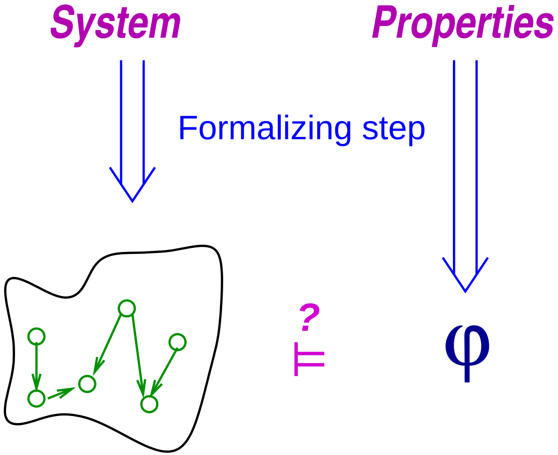
Models for Efficient Timed Verification

François Laroussinie

LSV / ENS de Cachan – CNRS UMR 8643

Monterey Workshop - “Composition of embedded systems”

Model checking



Model checking

We want. . .

- Expressive models: communication, variables, timing constraints, probabilities. . .
- Expressive specification languages: natural, powerful, intuitive. . .

Model checking

We want. . .

- Expressive models: communication, variables, timing constraints, probabilities. . .
- Expressive specification languages: natural, powerful, intuitive. . .

AND EFFICIENT ALGORITHM !

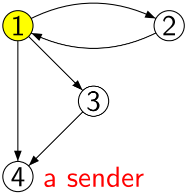
Main limit of model checking = state explosion problem

State explosion problem

An example: a “protocol”. . .

State explosion problem

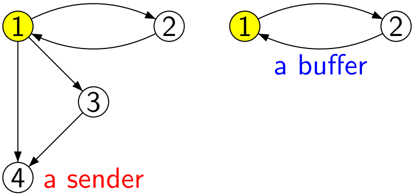
An example: a “protocol”...



a sender

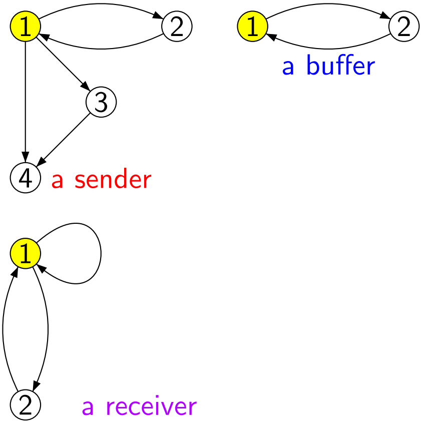
State explosion problem

An example: a "protocol"...



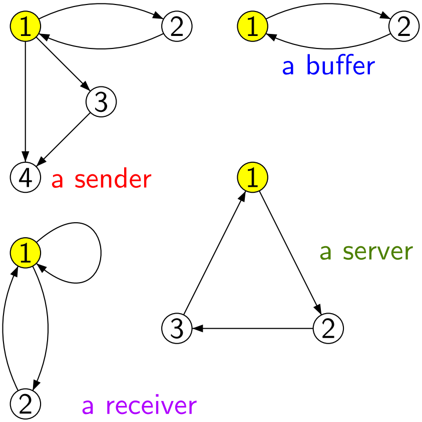
State explosion problem

An example: a “protocol”...



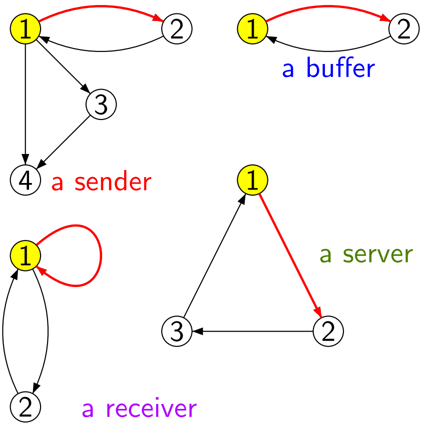
State explosion problem

An example: a "protocol"...



