A Game Approach to Determinize Timed Automata

Nathalie Bertrand[1], Amélie Stainer[1],
Thierry Jéron[1] and Moez Krichen[2]

[1]INRIA Rennes, France

[2]Sfax University, Tunisia

Automata Concurrency and Timed Systems III
January 27th 2011

Outline

Motivations

Determinization is central to many problems in formal methods:

- implementability
- diagnosis
- test generation

Unfortunately, determinization is not possible for all timed automata.

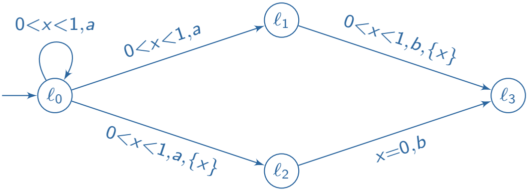
Syntax

Timed automata

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

- L finite set of locations $L_0 \subseteq L$ initial locations
- Σ finite alphabet
- X finite set of clocks
- $E \subseteq L \times \Sigma \times G \times 2^X \times L$ set of edges

where $G = \{\bigwedge x \sim c \mid x \in X, c \in \mathbb{N}\}$ is the set of guards.



Resources of $\mathcal{A}$: (X, M) with M the maximal constant

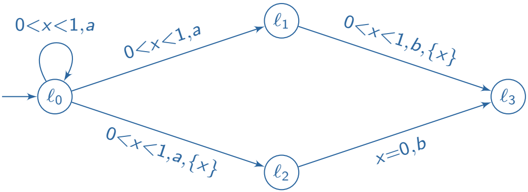
Semantics

States of $\mathcal{A}$: $L \times (\mathbb{R}_+)^X$

Transitions between states of $\mathcal{A}$: $(\ell, v) \xrightarrow{\tau,a} (\ell', v')$ if $\exists(\ell, a, g, Y, \ell') \in E$ with $v + \tau \models g$, $v'(x) = 0$ if $x \in Y$, and $v'(x) = v(x) + \tau$ otherwise.

Run of $\mathcal{A}$: $(\ell_0, v_0) \xrightarrow{\tau_0,a_0} (\ell_1, v_1) \xrightarrow{\tau_1,a_1} (\ell_2, v_2) \ldots$

Timed word: $(a_0, \tau_0)(a_1, \tau_0 + \tau_1) \cdots$



$(\ell_0, 0) \xrightarrow{0.3,a} (\ell_0, 0.3) \xrightarrow{0.5,a} (\ell_1, 0.8) \xrightarrow{0.1,b} (\ell_3, 0)$ $(a, 0.3)(a, 0.8)(b, 0.9)$

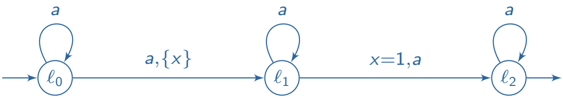
$(\ell_0, 0) \xrightarrow{0.3,a} (\ell_0, 0.3) \xrightarrow{0.6,a} (\ell_2, 0) \xrightarrow{0,b} (\ell_3, 0)$ $(a, 0.3)(a, 0.9)(b, 0.9)$

Deterministic timed automata

Deterministic timed automata

$\mathcal{A}$ is deterministic whenever for every timed word w, there is at most one initial run on w in $\mathcal{A}$.

Some timed automata are not determinizable [AD90].



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

Theorem [Finkel 06]

Checking whether a given timed automata is determinizable is undecidable.

Existing Approaches

- Exhibit determinizable subclasses
  - Event-recording automata [AFH94]
  - Integer-reset timed automata [SPKM08]
  - Unifying determinization procedure [BBBB09]
- Perform an approximate determinization
  - Deterministic over-approximation [KT09]

Determinization procedure [BBBB09]

Overview of the approach

- unfolding of the automaton, introducing a fresh clock at each step
- symbolic determinization
- reduction of the number of clocks* and folding back into an automaton
  * only possible under hypotheses.
- 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

Deterministic over-approximation [KT09]

