Let's Be Lazy, We Have Time

or, Lazy Reachability Analysis for Timed Automata

Loïg Jezequel[1,3] and Didier Lime[2,3]

[1]Université de Nantes

[2]École Centrale de Nantes

[3]LS2N, UMR CNRS 6004

FORMATS, September 6, 2017

Overview of the problem

Distributed systems

- Components
- Communications
- Clocks (can be shared, with invariants) $\Rightarrow$ Time-non blocking!

Reachability

- Component by component
- For a subset of components
- First step towards model checking

Solution

- Modular/compositional (component by component analysis)
- Lazy (use only components and clocks needed for the analysis)

The formalism

Components

Timed automata

- guards: $x \sim k$ with $k \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$
- invariants: $\sim \in \{<, \leq\}$

(extendable as long as reachability analysis is possible)

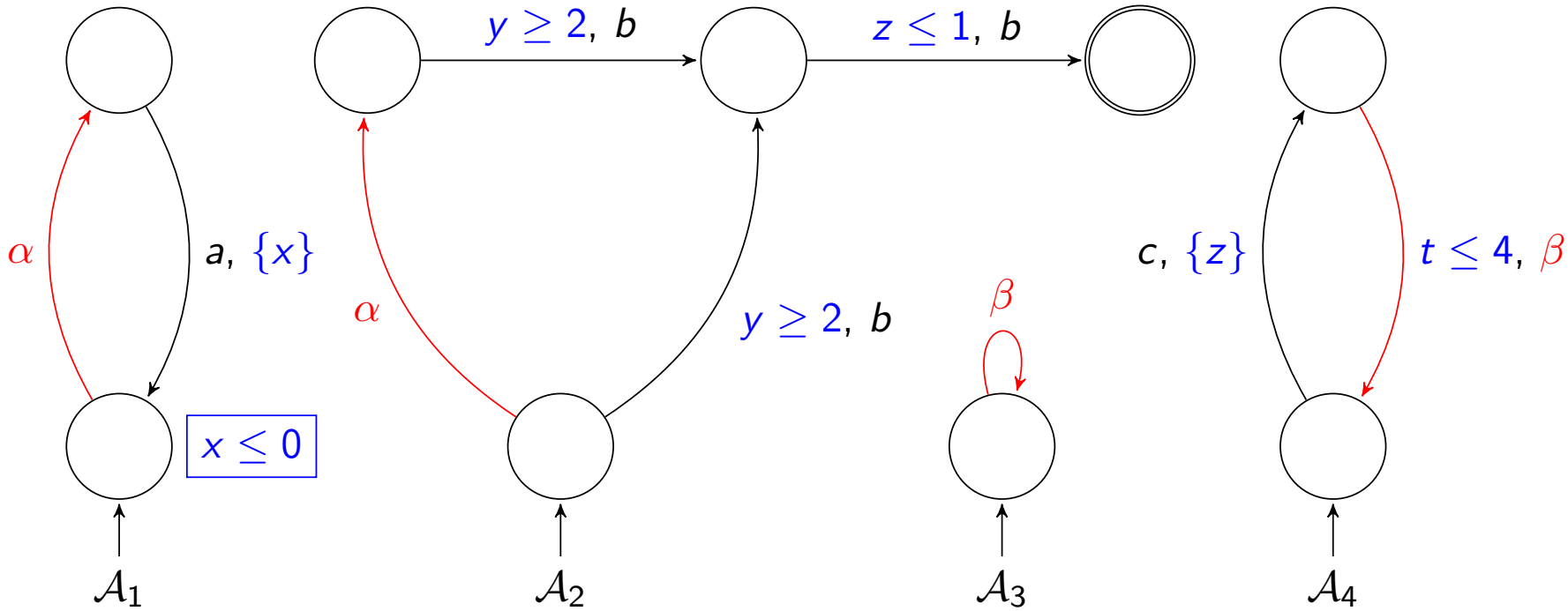
Communication

Synchronous product, union for guards, invariants, and resets

Reachability

Marked states

Overview of our algorithm



I. General principle of the algorithm

Initialization

Partition

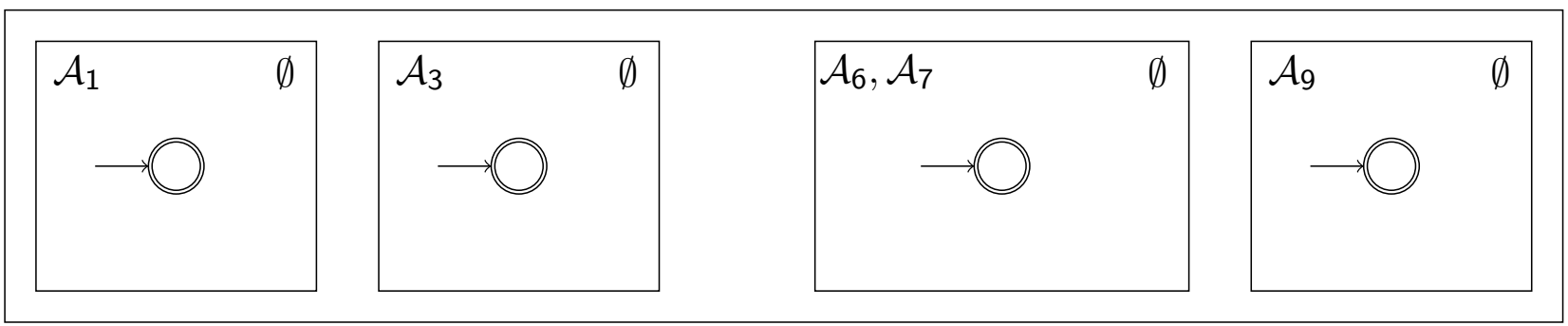
Choose a partition of the TAs involved in the reachability objective

⚠ Invariants on marked states

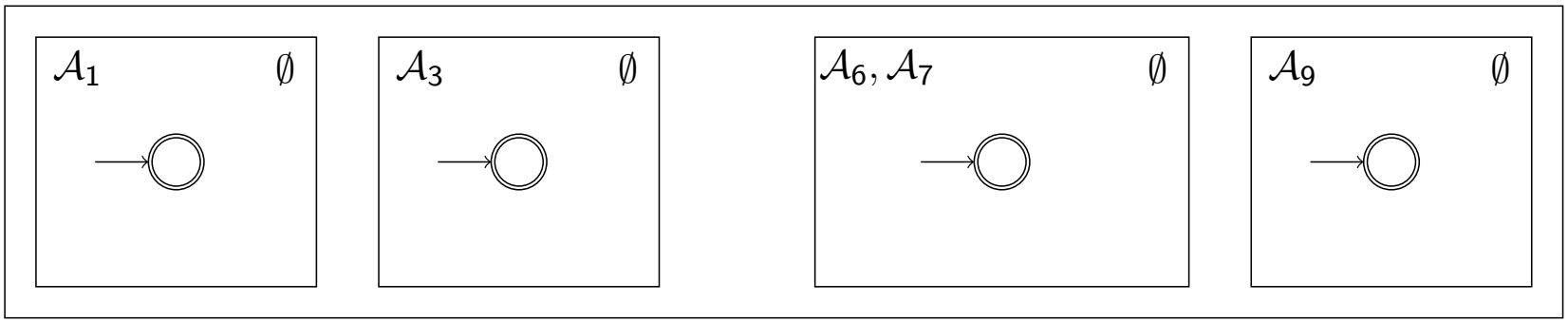
Initial paths

For each element of the partition

- initialize a set of paths with only the empty path in it
- initialize an empty set of clocks



Initialization continued



Finished?

Does this set of set of paths contains a solution?

