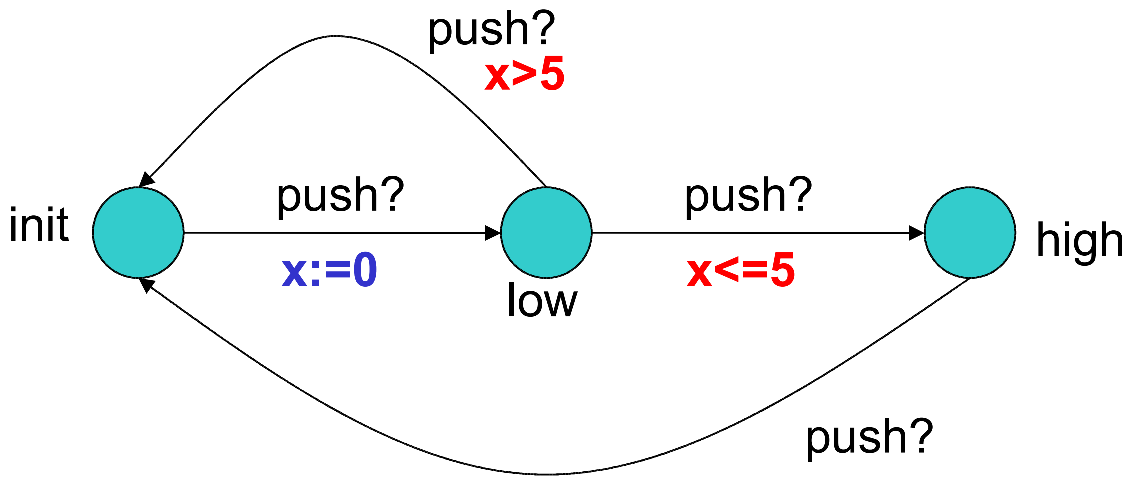
Merging DBMs Efficiently

Alexandre David
Aalborg University

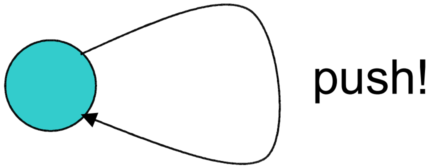
Plan

- Framework
  - Timed Automata
  - DBMs & Federations
  - Why Merging DBMs?
- Merging DBMs
  - The Problem
  - The Different Algorithms
  - Experiments

Warming Up:
Timed Automata in a Nutshell



Lamp



User

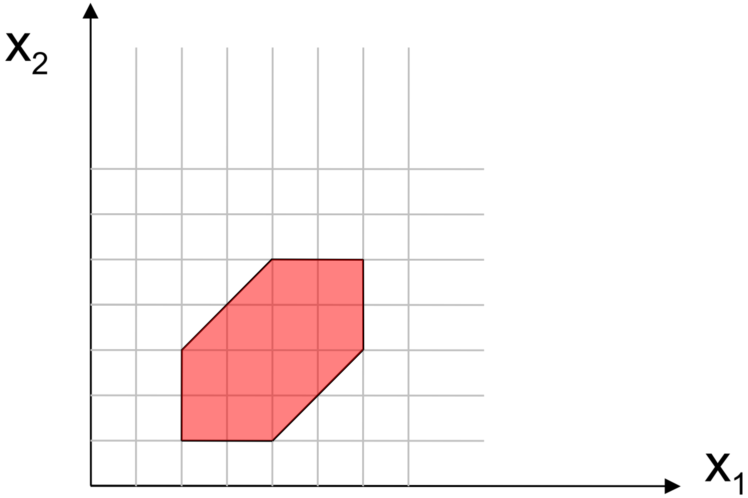
What is it all about?

