Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 287 (2012) 41–52

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

TreeKs: A Functor to Make Numerical Abstract Domains Scalable

Mehdi Bouaziz[1]

DI - École Normale Supérieure
45 rue d'Ulm
75230 Paris Cedex 05 - France

Abstract

Relational numerical abstract domains do not scale up. To ensure a linear cost of abstract domains, abstract interpretation-based tools analyzing large programs generally split the set of variables into independent smaller sets, sometimes sharing some non-relational information. We present a way to gain precision by keeping fully expressive relations between the subsets of variables, whilst retaining a linear complexity ensuring scalability.

Keywords: Abstract interpretation, abstract numerical domains, weakly relational domains.

1 Motivation

Abstract interpretation [4] is a theory of sound approximation of semantics of programming languages, mainly used in static analysis and verification of programs. A crucial point when designing an abstract interpretation-based analyzer is the choice of suitable abstract domains.

The domain of *intervals* was presented as one of the first abstract domains [3]. Although, even when the properties of interest are expressible with intervals, relational properties may be needed to compute them precisely. Shortly after, some of the limitations of intervals were overcome thanks to the domain of *polyhedra* [6]. While very precise, polyhedra are too costly and cannot be reasonably used to analyze programs with more than a few variables. Since then, numerous numerical abstract domains were designed to capture various properties, among them linear inequalities. Weakly relational domains restrict the shape of representable relations to achieve better computation times than polyhedra.

[1] Email: mehdi.bouaziz@ens.fr

doi:10.1016/j.entcs.2012.09.005

Between intervals and polyhedra, one can now find: pentagons [11], zones (or DBM) [12], weighted hexagons [8], octagons [13], logahedra [9], TVPI [16], octahedra [2], and templates [14].

Even the operations of the pentagon domain have a quadratic cost [2], which does not scale up [1,5]. Only a cost close to linear is acceptable when it comes to analyzing large programs.

Instead of looking for other shapes of relations, an orthogonal axis of research to the everlasting cost-precision trade-off is to modify the constraint graph (Def. 3.3), i.e., to restrain the set of variables between which there can be relations.

This idea has already been applied in Astree [1,5] and in C Global Surveyor [17], both analyzing large avionic and aerospace C programs. The set of variables is divided into several subsets called packs on which operators of the octagon domain are applied independently. However only non-relational information (i.e., intervals) are shared between packs.

The paper makes the following contributions:

- Several existing numerical domains are grouped together and generalized into a theoretical framework of linear inequality domains (Sect. 2).
- A domain functor (TreeKs, Sect. 3) is defined. It can be applied to any linear inequality domain to make a new scalable domain. The set of relations expressible in the new domain is a subset of those expressible in the underlying domain. The restriction is not made on the shape of the relations but on the variables between which the relations are defined.
- Taking advantage of the specific shape of the relation graph, an efficient completion algorithm (Sect. 4) for TreeKs domains is presented, along with its correctness proof. If packs are kept bounded, the cost of the domain remains linear.
- Finally, efficient algorithms (Sect. 5) are given for other operations required by abstraction interpretation.

2 Linear Inequality Domains

In this section we present theoretical properties of linear inequality domains, on which we will be able to apply our functor in the next section. This generalizes the work of Simon, King, and Howe on TVPI [16]

Let $\mathcal{V} \stackrel{\text{def}}{=} \{X_1, \ldots, X_N\}$ be a finite set of variables over $\mathbb{Q}$. Let $\mathbb{Q}^\star \stackrel{\text{def}}{=} \mathbb{Q} \setminus \{0\}$ denote non-zero and $\mathbb{Q}^\star_+ \stackrel{\text{def}}{=} \{x \in \mathbb{Q} \,|\, x > 0\}$ positive rational numbers.

Let $S^{\{n\}}$ denote the set of n-tuples made of members of S. If S is a set of tuples, let $\mathbb{Q}^\star S \stackrel{\text{def}}{=} \{\langle qx_1, \ldots, qx_k\rangle \,|\, \langle x_1, \ldots, x_k\rangle \in S, q \in \mathbb{Q}^\star\}$ denote the scalar multiplication of all members of S by a scalar of $\mathbb{Q}^\star$.

Definition 2.1 A *base set* $\mathscr{D}$ of a linear inequality domain is a subset of $\bigcup_{i\geq 1}(\mathbb{Q}^\star)^{\{i\}}$ closed by subset and closed by multiplication by a positive scalar of $\mathbb{Q}^\star_+$. It is said

[2] The cost of a domain is the amortized complexity, in the number of variables, of its operations. For weakly relational domain, this cost is dominated by the cost of the completion.

to be k-relational if it is a subset of $\bigcup_{i=1}^{k}(\mathbb{Q}^\star)^{\{i\}}$.

