Computational approaches to analyze complex dynamic systems: model-checking and its applications.

Part 2: Model-checking of timed transitions systems

Morgan Magnin
morgan.magnin@irccyn.ec-nantes.fr
www.morganmagnin.net

NII - *Inoue Laboratory*
École Centrale de Nantes - IRCCyN - *MeForBio team*

Lecture Series - Lecture 2 / NII - 2013/04/03

1. Timed models
   - Timed, Hybrid and Linear Hybrid Automata
   - Time Petri nets
   - Other timed models
   - State space abstractions
2. Formalizing specification through timed modal logics
   - Reminders about linear and branching-time logics
   - Timed extensions of linear logics
   - Timed extensions of branching-time logics
3. Biological application
4. An introduction to control of timed systems
   - Control of discrete-events systems
   - Control of timed systems

Motivations

Objective: formal verification of properties

- Model the system S :
  - Discrete models: finite state automata, Petri nets, ... ⇒ Lecture 1
  - Timed models:
    - timed extensions of finite state automata: timed/hybrid automata ⇒ **Lecture 2**
    - timed extensions of Petri nets: time/stopwatch Petri nets ⇒ Lecture 3
- Formalize the specification φ :
  - Observers
  - Temporal logics: LTL, CTL, ... ⇒ Lecture 1
  - Timed extensions of temporals logics: TCTL, ... ⇒ **Lectures 2** & 3
- Does $S \models \varphi$?

Model-checking algorithms

⇒ State space exploration

Motivations

Objective: formal verification of properties

- Model the system S :
  - Discrete models: finite state automata, Petri nets, ... $\Rightarrow$ Lecture 1
  - Timed models:
    - timed extensions of finite state automata: timed/hybrid automata $\Rightarrow$ **Lecture 2**
    - timed extensions of Petri nets: time/stopwatch Petri nets $\Rightarrow$ Lecture 3
- Formalize the specification φ :
  - Observers
  - Temporal logics: LTL, CTL, ... $\Rightarrow$ Lecture 1
  - Timed extensions of temporals logics: TCTL, ... $\Rightarrow$ **Lectures 2** & 3
- Does $S \models \varphi$?

Model-checking algorithms

$\Rightarrow$ State space exploration

Some major issues

Need for modeling tasks with suspending/resuming features

Expressivity/Decidability compromise to discuss ⇒ **Lectures 2** & 3

State space combinatorial explosion

- Need for symbolic approaches ⇒ **Lectures 2** & 3
- Need for new models and abstracted algorithms ⇒ Lecture 4

Some major issues

Need for modeling tasks with suspending/resuming features

Expressivity/Decidability compromise to discuss ⇒ **Lectures 2** & 3

State space combinatorial explosion

- Need for symbolic approaches ⇒ **Lectures 2** & 3
- Need for new models and abstracted algorithms ⇒ Lecture 4

Today's and next week's issue

Tricky question

Coming from France, why do I need an average 3-4 days period not to be jet-lagged anymore in Tōkyō?

Observation

- Discrete models do not encompass sufficient information to get a thorough description of the gene regulation network behind the circadian clock **w.r.t. time**
- Some related issues:
  - Is it possible to determine the lower limit of the day/night period cycle during which the circadian clock continues to stabilize?
  - Why does the body better support backward phase delay than advance phase delay?
- → On-going modeling project with **biologists** and **computer scientists** (CNRS PEPII funded project CirClock)

Contribution

Scientific challenge

How can we get information about the **production and degradation rates** of a protein in a biological regulatory network?

Objectives of this talk and the forthcoming one

- From discrete model to timed model $\rightarrow$ emphasize on the **progressive enrichment** of model and its drawbacks
- Focus on the introduction of **quantitative** timing information
- Discuss the most appropriate **time semantics** adapted to the model
- Apply the general methodology to practical examples coming from biology

Contribution

Scientific challenge

How can we get information about the **production and degradation rates** of a protein in a biological regulatory network?

Objectives of this talk and the forthcoming one

- From discrete model to timed model $\rightarrow$ emphasize on the **progressive enrichment** of model and its drawbacks
- Focus on the introduction of **quantitative** timing information
- Discuss the most appropriate **time semantics** adapted to the model
- Apply the general methodology to practical examples coming from biology

Overview

1. Timed models
   - Timed, Hybrid and Linear Hybrid Automata
   - Time Petri nets
   - Other timed models
   - State space abstractions
2. Formalizing specification through timed modal logics
   - Reminders about linear and branching-time logics
   - Timed extensions of linear logics
   - Timed extensions of branching-time logics
3. Biological application
4. An introduction to control of timed systems
   - Control of discrete-events systems
   - Control of timed systems

Discrete-event systems vs timed systems

Discrete-event systems

Focus on the sequence of *observable* events (**chronology**):

t_1 t_2 t_3 t_2 t_1 t_1 t_1 ...

Timed systems

Focus on dated *observable* events (**chronometry**):

(t_1, d_1) (t_2, d_2) (t_3, d_3) (t_2, d_4) (t_1, d_5) (t_1, d_6) (t_1, d_7) ...

with:

- d_1: date at which the first t_1 occurs
- d_2: date at which the first t_2 occurs, ...

Remark: events are **asynchronous**, but dates d_i are authorized to be equal to 0

Semantics of timed systems

Discrete-time semantics vs dense-time semantics

- **Discrete-time** semantics: events occur at integer dates only
- **Dense-time** semantics: events occur at any time

$\Rightarrow$ We will discuss the precise links between dense-time, discretization and discrete-time in Lecture 3.

Timed Automata

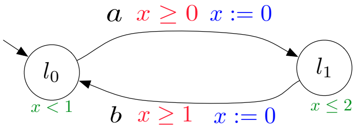


Figure: A **Timed Automaton** (from [CR08])

- State of a TA = (Location, clock valuations)
- The timed language $\mathcal{L}(\mathcal{A})$ of a TA $\mathcal{A}$ is the set of all words (traces) accepted by $\mathcal{A}$.
- The behavioral semantics of a TA $\mathcal{A}$ is a timed transition system $S_{\mathcal{A}}$

Timed Automata

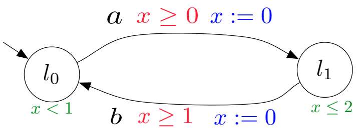


Figure: A **Timed Automaton** (from [CR08])

A path: $(\ell_0, 0) \xrightarrow{0.78} (\ell_0, 0.78) \xrightarrow{a} (\ell_1, 0) \xrightarrow{1.5} (\ell_1, 1.5) \xrightarrow{b} (\ell_0, 0) \cdots$

- State of a TA = (Location, clock valuations)
- The timed language $\mathcal{L}(\mathcal{A})$ of a TA $\mathcal{A}$ is the set of all words (traces) accepted by $\mathcal{A}$.
- The behavioral semantics of a TA $\mathcal{A}$ is a timed transition system $S_{\mathcal{A}}$

