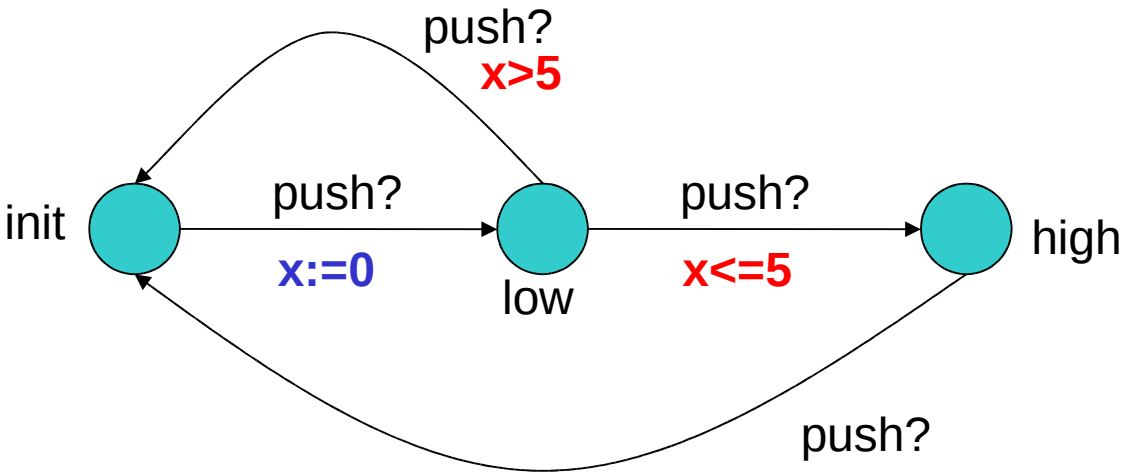
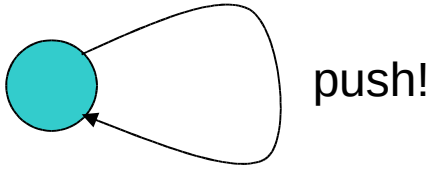
The DBM Library of UPPAAL and DBM Subtractions

Alexandre David
Gerd Behrmann
Kim Larsen
Johan Bengtsson
John Håkansson
Paul Pettersson
Wang Yi
...

Timed Automata in a Nutshell



Lamp



User

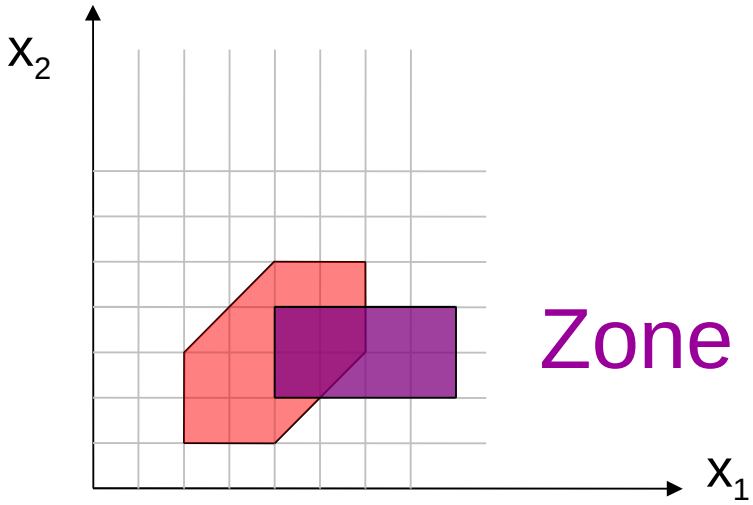
What is it about?

- Difference Bound Matrix: data structure for representing clock constraints, i.e., zones.
- DBMs represent convex zones.
  Note: canonical form.
- Subtraction may result in non-convex zones, i.e., DBMs must be *split*.
- Federations: unions of DBMs.

Example of a DBM

$x_0-x_0<=0$	$x_0-x_1<=-2$	$x_0-x_2<=-1$
$x_1-x_0<=6$	$x_1-x_1<=0$	$x_1-x_2<=3$
$x_2-x_0<=5$	$x_2-x_1<=1$	$x_2-x_2<=0$

$$\mathbf{x_i-x_j<=c_{ij}}$$



Zone (DBM) to subtract

Where is it needed?

- In UPPAAL: reachability and liveness analysis, deadlock detection.
- Subtraction needed for deadlock detection, implementation of priorities, and urgent transition with clock guards.
- Federations useful to handle split DBMs from subtractions and the extrapolation procedure.