Yes: we are done

No: add paths, clocks, or merge sets

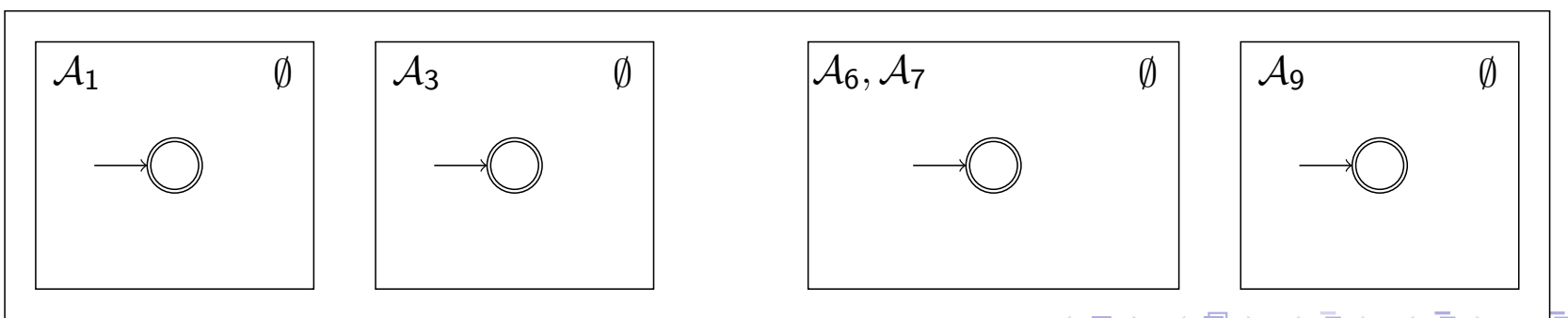
Completeness

Idea

A set P of paths is complete if $p \in P$ reaches the objective and:

- p uses only private actions
- all clocks needed for p are in the set of clocks
- no action in p is in conflict with an invariant not in P
- no clock in the set of clocks is reset from outside of P

Solution to incompleteness: concretisation



Completeness

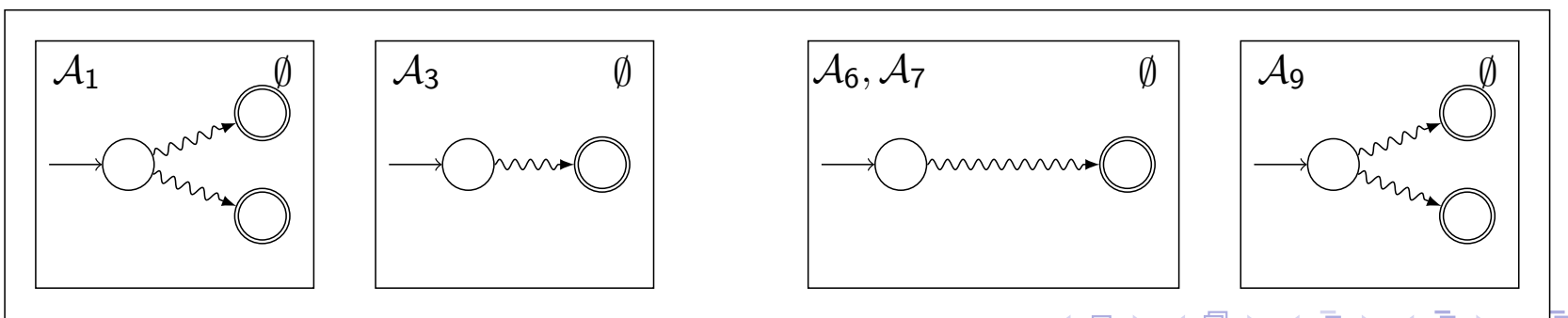
Idea

A set P of paths is complete if $p \in P$ reaches the objective and:

- p uses only private actions
- all clocks needed for p are in the set of clocks
- no action in p is in conflict with an invariant not in P
- no clock in the set of clocks is reset from outside of P

Solution to incompleteness: concretisation

- Add new paths to the incomplete set, or



Completeness

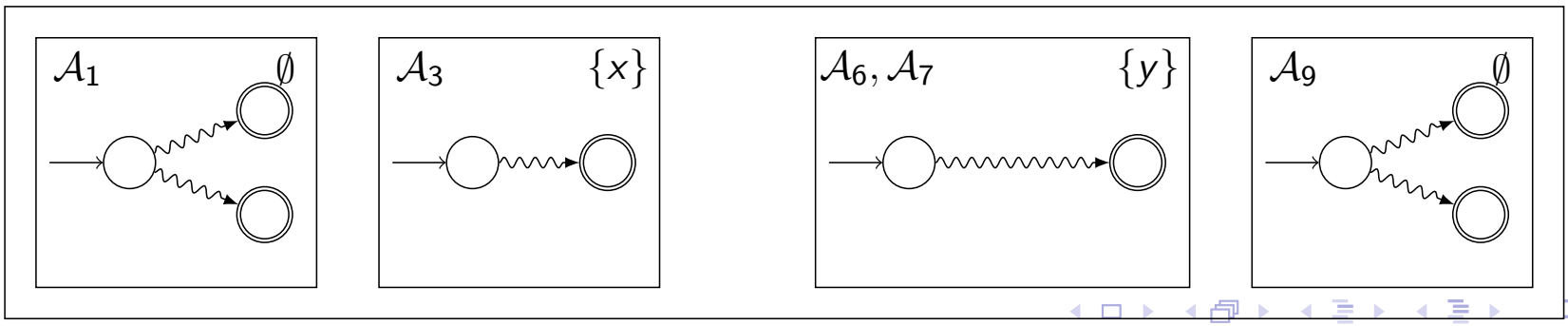
Idea

A set P of paths is complete if $p \in P$ reaches the objective and:

- p uses only private actions
- all clocks needed for p are in the set of clocks
- no action in p is in conflict with an invariant not in P
- no clock in the set of clocks is reset from outside of P

Solution to incompleteness: concretisation

- Add new paths to the incomplete set, or
- add clocks to the set of clocks, or



Completeness

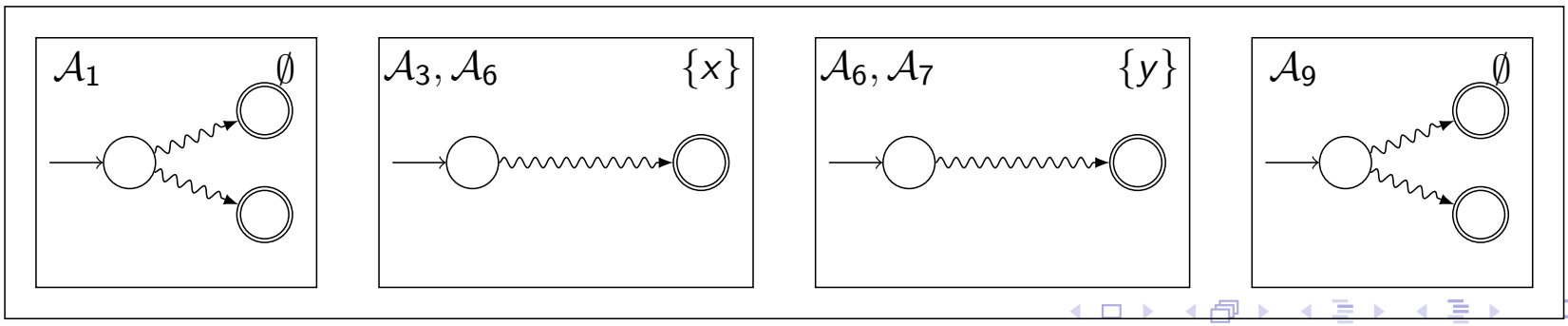
Idea

A set P of paths is complete if $p \in P$ reaches the objective and:

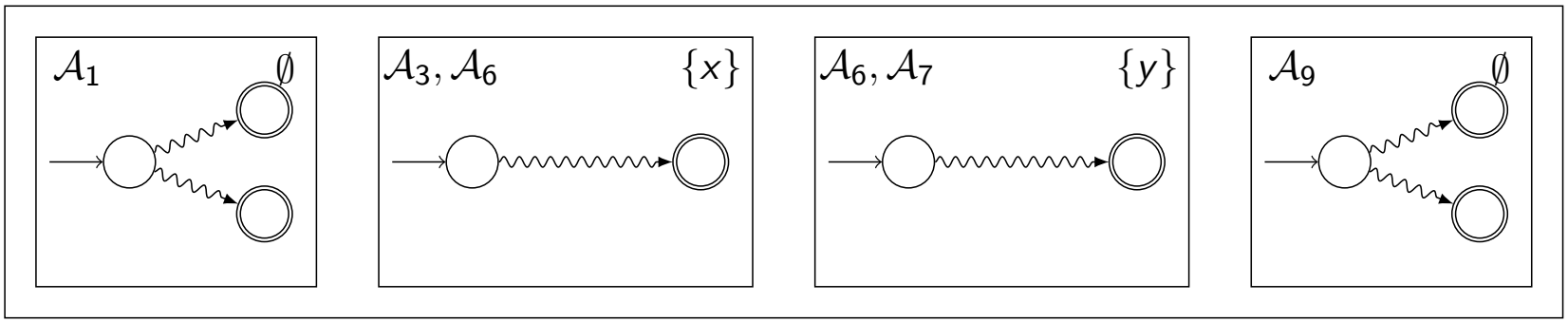
- p uses only private actions
- all clocks needed for p are in the set of clocks
- no action in p is in conflict with an invariant not in P
- no clock in the set of clocks is reset from outside of P

Solution to incompleteness: concretisation

- Add new paths to the incomplete set, or
- add clocks to the set of clocks, or
- add new automata to the set of automata



Completeness continued



Finished?

