Efficient Reactive Synthesis of MITL Properties

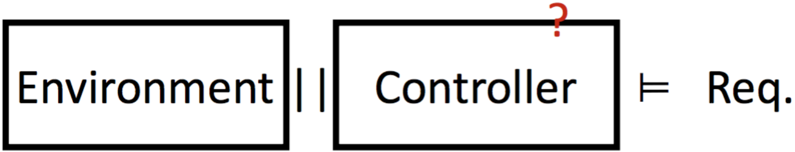
AVeRTS 2015, Bangalore

Benjamin Monmege
Aix-Marseille Université, LIF, CNRS, France

Thomas Brihaye, Morgane Estiévenart (UMONS)
Gilles Geeraerts, Hsi-Ming Ho (ULB), Nathalie Sznajder (UPMC, LIP6)

December 19, 2015

Controller synthesis problem

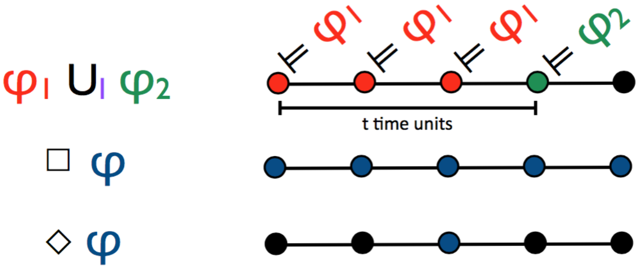


Metric Temporal Logic (MTL)

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathsf{U}_I\, \varphi$$

with $a \in \Sigma$, I interval of $\mathbb{R}^+$ with bounds in $\mathbb{N} \cup \{+\infty\}$

Model of a formula: (in)finite timed word $\sigma = (a_1, t_1)(a_2, t_2)\cdots$ with $a_i \in \Sigma$, (t_i) non-decreasing sequence of time stamps



Synthesis with plant: example of lift

$\Sigma = \Sigma_c \uplus \Sigma_e$

- controller's actions: closing of the doors, moving of the lift...
- environment's actions: pushing of the buttons, uncertainty on responses of the lift...

Synthesis with plant: example of lift

$\Sigma = \Sigma_c \uplus \Sigma_e$

- controller's actions: closing of the doors, moving of the lift...
- environment's actions: pushing of the buttons, uncertainty on responses of the lift...

Pre-existing system to be modelled: number of floors, timing constraints, buttons...

Synthesis with plant: example of lift

$\Sigma = \Sigma_c \uplus \Sigma_e$

- controller's actions: closing of the doors, moving of the lift...
- environment's actions: pushing of the buttons, uncertainty on responses of the lift...

Pre-existing system to be modelled: number of floors, timing constraints, buttons...

PLANT $\mathcal{P}$ = Time-det. timed automaton

Synthesis with plant: example of lift

$\Sigma = \Sigma_c \uplus \Sigma_e$

- controller's actions: closing of the doors, moving of the lift...
- environment's actions: pushing of the buttons, uncertainty on responses of the lift...

Pre-existing system to be modelled: number of floors, timing constraints, buttons...

PLANT $\mathcal{P}$ = Time-det. timed automaton

Specification via MTL: "*lift grants the calls in reasonable time*" $\Box(req \Rightarrow \Diamond_{\leqslant 2}\, grant)$

Synthesis with plant: example of lift

$\Sigma = \Sigma_c \uplus \Sigma_e$

- controller's actions: closing of the doors, moving of the lift...
- environment's actions: pushing of the buttons, uncertainty on responses of the lift...

Pre-existing system to be modelled: number of floors, timing constraints, buttons...

PLANT $\mathcal{P}$ = Time-det. timed automaton

Specification via MTL: "*lift grants the calls in reasonable time*" $\Box(req \Rightarrow \Diamond_{\leqslant 2}\, grant)$

Play: environment and controller propose timed actions

(t, req) $(t', grant)$

Only action(s) with the shortest delay $\min(t, t')$ may be played

Synthesis with plant: example of lift

$\Sigma = \Sigma_c \uplus \Sigma_e$

- controller's actions: closing of the doors, moving of the lift...
- environment's actions: pushing of the buttons, uncertainty on responses of the lift...

Pre-existing system to be modelled: number of floors, timing constraints, buttons...

PLANT $\mathcal{P}$ = Time-det. timed automaton

Specification via MTL: "*lift grants the calls in reasonable time*" $\Box(req \Rightarrow \Diamond_{\leqslant 2}\, grant)$

Play: environment and controller propose timed actions

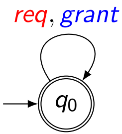
(t, req) $(t', grant)$

Only action(s) with the shortest delay $\min(t, t')$ may be played

Reactive synthesis problem (RS): *find strategy of controller such that every play verifies the specification*

A toy example

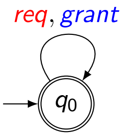
Universal plant $\mathcal{P}$:



Specification: $\square(req \wedge \lozenge_{\geqslant 1}\, req \Rightarrow \lozenge_{=1}\, grant)$

A toy example

Universal plant $\mathcal{P}$:

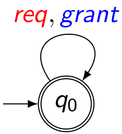


Specification: $\Box(\mathit{req} \wedge \Diamond_{\geqslant 1}\, \mathit{req} \Rightarrow \Diamond_{=1}\, \mathit{grant})$

CONTROLLABLE for RS: controller acknowledges each *req* in chronological order, by playing a *grant* 1 time unit after

A toy example

Universal plant $\mathcal{P}$:



Specification: $\square(\mathit{req} \wedge \Diamond_{\geqslant 1}\, \mathit{req} \Rightarrow \Diamond_{=1}\, \mathit{grant})$

CONTROLLABLE for RS: controller acknowledges each *req* in chronological order, by playing a *grant* 1 time unit after

- left hand side of the specification = fairness condition to give the time to the controller to answer...
- controller requires unbounded memory: unboundedly many events to remember as "*to be granted*" + infinite precision "$= 1$"

Implementable Reactive Synthesis (IRS)

Controller $=$ time-deterministic symbolic transition system $\mathcal{T}$

- set of locations (possibly infinite)
- finite set of clocks
- bounded precision: finite set of possible clock constraints

Implementable Reactive Synthesis (IRS)

Controller = time-deterministic symbolic transition system $\mathcal{T}$

- set of locations (possibly infinite)
- finite set of clocks
- bounded precision: finite set of possible clock constraints

With respect to all possible choices of the environment, $\mathcal{T}$ generates a set of possible plays: smallest set containing the empty play and closed by a *Post* operation...

Implementable Reactive Synthesis (IRS)

Controller $=$ time-deterministic symbolic transition system $\mathcal{T}$

- set of locations (possibly infinite)
- finite set of clocks
- bounded precision: finite set of possible clock constraints

With respect to all possible choices of the environment, $\mathcal{T}$ generates a set of possible plays: smallest set containing the empty play and closed by a *Post* operation...

if $\sigma \cdot (c, T)$ is possible,
and $\mathcal{T}$ may fire (t, grant) currently,
  then $\sigma \cdot (c, T) \cdot (\mathit{grant}, T + t)$ is possible if readable in the plant,
  and $\sigma \cdot (c, T) \cdot (\mathit{req}, T + t')$ is possible if readable in the plant, with $t' \leqslant t$.

Implementable Reactive Synthesis (IRS)

Controller $=$ time-deterministic symbolic transition system $\mathcal{T}$

- set of locations (possibly infinite)
- finite set of clocks
- bounded precision: finite set of possible clock constraints

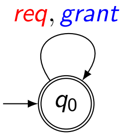
With respect to all possible choices of the environment, $\mathcal{T}$ generates a set of possible plays: smallest set containing the empty play and closed by a *Post* operation...

if $\sigma \cdot (c, T)$ is possible,
and $\mathcal{T}$ may fire (t, grant) currently,
then $\sigma \cdot (c, T) \cdot (\mathit{grant}, T + t)$ is possible if readable in the plant,
and $\sigma \cdot (c, T) \cdot (\mathit{req}, T + t')$ is possible if readable in the plant, with $t' \leqslant t$.

Implementable reactive synthesis problem (IRS): *find a set of clocks X, a precision, and a td STS $\mathcal{T}$ of controller such that every* **possible** *play* **accepted by the plant** *verifies the specification*

A toy example

Universal plant $\mathcal{P}$:

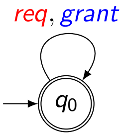


Specification: $\Box(\mathit{req} \wedge \Diamond_{\geqslant 1}\, \mathit{req} \Rightarrow \Diamond_{=1}\, \mathit{grant})$

CONTROLLABLE for RS: controller acknowledges each *req* in chronological order, by playing a *grant* 1 time unit after

A toy example

Universal plant $\mathcal{P}$:

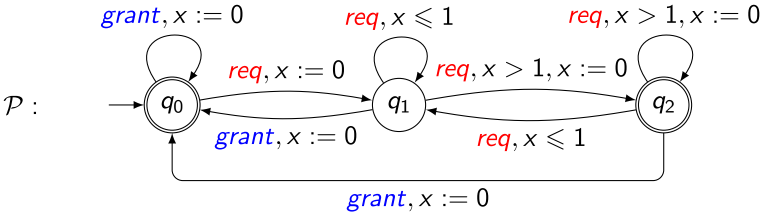


Specification: $\Box(\textit{req} \wedge \Diamond_{\geqslant 1} \textit{req} \Rightarrow \Diamond_{=1} \textit{grant})$

CONTROLLABLE for RS: controller acknowledges each *req* in chronological order, by playing a *grant* 1 time unit after

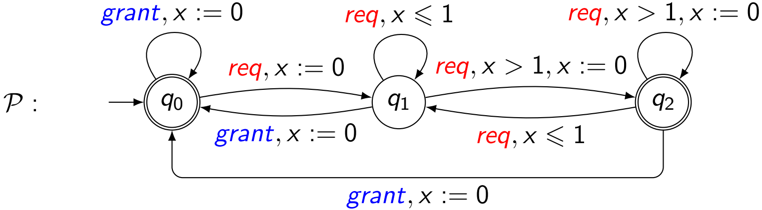
NOT CONTROLLABLE for IRS: requires infinite set of clocks, or infinite precision...