Base sets of known linear inequality domains are the following:

$$\begin{aligned}
\mathscr{I}ntervals &\stackrel{\text{def}}{=} (\mathbb{Q}^\star)^{\{1\}}\\
\mathscr{Z}ones &\stackrel{\text{def}}{=} (\mathbb{Q}^\star)^{\{1\}} \cup \mathbb{Q}^\star\{\langle 1,-1\rangle\}\\
\mathscr{O}ctagons &\stackrel{\text{def}}{=} (\mathbb{Q}^\star)^{\{1\}} \cup \mathbb{Q}^\star\{\langle 1,-1\rangle, \langle 1,1\rangle\}\\
\mathscr{L}ogahedra_B &\stackrel{\text{def}}{=} (\mathbb{Q}^\star)^{\{1\}} \cup \mathbb{Q}^\star\left\{\left\langle \pm 1, \pm 2^k\right\rangle \,\middle|\, -B \le k \le B\right\} \text{ with } B \in \mathbb{N}\cup\{+\infty\}\\
\mathscr{T}VPI &\stackrel{\text{def}}{=} (\mathbb{Q}^\star)^{\{1\}} \cup (\mathbb{Q}^\star)^{\{2\}}\\
\mathscr{O}ctahedra &\stackrel{\text{def}}{=} \bigcup_{k\ge 1} \mathbb{Q}^\star\{-1,1\}^{\{k\}}\\
\mathscr{P}olyhedra &\stackrel{\text{def}}{=} \bigcup_{k\ge 1} (\mathbb{Q}^\star)^{\{k\}}
\end{aligned}$$

Definition 2.2 The set of *inequalities* (or relations, or constraints) *of a domain* based on $\mathscr{D}$ over the set of variables $\mathcal{V}$ is defined:

$$\mathscr{D}_{\mathcal{V}} \stackrel{\text{def}}{=} \{\mathsf{True}, \mathsf{False}\} \cup \bigcup_{k\ge 1}\left\{\sum_{i=1}^{k} a_i x_i \le d \,\middle|\, \langle a_1,\ldots,a_k\rangle \in \mathscr{D}, x_i \in \mathcal{V}, d \in \mathbb{Q}\right\}$$

with $\mathsf{True} \stackrel{\text{def}}{=} 0 \le 1$ and $\mathsf{False} \stackrel{\text{def}}{=} 0 \le -1$.

Definition 2.3 The set of *values of a domain* based on $\mathscr{D}$ over the set of variables $\mathcal{V}$ is defined by $\mathcal{D}_{\mathcal{V}} \stackrel{\text{def}}{=} \mathcal{P}^f(\mathscr{D}_{\mathcal{V}})$, the finite sets of constraints of $\mathscr{D}_{\mathcal{V}}$.

Definition 2.4 If $c = \sum_{i=1}^{n} a_i x_i \le d$, and if $c \not\equiv \mathsf{True}$ and $c \not\equiv \mathsf{False}$, then the *concretization* of c is a half-space of $\mathbb{Q}^N$, the set of values satisfying the constraint c:

$$[\![c]\!] \stackrel{\text{def}}{=} \left\{(U_1,\ldots,U_N) \in \mathbb{Q}^N \,\middle|\, \sum_{i=1}^{n} a_i u_i \le d, x_j = X_i \Rightarrow u_j = U_i\right\}$$

$$[\![\emptyset]\!] = [\![\{\mathsf{True}\}]\!] = [\![\mathsf{True}]\!] \stackrel{\text{def}}{=} \mathbb{Q}^N \qquad [\![\mathsf{False}]\!] \stackrel{\text{def}}{=} \emptyset$$

The concretization of a value $C = \{c_1,\ldots,c_n\}$ is the intersection of the concretization of its members $[\![C]\!] \stackrel{\text{def}}{=} \bigcap_{i=1}^{n} [\![c_i]\!]$.

Definition 2.5 The set of values $\mathcal{D}_{\mathcal{V}}$ is ordered by *entailment*: $C_1 \vDash C_2 \stackrel{\text{def}}{\Longleftrightarrow} [\![C_1]\!] \subseteq [\![C_2]\!]$. Equivalence is defined as $C_1 \equiv C_2 \stackrel{\text{def}}{\Longleftrightarrow} C_1 \vDash C_2$ and $C_2 \vDash C_1$.

Definition 2.6 The *variables* of a linear inequality is the set of variables for which the coefficient is not zero. If $c = a_0X_0 + \ldots + a_NX_N \le d$, then $vars(c) \stackrel{\text{def}}{=} \{X_i \in \mathcal{V} \,|\, a_i \ne 0\}$. For a set of linear inequalities, $vars(C) \stackrel{\text{def}}{=} \bigcup_{c\in C} vars(c)$.

Definition 2.7 Let $\mathcal{X} \subseteq \mathcal{V}$ be a set of variables, the *restriction* of a value to this set of variables is defined: $\pi_{\mathcal{X}}(C) \stackrel{\text{def}}{=} \{c \in C \,|\, vars(c) \subseteq \mathcal{X}\}$.