Is this set of set of paths giving a solution?

Yes: we are done

No: add paths, clocks, or merge sets

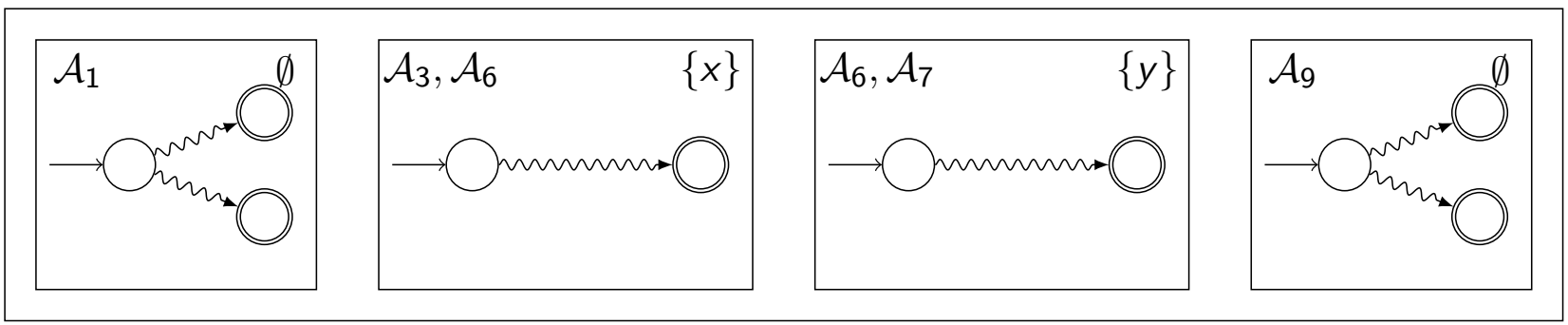
Consistency

Idea

A set of sets of paths is not consistent if two different sets of paths share at least one automaton

Solution to inconsistency: merging

- Select two sets of paths breaking consistency
- merge them (i.e. change the initial partition of the LTSs)



