Une comparaison de modèles pour les systèmes répartis temps-réel

Thomas Chatain

LSV, ENS Cachan, INRIA, CNRS

journée SED, CNAM, 26 juin 2014

Introduction

More and more complex information systems

Need for reliable techniques to

- design
- implement
- test
- maintain
- supervise

Formal methods offer a way to deal with their complexity

- Adapted to a variety of domains like design, model-checking, test and supervision
- Finite representation of the state space
- Formal definition of the semantics

Two challenging aspects

Concurrency

Systems more and more distributed

- combinatorial explosion
- partial-order techniques

Real-time

Time plays an important role in the behavior, for instance when execution time is critical

- combinatorial explosion
- symbolic techniques (state classes, zones, DBMs...)

Specific difficulties arise in presence of both concurrency and real-time

- The implicit synchronization due to time progress prevents independent actions from permuting freely.

Overview

Several formalisms

- (Safe) time Petri nets
- Networks of timed automata
- ...

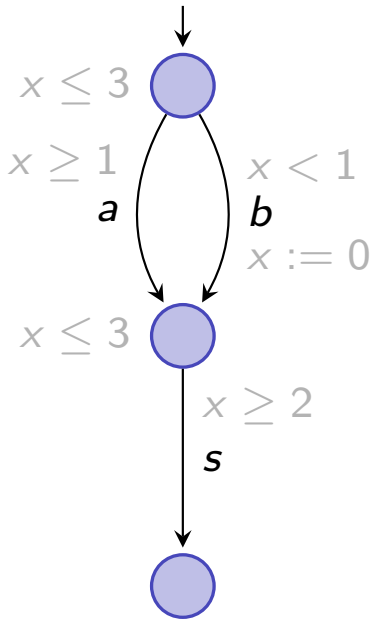
Several transformations proposed

- Use tools designed for another formalism
- Comparisons of expressiveness
- Usually, concurrency not taken into account
- Comparisons in terms of sequential semantics

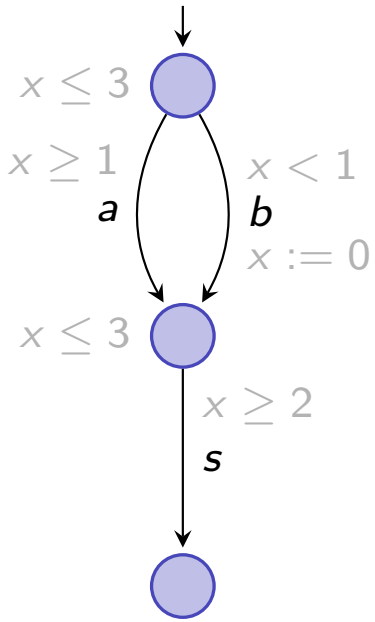
Preservation of distribution

- Transformation from safe TPN to NTA
- Avoiding shared clocks in NTA

Timed automata

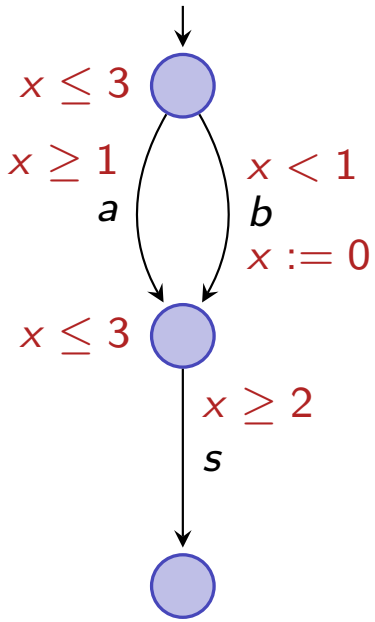


Timed automata



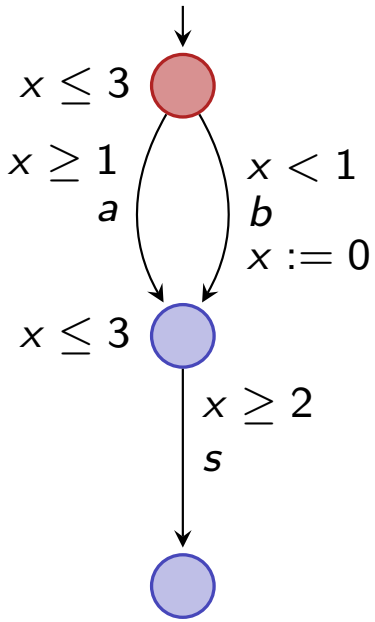
Timed automata

Real-valued clocks constrain the bahavior



Timed automata

Real-valued clocks constrain the bahavior

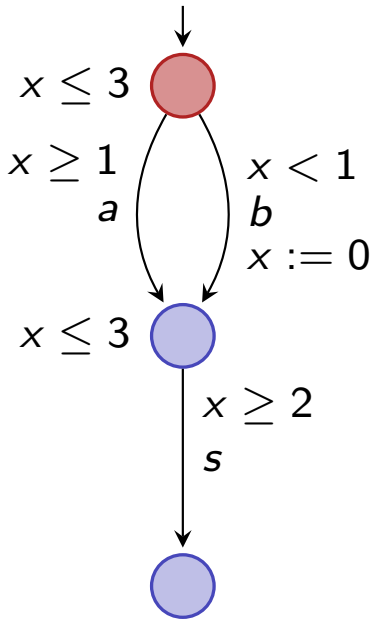


$x = 0.0$

run:

Timed automata

Real-valued clocks constrain the bahavior

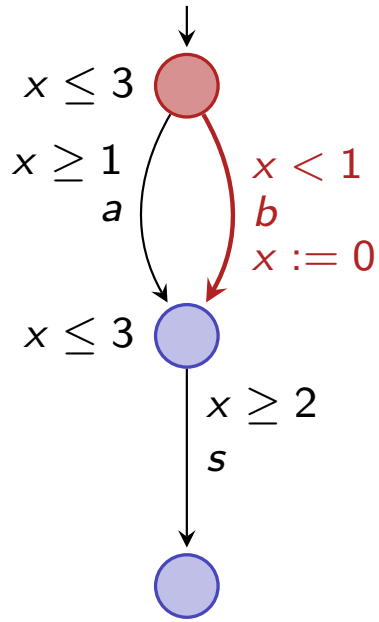


$x = 0.5$

run:

Timed automata

Real-valued clocks constrain the bahavior

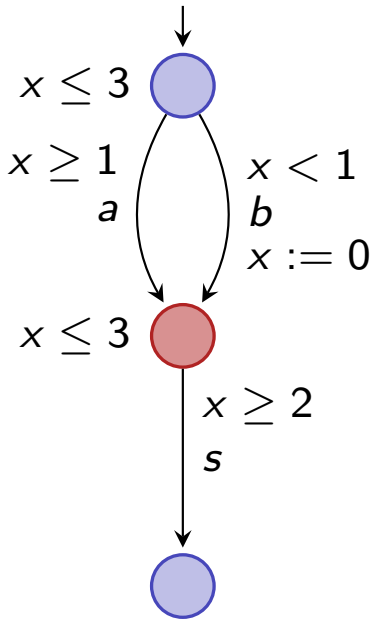


$x = 0.5$

run:

Timed automata

Real-valued clocks constrain the bahavior

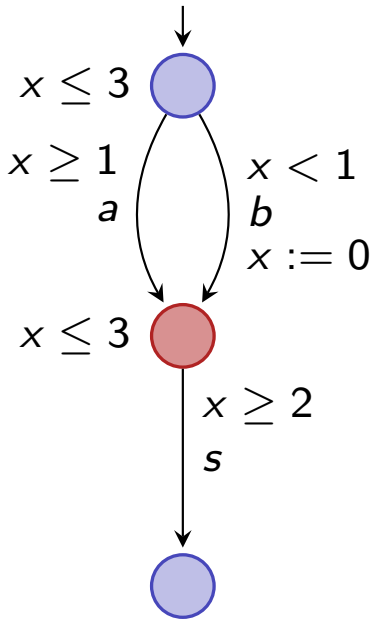


$x = 0.0$

run: $(0.5, b)$

Timed automata

Real-valued clocks constrain the bahavior

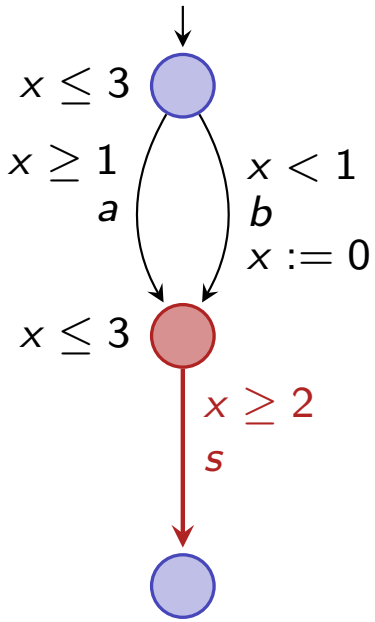


$x = 2.5$

run: $(0.5, b)$

Timed automata

Real-valued clocks constrain the bahavior

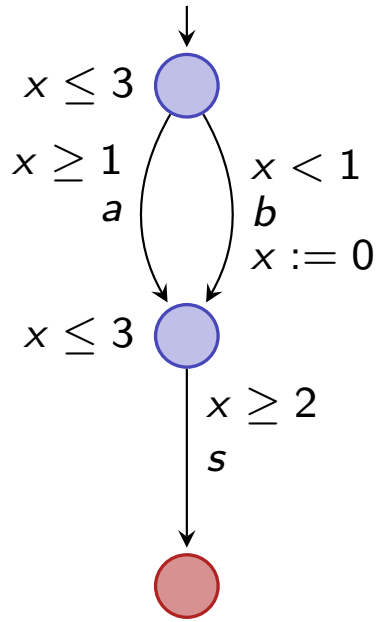


$x = 2.5$

run: $(0.5, b)$

Timed automata

Real-valued clocks constrain the bahavior



$x = 2.5$

run: $(0.5, b), (3, s)$

Timed automata

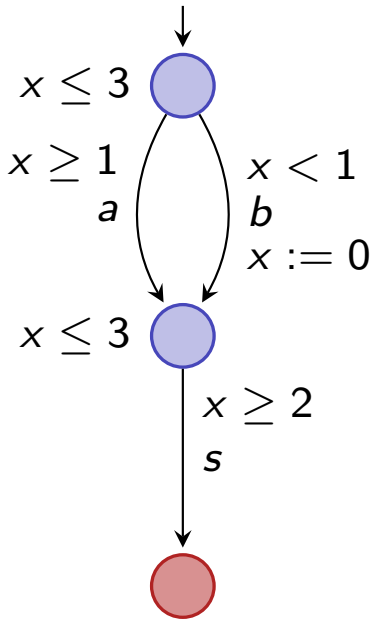
Real-valued clocks constrain the bahavior

A few results

- TA lie in between automata (where many problems are decidable) and hybrid automata (where almost none are)
- No determinization, no complement
- Language universality, inclusion, equivalence undicadable
  - PSPACE-complete for deterministic TA
- Reachability PSPACE-complete
- Time-abstract language is regular (region automaton)

Many verification tools

- Uppaal, Epsilon, Kronos...

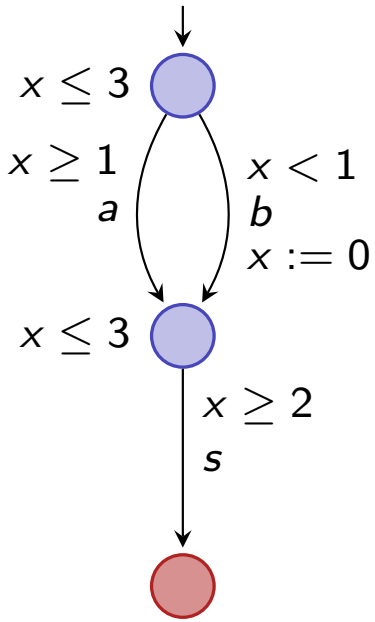


$x =$

run: $(0.5, b), (3, s)$

Timed automata

Real-valued clocks constrain the bahavior

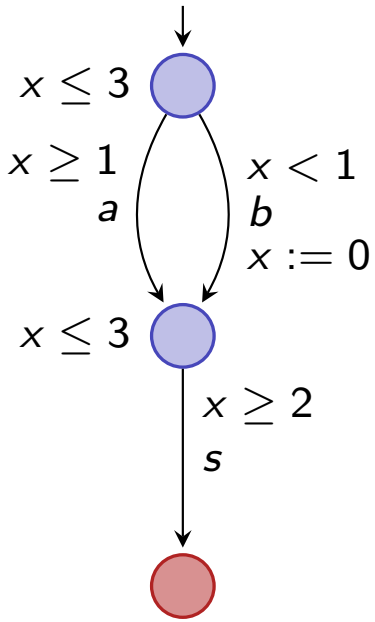


$x =$

run: $(0.5, b), (3, s)$

Timed automata

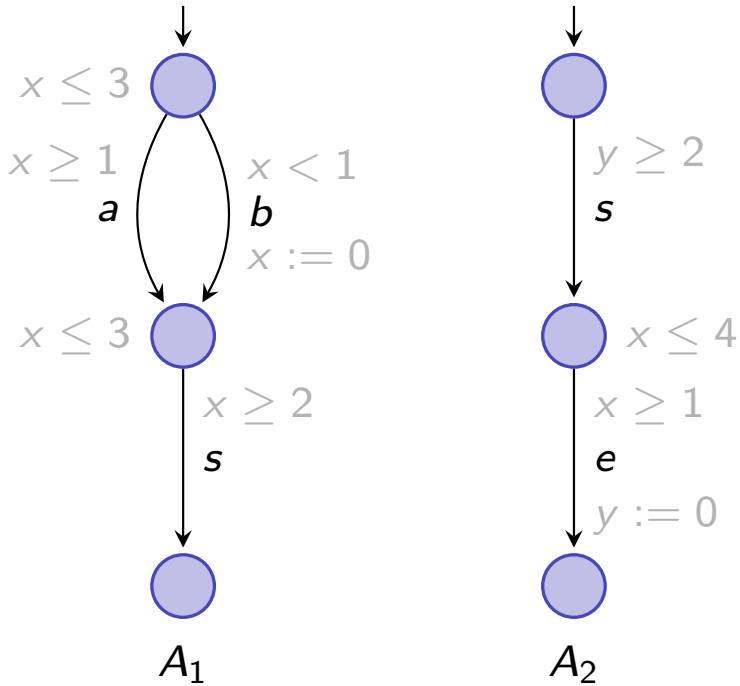
Real-valued clocks constrain the bahavior



$x =$

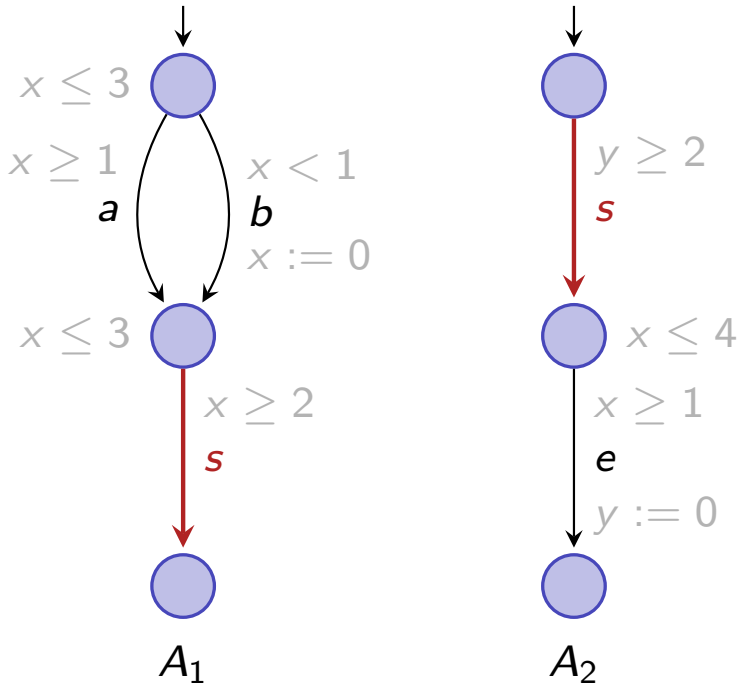
run: $(0.5, b), (3, s)$

Networks of timed automata



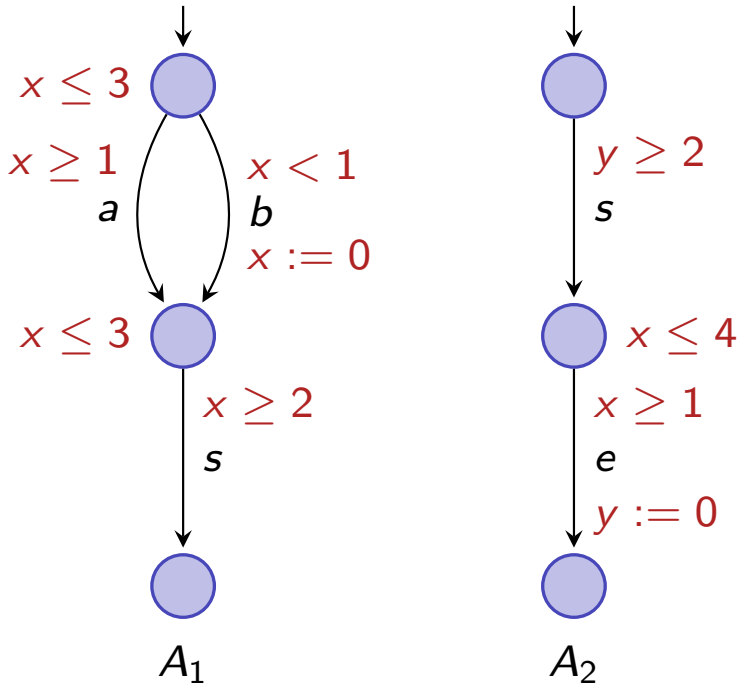
Networks of timed automata

- Synchronizations via common actions



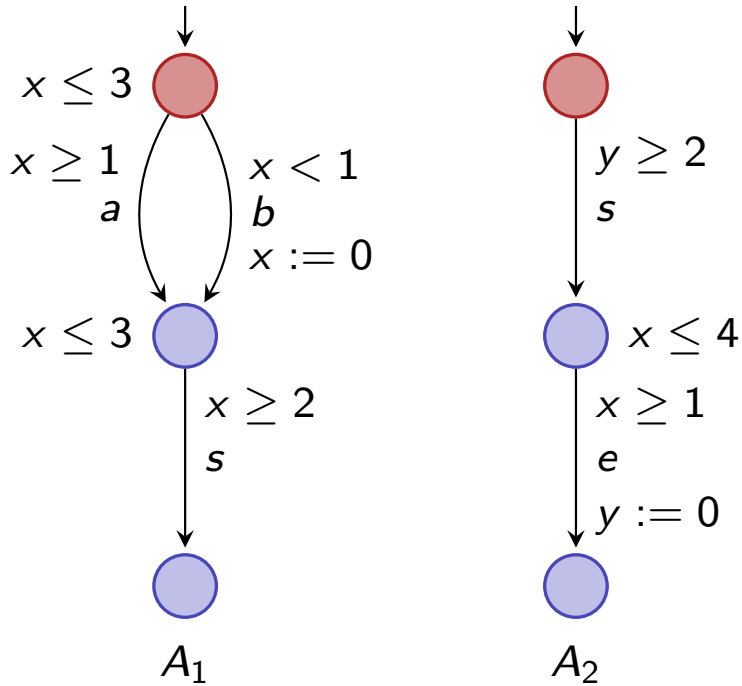
Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior



Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

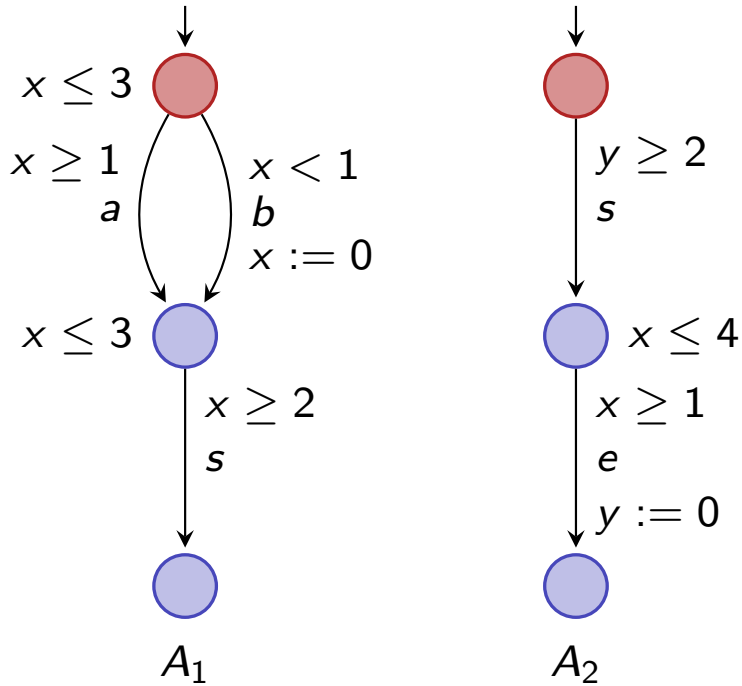


$x = 0.0$ $y = 0.0$

run:

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

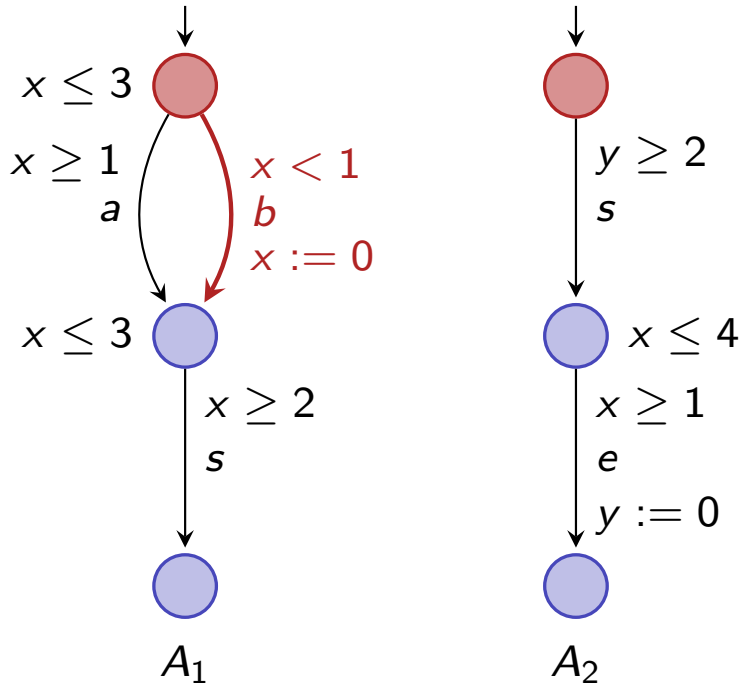


$x = 0.5$ $y = 0.5$

run:

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

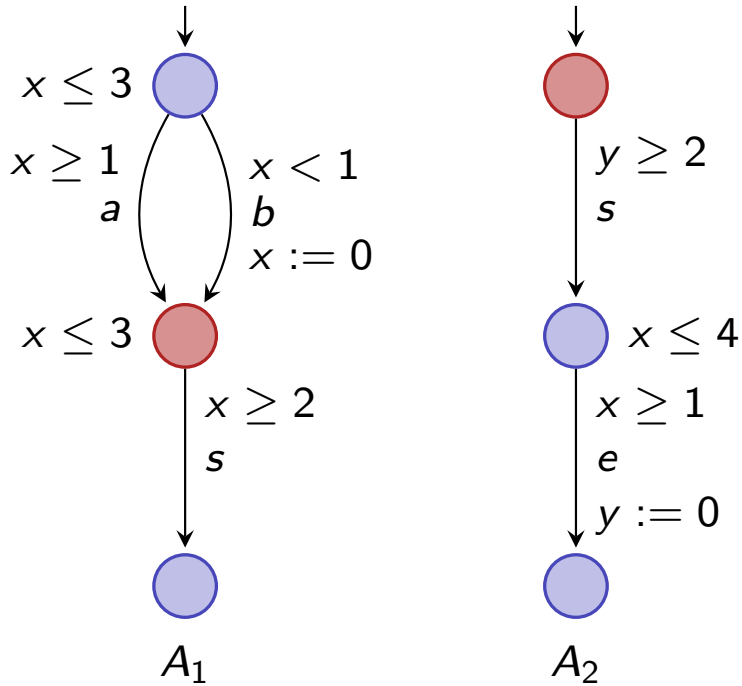


$x = 0.5$ $y = 0.5$

run:

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

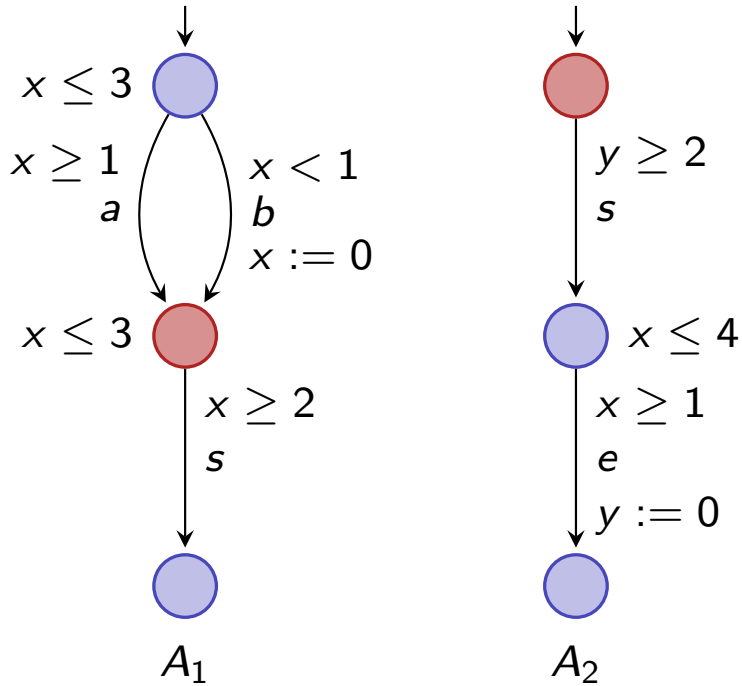


$x = 0.0$ $y = 0.5$

run: $(0.5, b)$

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

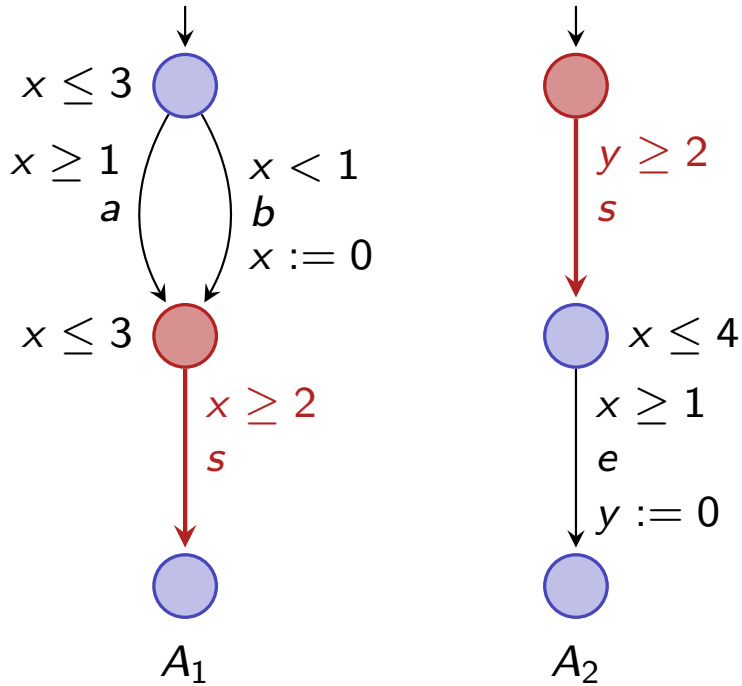


$x = 2.5$ $y = 3.0$

run: $(0.5, b)$

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

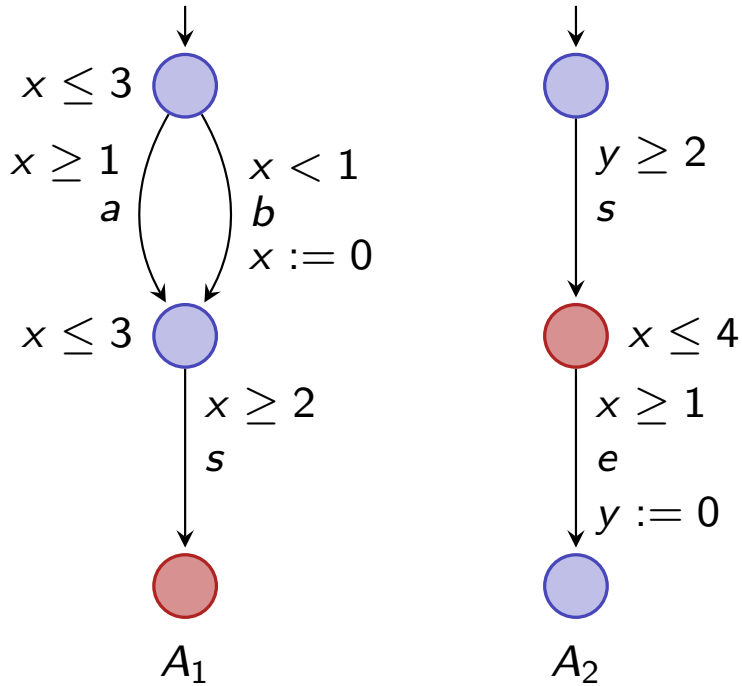


$x = 2.5$ $y = 3.0$

run: $(0.5, b)$

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

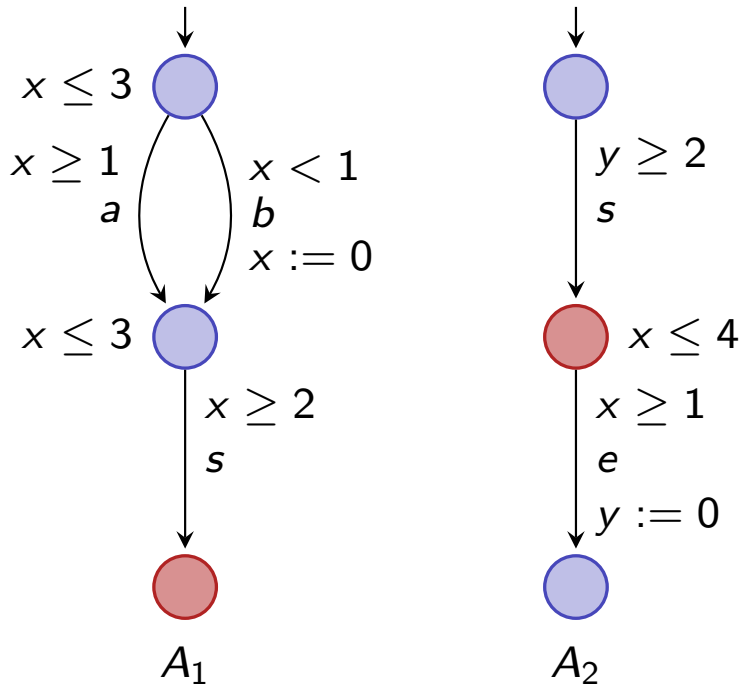


$x = 2.5$ $y = 3.0$

run: $(0.5, b), (3, s)$

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

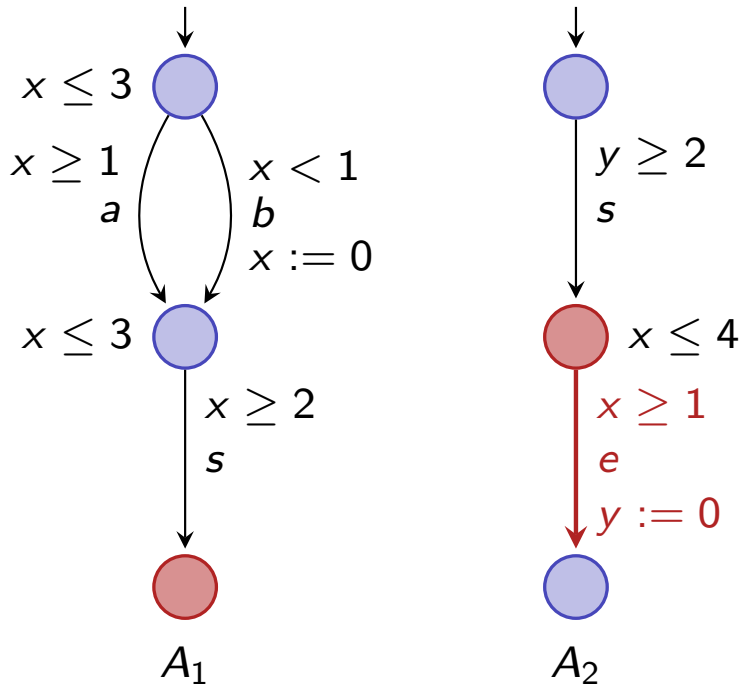


$x = 3.0$ $y = 3.5$

run: $(0.5, b), (3, s)$

Networks of timed automata

- Synchronizations via common actions
- Real-valued clocks constrain the bahavior

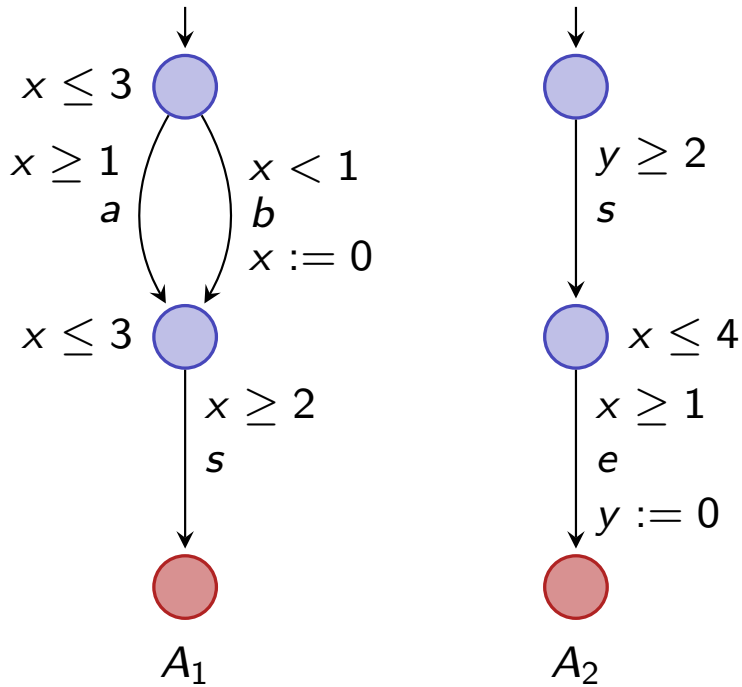


$x = 3.0$ $y = 3.5$

run: $(0.5, b), (3, s)$

Networks of timed automata

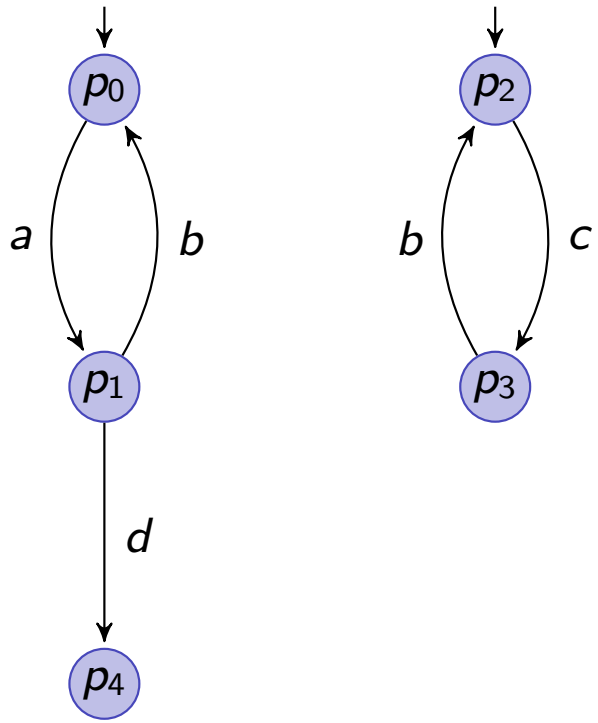
- Synchronizations via common actions
- Real-valued clocks constrain the bahavior