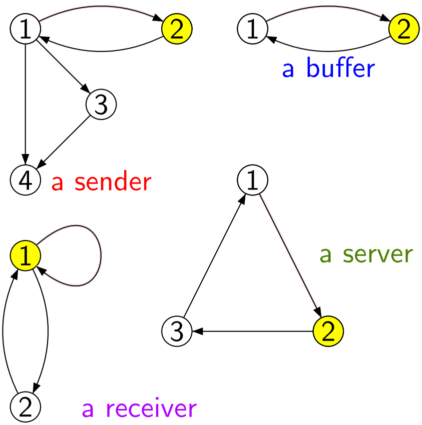
State explosion problem

An example: a "protocol"...

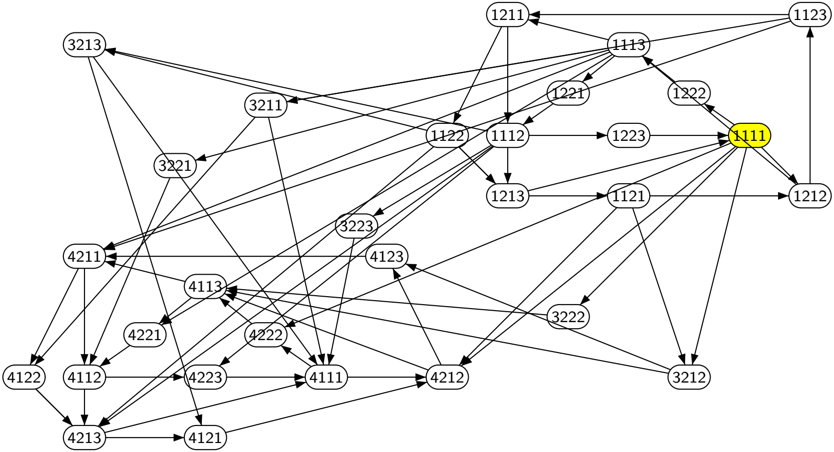


State explosion problem

An example: a “protocol”...



State explosion problem



State explosion problem

This has motivated

- symbolic methods
- heuristics:
  - on-the-fly algorithms
  - acceleration
  - partial order
  - ...
- abstraction
- ...

Model checkers exist and they work rather nicely !

Verification of compositions of embedded systems

Two difficulties:

- the composition. . .

- Usually we need to address verification of quantitative aspects:

  - timing constraints (Real-time systems)
  - probabilities
  - costs
  - data
  - . . .

Verification of compositions of embedded systems

Two difficulties:

- the composition...

- Usually we need to address verification of quantitative aspects:

    - timing constraints (Real-time systems)
    - probabilities
    - costs
    - data
    - ...

Each one may induce a complexity blow-up.

This can be measured by the structural complexity of verification problems: for ex. the composition and the state explosion problem.

State explosion problem

This state explosion holds for synchronized systems, systems with boolean variables, 1-safe Petri nets etc.

$\rightarrow$ These are "non-flat" systems.

In practice, a model S is described as a non-flat system (whose operational semantics is a "flat" transition system T)

State explosion problem

This state explosion holds for synchronized systems, systems with boolean variables, 1-safe Petri nets etc.

$\rightarrow$ These are "non-flat" systems.

In practice, a model S is described as a non-flat system (whose operational semantics is a "flat" transition system T)

Complexity of model checking non-flat systems

Model checking	$T \models \Phi$	"non-flat syst." $\models \Phi$
Reachability	NLOGSPACE-C	PSPACE-C
CTL model checking	P-C	PSPACE-C
AF μ-calculus	P-C	EXPTIME-C

(Papadimitriou, Vardi, Kupferman, Wolper, Rabinovich,...)

Challenge

Define new formalisms to model and verify the embedded systems, with. . .

- timing constraints
- probabilities
- costs, dynamic variables
- . . .
- extended specification languages

Challenge

Define new formalisms to model and verify the embedded systems, with. . .