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

Example of a DBM

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

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

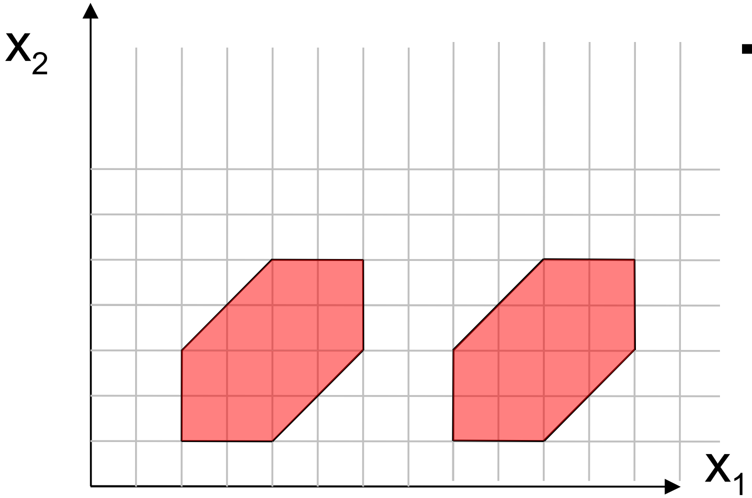


Example of a Federation

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

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

+matrix of the second DBM



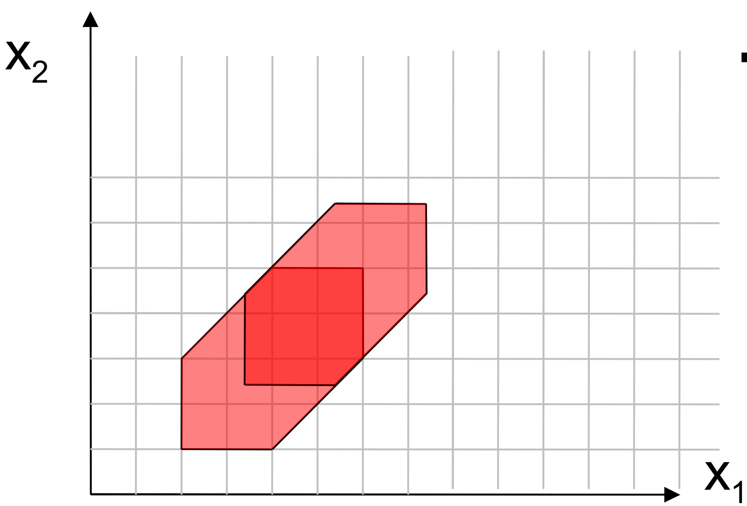
Disjoint

Cannot be simplified

Example of a Federation

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

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



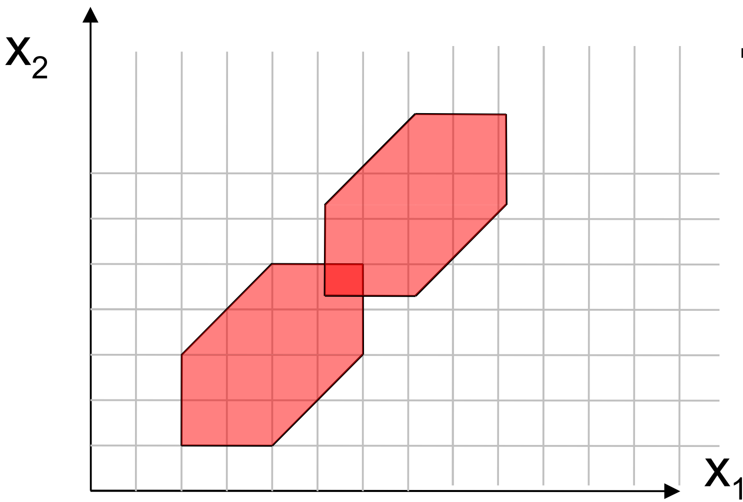
+matrix of the second DBM

Can be simplified

Example of a Federation

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

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

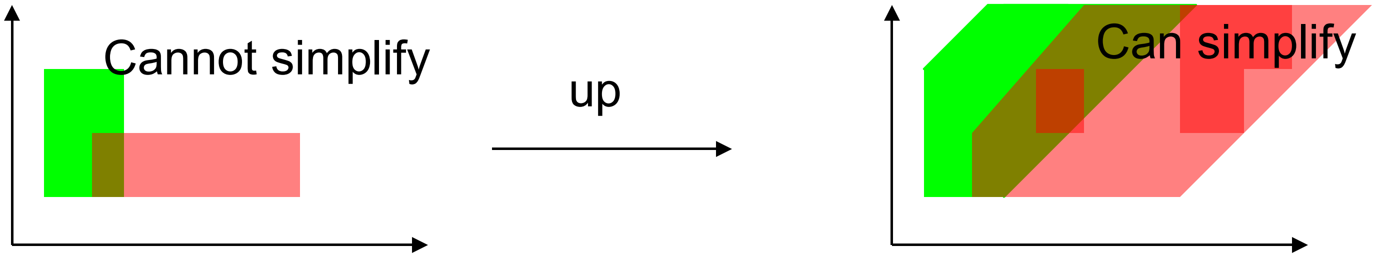


+matrix of the second DBM

Cannot be simplified

Why Merging DBMs?