Another example



- every timed word fireable;
- but only certain prefixes are checked against the specification: if at least 1 time unit since the first *req* without *grant* since...

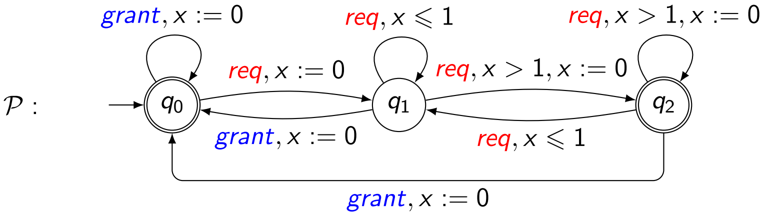
Another example



- every timed word fireable;
- but only certain prefixes are checked against the specification: if at least 1 time unit since the first *req* without *grant* since...

Specification: $\Box(req \Rightarrow \Diamond_{\leqslant 1}\, grant)$

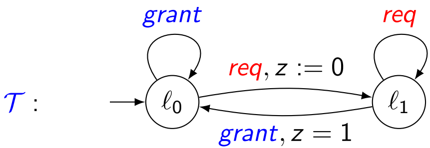
Another example



- every timed word fireable;
- but only certain prefixes are checked against the specification: if at least 1 time unit since the first *req* without *grant* since...

Specification: $\square(\mathit{req} \Rightarrow \lozenge_{\leqslant 1}\, \mathit{grant})$

CONTROLLABLE for IRS: controller only keeps track of the first *req* in the sequence, and proposes to grant it 1 time unit later with a *grant*



Unfortunately...

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

IRS is undecidable for specifications in MTL (over finite words).

Unfortunately...

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

IRS is undecidable for specifications in MTL (over finite words).

Theorem: [D'Souza and Madhusudan, 2002]

IRS is undecidable for specifications given as **timed regular languages**, or **complement of timed regular languages** (over infinite words, and also finite words).

- Reduction of the universality of non-deterministic timed automata

Recovering decidability...

Bounding *a priori* the resources: set of clocks X and precision (m, K) of the controller
Comparisons with maximal guards in $G^{\max}_{m,K}(X)$

$$g ::= \top \mid g \wedge g \mid x < \alpha/m \mid x \leqslant \alpha/m \mid x = \alpha/m \mid x \geqslant \alpha/m \mid x > \alpha/m$$

with $x \in X$, and $0 \leqslant \alpha \leqslant K$.

Recovering decidability...

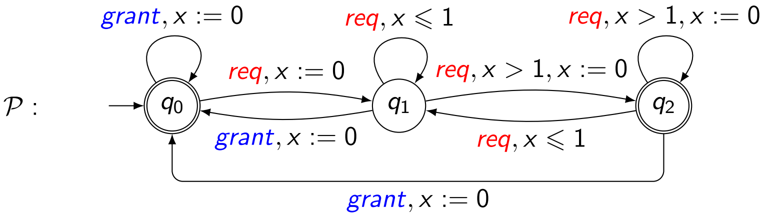
Bounding *a priori* the resources: set of clocks X and precision (m, K) of the controller
Comparisons with maximal guards in $G^{\max}_{m,K}(X)$

$$g ::= \top \mid g \wedge g \mid x < \alpha/m \mid x \leqslant \alpha/m \mid x = \alpha/m \mid x \geqslant \alpha/m \mid x > \alpha/m$$

with $x \in X$, and $0 \leqslant \alpha \leqslant K$.

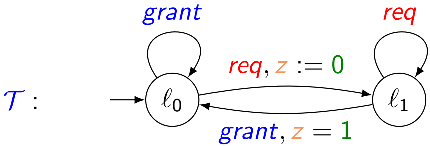
Bounded-resources reactive synthesis problem (BRessRS): *find a td STS $\mathcal{T}$ of controller with* **a given set of clocks X and precision (m, K)** *such that every possible play accepted by the plant verifies the specification*

Example



Specification: $\square(req \Rightarrow \lozenge_{\leqslant 1}\, grant)$

CONTROLLABLE for BRessRS: a single clock $X = \{z\}$, and precision $(m = 1, K = 1)$



Previous results

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

BRessRS is decidable for specifications in MTL (over finite words), with a non-primitive recursive complexity.

Previous results

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

BRessRS is decidable for specifications in MTL (over finite words), with a non-primitive recursive complexity.

Theorem: [D'Souza and Madhusudan, 2002]

BRessRS is decidable for specifications given as **complement of timed regular languages** (over infinite words, and also finite words), with a 2-EXPTIME complexity.

- Build the region automaton, determinise and complement it, and solve a timed game on the synchronous product with the plant and all possible behaviours of the controller

First contribution

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

BRessRS is decidable for specifications in MTL (over finite words), with a non-primitive recursive complexity.

Theorem: [D'Souza and Madhusudan, 2002]

BRessRS is decidable for specifications given as **complement of timed regular languages** (over infinite words, and also finite words), with a 2-EXPTIME complexity.

First contribution

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

BRessRS is decidable for specifications in MTL (over finite words), with a non-primitive recursive complexity.

Theorem: [D'Souza and Madhusudan, 2002]

BRessRS is decidable for specifications given as **complement of timed regular languages** (over infinite words, and also finite words), with a 2-EXPTIME complexity.

Restrict the specification language: MITL

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathsf{U}_I \, \varphi$$

with I **non-singular** interval of $\mathbb{R}^+$ with bounds in $\mathbb{N} \cup \{+\infty\}$

First contribution

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

BRessRS is decidable for specifications in MTL (over finite words), with a non-primitive recursive complexity.

Theorem: [D'Souza and Madhusudan, 2002]

BRessRS is decidable for specifications given as **complement of timed regular languages** (over infinite words, and also finite words), with a 2-EXPTIME complexity.

Restrict the specification language: MITL

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathsf{U}_I \, \varphi$$

with I **non-singular** interval of $\mathbb{R}^+$ with bounds in $\mathbb{N} \cup \{+\infty\}$

Theorem: [Doyen, Geeraerts, Raskin, and Reichert, 2009]

RS is undecidable for specifications in MITL (over infinite words), even without plants.

- Reduction of a lossy 3-counter machine

First contribution

Theorem: [Bouyer, Bozzelli, and Chevalier, 2006]

BRessRS is decidable for specifications in MTL (over finite words), with a non-primitive recursive complexity.

Theorem: [D'Souza and Madhusudan, 2002]

BRessRS is decidable for specifications given as **complement of timed regular languages** (over infinite words, and also finite words), with a 2-EXPTIME complexity.

Our result

Practical algorithm for BRessRS of MITL over finite words, with 3-EXPTIME theoretical complexity.

- Via [D'Souza and Madhusudan, 2002], BRessRS of MITL is 3-EXPTIME
  - build non-deterministic timed automaton equivalent to the negation of the MITL formula...
  - requires the determinisation of the full region automaton!

Alternating automata combine:

From MTL to One-Clock Alternating Timed Automata [Ouaknine and Worrell, 2007]

Alternating automata combine:

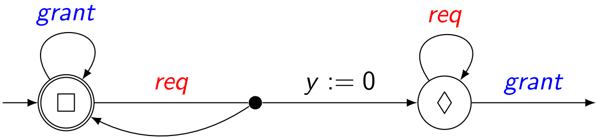
- disjunctive transitions = non-determinism = the suffix of timed word must be accepted from **at least one** of the successor states

Alternating automata combine:

- disjunctive transitions = non-determinism = the suffix of timed word must be accepted from **at least one** of the successor states
- conjunctive transitions = parallelism = the suffix must be accepted from **all** successor states

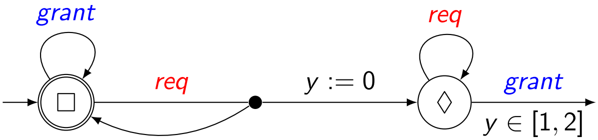
From MTL to OCATA

$$\varphi = \Box(\textit{req} \Rightarrow \Diamond_{[1,2]}\, \textit{grant})$$



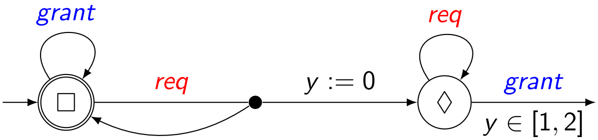
From MTL to OCATA

$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\, \mathit{grant})$$



From MTL to OCATA

$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\, \mathit{grant})$$

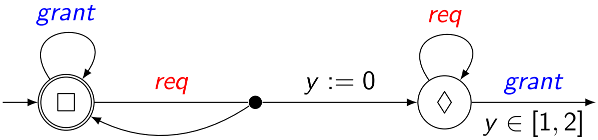


Execution on the timed word $(\mathit{req}, 0.5)(\mathit{req}, 0.6)(\mathit{req}, 1.2)(\mathit{grant}, 2.3)$:

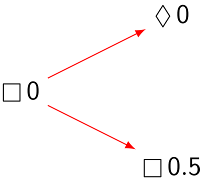
$\Box\, 0$

From MTL to OCATA

$$\varphi = \Box(\textit{req} \Rightarrow \Diamond_{[1,2]}\, \textit{grant})$$

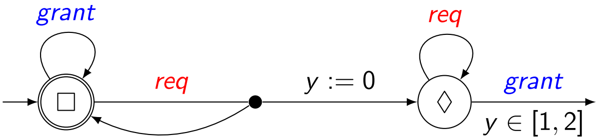


Execution on the timed word $(\textit{req}, 0.5)(\textit{req}, 0.6)(\textit{req}, 1.2)(\textit{grant}, 2.3)$:

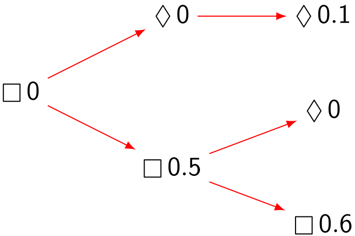


From MTL to OCATA

$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\, \mathit{grant})$$

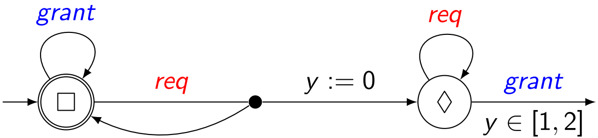


Execution on the timed word $(\mathit{req}, 0.5)(\mathit{req}, 0.6)(\mathit{req}, 1.2)(\mathit{grant}, 2.3)$:

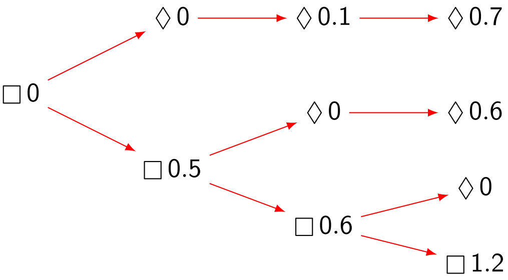


From MTL to OCATA

$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\, \mathit{grant})$$

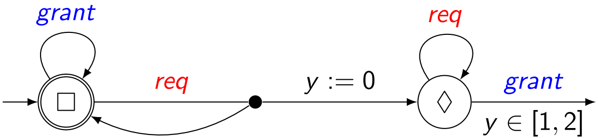


Execution on the timed word $(\mathit{req}, 0.5)(\mathit{req}, 0.6)(\mathit{req}, 1.2)(\mathit{grant}, 2.3)$:

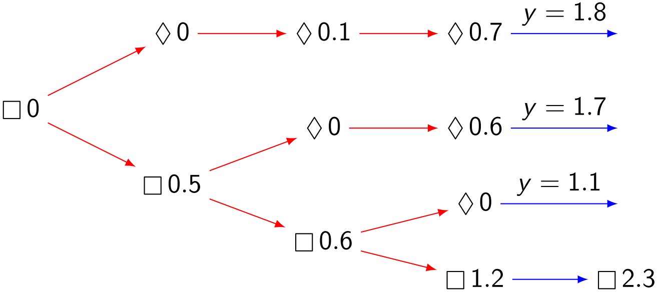


From MTL to OCATA

$$\varphi = \Box(\textit{req} \Rightarrow \Diamond_{[1,2]}\, \textit{grant})$$



Execution on the timed word $(\textit{req}, 0.5)(\textit{req}, 0.6)(\textit{req}, 1.2)(\textit{grant}, 2.3)$:



From MTL to OCATA

- Translation from MTL to OCATA is structural: the OCATA has **one state per subformula**
- **One clock** in the **syntax** of the automaton but... many clocks in the semantics!

Bounded-Ressources Reactive Synthesis for MTL

- Plant: $\mathcal{P}$, Specification: φ in MTL, Ressources: (X, m, K)
- **Convert** the MTL formula $\neg\varphi$ into an **OCATA** $\mathcal{A}$
- **Cast** the control problem into a **timed game** played on a tree
- The tree **unravels** the execution of the parallel composition of: the plant $\mathcal{P}$, the OCATA $\mathcal{A}$, the controller $\mathcal{T}$

Bounded-Ressources Reactive Synthesis for MTL

- Plant: $\mathcal{P}$, Specification: φ in MTL, Ressources: (X, m, K)
- **Convert** the MTL formula $\neg\varphi$ into an **OCATA** $\mathcal{A}$
- **Cast** the control problem into a **timed game** played on a tree
- The tree **unravels** the execution of the parallel composition of: the plant $\mathcal{P}$, the OCATA $\mathcal{A}$, the controller $\mathcal{T}$
- Branching corresponds to the **possible actions**

Bounded-Ressources Reactive Synthesis for MTL

- Plant: $\mathcal{P}$, Specification: φ in MTL, Ressources: (X, m, K)
- **Convert** the MTL formula $\neg\varphi$ into an **OCATA** $\mathcal{A}$
- **Cast** the control problem into a **timed game** played on a tree
- The tree **unravels** the execution of the parallel composition of: the plant $\mathcal{P}$, the OCATA $\mathcal{A}$, the controller $\mathcal{T}$
- Branching corresponds to the **possible actions**
- Labels of the nodes in the tree: finite abstraction of the timed configurations of plant, OCATA and controller
  - q: (unique) location of the (deterministic) plant

$$\boxed{(q, \{H_1, H_2, \ldots, H_n\})}$$

Bounded-Ressources Reactive Synthesis for MTL

- Plant: $\mathcal{P}$, Specification: φ in MTL, Ressources: (X, m, K)
- **Convert** the MTL formula $\neg\varphi$ into an **OCATA** $\mathcal{A}$
- **Cast** the control problem into a **timed game** played on a tree
- The tree **unravels** the execution of the parallel composition of: the plant $\mathcal{P}$, the OCATA $\mathcal{A}$, the controller $\mathcal{T}$
- Branching corresponds to the **possible actions**
- Labels of the nodes in the tree: finite abstraction of the timed configurations of plant, OCATA and controller
  - q: (unique) location of the (deterministic) plant
  - each $H_i = \lambda_1 \cdots \lambda_k$: finite words of subsets of letters (one for each fractional part of the clocks)

$$\boxed{(q, \{H_1, H_2, \ldots, H_n\})}$$

Bounded-Ressources Reactive Synthesis for MTL

- Plant: $\mathcal{P}$, Specification: φ in MTL, Ressources: (X, m, K)
- **Convert** the MTL formula $\neg\varphi$ into an **OCATA** $\mathcal{A}$
- **Cast** the control problem into a **timed game** played on a tree
- The tree **unravels** the execution of the parallel composition of: the plant $\mathcal{P}$, the OCATA $\mathcal{A}$, the controller $\mathcal{T}$
- Branching corresponds to the **possible actions**
- Labels of the nodes in the tree: finite abstraction of the timed configurations of plant, OCATA and controller
  - q: (unique) location of the (deterministic) plant
  - each $H_i = \lambda_1 \cdots \lambda_k$: finite words of subsets of letters (one for each fractional part of the clocks)
  - each $\lambda_i \subseteq 2^{(X_{\mathcal{P}} \cup X \cup Q_{\mathcal{A}}) \times \mathrm{REG}_{m,K}}$: region associated to all clocks

$$\boxed{(q, \{H_1, H_2, \ldots, H_n\})}$$

Bounded-Ressources Reactive Synthesis for MTL

- Action (a, g, R)
  - a: letter of $\Sigma_c \cup \Sigma_e$
  - g: guard over clocks of X and $X_{\mathcal{P}}$
  - R: resets of clocks of X

$$\begin{tikzcd} & \boxed{(q, \{H_1, H_2, \ldots, H_n\})} \arrow[dl, "{a, g, R}"] \\ \boxed{(q', \{H'_1, H'_2, \ldots, H'_{n'}\})} & \end{tikzcd}$$

Bounded-Ressources Reactive Synthesis for MTL

- Action (a, g, R)
  - a: letter of $\Sigma_c \cup \Sigma_e$
  - g: guard over clocks of X and $X_{\mathcal{P}}$
  - R: resets of clocks of X

$$\boxed{(q, \{H_1, H_2, \ldots, H_n\})} \xrightarrow{a, g, R} \boxed{(q', \{H'_1, H'_2, \ldots, H'_{n'}\})}$$

- Finite abstraction is a (time-abstract) bisimulation

Bounded-Ressources Reactive Synthesis for MTL

- Action (a, g, R)
  - a: letter of $\Sigma_c \cup \Sigma_e$
  - g: guard over clocks of X and $X_{\mathcal{P}}$
  - R: resets of clocks of X

$$\begin{tikzcd} \boxed{(q, \{H_1, H_2, \ldots, H_n\})} \arrow[d, "a, g, R"] \\ \boxed{(q', \{H'_1, H'_2, \ldots, H'_{n'}\})} \end{tikzcd}$$

- Finite abstraction is a (time-abstract) bisimulation
- Sufficient to detect when a bad configuration has been reached: one H_i contains only accepting locations of the OCATA $\mathcal{A}$ ($\equiv \neg\varphi$)

Bounded-Ressources Reactive Synthesis for MTL

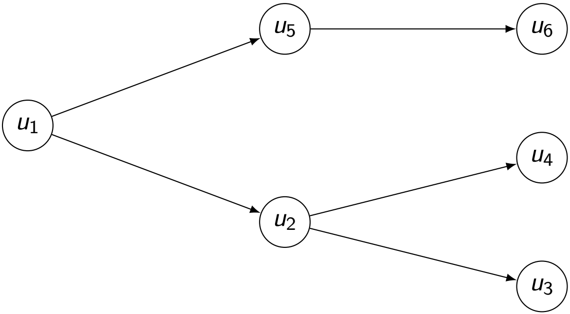
- Action (a, g, R)
  - a: letter of $\Sigma_c \cup \Sigma_e$
  - g: guard over clocks of X and $X_{\mathcal{P}}$
  - R: resets of clocks of X

$$
\begin{tikzcd}
 & \boxed{(q, \{H_1, H_2, \ldots, H_n\})} \arrow[dl, "{a, g, R}"] \\
\boxed{(q', \{H'_1, H'_2, \ldots, H'_{n'}\})} &
\end{tikzcd}
$$