Timed Automata

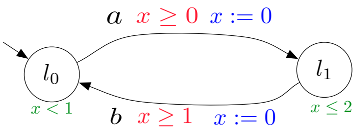


Figure: A **Timed Automaton** (from [CR08])

A path: $(\ell_0, 0) \xrightarrow{0.78} (\ell_0, 0.78) \xrightarrow{a} (\ell_1, 0) \xrightarrow{1.5} (\ell_1, 1.5) \xrightarrow{b} (\ell_0, 0) \cdots$

- State of a TA = (Location, clock valuations)
- The timed language $\mathcal{L}(\mathcal{A})$ of a TA $\mathcal{A}$ is the set of all words (traces) accepted by $\mathcal{A}$.
- The behavioral semantics of a TA $\mathcal{A}$ is a timed transition system $S_{\mathcal{A}}$

Timed Automata

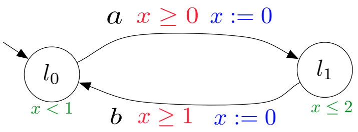


Figure: A **Timed Automaton** (from [CR08])

A path: $(\ell_0, 0) \xrightarrow{0.78} (\ell_0, 0.78) \xrightarrow{a} (\ell_1, 0) \xrightarrow{1.5} (\ell_1, 1.5) \xrightarrow{b} (\ell_0, 0) \cdots$

- State of a TA = (Location, clock valuations)
- The timed language $\mathcal{L}(\mathcal{A})$ of a TA $\mathcal{A}$ is the set of all words (traces) accepted by $\mathcal{A}$.
- The behavioral semantics of a TA $\mathcal{A}$ is a timed transition system $S_{\mathcal{A}}$

Timed Automata [AD91]

Definition

- A finite set of **locations** l
- A finite set of **clocks** v (over $\mathbb{R}$ or $\mathbb{N}$)
- An **invariant function**, mapping each location with a predicate over v
- A finite set of **transitions**
- A *labelling* function
- An **initial location**

Timed Automata [AD91]

About transition

A transition is composed of

- a unique **source** location
- a unique **target** location
- a **guard**, *i.e.* an enabling condition ($g := x \sim c | g \wedge g$, where $\sim \in \{<, \leq, =, \geq, >\}$
- a **label** (that can be used for **synchronization**)
- a subset (potentially empty) of clocks to be **reset**

Timed Automata [AD91]

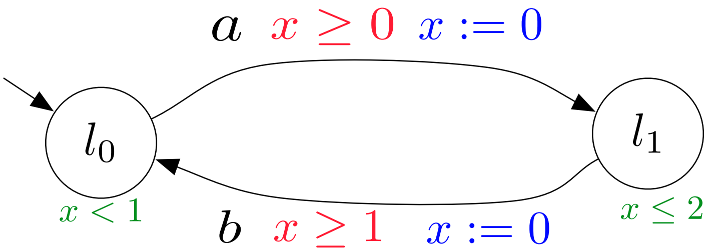


Figure: A Timed Automaton with its invariants, guards and clocks to reset

Semantics of a timed automaton

Definition as a **timed transition system**

- An **action** transition: $(l, v) \xrightarrow{a} (l', v')$ if there exists an a-labelled transition from l to l' such that:
  - v satisfies the guard of the transition
  - $v' = v[r \leftarrow 0]$, with r the set of clocks to be reset
- A **delay** transition: $(l, v) \xrightarrow{\delta(d)} (l, v + d)$, where (l, v) is a state of the timed automaton, and d belongs to the time domain in (l, v)

Hybrid automata [ACH+95]

Key idea

Every location is mapped with a set of **ordinary differential equations** defining the evolution of the variables

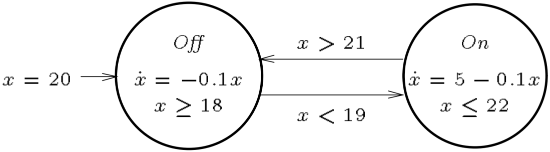


Figure: **Hybrid Automaton** describing a thermostat (from [ACH+95])

Hybrid automata

Definition

- A finite set of **locations** l
- A finite set of **variables** v over $\mathbb{R}$
- A finite set of **initial states** (couples (l, v))
- A finite set of **transitions**
- A **flow function**, mapping each location with with a predicate over v and $\dot{v}$
- An **invariant function**, mapping each location with a predicate over v
- A **jump condition function**, mapping each transition with a predicate over v
- An **initialization condition**, mapping the initial state with a predicate
- *A finite set of synchronization labels*

Linear Hybrid Automata [Hen96]

Key ideas

- The invariant, flow and jump conditions are **boolean combinations of linear equalities**.
- Every location is mapped with **a set of ordinary differential equations** $\sum \dot{x} \leq k$, with $k \in \mathbb{R}$, defining the evolution of the variables.

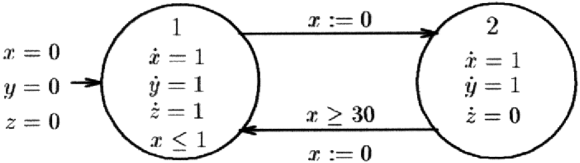


Figure: **Linear Hybrid Automaton** describing a leak in a gas-heating process (from [Hen96])

Petri net - Reminder

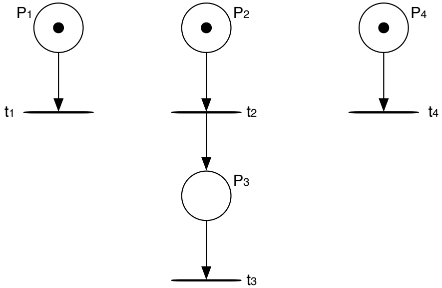


Figure: A Petri net

$$\{P_1, P_2, P_4\}$$

Petri net - Reminder

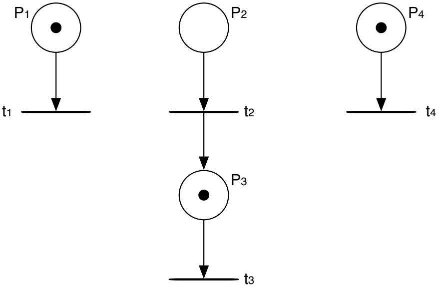


Figure: A Petri net

$$\{P_1, P_2, P_4\} \xrightarrow{t_2} \{P_1, P_3, P_4\} \xrightarrow{t_1} \ldots$$

Time Petri nets - Introduction

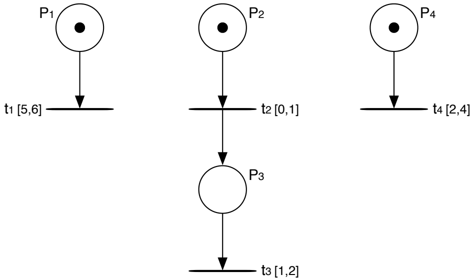


Figure: A time Petri net

$$\begin{array}{l}\{P_1, P_2, P_4\}\\ \theta(t_1) = 0\\ \theta(t_2) = 0\\ \theta(t_4) = 0\end{array} \xrightarrow{0.2} \begin{array}{l}\{P_1, P_2, P_4\}\\ \theta(t_1) = 0.2\\ \theta(t_2) = 0.2\\ \theta(t_4) = 0.2\end{array}$$