Overview of the approach

- behaviour is observed by a new clock, reset at each step
- over-approximation of the guards according to the observation clock
- estimation of the current possible states
- extension to several new clocks with reset policy given by a DFA

Essential features

- fixed policy for the resets of the new clock
- no assumptions for termination

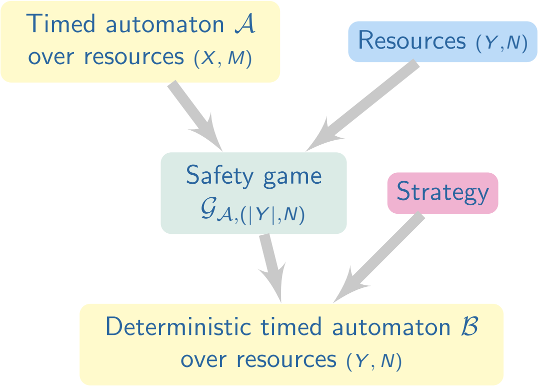
Outline

1. Introduction
2. A game approach
   - Presentation
   - The approach examplified
   - Comparison with existing methods and limits
3. Application to testing
4. Conclusion

A game approach

- 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



Game overview

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
- Unsafe states when a strict over-approximation possibly happened.

Properties of the game

Given a timed automaton, and fixed resources,

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

$\longrightarrow$ Restriction fo finite-memory (or even memoryless) strategies.

States and moves

States of Spoiler (S-states):

- a set of configurations each with a marker
  - configuration: location + relation between old and new clocks
  - marker: $\top$ or $\bot$ to indicate possible overapproximations
- a region (on the new set of clocks)

$\ell_0, x - y = 0, \top$ $\ell_1, 0 < x - y < 1, \top$ $\ell_2, -1 < x - y < 0, \bot$	(0,1)

Spoiler chooses a successor region and an action.

Given (X, M) original resources and (Y, N) new ones,
a relation is a conjunction of constraints
$x - y \sim c$ for $x \in X$, $y \in Y$ and $c \in [-N, M]$.

States and moves

States of Spoiler (S-states):

- a set of configurations each with a marker
  - configuration: location + relation between old and new clocks
  - marker: $\top$ or $\bot$ to indicate possible overapproximations
- a region (on the new set of clocks)

$\ell_0, x - y = 0, \top$
$\ell_1, 0 < x - y < 1, \top$ (0,1)
$\ell_2, -1 < x - y < 0, \bot$

Spoiler chooses a successor region and an action.

States of Determinizator (D-states):

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

$\ell_0, x - y = 0, \top$
$\ell_1, 0 < x - y < 1, \top$ (0,1) $\xrightarrow{y = 1, b}$
$\ell_2, -1 < x - y < 0, \bot$

Determinizator chooses a reset set.

States' update, bad states

Given a D-state and a reset set, how to compute the next S-state?

- For each configuration ℓ, C, b in the state
- given the moves of Spoiler (r', a) and Determinizator Y'
- 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' with
  - C' is the update of C according to r', g, X', Y'

    $$C' = \overleftrightarrow{(r' \cap C \cap g)_{[X' \leftarrow 0][Y' \leftarrow 0]}}$$

  - b' indicates if some over-approximation possibly occurred

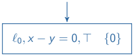
    $$b' = b \wedge ([r' \cap C]_{|X} \cap \neg g = \emptyset)$$

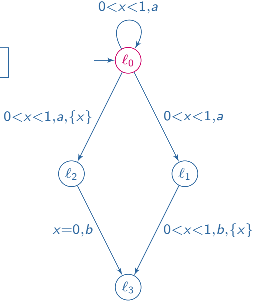
Bad states: S-states of the form $(\{\ell_i, C_i, \bot\}_{i \in I}, r)$

Outline

Arena on the example

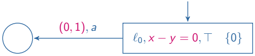
Construction of the game with resources $(y, 1)$

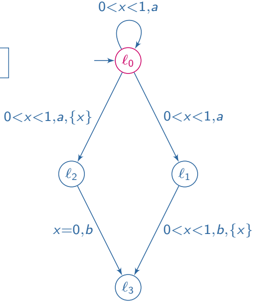




Arena on the example

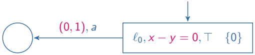
Construction of the game with resources $(y, 1)$