- State explosion: “Split” states give “split” successors etc...
- Even if it is costly (see algorithms), it does work. Justified by operations that make it possible.



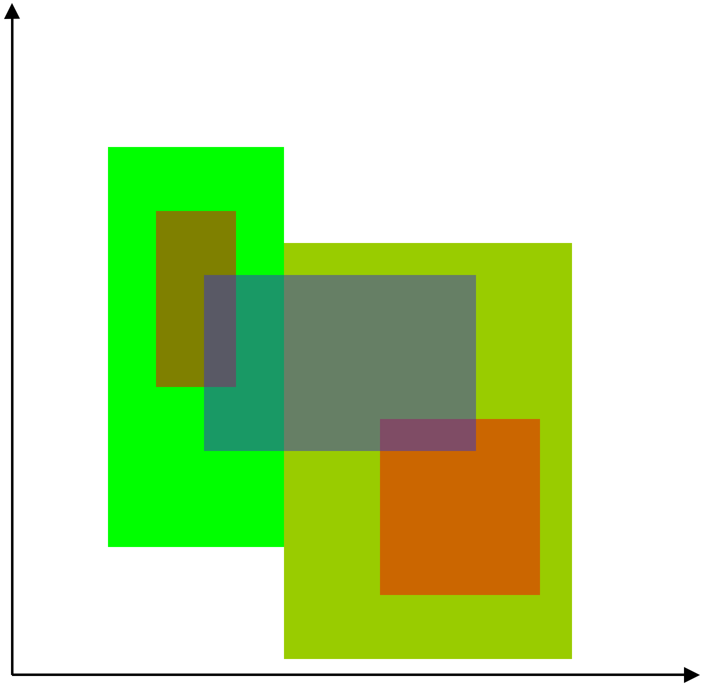
- Note: We have not used alternative representations yet on our experiments, e.g., CDDs. We do our best with what we have, i.e., federations.

The Problem

- Given a Federation, is it possible to simplify it?
  - Remove included DBMs
  - Merge adjacent DBMs
- Sure it is possible but **how** do you choose your DBMs? **How many** DBMs can you merge?

Removing DBMs

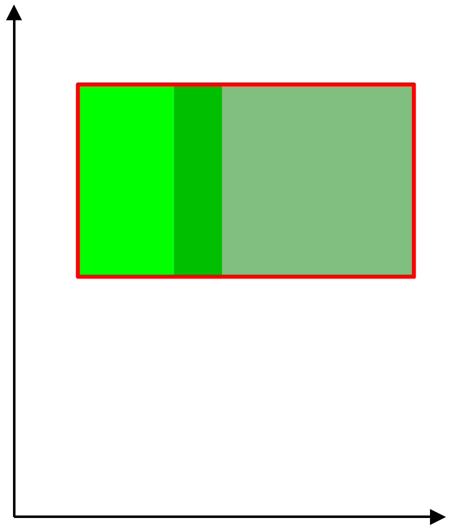
- DBM inclusion (cheap) or exact inclusion (more expensive).

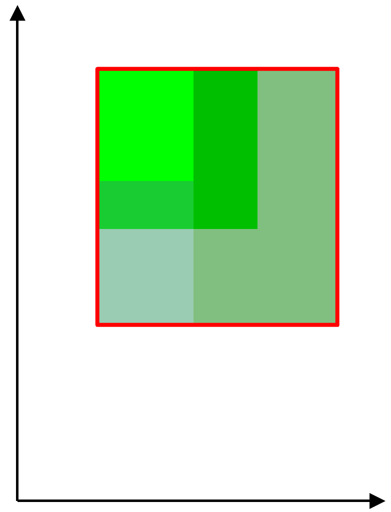


Note: In practice we have dimension *n*.