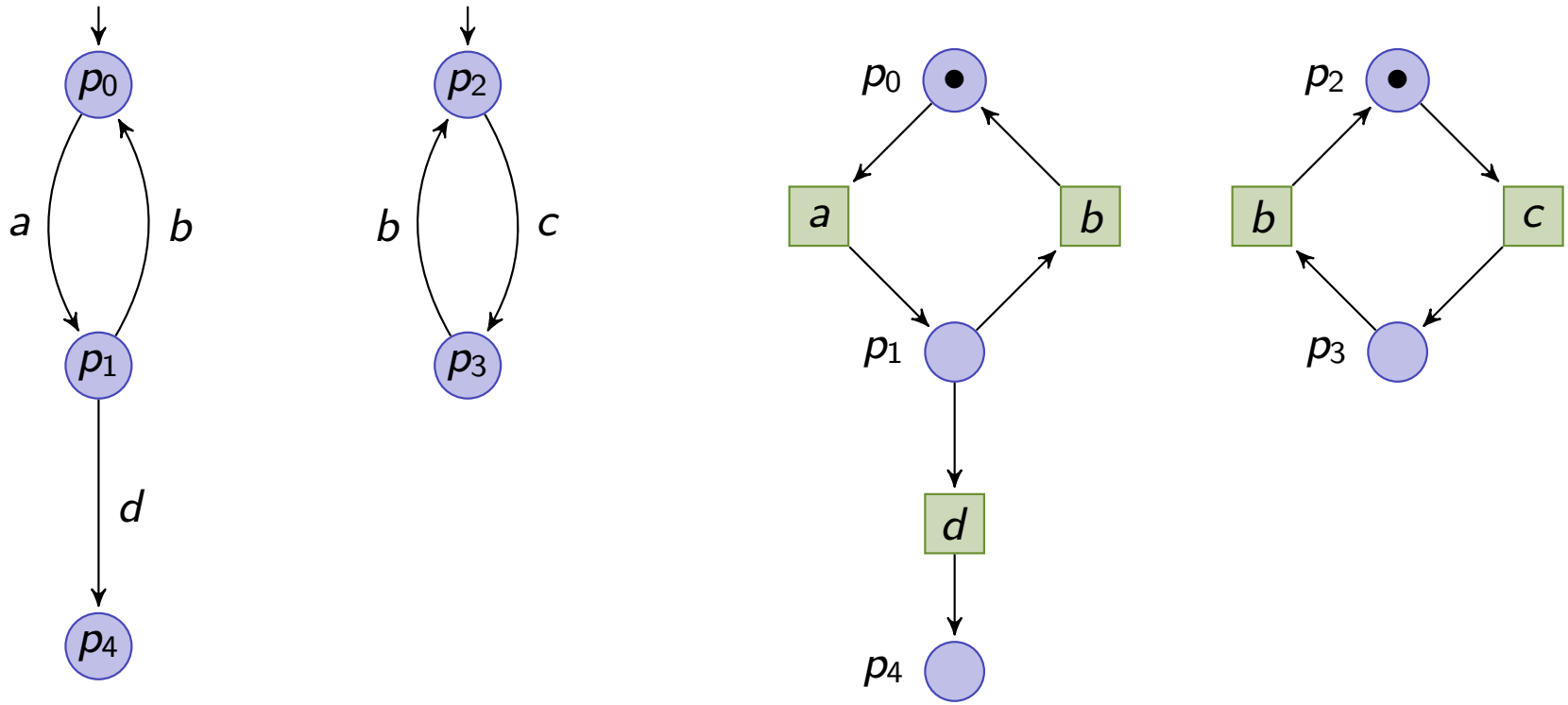
$x = 3.0$ $y = 0.0$

run: $(0.5, b), (3, s), (3.5, e)$

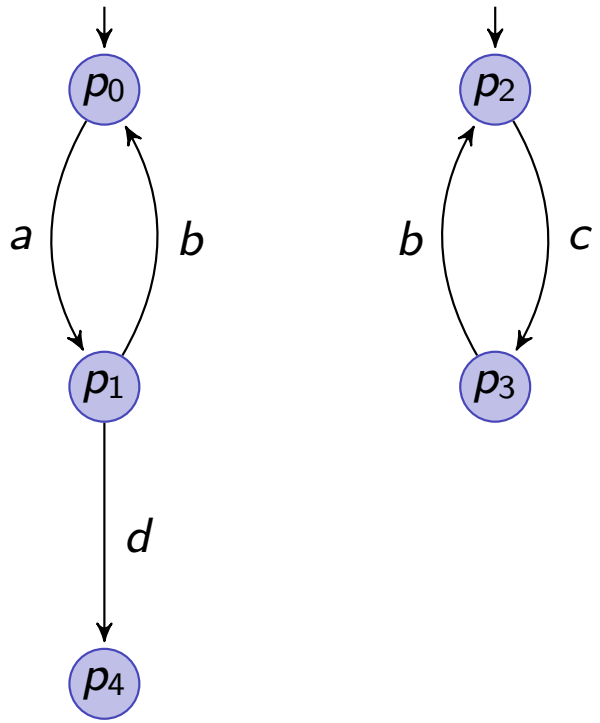
Petri nets: introduction

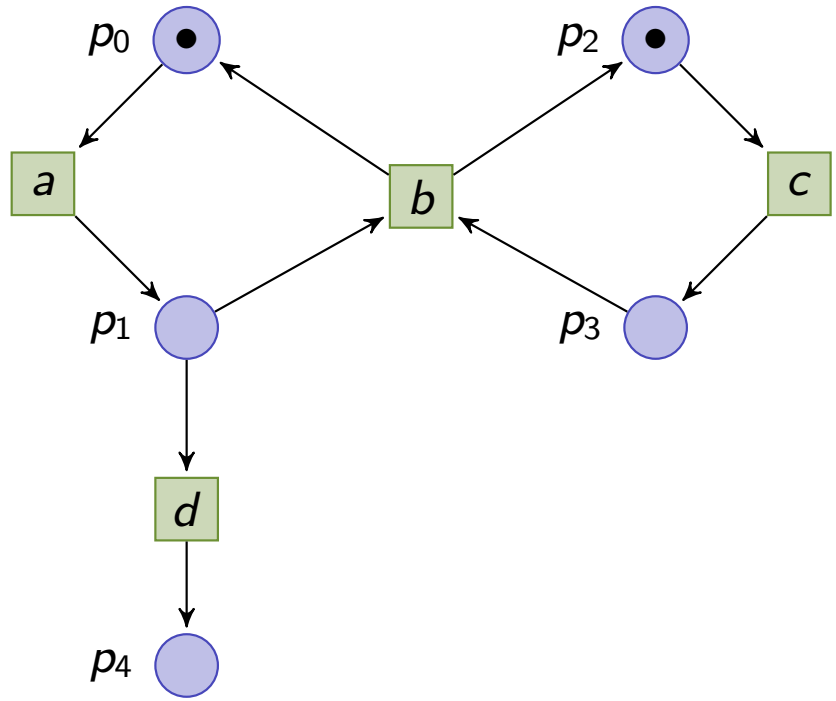


Petri nets: introduction



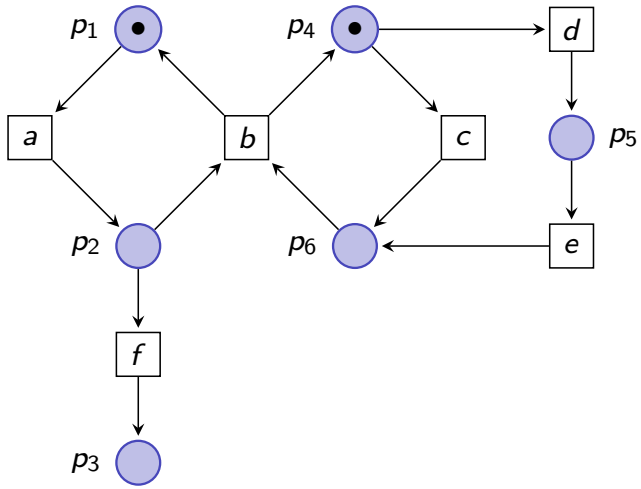
Petri nets: introduction



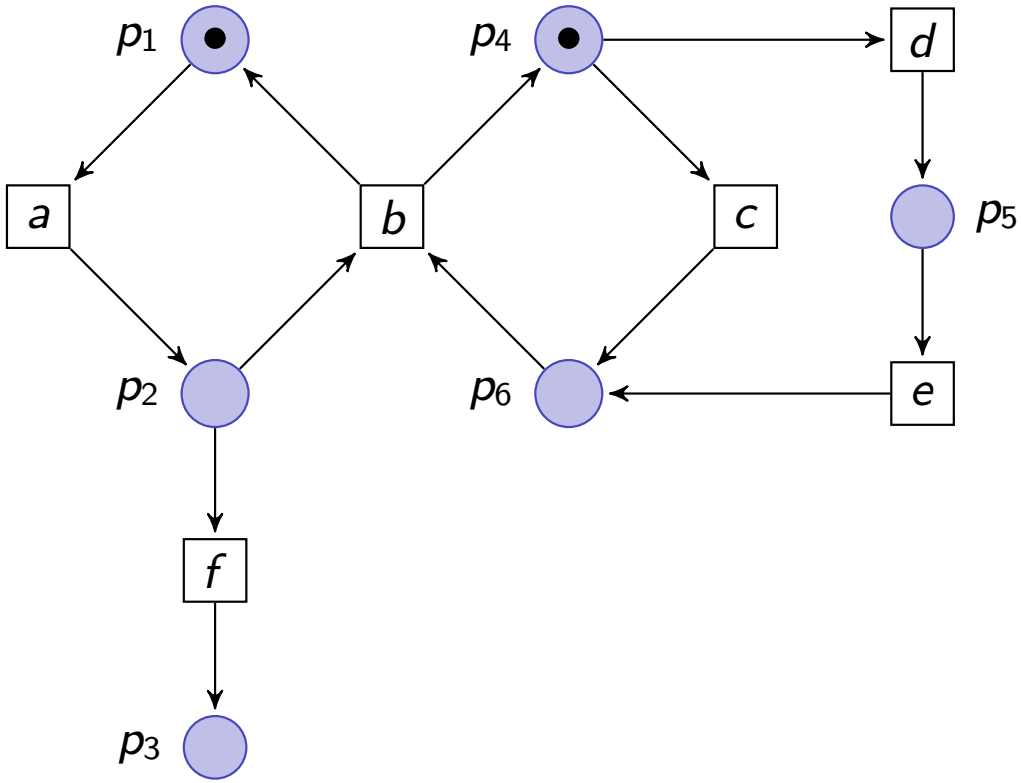


Petri nets

- Specification of distributed systems
- Places
- Transitions
- Tokens (possibly several in a place)

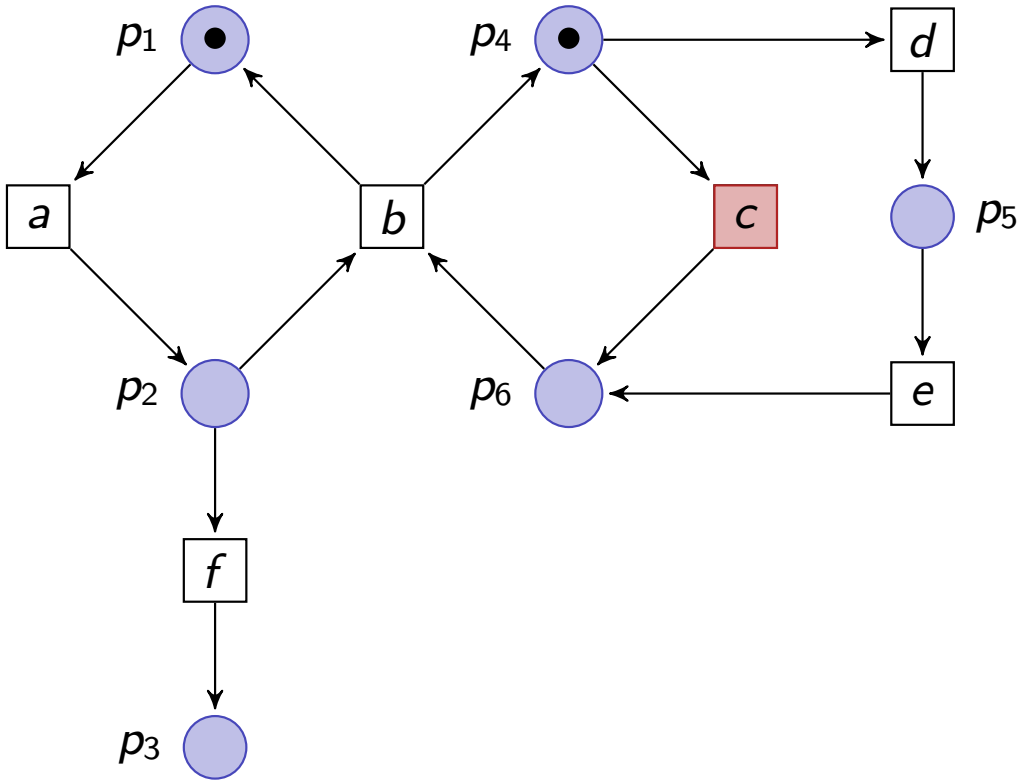


Petri nets semantics



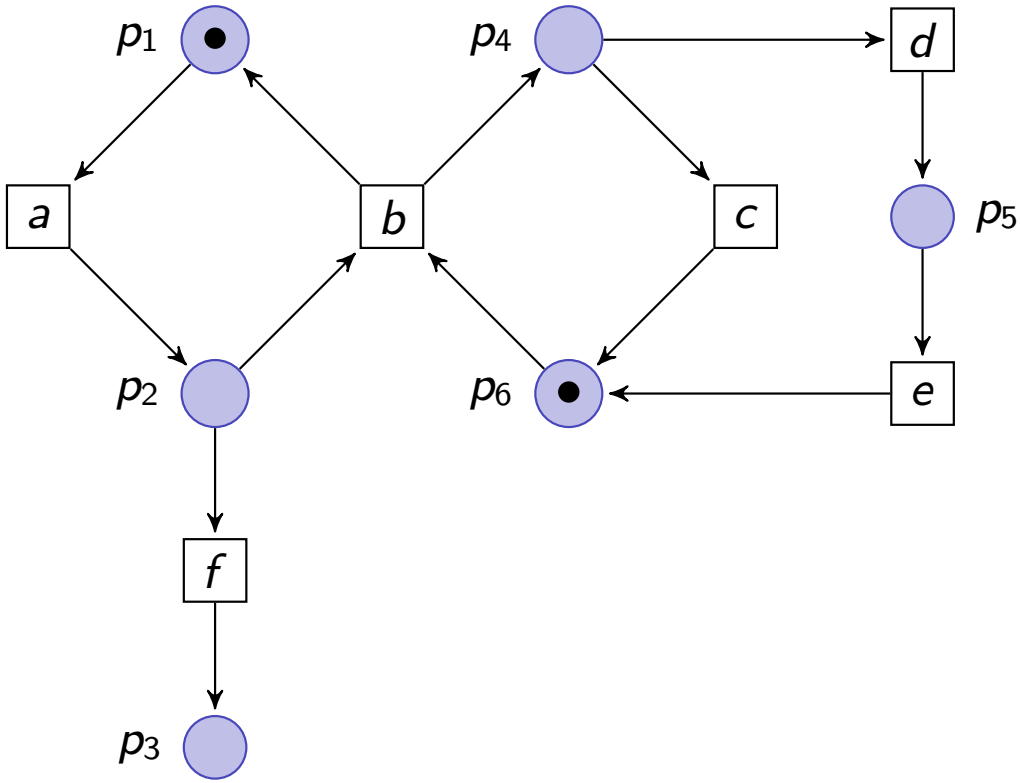
Run:

Petri nets semantics



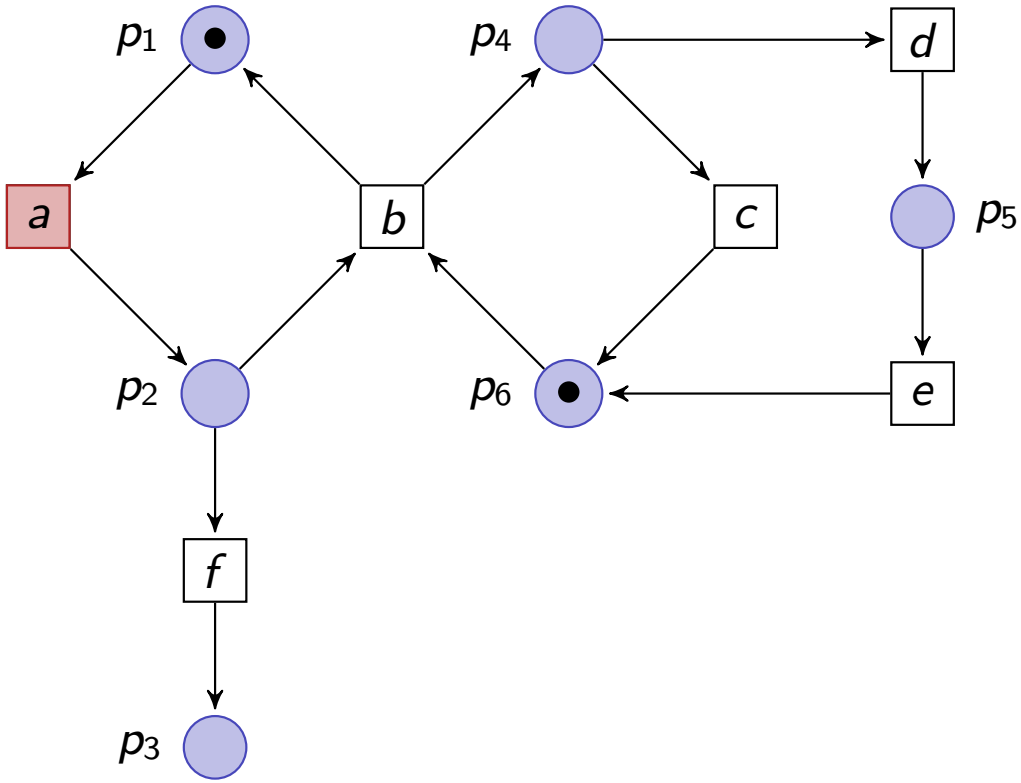
Run: c

Petri nets semantics



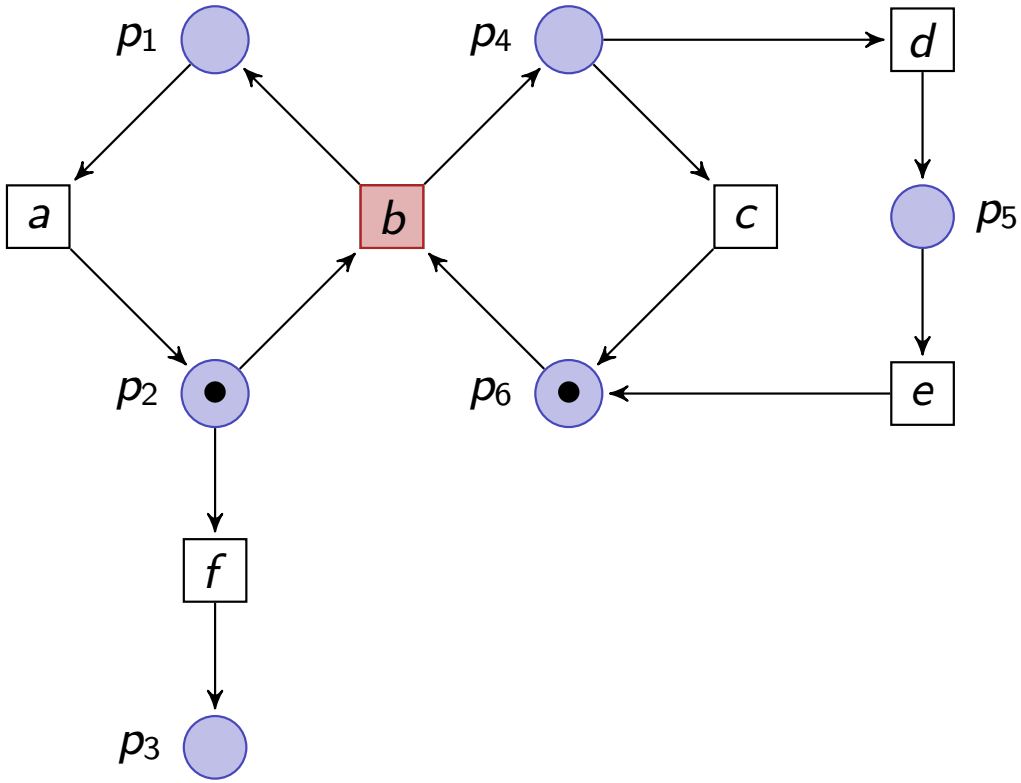
Run: c

Petri nets semantics



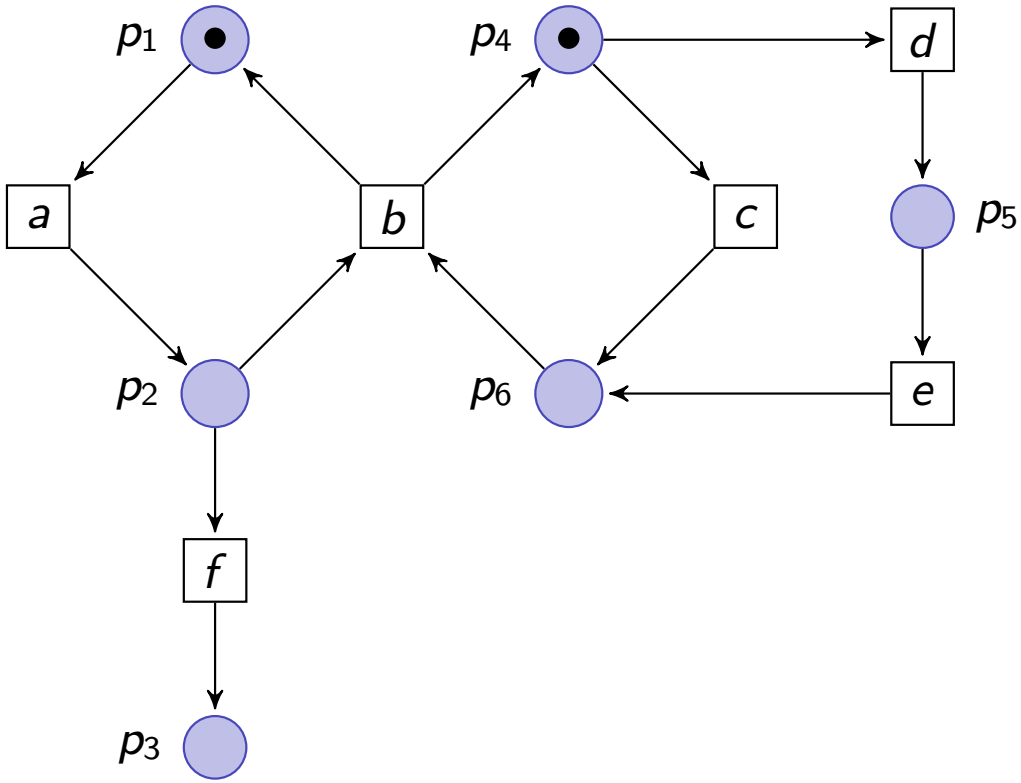
Run: c, a

Petri nets semantics



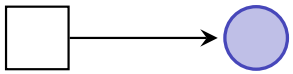
Run: c, a

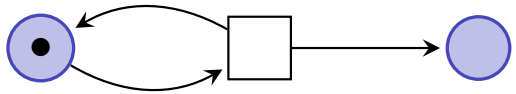
Petri nets semantics



Run: c, a, b

A few results about Petri nets (see survey [Esparza96])





The semantics of Petri nets may put several tokens in a place

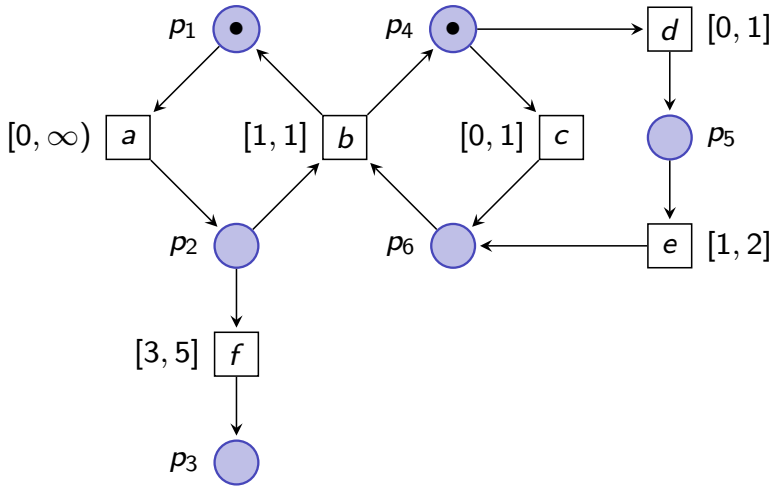
- unsafe Petri net
- unbounded if arbitrarily many tokens

A few results

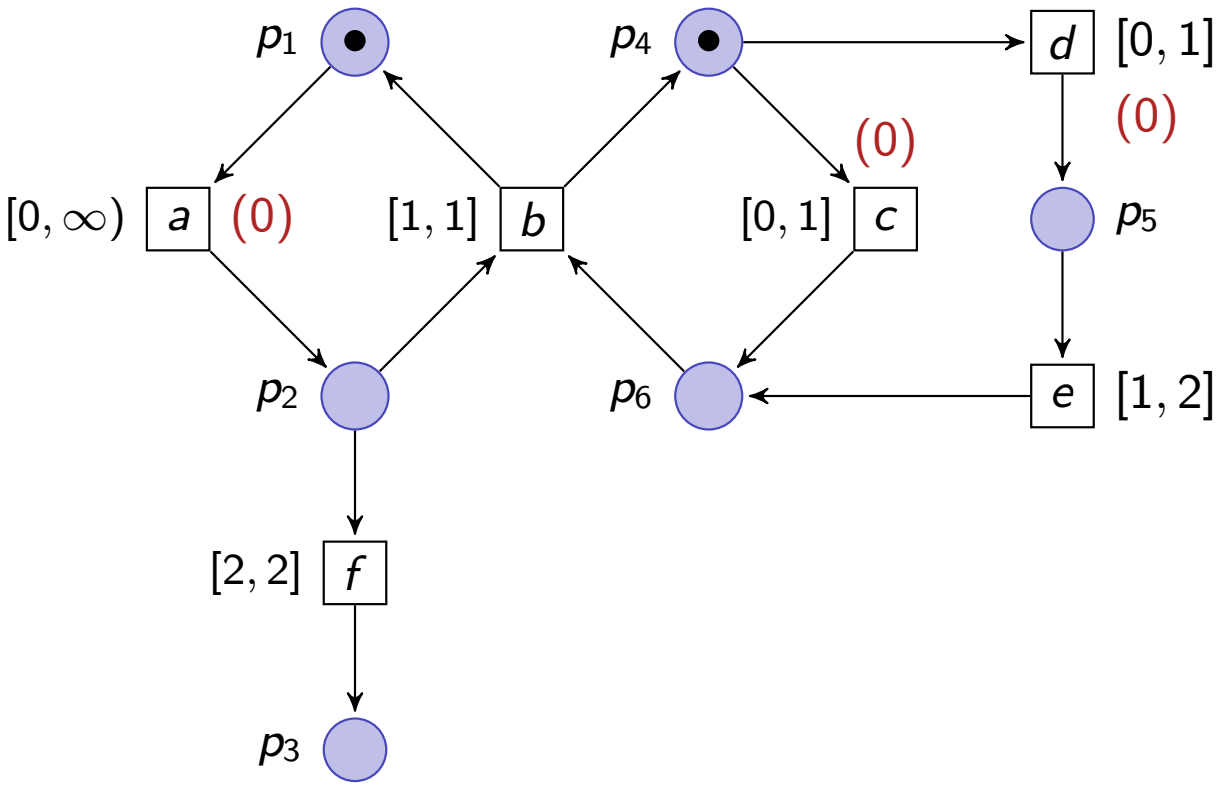
- Reachability is decidable and EXPSPACE-hard (complexity unknown)
- All equivalence problems for Petri nets are undecidable
- k-boundedness for Petri nets is PSPACE-complete
- Nearly all interesting questions about the behavior of k-bounded Petri nets are PSPACE-complete