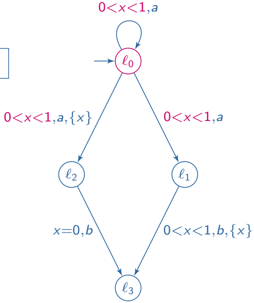




Arena on the example

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



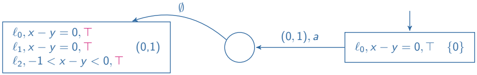


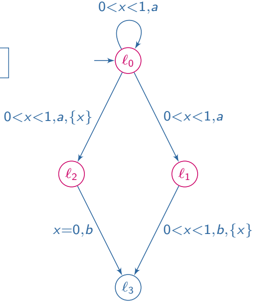
$y \in (0,1) \wedge x - y = 0 \implies x \in (0,1)$

no overapproximation

Arena on the example

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



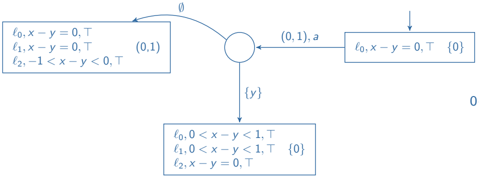


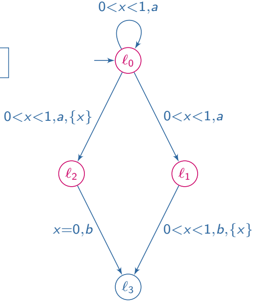
$y \in (0,1) \wedge x-y=0 \implies x \in (0,1)$

no overapproximation

Arena on the example

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



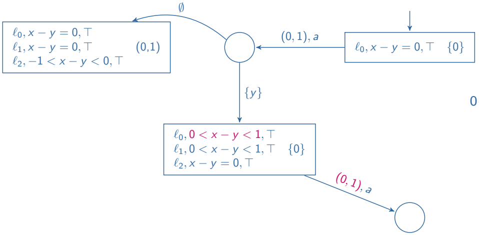


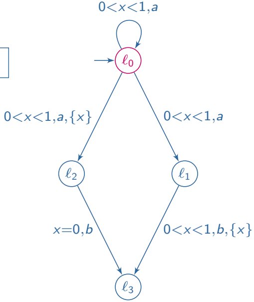
$y \in (0,1) \wedge x-y=0 \implies x \in (0,1)$

no overapproximation

Arena on the example

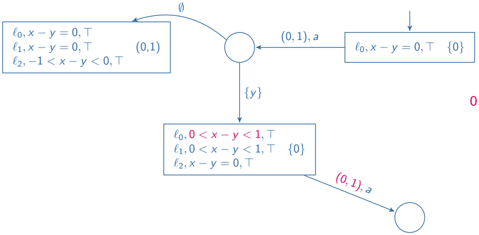
Construction of the game with resources $(y, 1)$

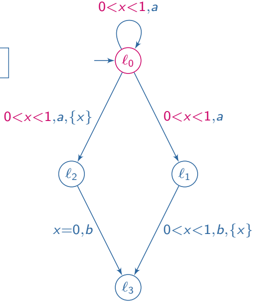




Arena on the example

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



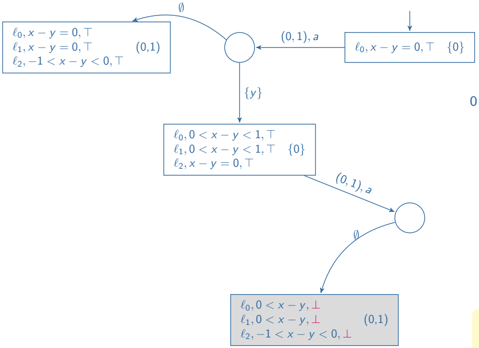


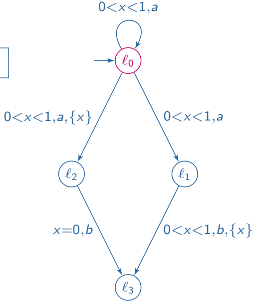
$y \in (0,1) \wedge 0<x-y<1 \implies 0<x<2$