Definition 2.8 A *forget operator* (or *projection operator*) of $\mathcal{D}_{\mathcal{V}}$ is a mapping $\exists : \mathcal{P}(\mathcal{V}) \longrightarrow {\mathcal{D}_{\mathcal{V}}}^{\mathcal{D}_{\mathcal{V}}}$ such that $\forall\, C_1 \in \mathcal{D}_{\mathcal{V}}$, $\forall\, \mathcal{X} = \{X_{i_1}, \ldots, X_{i_n}\} \subseteq \mathcal{V}$, we have $\exists_{\mathcal{X}}(C_1) = C_2$ such that $vars(C_2) \subseteq \mathcal{V} \setminus \mathcal{X}$ and

$$[\![C_2]\!] = \{(u_1, \ldots, u_{i_1-1}, a_1, u_{i_1+1}, \ldots, u_{i_n-1}, a_n, u_{i_n+1}, \ldots, u_N) \mid a_1, \ldots, a_n \in \mathbb{Q}, (u_1, \ldots, u_N) \in [\![C_1]\!]\}$$

Definition 2.9 A domain $\mathcal{D}_{\mathcal{V}}$ is *stable by elimination of variables* if $\forall\, c_1, c_2 \in \mathcal{D}_{\mathcal{V}}$ such that $c_1 = \sum_{i=1}^{N} a_i X_i \leq d_1$, $c_2 = \sum_{i=1}^{N} b_i X_i \leq d_2$ and if $\exists\, j$ such that $1 \leq j \leq N$, $a_j < 0$ and $b_j > 0$, we have $(b_j c_1 + a_j c_2) \in \mathcal{D}_{\mathcal{V}}$.

The domains $\mathcal{I}ntervals$, $\mathcal{Z}ones$, $\mathcal{O}ctagons$, $\mathcal{L}ogahedra_{\infty}$, $\mathcal{T}VPI$ as well as $\mathcal{P}olyhedra$ are stable by elimination of variables.

If $|\mathcal{V}| \geq 3$ then the domains $\mathcal{O}ctahedra$ and $\mathcal{L}ogahedra_B$ with $1 \leq B < \infty$ are not stable by elimination of variables [3] [4].

If $\mathcal{D}_{\mathcal{V}}$ is stable by elimination of variables then Fourier-Motzkin elimination is a forget operator.

Abstraction

There is no best abstraction on $\mathbb{Q}$ (e.g., for $X \times X \leq 2$). We will call $\bar{\alpha}$ the partial abstraction defined, when it makes sense, by the topological closure of the convex hull.

Definition 2.10 The *intersection* between values of $\mathcal{D}_{\mathcal{V}}$ is exact and is defined by the union of the inequalities set: $C_1 \sqcap^{\mathcal{D}_{\mathcal{V}}} C_2 \stackrel{\text{def}}{=} C_1 \cup C_2$.

The *union* between values is defined as the best abstraction: $C_1 \sqcup^{\mathcal{D}_{\mathcal{V}}} C_2 \stackrel{\text{def}}{=} \bar{\alpha}([\![C_1]\!] \cup [\![C_2]\!])$ [5].

The set $(\mathcal{D}_{\mathcal{V}}/\equiv, \vDash, \sqcup^{\mathcal{D}_{\mathcal{V}}}, \sqcap^{\mathcal{D}_{\mathcal{V}}}, \mathsf{False}, \mathsf{True})$ is a lattice.

3 The Domain Functor

In this section we build a new numerical abstract domain [6] based on an existing relational numerical domain, such as zones, octagons, logahedra, TVPI, octahedra, or polyhedra. Note that our construction will also work for other numerical domains that fall within the framework of the previous section.

3.1 Underlying Domain Properties

Suppose that the underlying domain is a numerical abstract domain over $\mathbb{Q}$, that is, it provides the following mathematical objects:

[3] Indeed $X_1 + X_2 \leq 0$ and $-X_1 + X_2 + X_3 \leq 0$ are in $\mathcal{O}ctahedra$ but their sum $2X_2 + X_3 \leq 0$ is not.

[4] Indeed $X_1 - 2^B X_2 \leq 0$ and $X_2 - 2X_3 \leq 0$ are in $\mathcal{L}ogahedra_B$ but their normalized sum $X_1 - 2^{B+1} X_3 \leq 0$ is in $\mathcal{L}ogahedra_{B+1}$ but not in $\mathcal{L}ogahedra_B$.

[5] Note that $\bar{\alpha}$ is always defined for such an input since the union of a finite number of polyhedra has a finite number of vertices and generators.

[6] Our domain is called TreeKs since, for a 2-relational underlying domain, the relation graph (Def. 3.3) is a tree of complete graphs (generally denoted by K).

- a base set $\mathscr{D} \ni \langle 1, -1 \rangle$, i.e., the domain contains zones, in particular equalities are representable,
- its associated domain $\mathcal{D}_\mathcal{V}$, with a computer representation D of its members,
- an effective algorithm to compare abstract values,
- effective algorithms to compute: exact variable elimination $\exists^{\mathcal{D}_\mathcal{V}}$ and intersection $\sqcap^{\mathcal{D}_\mathcal{V}}$, sound abstraction of union $\sqcup^{\mathcal{D}_\mathcal{V}}$, widening $\triangledown^{\mathcal{D}_\mathcal{V}}$ and possibly narrowing $\triangle^{\mathcal{D}_\mathcal{V}}$.

3.2 Packs and Graphs

In order to ensure scalability to large variable sets, we want to restrain the domain $\mathcal{D}$ to some chosen relations instead of all relations expressible in the domain.

Definition 3.1 A *pack set* $P = \{P_1, \ldots, P_m \mid P_i \subseteq \mathcal{V}\}$ is a set of subsets of variables of $\mathcal{V}$, such that $\bigcup_{i=1}^m P_i = \mathcal{V}$, and for all $i \neq j$, we have $P_i \nsubseteq P_j$.