Time Petri nets

- Specification of real-time distributed systems
- Time constraints: intervals of possible firing delays
  - Clock reset when transition enabled
  - Transition must fire before latest firing delay
  - Strong time semantics

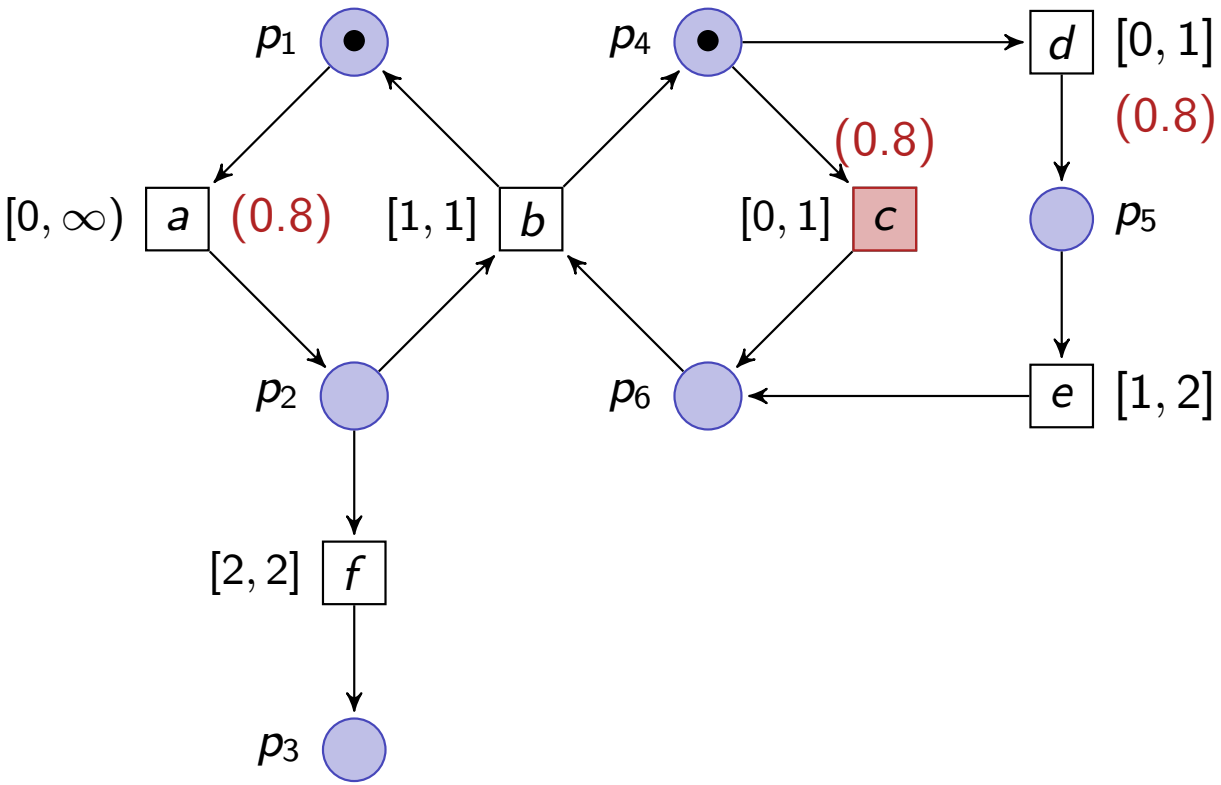


TPN semantics



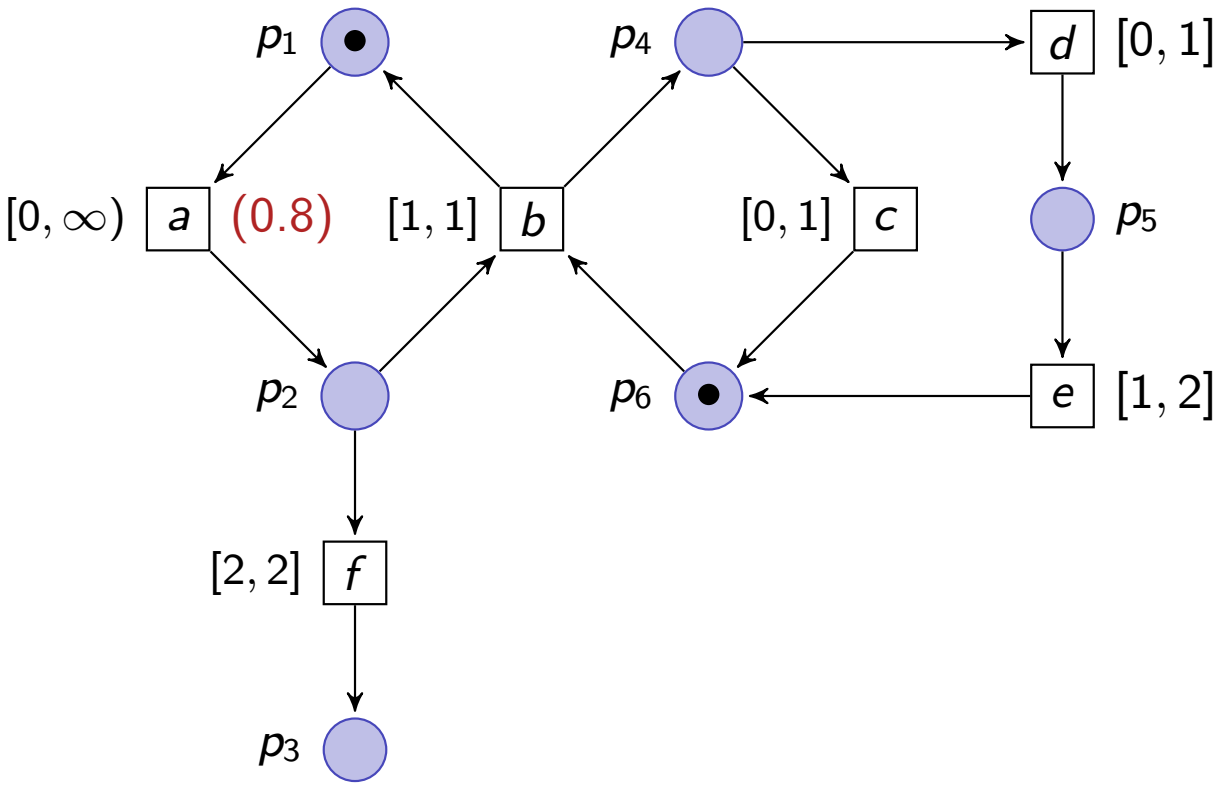
Run:

TPN semantics



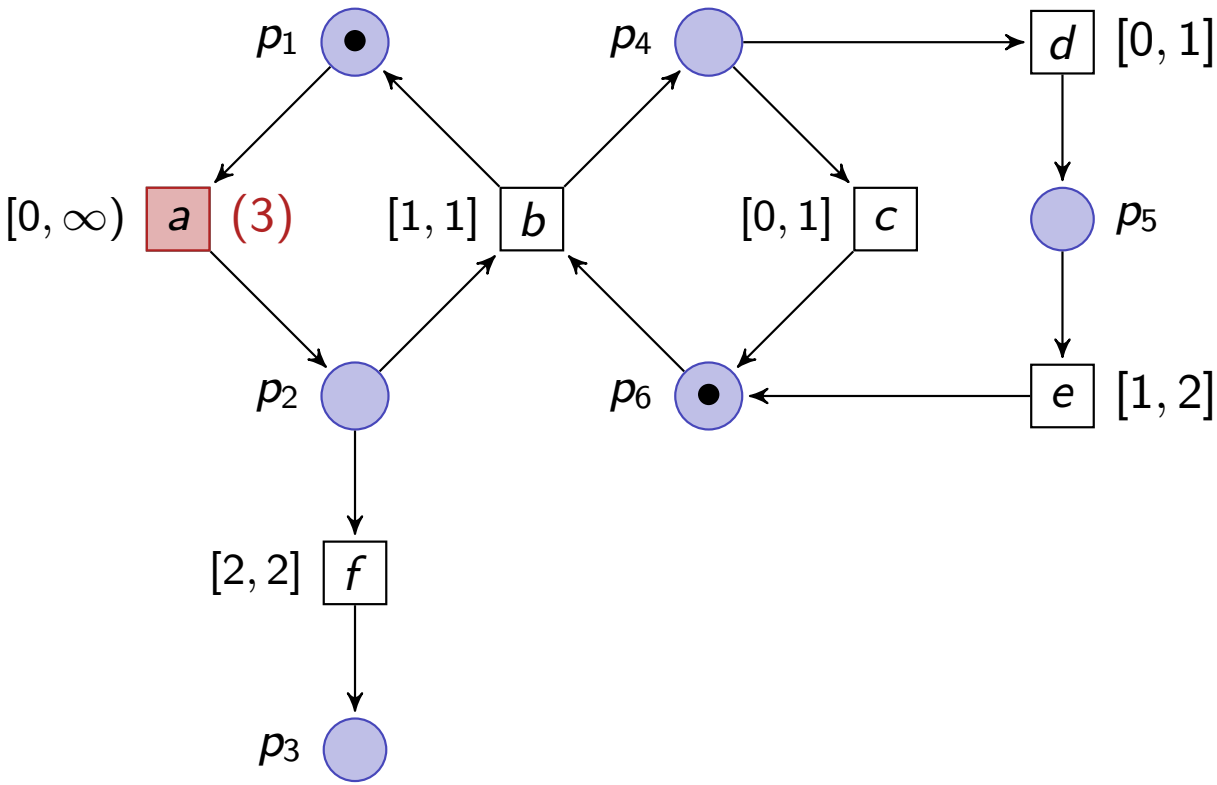
Run: $(c, 0.8)$

TPN semantics



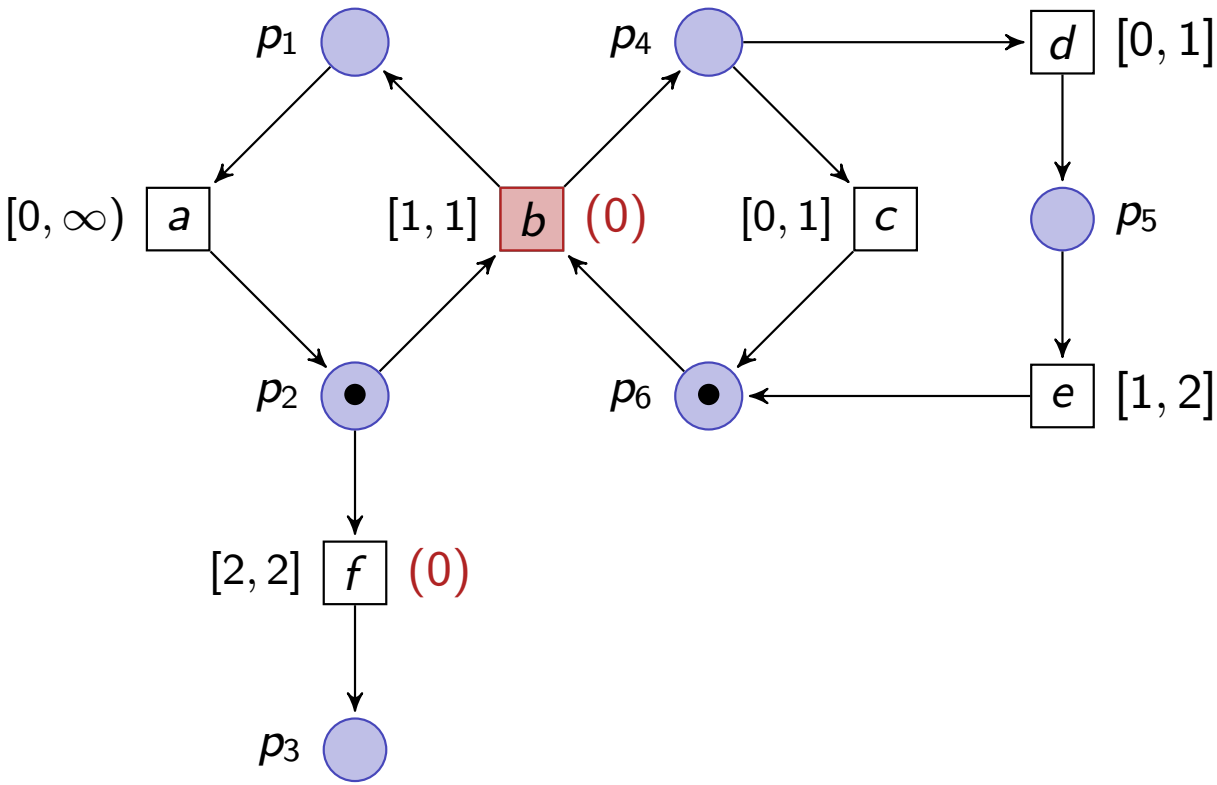
Run: $(c, 0.8)$

TPN semantics



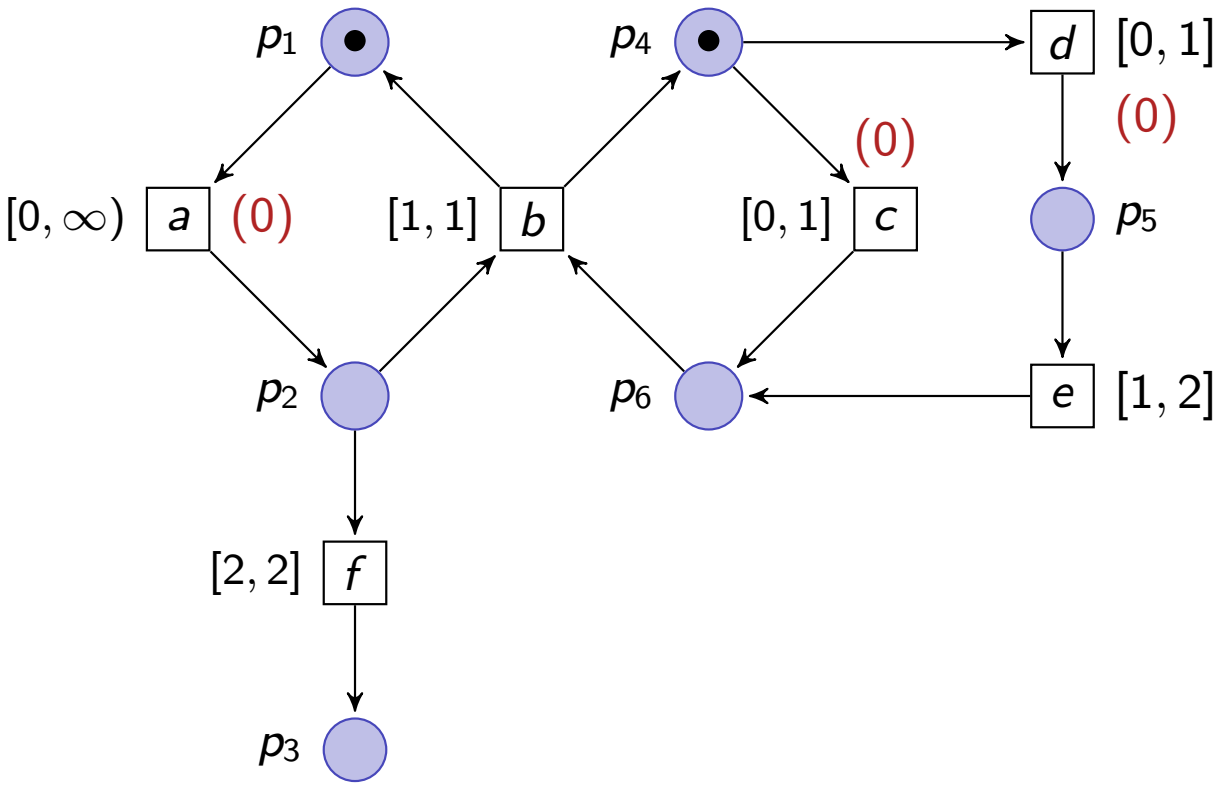
Run: $(c, 0.8), (a, 3)$

TPN semantics



Run: $(c, 0.8), (a, 3)$

TPN semantics

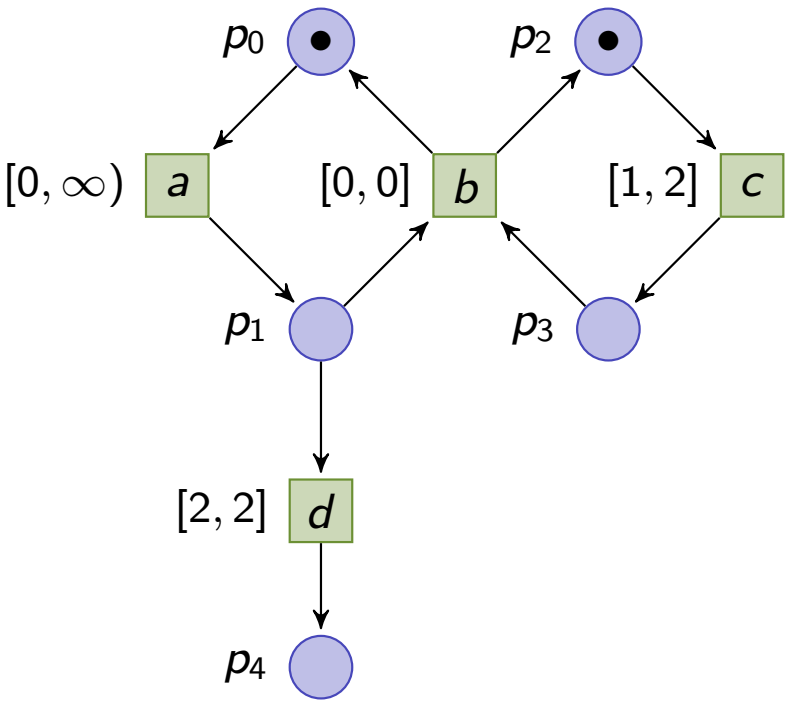


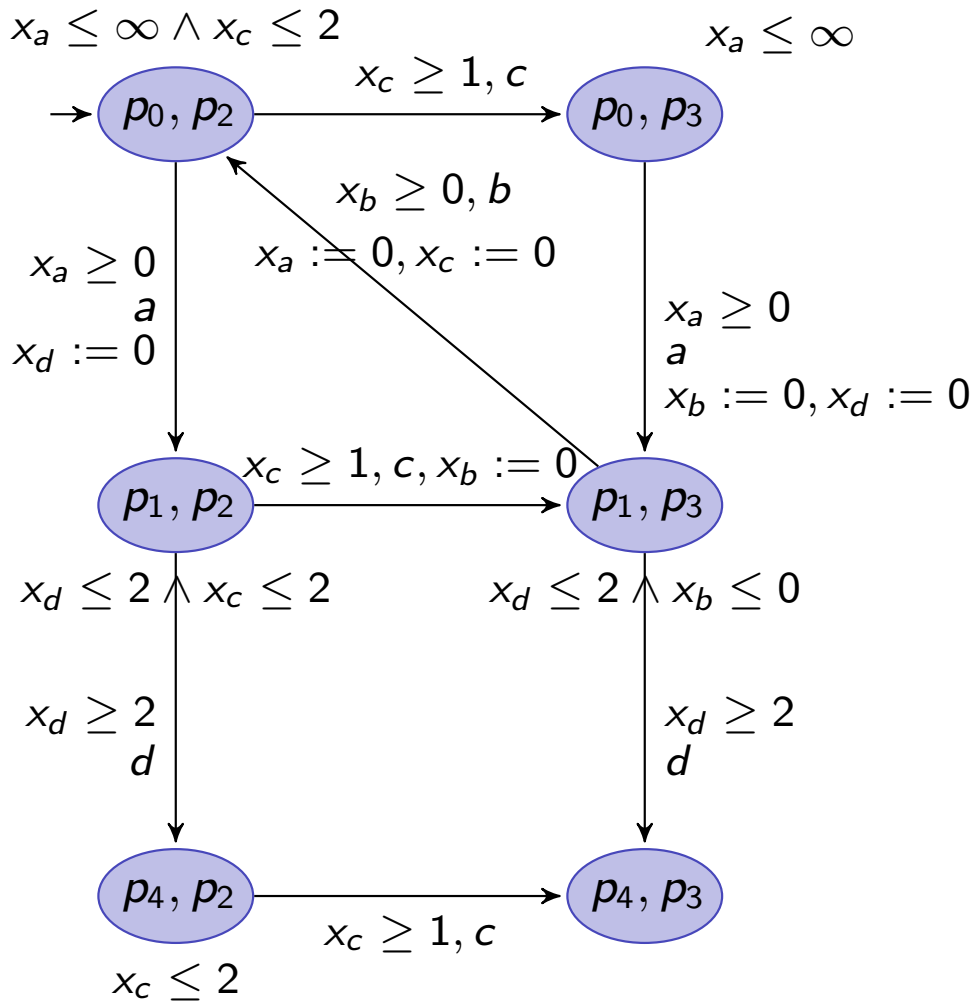
Run: $(c, 0.8), (a, 3), (b, 4)$

A few results about time Petri nets

- Boundedness and reachability undecidable
- k-boundedness PSPACE-complete
- For k-bounded time Petri nets, reachability is PSPACE-complete
- Time-abstract behavior is regular (state class graph)

From safe TPN to TA





Comparisons in terms of sequential semantics

A timed bisimulation relation between two models T_1 and T_2 is a binary relation $\approx$ between their sets of states S_1 and S_2 if

- $s_1^0 \approx s_2^0$ and
- for any $s_1 \approx s_2$, for any $a \in \Sigma \cup \mathbb{R}_{\geq 0}$,
  - if $s_1 \xrightarrow{a}_1 s_1'$, then, for some s_2', $s_2 \xrightarrow{a}_2 s_2'$ and $s_1' \approx s_2'$;
  - if $s_2 \xrightarrow{a}_2 s_2'$, then, for some s_1', $s_1 \xrightarrow{a}_1 s_1'$ and $s_1' \approx s_2'$.