overapproximation

Arena on the example

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



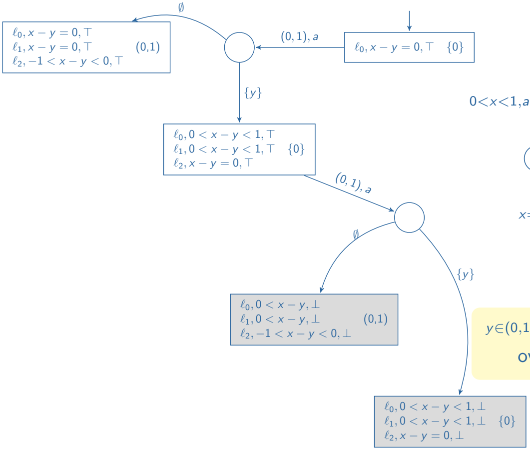


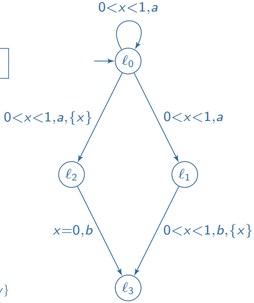
$y \in (0,1) \wedge 0 < x - y < 1 \implies 0 < x < 2$

overapproximation

Arena on the example

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



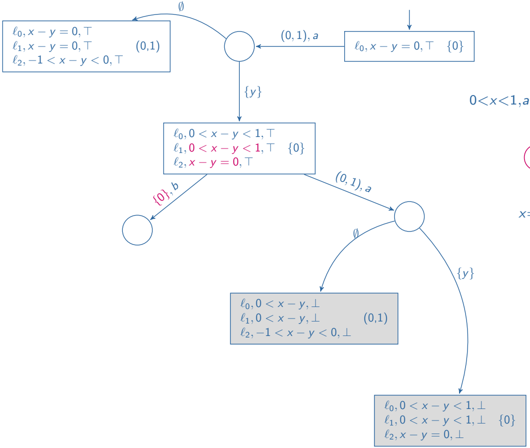


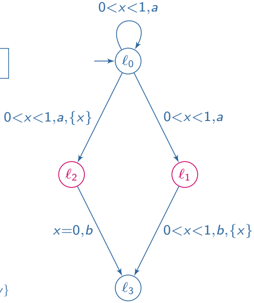
$y \in (0,1) \wedge 0<x-y<1 \implies 0<x<2$

overapproximation

Arena on the example

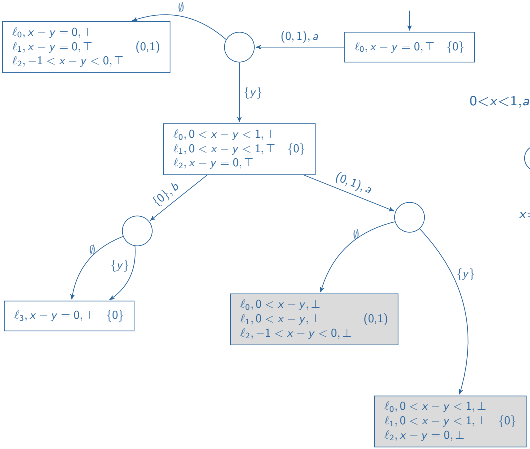
Construction of the game with resources $(y, 1)$