- Finite abstraction is a (time-abstract) bisimulation
- Sufficient to detect when a bad configuration has been reached: one H_i contains only accepting locations of the OCATA $\mathcal{A}$ ($\equiv \neg\varphi$)
- If tree finite and winning strategy: we have a (finite) controller $\mathcal{T}$

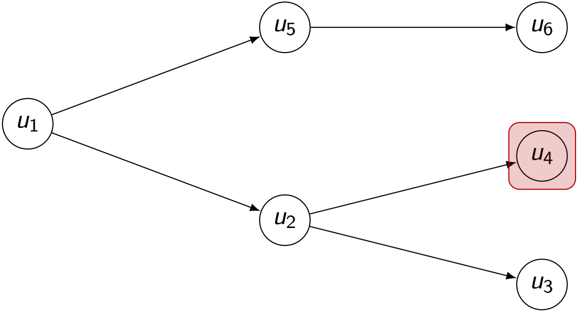
Make the tree finite

For MTL specifications [Bouyer, Bozzelli, and Chevalier, 2006]: stop the computation with a well-quasi order $\sqsubseteq$ on the labels of the nodes



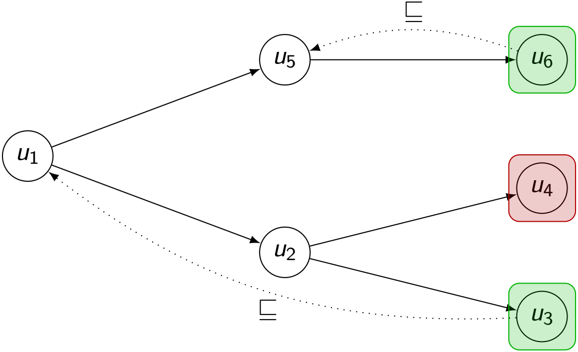
Make the tree finite

For MTL specifications [Bouyer, Bozzelli, and Chevalier, 2006]: stop the computation with a well-quasi order $\sqsubseteq$ on the labels of the nodes



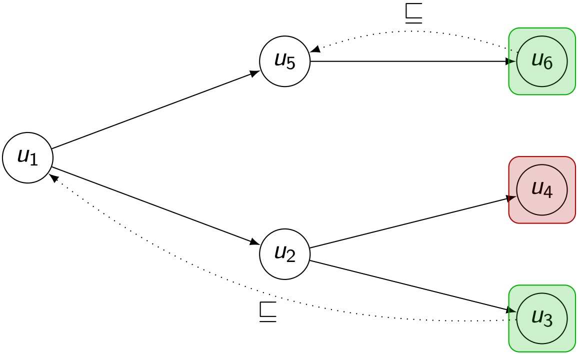
Make the tree finite

For MTL specifications [Bouyer, Bozzelli, and Chevalier, 2006]: stop the computation with a well-quasi order $\sqsubseteq$ on the labels of the nodes



Make the tree finite

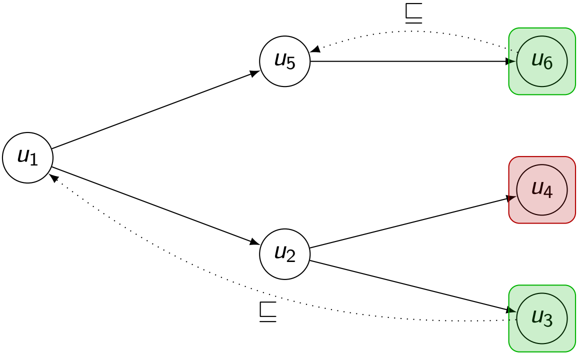
For MTL specifications [Bouyer, Bozzelli, and Chevalier, 2006]: stop the computation with a well-quasi order $\sqsubseteq$ on the labels of the nodes



- Correctness: this finite tree is **sufficient** to answer the realisability problem

Make the tree finite

For MTL specifications [Bouyer, Bozzelli, and Chevalier, 2006]: stop the computation with a well-quasi order $\sqsubseteq$ on the labels of the nodes



- Correctness: this finite tree is **sufficient** to answer the realisability problem
- Complexity: **non-primitive recursive** due to well-quasi orderings

For MITL: interval semantics for OCATA

New semantics for OCATA [Brihaye, Estiévenart, and Geeraerts, 2013]:

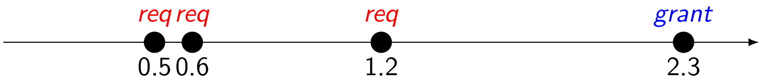
- allows one to **bound the number of clock copies**
- **sufficiently expressive** for MITL

For MITL: interval semantics for OCATA

New semantics for OCATA [Brihaye, Estiévenart, and Geeraerts, 2013]:

- allows one to **bound the number of clock copies**
- **sufficiently expressive** for MITL

$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\, \mathit{grant})$$

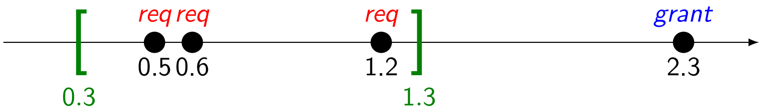


For MITL: interval semantics for OCATA

New semantics for OCATA [Brihaye, Estiévenart, and Geeraerts, 2013]:

- allows one to **bound the number of clock copies**
- **sufficiently expressive** for MITL

$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\, \mathit{grant})$$

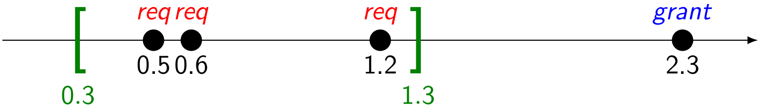


For MITL: interval semantics for OCATA

New semantics for OCATA [Brihaye, Estiévenart, and Geeraerts, 2013]:

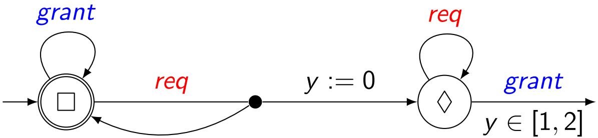
- allows one to **bound the number of clock copies**
- **sufficiently expressive** for MITL

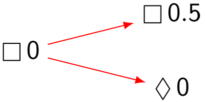
$$\varphi = \Box(\mathit{req} \Rightarrow \Diamond_{[1,2]}\,\mathit{grant})$$

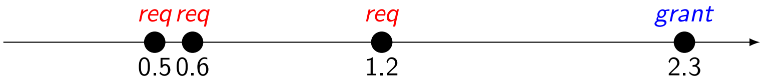


To check that this timed word satisfies φ, we **do not need to remember** the exact timestamp of each *req*

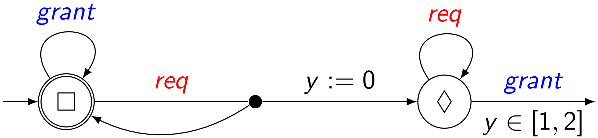
Example run with the interval semantics

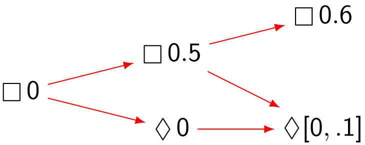


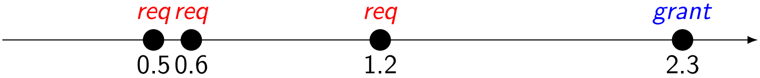




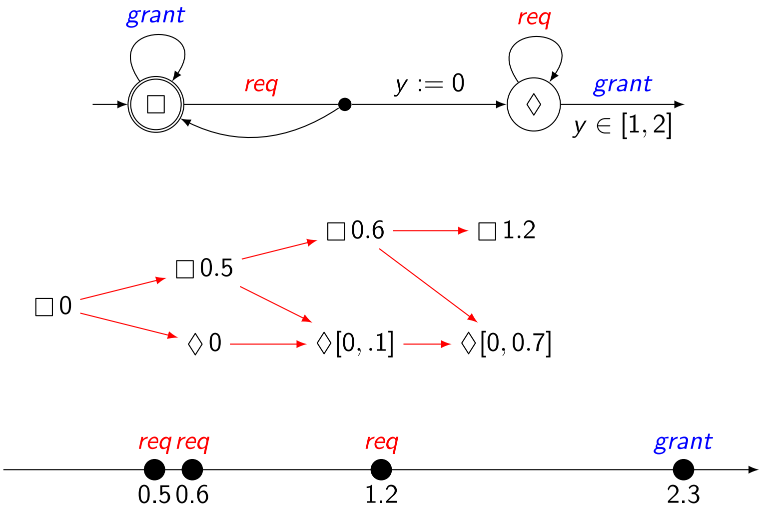
Example run with the interval semantics



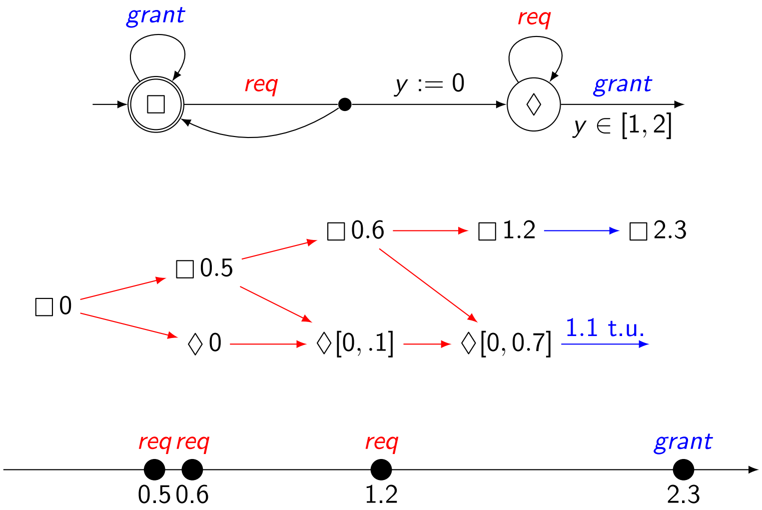




Example run with the interval semantics



Example run with the interval semantics



Control for MITL specification

Tree construction of [Bouyer, Bozzelli, and Chevalier, 2006]