Transformations which preserve distribution

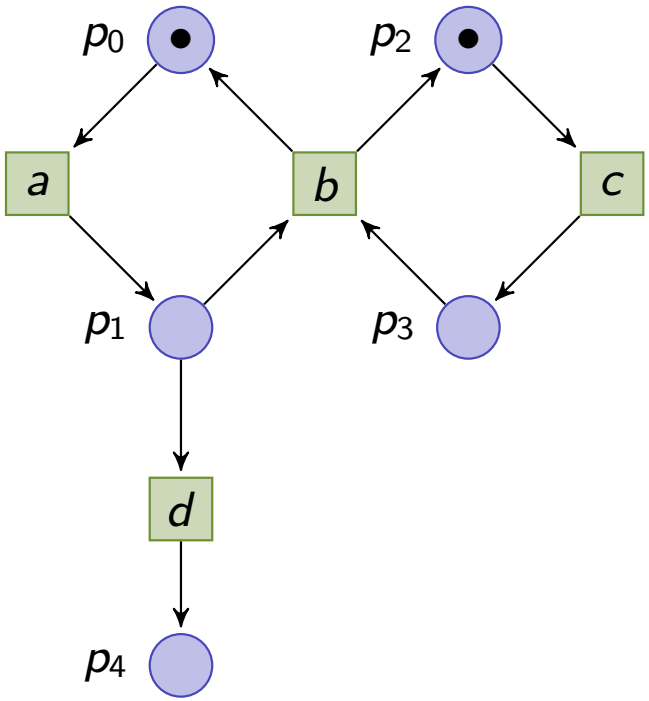
NTA and TPNs represent distributed systems

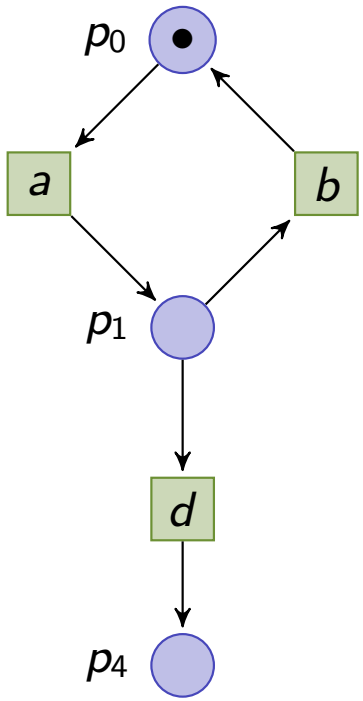
Why preserving distribution?

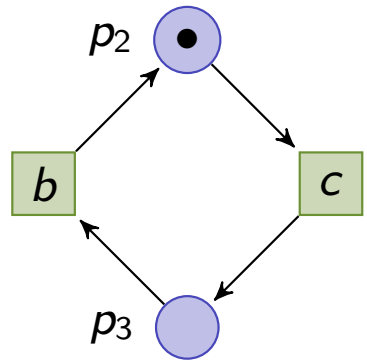
- Readability of the transformations
- Analysis of distributed timed systems exploiting distribution/concurrency
  - avoiding state explosion,
  - using a modular analysis.
- Implementability of models on distributed architectures
  - From high-level to low-level models
  - Preserving distribution
  - Detect unrealistic models

From safe TPN to NTA [BCH10]

Decompose the untimed PN into S-subnets.

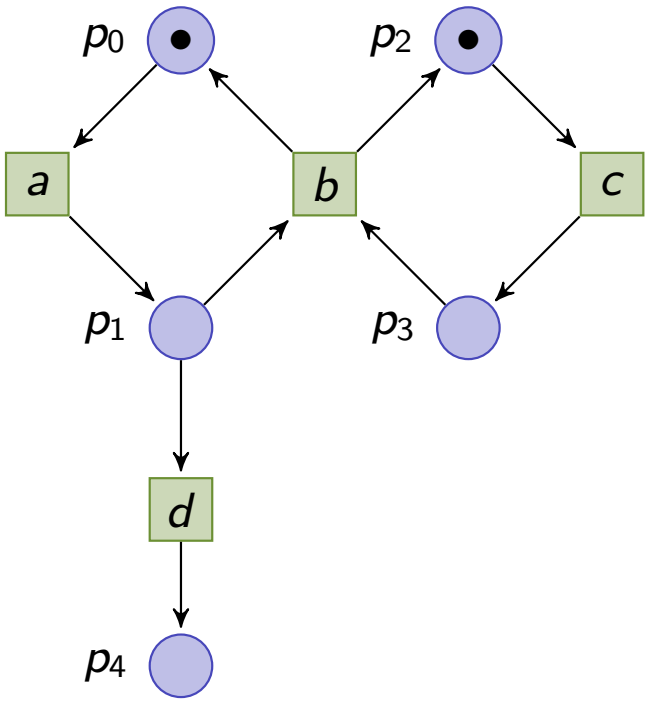


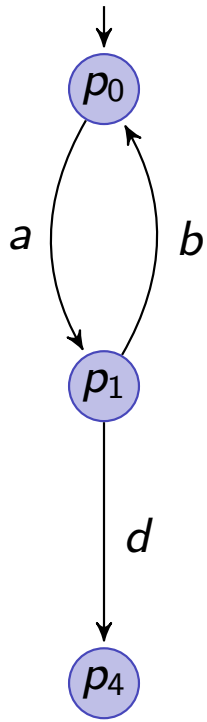


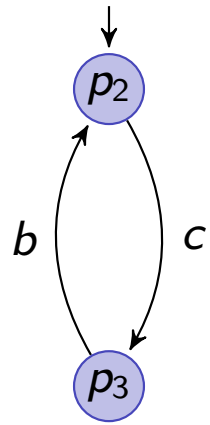


From safe TPN to NTA [BCH10]

Translate each S-subnet into an automaton.

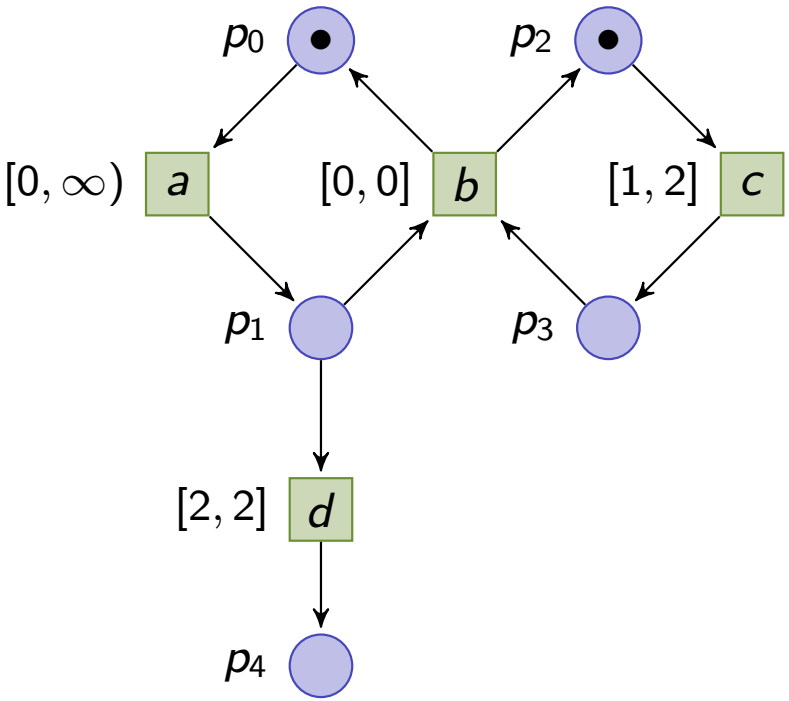


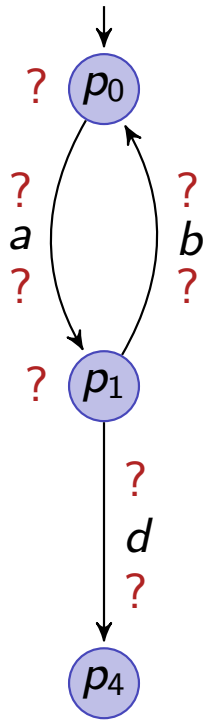


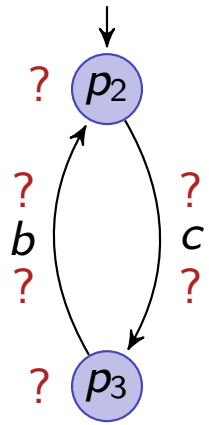


From safe TPN to NTA [BCH10]

Add time constraints.

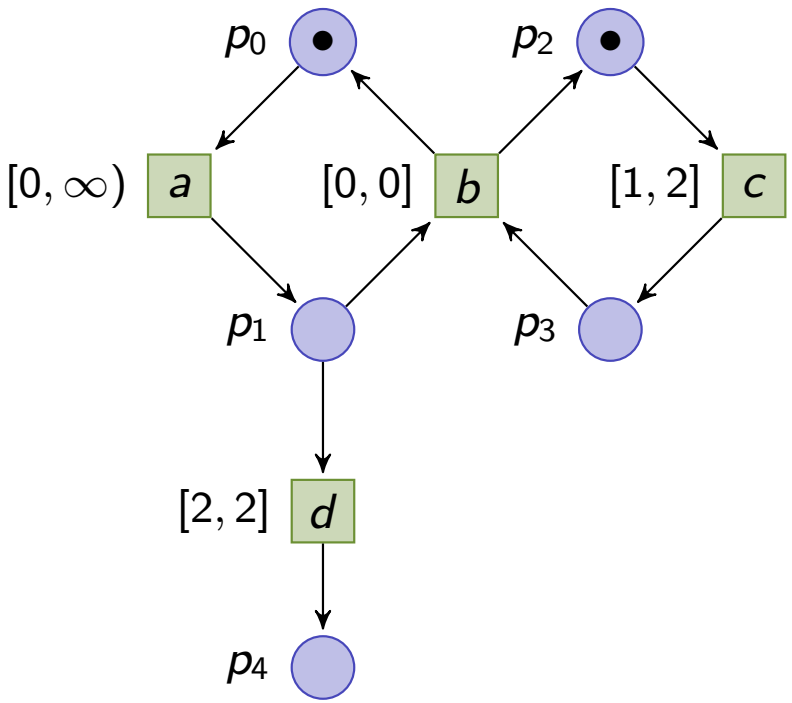


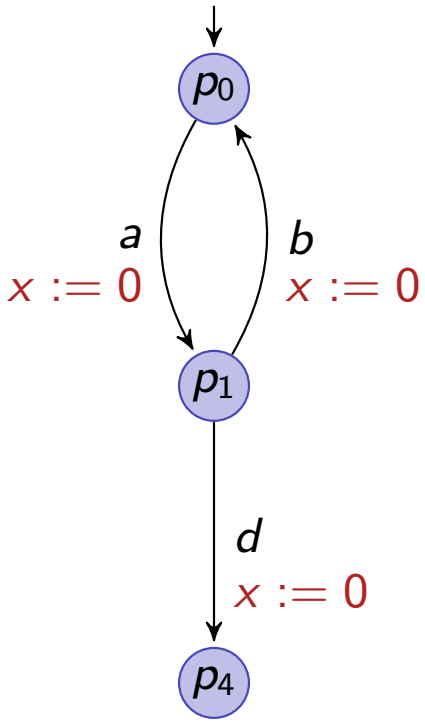


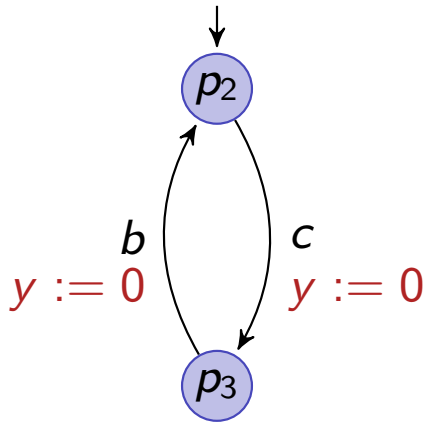


From safe TPN to NTA [BCH10]

Add one clock to each automaton. The clock is reset on each edge.

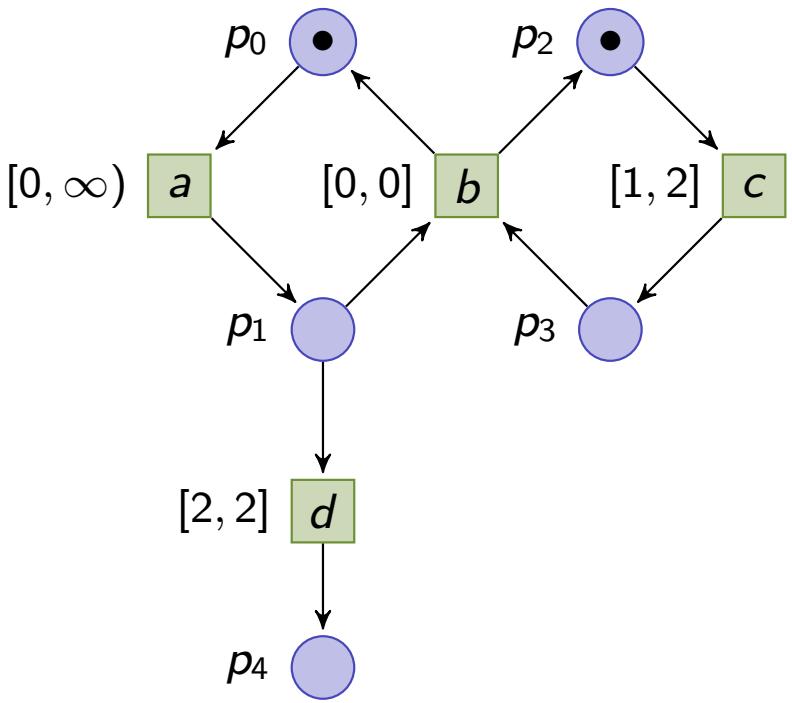


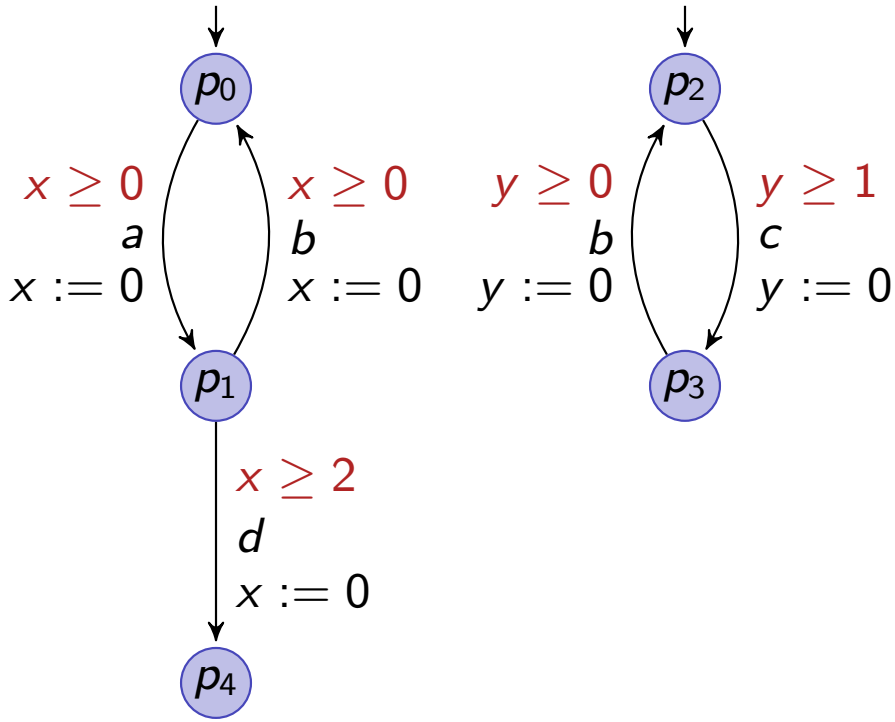




From safe TPN to NTA [BCH10]

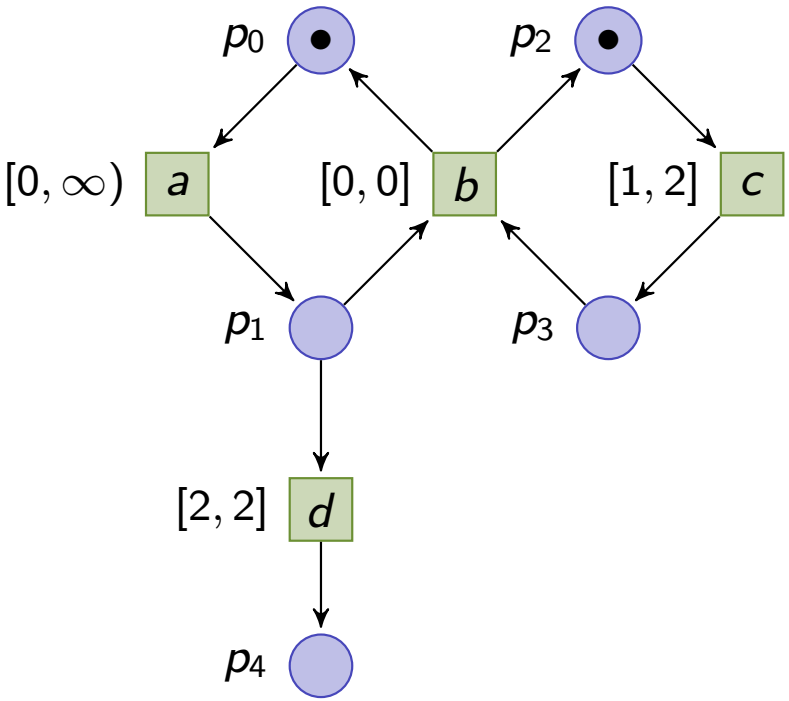
Add guards.

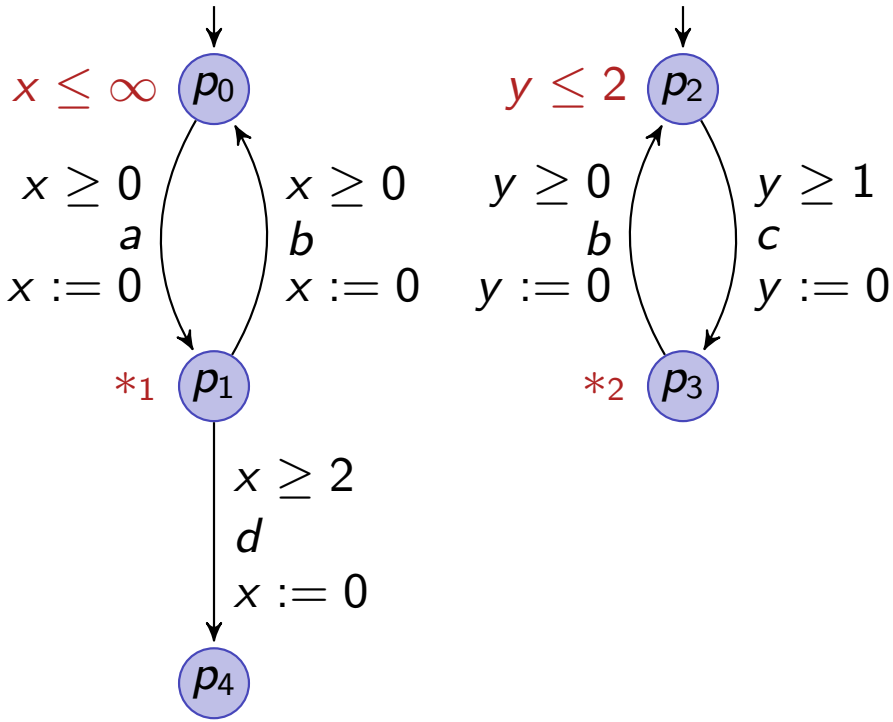




From safe TPN to NTA [BCH10]

Add invariants.

