Determinizing timed automata

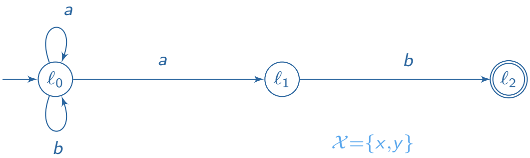
Nathalie Bertrand

INRIA Rennes, France

Séminaire CFV
20 mai 2011

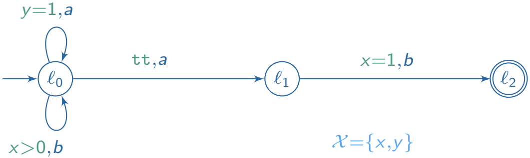
Timed automata on example

Timed automaton: Finite automaton enriched with clocks.



Timed automata on example

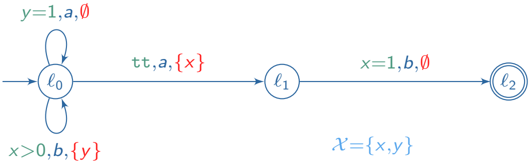
Timed automaton: Finite automaton enriched with clocks.



Transitions are equipped with guards

Timed automata on example

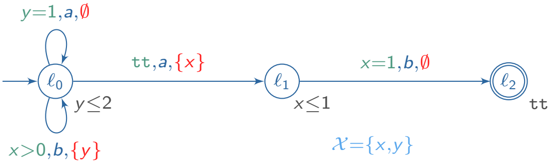
Timed automaton: Finite automaton enriched with clocks.



Transitions are equipped with guards and sets of reset clocks.

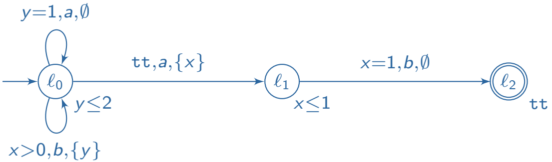
Timed automata on example

Timed automaton: Finite automaton enriched with clocks.



Transitions are equipped with guards and sets of reset clocks.
Locations are equipped with invariants.

Syntax



Timed automata

A timed automaton is a tuple $\mathcal{A} = (L, L_0, L_{acc}, \Sigma, X, E, Inv)$ with

- L finite set of locations
- $L_0 \subseteq L$ initial locations and $L_{acc} \subseteq L$ set of accepting locations
- Σ finite alphabet and X finite set of clocks
- $E \subseteq L \times \mathcal{G} \times \Sigma \times 2^X \times L$ set of edges is the set of guards. (with $\bowtie \in \{<, \leq, =, \geq, >\}$)
- $Inv : L \to \mathcal{G}$ invariant function

Semantics

Valuation: $v \in \mathbb{R}_+^X$ assigns to each clock a clock-value

State: $(\ell, v) \in L \times \mathbb{R}_+^X$ composed of a location and a valuation.

Transitions between states of $\mathcal{A}$:

- Delay transitions: $(\ell, v) \xrightarrow{\tau} (\ell, v + \tau)$
- Discrete transitions: $(\ell, v) \xrightarrow{a} (\ell', v')$

  if $\exists(\ell, g, a, Y, \ell') \in E$ with $v \models g$ and $\begin{cases} v'(x) = 0 & \text{if } x \in Y, \\ v'(x) = v(x) & \text{otherwise.} \end{cases}$

Run of $\mathcal{A}$:
$(\ell_0, v_0) \xrightarrow{\tau_1} (\ell_0, v_0 + \tau_1) \xrightarrow{a_1} (\ell_1, v_1) \xrightarrow{\tau_2} (\ell_1, v_1 + \tau_2) \xrightarrow{a_2} \cdots \xrightarrow{a_k} (\ell_k, v_k)$
or simply: $(\ell_0, v_0) \xrightarrow{\tau_1, a_1} (\ell_1, v_1) \xrightarrow{\tau_2, a_2} \cdots \xrightarrow{\tau_k, a_k} (\ell_k, v_k)$

Timed language

Time sequence: $\mathbf{t} = (t_i)_{1 \le i \le k}$ finite non-decreasing sequence over $\mathbb{R}_+$.

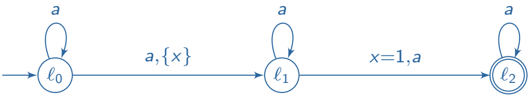
Timed word: $w = (\sigma, \mathbf{t}) = (a_i, t_i)_{1 \le i \le k}$ where $a_i \in \Sigma$ and $\mathbf{t}$ time sequence.

Accepted timed word

A timed word $w = (a_0, t_0)(a_1, t_1) \dots (a_k, t_k)$ is accepted in $\mathcal{A}$, if there is a run $\rho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \xrightarrow{\tau_1, a_1} \dots (\ell_{k+1}, v_{k+1})$ with $\ell_0 \in L_0$, $\ell_{k+1} \in L_{acc}$, and $t_i = \sum_{j<i} \tau_j$.

Accepted timed language: $\mathcal{L}(\mathcal{A}) = \{w \mid w \text{ accepted by } \mathcal{A}\}$.

An example



$w = (a, 0.1)(a, 0.3)(a, 1.1)(a, 1.2)(a, 1.3)$ is an accepted timed word

Example of an accepting run for w:

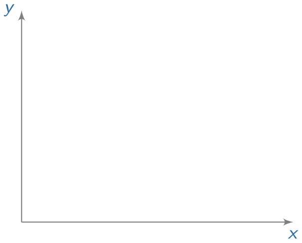
$(\ell_0, 0) \xrightarrow{0.1,a} (\ell_1, 0) \xrightarrow{0.2,a} (\ell_1, 0.2) \xrightarrow{0.8,a} (\ell_2, 0) \xrightarrow{0.1,a} (\ell_2, 0.1) \xrightarrow{0.1,a} (\ell_2, 0.3)$

$$\mathcal{L}(\mathcal{A}) = \{(a, t_1) \cdots (a, t_k) \mid \exists i < j,\ t_j - t_i = 1\}$$

Regions

Problem: Timed automata have infinite state space.

Regions form a finite partition of the set of valuations

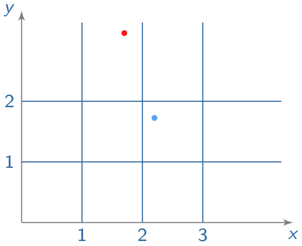


Regions

Problem: Timed automata have infinite state space.

Regions form a finite partition of the set of valuations, compatible with

- constraints on clock values (guards and invariants)

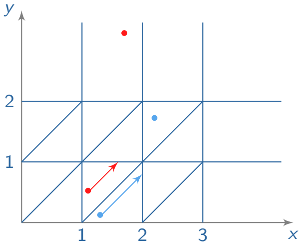


Regions

Problem: Timed automata have infinite state space.

Regions form a finite partition of the set of valuations, compatible with

- constraints on clock values (guards and invariants)
- time-elapsing

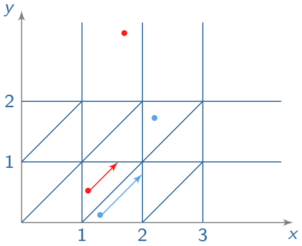


Regions

Problem: Timed automata have infinite state space.