- Finite abstraction making use of interval semantics for OCATA

Control for MITL specification

Tree construction of [Bouyer, Bozzelli, and Chevalier, 2006]

- Finite abstraction making use of interval semantics for OCATA
- Tree is always finite! **No need for well-quasi orderings**

Control for MITL specification

Tree construction of [Bouyer, Bozzelli, and Chevalier, 2006]

- Finite abstraction making use of interval semantics for OCATA
- Tree is always finite! **No need for well-quasi orderings**

Theorem:

3-EXPTIME complexity by a tight count on the number of necessary clock copies [Brihaye, Estiévenart, and Geeraerts, 2013]

Control for MITL specification

Tree construction of [Bouyer, Bozzelli, and Chevalier, 2006]

- Finite abstraction making use of interval semantics for OCATA
- Tree is always finite! **No need for well-quasi orderings**

Theorem:

3-EXPTIME complexity by a tight count on the number of necessary clock copies [Brihaye, Estiévenart, and Geeraerts, 2013]

- Same complexity as in [D'Souza and Madhusudan, 2002]...
- but **on-the-fly** algorithm

Control for MITL specification

Tree construction of [Bouyer, Bozzelli, and Chevalier, 2006]

- Finite abstraction making use of interval semantics for OCATA
- Tree is always finite! **No need for well-quasi orderings**

Theorem:

3-EXPTIME complexity by a tight count on the number of necessary clock copies [Brihaye, Estiévenart, and Geeraerts, 2013]

- Same complexity as in [D'Souza and Madhusudan, 2002]...
- but **on-the-fly** algorithm

- **Zone**-based implementation doable: future work!
- **Heuristics**

Heuristics

- **Antichains**:
  - in a label $(q, \{H_1, \ldots, H_n\})$, do not keep H_i such that $H_i \leqslant H_j$

Heuristics

- **Antichains**:
  - in a label $(q, \{H_1, \ldots, H_n\})$, do not keep H_i such that $H_i \leqslant H_j$
  - **Reduce** the size of the node's labels, and the computation cost

Heuristics

- **Antichains**:
  - in a label $(q, \{H_1, \ldots, H_n\})$, do not keep H_i such that $H_i \leqslant H_j$
  - **Reduce** the size of the node's labels, and the computation cost
- Stop branches earlier using well-quasi-order $\sqsubseteq$ of [Bouyer, Bozzelli, and Chevalier, 2006]:
  - still valid, even though we do not use it for termination

What else?

Bounded-ress. reactive synthesis

- Decidable in 3-EXPTIME for complement of timed automata
- Undecidable for nd timed automata
- Decidable in non-primitive recursive complexity for MTL
- On-the-fly algorithm for MITL

Implementable reactive synthesis

- Undecidable for complement of timed automata
- Undecidable for nd timed automata
- Undecidable for MTL
- **For MITL??**

What else?

Bounded-ress. reactive synthesis

- Decidable in 3-EXPTIME for complement of timed automata
- Undecidable for nd timed automata
- Decidable in non-primitive recursive complexity for MTL
- On-the-fly algorithm for MITL

Implementable reactive synthesis

- Undecidable for complement of timed automata
- Undecidable for nd timed automata
- Undecidable for MTL
- **For MITL??**

Trying to push further the **undecidability boundaries**?

Undecidability of IRS for MTL [Bouyer, Bozzelli, and Chevalier, 2006]

Reduction of the halting problem of a **deterministic channel machine** with

- single halting state s_{halt} with no outgoing transition
- no cycle with only **write actions** m!
- if the unique (maximal) path from initial state is infinite, then the size of the channel is unbounded

Undecidability of IRS for MTL [Bouyer, Bozzelli, and Chevalier, 2006]

Reduction of the halting problem of a **deterministic channel machine** with

- single halting state s_{halt} with no outgoing transition
- no cycle with only **write actions** $m!$
- if the unique (maximal) path from initial state is infinite, then the size of the channel is unbounded

Encoding of an execution: $(a_1, t_1)(a_2, t_2)\cdots$ over $\Sigma_C = \{m?, m!, ...\}$:

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
2. no two actions on the same time: $t_i < t_{i+1}$
3. every $m!$ action matched by an $m?$ action 1 t.u. later
4. every $m?$ action matched by an $m!$ action 1 t.u. before

Undecidability of IRS for MTL [Bouyer, Bozzelli, and Chevalier, 2006]

Reduction of the halting problem of a **deterministic channel machine** with

- single halting state s_{halt} with no outgoing transition
- no cycle with only **write actions** $m!$
- if the unique (maximal) path from initial state is infinite, then the size of the channel is unbounded

Encoding of an execution: $(a_1, t_1)(a_2, t_2)\cdots$ over $\Sigma_C = \{m?, m!, ...\}$:

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant
2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant
3. every $m!$ action matched by an $m?$ action 1 t.u. later
4. every $m?$ action matched by an $m!$ action 1 t.u. before

Undecidability of IRS for MTL [Bouyer, Bozzelli, and Chevalier, 2006]

Reduction of the halting problem of a **deterministic channel machine** with

- single halting state s_{halt} with no outgoing transition
- no cycle with only **write actions** $m!$
- if the unique (maximal) path from initial state is infinite, then the size of the channel is unbounded

Encoding of an execution: $(a_1, t_1)(a_2, t_2) \cdots$ over $\Sigma_C = \{m?, m!, ...\}$:

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant
2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant
3. every $m!$ action matched by an $m?$ action 1 t.u. later
   - MTL formula $\varphi = \Box(m! \wedge \Diamond_{\geqslant 1} \Sigma_C \Rightarrow \Diamond_{=1} m?)$
4. every $m?$ action matched by an $m!$ action 1 t.u. before

Role of the environment

4. every $m?$ action matched by an $m!$ action 1 t.u. before

$\Sigma_E = \{Check, Nil\}$

Plant $\mathcal{P}$: ensures a turn-based behaviour, Environment plays after 0 t.u., *Check* action is played only once...

Role of the environment

4. every $m?$ action matched by an $m!$ action 1 t.u. before

$\Sigma_E = \{Check, Nil\}$

Plant $\mathcal{P}$: ensures a turn-based behaviour, Environment plays after 0 t.u., *Check* action is played only once...

Then, formula $\varphi' = \Diamond(m? \wedge \Diamond_{=0} Check) \Rightarrow \Diamond(m! \wedge \Diamond_{=1} Check)$ checks 4.

Role of the environment

4. every $m?$ action matched by an $m!$ action 1 t.u. before

$\Sigma_E = \{Check, Nil\}$

Plant $\mathcal{P}$: ensures a turn-based behaviour, Environment plays after 0 t.u., *Check* action is played only once...

Then, formula $\varphi' = \Diamond(m? \wedge \Diamond_{=0}\, Check) \Rightarrow \Diamond(m! \wedge \Diamond_{=1}\, Check)$ checks 4.

Theorem:

There exists a controller $\mathcal{T}$ if and only if the channel machine halts.

Role of the environment

4. every $m?$ action matched by an $m!$ action 1 t.u. before

$\Sigma_E = \{Check, Nil\}$

Plant $\mathcal{P}$: ensures a turn-based behaviour, Environment plays after 0 t.u., *Check* action is played only once...

Then, formula $\varphi' = \Diamond(m? \wedge \Diamond_{=0}\, Check) \Rightarrow \Diamond(m! \wedge \Diamond_{=1}\, Check)$ checks 4.

Theorem:

There exists a controller $\mathcal{T}$ if and only if the channel machine halts.

$\Leftarrow$: construct a controller that plays a halting execution

- either with 1 clock, but $m = K =$ maximal capacity of the channel
- or with $m = K = 1$, but as many clocks as the maximal capacity

Role of the environment

4. every $m?$ action matched by an $m!$ action 1 t.u. before

$\Sigma_E = \{Check, Nil\}$

Plant $\mathcal{P}$: ensures a turn-based behaviour, Environment plays after 0 t.u., *Check* action is played only once...

Then, formula $\varphi' = \Diamond(m? \wedge \Diamond_{=0}\, Check) \Rightarrow \Diamond(m! \wedge \Diamond_{=1}\, Check)$ checks 4.

Theorem:

There exists a controller $\mathcal{T}$ if and only if the channel machine halts.