Definition 3.2 The *pack graph* is defined by $\mathcal{G}_P \stackrel{\text{def}}{=} (P, F)$ where $(P_i, P_j) \in F$ if and only if $i \neq j$ and $P_i \cap P_j \neq \emptyset$. We will call any non-empty set $P_i \cap P_j$ a *frontier*.

Moreover we demand that the graph pack is a tree, i.e., there exists exactly one path from a pack to another pack in this graph.

This implies that a variable only appears in at most two packs. To make a variable appear in three packs P_1, P_2, P_3 (in the pack graph P_1—P_2—P_3), we can make a copy of it in P_2 and keep an equality constraint between these two instances.

We will use the following variables: $m \stackrel{\text{def}}{=} |P| \leq N$ the number of packs; $p \stackrel{\text{def}}{=} \max_{1 \leq i \leq m} |P_i| \leq N$ the size of the largest pack; $f \stackrel{\text{def}}{=} \max_{1 \leq i < j \leq m} |P_i \cap P_j| \leq p$ the size of the largest frontier; and $d \leq m$ the diameter of the pack tree.

3.3 The Functor

Given a underlying domain $\mathscr{D}$ and a pack set P, inequalities and values of our new domain are respectively defined:

$$\mathscr{T}reeKs_P^{\mathscr{D}} \stackrel{\text{def}}{=} \{c \in \mathscr{D}_\mathcal{V} \mid \exists\, i, vars(c) \subseteq P_i\} \qquad \mathcal{T}reeKs_P^{\mathcal{D}} \stackrel{\text{def}}{=} \mathcal{P}^f(\mathscr{T}reeKs_P^{\mathscr{D}})$$

The set $(\mathcal{T}reeKs_P^{\mathcal{D}}/\equiv, \vDash, \sqcup^{\mathcal{T}reeKs_P^{\mathcal{D}}}, \sqcap^{\mathcal{D}_\mathcal{V}}, \mathsf{False}, \mathsf{True})$ is a lattice.

Definition 3.3 The *constraint hypergraph* of a value C of $\mathcal{T}reeKs_P^{\mathcal{D}}$ is representation with constraints stored in edges. It is defined by $\mathcal{H}_P(C) \stackrel{\text{def}}{=} (\mathcal{V}, E, \ell)$ where non-oriented hyperedges are $E \stackrel{\text{def}}{=} \bigcup_{i=1}^m \{\mathcal{X} \subseteq P_i \mid \exists\, c \in \mathcal{D}_\mathcal{V}, vars(c) = \mathcal{X}\}$ and ℓ is a labelling of the hyperedges $\ell(e) \stackrel{\text{def}}{=} \{c \in C \mid vars(c) = e\}$.

3.4 The Representation Functor

Representing values as constraint hypergraphs could be well suited if the operations of the underlying domain are working on a dense graph representation, e.g., TVPI. However, for octagons, it is preferable to keep the original representation [13] where

each variable uses two vertices and constraints are stored in a half adjacency matrix. In order to make our functor independant of the underlying domain, we will keep the value representation of the underlying domain, associating an abstract value to each pack.

Thus, relations will appear twice at frontiers, our representation will be redundant but more efficient to use in algorithms.

We define this new domain representation as the cartesian product of the representation D of the underlying domain, restricted to subsets of variables corresponding to the packs of P, where all $\bot$ are merged into a single one:

$$\mathsf{D}_P \stackrel{\text{def}}{=} (\mathsf{D}_{P_1} \setminus \{\bot^{D_{P_1}}\}) \times \ldots \times (\mathsf{D}_{P_m} \setminus \{\bot^{D_{P_m}}\}) \cup \{\bot_{\mathsf{D}_P}\}$$

The set $(\mathsf{D}_P/\equiv_{\mathsf{D}_P}, \sqsubseteq_{\mathsf{D}_P}, \sqcup_{\mathsf{D}_P}, \sqcap_{\mathsf{D}_P}, \bot_{\mathsf{D}_P}, \top_{\mathsf{D}_P})$ is a lattice.

$\mathbf{v} \sqsubseteq_{\mathsf{D}_P} \mathbf{w} \iff^{\text{def}} \forall i, v_i \sqsubseteq_{\mathsf{D}_{P_i}} w_i$ $\qquad$ $(\top_{\mathsf{D}_P})_i \stackrel{\text{def}}{=} \top_{\mathsf{D}_{P_i}}$

$\mathbf{v} \equiv_{\mathsf{D}_P} \mathbf{w} \iff^{\text{def}} \mathbf{v} \sqsubseteq_{\mathsf{D}_P} \mathbf{w} \sqsubseteq_{\mathsf{D}_P} \mathbf{v}$ $\qquad$ $\bot_{\mathsf{D}_P} \sqsubseteq_{\mathsf{D}_P} \mathbf{x}$

$(\mathbf{v} \sqcup_{\mathsf{D}_P} \mathbf{w})_i \stackrel{\text{def}}{=} v_i \sqcup_{\mathsf{D}_{P_i}} w_i$ $\qquad$ $\bot_{\mathsf{D}_P} \sqcup_{\mathsf{D}_P} \mathbf{x} \stackrel{\text{def}}{=} \mathbf{x} \sqcup_{\mathsf{D}_P} \bot_{\mathsf{D}_P} \stackrel{\text{def}}{=} \mathbf{x}$