Regions form a finite partition of the set of valuations, compatible with

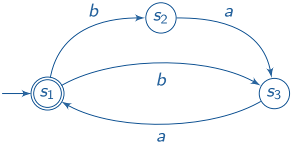
- constraints on clock values (guards and invariants)
- time-elapsing



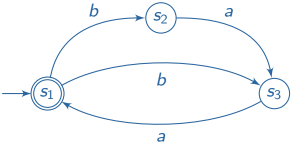
Emptiness problem

Emptiness is decidable for timed automata and is PSPACE-complete.

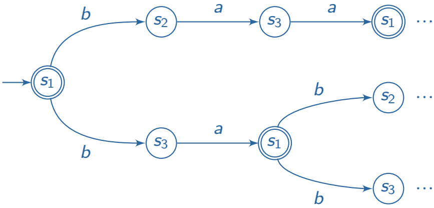
Determinizing finite automata: Subset construction



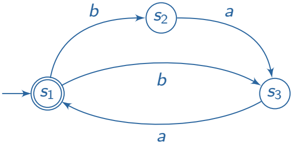
Determinizing finite automata: Subset construction



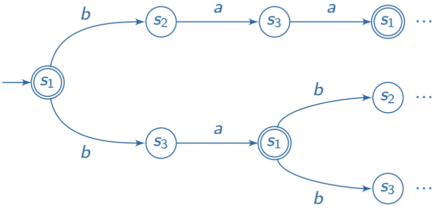
Unfolding the automaton



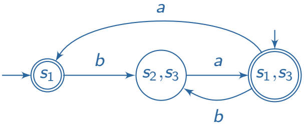
Determinizing finite automata: Subset construction



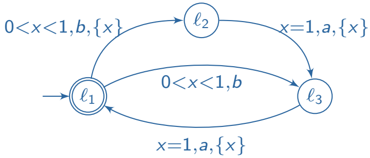
Unfolding the automaton



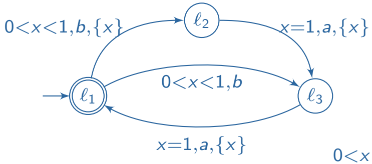
Deterministic equivalent



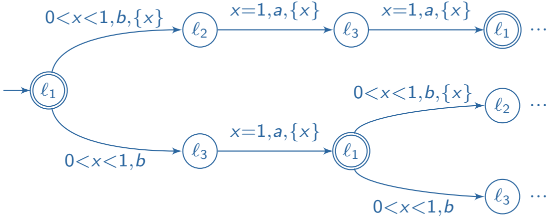
Naive adaptation to timed automata



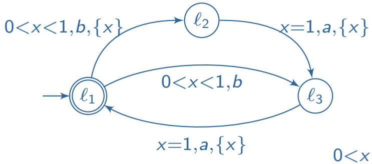
Naive adaptation to timed automata



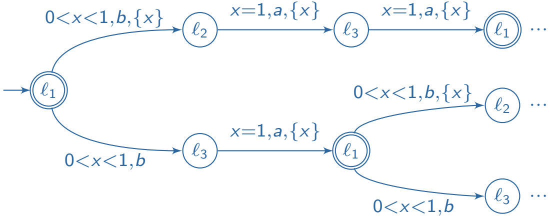
Unfolding the automaton



Naive adaptation to timed automata



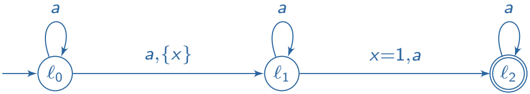
Unfolding the automaton



Subset construction fails because of non-uniform clock resets.

Determinizability of timed automata

Some timed automata are not determinizable [AD90].



$$\mathcal{L}(\mathcal{A}) = \{(a, t_1)\ldots(a, t_n) \mid n \geq 2 \text{ and } \exists i < j \text{ s.t. } t_j - t_i = 1\}$$

Theorem [Finkel 06]

Checking whether a given timed automata is determinizable is undecidable, even under fixed resources.

Workarounds

Two approaches to overcome unfeasible determinization:

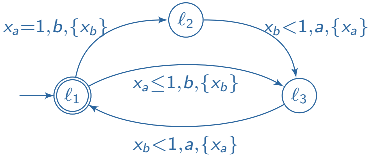
- exhibit determinizable subclasses
- perform an approximate determinization

1. Determinizable subclasses
2. Determinization procedure
3. Overapproximate determinization
4. A game approach

1. Determinizable subclasses
   - Event-recording automata
   - Integer-reset timed automata
2. Determinization procedure
3. Overapproximate determinization
4. A game approach

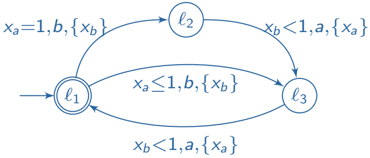
Event-recording automata [AFH94]

Finite automata with one clock associated with each action $a \in \Sigma$ which is reset exactly when action a occurs.



Event-recording automata [AFH94]

Finite automata with one clock associated with each action $a \in \Sigma$ which is reset exactly when action a occurs.



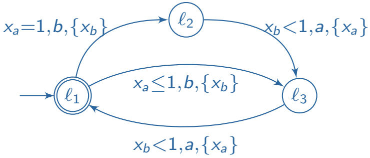
Input-determinacy property

Valuation only depends on input word, not on the precise execution.

Event-recording automata

[AFH94]

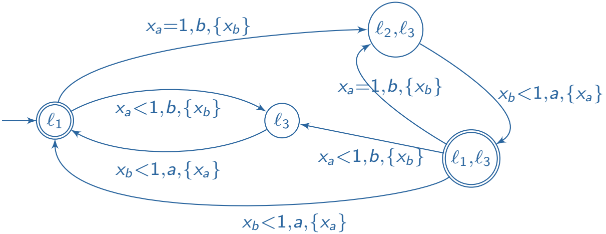
Finite automata with one clock associated with each action $a \in \Sigma$ which is reset exactly when action a occurs.



Input-determinacy property

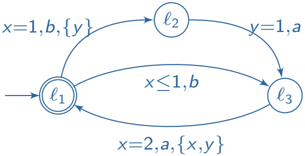
Valuation only depends on input word, not on the precise execution.

Determinization via subset construction



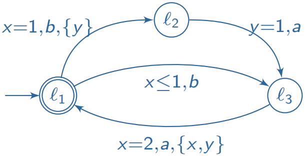
Integer-reset timed automata [SPKM08]

Resets only allowed when some clock value equals a constant.



Integer-reset timed automata [SPKM08]

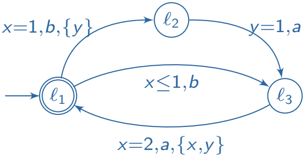
Resets only allowed when some clock value equals a constant.



Tick property All clocks share the same fractional part.

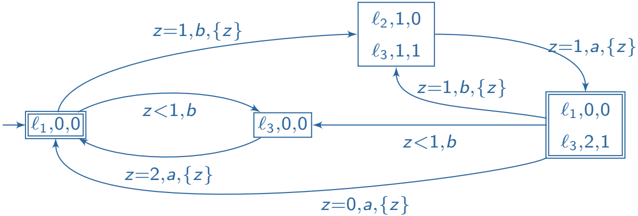
Integer-reset timed automata [SPKM08]

Resets only allowed when some clock value equals a constant.



Tick property All clocks share the same fractional part.

Determinization with a single clock!