Wait a second…

- Deadlock detection is available in UPPAAL: conversion to CDDs internally.
- Priority is work in progress.
- Urgent transition with clock guards is on the TODO list.
- UPPAAL uses DBMs, so we solve this problem with DBMs.

The DBM Library

- Stand-alone library usable in different languages: C, C++, ML (Emmanuel has promised a wrapper).
- Unit testing, robustness (it is used for *formal* verification right?).

The DBM Library

- New API based on past experience and new needs:
  - optimizations for the “close” operation
  - new extrapolations
  - federations
- Written in C, C interface to DBMs and federations.
- Federation C++ class.

Features of the Library

- Classical operations with a reduced “close”, DBMs always in canonical form.
- Different update functions (x:=y+c).
- Test suite with test API (generation of DBMs, points ...).
- Operations at the federation level.
- Subtractions (federation only).
- Different extrapolations.
- Minimal graph reduction.
- Priced DBMs coming...

General Features

- Well-known constraint encoding
  - (c << 1) | w : c constraint, w weakness of inequality ($x-y < c$ or $x-y \leq c$).
  - operations made directly on the encoded format – no decoding most of the time.
- Resizing of DBMs supported.
- Test suite, debugging API.

Minimal Graph – RTSS'97

- Mainly used for saving and restoring DBMs:
  - save and load
  - equality testing (with DBMs)
  - guaranteed to reduce memory footprint – features dynamic internal data representation (16/32 bits, bit matrix, list of indices, copy without diagonal)
  - Separated minimal graph analysis (for CDDs, priced DBM, subtraction)
  - Relation operation (inclusion checking).

Federations

- A federation is an arbitrary union of zones
  - implemented as a list of DBMs
  - alternative representation as CDDs
- Semantically manipulated as a whole.
- Internal memory management (recycle DBMs).

Features of Federations

- Special relation (inclusion checking) DBM~fed:
  - *subset* if DBM *included in one* DBM of fed
  - *superset* if DBM *includes all* DBMs of fed
  - *equal* if subset and superset
  - *different* otherwise
  - *safe but can miss some inclusions*
- Exact relation also available but very expensive.

C++ Federation

- Wraps operations on the C data structure.
- Supports active clocks and resizing.
- Encapsulated memory management.

Subtractions

- Supported:
  - FED2 = DBM1 – FED1
  - FED1 = DBM1 – DBM2
  - FED2 = FED1 – DBM1
  - FED3 = FED1 – FED2
  - FED1 = FED1 – DBM1
  - FED1 = FED1 – FED2
- Minimal split.
- Disjoint DBMs.

Subtractions

- Problems with subtractions:
  - Splitting DBMs means to partition the symbolic states.
  - Splitting “propagates”: states are used to generate successors, that will generate successors…
  - Issues: inclusion checking, state-space explosion.

What to do?