$(\mathbf{v} \sqcap_{\mathsf{D}_P} \mathbf{w})_i \stackrel{\text{def}}{=} v_i \sqcap_{\mathsf{D}_{P_i}} w_i$ $\qquad$ $\bot_{\mathsf{D}_P} \sqcap_{\mathsf{D}_P} \mathbf{x} \stackrel{\text{def}}{=} \mathbf{x} \sqcap_{\mathsf{D}_P} \bot_{\mathsf{D}_P} \stackrel{\text{def}}{=} \bot_{\mathsf{D}_P}$

Definition 3.4 Since this representation is redundant, we will say that a value $\mathbf{v} = \langle v_1, \ldots, v_m \rangle$ is *coherent* if and only if constraints coincide at frontiers, i.e., for all i, j, we have $\pi_{P_i \cap P_j}(v_i) \equiv \pi_{P_i \cap P_j}(v_j)$.

If $\exists^{\mathcal{D}}$ and $\cap^{\mathcal{D}}$ are exact then for any value $\mathbf{v}$, a coherent value $coh(\mathbf{v})$ can be built such that $\mathbf{v} \equiv coh(\mathbf{v})$. Indeed, simply do for all i, j, $v_i \leftarrow v_i \cap^{\mathcal{D}} \pi_{P_i \cap P_j}(v_j)$.

4 Completion

The completion operation aims at making explicit the implicit relations. For weakly relational domains, it is needed for most of the other operations, that is why its cost dominates the efficiency of the domain.

In our case, completion has an extra goal: to transfer information between the different packs.

Definition 4.1 A value C of $\mathcal{P}olyhedra_{\mathcal{V}}$ is said to be $\mathscr{D}$*-complete* if for all $c \in \mathscr{D}_{\mathcal{V}}$, $C \vDash c$ implies $\pi_{vars(c)}(C) \vDash c$.

Definition 4.2 A domain $\mathcal{D}$ is said to be *completable* if for all value C of $\mathcal{D}_{\mathcal{V}}$, there exists $C' \in \mathcal{D}_{\mathcal{V}}$ such that $C' \equiv C$ and C' is $\mathscr{D}$-complete.

If $\mathcal{D}$ is completable, let $\mathcal{D}'$ denote the set of its $\mathscr{D}$-complete values. If $\mathcal{D}$ owns a completion operation, let `complete` denote this function. Otherwise, let $\mathcal{D}'$ equal $\mathcal{D}$ and `complete` be the identity function.

An easy way to complete a value $V = \langle C_1, \ldots, C_m \rangle \in \mathsf{D}_P$ is to use the completion of $\mathcal{D}_{\mathcal{V}}$. Let $V^\star = \texttt{complete}^{\mathcal{D}_{\mathcal{V}}}(C_1 \cup \ldots \cup C_m)$. Then $V' = \langle \pi_{P_1}(V^\star), \ldots, \pi_{P_m}(V^\star) \rangle \in \mathsf{D}'_P$ and $V' \equiv V$. While this completion is correct, it is more expensive than the

completion of the underlying domain so there would be no point in restricting to a subgraph of relations.

However a pointwise completion is not sufficient: suppose that $\langle C_1, C_2\rangle \in \mathcal{Z}ones_{\{P_1,P_2\}}, P_1 = \{x, y, z\}, P_2 = \{y, z, t\}, C_1 = \{x \leq y, z \leq x\}$ and $C_2 = \{y \leq t\}$. $\texttt{complete}(C_1) = \{x \leq y, z \leq x, z \leq y\}$ and $\texttt{complete}(C_2) = \{y \leq t\}$. Whereas $\texttt{complete}(C_1 \cup C_2) = \{x \leq y, y \leq t, z \leq x, z \leq y, z \leq t\}$. Completion over P_1 provides information that must be injected into C_2 and vice versa.

Theorem 4.4, whose proof is based on Farkas' lemma, shows that nevertheless exchanges between packs can be kept limited.

Lemma 4.3 ***(Generalized Farkas' Lemma)***

Let E be a finite-dimensional affine space on a field $\mathbb{K}$. Let $f_1, \ldots, f_k$ and g be affine functionals on E, such that $\{x \in E \mid f_1(x) \geq 0, \ldots, f_k(x) \geq 0\}$ is non empty.

Then $\{x \in E \mid f_1(x) \geq 0, \ldots, f_k(x) \geq 0\} \subseteq \{x \in E \mid g(x) \geq 0\}$ if and only if there exists $\alpha_1, \ldots, \alpha_k, \beta \geq 0$ such that $g = \sum_{i=0}^{k} \alpha_i f_i + \beta$.

A proof can be found in standard references [15].

Theorem 4.4 *Let $C \in \mathcal{D}'_{\mathcal{V}}$.*

Let $\mathcal{X} \subseteq \mathcal{V}$.

Let $C^+ \in \mathcal{D}_{\mathcal{X}}$ such that $(\pi_{\mathcal{X}}(C) \cup C^+) \in \mathcal{D}'_{\mathcal{X}}$.

Let $C^\cup = C \cup C^+$.

Let $C' \in \mathcal{D}'_{\mathcal{V}}$ such that $C' \equiv C^\cup$, e.g., $C' = \texttt{complete}(C^\cup)$.