1. Determinizable subclasses
2. Determinization procedure
   - Overview
   - Details
3. Overapproximate determinization
4. A game approach

Determinization procedure

joint work w. Baier, Bouyer, Brihaye

Approach overview

- unfolding of the automaton, introducing a fresh clock at each step, into a timed tree with infinitely many clocks and nodes
- symbolic determinization
- reduction of the number of clocks (under some assumption) and folding back into an automaton
- effective algorithm with fixed upper bound on resources.

Determinization procedure

joint work w. Baier, Bouyer, Brihaye

Approach overview

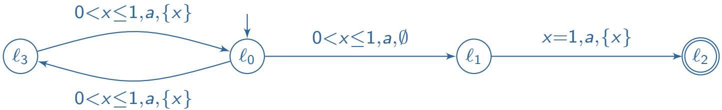
- unfolding of the automaton, introducing a fresh clock at each step, into a timed tree with infinitely many clocks and nodes
- symbolic determinization
- reduction of the number of clocks (under some assumption) and folding back into an automaton
- effective algorithm with fixed upper bound on resources.

Essential features

- in each location of the new automaton, original clocks are mapped to new clocks
- termination of the procedure is not guaranteed
- exact determinization

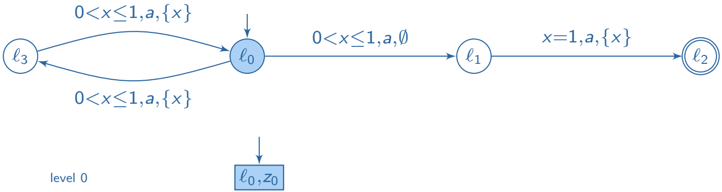
Unfolding

- Unfolding into an infinite timed tree with a fresh clock at each step.
- Original clocks are mapped to their reference in the new set of clocks.



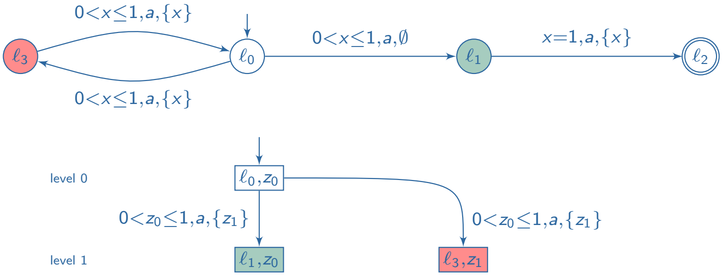
Unfolding

- Unfolding into an infinite timed tree with a fresh clock at each step.
- Original clocks are mapped to their reference in the new set of clocks.



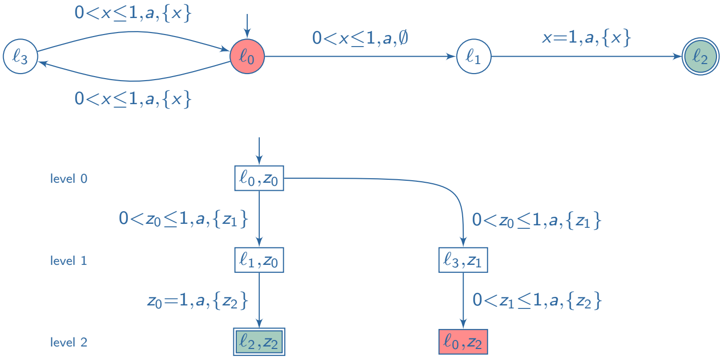
Unfolding

- Unfolding into an infinite timed tree with a fresh clock at each step.
- Original clocks are mapped to their reference in the new set of clocks.

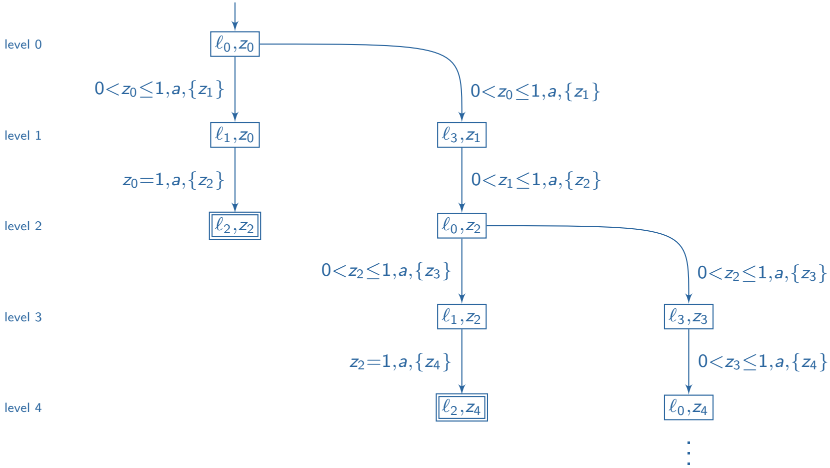


Unfolding

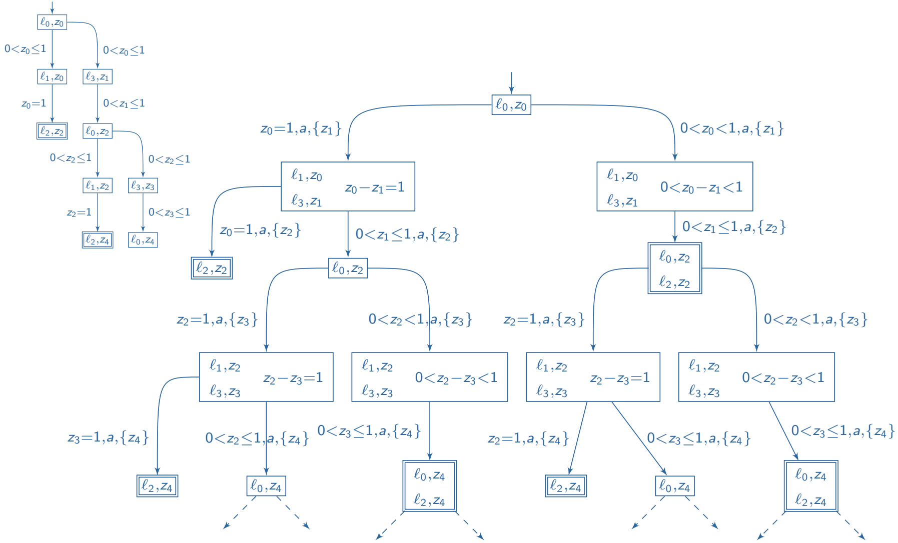
- Unfolding into an infinite timed tree with a fresh clock at each step.
- Original clocks are mapped to their reference in the new set of clocks.