$\Leftarrow$: construct a controller that plays a halting execution

- either with 1 clock, but $m = K =$ maximal capacity of the channel
- or with $m = K = 1$, but as many clocks as the maximal capacity

$\Rightarrow$: if machine does not halt, a controller would need to cheat or to play an infinite computation that requires infinite number of clocks (because of the unboundedness of the channel)

Adaptation of proof for MITL

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant
2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant
3. every $m!$ action, is matched by an $m?$ action 1 t.u. later
   - MTL formula $\varphi = \Box(m! \wedge \Diamond_{\geqslant 1} \Sigma_C \Rightarrow \Diamond_{=1} m?)$
4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MTL formula $\varphi' = \Diamond(m? \wedge \Diamond_{=0} \mathit{Check}) \Rightarrow \Diamond(m! \wedge \Diamond_{=1} \mathit{Check})$

Adaptation of proof for MITL

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant

2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant

3. every $m!$ action, is matched by an $m?$ action 1 t.u. later
   - MTL formula $\varphi = \Box(m! \wedge \Diamond_{\geqslant 1} \Sigma_C \Rightarrow \Diamond_{=1} m?)$

4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MITL formula
     $\varphi' = \Diamond\big(m? \wedge (m? \,\mathsf{U}\, Check)\big) \Rightarrow \Diamond(m! \wedge \Diamond_{\leqslant 1} Check \wedge \Diamond_{\geqslant 1} Check)$

Adaptation of proof for MITL

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant
2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant
3. every $m!$ action, **followed by a Σ_C action after at least 1 t.u.**, is matched by an $m?$ action 1 t.u. later
   - MITL formula using *Check* again...
     $$\varphi = \Diamond\left(m! \wedge \Diamond_{<1}(Nil \wedge Nil \, \mathsf{U}(\Sigma_C \, \mathsf{U} \, Check)) \wedge \Diamond_{\geqslant 1} Check\right) \Rightarrow \Diamond\left(m? \wedge (m? \, \mathsf{U} \, Check)\right)$$
4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MITL formula
     $$\varphi' = \Diamond\left(m? \wedge (m? \, \mathsf{U} \, Check)\right) \Rightarrow \Diamond(m! \wedge \Diamond_{\leqslant 1} Check \wedge \Diamond_{\geqslant 1} Check)$$

Adaptation of proof for MITL

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant

2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant

3. every $m!$ action, **followed by a Σ_C action after at least 1 t.u.**, is matched by an $m?$ action 1 t.u. later
   - MITL formula using *Check* again...
     $$\varphi = \Diamond \big(m! \wedge \Diamond_{<1}(Nil \wedge Nil \,\mathsf{U}(\Sigma_C \,\mathsf{U}\, Check)) \wedge \Diamond_{\geqslant 1} Check\big) \Rightarrow \Diamond \big(m? \wedge (m? \,\mathsf{U}\, Check)\big)$$
   - assumption OK: because no loop containing only $m!$ action...

4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MITL formula
     $$\varphi' = \Diamond \big(m? \wedge (m? \,\mathsf{U}\, Check)\big) \Rightarrow \Diamond(m! \wedge \Diamond_{\leqslant 1} Check \wedge \Diamond_{\geqslant 1} Check)$$

Adaptation of proof for MITL

1. there exist $s_1, s_2, \cdots$ such that s_1 initial, $s_i \xrightarrow{a_i} s_{i+1}$ $\forall i$
   - encodable in the plant
2. no two actions on the same time: $t_i < t_{i+1}$
   - encodable in the plant
3. every $m!$ action, **followed by a Σ_C action after at least 1 t.u.**, is matched by an $m?$ action 1 t.u. later
   - MITL formula using *Check* again...
     $$\varphi = \Diamond\left(m! \wedge \Diamond_{<1}(Nil \wedge Nil \,\mathsf{U}(\Sigma_C \,\mathsf{U}\, Check)) \wedge \Diamond_{\geqslant 1} Check\right) \Rightarrow \Diamond\left(m? \wedge (m? \,\mathsf{U}\, Check)\right)$$
   - assumption OK: because no loop containing only $m!$ action...
4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MITL formula
     $$\varphi' = \Diamond\left(m? \wedge (m? \,\mathsf{U}\, Check)\right) \Rightarrow \Diamond(m! \wedge \Diamond_{\leqslant 1} Check \wedge \Diamond_{\geqslant 1} Check)$$

Theorem:

Implementable Reactive Synthesis for MITL specifications over finite words is undecidable.

Results for MITL

	RS	IRS	BRessRS
Finite	??	Undecidable	**on-the-fly** 3-EXPTIME
Infinite	Undecidable [Doyen, Geeraerts, Raskin, and Reichert, 2009]	Undecidable	3-EXPTIME [D'Souza and Madhusudan, 2002]

Results for MITL

	RS	IRS	BRessRS
Finite	...Ackerman-hard...	Undecidable	**on-the-fly** 3-EXPTIME
Infinite	Undecidable [Doyen, Geeraerts, Raskin, and Reichert, 2009]	Undecidable	3-EXPTIME [D'Souza and Madhusudan, 2002]

An interesting sub-problem

Bounded-precision reactive synthesis problem (BPrecRS): *find a* **finite set of clocks** X, *and a td STS* $\mathcal{T}$ *of controller with* X *as clocks, and* **a given precision** (m, K) *such that every possible play accepted by the plant verifies the specification*

An interesting sub-problem

Bounded-precision reactive synthesis problem (BPrecRS): *find a* **finite set of clocks** X, *and a td STS* $\mathcal{T}$ *of controller with* X *as clocks, and* **a given precision** (m, K) *such that every possible play accepted by the plant verifies the specification*

- Natural in practice...

An interesting sub-problem

Bounded-precision reactive synthesis problem (BPrecRS): *find a* **finite set of clocks** X, *and a td STS* $\mathcal{T}$ *of controller with* X *as clocks, and* **a given precision** (m, K) *such that every possible play accepted by the plant verifies the specification*

- Natural in practice...
- Bound on the precision: reflects hardware restrictions on the sensors and information transmission

An interesting sub-problem

Bounded-precision reactive synthesis problem (BPrecRS): *find a* **finite set of clocks** X, *and a td STS* $\mathcal{T}$ *of controller with* X *as clocks, and* **a given precision** (m, K) *such that every possible play accepted by the plant verifies the specification*

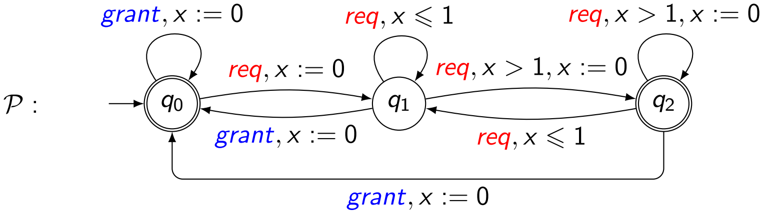
- Natural in practice...
- Bound on the precision: reflects hardware restrictions on the sensors and information transmission
- No real reasons for restricting the **number of clocks** that can easily grow without harm

An interesting sub-problem

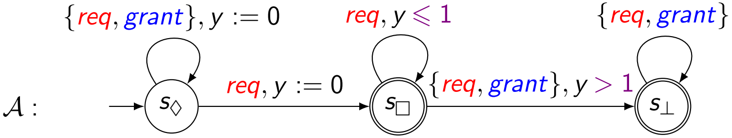
Bounded-precision reactive synthesis problem (BPrecRS): *find a* **finite set of clocks** X, *and a td STS* $\mathcal{T}$ *of controller with* X *as clocks, and* **a given precision** (m, K) *such that every possible play accepted by the plant verifies the specification*

- Natural in practice...
- Bound on the precision: reflects hardware restrictions on the sensors and information transmission
- No real reasons for restricting the **number of clocks** that can easily grow without harm
- But also **undecidable** via the previous proof!!

Running example

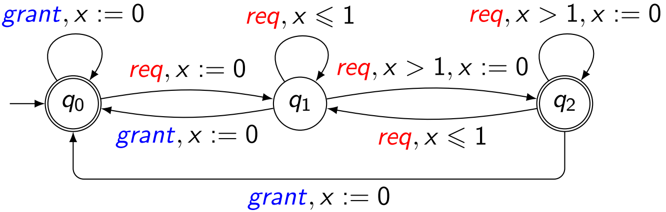


Specification: $\Box(req \Rightarrow \Diamond_{\leqslant 1}\, grant)$ equivalent to complement of



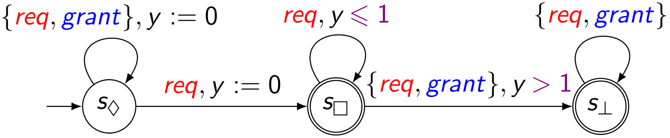
Running example

$\mathcal{P}$:



Specification: $\Box(req \Rightarrow \Diamond_{\leqslant 1}\, grant)$ equivalent to complement of

$\mathcal{A}$:



Question: find a controller $\mathcal{T}$ with precision $(m = 1, K = 1)$ such that "$(\mathcal{P} \| \mathcal{T}) \cap \mathcal{A} = \emptyset$"
Warning: set of clocks X for the controller not fixed a priori

Algorithm in a nutshell

- Construct the unfolding of all possible parallel executions of $\mathcal{P}$, $\mathcal{A}$, and all the possible controllers: **infinite tree**

Algorithm in a nutshell

- Construct the unfolding of all possible parallel executions of $\mathcal{P}$, $\mathcal{A}$, and all the possible controllers: **infinite tree**
- **Infinitely branching** (density of time): make it finitely branching by
  - only allowing the controller to reset at most one fresh clock at each step
  - merging equivalent choices with respect to regions (based on the precision (m, K) and the current set of clocks)

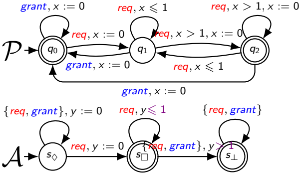
Algorithm in a nutshell

- Construct the unfolding of all possible parallel executions of $\mathcal{P}$, $\mathcal{A}$, and all the possible controllers: **infinite tree**
- **Infinitely branching** (density of time): make it finitely branching by
  - only allowing the controller to reset at most one fresh clock at each step
  - merging equivalent choices with respect to regions (based on the precision (m, K) and the current set of clocks)
- Semi-algorithm:
  - build the tree...
  - ... while testing on-the-fly if it is winning;
  - map a winning strategy to a controller $\mathcal{T}$.