Then for all $\mathcal{Y} \subseteq \mathcal{X}$, $\pi_{\mathcal{Y}}(C') \equiv \pi_{\mathcal{Y}}(C^\cup)$.

Proof On one side, we have $C' \vDash C^\cup$, in particular $C' \vDash \pi_{\mathcal{Y}}(C^\cup)$. But C' is $\mathscr{D}$-complete and $\pi_{\mathcal{Y}}(C^\cup) \in \mathcal{D}_{\mathcal{V}}$, hence $\pi_{\mathcal{Y}}(C') \vDash \pi_{\mathcal{Y}}(C^\cup)$ (from Def. 4.1).

On the other side, the case $C^\cup \equiv \emptyset$ is trivial, so let suppose that $C^\cup \not\equiv \emptyset$.

Let $C^{\mathcal{X}} = \pi_{\mathcal{X}}(C^\cup) = \pi_{\mathcal{X}}(C) \cup C^+$ and $C^- = C^\cup \setminus C^\cup_{\mathcal{X}} \subseteq C$.

Let $c \in \mathscr{D}_{\mathcal{Y}}$ such that $\pi_{\mathcal{Y}}(C') \vDash c$. In particular $C' \vDash c$ and $C^\cup \vDash c$.

From Lemma 4.3, there exists $\alpha_0, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \mathbb{Q}_+$, $c_1^{\mathcal{X}}, \ldots, c_n^{\mathcal{X}} \in C^{\mathcal{X}}$, and $c_1^-, \ldots, c_m^- \in C^-$ such that $c = c^{\mathcal{X}} + c^-$ with $c^{\mathcal{X}} = \sum_{i=0}^{n} \alpha_i c_i^{\mathcal{X}}$, $c_0^{\mathcal{X}} = \mathsf{True}$, and $c^- = \sum_{j=1}^{m} \beta_j c_j^-$.

We successively have $C^- \vDash c^-$, $C \vDash c^-$, $\pi_{\mathcal{X}}(C) \vDash c^-$ (from Def. 4.1, since C is $\mathscr{D}$-complete and $vars(c^-) \subseteq vars(c) \cup vars(c^{\mathcal{X}}) \subseteq \mathcal{Y} \cup \mathcal{X} = \mathcal{X}$), and finally $C^{\mathcal{X}} \vDash c^-$. Moreover $C^{\mathcal{X}} \vDash c^{\mathcal{X}}$; summing gives $C^{\mathcal{X}} \vDash c$, and $\pi_{\mathcal{Y}}(C^{\mathcal{X}}) \vDash c$ because $C^{\mathcal{X}}$ is $\mathscr{D}$-complete. Thus for all $c \in \mathscr{D}_{\mathcal{Y}}$, $\pi_{\mathcal{Y}}(C') \vDash c$ implies $\pi_{\mathcal{Y}}(C^\cup) \vDash c$.

And we get $\pi_{\mathcal{Y}}(C^\cup) \vDash \pi_{\mathcal{Y}}(C')$ from:

$$[\![\pi_{\mathcal{Y}}(C^\cup)]\!] \subseteq \bigcap_{\substack{c \in \mathscr{D}_{\mathcal{Y}} \\ \pi_{\mathcal{Y}}(C^\cup) \vDash c}} [\![c]\!] \subseteq \bigcap_{\substack{c \in \mathscr{D}_{\mathcal{Y}} \\ \pi_{\mathcal{Y}}(C') \vDash c}} [\![c]\!] \subseteq \bigcap_{\substack{c \in \mathscr{D}_{\mathcal{Y}} \\ c \in \pi_{\mathcal{Y}}(C')}} [\![c]\!] = [\![\pi_{\mathcal{Y}}(C')]\!]$$

□

This theorem means that if we add new constraints C^+ to a complete value C, such that the result is complete on a subset of variables $\mathcal{X}$, and the result $C^\cup$ is

completed again, giving C', then contraints on $\mathcal{X}$ cannot be improved. The following algorithm uses this theorem to build completions efficiently.

Completion Algorithm

Let us choose arbitrarily a root in the pack tree and direct this tree such that arcs are directed from the root to the leaves. Suppose that the root is P_1 and that if there is an arc from P_i to P_j then $i < j$. Let $\texttt{father}(i)$ denote the pack father of P_i in this directed tree (undefined for P_1).

Function $\texttt{complete}^{\mathsf{D}_P}(\langle C_1, \ldots, C_m \rangle)$

for $i \leftarrow m$ **to** 2 **do** {from the leaves to the root}
  $C_i \leftarrow \texttt{complete}^{\mathcal{D}_{P_i}}(C_i)$
  if $C_i = \bot^{\mathcal{D}_{P_i}}$ **then return** $\bot^{\mathsf{D}_P}$
  $C_{\texttt{father}(i)} \leftarrow C_{\texttt{father}(i)} \cup \pi_{P_{\texttt{father}(i)}}(C_i)$
$C_1 \leftarrow \texttt{complete}^{\mathcal{D}_{P_1}}(C_1)$
if $C_1 = \bot^{\mathcal{D}_{P_1}}$ **then return** $\bot^{\mathsf{D}_P}$
for $i \leftarrow 2$ **to** m **do** {from the root to the leaves}
  $C_i \leftarrow C_i \cup \pi_{P_i}(C_{\texttt{father}(i)})$
  $C_i \leftarrow \texttt{complete}^{\mathcal{D}_{P_i}}(C_i)$
  if $C_i = \bot^{\mathcal{D}_{P_i}}$ **then return** $\bot^{\mathsf{D}_P}$
return $\langle C_1, \ldots, C_m \rangle$

Correctness

On one hand, Theorem 4.4 shows that completing packs back and forth only once is enough to ensure saturation. Any further $\texttt{complete}^{\mathcal{D}_{P_i}}$ on a pack will have no effect. This is why we chose a tree-shaped relation graph.