Time Petri nets - Introduction

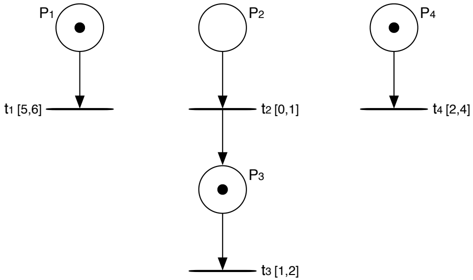


Figure: A time Petri net

$$\begin{array}{l}\{P_1, P_2, P_4\}\\ \theta(t_1) = 0\\ \theta(t_2) = 0\\ \theta(t_4) = 0\end{array} \xrightarrow{0.2} \begin{array}{l}\{P_1, P_2, P_4\}\\ \theta(t_1) = 0.2\\ \theta(t_2) = 0.2\\ \theta(t_4) = 0.2\end{array} \xrightarrow{t_2} \begin{array}{l}\{P_1, P_3, P_4\}\\ \theta(t_1) = 0.2\\ \theta(t_3) = 0\\ \theta(t_4) = 0.2\end{array} \xrightarrow{0.9} \dots$$

Time Petri nets: Definition [Mer74]

A *Time Petri Net* (TPN) is a tuple $\mathcal{T} = (P, T, {}^\bullet(), ()^\bullet, M_0, a, b)$ where :

- $P = \{p_1, p_2, \ldots, p_m\}$ is a non-empty finite set of *places*;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a non-empty finite set of *transitions* $(T \cap P = \emptyset)$;
- ${}^\bullet() \in (\mathbb{N}^P)^T$ is the *backward incidence function*; $()^\bullet \in (\mathbb{N}^P)^T$ is the *forward incidence function*;
- $M_0 \in \mathbb{N}^P$ is the *initial marking* of the net;
- $a \in (\mathbb{Q}^+)^T$ and $b \in (\mathbb{Q}^+ \cup \{\infty\})^T$ are functions giving for each transition respectively its *earliest* and *latest* firing times $(a \leq b)$.

(Un)decidability results

Problem [JLL77]

Reachability, liveness and boundedness problems are **undecidable** for time Petri nets.

Berthomieu et al. proved [BM83]:

Theorem

Reachability and liveness problems are decidable for **bounded** *time Petri nets.*

(Un)decidability results

Problem [JLL77]

Reachability, liveness and boundedness problems are **undecidable** for time Petri nets.

Berthomieu et al. proved [BM83]:

Theorem

Reachability and liveness problems are decidable for **bounded** *time Petri nets.*

About newly enabled transitions

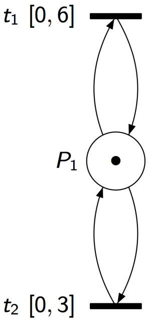


Figure: A time Petri net

We fire t_1

About newly enabled transitions

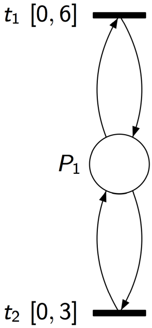


Figure: A time Petri net

We fire t_1

t_1 and t_2 are not enabled by $M - {}^{\bullet}t_1$ (M represents the marking of the net)

About newly enabled transitions

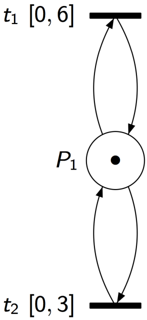


Figure: A time Petri net

We fire t_1

t_1 and t_2 are not enabled by $M - {}^\bullet t_1$

t_1 and t_2 are newly enabled

About newly enabled transitions

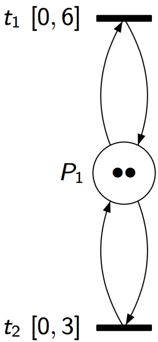


Figure: A time Petri net

We fire t_1

About newly enabled transitions

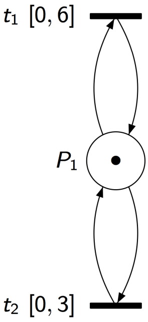


Figure: A time Petri net

We fire t_1

t_1 and t_2 are enabled by $M - {}^\bullet t_1$ but t_1 is the fired transition

About newly enabled transitions

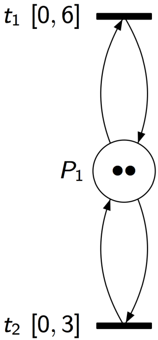


Figure: A time Petri net

We fire t_1

t_1 and t_2 are enabled by $M - {}^\bullet t_1$ but t_1 is the fired transition

t_2 remains enabled, t_1 is newly enabled

Other timed models

A large family of models

- On the thin red **line between decidability and undecidability**
- Variants of **timed automata**:
  - Stopwatch automata: clocks can be stopped in some locations
  - Updatable timed automata: not only clock resets, but also clock updates $x := c$ or $x := y + c$
  - Priced Timed Automata
- Variants of **time Petri nets**:
  - TPNs with self modification
  - Different semantics w.r.t.:
    - time elapsing: strong, weak
    - transition firing: intermediate, atomic

Need for abstractions for timed models

Problem

The state space of a timed transition system is infinite (in general)

$\Rightarrow$ Group states into equivalence classes (abstraction)

Major challenge

What is a relevant abstraction for the model, that **preserves desired properties**?

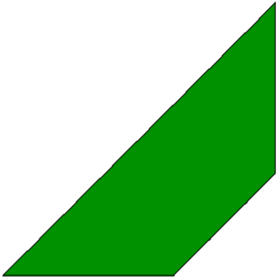
$\Rightarrow$ We will illustrate this abstraction-based approach on one example targeting TPNs.

Abstractions for TPNs

- Infinite state-space $\Rightarrow$ Abstractions
- TPNs: Zone-based simulation graph [GRR06]
- TPNs: State class graph [BD91]

State Class

$$C = \left\{ \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \right.$$



$$\left. \right\}$$

TPNs: Zone (encoded by a Difference Bound Matrix (DBM) $[d_{ij}]_{i,j \in [0..n]}$):

$$\begin{cases} -d_{0i} \leq \theta_i - 0 \leq d_{i0}, \\ \theta_i - \theta_j \leq d_{ij} \end{cases}$$

Basic Algorithm for state space computation

begin
  $Passed = \emptyset$
  $Waiting = \{C_0\}$
  while $Waiting \neq \emptyset$
    $C = \text{pop}(Waiting)$
    $Passed = Passed \cup C$
    for t firable from C
      $C' = \text{AbstractSuccessor}(C, t)$
      if $C' \notin Passed$
        $Waiting = Waiting \cup C'$
      end if
    end for
  end while
end

Computing the state class graph

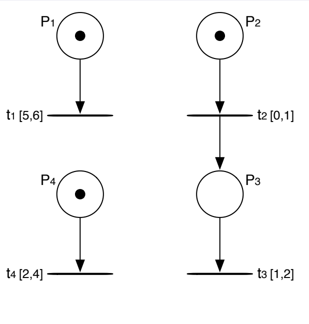
Let $C = (M, D)$ and $D = (A.\Theta \leq B)$. We fire t_f.