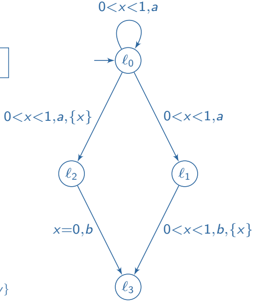




Arena on the example

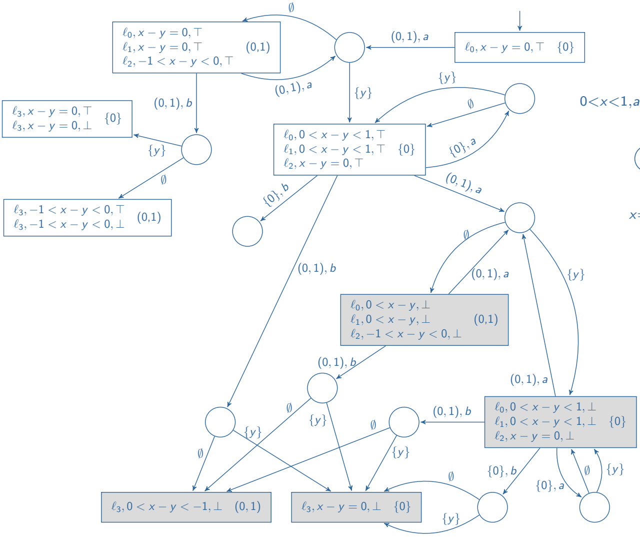
Construction of the game with resources $(y, 1)$

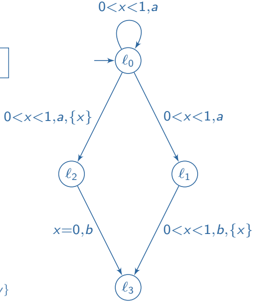




Arena on the example

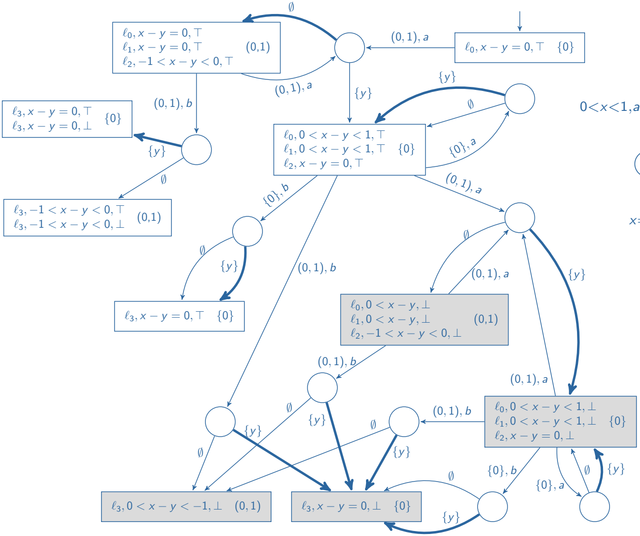
Construction of the game with resources $(y, 1)$

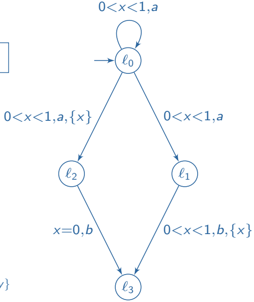




Resolution of the game

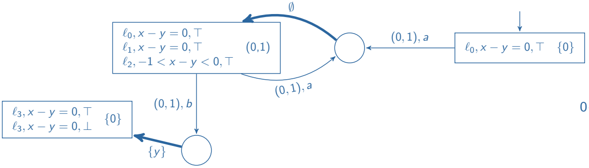
Winning strategy for Determinizator

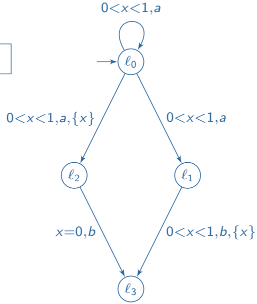




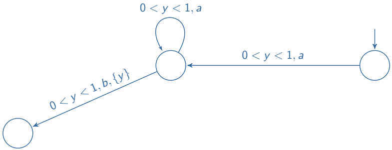
Resolution of the game

Winning strategy for Determinizator





Deterministic equivalent



Outline

Comparison with existing methods

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

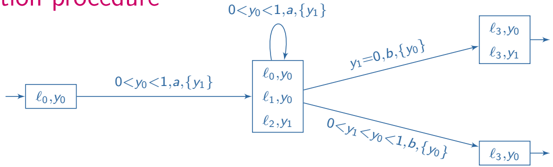
  $\longrightarrow$ Exact determinization in more cases.

