 *k*1) ⇒ (*x*2 Ç2 *k*2)) where Ç1*,* Ç2 ∈ {*<,* ≤} and then to nodes:

*u* ≤ *v* ⇐⇒ (*v* ∈ {*true, false*}) 𝖵 (*u* g {*true, false*}∧ *label*(*u*) ≤ *label*(*v*))

LDD’s that satisfy certain ordering and reduction constraints are canonical repre- sentations of propositional formulæ. An example of an LDD for an element of *boxes* is presented in Figure [2](#_bookmark7)(c).

The proposed representation can be considered as a generalisation of the repre- sentation 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 Fig- ure [2](#_bookmark7)(c)). The proposed representation sorts in the same way (see Figure [2](#_bookmark7)(b)) — each special point corresponds to a constraint (node) in the LDD. The main dif- ference is that the LDD is optimised so that there are no duplicate nodes — in Figure [2](#_bookmark7)(b) sequences for *x* = (0*,* ⊕) and *x* = (2*,* ⊕) are the same thus in the cor- responding LDD black arrows from *x* ≤ 1 and *x* ≤ 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.

# Domain Operations

In this section we describe an implementation of exact ∪, ∩ and ⊆ operations on elements of sequences S0*,* S1*,.* We use these algorithms to prove the correspon-

dence 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*
3. **def** Op\_aux ( S , S*'* , *n* , ♦ ):
4. **if** *n* = 0 : yield S ♦ S*'* , **return**

*prev* → *є*

1. **for** (*p, v*) **in** S :
2. **for** (*p, p'*) **in** *segm*(*spec*(S) ∪ *spec*(S*'*)) :
3. **if** *v* ≠ *v* :

4 y ield (*p,* Op ( S[*p*], S*'*[*p*], *n* − 1, ♦)

*prev*

5

*vprev* → *v*

1. **def** Op ( S , S*'* , *n* , ♦ ):
2. y ield (*x, v*) 6 **return** Fix ( Op\_aux ( S , S*'* , *n* , ♦ ))

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

t

We define a function *specL* : *n>*0 S*n* → 2P*∞* which outputs a set of local special points for a given sequence as follows:

*specL*(S)= {*p*1*,..., pk*} where S = ((*p*1*, v*1)*,...,* (*pk, vk*)) *.* (8) We say that a sequence *segm*(*X*)= ((−∞*, p*1)*,* (*p*1*, p*2)*,...,* (*pk*−1*,* +∞)) for *X* ⊆ P∞

is a segmentation by *X* if *X* \ {−∞} = {*p*1*,..., pk*−1} and for all *i* ∈ {1*,...,k* − 2} it holds that *pi* ≺ *pi*+1. Naturally for a set *X* there exists only one segmentation by *X*.

Let ♦ : S0 ×S0 → S0 be some operator defined for S0. We extend the operator to S*n* for any *n* ≥ 0 by an inductive construction. Let S*,* S' ∈ S*n* and *segm*(*specL*(S) ∪ *specL*(S'))= ((*p*0*, p*1)*,* (*p*1*, p*2)*,...,* (*pk*−1*, pk*)). We define ♦-extension ♦ : S*n* × S*n* → S*n* as S ♦ S' = 𝑌 for *n >* 0, such that:

𝑌[*pi*]= S[*pi*] ♦ S'[*pi*] (9)

where *i* ∈ {0*,...,k* − 1}. When 𝑌 is then normalised so that 𝑌[*pi*] ≠ 𝑌[*pi*+1], such definition yields exactly one element of S*n*. A natural implementation of ♦-extension is presented in Figure [3](#_bookmark9). It has time complexity *O*(||S||· ||S'||) for S*,* S' ∈ S*n*, where

||S|| for S ∈ S*n*, *n >* 0 denotes the size of S (the sum of lengths of all sequences that appear in S).