- $M' = M - {}^\bullet t_f + t_f{}^\bullet$
- D' is computed by:
  - for all enabled transitions t_i, constrain by $\theta_f \leq \theta_i$
  - for all enabled transitions t_i, $\theta_i' = \theta_i - \theta_f$
  - eliminate variables for disabled transitions (e.g. using Fourier-Motzkin method)
  - add new variables for newly enabled transitions t_i:

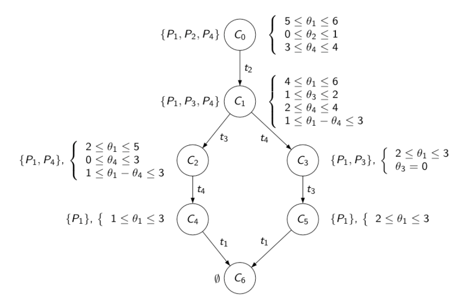
$$\alpha(t_i) \leq \theta_i \leq \beta(t_i)$$

State class graph computation: an example

TPN



State class graph



Overview

1. Timed models
   - Timed, Hybrid and Linear Hybrid Automata
   - Time Petri nets
   - Other timed models
   - State space abstractions
2. Formalizing specification through timed modal logics
   - Reminders about linear and branching-time logics
   - Timed extensions of linear logics
   - Timed extensions of branching-time logics
3. Biological application
4. An introduction to control of timed systems
   - Control of discrete-events systems
   - Control of timed systems

Computation paths vs computation tree - Reminder

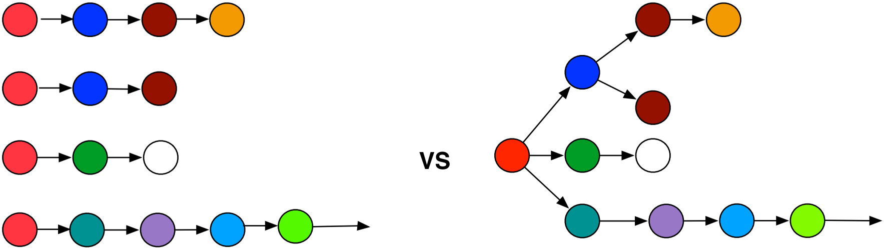


Figure: Execution can be seen as a set of execution paths or as an execution tree

Model-checking formal properties - Reminder

Qualitative properties

- LTL (linear-time properties): *on a given* **path**, $X\varphi$, $\varphi U\psi$ + $G\varphi$, $F\varphi$
- CTL (branching-time properties): *in a given* **state**,
  - $EX\varphi$, $E\varphi U\psi$ + $EG\varphi$, $EF\varphi$
  - $AX\varphi$; $A\varphi U\psi$ + $AG\varphi$, $AF\varphi$
- CTL* (superset including, but not equal, to the union of LTL and CTL)