- More general than the determinization procedure of [BBBB09]
  - relations are more expressive than mapping
  - some trace inclusions are treated

  $\longrightarrow$ Stricly more timed automata can be determinized, and some timed automata are determinized with less resources.

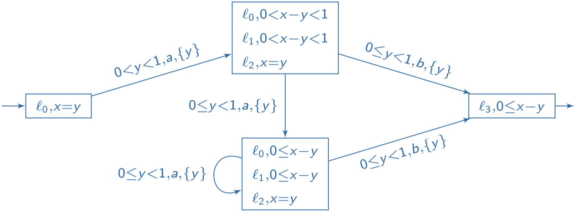
Existing methods on the example

Determinization procedure



→ Needs two clocks.

Overapproximation algorithm

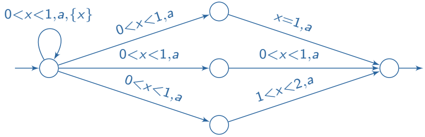


→ Strict over-approximation.

Limits

No winning strategy for D in $\mathcal{G}_{\mathcal{A},(k,N)}$
$\not\Rightarrow$ no deterministic equivalent for $\mathcal{A}$ with resources (k, N)

- Example



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

- How to choose a good losing strategy?
  - heuristic: maximize distance to Bad states
  - other possibilities: use quantities on timed languages such as volume

Outline

1. Introduction
2. A game approach
   - Presentation
   - The approach examplified
   - Comparison with existing methods and limits
3. Application to testing
4. Conclusion

Test generation for timed systems

Problem

Given a specification (nondeterministic timed automaton with inputs and outputs), generate off-line tests (deterministic timed automaton).

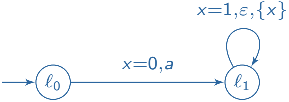
Essential features for testing real-time systems

- internal actions $\longrightarrow$ ε-closure
- input/output $\longrightarrow$ under- and over-approximations
- urgency $\longrightarrow$ over-approximation of invariants

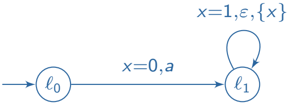
Timed automata with input-output

A TAIO is a timed automaton over alphabet $\Sigma = \Sigma_! \sqcup \Sigma_?$, possibly with ε-transitions and invariants on locations.

Dealing with ε-transitions



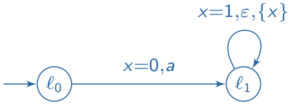
Dealing with ε-transitions



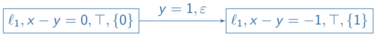
ε-closure in the game with resources $(y, 2)$:

$\ell_1, x - y = 0, \top, \{0\}$

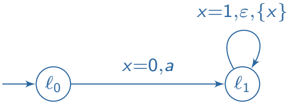
Dealing with ε-transitions



ε-closure in the game with resources $(y, 2)$:



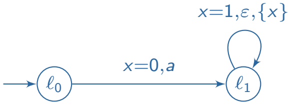
Dealing with ε-transitions



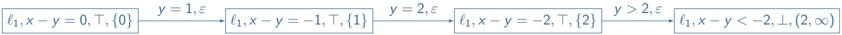
ε-closure in the game with resources $(y, 2)$:

$$\boxed{\ell_1, x-y=0, \top, \{0\}} \xrightarrow{y=1,\varepsilon} \boxed{\ell_1, x-y=-1, \top, \{1\}} \xrightarrow{y=2,\varepsilon} \boxed{\ell_1, x-y=-2, \top, \{2\}}$$

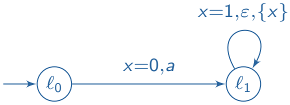
Dealing with ε-transitions



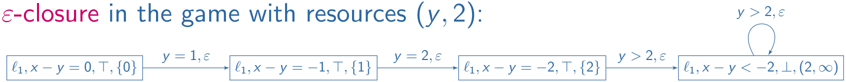
ε-closure in the game with resources $(y,2)$:



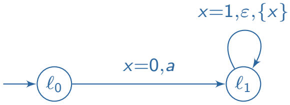
Dealing with ε-transitions



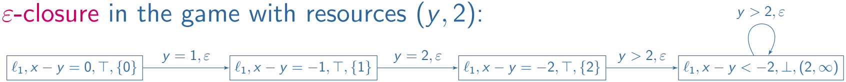
ε-closure in the game with resources $(y, 2)$:



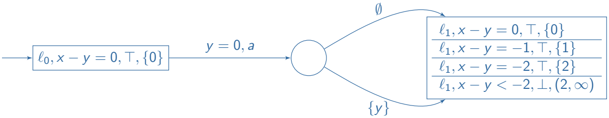
Dealing with ε-transitions



ε-closure in the game with resources $(y, 2)$:



Resulting game



Extension of the approach

Refinement relation

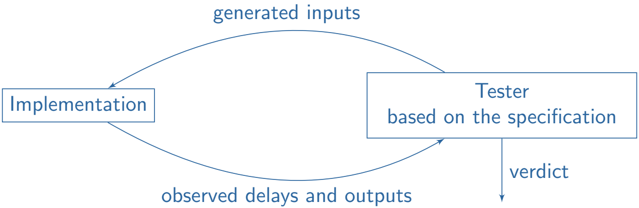
Given $\mathcal{A}$ a TAIO and $\mathcal{A}'$ a deterministic TAIO, $\mathcal{A}$ *refines* $\mathcal{A}'$ (noted $\mathcal{A} \preceq \mathcal{A}'$) if there exists a relation $\rho \subseteq S \times S'$ such that: $(s_0, s'_0) \in \rho$ and

- $\forall (s, s') \in \rho$, $\forall s \xrightarrow{\tau_1,\varepsilon} s_1 \cdots s_{k-1} \xrightarrow{\tau_k,a} t$ with a an **output** action, $\exists s' \xrightarrow{\sum \tau_i, a} t'$ with $(s', t') \in \rho$, and
- $\forall s' \xrightarrow{\tau,a} t'$ with a an **input** action, $\exists (s, s') \in \rho$, $(t, t') \in \rho$ and $s \xrightarrow{\tau_1,\varepsilon} s_1 \cdots s_{k-1} \xrightarrow{\tau_k,a} t$ where $\sum \tau_i = \tau$.

Properties of the game

- Every strategy of Determinizator yields a deterministic TAIO $\mathcal{B}$ with $\mathcal{A} \preceq \mathcal{B}$.
- Every winning strategy of Determinizator yields a deterministic equivalent.

Conformance testing



- Soundness: preserves verdict Pass (risk to forget some Fail)
- Strictness: no forgotten Fail

Tester's properties

Exact determinization → sound and strict test suite.
Deterministic abstraction → sound test suite.

Outline

Conclusion

Contribution: Game-based approach to (approximately) determinize timed automata

- improves existing approaches [BBBB09,KT09]
  - 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

Future work

- Implementation?
- Application to other problems and models.

References

[AD90] Alur, Dill. *A theory of timed automata*. ICALP 1990.
[Finkel06] Finkel. *Undecidable problems about timed automata*. Formats 2006.
[AFH94] Alur, Fix, Henzinger. *Event-clock automata: a determinizable class of timed automata*. CAV 2004.
[SPKM08] Suman, Pandya, Krishna, Manasa. *Timed automata with integer resets: language inclusion and expressiveness*. Formats 2008.

[BBBB09] Baier, B. , Bouyer, Brihaye. *When are timed automata determinizable?* ICALP 2009.
[KT09] Krichen, Tripakis. *Conformance testing for real-time systems*. FMSD 2009.
[BCD05] Bouyer, Chevalier, D'Souza. *Fault diagnosis using timed automata*. FoSSaCS 2005.

[BSJK11] B. , Stainer, Jéron, Krichen. *A game approach to determinize timed automata*. FoSSaCS 2011.
[BJSK11] B. , Jéron, Stainer, Krichen. *Off-line test selection with test purposes for non-deterministic timed automata*. TACAS 2011.