**Lemma 5.1 (Domain operations)** *Exact domain operations can be deﬁned as*

* *-extensions of the operations for* S*,* S' ∈ S0*:*
* ***join*** *by an extension of* S ♦ S' = 𝖳0 ⇐⇒ S = 𝖳0 *or* S' = 𝖳0*;*
* ***meet*** *by an extension of* S ♦ S' = 𝖳0 ⇐⇒ S = S' = 𝖳0*;*
* ***inclusion*** *for* ⊆ *we ﬁrst compute extension of* S ♦ S' = 𝖳0 ⇐⇒ S = 𝖳0 *and* S' =

*ϵ. For* S*n,* S' ∈ S*n it holds that* S*n* ⊆ S' ⇐⇒ S*n* ♦ S' = *ϵ.*

*n*

*n*

*n*

Lemma [5.1](#_bookmark10) 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* BS*n and elements of* S*n, for any* ***i*** ∈ I*n:*

***i*** ∈ BS ⇐⇒ *sat*(***i****,* S) (10)

Theorem [5.2](#_bookmark11) states that the introduced representation of elements of domain of *boxes* along with the operations from Lemma [5.1](#_bookmark10) can be used to represent the domain of *boxes*.

# Widening Operator

We use a variation of the classical definition of the widening operator (see [[1](#_bookmark20)] or [[6](#_bookmark25)]) 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*o : Var → 2P*∞* , where *spec*o(*v*) for *v* ∈ 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 𝑌 = SoS' for S*,* S' ∈ S*n*, two segmentations are prepared: *sg* and *sg*'. The first one is a segmentation by a set *specL*(S)∪*spec*o(*vn*) and the second one is by the set extended by *specL*(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 S[*p*] = 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]*,.* 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 o : S0 × S0 → S0 such that S0oS0' = S0 ∪ S0' for S0*,* S0' ∈ S0. We extend

the operator to S*n* for any *n >* 0 by an inductive construction. Let S*,* S' ∈ S*n*, *sg* = *segm*(*spec*o(*v*) ∪ *specL*(S)), *sg*' = *segm*(*spec*o(*v*) ∪ *specL*(S) ∪ *specL*(S')) = (*p*0*, p*1)*,* (*p*1*, p*2)*,...,* (*pk*−1*, pk*). We define o-extension, o : S*n*+1 × S*n*+1 → S*n*+1 as SoS' = 𝑌, such that 𝑌 = S' if S = *ϵ* and otherwise:

⎧⎪⎨S[*pi*]o*n*S'[*pi*] if S[*pi*] ≠ S'[*pi*] or *sg*'(*pi*)= *sg*(*pi*)

𝑌[*pi*]=

S[*pi*]o*n*S'[*pi*+1] if *i < k* and *sg*'(*pi*+1) ⊆ *sg*(*pi*)

(11)

⎪S[*pi*]o*n*S'[*pi*−1] if 0 *< i* and *sg*'(*pi*−1) ⊆ *sg*(*pi*)

⎩

where *i* ∈ {0*,...,k* − 1}. In the widening, when a refinement is done, a segment from S is replaced by finitely many segments for which value is strictly greater than the original one.

Initial special points *spec*o are computed once at the beginning of the whole widening sequence. There are many tactics for building *spec*o. These are a few examples:

* take global special points from the first element of the sequence: *spec*o = *specG*(S0),

where *specG* : S*n* → Var → 2P*∞* is a function defined as follows:

*spec*

(S)(*v*)= *specL*(S) if *v* = *vm,*

*G*

t

*j*∈{1*,...,k*}

*spec*

*G*(*v, sj*) otherwise

where S = ((*p*1*, s*1)*,...,* (*pk, sk*)).

* 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*o(*v*) → *spec*o(*v*) ∪ *specG*(£*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* o*-extension is a proper widening operator.*

## Comparison with the widening operator based on LDD’s

The proposed widening operator is similar to o*ldd* — the widening operator based on LDD’s. Construction of the widening o*ldd* expressed in terms of the presented sweeping line implementation of the domain of *boxes* differs by segmentations used in ([11](#_bookmark13)). It does not take advantage of *spec*o thus *sgldd* = *segm*(*specL*(£)) and

*sg*

'

*ldd*

= *segm*(*specL*(£) ∪ *specL*(£')). The consequence is stated by Theorem [6.2](#_bookmark14) —

the widening operator proposed in this paper is more precise (in single step) than

o*ldd*.

