Scalable Numerical Abstract Domains

Mehdi Bouaziz

École normale supérieure, Paris

Second Workshop on Analysis and Verification of Dependable Cyber Physical Software

November 23–24, 2013 – Changsha, China

Motivation

Numerical static analysis:

- automatic and static discovery of properties on the numerical variables of a program

Applications:

- static verification of programs
- invariant discovery
- program optimization

Context: abstract numerical domains

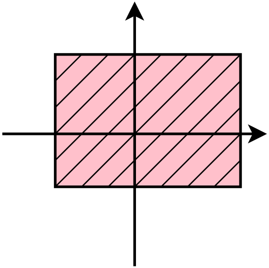
Abstract interpretation [Cousot Cousot 77] defines a formal framework of sound approximations of semantics.

A numerical abstract domain is:

- a set $\mathcal{D}_\mathcal{V}$ of computer-representable abstract values,
- a concretisation $[\![.]\!] : \mathcal{D}_\mathcal{V} \longrightarrow \mathcal{P}(\mathcal{V} \mapsto \mathbb{Q})$,
- a comparison algorithm $\sqsubseteq^{\mathcal{D}_\mathcal{V}}$ of abstract values,
- effective algorithms to compute sound abstractions of the operations: intersection $\sqcap^{\mathcal{D}_\mathcal{V}}$, union $\sqcup^{\mathcal{D}_\mathcal{V}}$, projection $\exists^{\mathcal{D}_\mathcal{V}}$, ...
- a widening $\nabla^{\mathcal{D}_\mathcal{V}}$ to ensure termination, if needed.

Numerical abstract domains: basics

Intervals [Cousot Cousot 76]

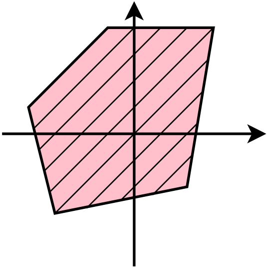


$\bigwedge_i a_i \leq X_i \leq b_i$

Non-relational

Linear cost

Polyhedra [Cousot Halbwachs 78]



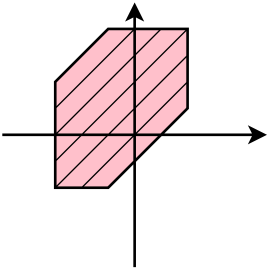
$\bigwedge_j \sum_i a_{ij} X_i \leq b_j$

Relational and very precise

Worst-case exponential cost

Weakly relational numerical abstract domains

Zones [Miné 01]



$\bigwedge_{ij} X_i - X_j \leq c_{ij}$

Weakly relational
Cubic cost

Octagons [Miné 01]
$\bigwedge_{ij} \pm X_i \pm X_j \leq c_{ij}$
Cubic cost

Logahedra [Howe King 09]
$\bigwedge_{ij} \pm 2^{a_i} X_i \pm 2^{b_j} X_j \leq c_{ij}$
Cubic cost

TVPI [Simon King Howe 02]
$\bigwedge_{ij} a_i X_i + b_j X_j \leq c_{ij}$
Quasi-cubic cost

Octahedra [Clarisó Cortadella 07]
$\bigwedge \sum_i \pm X_i \leq c$
Worst-case exponential cost

Why abstract domains do not scale up

Execution time of an analysis is roughly the multiplication of:

- the number of lines of codes,
- the number of variables ($\propto$ LOC),
- the number of iterations,
- the cost of each domain operation,
- hidden costs (garbage collection, cache database).

When analyzing programs with 10,000+ variables, you need the domain operations to have a linear cost.

Our contribution: TreeKs

- a domain functor
- applied to linear inequality domains
- with a configurable cost/precision tradeoff

Our contribution: TreeKs

- a domain functor
- applied to linear inequality domains
- with a configurable cost/precision tradeoff

Outline:

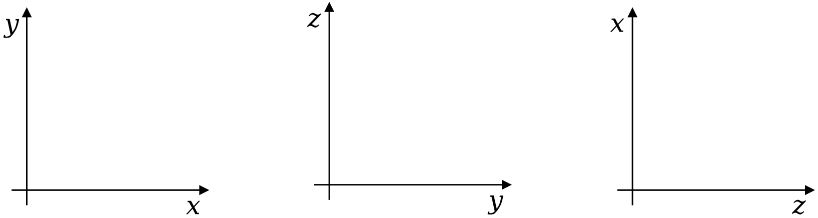
- the completion operation
- scaling up with packs
- application and optimizations for zones/octagons
- discussion of extensions

Completion: a key operation

- Common point of the weakly relational domains
- Goal: making explicit the implicit relations
- Done by constraint combination/propagation
- Needed for the other operations ($\sqcup$, $\sqcap$, $\sqsubseteq$, ...)
- Dominates the cost of the domain

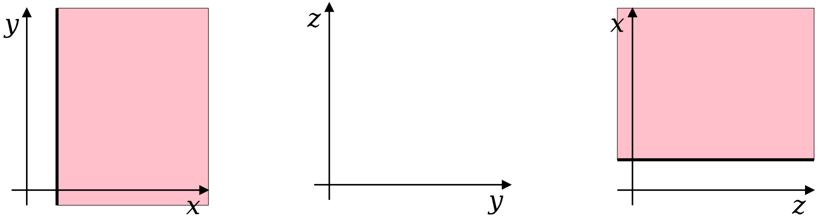
Closure operation: example

Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$
$\mathcal{V} = \{x, y, z\}$