- timing constraints
- probabilities
- costs, dynamic variables
- . . .
- extended specification languages
  - request $\Rightarrow$ AF grant
  - request $\Rightarrow$ $\mathsf{AF}_{<200}$ grant
  - request $\Rightarrow$ $\mathbb{P}_{>0.9}\mathsf{F}_{<200}$ grant
  - request $\Rightarrow$ $\langle \mathsf{Agt}_1 \rangle \mathbb{P}_{>0.9}\mathsf{F}_{<200}$ grant

Challenge

Define new formalisms to model and verify the embedded systems, with. . .

- timing constraints
- probabilities
- costs, dynamic variables
- . . .
- extended specification languages
  - request $\Rightarrow$ AF grant
  - request $\Rightarrow$ $\mathsf{AF}_{<200}$ grant
  - request $\Rightarrow$ $\mathbb{P}_{>0.9}\mathsf{F}_{<200}$ grant
  - request $\Rightarrow$ $\langle \mathsf{Agt}_1 \rangle \mathbb{P}_{>0.9}\mathsf{F}_{<200}$ grant

without increasing (too much) the complexity !

Efficient timed model-checking

obj: find timed models and timed specification languages for which verification can be done efficiently.

$\longrightarrow$ To model simply timed systems

$\longrightarrow$ To use to abstract complex timed systems

$\longrightarrow$ To combine with other quantitative extensions

Outline

1. Timed specification languages
2. Timed models
   - Discrete-time models
   - Timed automata
3. Conclusion

Outline

1. Timed specification languages
2. Timed models
   - Discrete-time models
   - Timed automata
3. Conclusion

Classical verification problems

- Reachability of a control state
- $\mathcal{S} \sim \mathcal{S}'$? : (bi)simulation, etc.
- $L(\mathcal{S}) \subseteq L(\mathcal{S})$? : language inclusion
- $(\mathcal{S}|A_T)$ + reachability : testing automata
- $S \models \Phi$ with Φ a temporal logic formula : model checking

Timed properties

Temporal properties:

"Any problem is followed by an alarm"

With CTL: $\mathsf{AG}\left(\mathsf{problem} \Rightarrow \mathsf{AF}\,\mathsf{alarm}\right)$

Timed properties

Temporal properties:

"Any problem is followed by an alarm"

With CTL: $\mathsf{AG}\left(\mathsf{problem} \Rightarrow \mathsf{AF}\,\mathsf{alarm}\right)$

Timed properties:

"Any problem is followed by an alarm in at most 10 time units"

Timed properties

Temporal properties:

"Any problem is followed by an alarm"

With CTL: $\mathsf{AG}\left(\mathsf{problem} \Rightarrow \mathsf{AF}\,\mathsf{alarm}\right)$

Timed properties:

"Any problem is followed by an alarm in at most 10 time units"

With **T**CTL:

$$\mathsf{AG}\left(\mathsf{problem} \Rightarrow \mathsf{AF}_{\leq 10}\,\mathsf{alarm}\right)$$

Timed CTL

We use *TCTL* whose formulae are built from:

- atomic propositions in Prop (For ex. Alarm, Problem, ...)
- boolean combinators ($\wedge$, $\vee$, $\neg$), and
- temporal operators tagged with timing constraints: $\mathsf{E}\ \_\ \mathsf{U}_{\sim c}\ \_$ and $\mathsf{A}\ \_\ \mathsf{U}_{\sim c}\ \_$ with $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

+ all the standard abbreviations: $\mathsf{AG}_{\sim c}$, $\mathsf{AF}_{\sim c}$ etc.

Timed CTL

We use *TCTL* whose formulae are built from:

- atomic propositions in Prop (For ex. Alarm, Problem, ...)
- boolean combinators ($\wedge$, $\vee$, $\neg$), and
- temporal operators tagged with timing constraints: $\mathsf{E}\,\_\,\mathsf{U}_{\sim c}\,\_$ and $\mathsf{A}\,\_\,\mathsf{U}_{\sim c}\,\_$ with $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

+ all the standard abbreviations: $\mathsf{AG}_{\sim c}$, $\mathsf{AF}_{\sim c}$ etc.

$$\begin{aligned} s \models \mathsf{E}\varphi\mathsf{U}_{\sim c}\psi \quad \Leftrightarrow \quad & \exists \rho = \rho' \cdot \rho'' \in \mathit{Exec}(s) \text{ with } s \xRightarrow{\rho'} s' \text{ and} \\ & \mathsf{Time}(\rho') \sim c \text{ and } s' \models \psi, \\ & \text{and } s'' \models \varphi \text{ for all } s <_\rho s'' <_\rho s' \end{aligned}$$

Outline

Questions

Studying timing constraints & complexity

Two problems:

1. Are there simpler models – with discrete time – for which model checking can be efficient ?
2. Are there classes of Timed Automata which allow efficient model checking ?