Unfolding

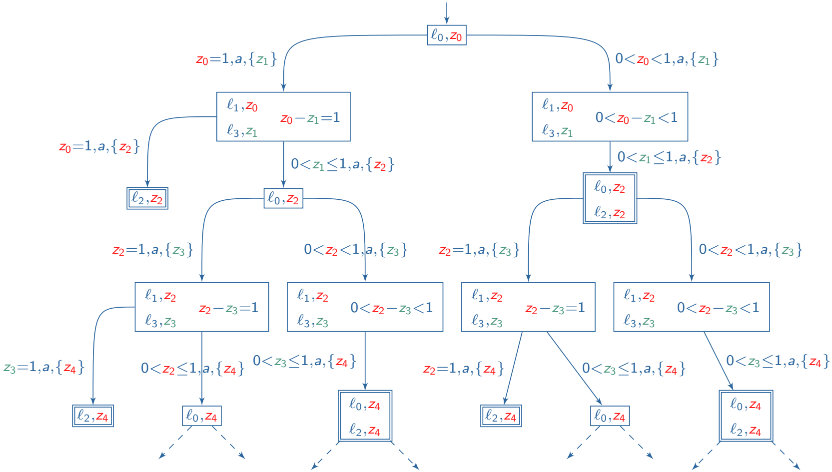

level 0
level 1
level 2
level 3
level 4
ℓ_0, z_0
$0 < z_0 \leq 1, a, \{z_1\}$
$0 < z_0 \leq 1, a, \{z_1\}$
ℓ_1, z_0
ℓ_3, z_1
$z_0 = 1, a, \{z_2\}$
$0 < z_1 \leq 1, a, \{z_2\}$
ℓ_2, z_2
ℓ_0, z_2
$0 < z_2 \leq 1, a, \{z_3\}$
$0 < z_2 \leq 1, a, \{z_3\}$
ℓ_1, z_2
ℓ_3, z_3
$z_2 = 1, a, \{z_4\}$
$0 < z_3 \leq 1, a, \{z_4\}$
ℓ_2, z_4
ℓ_0, z_4

Symbolic determinization



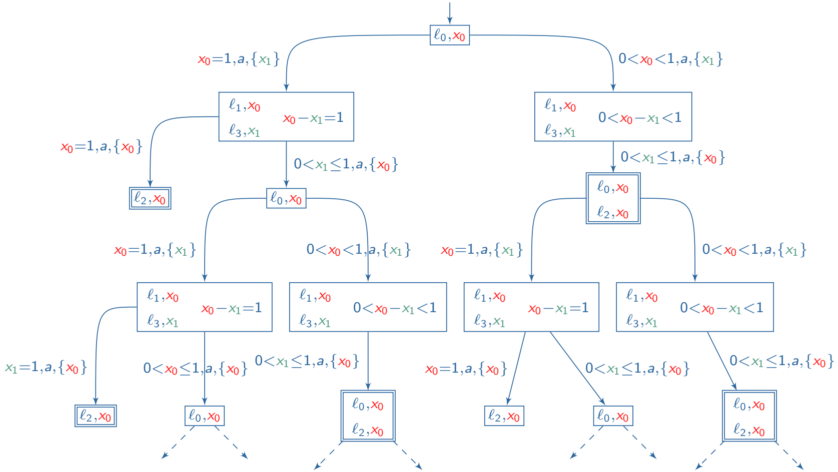
Clock reduction

Two clocks are sufficient to encode all timing information!



Clock reduction

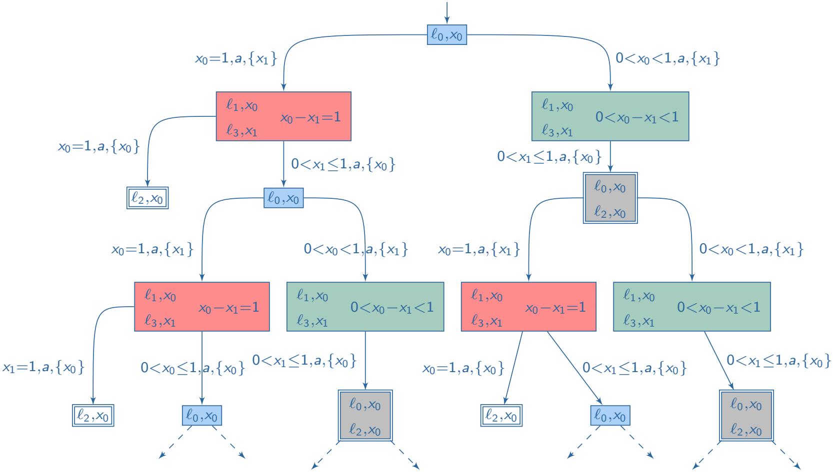
Two clocks are sufficient to encode all timing information!



Renaming: $z_{2n} \leftarrow x_0$ and $z_{2n+1} \leftarrow x_1$

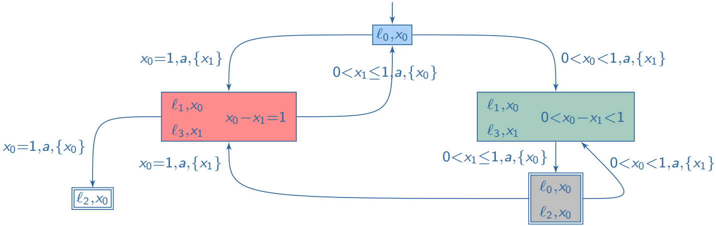
Location reduction

Finitely many node labels.



Location reduction

Finitely many node labels.



Merging of isomorphic nodes

Overapproximate determinization [KT09]

Approach overview

- observation of the behaviour using a new clock, reset at each step
- over-approximation of the guards according to the new clock
- estimation of the possible current states

- can be extended to several observation clocks (resets fixed by DFA)

Overapproximate determinization [KT09]

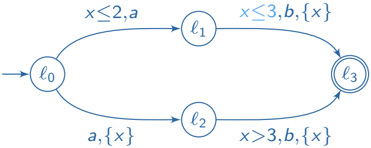
Approach overview

- observation of the behaviour using a new clock, reset at each step
- over-approximation of the guards according to the new clock
- estimation of the possible current states
- can be extended to several observation clocks (resets fixed by DFA)

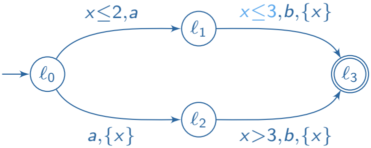
Essential features

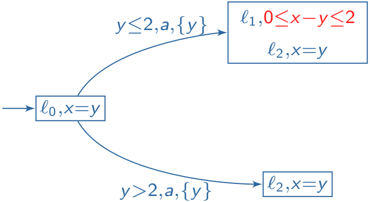
- fixed resources (number of clocks and maximal constant)
- flexible relations between old and new clocks
- no assumptions for termination
- deterministic over-approximation