Closure operation: example

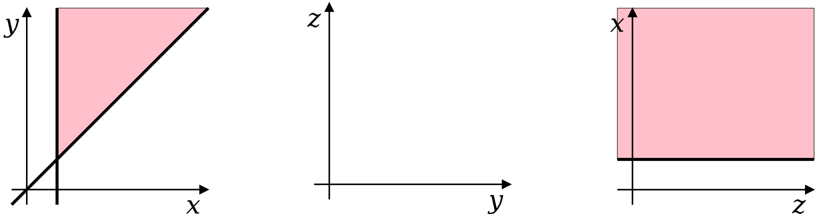
Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$
$\mathcal{V} = \{x, y, z\}$



$-x \leq -1$

Closure operation: example

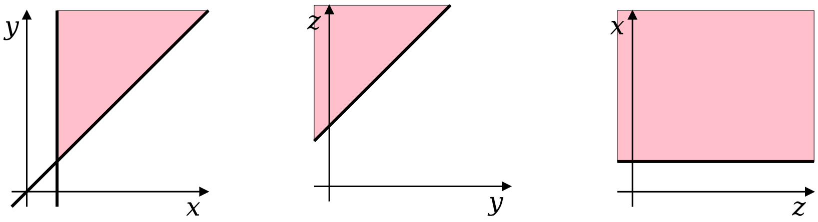
Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$
$\mathcal{V} = \{x, y, z\}$



$-x \leq -1$
$x - y \leq 0$

Closure operation: example

Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$
$\mathcal{V} = \{x, y, z\}$

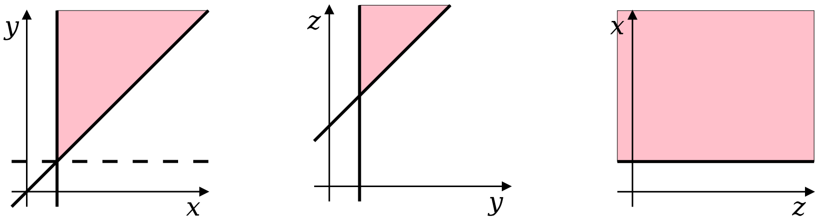


$-x \leq -1$
$x - y \leq 0$
$y - z \leq -2$

Closure operation: example

Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$

$\mathcal{V} = \{x, y, z\}$



$-y \leq -1$

$-x \leq -1$

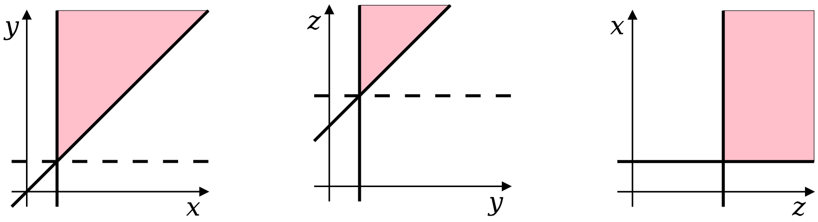
$x - y \leq 0$

$y - z \leq -2$

Closure operation: example

Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$

$\mathcal{V} = \{x, y, z\}$



$-x \leq -1$

$x - y \leq 0$

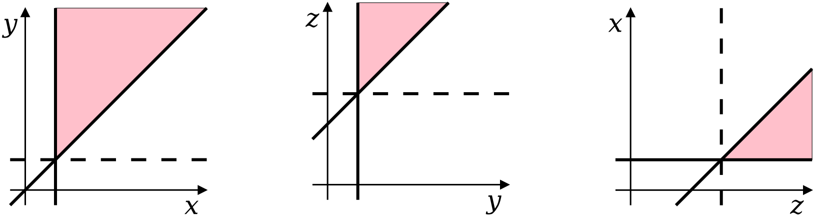
$y - z \leq -2$

$-y \leq -1$

$-z \leq -3$

Closure operation: example

Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$
$\mathcal{V} = \{x, y, z\}$

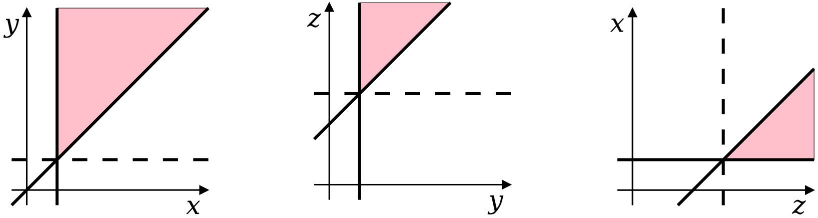


$-x \leq -1$
$x - y \leq 0$
$y - z \leq -2$

$-y \leq -1$
$-z \leq -3$
$x - z \leq -2$

Closure operation: example

Domain of zones $(\bigwedge_{ij} X_i - X_j \leq b_{ij})$
$\mathcal{V} = \{x, y, z\}$

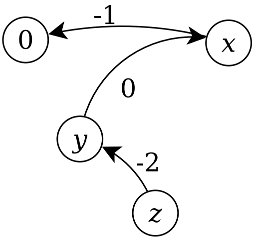


$-x \leq -1$
$x - y \leq 0$
$y - z \leq -2$

$-y \leq -1$
$-z \leq -3$
$x - z \leq -2$
Done!