Model-checking of LTL properties - Reminder

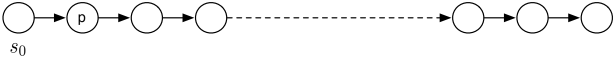


Figure: $s_0 \models Xp$

Model-checking of LTL properties - Reminder

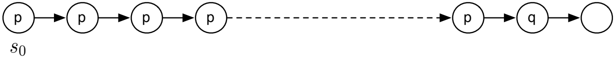


Figure: $s_0 \models pUq$

Model-checking of LTL properties - Reminder

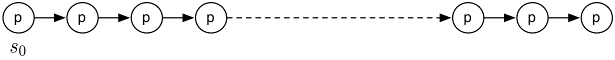


Figure: $s_0 \models Gp$

Model-checking of LTL properties - Reminder

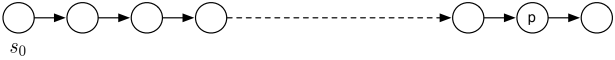


Figure: $s_0 \models Fp$

Model-checking of CTL properties - Reminder

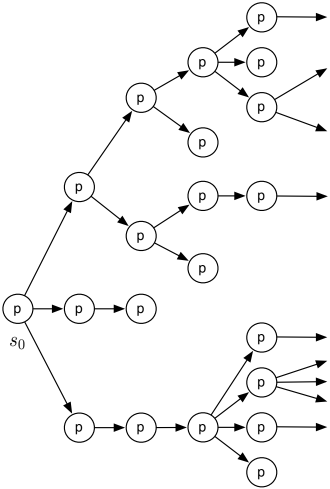


Figure: $s_0 \models AGp$

Model-checking of CTL properties - Reminder

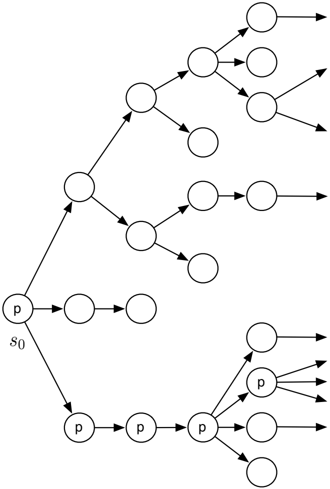


Figure: $s_0 \models EG\mathrm{p}$

Model-checking of CTL properties - Reminder

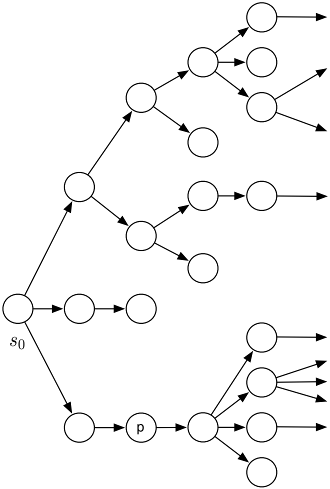


Figure: $s_0 \models EF\,p$

Model-checking of CTL properties - Reminder

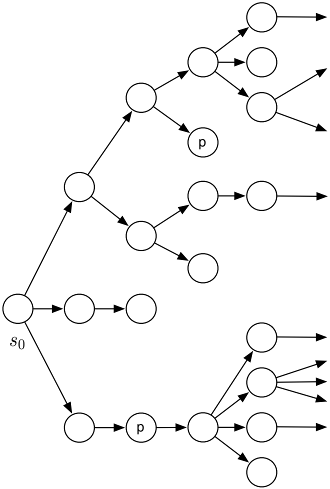


Figure: $s_0 \models EF\,p$

Model-checking of CTL properties - Reminder

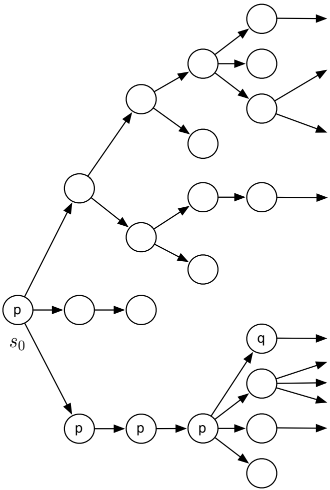


Figure: $s_0 \models$ pEUq

Model-checking of CTL properties - Reminder

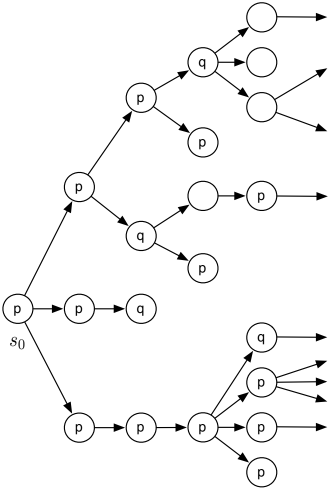


Figure: $s_0 \models pAUq$

Need for timed extensions of modal logics

Quantitative timing properties

How can we formalize a sentence like: "any problem is followed by an alarm **in at most 5 time units**"?

Enrich temporal logics

- "Any problem is followed by an alarm": $AG(\text{problem} \rightarrow AF\text{alarm})$
- Extend temporal logics:
  - Add subscripts to temporal operators, e.g. $AG(\text{problem} \rightarrow AF_{\leq 5}\text{alarm})$
  - Use real clocks to assert timed constraints, e.g. $AG(\text{problem} \rightarrow x \in (x \leq 5 \wedge AF\text{alarm}))$

$\Rightarrow$ **Timed** temporal logics

Timed temporal logics: From a **path** point of view

Extensions of Linear Temporal Logics

- **Metric Temporal Logic** (MTL) [Koy90]
  - Add **subscripts** to temporal operators
  - Example: $G(\text{problem} \rightarrow F_{\leq 5}\text{alarm})$
- **Timed Propositional Temporal Logic** (TPTL) [AH94]
  - Add **real clocks** to formulae
  - Example: $G(\text{problem} \rightarrow x.F \in (x \leq 5 \wedge \text{alarm}))$, where $x.\varphi$ means that clock x is reset at the current position (*i.e.* before evaluating φ).

Remark: next (X) operator from LTL is **removed** (no meaning in dense-time semantics)

Model-checking of MTL properties: An example

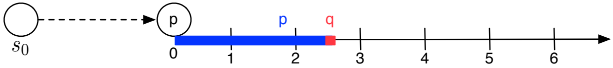


Figure: $s_0 \models p U_{[2;4]} q$

Timed temporal logics: From a **branching-time** point of view [ACD93]

Extensions of CTL*

- TCTL with **subscripts**, e.g. $AG(\text{problem} \rightarrow AF_{\leq 5}\text{alarm})$
- TCTL with **explicit clocks** added to formulae, e.g. $AG(\text{problem} \rightarrow x \in (x \leq 5 \wedge AF\text{alarm}))$

Remark: next (X) operator from CTL* is **removed** (no meaning in dense-time semantics)

Model-checking of TCTL properties: An example

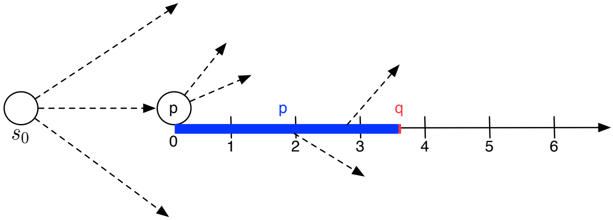


Figure: $s_0 \models E(\mathrm{p} U_{[2;4]} \mathrm{q})$

Timed temporal logics: Expressiveness results [BCM05]

Subscripts vs explicit clocks

- TPTL has been proven to be **strictly more expressive** than MTL (e.g. $x.F(a \wedge x \leq 1 \wedge G(x \leq 1 \Rightarrow \neg b)))$
- TCTL with explicit clocks has been proven to be **strictly more expressive** than TCTL with subscripts.

Timed temporal logics

Quantitative timing properties

A TCTL formula:

$$\varphi := ap \mid \neg ap \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid A\varphi U_I \varphi \mid E\varphi U_I \varphi$$

with:

- *ap* an atomic assertion
- I an interval from $\mathbb{R}^+$ with **integer bounds** s.t. $[n, m]$, $[n, m[$, $]n, m]$, $]n, m[$, or $[m, \infty[$, $n, m \in \mathbb{N}$

Some additional TCTL examples

Bounded liveness/response [DT98]

- "Whenever a property p becomes true, q must be true within n seconds" ($n \in \mathbb{N}$)
- $AG(p \Rightarrow AF_{[0,n]} q)$
- Denoted $p \rightarrow_{[0,n]} q$ in most model-checkers

Decidability results w.r.t. model-checking [Alu99]

Following problems are **undecidable**

- Model-checking of timed automata for MTL properties
- Model-checking of TPNs for TCTL properties
- Satisfaction problem for TCTL (TA/TPN)

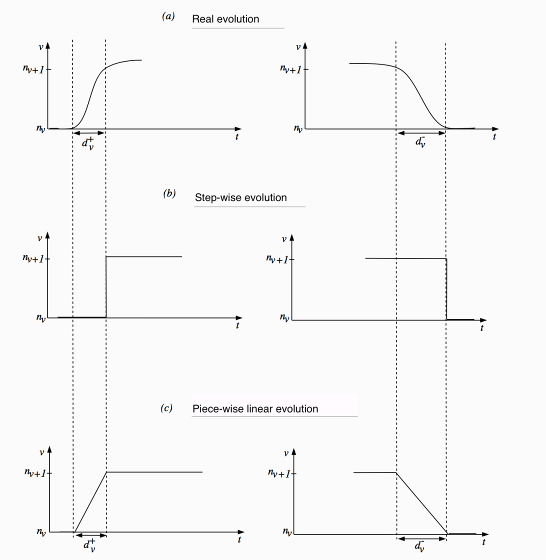
Following problems are **decidable**

- Model-checking of timed automata for TCTL properties
- Model-checking of **bounded** TPNs for a subset (no nesting) of TCTL with subscripts

Overview

1. Timed models
   - Timed, Hybrid and Linear Hybrid Automata
   - Time Petri nets
   - Other timed models
   - State space abstractions
2. Formalizing specification through timed modal logics
   - Reminders about linear and branching-time logics
   - Timed extensions of linear logics
   - Timed extensions of branching-time logics
3. Biological application
4. An introduction to control of timed systems
   - Control of discrete-events systems
   - Control of timed systems

Biological application (from [AR10])



Biological application (from [AR10])