Overapproximation on an example

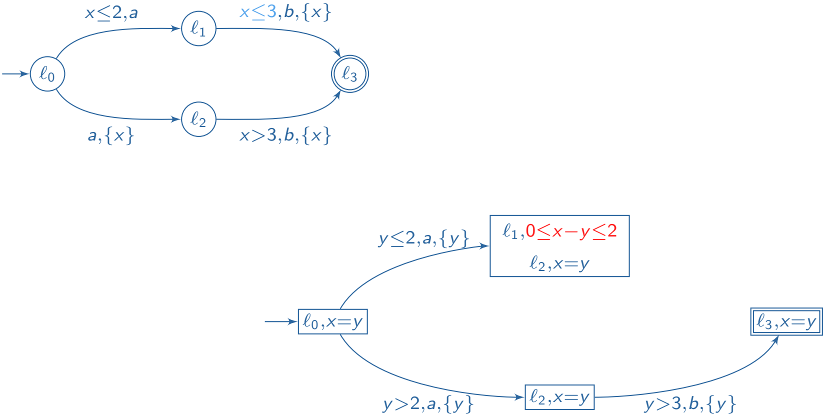


Overapproximation on an example

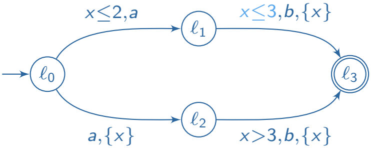


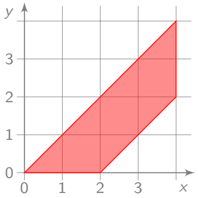


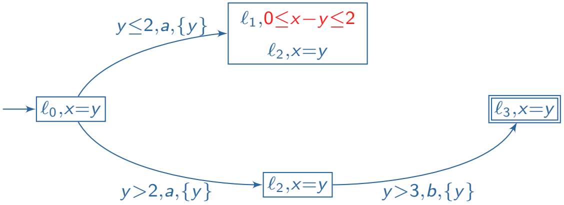
Overapproximation on an example



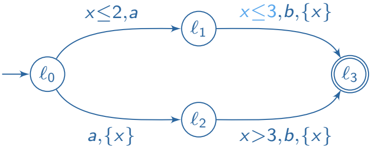
Overapproximation on an example

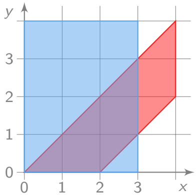


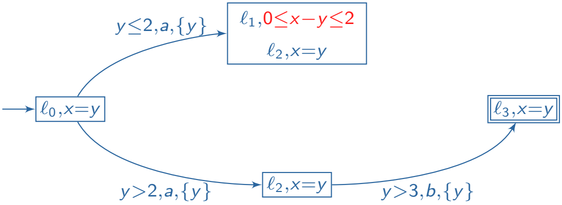




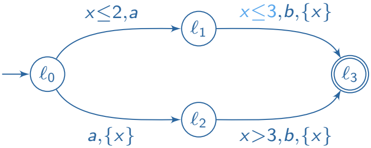
Overapproximation on an example

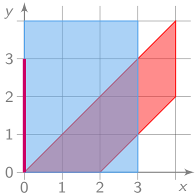


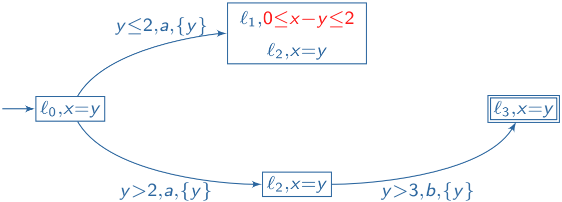




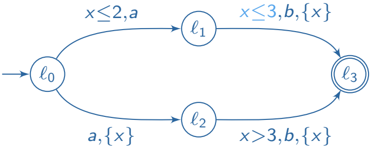
Overapproximation on an example

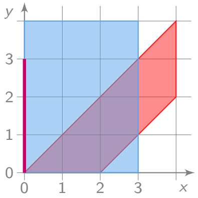


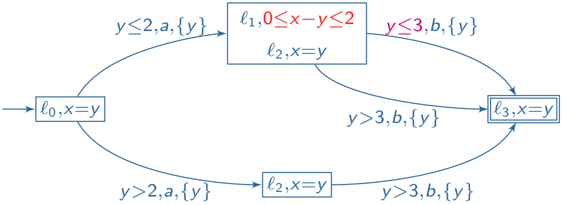




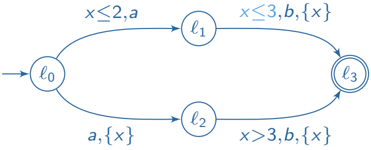
Overapproximation on an example

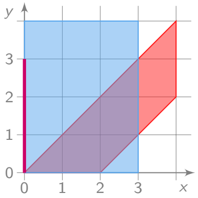


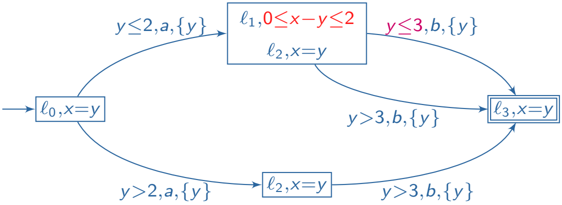




Overapproximation on an example







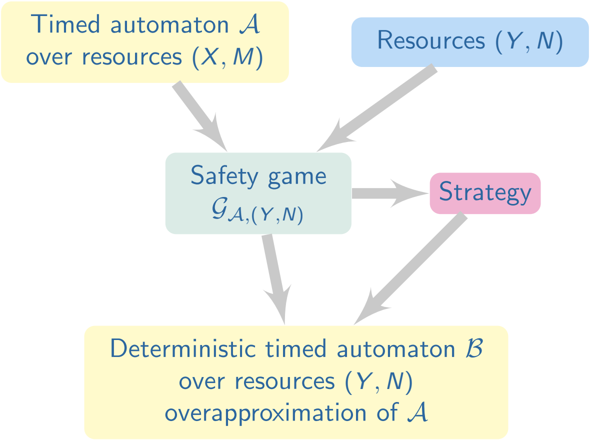
Timed word $(a, 0.6)(b, 3.2)$ is accepted in the deterministic over-approximation but not in the original timed automaton.

A game approach

joint work w. Stainer, Jéron, Krichen

- Goal: extend existing approaches
  - fixed resources (number of clocks and maximal constant)
  - determinization or deterministic over-approximation
- Method
  - inspired by [BCD05] for diagnosis of timed automata
  - turn-based game to choose when to reset the new clocks
  - coding of the relations between old and new clocks similar to [KT09]

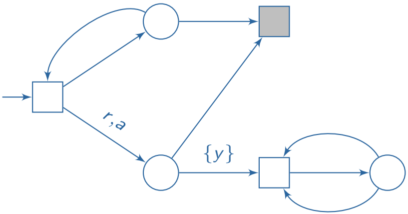
Overview of the approach



Closer look to the game

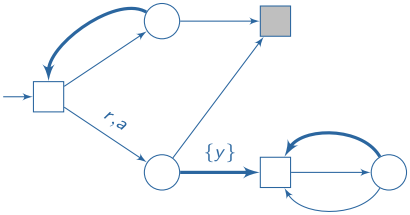
Finite turn-based safety game between Spoiler and Determinizator.

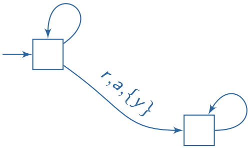
- First, Spoiler chooses an action and when to fire it (region over the new clocks)
- Then, Determinizator chooses which (new) clocks to reset
- States are unsafe when an over-approximation possibly happened
- Determinizator wants to avoid unsafe states



Properties of the game

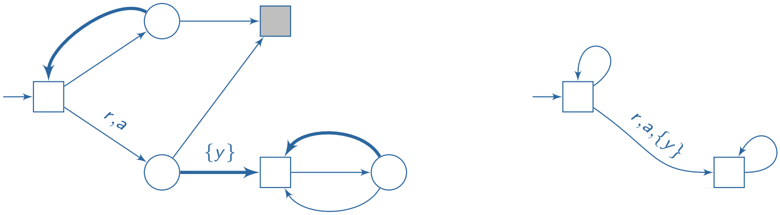
Any strategy for Determinizator yields a timed automaton by merging each move of Spoiler with the next move of Determinizator.





Properties of the game

Any strategy for Determinizator yields a timed automaton by merging each move of Spoiler with the next move of Determinizator.



Properties

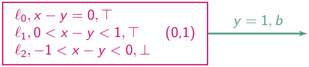
- Every strategy for Determinizator yields a deterministic over-approximation.
- Every **winning** strategy for Determinizator yields a deterministic equivalent.