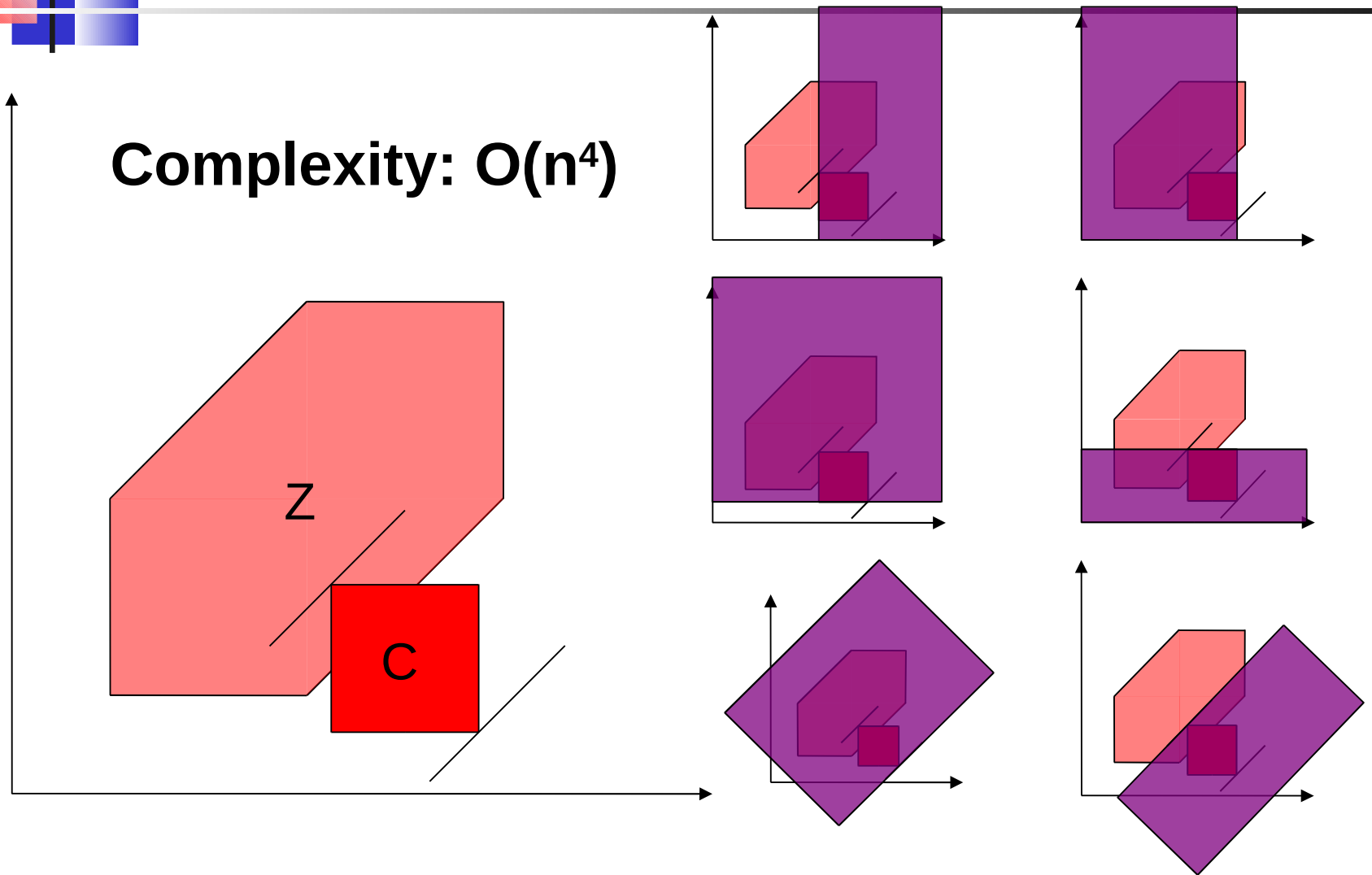
- Best data structure for most operations in timed model-checkers.
- Compute the “best” subtraction possible
  - No redundancy
  - Fewest number of splits

How to do it?

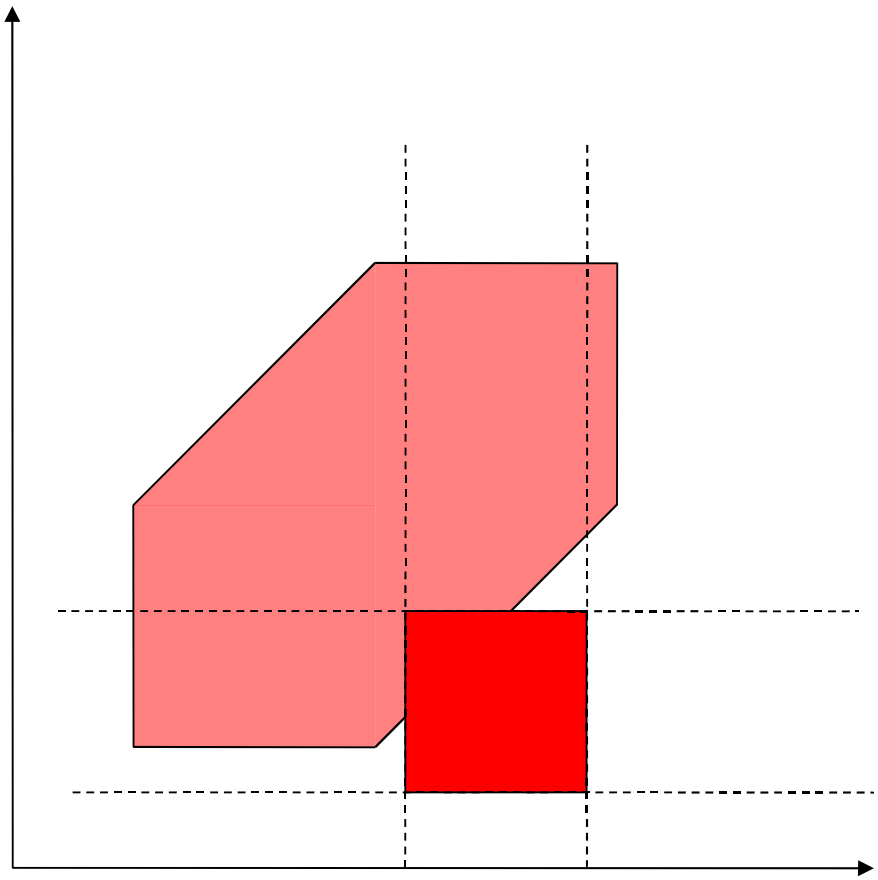
- Compute the minimal graph (RTSS’97).
- Remove non necessary edges (w.r.t. the subtraction).
- Compute disjoint subtraction.