Timed transition systems

$=$ a transition system $+$ a notion of time
$\Rightarrow$ used to define the semantics of real-time systems

Let $\mathbb{T}$ be a time domain: $\mathbb{N}$, $\mathbb{R}_+$ or $\mathbb{Q}_+$.

Timed transition system

$\mathcal{T} = \langle S, s_0, \rightarrow, l \rangle$

- S is an (possibly infinite) set of states, $s_0 \in S$
- $\rightarrow \subseteq S \times \mathbb{T} \times S$
- $l : S \rightarrow 2^{\mathsf{Prop}}$: assigns atomic propositions to states

Timed transition systems

$=$ a transition system $+$ a notion of time
$\Rightarrow$ used to define the semantics of real-time systems

Let $\mathbb{T}$ be a time domain: $\mathbb{N}$, $\mathbb{R}_+$ or $\mathbb{Q}_+$.

Timed transition system

$\mathcal{T} = \langle S, s_0, \rightarrow, l \rangle$

- S is an (possibly infinite) set of states, $s_0 \in S$
- $\rightarrow \subseteq S \times \mathbb{T} \times S$
- $l : S \rightarrow 2^{\text{Prop}}$: assigns atomic propositions to states

Every finite run σ in $\mathcal{T}$ has a finite duration $\text{Time}(\sigma)$.

Outline

1. Timed specification languages
2. Timed models
   - Discrete-time models
   - Timed automata
3. Conclusion

Simple models for timed verification

It is possible to use classical Kripke structures as timed models.

There is no inherent concept of time: time elapsing is encoded by events.

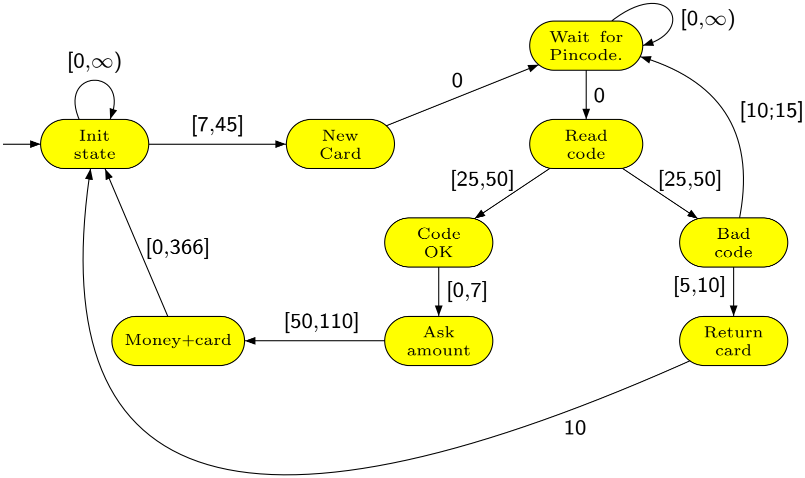
For example:

- each transition = one time unit (Emerson et al.)
- or: a "tick" proposition labels states where one t.u. elapses.

Timed model checking can be efficient (polynomial-time) !

Durational Transition Graphs extend these models without loosing efficiency.

Durational Transition Graph



Durational Transition Graph

$\mathbb{T} = \mathbb{N}$

A durational transition graph S is $\langle Q, R, l \rangle$ where

- Q is a (finite) set of *states*,
- $R \subseteq Q \times \mathcal{I} \times Q$ is a total *transition relation with duration*
- $l : Q \to 2^{\mathsf{Prop}}$ labels every state with a subset of Prop.

$\mathcal{I} =$ set of intervals "$[n, m]$" or "$[n, \infty)$" (with $n, m \in \mathbb{N}$)

$q \xrightarrow{[n,m]} q'$: "moving from q to q' takes some duration d in $[n, m]$."