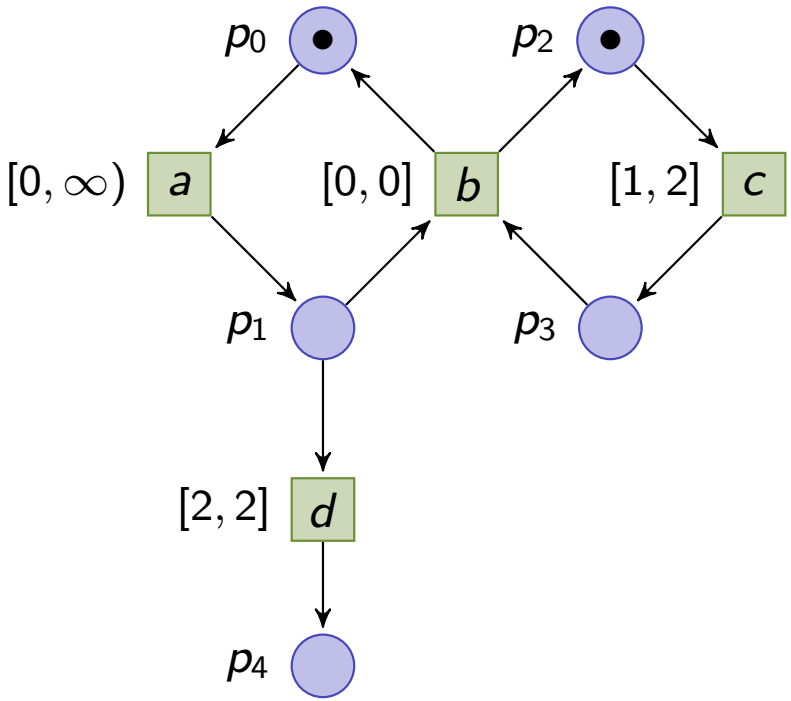


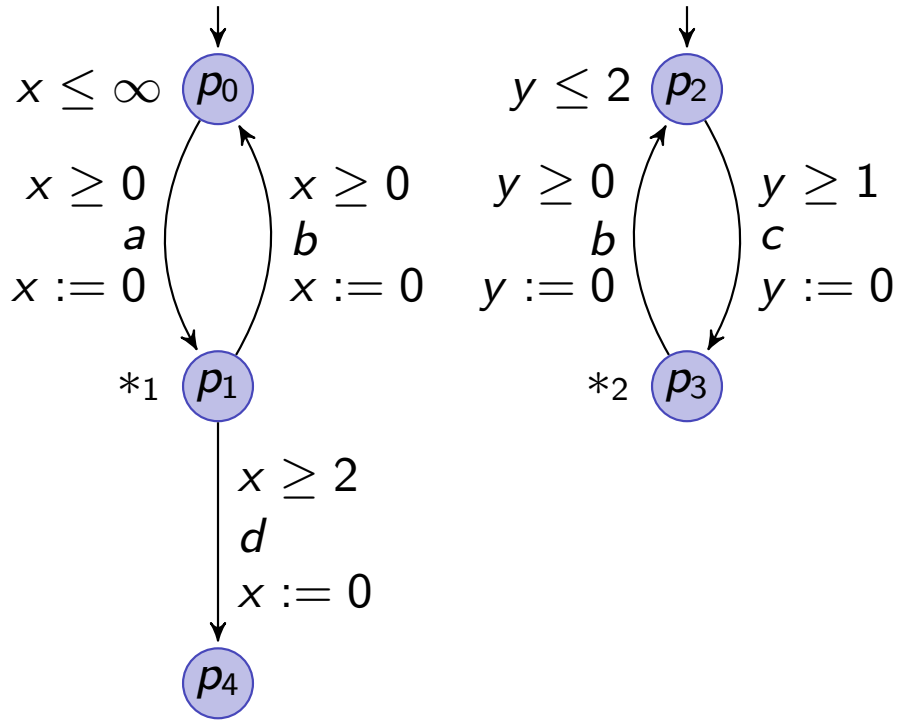


$$*_1 \equiv \overbrace{(p_1 \Rightarrow x \leq 2)}^{Inv(d)} \wedge \overbrace{((p_1 \wedge p_3) \Rightarrow (\min(x,y) \leq 0))}^{Inv(b)} \equiv (x \leq 2) \wedge (\neg p_3 \vee (x \leq 0 \vee y \leq 0))$$

$$*_2 \equiv (p_1 \wedge p_3) \Rightarrow (\min(x,y) \leq 0)) \equiv (\neg p_1 \vee (x \leq 0 \vee y \leq 0))$$

From safe TPN to NTA [BCH10]





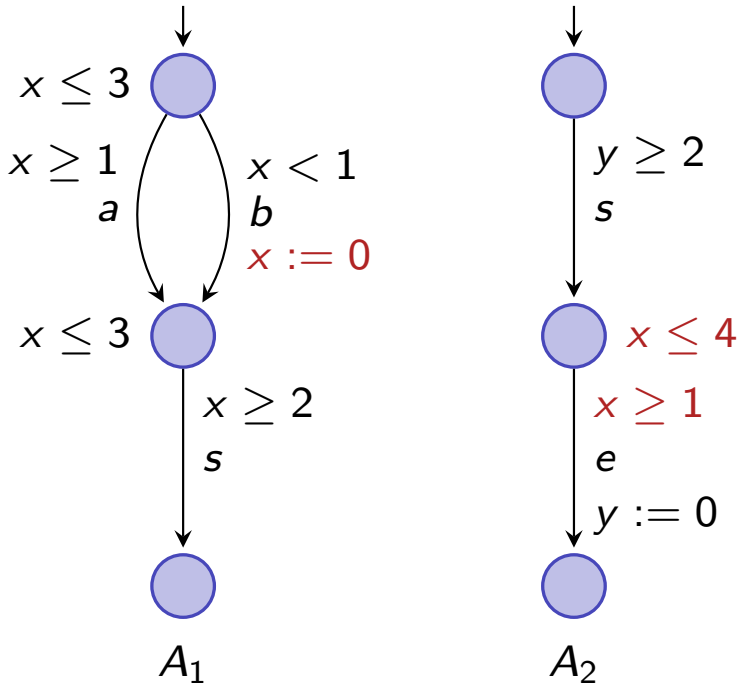
$$*_1 \equiv (x \leq 2) \wedge (\neg p_3 \vee (x \leq 0 \vee y \leq 0))$$
$$*_2 \equiv (\neg p_1 \vee (x \leq 0 \vee y \leq 0))$$

Shared clocks cannot be avoided in general.

Avoiding shared clocks [BC12]

Shared clocks in NTA

- Rather common feature
- Supported by tools (Uppaal...)
- No problem for sequential semantics



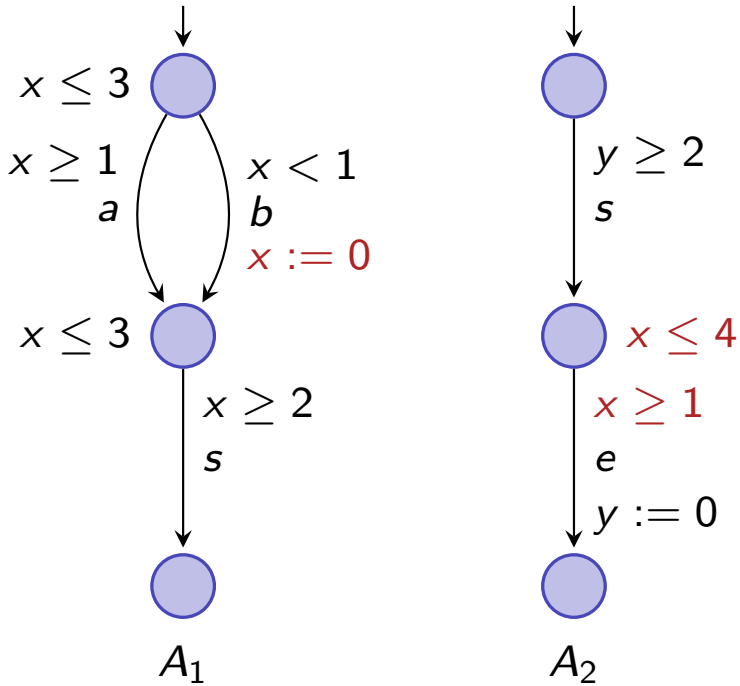
Avoiding shared clocks [BC12]

Shared clocks in NTA

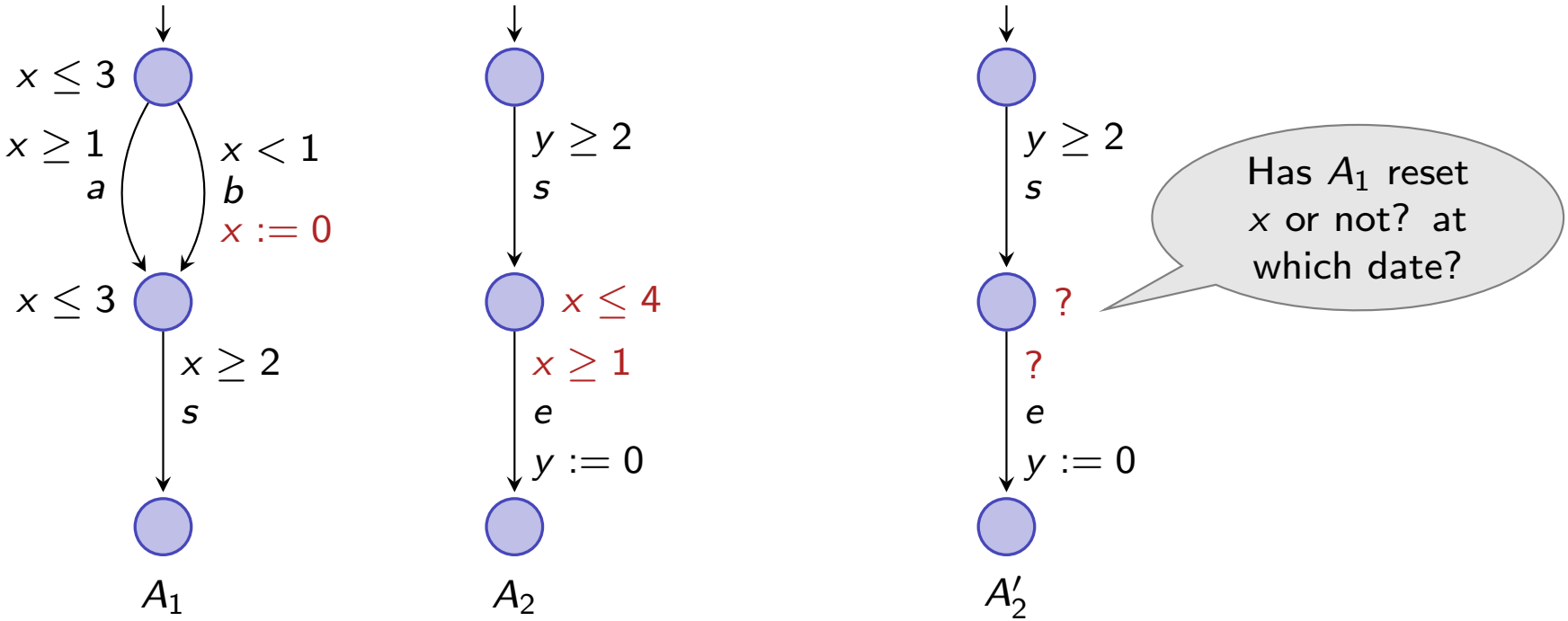
- Rather common feature
- Supported by tools (Uppaal...)
- No problem for sequential semantics

NTA are supposed to model distributed systems!

- Implementation of shared clocks on distributed architectures
- No other communication than those explicitly specified (here s)

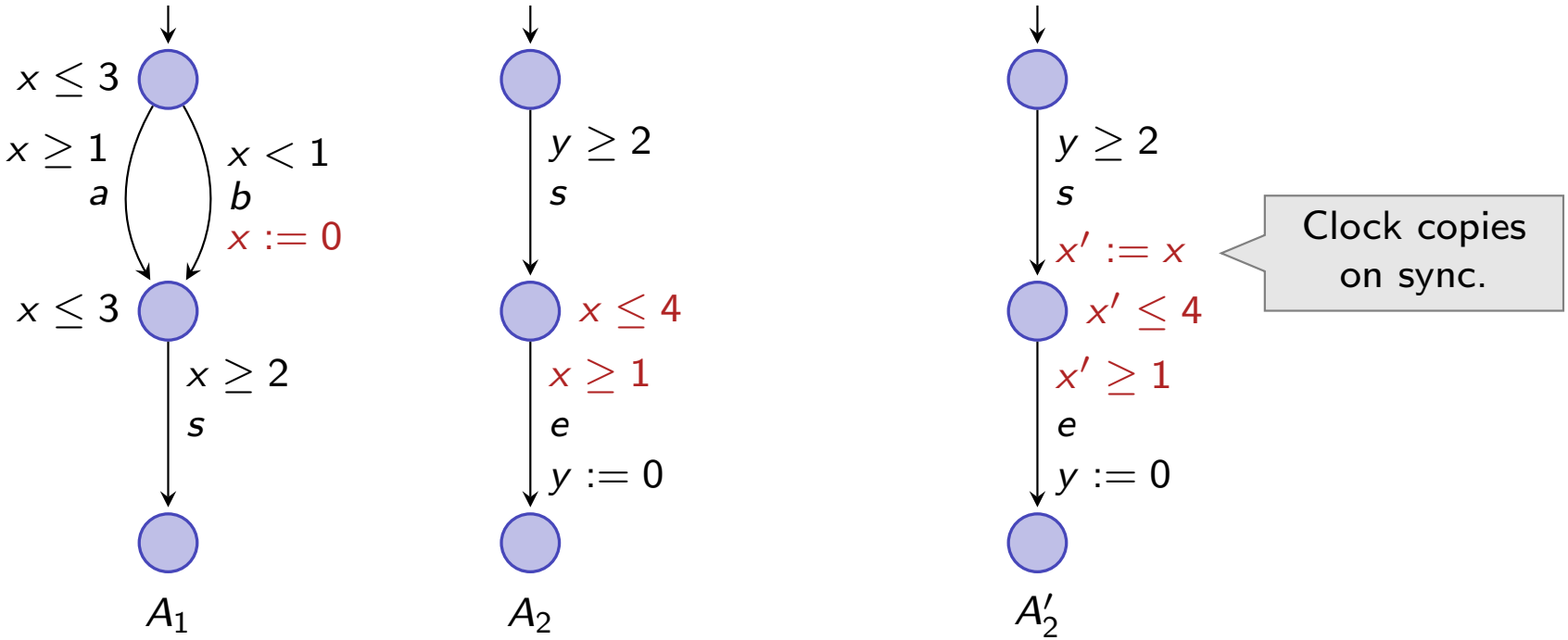


Transmitting information



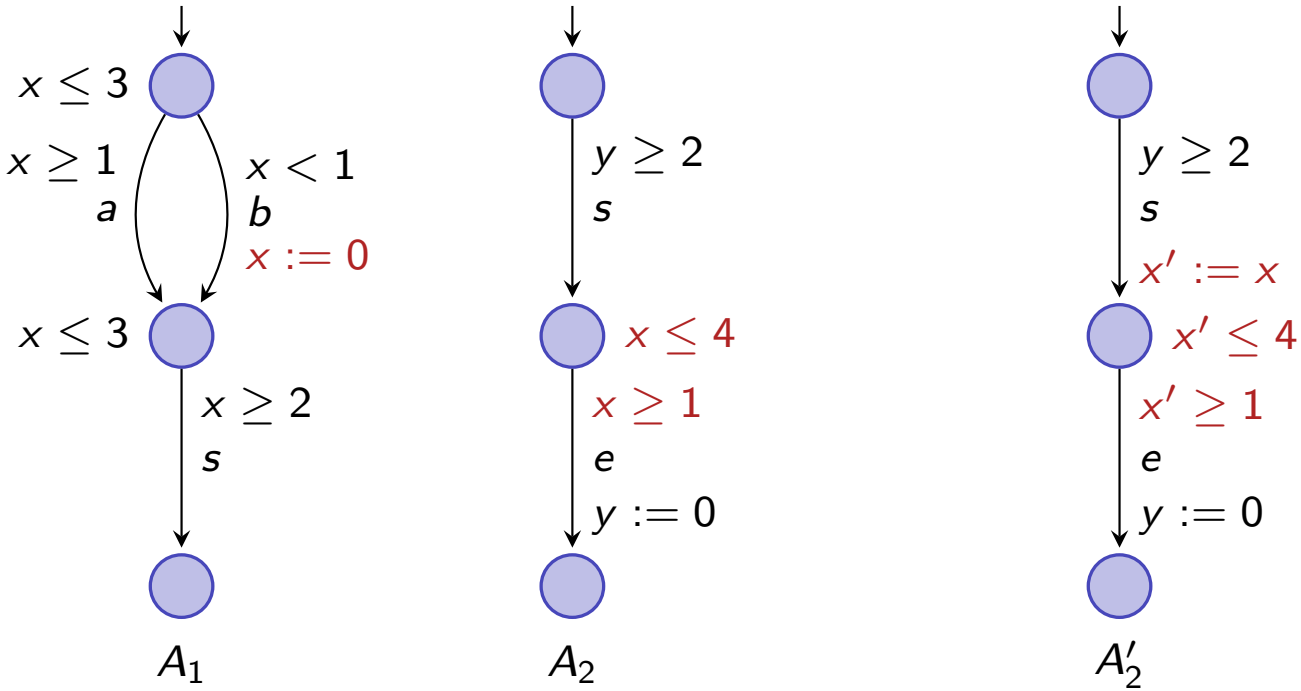
- A'_2 has to be able to infer the value of x from some information.

Transmitting information



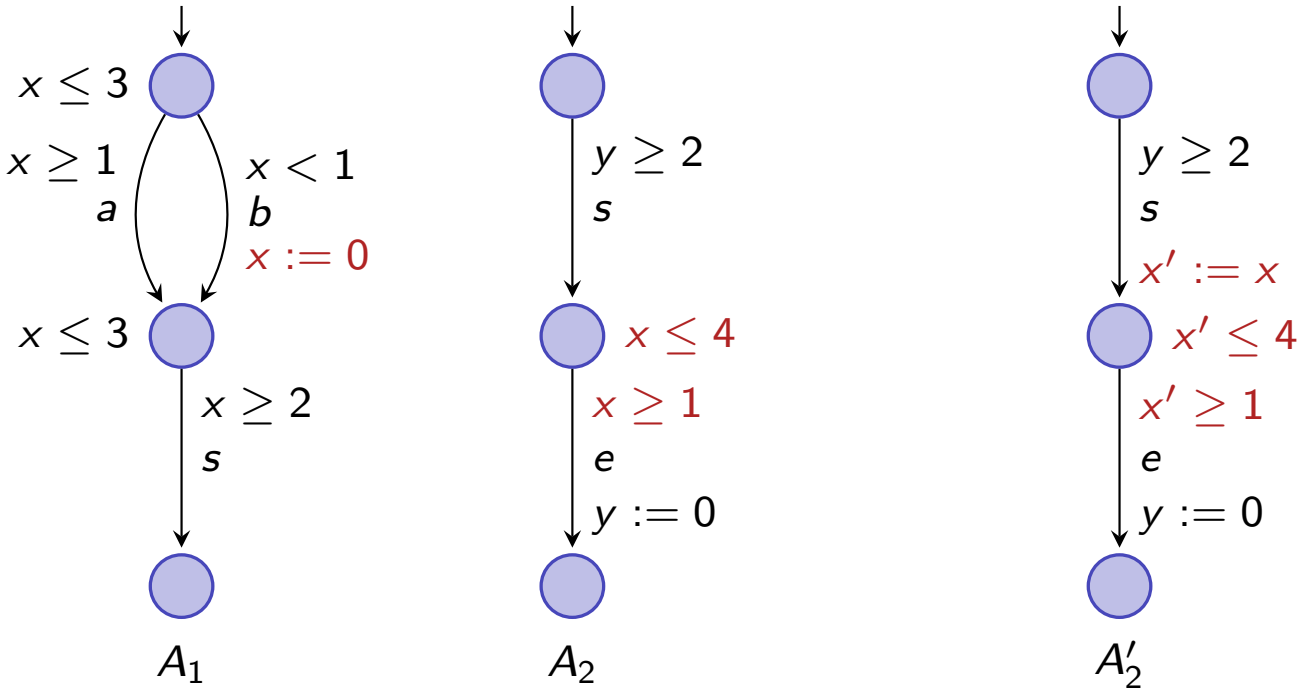
- A'_2 has to be able to infer the value of x from some information.
- For this, we allow transmission of information during synchronizations.