Subtraction (Z-C): Basics

Complexity: $O(n^4)$

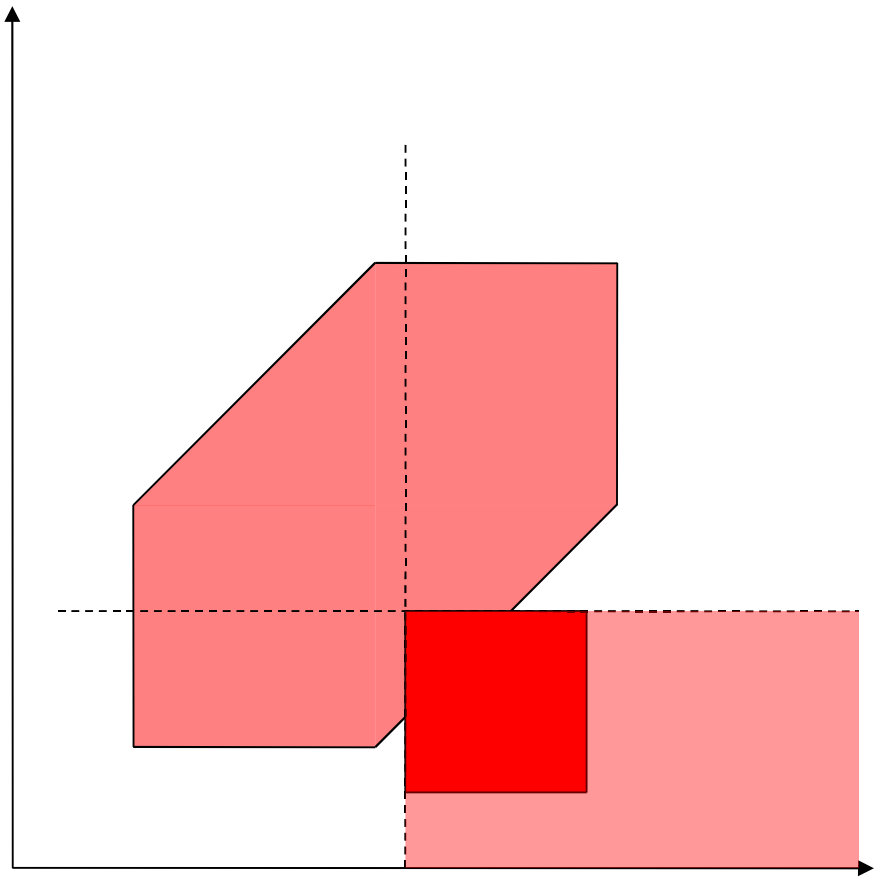


Compute Minimal Graph



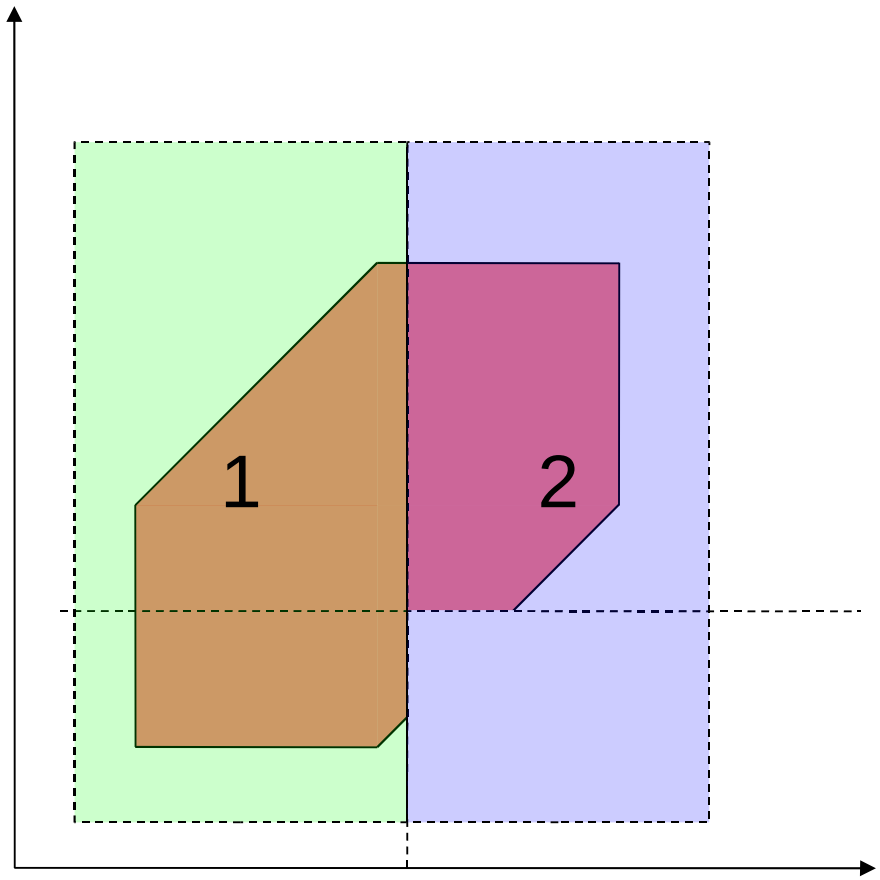
- Here only in 2D, seems obvious.
- Complexity: cubic.