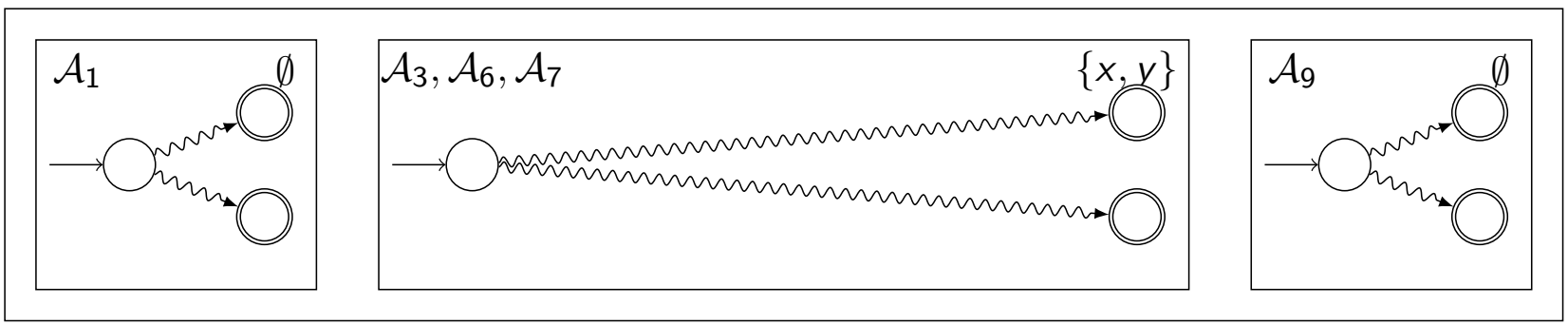
Consistency

Idea

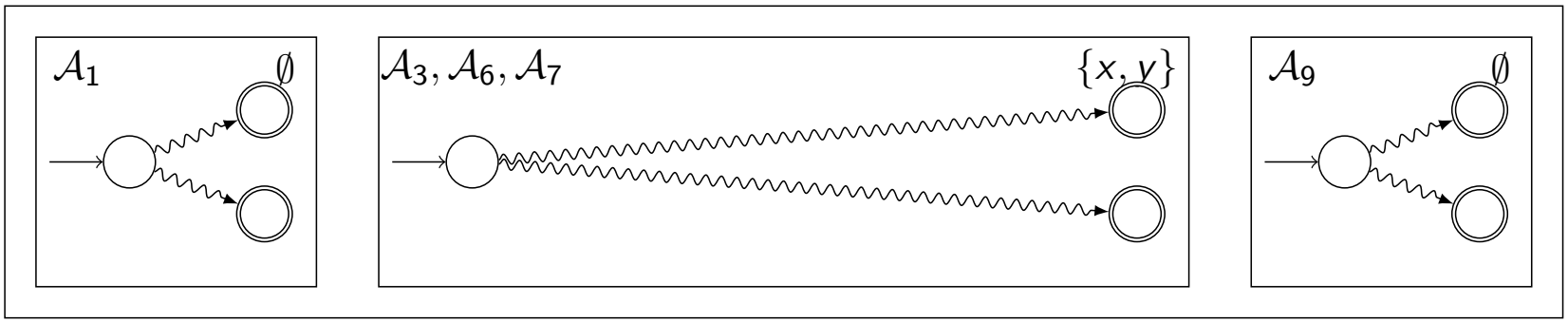
A set of sets of paths is not consistent if two different sets of paths share at least one automaton

Solution to inconsistency: merging

- Select two sets of paths breaking consistency
- merge them (i.e. change the initial partition of the LTSs)



Consistency continued



Finished?

Is this set of set of paths giving a solution?

Yes: we are done

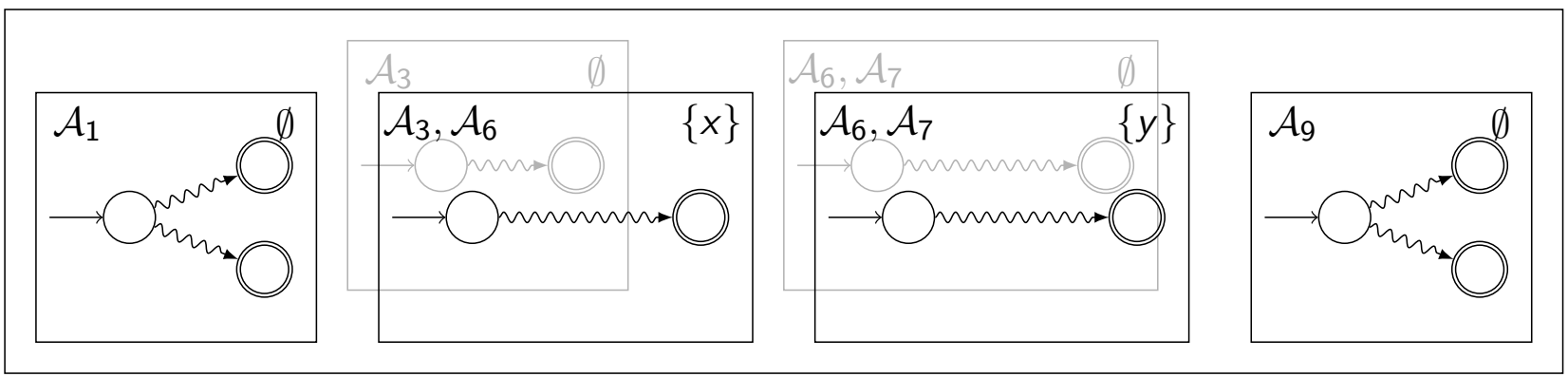
No: add paths or merge sets

Backtracking

Concretisation: limiting exploration when adding paths to a set

In practice:

- only actions from the automata added at the very last concretisation step can be added,
- adding actions from other automata requires to backtrack (go back before previous concretisation steps),
- hence we record an *history* of concretisation steps