States and moves: Spoiler

States of Spoiler (□-states):

- a set of configurations each with a marker
  - configuration: location + relation between old and new clocks
    relation: conjunction of $x - y \equiv c$
  - marker: $\top$ or $\perp$ to indicate possible over-approximations
- a region (over the new set of clocks)

Moves: Spoiler chooses a successor region and an action.



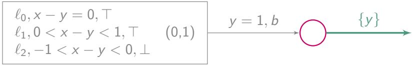
Unsafe states: □-states of the form $(\{\ell_i, C_i, \perp\}_{i \in I}, r)$

States and moves: Determinizator

States of Determinizator (◯-states):

- a state of Spoiler + a region over new clocks + an action

Moves: Determinizator chooses a reset set.



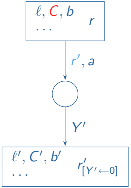
States' update

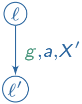
Given a ◯-state and a reset set, how to compute the next □-state?

States' update

Given a ◯-state and a reset set, how to compute the next ☐-state?

- For each configuration ℓ, C, b, given moves (r', a) of ☐ and Y' of ◯

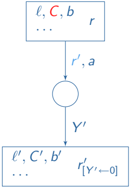


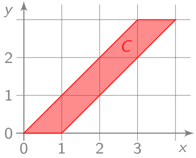


States' update

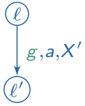
Given a ◯-state and a reset set, how to compute the next □-state?

- For each configuration ℓ, C, b, given moves (r', a) of □ and Y' of ◯





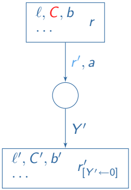
C: $0 < x - y < 1$

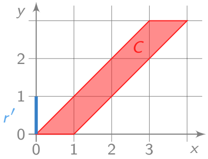


States' update

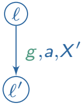
Given a ◯-state and a reset set, how to compute the next □-state?

- For each configuration ℓ, C, b, given moves (r', a) of □ and Y' of ◯





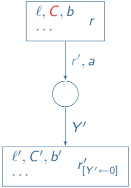
C: $0<x-y<1$ r': $0<y<1$

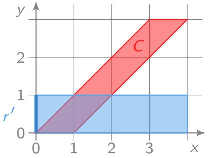


States' update

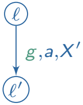
Given a ◯-state and a reset set, how to compute the next □-state?

- For each configuration ℓ, C, b, given moves (r', a) of □ and Y' of ◯





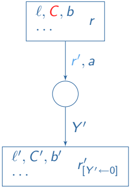
C: $0<x-y<1$ r': $0<y<1$

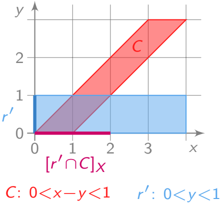


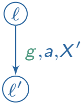
States' update

Given a ◯-state and a reset set, how to compute the next ☐-state?

- For each configuration ℓ, C, b, given moves (r', a) of ☐ and Y' of ◯



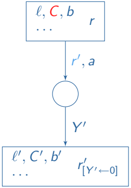


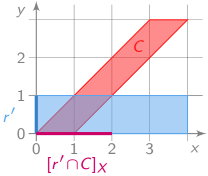


States' update

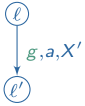
Given a ◯-state and a reset set, how to compute the next □-state?

- For each configuration ℓ, C, b, given moves (r', a) of □ and Y' of ◯





C: $0 < x - y < 1$ $\quad$ r': $0 < y < 1$

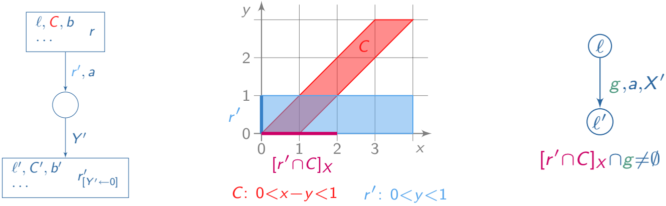


$[r' \cap C]_X \cap g \neq \emptyset$

States' update

Given a ◯-state and a reset set, how to compute the next □-state?

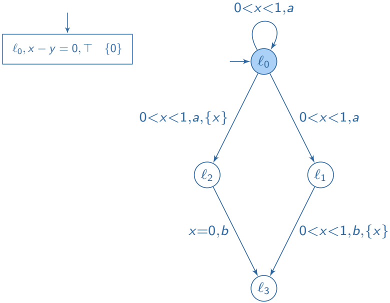
- For each configuration ℓ, C, b, given moves (r', a) of □ and Y' of ◯



- for each transition $\ell \xrightarrow{g,a,X'} \ell'$ with $[r' \cap C]_{|X} \cap g \neq \emptyset$ build a successor configuration ℓ', C', b'
  - C' updates C according to r', g, X', Y': $C' = \overleftrightarrow{(r' \cap C \cap g)}_{[X' \leftarrow 0][Y' \leftarrow 0]}$
  - b' indicates possible over-approximations: $b' = b \wedge ([r' \cap C]_{|X} \cap \neg g = \emptyset)$

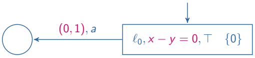
Game on the example

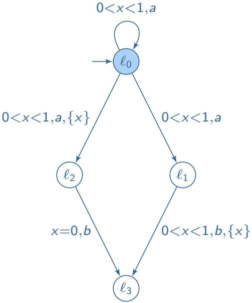
Construction of the game with resources $(y, 1)$



Game on the example

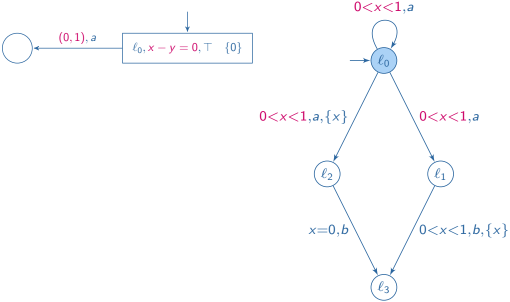
Construction of the game with resources $(y, 1)$





Game on the example

Construction of the game with resources $(y, 1)$

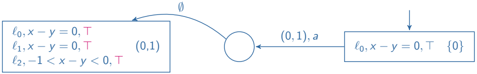


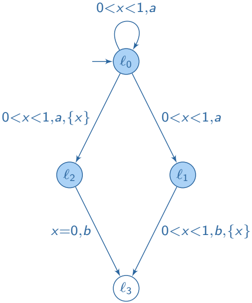
$y \in (0,1) \wedge x - y = 0 \implies x \in (0,1)$

no overapproximation

Game on the example

Construction of the game with resources $(y, 1)$



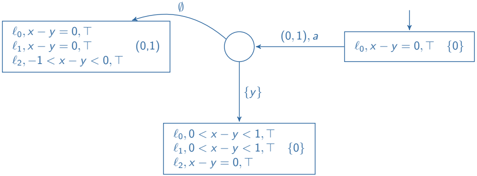


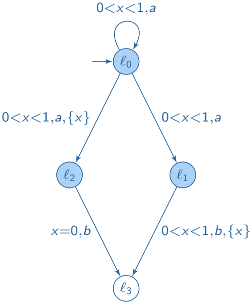
$y \in (0,1) \wedge x-y=0 \implies x \in (0,1)$