Remove Edges



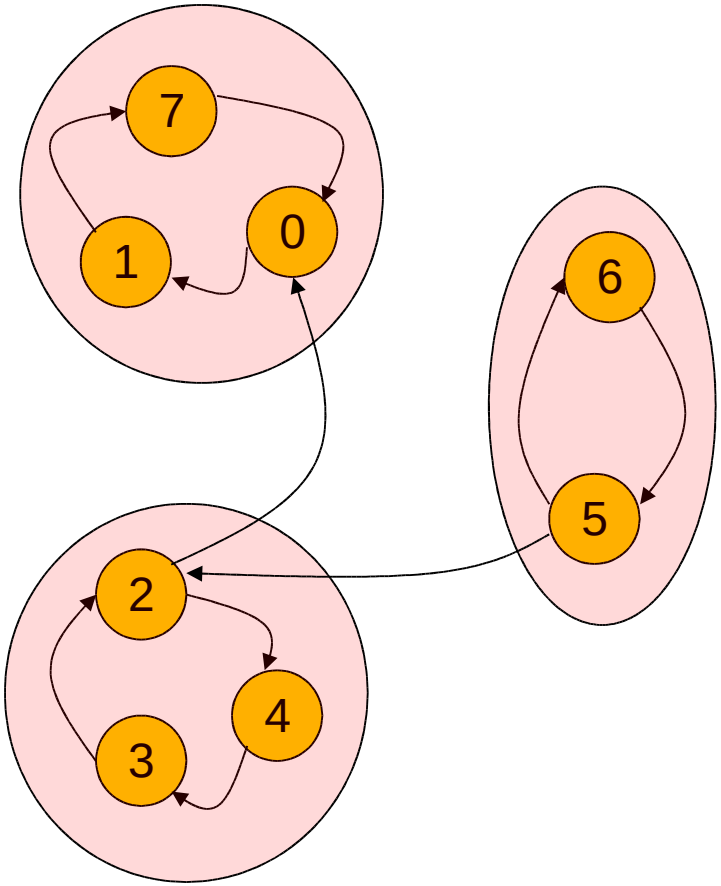
- Decision algorithm linear in the number of clocks for every facet.
- Remove edges not necessary for the subtraction, i.e., non intersecting facets.
- Complexity: cubic.