Domain of zones: representation

We represent a set of difference constraints between two variables ($X_i - X_j \leq \mathbf{m}_{ji}$) by a potential graph or by a DBM (*Difference Bound Matrix*).

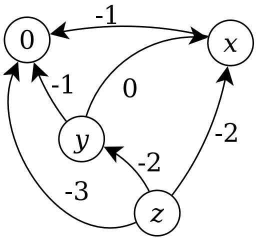


	0	x	y	z
0	0	$+\infty$	$+\infty$	$+\infty$
x	-1	0	$+\infty$	$+\infty$
y	$+\infty$	0	0	$+\infty$
z	$+\infty$	$+\infty$	-2	0

$$\begin{aligned} 0 - x &\leq -1 \\ x - y &\leq 0 \\ y - z &\leq -2 \end{aligned}$$

Domain of zones: representation

We represent a set of difference constraints between two variables ($X_i - X_j \leq \mathbf{m}_{ji}$) by a potential graph or by a DBM (*Difference Bound Matrix*).



	0	x	y	z
0	0	$+\infty$	$+\infty$	$+\infty$
x	-1	0	$+\infty$	$+\infty$
y	-1	0	0	$+\infty$
z	-3	-2	-2	0

$$\begin{aligned} 0 - x &\leq -1 \\ x - y &\leq 0 \\ y - z &\leq -2 \end{aligned}$$

$$\begin{aligned} 0 - y &\leq -1 \\ 0 - z &\leq -3 \\ x - z &\leq -2 \end{aligned}$$

Domain of zones: completion

In the domain of zones, the completion operation is a shortest-path closure.

Floyd-Warshall algorithm $O(n^3)$

for $k \leftarrow 1$ **to** N **do**
  for $i \leftarrow 1$ **to** N **do**
    for $j \leftarrow 1$ **to** N **do**
      $\mathbf{m}_{ij} \leftarrow \min(\mathbf{m}_{ij}, \mathbf{m}_{ik} + \mathbf{m}_{kj})$

At the end: $\begin{cases} \forall i, j, k, \mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj} & \text{if satisfiable} \\ \exists i, \mathbf{m}_{ii} < 0 & \text{if unsatisfiable} \end{cases}$

Domain of zones: operators

After completion, operators are pointwise.

Join (best approximation of union):

$$(\mathbf{m} \sqcup \mathbf{n})_{ij} = \max(\mathbf{m}_{ij}, \mathbf{n}_{ij})$$

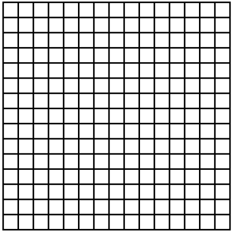
Forget operator (projection):

$$(\exists_{X_k} \mathbf{m})_{ij} = \begin{cases} \mathbf{m}_{ij} & \text{if } i \neq k \text{ and } j \neq k \\ 0 & \text{if } i = j = k \\ +\infty & \text{otherwise} \end{cases}$$

How to scale up: variable packing

Principle:

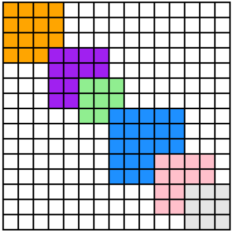
- split variables into packs
- use a DBM per pack



How to scale up: variable packing

Principle:

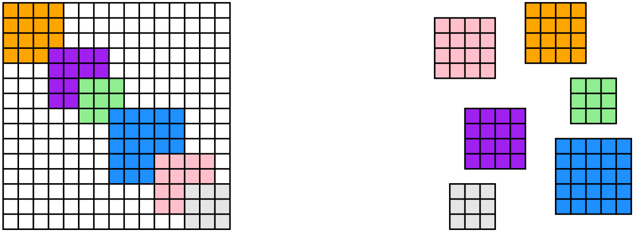
- split variables into packs
- use a DBM per pack



How to scale up: variable packing

Principle:

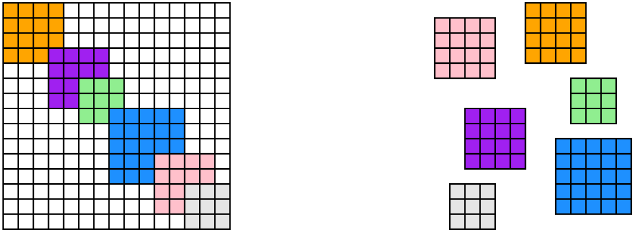
- split variables into packs
- use a DBM per pack



How to scale up: variable packing

Principle:

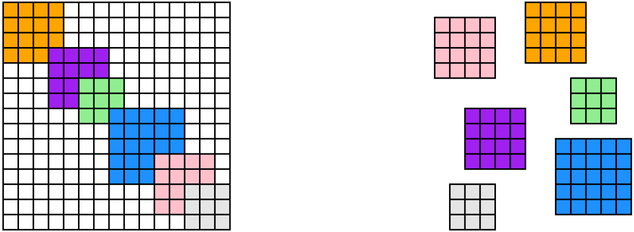
- split variables into packs
- use a DBM per pack



Cost: linear for bounded-size packs

How to scale up: variable packing

Principle:
- split variables into packs
- use a DBM per pack



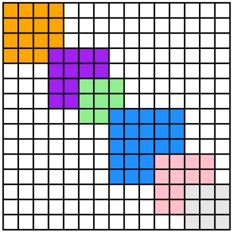
Cost: linear for bounded-size packs

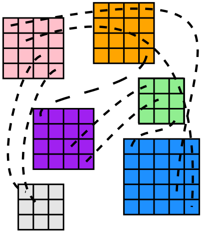
Information loss: no communication between packs!