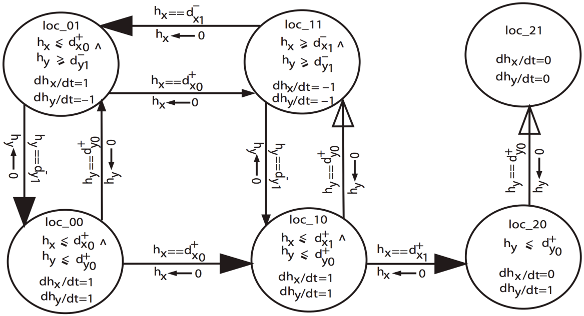


Figure: Linear Hybrid Automaton modeling Pseudomonas Aeruginas

Biological application (from [AR10])

Aim

Identify **cycles** and **attractors**

Methodogy

- Use a **model-checker** on hybrid automata (e.g. HyTech, PHAVer, ...)
- Interpret results thanks to a **parameterized polyhedra library** (e.g. PolyLib)

Overview

1. Timed models
   - Timed, Hybrid and Linear Hybrid Automata
   - Time Petri nets
   - Other timed models
   - State space abstractions
2. Formalizing specification through timed modal logics
   - Reminders about linear and branching-time logics
   - Timed extensions of linear logics
   - Timed extensions of branching-time logics
3. Biological application
4. An introduction to control of timed systems
   - Control of discrete-events systems
   - Control of timed systems

The **control** problem

Real-life system

- Uncontrollable events
- Controllable events
- To be discussed: **Observability** $\Rightarrow$ **full** observability vs **partial** observability

The control problem

Control problem

Does there exist a controller C that guarantees the given properties φ such that $S \parallel C \models \varphi$?

Controller synthesis problem

Can we build a controller C that guarantees the given properties $\varphi \Rightarrow \exists C,\ S \parallel C \models \varphi$?

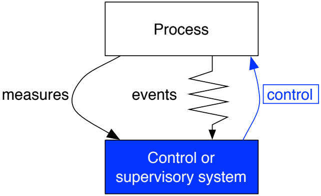


Figure: The control problem

A first approach to **control** problem

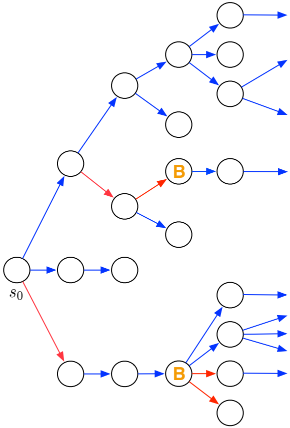


Figure: Branching execution of a model: blue actions stand for controllable actions; red actions stand for uncontrollable ones; B stands for bad states that should be avoided

A first approach to **control** problem

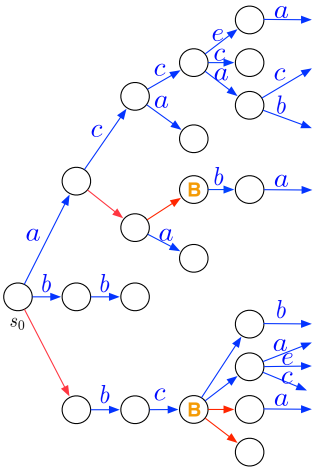


Figure: Blue actions = controllable ones; red actions = uncontrollable ones; B = bad states

A first approach to **control** problem

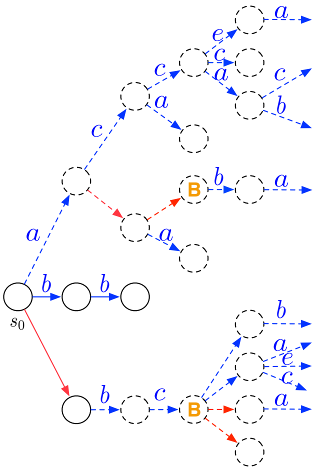


Figure: Blue actions = controllable ones; red actions = uncontrollable ones; B = bad states

A first approach to **control** problem

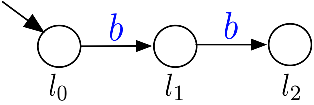


Figure: Supervisor automaton to avoid that the system reach bad states

A first approach to **control** problem

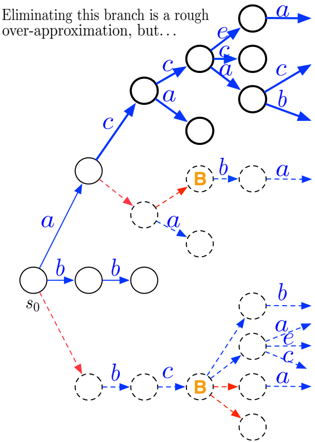


Figure: Blue actions = controllable ones; red actions = uncontrollable ones; B = bad states

A first approach to **control** problem

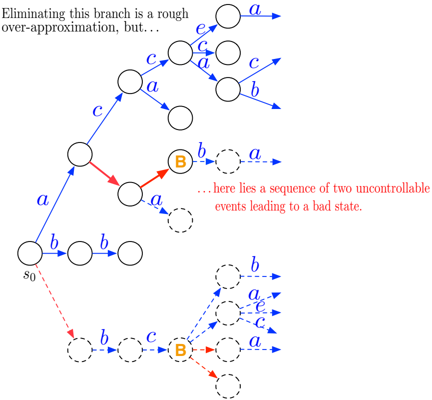


Figure: Blue actions = controllable ones; red actions = uncontrollable ones; B = bad states

Supervisory control theory

Ramadge-Wonham framework [RW89]

- Discrete-events system, modeled as a **finite** automaton with:
  - Uncontrollable events
  - Controllable events
- **Specification**
  - E.g.: Avoid any sequences leading to a state where the property bad is satisfied
  - $\Rightarrow$ specifications as a **language**
- Principle: **Supervisor**, described as a synchronous automaton, observes the events generated by the system and might prevent it from generating a subset of the controllable events

Solving a **control** problem

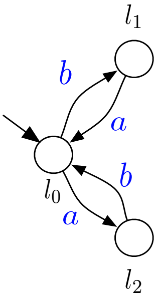


Figure: System S (both a and b are controllable). We would like that only one execution $a.b$ can occur (specification φ). Does there exist a controller C such that $S \parallel C \models \varphi$?

Solving a **control** problem

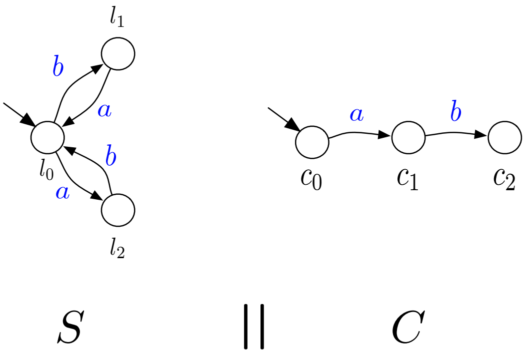


Figure: System S with its supervisor C so that only one execution $a.b$ can occur.