Disjoint Result



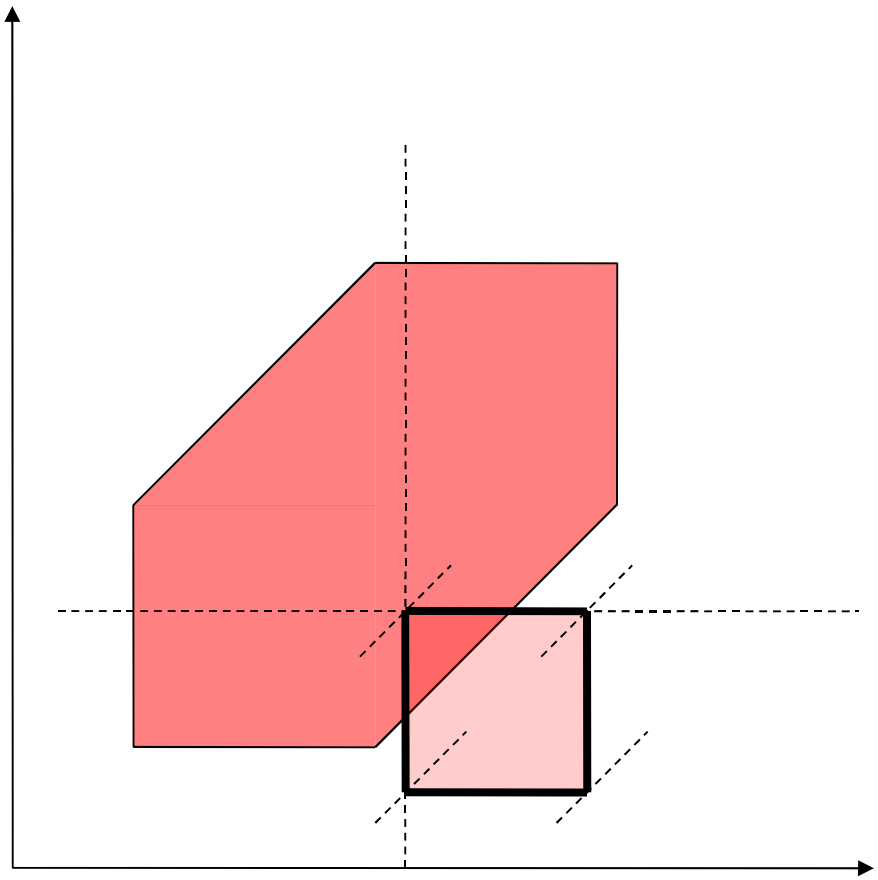
- Remove redundancy between split DBMs.
- Result = union of disjoint zones.
- Complexity: $O(n^4)$

How to Compute the Minimal Graph?



- Identify equivalence classes.
- Remove redundant edges between equivalence classes.
- See paper RTSS'97.
- Correctness: the minimal graph describe the same zone.

How to Remove Edges?



- Project the zone on every facet.
- Consider the “closed” form but do not compute it.
- Test for intersection with on-the-fly tightened constraints.

Technically…

Go through edges c_{ij} (minimal graph) and:

2. Compute partial projections with
   $c_{ji}'=-c_{ij}$
   $c_{ki}'=c_{kj}+c_{ji}'$
   $c_{jk}'=c_{ji}'+c_{ik}$
3. Compare against z_{ij}, z_{ik}, z_{kj}:
   if $-c_{ji}'>=z_{ij}$ or $-c_{ki}'>=z_{ik}$ or $-c_{jk}'>=z_{kj}$ then remove c_{ij}.