How to scale up: variable packing

Principle:

- split variables into packs
- use a DBM per pack





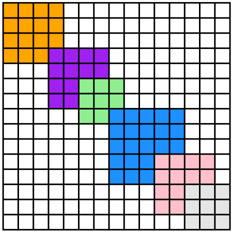
Cost: linear for bounded-size packs

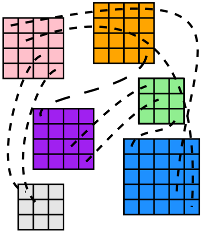
Information loss: no communication between packs!

Solution: intervals constraints sharing

How to scale up: variable packing

Principle:
- split variables into packs
- use a DBM per pack





Cost: linear for bounded-size packs

Information loss: no communication between packs!

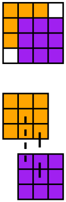
Solution: intervals constraints sharing

Not good enough!

How to scale up: variable packing

Principle:

- split variables into packs
- use a DBM per pack



$P_1 = \{t, x, y\}$

$t \leq y$
$y \leq x$

$P_2 = \{t, x, z\}$

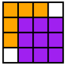
$x \leq z$
$z \leq t$

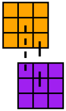
Cost: linear for bounded-size packs
Information loss: no communication between packs!
Solution: intervals constraints sharing
Not good enough!

How to scale up: variable packing

Principle:

- split variables into packs
- use a DBM per pack





$P_1 = \{t, x, y\}$	$P_2 = \{t, x, z\}$
$t \leq y$	$x \leq z$
$y \leq x$	$z \leq t$
$t \leq x$	$x \leq t$

Cost: linear for bounded-size packs

Information loss: no communication between packs!

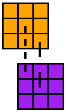
Solution: intervals constraints sharing

Not good enough!

How to scale up: variable packing

Principle:

- split variables into packs
- use a DBM per pack



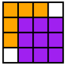
$P_1 = \{t, x, y\}$	$P_2 = \{t, x, z\}$
$t \leq y$	$x \leq z$
$y \leq x$	$z \leq t$
$t \leq x$	$x \leq t$

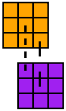
Cost: linear for bounded-size packs
Information loss: no communication between packs!
Solution: intervals constraints sharing
Not good enough!

How to scale up: variable packing

Principle:

- split variables into packs
- use a DBM per pack





$P_1 = \{t, x, y\}$

$t \leq y$
$y \leq x$
$t \leq x$

$x = t$

$P_2 = \{t, x, z\}$

$x \leq z$
$z \leq t$
$x \leq t$

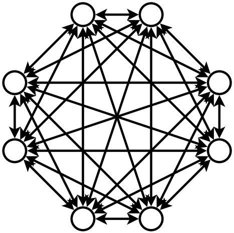
Cost: linear for bounded-size packs
Information loss: no communication between packs!
Solution: intervals constraints sharing
Not good enough!