Towards a formalization



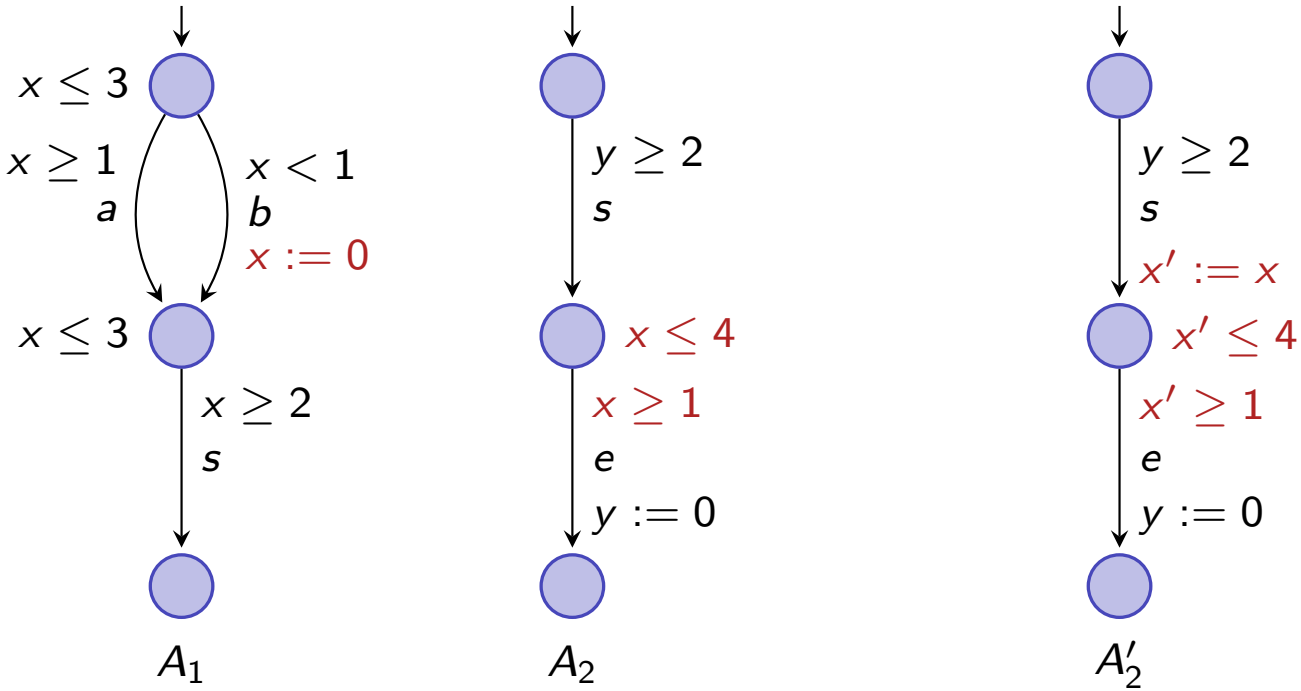
- A'_2 has the same alphabet of actions as A_2

Towards a formalization



- A'_2 has the same alphabet of actions as A_2
- $A_1 \parallel A_2 \sim A_1 \parallel A'_2$
  $\sim$: (weak) timed bisimulation

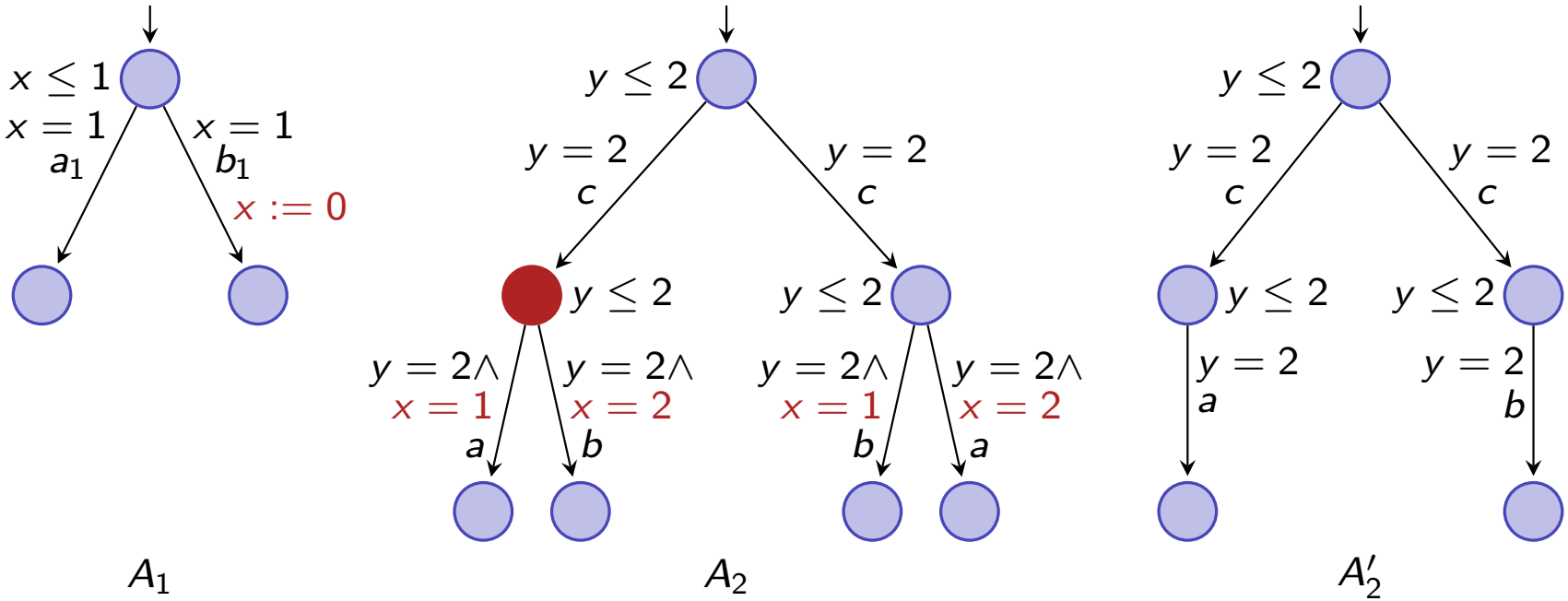
Towards a formalization



- A'_2 has the same alphabet of actions as A_2
- $A_1 \parallel A_2 \sim A_1 \parallel A'_2$
  $\sim$: (weak) timed bisimulation
- but not sufficient!

Global bisimulation $A_1 \parallel A_2 \sim A_1 \parallel A'_2$ does not take into account the partial knowledge of A_2

Towards a formalization



A_1 A_2 A_2'

- $A_1 \parallel A_2 \sim A_1 \parallel A_2'$ ✓
- and yet A_2 really needs to read x
  - in ● at time 2, A_2 performs a or b according to the value of x
  - whereas at time 2, the behavior of A_2' does not depend on A_1

Avoiding shared clocks: formalization

Definition

A_2 does not need to read the clocks of A_1 iff
there exists A_2' which does not read the clocks of A_1 (but is allowed to copy them during synchronizations) and such that

1. A_2' has the same alphabet of actions as A_2,
2. $A_1 \parallel A_2 \sim A_1 \parallel A_2'$, and
3. $TTS_{A_1}(A_2) \sim TTS_{A_1}(A_2')$

Contextual TTS

Contextual Bisimulation: "A_2 and A_2' have the same behavior in the context of A_1"

Contextual TTS

- Represent the behavior of A_2 in the context of A_1
- Knowledge of A_2 about the current state of A_1
- Similar powerset constructions in games, epistemic logics...

Contextual TTS

- Represent the behavior of A_2 in the context of A_1
- Knowledge of A_2 about the current state of A_1
- Similar powerset constructions in games, epistemic logics...

States: (S_1, s_2)

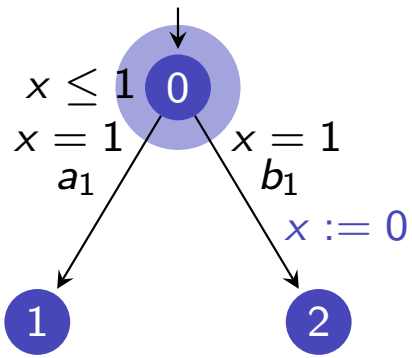
s_2: state of A_2

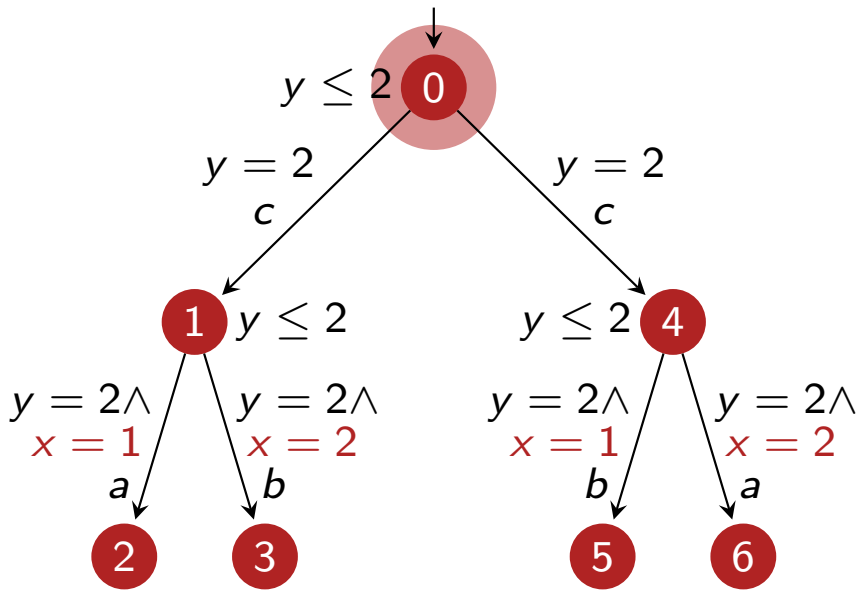
S_1: possible states of A_1

Labels: in $\Sigma_2 \cup \mathbb{R}_{\geq 0}$

Contextual TTS

Example path

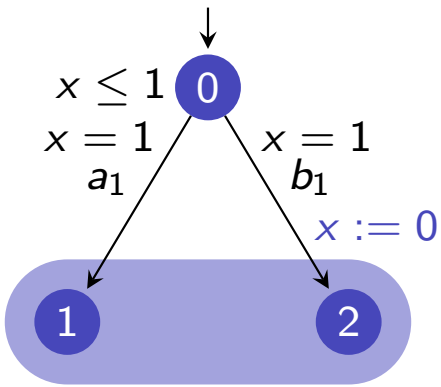


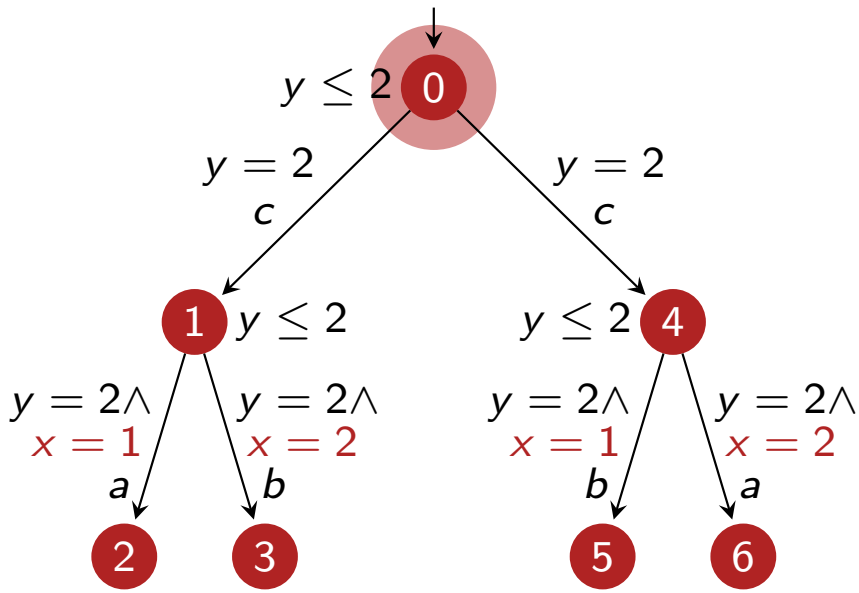


$(\{(0, 0)\}, (0, 0)) \xrightarrow{2}$

Contextual TTS

Example path

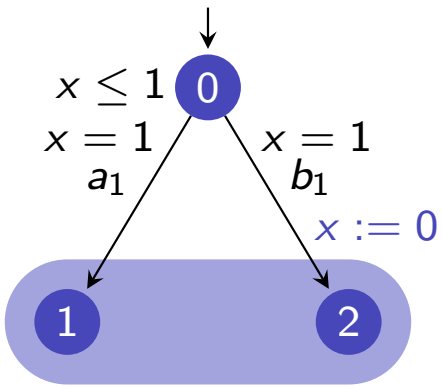


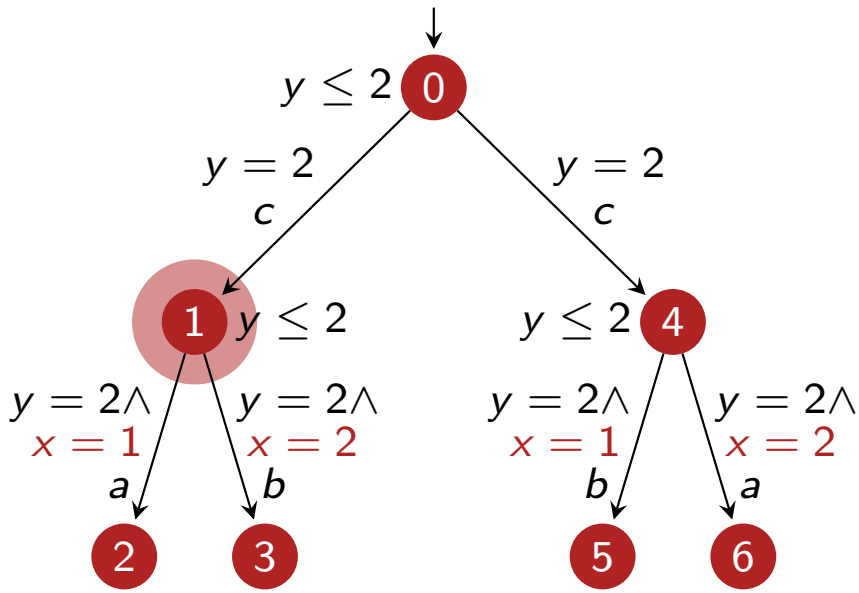


$$(\{(0,0)\},(0,0)) \xrightarrow{2} (\{(1,2),(2,1)\},(0,2)) \xrightarrow{c}$$

Contextual TTS

Example path

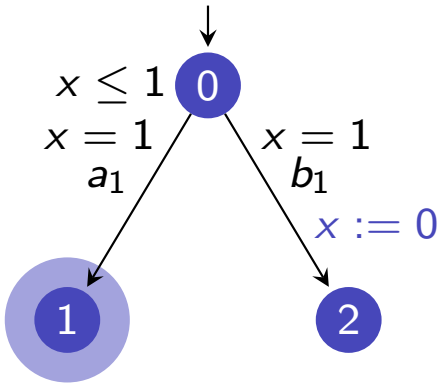


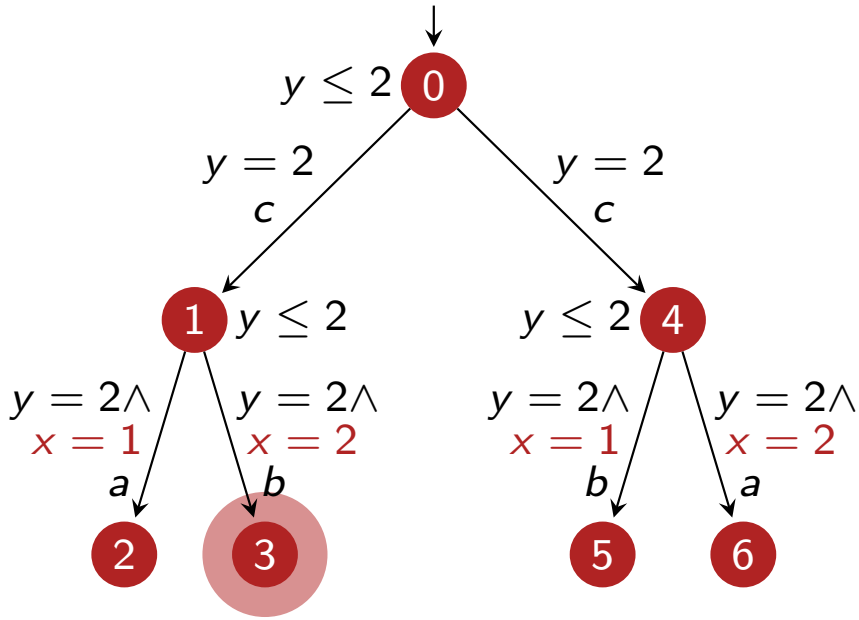


$$(\{(0,0)\},(0,0)) \xrightarrow{2} (\{(1,2),(2,1)\},(0,2)) \xrightarrow{c} (\{(1,2),(2,1)\},(1,2)) \xrightarrow{b}$$

Contextual TTS

Example path

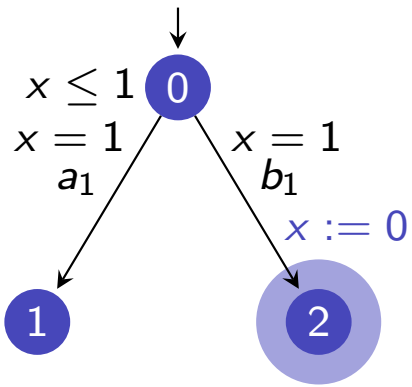


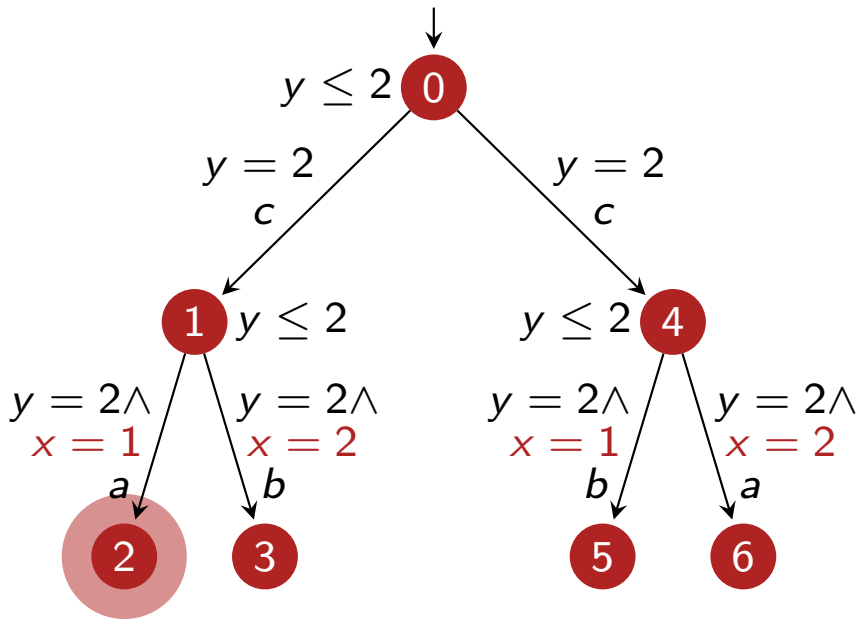


$$(\{(0,0)\},(0,0)) \xrightarrow{2} (\{(1,2),(2,1)\},(0,2)) \xrightarrow{c} (\{(1,2),(2,1)\},(1,2)) \xrightarrow{b} (\{(1,2)\},(3,2))$$

Contextual TTS

Example path





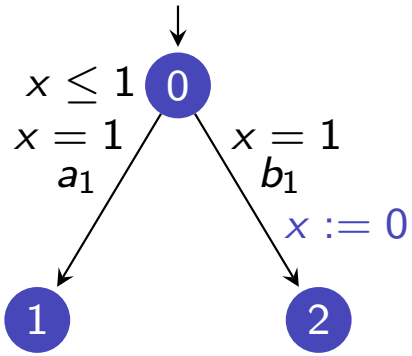
$$(\{(0,0)\},(0,0)) \xrightarrow{2} (\{(1,2),(2,1)\},(0,2)) \xrightarrow{c} (\{(1,2),(2,1)\},(1,2))$$