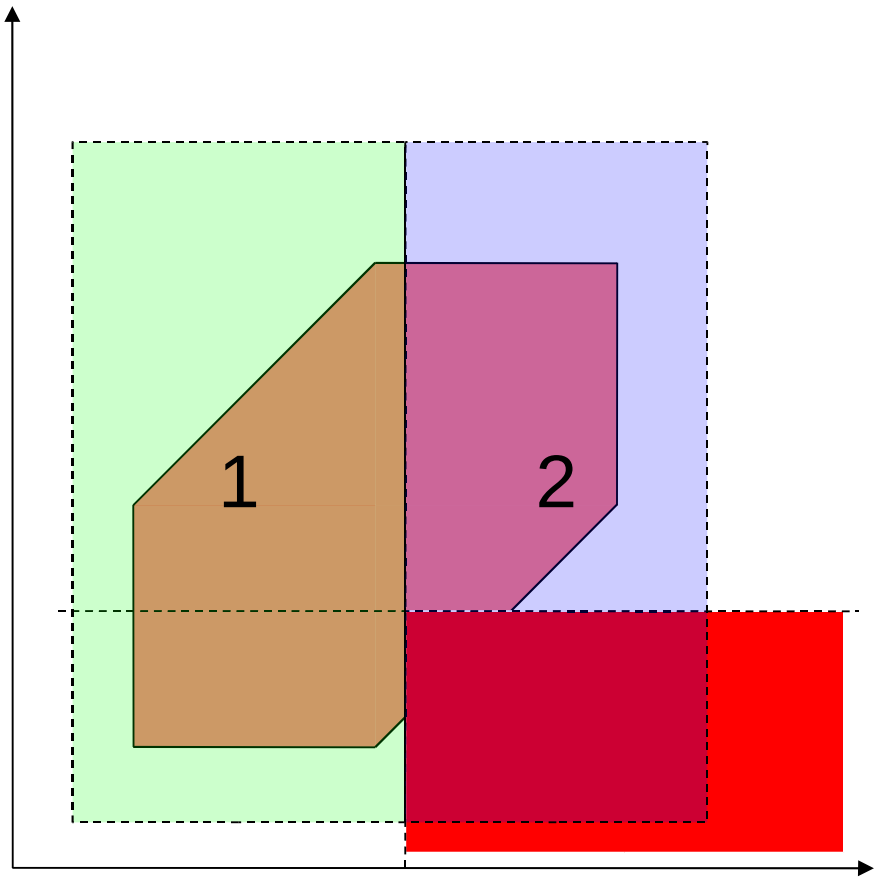
Soundness

- The computed subtraction is the same:
  - Removed edges correspond to non intersecting facets and thus have no effect on the result because the zone is convex.
  - Special case: all edges removed if Z included in C and the result is empty. Also correct because there is no subtraction at all.

Completeness

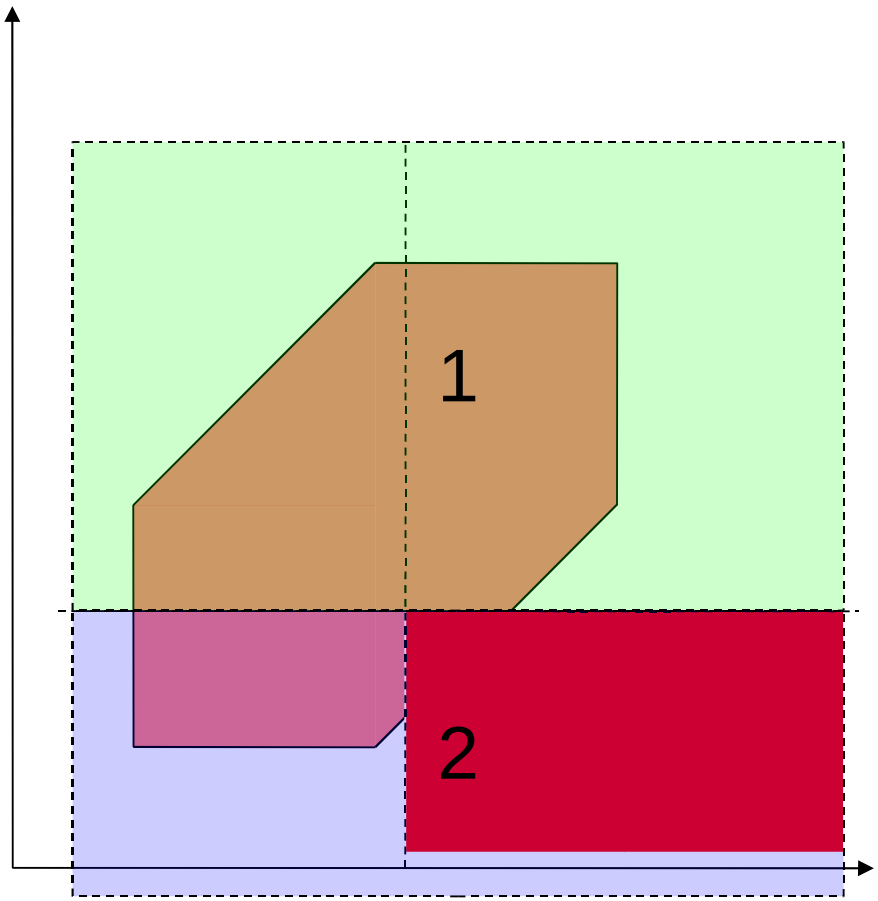
- The subtraction has the fewest number of splits possible:
  - Remaining edges are the necessary ones needed to describe the zone to subtract.
  - Remaining edges intersect the zone to subtract from.