Two semantics are possible. . .

Two semantics for DTGs

Consider the run: Init_State $\xrightarrow{15}$ New_Card $\xrightarrow{0}$ Wait_for_Pin C. $\xrightarrow{20}$ ...

Two semantics for DTGs

Consider the run: Init_State $\xrightarrow{15}$ New_Card $\xrightarrow{0}$ Wait_for_Pin C. $\xrightarrow{20}$ ...

- Jump semantics:
  - date: 0 $\rightsquigarrow$ Init_State
  - date: 15 $\rightsquigarrow$ New_Card   no intermediary states !
  - ...

$\oplus$ Simple semantics, no extra states.

$\ominus$ Not always natural. Difficult to synchronize two DTGs.

Two semantics for DTGs

Consider the run: Init_State $\xrightarrow{15}$ New_Card $\xrightarrow{0}$ Wait_for_Pin C. $\xrightarrow{20}$ ...

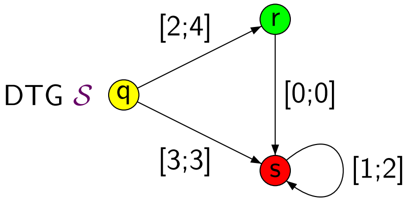
- Jump semantics:
  - date: 0 $\rightsquigarrow$ Init_State
  - date: 15 $\rightsquigarrow$ New_Card — no intermediary states !
  - ...

  $\oplus$ Simple semantics, no extra states.

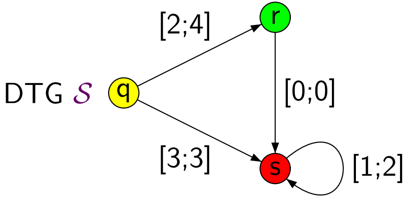
  $\ominus$ Not always natural. Difficult to synchronize two DTGs.

- Continuous semantics:
  - date: $0 \ldots 14$ $\rightsquigarrow$ Init_state
  - date: 15 $\rightsquigarrow$ New_card
  - date: $15 \ldots 34$ $\rightsquigarrow$ Wait_for_Pin_C
  - ...

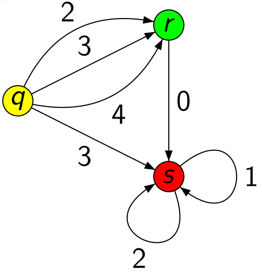
jump vs continuous semantics



jump vs continuous semantics

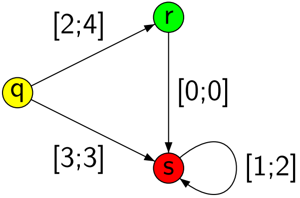


jump sem. of $\mathcal{S}$:

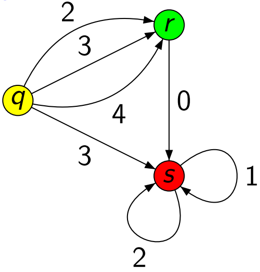


jump vs continuous semantics

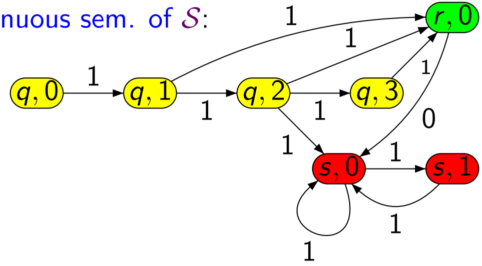
DTG $\mathcal{S}$



jump sem. of $\mathcal{S}$:



continuous sem. of $\mathcal{S}$:

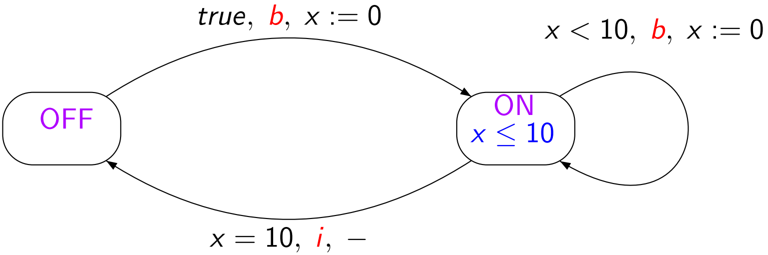


(q, i) : "i time units have already been spent in q"