Algorithm in a nutshell

- Construct the unfolding of all possible parallel executions of $\mathcal{P}$, $\mathcal{A}$, and all the possible controllers: **infinite tree**
- **Infinitely branching** (density of time): make it finitely branching by
  - only allowing the controller to reset at most one fresh clock at each step
  - merging equivalent choices with respect to regions (based on the precision (m, K) and the current set of clocks)
- Semi-algorithm:
  - build the tree...
  - ... while testing on-the-fly if it is winning;
  - map a winning strategy to a controller $\mathcal{T}$.
- Cut some useless branches with an order $\widetilde{\sqsubseteq}$ (that is not a wqo)

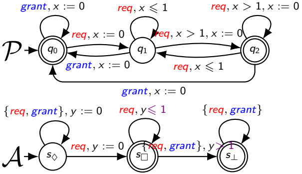
Running example: finite tree

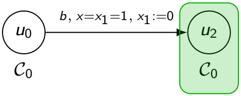


$\mathcal{C}_0$

$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

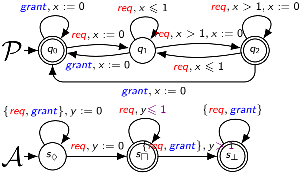
Running example: finite tree

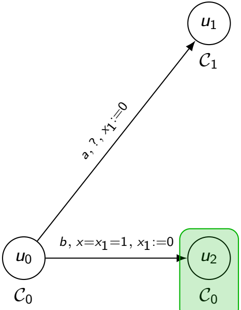




$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

Running example: finite tree

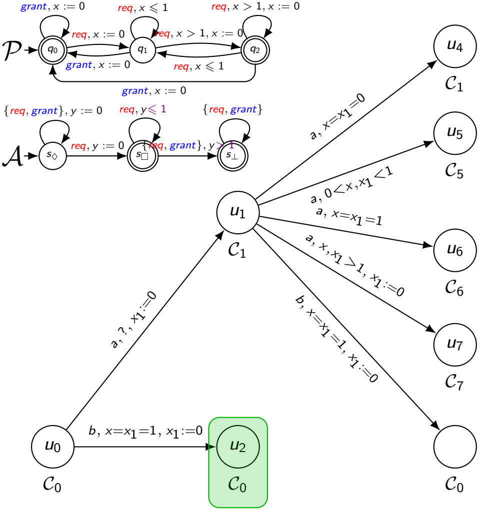




$$\mathcal{C}_0 = \big(q_0, \big\{\big(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}\big)\big\}\big)$$

$$\mathcal{C}_1 = \big(q_1, \big\{\big(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}\big), \big(s_\Box, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}\big)\big\}\big)$$

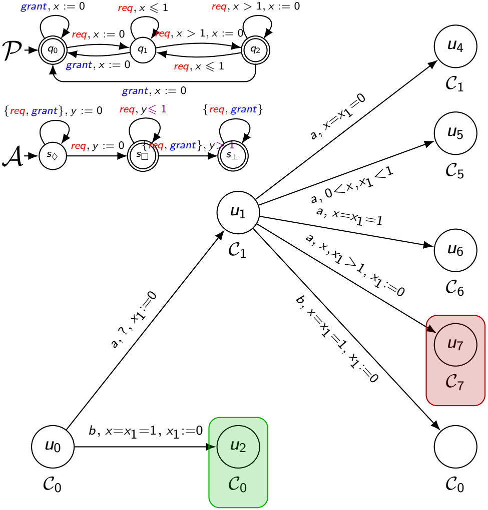
Running example: finite tree



$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\diamondsuit, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

$$\mathcal{C}_1 = \left(q_1, \left\{\left(s_\diamondsuit, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right), \left(s_\square, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

Running example: finite tree

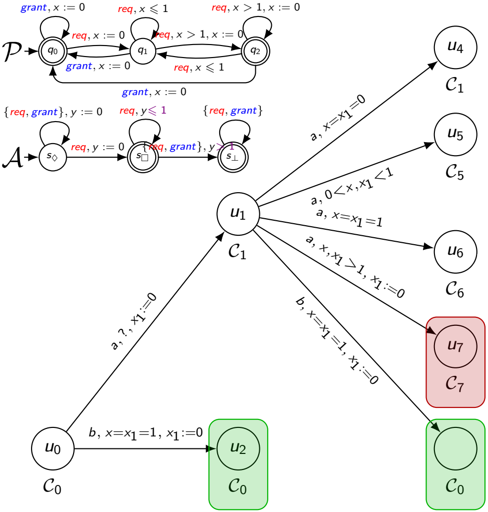


$$\mathcal{C}_0 = (q_0, \{(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\})\})$$

$$\begin{aligned}\mathcal{C}_1 = (q_1, \{&(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}),\\ &(s_\Box, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\})\})\end{aligned}$$

$$\begin{aligned}\mathcal{C}_7 = (q_2, \{&(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}),\\ &(s_\Box, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\})\ldots\})\end{aligned}$$

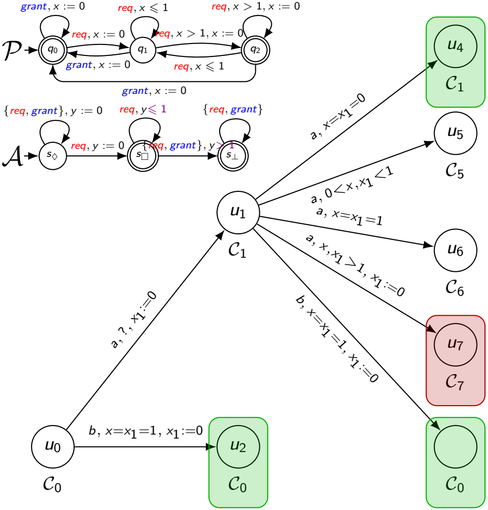
Running example: finite tree



$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

$$\begin{aligned}\mathcal{C}_1 = \big(q_1, \{&(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}),\\ &(s_\Box, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\})\}\big)\end{aligned}$$

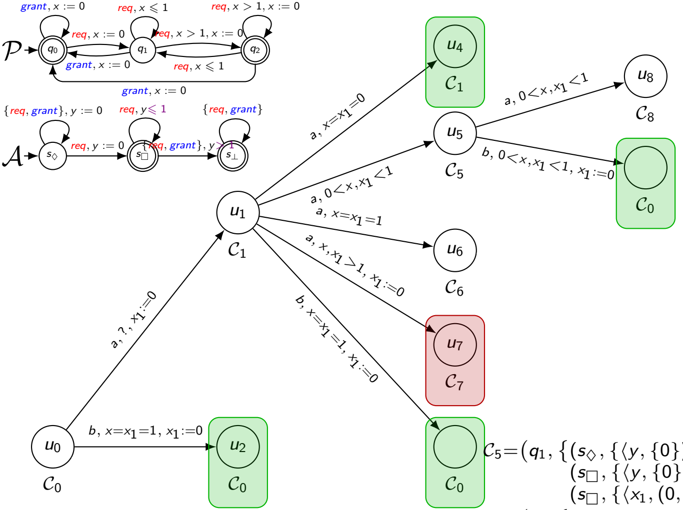
Running example: finite tree



$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

$$\begin{aligned}\mathcal{C}_1 = \big(q_1, \{&\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right),\\ &\left(s_\Box, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\}\big)\end{aligned}$$

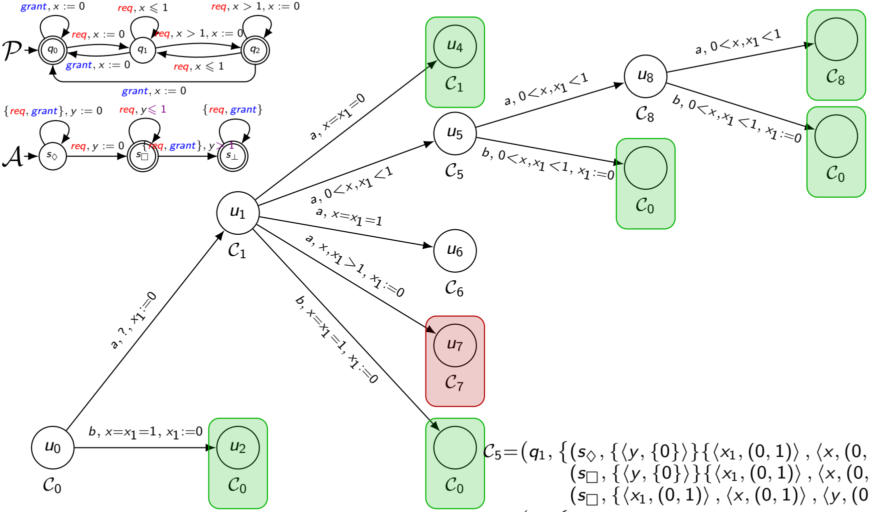
Running example: finite tree



$$\mathcal{C}_5 = \Big(q_1, \big\{(s_\Diamond, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}),\\ (s_\Box, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}),\\ (s_\Box, \{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle, \langle y, (0,1)\rangle\})\big\}\Big)$$

$$\mathcal{C}_8 = \Big(q_1, \big\{(s_\Diamond, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}),\\ (s_\Box, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}),\\ (s_\Box, \{\langle y, (0,1)\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}),\\ (s_\Box, \{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle, \langle y, (0,1)\rangle\})\big\}\Big)$$

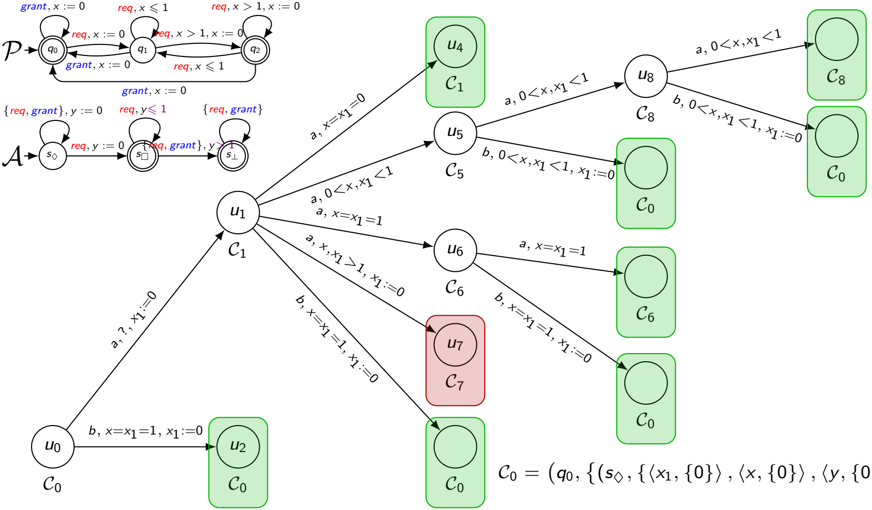
Running example: finite tree



$$\mathcal{C}_5 = \Big(q_1, \big\{ (s_\Diamond, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}), \\ (s_\Box, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}), \\ (s_\Box, \{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle, \langle y, (0,1)\rangle\})\big\}\Big)$$

$$\mathcal{C}_8 = \Big(q_1, \big\{ (s_\Diamond, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}), \\ (s_\Box, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}), \\ (s_\Box, \{\langle y, (0,1)\rangle\}\{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle\}), \\ (s_\Box, \{\langle x_1, (0,1)\rangle, \langle x, (0,1)\rangle, \langle y, (0,1)\rangle\})\big\}\Big)$$