no overapproximation

Game on the example

Construction of the game with resources $(y, 1)$



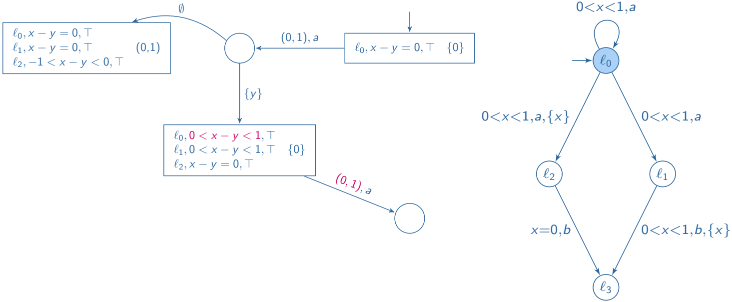


$y \in (0,1) \wedge x-y=0 \implies x \in (0,1)$

no overapproximation

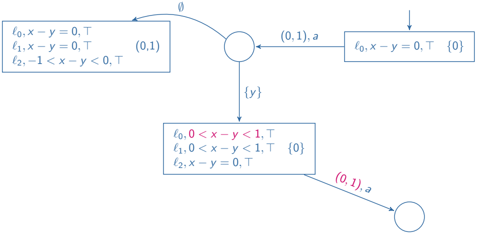
Game on the example

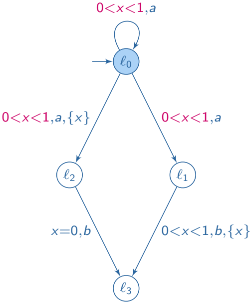
Construction of the game with resources $(y, 1)$



Game on the example

Construction of the game with resources $(y, 1)$



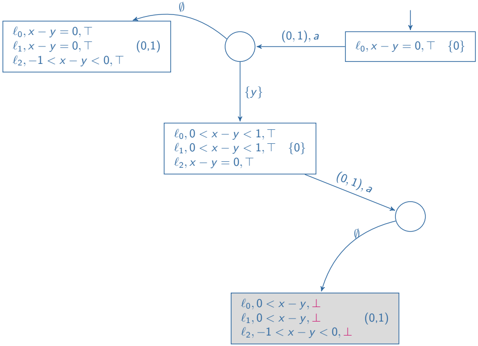


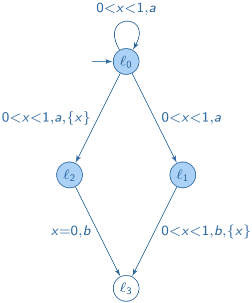
$y \in (0,1) \wedge 0 < x-y < 1 \implies 0 < x < 2$

overapproximation

Game on the example

Construction of the game with resources $(y, 1)$



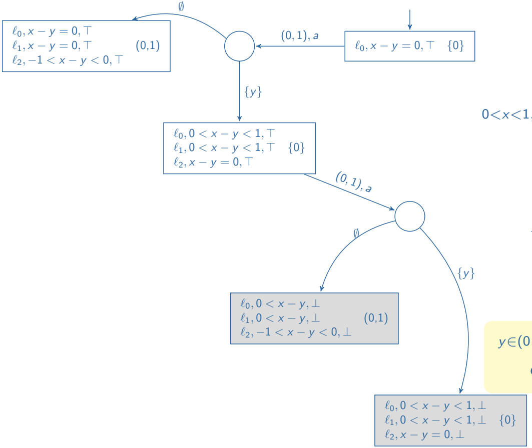


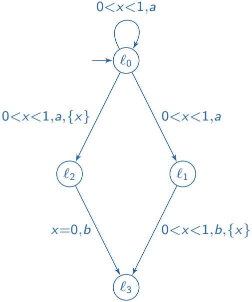
$y \in (0,1) \wedge 0 < x - y < 1 \implies 0 < x < 2$

overapproximation

Game on the example

Construction of the game with resources $(y, 1)$



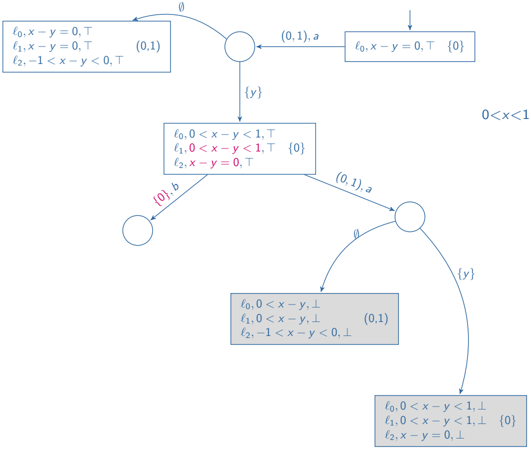


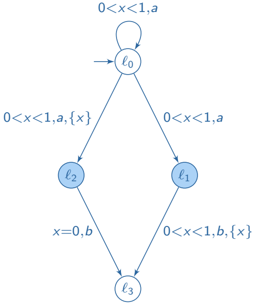
$y \in (0,1) \wedge 0 < x - y < 1 \implies 0 < x < 2$

overapproximation

Game on the example

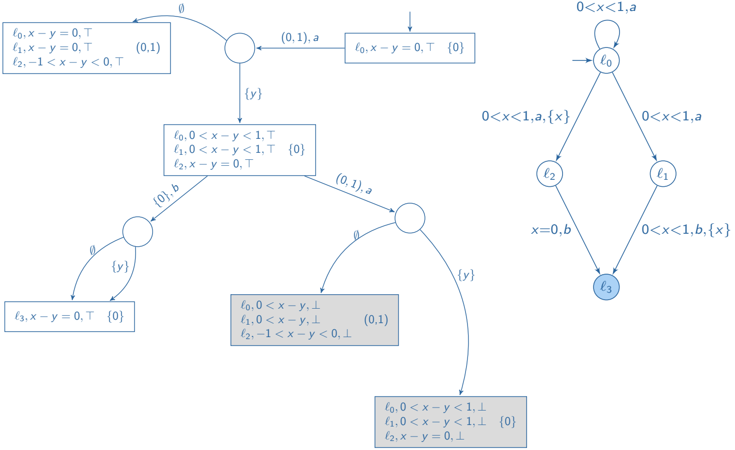
Construction of the game with resources $(y, 1)$





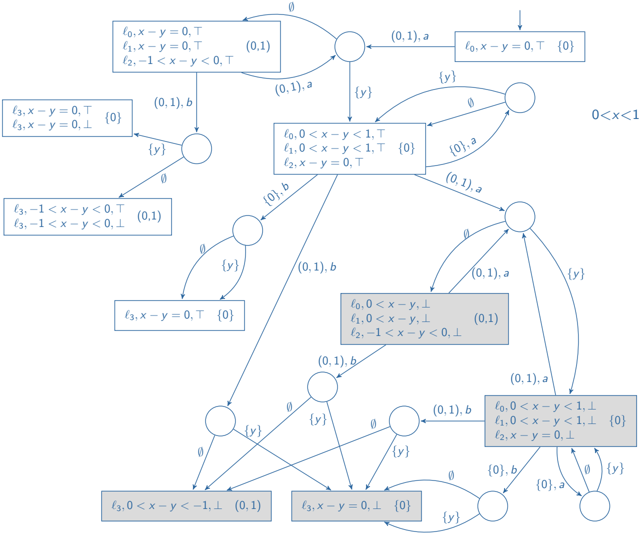
Game on the example

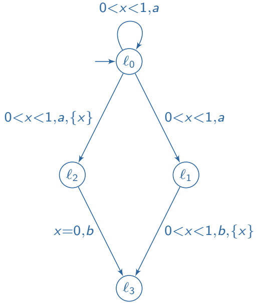
Construction of the game with resources $(y, 1)$