On the other hand, for all $C \in \mathsf{D}_P$, $\texttt{complete}^{\mathsf{D}_P}(C) \equiv C$. This holds at every step of the algorithm: this is obvious for $\texttt{complete}^{\mathcal{D}_{P_i}}$ steps; union steps do not change the concretization of a value since the union is made with constraints already present in the value; and checks for $\bot$ only make implicit $\bot$ explicit.

Complexity

Let A_p denote the cost of completing a pack of size $\leq p$ and $B_{p,f}$ denote the cost of the projection and union of a pack of size $\leq p$ on a pack of size $\leq p$, with a frontier of size $\leq f$.

The cost $C_{p,f}$ of the completion algorithm is bounded by:

$$A_{|P_1|} + 2\sum_{i=2}^{m} A_{|P_i|} + 2\sum_{i=2}^{m} B_{max(|P_i|,|P_{\texttt{father}(i)}|),|P_i \cap P_{\texttt{father}(i)}|} \leq 2m(A_p + B_{p,f})$$

For a bounded pack size, our completion algorithm has a linear complexity in the number of variables, whatever the underlying domain is. If the underlying domain owns an incremental completion [10], it can be used to replace the global

completion in the second loop. The algorithm will be faster but its complexity remains unchanged.

5 Abstract Operators

In this section we provide domain operations needed by abstract interpretation. Generally, operators on D_P will be easily defined from operators on $\mathcal{D}$. However, for operators on $\mathcal{D}$, each time a completion is needed, our completion will actually have to be used.

5.1 Operators on sets

In this section, operator arguments will be considered completed.

Inclusion and equality tests are pointwise on complete arguments. If they are exact on $\mathcal{D}$ then they are exact on D_P too.

Intersection being exact (it is a constraint union), it is extented pointwisely on each pack and remains exact.

If $\sqcup^{\mathcal{D}}$ is the best abstraction of *union* in $\mathcal{D}$ then $\sqcup^{\mathsf{D}_P}$, pointwise extension of $\sqcup^{\mathcal{D}}$ on each pack of P, is the best abstraction of union in D_P.

The *forget operator* is a projection on a space not containing some given variables. From a complete value, just remove all constraints involving these variables.

5.2 Widenings, Narrowings

Widenings of the underlying domain can be applied to each pack independently. Convergence of increasing chains in finite time is immediately ensured. However so formed values can be uncoherent or incomplete. But trying to make them coherent or complete can jeopardize converge [13]. Indeed widening relaxes constraints towards $+\infty$ whereas completion has an opposite goal, the same applies to coherence because it is obtained by intersection.

Similarily, narrowings of the underlying domain can be applied to each pack independently.

5.3 Constraint Extraction and Addition

We also provide two operations that are useful to build abstract interpretation-based analysis tools and that cannot be pointwisely extended from the underlying domain.

Constraint Extraction

Let $\langle C_1, \ldots, C_m \rangle$ denote a complete value. Suppose that we want to extract the set of constraints existing between variables from a set $\mathcal{X} \subseteq \mathcal{V}$, $1 \leq |\mathcal{X}| \leq N$. If all these variables are in the same pack (e.g., $|\mathcal{X}| = 1$ for interval extraction) then a simple projection is sufficient. Otherwise, things are more complicated. For each

pack P_i, let $\mathcal{X}_{(i)} \stackrel{\text{def}}{=} \mathcal{X} \cap P_i$, and suppose that $\mathcal{X}_{(1)} \neq \emptyset$.

Function $\texttt{extract}^{\mathsf{D}_P}(\langle C_1, \ldots, C_m \rangle, \mathcal{X})$

$V_{1..m} \leftarrow \emptyset$
$D_{1..m} \leftarrow \emptyset$
for $i \leftarrow m$ **to** 2 **do**
  $V_i \leftarrow V_i \cup \mathcal{X}_{(i)}$
  if $V_i \neq \emptyset$ **then**
    $V_{\texttt{father}(i)} \leftarrow V_{\texttt{father}(i)} \cup V_i$
    $D_{\texttt{father}(i)} \leftarrow D_{\texttt{father}(i)} \cup \pi_{V_i \cup (P_i \cap P_{\texttt{father}(i)})}(\texttt{complete}^{\mathcal{D}_{(P_i \cup V_i)}}(C_i \cup D_i))$
return $\pi_{\mathcal{X}}(\texttt{complete}^{\mathcal{D}_{(P_1 \cup \mathcal{X})}}(C_1 \cup D_1))$

This function has a cost bounded by $D_{p,f} = m(A_{p+|\mathcal{X}|} + B_{(p+|\mathcal{X}|),(f+|\mathcal{X}|)})$. For a bounded pack size and a bounded number of variables of interest, this function has a linear complexity in the total number of variables. If $|\mathcal{X}| = 2$, its cost can even be bounded by $D'_{p,f} = d(A_{p+2} + B_{(p+2),(f+2)})$, which is linear in the diameter of the pack tree.