Outline

Timed Automata – Example

$\mathcal{A}$ = an automaton (locations and transitions) + clocks



Timed automata - definition [Alur & Dill]

$\mathcal{A}$ = an automaton (locations and transitions) + clocks

Clocks progress synchronously with time (Time domain = $\mathbb{R}_+$)

Transitions: $\ell \xrightarrow{g,a,r} \ell' \in T$ with:

- g is the guard,
- a is the label,
- r is the set of clocks to be reset to 0

Timed automata - definition [Alur & Dill]

$\mathcal{A}$ = an automaton (locations and transitions) + clocks

Clocks progress synchronously with time (Time domain $= \mathbb{R}_+$)

Transitions: $\ell \xrightarrow{g,a,r} \ell' \in T$ with:

- g is the guard,
- a is the label,
- r is the set of clocks to be reset to 0

Semantics:

▲ **States:** (ℓ, v) where v is a valuation for the clocks

Timed automata - definition [Alur & Dill]

$\mathcal{A}$ = an automaton (locations and transitions) + clocks

Clocks progress synchronously with time (Time domain = $\mathbb{R}_+$)

Transitions: $\ell \xrightarrow{g,a,r} \ell' \in T$ with:

- g is the guard,
- a is the label,
- r is the set of clocks to be reset to 0

Semantics:

▲ **States:** (ℓ, v) where v is a valuation for the clocks

▲ **Action transition**:

$$(\ell, v) \xrightarrow{0}_a (\ell', v') \text{ iff } \begin{cases} \exists\, \ell \xrightarrow{g,a,r} \ell' \in \mathcal{A}, \\ v \models g, \quad v' = v[r \leftarrow 0] \end{cases}$$

Timed automata - definition [Alur & Dill]

$\mathcal{A}$ = an automaton (locations and transitions) + clocks

Clocks progress synchronously with time (Time domain = $\mathbb{R}_+$)

Transitions: $\ell \xrightarrow{g,a,r} \ell' \in T$ with:

- g is the guard,
- a is the label,
- r is the set of clocks to be reset to 0

Semantics:

▲ **States:** (ℓ, v) where v is a valuation for the clocks

▲ **Action transition**:

$$(\ell, v) \xrightarrow{0}_a (\ell', v') \text{ iff } \begin{cases} \exists\, \ell \xrightarrow{g,a,r} \ell' \in \mathcal{A}, \\ v \models g, \quad v' = v[r \leftarrow 0] \end{cases}$$

▲ **Delay transition**:

$$\forall t \in \mathbb{R}_+,\ (\ell, v) \xrightarrow{t} (\ell, v + t) \text{ iff } \forall 0 \leq t' \leq t,\ v + t' \models \mathtt{Inv}(\ell)$$

Model checking for TA

$\mathcal{A}$ defines an infinite timed transition system.

Decision procedures are based on the region graph technique (Alur, Courcoubetis, Dill)

From $\mathcal{A}$ and Φ, one defines an equivalence $\equiv_{\mathcal{A},\Phi}$ s.t.:

- $v \equiv_{\mathcal{A},\Phi} v' \;\Rightarrow\; \Big((\ell, v) \models \Phi \text{ iff } (\ell, v') \models \Phi\Big)$
- ${\mathbb{R}_+}^X_{/\equiv_{\mathcal{A},\Phi}}$ is finite

A region = An equivalence class of $\equiv_{\mathcal{A},\Phi}$

We can reduce $\mathcal{A} \models \Phi$ to $\Big(\mathcal{A} \times {\mathbb{R}_+}^X_{/\equiv_{\mathcal{A},\Phi}}\Big) \models \Phi$

…and use a standard model checking algorithm.

Model checking for TA

$\mathcal{A}$ defines an infinite timed transition system.