An idea: a subgraph

Goal: share relational constraints

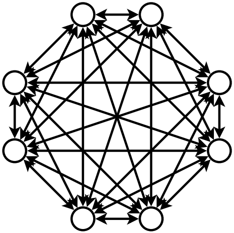
An idea: a subgraph

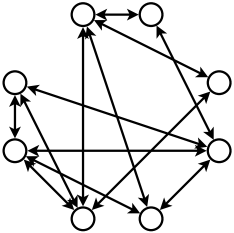
Goal: share relational constraints

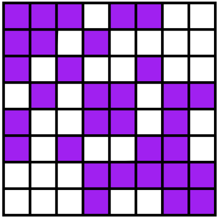


An idea: a subgraph

<u>Goal</u>: share relational constraints

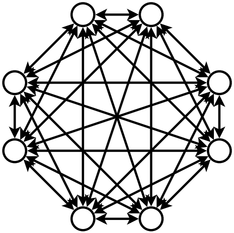


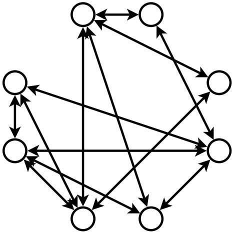


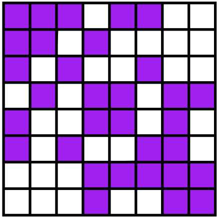


An idea: a subgraph

Goal: share relational constraints



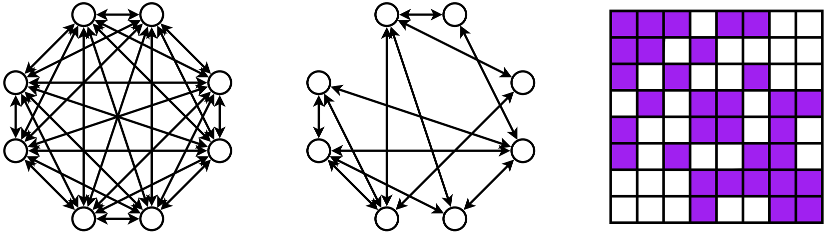




Issues: we need to keep

An idea: a subgraph

Goal: share relational constraints



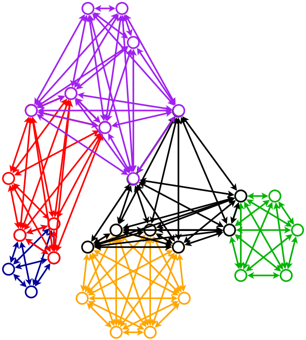
Issues: we need to keep

- a good expressiveness
- a structure with packs
- precise and efficient algorithms

TreeKs: a certain subgraph

Shape:

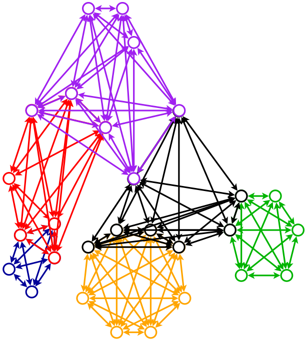
- a tree of complete graphs (packs)
- sharing frontiers

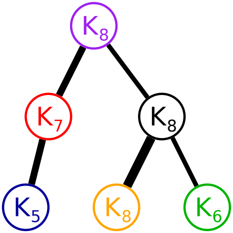


TreeKs: a certain subgraph

Shape:

- a tree of complete graphs (packs)
- sharing frontiers



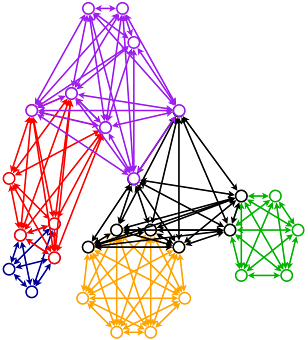


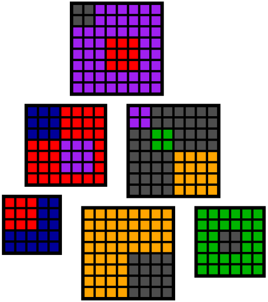
packs tree

TreeKs: a certain subgraph

Shape:

- a tree of complete graphs (packs)
- sharing frontiers



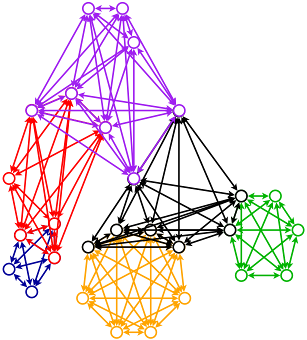


Abstract value: tuple of DBMs

TreeKs: a certain subgraph

Shape:

- a tree of complete graphs (packs)
- sharing frontiers



Parameters:

N	number of variables
m	number of packs
p	size of a pack
f	size of a frontier
d	diameter of the graph

TreeKs: abstract operators

On complete values, all operations can be done pointwisely:

- inclusion test
- intersection
- union

but contraint extraction and addition. . .

Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**

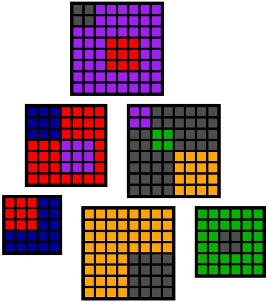
  Apply completion on this pack in the domain of zones

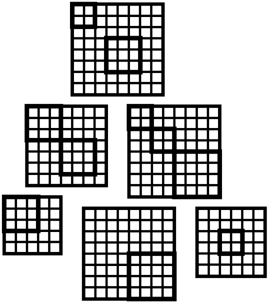
  Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**

  Apply completion on this pack in the domain of zones

  Pass the new constraints to its children



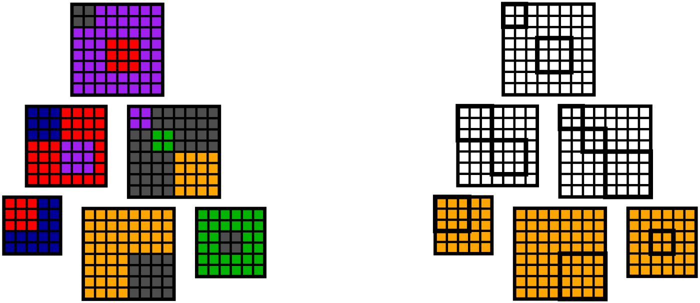


Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**
  Apply completion on this pack in the domain of zones
  Pass the new constraints to its father

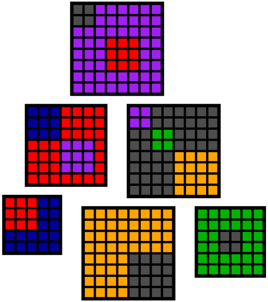
foreach *pack from the root to the leaves* **do**
  Apply completion on this pack in the domain of zones
  Pass the new constraints to its children