Solving a **control** problem: key idea

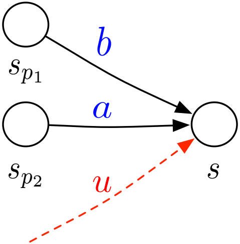


Figure: Basic idea behind the notion of controllable predecessors: I_{p_1} and I_{p_2} might be in the set of controllable predecessors of I

Solving a **control** problem: key idea

Controllable predecessors technique

Let:

- S be the "safe" states, *i.e.* the ones meeting the specification φ
- $\pi(X)$ is the set of **controllable predecessors** of a given state X [MPS95]: $\pi(X)$ is computed as the **greatest fix-point** of $\pi(X) = \pi(X) \cap S$

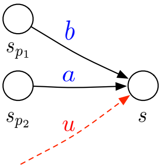


Figure: Basic idea behind the notion of controllable predecessors: l_{p_1} and l_{p_2} might be in the set of controllable predecessors of l

Solving a **control** problem: key idea

Controllable predecessors technique

Let:

- S be the "safe" states, *i.e.* the ones meeting the specification φ
- $\pi(X)$ is the set of **controllable predecessors** of a given state X [MPS95]: $\pi(X)$ is computed as the **greatest fix-point** of $\pi(X) = \pi(X) \cap S$

Control

If the initial state of the automaton belongs to $\pi(S)$, then there exists a supervisor satisfying the specification φ.

Solving a **control** problem: controllable predecessors

Theorems

- For finite automata, the semi-algorithm that computes the set of controllable predecessors **terminates** (because of the finite number of discrete states)
- For Petri nets, the semi-algorithm that computes the set of controllable predecessors **may not terminate**.

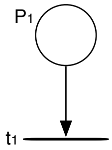


Figure: Example of Petri net for which the computation of the set of **controllable predecessors** will not terminate.

Control as a game (from [CM07])

Definition of the problem

- Open-system = game with two players:
  - Environment plays uncontrollable events
  - Controller plays controllable events
- **Control** objective = **Winning** condition (e.g. avoid bad states)
- Control problem: find a **strategy** (a controller) to win the game

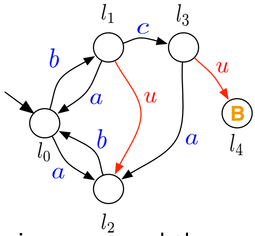


Figure: Game between the environment and the controller: bad state must be avoided (blue actions are controllable; red ones are uncontrollable)

Control as a game (from [CM07])

Related concepts

- **Strategy**: gives, for each finite run, the controllable action to perform
- **Winning strategy**: strategy which generates only runs that leads to a set of states S meeting the specification φ
- **Winning states**: set of states s in which there exists a winning strategy from s (*i.e.* $\pi(S)$)

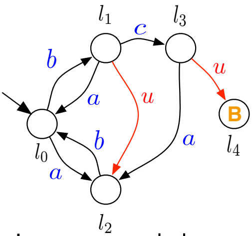


Figure: Game between the environment and the controller: bad state must be avoided (blue actions are controllable; red ones are uncontrollable)

Key issues w.r.t. control as a game problem

Criteria needed to the correct definition of the problem

- **Observability**, again: **full** observability vs **partial** observability
- Type of games:
  - Concurrent games: each opponent can play at any turn
  - Turn-based games: each opponent plays alternatively

Introduction to timed control

Control for timed systems

- Natural extension of the control of discrete-events systems
- A *run* = a succession of **discrete** and **time elapsing** steps
- Extension of the controllable predecessors algorithm

Application to the control problem for timed automata

Control is viewed as a **Timed Game Automaton** [AMPS98]

Control of timed automata

Principle

- **Full observability**: the controller observes both discrete and time-elapsing steps
- **Two options** for the controller:
  - Delay action
  - Perform a controllable action (among the possible ones)
- Define a **strategy**
  - "Wait as long as the system permits"
  - Build the most permissive controller (i.e. the one that restricts the behavior of the environment as little as possible)
  - Towards **optimal** control
- Extension of the controllable predecessors algorithm

Remark: the controller can prevent time to elapse by taking only controllable moves ⇒ **zeno-controllers** (which are usually excluded)

Control of timed automata

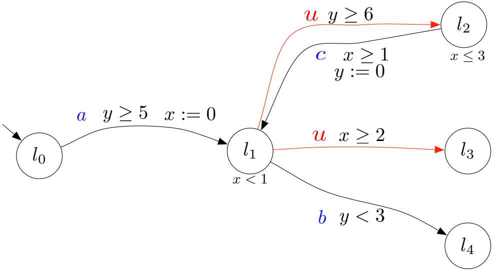


Figure: Timed automaton with controllable and uncontrollable actions

Extension of the controllable predecessors algorithm

Key ideas

- A state s_p is a time controllable predecessor of state s iff, on the time elapsing path between s_p and s, there is no uncontrollable discrete step leading to a bad state s_b
- A **symbolic version** of $\pi(X)$, the set of **controllable predecessors** of a given state X, can be defined [AMPS98]

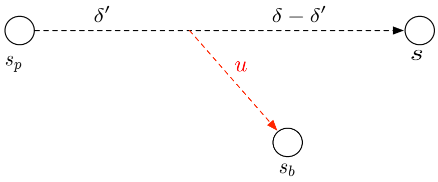


Figure: Time controllable predecessor(s)

Verification vs Optimization

Verification

- **Checks** logical properties
- Implementation: consider the whole state-space of the model

Optimization

- Find **optimal** solutions w.r.t. a set of criteria
- Implementation: cut techniques to avoid non-optimal parts of the state space

Introduction to optimal control

Given a logical property, does there exist an **optimal controller** that guarantees the property, *i.e.* a controller that guarantees the property and optimizes a set of criteria?

Verification vs Optimization

Verification

- **Checks** logical properties
- Implementation: consider the whole state-space of the model

Optimization

- Find **optimal** solutions w.r.t. a set of criteria
- Implementation: cut techniques to avoid non-optimal parts of the state space

Introduction to optimal control

Given a logical property, does there exist an **optimal controller** that guarantees the property, *i.e.* a controller that guarantees the property and optimizes a set of criteria?

Introduction to Optimal Timed Games [BCFL04]

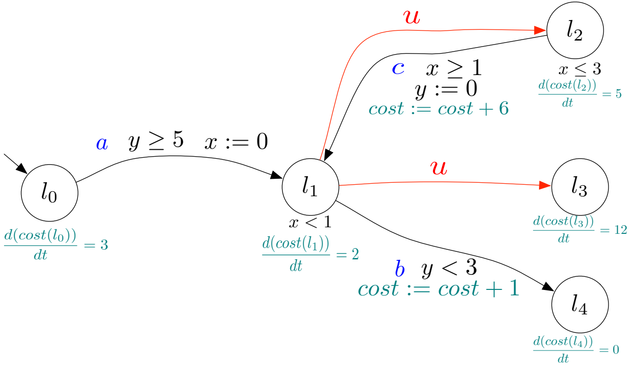


Figure: Game between the environment and the controller: blue actions are controllable; red ones are uncontrollable

Introduction to Optimal Timed Games [BCFL04]

Principle of a reachability timed game

- Does a **best cost** *whatever the environment does* exist? If yes, what is its value?
- Is there a **strategy** to achieve this optimal cost?
- Is this strategy **computable**?