Decision procedures are based on the region graph technique (Alur, Courcoubetis, Dill)

From $\mathcal{A}$ and Φ, one defines an equivalence $\equiv_{\mathcal{A},\Phi}$ s.t.:

- $v \equiv_{\mathcal{A},\Phi} v' \;\Rightarrow\; \Big((\ell, v) \models \Phi \text{ iff } (\ell, v') \models \Phi\Big)$
- ${\mathbb{R}_+}^X_{/\equiv_{\mathcal{A},\Phi}}$ is finite

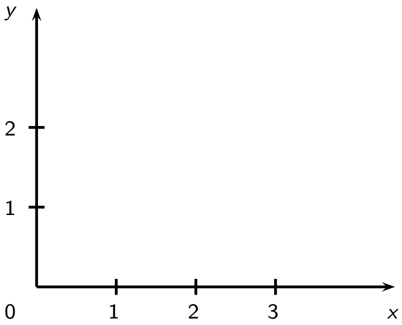
A region = An equivalence class of $\equiv_{\mathcal{A},\Phi}$

We can reduce $\mathcal{A} \models \Phi$ to $\left(\mathcal{A} \times {\mathbb{R}_+}^X_{/\equiv_{\mathcal{A},\Phi}}\right) \models \Phi$

...and use a standard model checking algorithm.

! The size of ${\mathbb{R}_+}^X_{/\equiv_{\mathcal{A},\Phi}}$ is in $O(|X|! \cdot M^{|X|})$ **!**