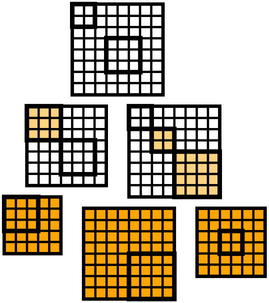


Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**
  Apply completion on this pack in the domain of zones
  Pass the new constraints to its father
foreach *pack from the root to the leaves* **do**
  Apply completion on this pack in the domain of zones
  Pass the new constraints to its children





Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**

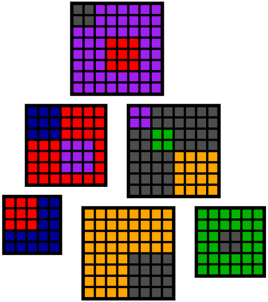
  Apply completion on this pack in the domain of zones

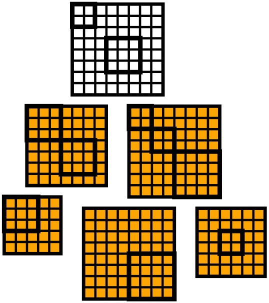
  Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**

  Apply completion on this pack in the domain of zones

  Pass the new constraints to its children



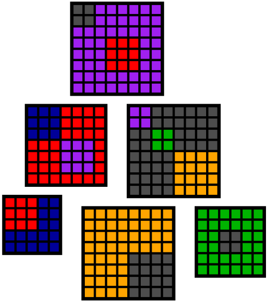


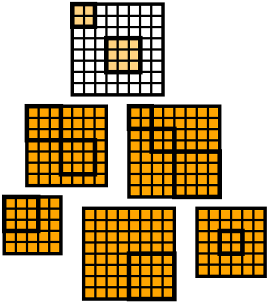
Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**
- Apply completion on this pack in the domain of zones
- Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**
- Apply completion on this pack in the domain of zones
- Pass the new constraints to its children





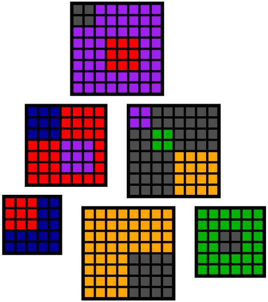
Completion algorithm

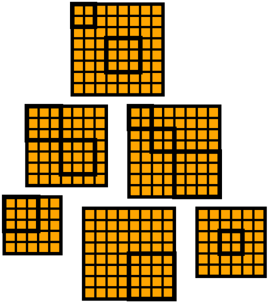
Completion algorithm in TreeKs $O(mp^3)$

---

foreach *pack from the leaves to the root* **do**
- Apply completion on this pack in the domain of zones
- Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**
- Apply completion on this pack in the domain of zones
- Pass the new constraints to its children





Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**

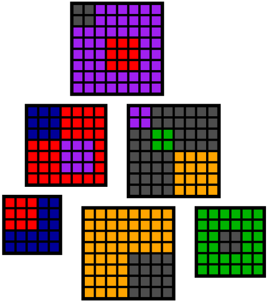
  Apply completion on this pack in the domain of zones

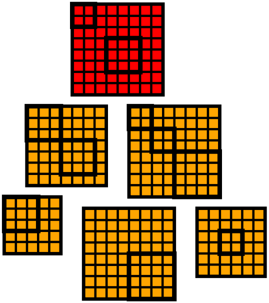
  Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**

  Apply completion on this pack in the domain of zones

  Pass the new constraints to its children



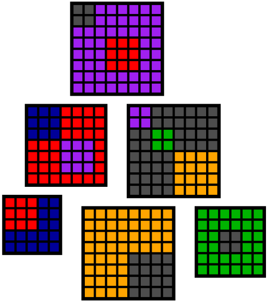


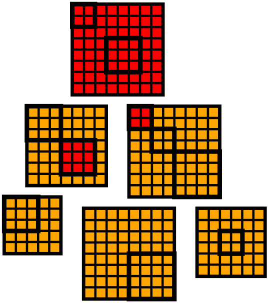
Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**
  Apply completion on this pack in the domain of zones
  Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**
  Apply completion on this pack in the domain of zones
  Pass the new constraints to its children





Completion algorithm

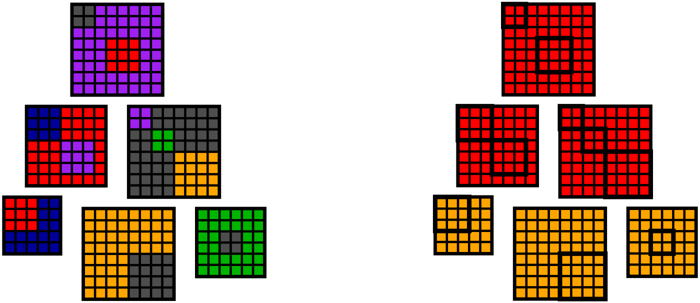
Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**

  Apply completion on this pack in the domain of zones
  Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**

  Apply completion on this pack in the domain of zones
  Pass the new constraints to its children