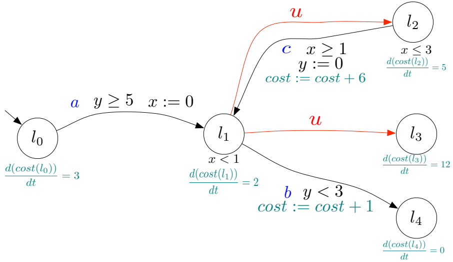


Figure: **Priced timed game automaton** between the environment and the controller: blue actions are controllable; red ones are uncontrollable

Optimal Timed Games [BCFL04]

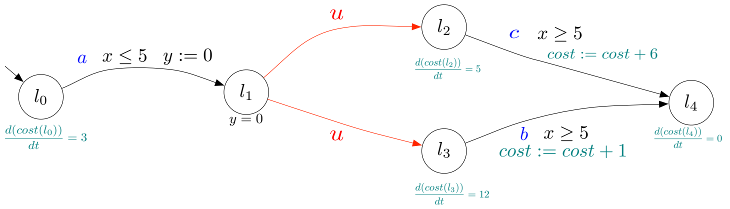


Figure: **Priced timed game automaton** between the environment and the controller: blue actions are controllable; red ones are uncontrollable

Basic illustration of a reachability timed game

- **Best cost** *to reach l_4 whatever the environment does*: $\inf_{0 \leq t \leq 5} max(3t + 5(5-t) + 6; 3t + 12(5-t) + 1) = \frac{11}{9}$, where t represents the time to remain in l_0
- **Strategy** to achieve this optimal cost: wait in l_0 till $t = \frac{11}{9}$, then fire a

Optimal Timed Games [BCFL04]

Problem

- **Priced Timed Game Automaton** (PTGA) = Timed Automaton + **cost function** which associates to each location a cost rate and to each discrete transition a cost
- Usual assumptions on PTGA:
  - **Deterministic w.r.t. controllable actions**
  - **Time-deterministic**: let s, s_1 and s_2 be three states of a timed transition system and $d \in \mathbb{R}$. If $s \xrightarrow{d} s_1$ and $s \xrightarrow{d} s_2$, then $s_1 = s_2$
- Link between **optimal control** for a PTGA and **reachability control** for a Linear Hybrid Game Automaton

Application to scheduling [BLR04]

- Aircraft landing
- Job shop scheduling

Adding timed informations to models

Key factors

- Expressivity: **clocks** vs **stopwatches** vs **variables with more complex dynamics**
- **Asynchronous** events vs **synchronous** events
- **Zenoness**

Timed and hybrid models

Summary

- A wide range of models
- Gaining **expressively** often leads to undecidability
- But **undecidability** is not always incompatible with practical problems

Further work

- Discuss the quantitative **time semantics**
- Discuss the respective **expressivity** of models (timed extensions of automata vs timed extensions of Petri nets)
- Application to practical biological problems

Rajeev Alur, Costas Courcoubetis, and David Dill.
Model-checking in dense real-time.
Information and computation, 104(1):2–34, 1993.

R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine.
The algorithmic analysis of hybrid systems.
THEORETICAL COMPUTER SCIENCE, 138:3–34, 1995.

Rajeev Alur and David L. Dill.
The theory of timed automata.
In *REX Workshop*, pages 45–73, 1991.

Rajeev Alur and Thomas A. Henzinger.
A really temporal logic.
J. ACM, 41(1):181–203, January 1994.

Rajeev Alur.
Timed automata.
Theoretical Computer Science, 126:183–235, 1999.

Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis.
Controller synthesis for timed automata, 1998.

Jamil Ahmad and Olivier Roux.
Invariance kernel of biological regulatory networks.
Int. J. Data Min. Bioinformatics, 4(5):553–570, October 2010.

Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen.

Synthesis of optimal strategies using hytech.
In *In Proc. Games in Design and Verification (GDV?04), ENTCS*, pages 11–31. Elsevier, 2004.

Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey.
On the expressiveness of tptl and mtl.
In *PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON FOUNDATIONS OF SOFTWARE TECHNOLOGY AND THEORETICAL COMPUTER SCIENCE (FSTTCS?05), VOLUME*

3821 OF LECTURE NOTES IN COMPUTER SCIENCE, pages 432–443. Springer, 2005.

B. Berthomieu and M. Diaz.
Modeling and verification of time dependent systems using time Petri nets.
IEEE transactions on software engineering, 17(3):259–273, 1991.

Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen.
Priced timed automata: Algorithms and applications.
In *in International Symposium Formal Methods for Components and Objects (FMCO*, pages 162–182, 2004.

B. Berthomieu and M. Menasche.
An enumerative approach for analyzing time Petri nets.
IFIP Congress Series, 9:41–46, 1983.

Patricia Bouyer, Nicolas Markey, and Ocan Sankur.
Robust reachability in timed automata: A game-based approach.
In *ICALP (2)*, pages 128–140, 2012.

Franck Cassez and Nicolas Markey.
Contrôle des systèmes temporisés.
Actes de l'école d'été ETR'07, 2007.
Nantes.

Franck Cassez and Olivier (H.) Roux.
From Time Petri nets to Timed Automata.
In *Petri Net: Theory and Application*. Edited by Vedran Kordic, I-Tech Publishing, Vienna, Austria, 2008.
ISBN 978-3-902613-12-7.

Conrado Daws and Stavros Tripakis.
Model checking of real-time reachability properties using abstractions.
In *Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '98, pages 313–329, London, UK, UK, 1998. Springer-Verlag.

Guillaume Gardey, Olivier (H.) Roux, and Olivier (F.) Roux.
State space computation and analysis of time Petri nets.

Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems, 6(3):301–320, 2006.

Thomas Henzinger.
The theory of hybrid automata.
In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292, New Brunswick, New Jersey, 1996.

N. D. Jones, L. H. Landweber, and Y. E. Lien.
Complexity of some problems in petri nets.
Theoretical Computer Science 4, pages 277–299, 1977.

Ron Koymans.
Specifying real-time properties with metric temporal logic.
Real-Time Syst., 2(4):255–299, October 1990.

P.M. Merlin.
A study of the recoverability of computing systems.

PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1974.

Oded Maler, Amir Pnueli, and Joseph Sifakis.
On the synthesis of discrete controllers for timed systems.
In *Proc. 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995.

Peter J. Ramadge and W. Murray Wonham.
The control of discrete event systems.
Proceedings of the IEEE, 77(1):81–98, January 1989.