Merging DBMs - Principle

- Check if convex_hull(A,B) == A|B
- Problem: 2^n ways of choosing DBMs (2, 3, … , n). We don't know how many DBMs we can merge together.





More complex configurations in practice.

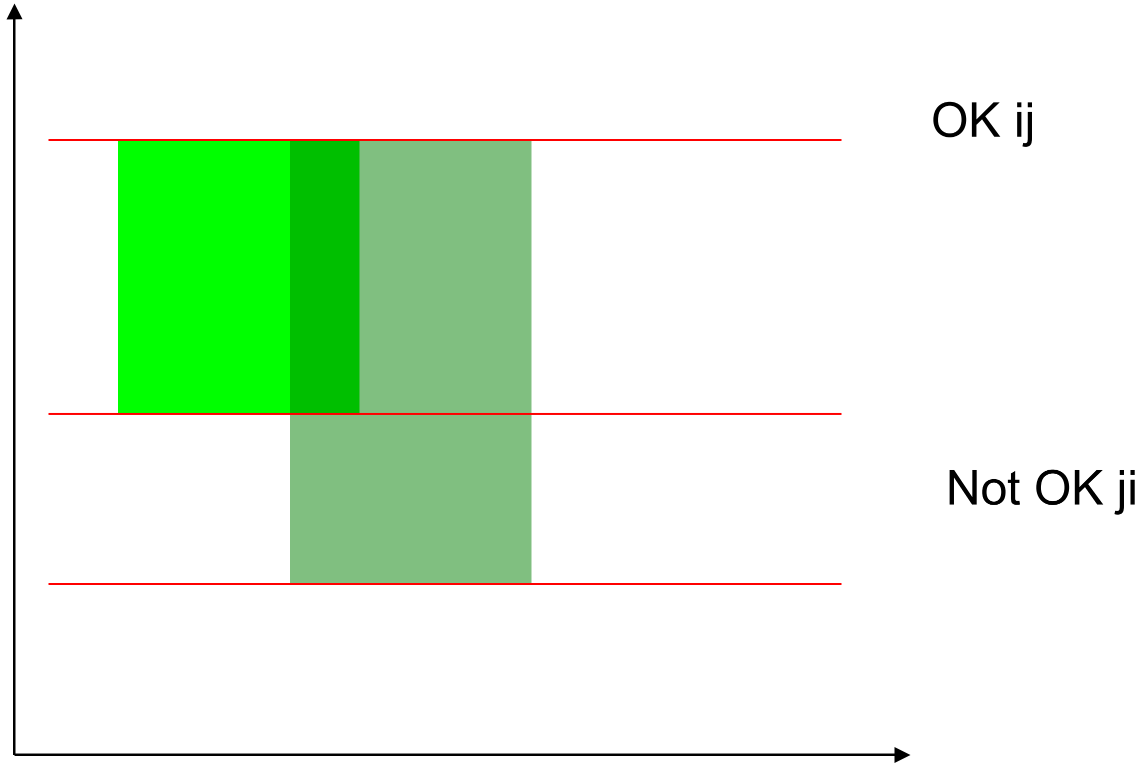
Let's Do It!

- Algorithms:
  - Reduce: Inclusion checking.
  - ExpensiveReduce: Exact inclusion checking.
  - 2-merge: Merge 2 by 2.
  - N-merge: Dynamically find N DBMs to merge.
  - Partitioned N-merge: Find partitions and apply N-merge + expensiveReduce.
  - ConvexReduce: Recompute the federation.