How to Have Disjoint DBMs?



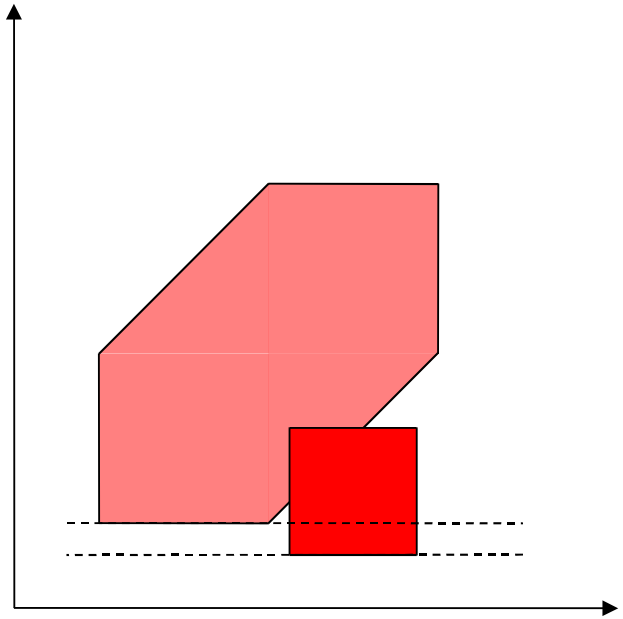
- Keep track of the removed DBMs by computing the remaining of the subtraction.
- Correctness: we only remove redundancy between the resulting DBMs.

Remark: Order Matters

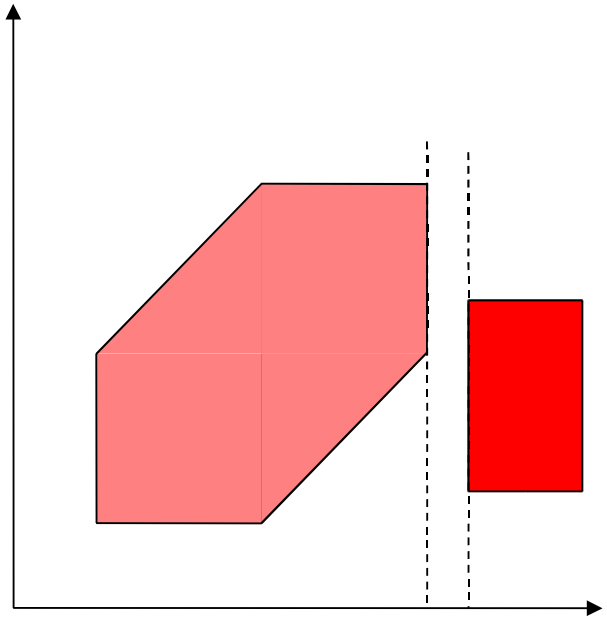


- A different order results in a different partition.
- The union is still the same and has no redundancy.

Optimization: 2 Easy Cases



Edge has obviously no effect



Identity

Is It Worth?

- There is a significant computation overhead (twice slower).
- The quality of the result is more important since it will be reused in the model-checker.
- How to evaluate: implemented very expensive operations on federations that use subtraction in a **recursive** manner.

Experiments

- Benchmark one of the tests in our DBM library:
  - Run one function 1000 times
  - Generate random arguments
    - But not any random: find bad cases
- Run the test with the same random seed (same random set) with different options.
- Note: operation is exponential in the number of DBMs, n^4 in the number of clocks.

Expensive Reduce: Detect Redundant DBMs

Dim/DBMs	Basic	Disjoint	Reduced	Minimal
7/4	1.5s/29.7M	0.7s/0.7M	1s/11.7M	0.5s/0.5M
7/5	2.9s/63.9M	0.6s/0.5M	1.4s/11.1M	0.6s/0.4M
7/6		0.5s/0.8M	7.9s/173.5M	0.8s/0.4M
7/14		4s/1.9M		3.5s/1.2M
7/30		24.8s/6.3M		18.7s/3.1M
8/20		11s/11.1M		9.2s/5.2M
9/20		15s/27.6M		13.2s/13.9M

Conclusions

- DBM Library
  - we have to release it (promised)
  - support for federations, active clocks ...
- Subtractions
  - **Minimal** subtraction:
    - Fewest number of splits
    - Smallest result (disjoint DBMs)
  - Important because the result is **propagated** in the model-checker.
  - Computation overhead is worth.