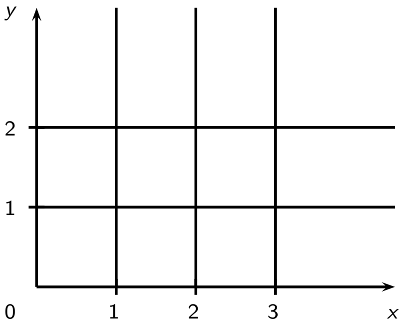
The region abstraction



$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

The region abstraction

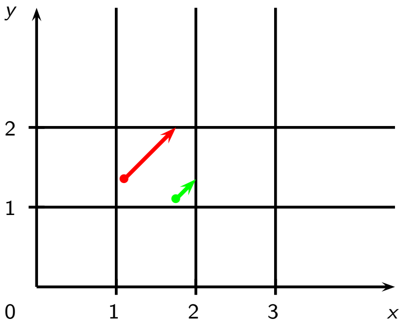


$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$

The region abstraction

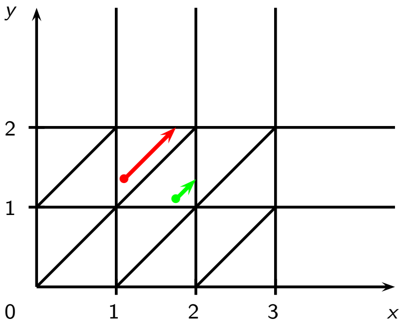


$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

The region abstraction

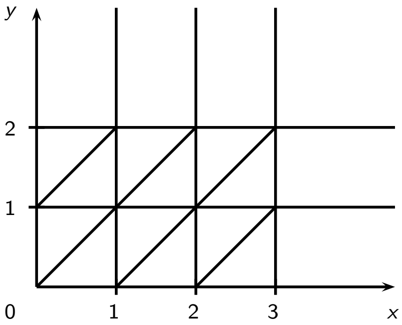


$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

The region abstraction

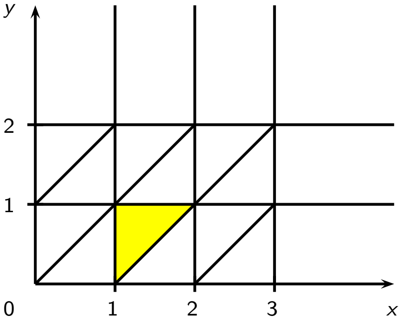


$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

The region abstraction



$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

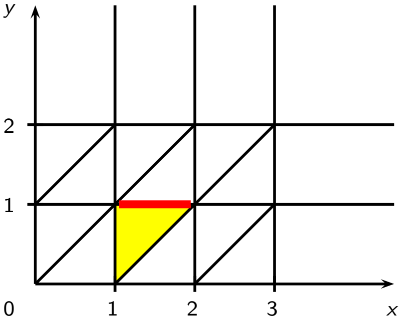
region defined by

$I_x =]1;2[$, $I_y =]0;1[$

$\{x\} < \{y\}$

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

The region abstraction



$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

region defined by

$I_x =]1;2[$, $I_y =]0;1[$

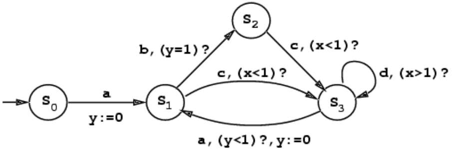
$\{x\} < \{y\}$

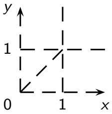
successor by delay transition:

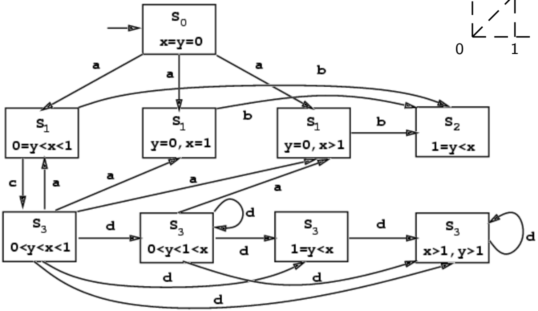
$I_x =]1;2[$, $I_y = [2;2]$

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

An example [AD 90's]







Complexity of timed verification

- Reachability in TA is PSPACE-C (Alur & Dill)
- Reachability in TA with three clocks is PSPACE-C (Courcoubetis & Yannakakis)
- Reachability in TA with constants in $\{0, 1\}$ is PSPACE-C (Courcoubetis & Yannakakis)
- Model checking Timed CTL over TA is PSPACE-C (Alur, Courcoubetis & Dill)
- Model checking AF Timed μ-calculus over TA is EXPTIME-C (Aceto & Laroussinie)

$\Big($ But tools (Kronos, UppAal) exist and have been applied successfully for verifying industrial case studies. $\Big)$

One/two clock timed automata

[LMS2004]

1. Model checking *TCTL* on 1C-TA is PSPACE-complete.
2. Reachability in 1C-TA is NLOGSPACE-complete.
3. Model checking $TCTL_{\leq,\geq}$ on 1C-TA is P-complete.

1. Reachability in 2C-TA is NP-hard.
2. Model checking *CTL* on 2C-TA is PSPACE-complete.

Outline

Conclusion

<table>
<tr><th rowspan="3"></th><th colspan="3">discrete time</th><th colspan="3">dense time</th></tr>
<tr><th rowspan="2">DTG $\xrightarrow{0/1}$
[LST03]</th><th colspan="2">DTG [LMS06]</th><th rowspan="2">1C-TA
[LMS04]</th><th rowspan="2">2C-TA
[LMS04]</th><th rowspan="2">TA [ACD93]
[CY92]</th></tr>
<tr><th>jump</th><th>cont.</th></tr>
<tr><td>Reachability</td><td colspan="4">NLOGSPACE-C</td><td>NP-hard</td><td>PSPACE-C</td></tr>
<tr><td>$TCTL_{\leq,\geq}$</td><td colspan="4">P-complete</td><td colspan="2">PSPACE-C</td></tr>
<tr><td>$TCTL$</td><td>P-compl.</td><td>Δ_2^p</td><td colspan="4">PSPACE-complete</td></tr>
<tr><td>$TCTL_c$</td><td colspan="6">PSPACE-complete</td></tr>
</table>

Conclusion

- No efficient algorithm for "*TCTL*+clocks".
- As soon as there are two clocks, a complexity blow-up occurs for any verification problem.
- For *TCTL*, model-checking may be efficient only for the simple DTGs with durations in $\{0, 1\}$.

- For $TCTL_{\leq,\geq}$, it is possible to have efficient model-checking algorithm for any kind of DTG and 1C-TA.
- The previous result can be extended to probabilistic timed systems (Probabilistic DTG and 1-clock Probabilistic TA).