Backtracking

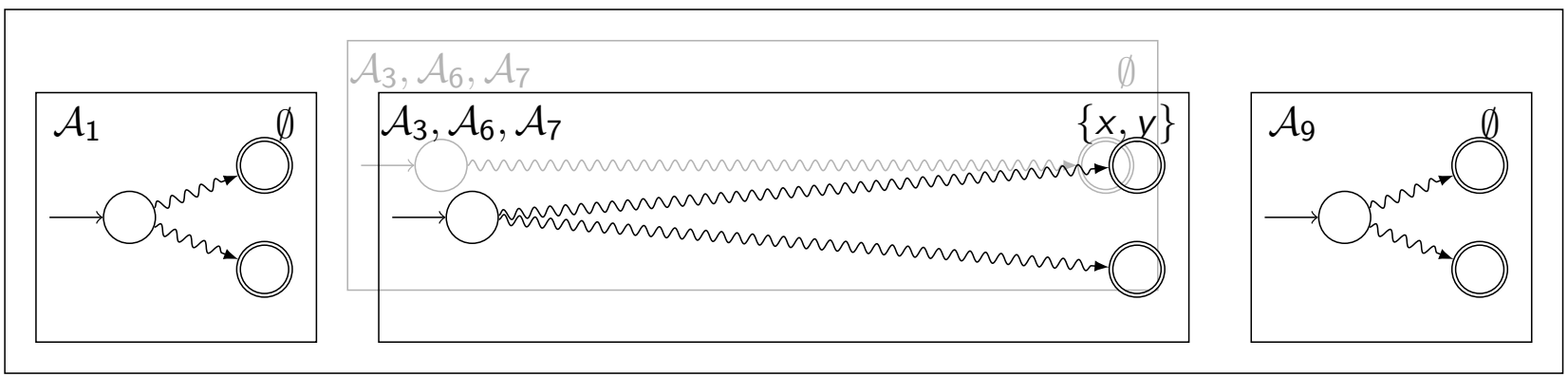
Concretisation: limiting exploration when adding paths to a set

In practice:

- only actions from the automata added at the very last concretisation step can be added,
- adding actions from other automata requires to backtrack (go back before previous concretisation steps),
- hence we record an *history* of concretisation steps

Link with merging

Do not merge sets of paths but *histories* of sets of paths



Laziness

Early finding of a solution

When completeness and consistency are achieved together

- not (necessarily) all automata involved
- not (necessarily) all clocks from involved automata considered

Laziness

Early finding of a solution

When completeness and consistency are achieved together

- not (necessarily) all automata involved
- not (necessarily) all clocks from involved automata considered

Early detection of absence of solution

When no path can be added at the beginning of an history

- not (necessarily) all automata involved
- not (necessarily) all clocks from involved automata considered
- not (necessarily) all merging done

II. Experimental results

LaRA-T: Lazy Reachability Analyzer

About LaRA-T

- About 2000 lines of Haskell code,
- built over the previous LaRA tool

Implementation choices

- Immediately take full sets of paths (products of TAs)
- Reachability with DBM-based symbolic state exploration
- Add only one automaton/clock at a time
- No shared clocks

Try it!

lara.rts-software.org

Experimental setting

- Runtime comparison with Uppaal
- 24 Core computer with 128GB of memory
- 20 minutes time limit for each instance of each problem

Problems

- CritReg: critical region protocol from PAT benchmarks
- Fddi: token/ring protocol from KRONOS benchmarks
- Fischer: fischer protocol as modeled in Uppaal papers
- Fischer2: broken fischer protocol
- Trains1: train model from Uppaal
- Trains2: same model but with a set of trains instead of a queue
- Trains3: railway crossing road from Berthomieu and Vernadat

Experimental results

Largest instance solved within 20 minutes

	CritReg	Fddi	Fischer	Fischer2	Trains1	Trains2	Trains3
LaRA-T	≥ 1500	≥ 5000	7	≥ 500	8	13	7
LaRA-T Full	4	15	6	5	8	13	5
Uppaal	46	13	13	65	10	16	6

Automata and clocks considered for solving instances of size n

	CritReg		Fddi		Fischer		Fischer2		Trains1		Trains2		Trains3	
	A	C	A	C	A	C	A	C	A	C	A	C	A	C
LaRA-T	3	1	4	5	$n+1$	n	3	2	3	3	3	3	$n+2$	3
total	$2n+1$	n	$n+1$	$3n+1$	$n+1$	n	$n+1$	n	$n+1$	n	$n+1$	n	$n+2$	$n+2$

To conclude

Conclusion and future work

What we have done

- An "as generic as possible" algorithm for lazy reachability analysis in distributed timed systems
- An early prototype giving promising results

What we are doing now

- Compute incomplete partial products in our prototype

What we plan to do next

- Parametric timed systems
- Parallel implementation
- Non-synchronous clocks?