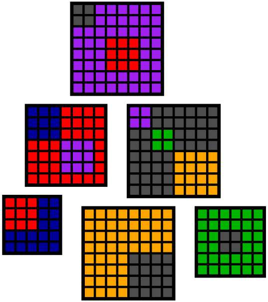


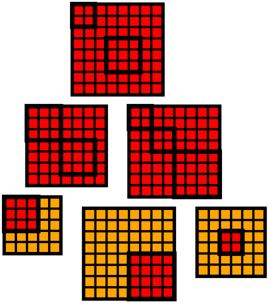
Completion algorithm

Completion algorithm in TreeKs $O(mp^3)$

foreach *pack from the leaves to the root* **do**
- Apply completion on this pack in the domain of zones
- Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**
- Apply completion on this pack in the domain of zones
- Pass the new constraints to its children





Completion algorithm

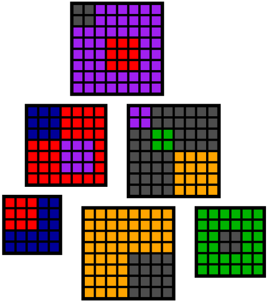
Completion algorithm in TreeKs $O(mp^3)$

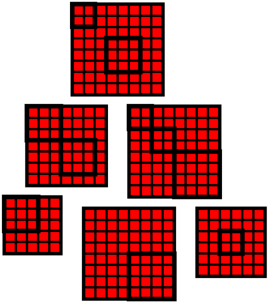
foreach *pack from the leaves to the root* **do**

- Apply completion on this pack in the domain of zones
- Pass the new constraints to its father

foreach *pack from the root to the leaves* **do**

- Apply completion on this pack in the domain of zones
- Pass the new constraints to its children

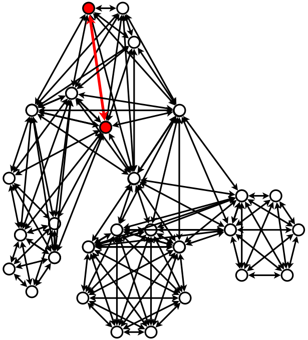


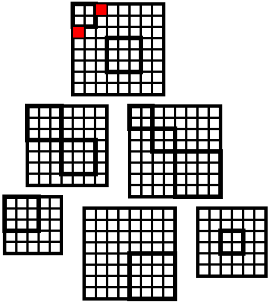


Constraint extraction

Goal: to bound $X_u - X_v$

Simple case: X_u and X_v are in the same pack

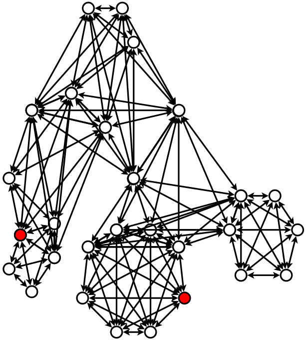


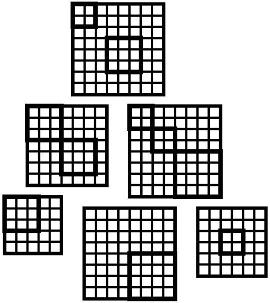


Constraint extraction

Goal: to bound $X_u - X_v$

Complex case: X_u and X_v are in different packs

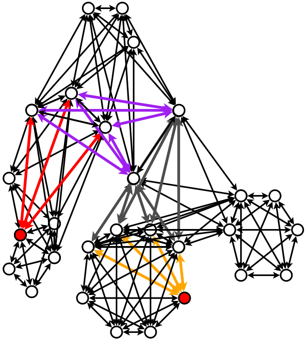


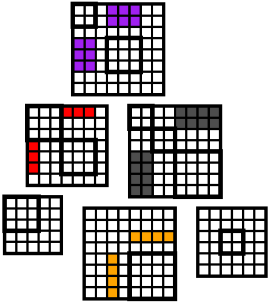


Constraint extraction

Goal: to bound $X_u - X_v$

Complex case: X_u and X_v are in different packs

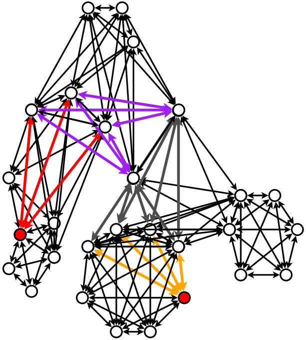




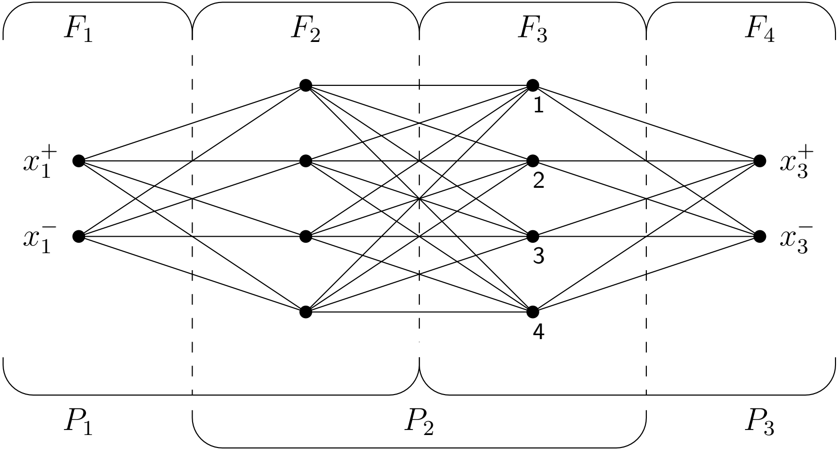
Only constraints in the path between X_v and X_u need to be considered

Constraint extraction (for zones/octagons)

The result is the shortest in a layered graph, which can be solved by dynamic programming, in time $O(df^2)$.