Game on the example

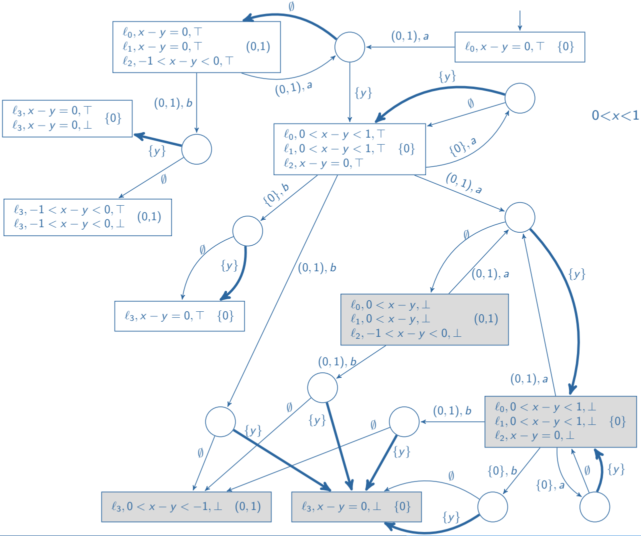
Construction of the game with resources $(y, 1)$

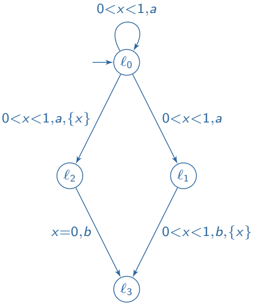




Resolution of the game

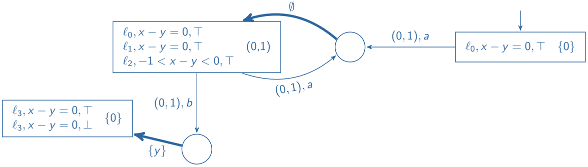
Winning strategy for Determinizator

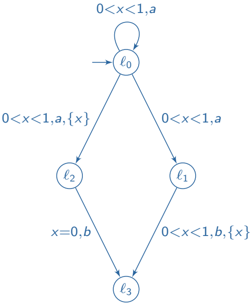




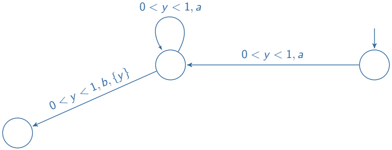
Resolution of the game

Winning strategy for Determinizator





Deterministic equivalent



Comparison with other approaches

- More precise than the over-approximation of [KT09]
  - general strategies compared to *a priori* fixed blind ones
  - determinism is preserved (under sufficient resources)

→ Exact determinization in more cases.

Comparison with other approaches

- More precise than the over-approximation of [KT09]
  - general strategies compared to *a priori* fixed blind ones
  - determinism is preserved (under sufficient resources)

$\longrightarrow$ Exact determinization in more cases.

- More general than the determinization procedure of [BBBB09]
  - relations are more expressive than mapping
  - dealing with some traces inclusion thanks to marking

$\longrightarrow$ Strictly more timed automata can be determinized.
$\longrightarrow$ Some timed automata are determinized with less resources.

Details

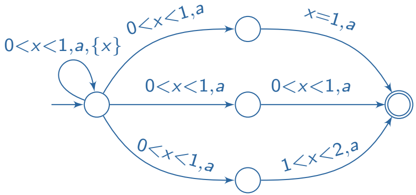
Limits

No winning strategy $\not\Rightarrow$ no deterministic equivalent

Limits

No winning strategy $\not\Rightarrow$ no deterministic equivalent

- Example

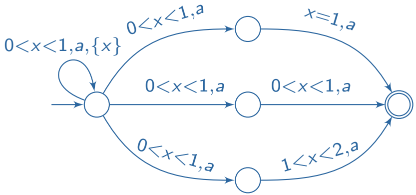


  - no winning strategy (with resources (1,1))
  - but some losing strategy yields a deterministic equivalent

Limits

No winning strategy $\not\Rightarrow$ no deterministic equivalent

- Example



  - no winning strategy (with resources (1,1))
  - but some losing strategy yields a deterministic equivalent

- How to choose a good losing strategy?
  - when possible, use language inclusion
  - heuristic: maximize distance to unsafe states
  - other possibilities: use quantities on timed languages such as volume

Summary

Determinization of timed automata is impossible in general, and determinizability is undecidable.

Approaches:

- determinizable subclasses (event-clock, integer reset)
- determinization procedure
  - terminates under some condition
  - produces a deterministic equivalent on known determinizable subclasses
- overapproximate determinization
  - produces a deterministic overapproximation
  - always terminates
- game-based approach

Recent contribution

Game-based approach to (approximately) determinize timed automata

- improves existing approaches
  - more timed automata determinized
  - exact determinization in more cases
  - less resources needed
- deals with timed automata with ε-transitions and invariants
- extension to timed automata with inputs and outputs
  $\longrightarrow$ application to testing

References

Baier, B. , Bouyer, Brihaye
When are timed automata determinizable?
ICALP 2009

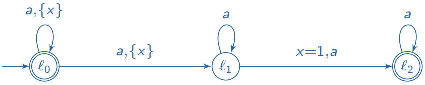
Krichen, Tripakis
Conformance testing for real-time systems
Formal Methods in System Design, 2009

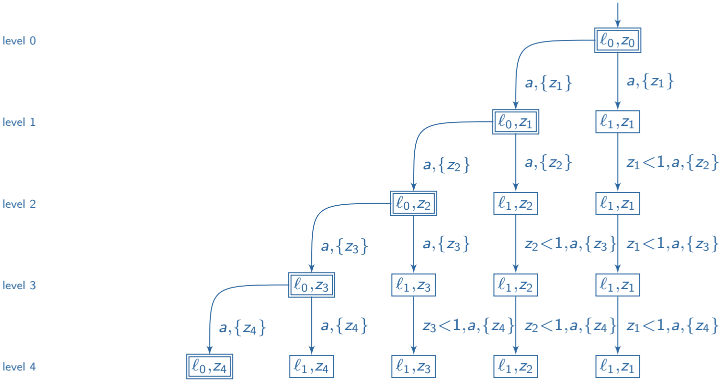
Bouyer, Chevalier, D'Souza
Fault diagnosis using timed automata
FoSSaCS 2005

B., Stainer, Jéron, Krichen
A game approach to determinize timed automata
FoSSaCS 2011

B., Jéron, Stainer, Krichen
Off-line test selection with test purposes for non-deterministic timed automata
TACAS 2011

Detailed comparison

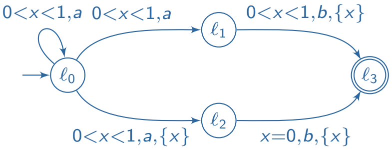


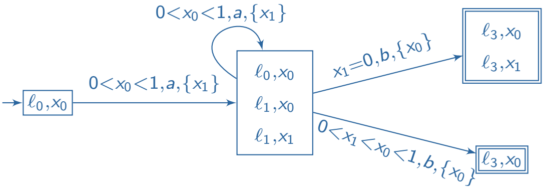


Determinization procedure fails.

Back

Detailed comparison





Determinization procedure produces a 2-clock automaton.

Back