Adding Constraints

Given a complete value $\langle C_1, \ldots, C_m \rangle$, suppose that we want to add new constraints C^+. If all the variables of the constraints to add are in a single pack then we can add the constraints to this pack only. Otherwise, we need to extract from C^+ other constraints that can be independently added to the packs, precisely all constraints that we can get from C^+ expressible in $\mathcal{T}reeKs^{\mathcal{D}}$. We keep the same notations, but now $\mathcal{X} = vars(C^+)$.

Function $\texttt{add}^{\mathsf{D}_P}(\langle C_1, \ldots, C_m \rangle, C^+)$

$V_{1..m} \leftarrow \emptyset$
$D_{1..m} \leftarrow \emptyset$
for $i \leftarrow m$ **to** 2 **do**
  $V_i \leftarrow V_i \cup \mathcal{X}_{(i)}$
  if $V_i \neq \emptyset$ **then**
    $V_i \leftarrow V_i \cup (P_i \cap P_{\texttt{father}(i)})$
    $V_{\texttt{father}(i)} \leftarrow V_{\texttt{father}(i)} \cup V_i$
    $D_{\texttt{father}(i)} \leftarrow D_{\texttt{father}(i)} \cup \pi_{V_i}(\texttt{complete}^{\mathcal{D}_{(P_i \cup V_i)}}(C_i \cup D_i))$
$D_0 \leftarrow \pi_{V_1}(\texttt{complete}^{\mathcal{D}_{(P_1 \cup V_1)}}(C_1 \cup D_1 \cup C^+))$
for $i \leftarrow 1$ **to** m **do**
  if $V_i \neq \emptyset$ **then** $C_i \leftarrow C_i \cup \pi_{P_i}(D_0)$
return $\langle C_1, \ldots, C_m \rangle$

This function has a worst-case cost bounded by $E_{p,f} = m(A_{N'} + B_{N',N'})$ where $N' = min(N, mf + p + |\mathcal{X}|)$. If $|\mathcal{X}| = 2$ then this cost is generally linear in the diameter of the graph and is bounded by $E'_{p,f} = d(A_{df+p} + B_{df+p})$. Therefore, adding constraints between distant variables should be avoided as much as possible.

6 Conclusion and Future Work

This paper has introduced TreeKs, a functor to make numerical abstract domain scalable, by restraining the relation graph to a specific shape allowing efficient algorithms for completion and abstract operations.

Like related work [1,17,5], it relies on packs of variables. Whereas they did not share relational information in previous work, it is made possible with TreeKs whilst retaining scalable.

Implementations are warmly welcome and comparisons with existing domains would be interesting. Theoretically, the domain obtained by applying TreeKs lies, for both precision and cost, between packs with only non-relational sharing and the underlying domain itself.

This paper does not describe how to generate packs. ASTREE [1,5] uses a syntactic criterion whereas C Global Surveyor [17] build them dynamically. Different software systems may require different packing strategies and coming to a decision will demand experimental comparisons.

Extensions for domains like pentagons and weighted hexagons, or generally any convex domain (e.g., ellipsoids [7]), seem conceivable with a more general framework. However it is unclear how TreeKs could efficiently be applied to non-convex domains.

References

[1] B. Blanchet, 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*, pages 196–207. ACM Press, June 2003.

[2] R. Clarisó and J. Cortadella. The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139, 2007.

[3] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. 2nd Int. Symp. on Programming*, pages 106–130, Paris, 1976. Dunod.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, 1977. ACM Press, New York.

[5] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does Astrée scale up? *Formal Methods in System Design*, 35(3):229–264, 2009.

[6] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th POPL*, pages 84–97, Tucson, 1978. ACM Press, NY.

[7] J. Feret. Numerical abstract domains for digital filters. In *International workshop on Numerical & Symbolic Abstract Domains*, 2005.

[8] J. Fulara, K. Durnoga, K. Jakubczyk, and A. Schubert. Relational abstract domain of weighted hexagons. *Electronic Notes in Theoretical Computer Science*, 267(1):59–72, 2010.

[9] J. Howe and A. King. Logahedra: A new weakly relational domain. *Automated Technology for Verification and Analysis*, pages 306–320, 2009.

[10] J.M. Howe and A. King. Closure Algorithms for Domains with Two Variables per Inequality. Technical report, School of Informatics, City University London, 2009.

[11] 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*, pages 184–188, 2008.

[12] A. Miné. A new numerical abstract domain based on difference-bound matrices. In *Proc. of the PADO II*, LNCS vol. 2053, pages 155–172, Aarhus, 2001. Springer.

[13] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[14] S. Sankaranarayanan, H.B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, pages 25–41. Springer, 2005.

[15] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons Inc, 1998.

[16] A. Simon, A. King, and J. Howe. Two variables per linear inequality as an abstract domain. *LOPSTR*, pages 955–955, 2003.

[17] A. Venet and G. Brat. Precise and efficient static array bound checking for large embedded C programs. In *PLDI'04*, pages 231–242, Washington, 2004. ACM Press.