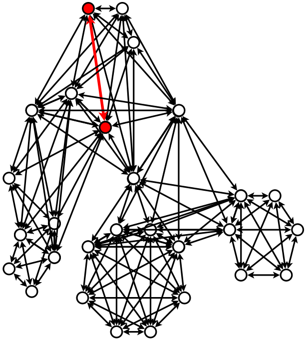
Constraint extraction (for zones/octagons)

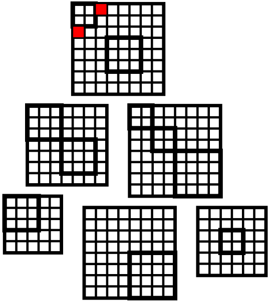


Adding constraints

Goal: to add the constraint $X_u - X_v \leq c$

Simple case: X_u and X_v are in the same pack

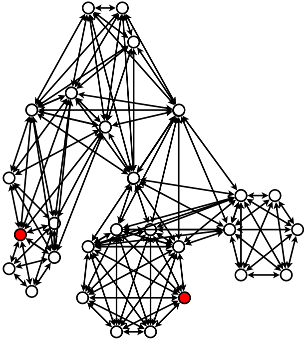


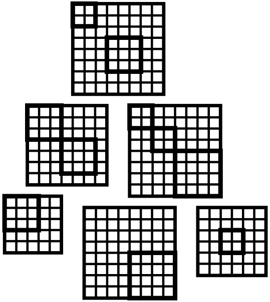


Adding constraints

Goal: to add the constraint $X_u - X_v \leq c$

Complex case: X_u and X_v are in different packs

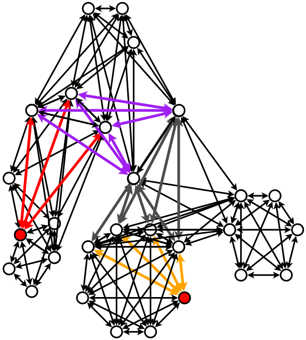


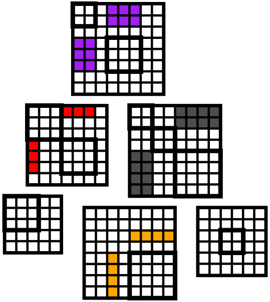


Adding constraints

Goal: to add the constraint $X_u - X_v \leq c$

Complex case: X_u and X_v are in different packs

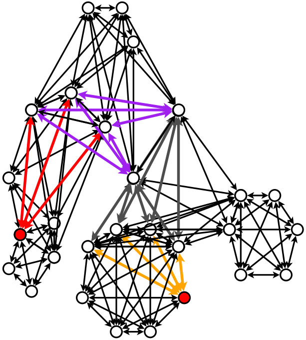




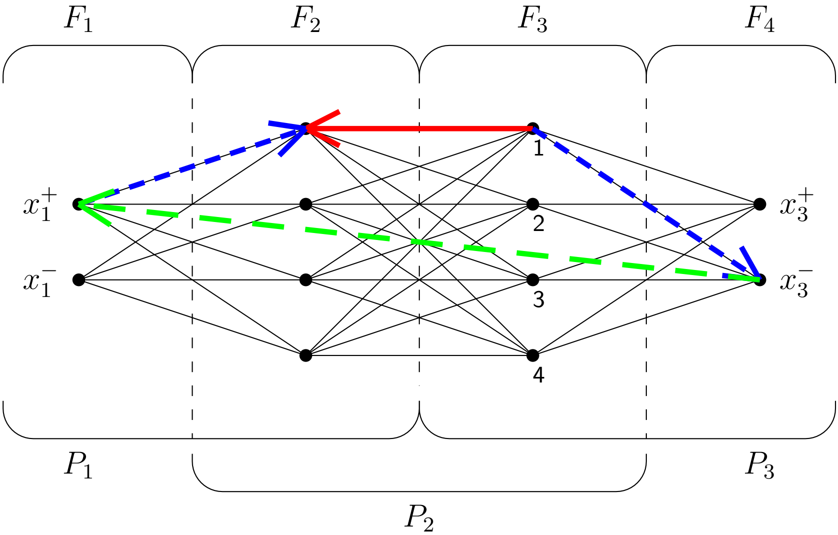
Only constraints in the path between X_v and X_u have to be updated

Adding constraints (for zones/octagons)

Like constraint extraction, shortest paths to successive frontiers help compute the best constraints between X_v and X_u in time $O(df^2)$.



Adding constraints (for zones/octagons)



Summary

We showed a method to build new numerical abstract domains:

- can be applied to many numerical abstract domains (zones, octagons, logahedra, TVPI, octahedra, polyhedra, ...)
- can be applied to other linear inequality domains to come
- with linear cost completion when pack size is bounded
- simple, precise, and efficient algorithms

Discussions:

- application to other convex domains and non-convex domains (e.g. AV domains)
- pack generation strategies