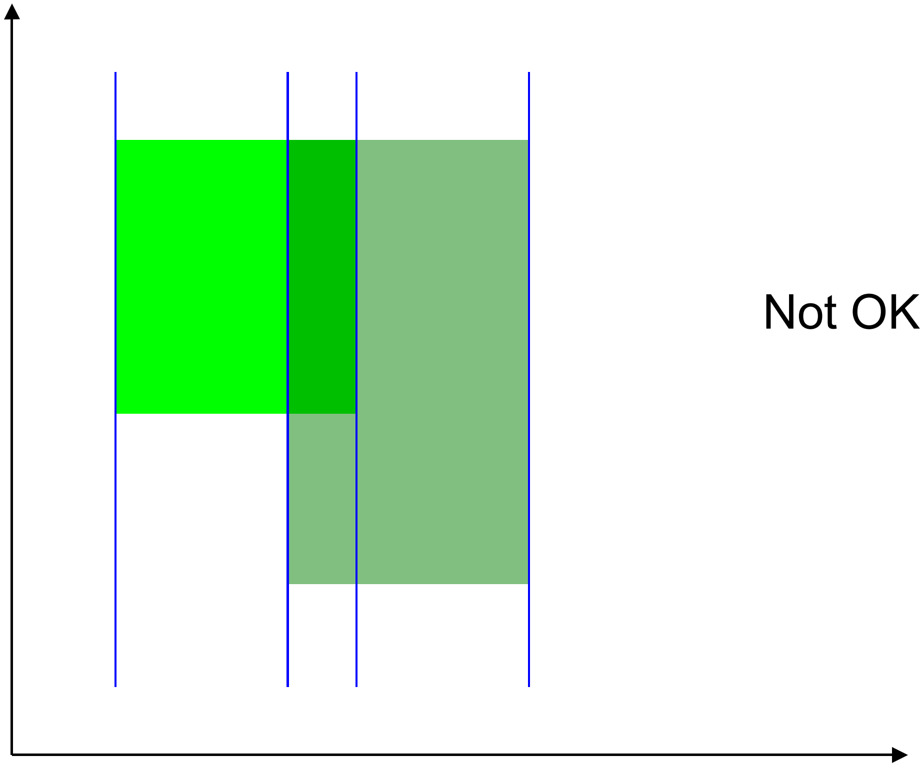
2-merge

- N^2 pairs to try.
- Use cheap test based on 2 necessary conditions (not sufficient):
  - 2 opposite constraints of 2 DBMs must be equal, e.g., $a_{ij} = b_{ij}$ and $a_{ji} = b_{ji}$.
  - Intersection of adherence is not empty.
- Then we try the merge with the convex hull – needs subtractions.