**Theorem 6.2** *For any n* ≥ 0 *and* £*,* £' ∈ S*n it holds that:*

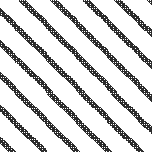
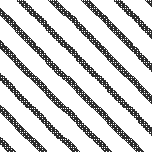
£o£' ⊆ £o*ldd*£' (12)

*.*

For the comparison of o and o*ldd* consider an example from Figure [4](#_bookmark15). Values of

the variable *y* for the block *x* ∈ [0*,* 1] need a refinement. The widening o*ldd* refines the interval for *y* to [0*,* +∞). In the example 2 ∈ *spec*o(*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*o(*y*) is finite.

S S*'*



*y*

3

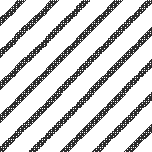
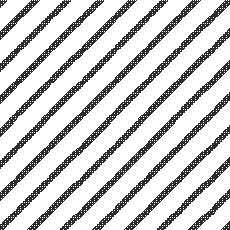
2

1

1 2 3

*x* *x*

So*ldd*S*'*



1 2 3

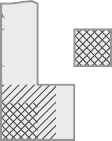
*y*

3

2

1

*x*



*y*

3

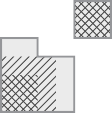
2

1

1 2 3

*x*

SoS*'*



*y*

3

2

1

1 2 3

Figure 4. Comparison of widening operators for *spec*o(*y*)= *specG*(S)(*y*)= {0*,* 1*,* 2*,* 3}.

Theorem [6.3](#_bookmark16) states what accuracy of the widening operator can be expected. By the introduction of special points *spec*o 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](#_bookmark15) 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 se- quence 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](#_bookmark17). 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](#_bookmark17)(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. □

(a)

*y*

*x*0 *x*

*y*

*y*1 = *x*0o*x*1

*x*1

*y*

*x*

*y*

*x*2 *x*

*y*

*y*3 = *y*2o*x*3

*x*3

*y*

*x*

*y*

*x*4 *x*

*y*

*y*4 = *y*3o*x*4

(b)

*y*

*y*2 = *y*1o*x*2

*x x x* *x*

Figure 5. Divergent widening (a) Sequence for widening *x*1*, x*2*, x*3*,.. .*, (b) intersection of possible results of widening *y*1 = *x*1o*x*2*, y*2 = *y*2o*x*3*,...*

# Transfer Functions

In this section we show a few examples of a transfer function for the presented implementation of *boxes*. To apply a guard *gin* = *pa* ≤ *vk* < *pb* where *pa, pb* ∈ P to £ ∈ S*n*, we intersect all sequences for the variable *vk* with ((*pa,* 𝖳*k*−1)*,* (*pb, ϵ*)). To apply the assignment *vk* → *a* for *a* ∈ I, every sequence for the variable *vk* is replaced by (((*a,* ⊕)*,* £')*,* ((*a,* g)*, ϵ*)) where £' ∈ S*k*−1 is computed as join of values (second elements) in the original sequence. Shift operation *vk* → *vk* + *a* for some *a* ∈ I comes down to shift by *a* first elements in all sequences for *vk*. An assignment *vk* → *a* · *vk* is a bit tricky. For *a <* 0 we first reverse sequences for *vk* (remembering they have to belong to S*k*) and then perform the multiplication. In case I = Z and *a* g Z we might have to additionally fix the result.

Transfer function becomes more complicated for the case *vk* → *vk* + *a* · *vl* where *k* ≠ *l* and *a* ≠ 0. If *l < k* then for sequences that represent variable *vk* we already have an interval for *vl* 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 *vk* we do not have set limitations on *vl* yet. We have to go down to sequences for *vl*, compute value for *vk* and propagate this up to the sequence for *vk*.

# Conclusion

In this paper we have introduced a sweeping line technique to the abstract inter- pretation. 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. Addition- ally 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 in- vestigate 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 intro- duce 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.