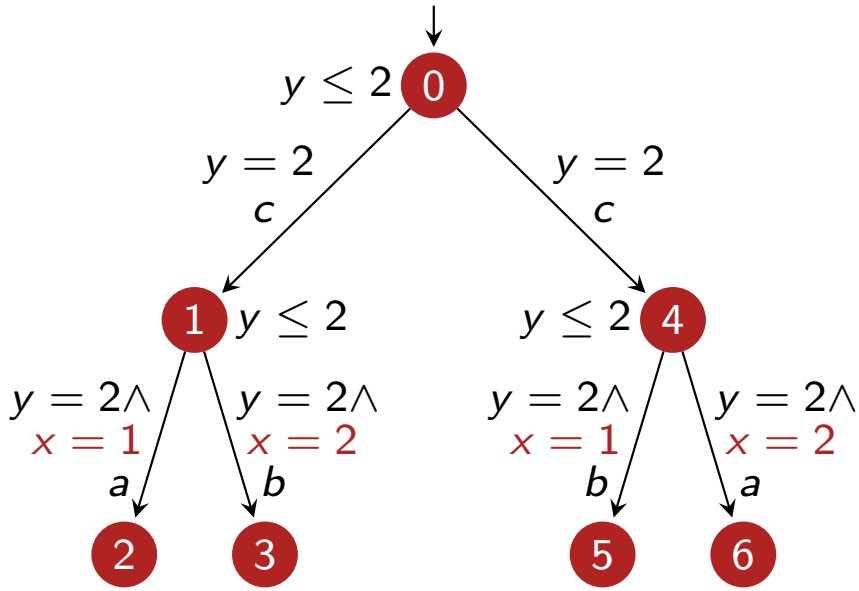
$$(\{(1,2),(2,1)\},(1,2)) \xrightarrow{a} (\{(2,1)\},(2,2))$$

$$(\{(1,2),(2,1)\},(1,2)) \xrightarrow{b} (\{(1,2)\},(3,2))$$

Contextual TTS

Example path





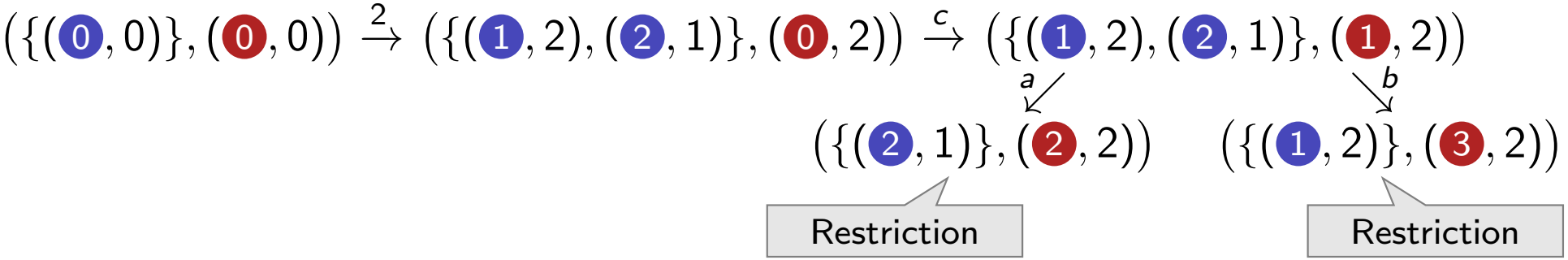
$$(\{(0,0)\},(0,0)) \xrightarrow{2} (\{(1,2),(2,1)\},(0,2)) \xrightarrow{c} (\{(1,2),(2,1)\},(1,2))$$

$$\xrightarrow{a} (\{(2,1)\},(2,2)) \qquad \xrightarrow{b} (\{(1,2)\},(3,2))$$

Restriction

Restriction

Unrestricted Contextual TTS



- If A_2 does not read the clocks reset by A_1, then $noRestriction_{A_1}(A_2)$.
- But $noRestriction_{A_1}(A_2)$ may hold even if A_2 reads these clocks.

Formalization and Theorem

Definition

A_2 does not need to read the clocks of A_1 iff
there exists A_2' which does not read the clocks of A_1 (but is allowed to copy them during synchronizations) and such that

1. A_2' has the same alphabet of actions as A_2,
2. $A_1 \parallel A_2 \sim A_1 \parallel A_2'$, and
3. $TTS_{A_1}(A_2) \sim TTS_{A_1}(A_2')$

Formalization and Theorem

Definition

A_2 does not need to read the clocks of A_1 iff
there exists A_2' which does not read the clocks of A_1 (but is allowed to copy them during synchronizations) and such that

1. A_2' has the same alphabet of actions as A_2,
2. $A_1 \parallel A_2 \sim A_1 \parallel A_2'$, and
3. $TTS_{A_1}(A_2) \sim TTS_{A_1}(A_2')$

Theorem

$noRestriction_{A_1}(A_2) \implies A_2$ *does not need to read the clocks of* A_1.
If A_2 *deterministic, equivalence*

Idea of the construction

A'_2 is equipped with its own copy of A_1, denoted $A_{1,copy}$.

At each synchronization

- $A_{1,copy}$ is updated to the actual state of A_1.

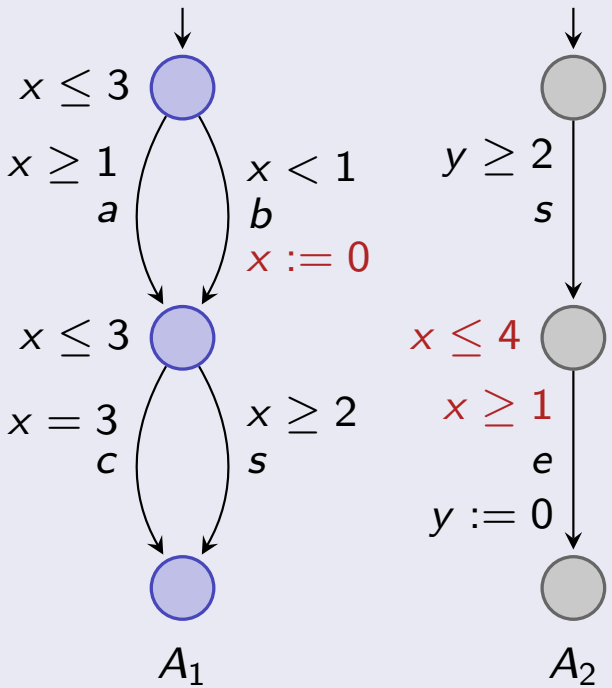
Between two synchronizations

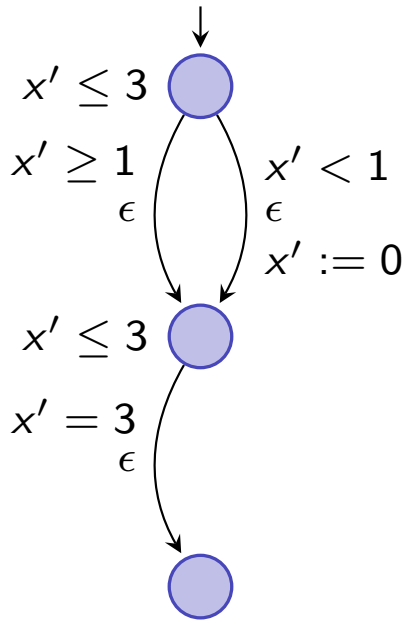
- $A_{1,copy}$ "simulates" a run of A_1.
- A'_2 reads the clocks of $A_{1,copy}$.
- The simulated run may differ from the actual run of A_1.

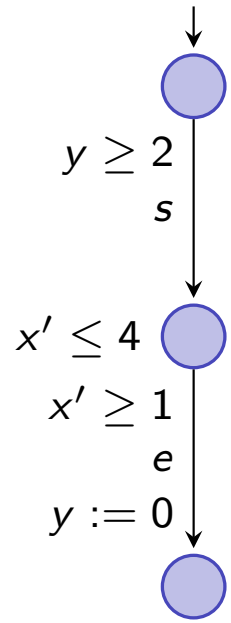
Error state

- If a contradiction between A_1 and $A_{1,copy}$ has an impact on the possible behavior, it goes to an error state ☹.

Construction





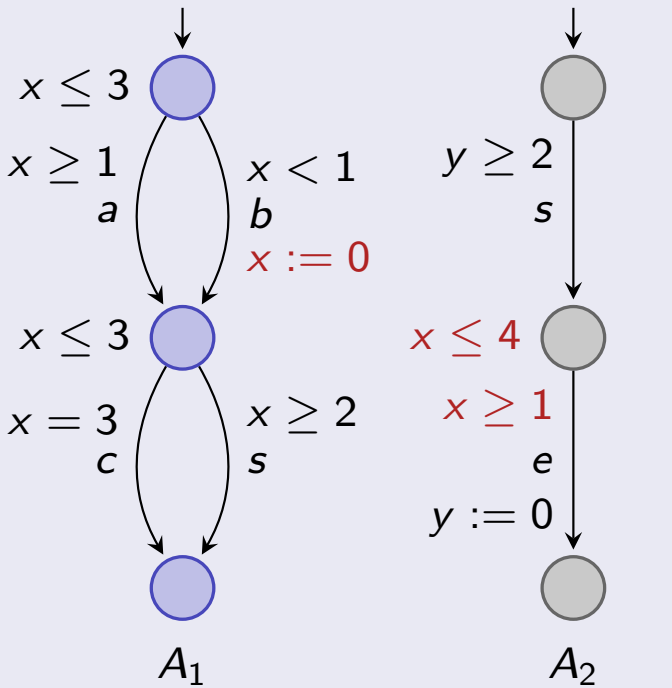


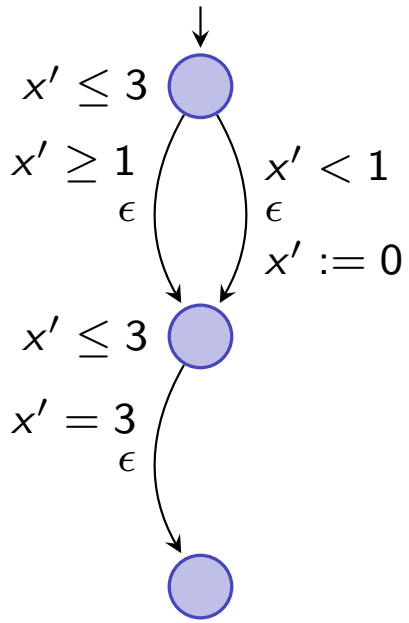
$$\underbrace{A_{1,copy} \quad \times \quad A_{2,\frowny}}_{A'_2}$$

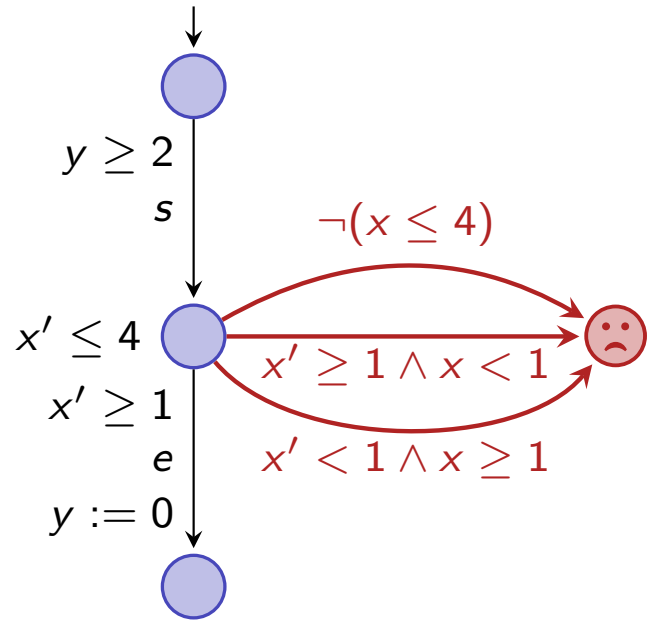
Between two synchronizations:

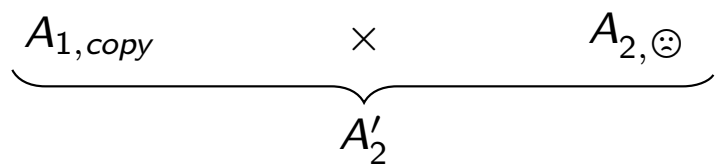
- $A_{1,copy}$ "simulates" a run of A_1.
- A_2 reads the clocks of $A_{1,copy}$.

Construction



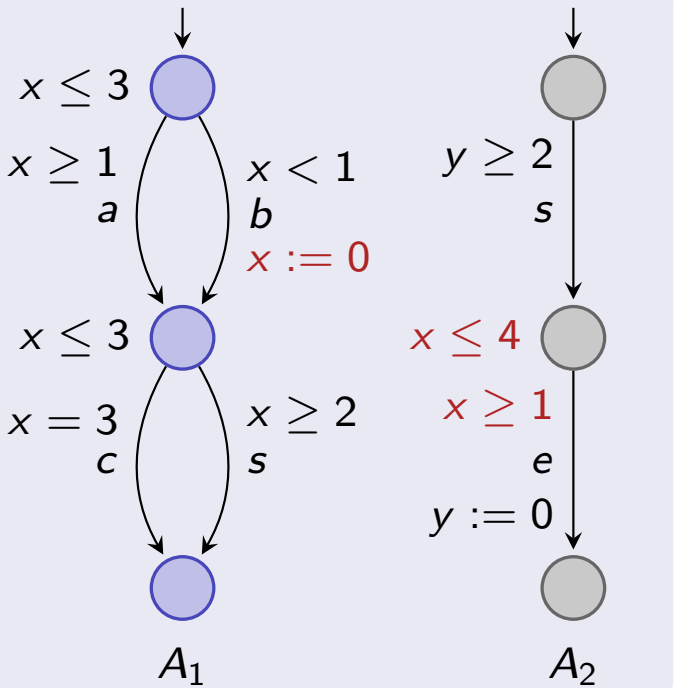


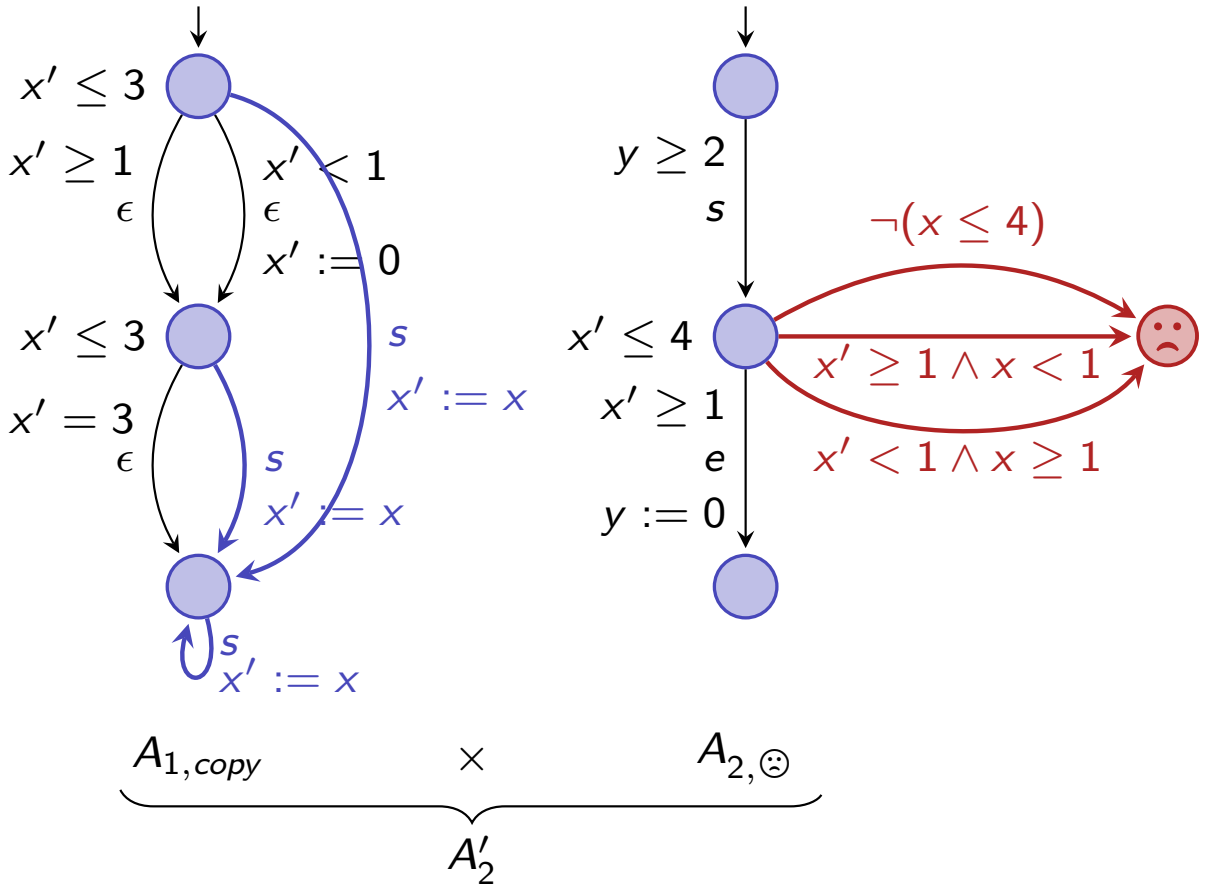




Error state: If A_2 observes a contradiction between the clocks of A_1 and $A_{1,copy}$, it goes to ☹.

Construction





Synchronizations: $A_{1,copy}$ is updated to the state of A_1 (clocks and location)

Avoiding Shared Clocks

Checking the Absence of Restrictions

$noRestriction_{A_1}(A_2) \iff$ ☹ is not reachable in $A_1 \parallel (A_{1,copy} \times A_{2,☹})$

Can be checked (PSPACE-complete)

Avoiding Shared Clocks

Checking the Absence of Restrictions

$noRestriction_{A_1}(A_2)$ $\iff$ ☹ is not reachable in $A_1 \parallel (A_{1,copy} \times A_{2,☹})$

$\implies$ shared clocks can be avoided

(equivalence when A_2 is deterministic)

Can be checked (PSPACE-complete)

Avoiding Shared Clocks

Checking the Absence of Restrictions

$noRestriction_{A_1}(A_2)$ $\iff$ ☹ is not reachable in $A_1 \parallel (A_{1,copy} \times A_{2,☹})$

$\implies$ shared clocks can be avoided

(equivalence when A_2 is deterministic)

Can be checked (PSPACE-complete)

Constructing $A'_1 \parallel A'_2$ without shared clocks when ☹ is not reachable

- A'_1 is A_1 with synchronizations relabeled to transmit the state
- $A'_2 = A_{1,copy} \times A_{2,☹} \setminus \{☹\}$

Avoiding Shared Clocks

Checking the Absence of Restrictions

$noRestriction_{A_1}(A_2)$ $\iff$ ☹ is not reachable in $A_1 \parallel (A_{1,copy} \times A_{2,☹})$

$\implies$ shared clocks can be avoided
(equivalence when A_2 is deterministic)

Can be checked (PSPACE-complete)

Constructing $A'_1 \parallel A'_2$ without shared clocks when ☹ is not reachable

- A'_1 is A_1 with synchronizations relabeled to transmit the state
- $A'_2 = A_{1,copy} \times A_{2,☹} \setminus \{☹\}$
- Suitable when no urgent synchronization in A_1,
  i.e. each time an invariant expires, a local action is enabled.

Constructing $A'_1 \parallel A'_2$ without Shared Clocks

General case: urgent synchronizations in A_1

The Difficulty

- $A_{1,copy}$ may reach a state where the invariant expires and only a synchronization is possible
- Then A'_2 is "expecting" a synchronization with A'_1,
- But the actual A'_1 may not be ready to synchronize.
- Intuitively, A'_2 should then realize that the simulated run cannot be the actual one and try another run compatible with the absence of synchronization.

Constructing $A'_1 \parallel A'_2$ without Shared Clocks

General case: urgent synchronizations in A_1

The Difficulty

- $A_{1,copy}$ may reach a state where the invariant expires and only a synchronization is possible
- Then A'_2 is "expecting" a synchronization with A'_1,
- But the actual A'_1 may not be ready to synchronize.
- Intuitively, A'_2 should then realize that the simulated run cannot be the actual one and try another run compatible with the absence of synchronization.

Solution: Avoid this Situation

- $A_{1,copy}$ is replaced by $A_{1,max}$ which "simulates" an run of A_1 of maximal duration until synchronization (computed from the region automaton).
- The construction can still be adapted when such run does not exist (strict time constraints).

Forcing $A^2_{1,\max}$ to "simulate" an execution of A_1 of maximal duration until synchronization

- $A^2_{1,\max}$ built over the region automaton of A_1
- Synchronizations removed (treated separately, like in $A_{1,copy}$)
- Keep the paths that lead to cycles (yield executions of infinite duration[1])
- For any region q from which all paths are finite
  - Compute the supremum of the duration for each path (finitely many)
  - Keep only paths with maximal supremum
  - Impose maximal duration using a fresh clock and appropriate guards and invariants

[1] with non-Zeno assumption

Conclusion

Transformations

- which preserve distribution
- from higher-level to lower-level models

Behavioral comparisons

- Limitations of global comparisons
- Importance of identifying the components
- Contextual bisimulation

Shared clocks

- Problem of deciding whether A_2 needs to read the clocks of A_1 solved
- Construction of $A'_1 \parallel A'_2$ without shared clocks when it exists
- Importance of transmitting information during synchronizations