2-merge

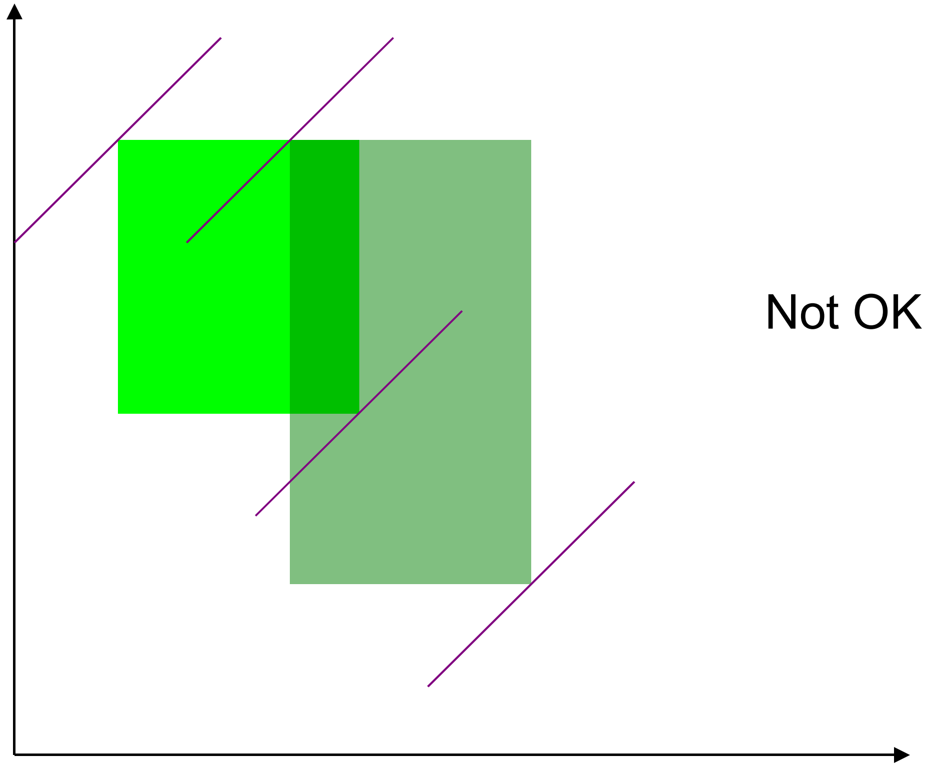


2-merge



Not OK ij + ji

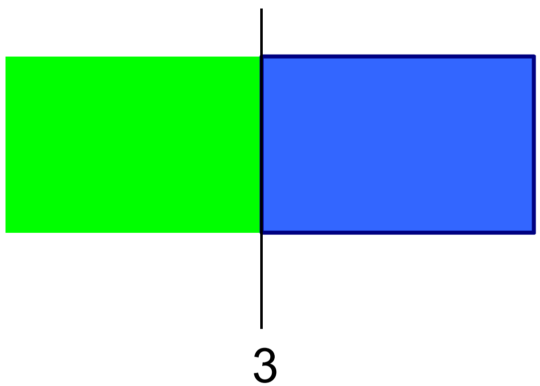
2-merge



Not OK ij + ji

2-merge

- Adherence:
  - $x < 3$ and $x \geq 3 \Rightarrow x \leq 3$ and $x \geq 3$



- We also check for DBM inclusion.
- Finally if the conditions are met, we check if convex_hull(A,B)-(A|B) is empty.

N-merge

- Relaxed 2-merge: only one compatible constraint.
- Algorithm (inclusion check ommitted):
  - For all i < n, for all j < *n* & j > i: n^2
    - union := DBM[i]
      if 2-merge DBM[j] := DBM[i]|DBM[j] &
        retry on all j
      else if “1/2-merge” union |= DBM[j]
    - C := convex_hull(union)
    - For all j < n: if DBM[j] included in C, union |= DBM[j]
    - If R := C-union is empty replace union by C
      - Else if size(C-(C-union)) < size(union) replace union by C-(C-union)
      - Else ExpensiveReduce on union.

Partition N-merge

- Algorithm:
  - Find a partition of our federation
  - Fixpoint on the sub-sets of
    - N-merge
    - Followed by ExpensiveReduce if there was a reduction

ConvexReduce

- Idea: Recompute the federation and reduce "fragmentation".
- Algorithm:
  - C = convex_hull(Fed)
  - F = C-(C-fed)
  - Fed = F if size(F) < size(Fed)

Experiments: Does it work?

- We need a real case example where federations are heavily used **and** there is much split:
  - Timed game reachability algorithm, backward & forward [CDFLL05].
  - Current work: Applying this algorithm to jobshop scheduling.
  - Experiments on one instance with and without uncertainties – difficult instance.
  - Question: Is there a winning strategy?

Based on 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.
- *Dual Xeon 2.8GHz, 4GB RAM, Linux 2.4.*

Without Uncertainties - Easy

	+N-mrg	Time	Memory
No Reduce	1.9s	5.3s	44.6M
Reduce	1.9s	2.2s	20.8M
ExpensiveReduce	2.0s	2.5s	21.1M
2-merge	2.0s	2.0s	19.9M
N-merge	2.4s	2.4s	19.9M
Partition N-merge	2.4s	2.4s	19.9M
ConvexReduce	2.1s	2.1s	19.9M

Without Uncertainties - Easy

- Small federations.
- Small difference between methods.
- Reduce still important.
- 2-merge best.
- Only one bottleneck in the experiment that really matters.

With Uncertainties - Difficult

	+N-merg	Time	Memory
No Reduce	203s	4051s	918M
Reduce	201s	7147s	732M
ExpensiveReduce	257s	3831s	784M
2-merge	190s	1897s	572M
N-merge	372s	372s	526M
Partition N-merge	339s	345s	525M
ConvexReduce	201s	415s	532M

With Uncertainties - Difficult

- 2-merge best for simple cases, as before.
- Partition & N-merge best for complex cases. If we generate the strategy, N-merge is best.
- One bottleneck that *really* matters.

Conclusion

- It works and it is **very** important to reduce federations.
- Best method (cheap/expensive) depends on the application.
  - Expensive method on critical bottlenecks.
- Efficient in practice.

References

- [CDFLL05] *Efficient on-the-fly algorithms for the analysis of timed games.* CONCUR'05, LNCS 3653, pp 66-80.
- UPPAAL: *www.uppaal.com.*