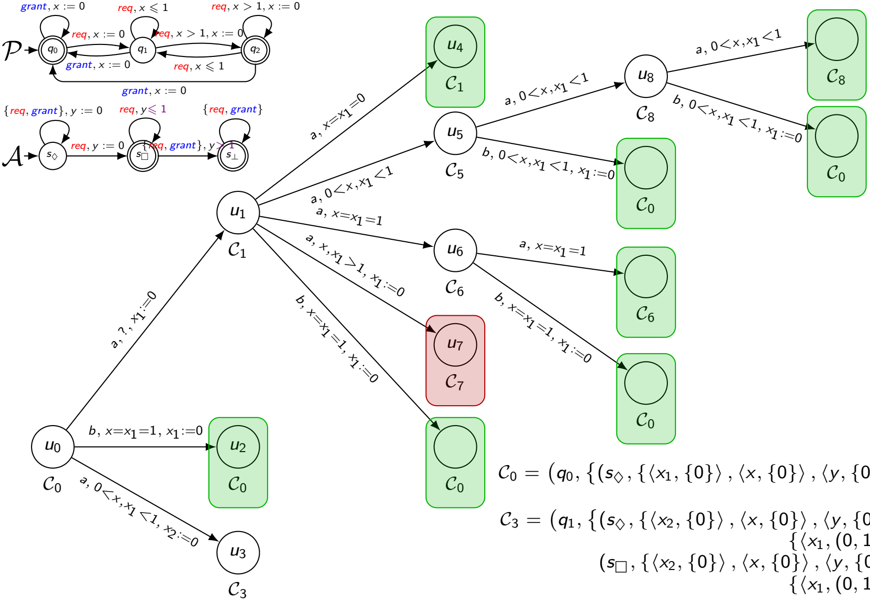
Running example: finite tree



$$\mathcal{C}_0 = (q_0, \{(s_\Diamond, \{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\})\})$$

$$\begin{aligned}\mathcal{C}_6 = (q_1, \{&(s_\Diamond, \{\langle y, \{0\}\rangle, \langle x_1, \{1\}\rangle, \langle x, \{1\}\rangle\}),\\ &(s_\Box, \{\langle y, \{0\}\rangle, \langle x_1, \{1\}\rangle, \langle x, \{1\}\rangle\}),\\ &(s_\Box, \{\langle x_1, \{1\}\rangle, \langle x, \{1\}\rangle, \langle y, \{1\}\rangle\})\})\end{aligned}$$

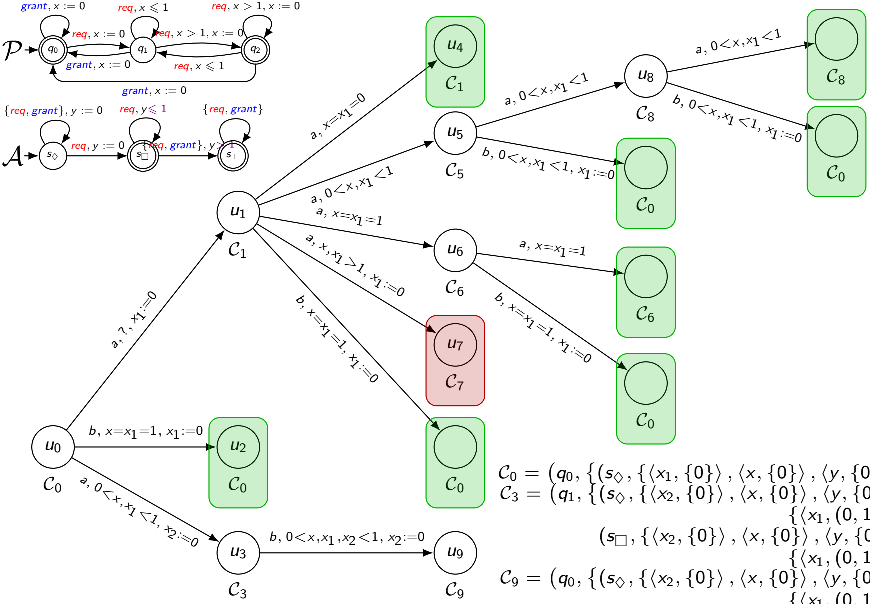
Running example: finite tree



$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

$$\begin{aligned}\mathcal{C}_3 = \Big(q_1, \Big\{&\left(s_\Diamond, \left\{\langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\} \left\{\langle x_1, (0,1)\rangle\right\}\right),\\ &\left(s_\Box, \left\{\langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\} \left\{\langle x_1, (0,1)\rangle\right\}\right)\Big\}\Big)\end{aligned}$$

Running example: finite tree

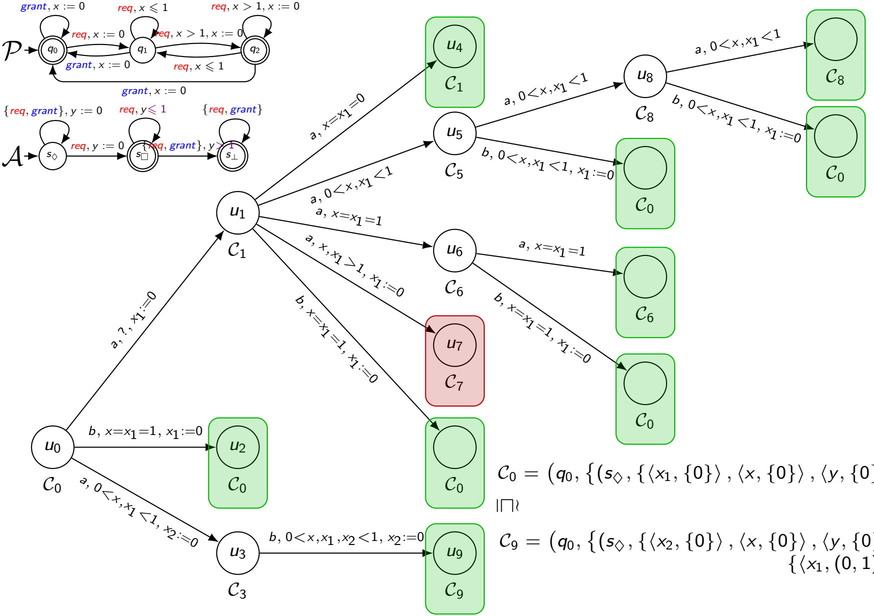


$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

$$\begin{aligned}\mathcal{C}_3 = \Big(q_1, \Big\{&\left(s_\Diamond, \left\{\langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\} \left\{\langle x_1, (0,1)\rangle\right\}\right),\\ &\left(s_\Box, \left\{\langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\} \left\{\langle x_1, (0,1)\rangle\right\}\right)\Big\}\Big)\end{aligned}$$

$$\mathcal{C}_9 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\} \left\{\langle x_1, (0,1)\rangle\right\}\right)\right\}\right)$$

Running example: finite tree



$$\mathcal{C}_0 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\}\right)\right\}\right)$$

$$\sqsubseteq$$

$$\mathcal{C}_9 = \left(q_0, \left\{\left(s_\Diamond, \left\{\langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\right\} \; \left\{\langle x_1, (0,1)\rangle\right\}\right)\right\}\right)$$

Conclusion

Reactive synthesis with plant for MITL specifications

	RS	IRS	BPrecRS	BRessRS
Finite	...Ackerman-hard...	Undecidable	Undecidable + semi-algo	**on-the-fly** 3-EXPTIME
Infinite	Undecidable [Doyen, Geeraerts, Raskin, and Reichert, 2009]	Undecidable	Undecidable	3-EXPTIME [D'Souza and Madhusudan, 2002]

Conclusion

Reactive synthesis with plant for MITL specifications

	RS	IRS	BPrecRS	BRessRS
Finite	...Ackerman-hard...	Undecidable	Undecidable + semi-algo	**on-the-fly** 3-EXPTIME
Infinite	Undecidable [Doyen, Geeraerts, Raskin, and Reichert, 2009]	Undecidable	Undecidable	3-EXPTIME [D'Souza and Madhusudan, 2002]

Future works:

- Test on benchmarks algorithm for BRessRS (over MITL), and semi-algorithm for BPrecRS (over timed automata)
- Explore other timed logics: Event-Clock Logic / Event-Clock Automata?
- Semi-algorithm for BPrecRS over infinite automata?
- Decidable fragments for BPrecRS

Conclusion

Reactive synthesis with plant for MITL specifications

	RS	IRS	BPrecRS	BRessRS
Finite	...Ackerman-hard...	Undecidable	Undecidable + semi-algo	**on-the-fly** 3-EXPTIME
Infinite	Undecidable [Doyen, Geeraerts, Raskin, and Reichert, 2009]	Undecidable	Undecidable	3-EXPTIME [D'Souza and Madhusudan, 2002]

Future works:

- Test on benchmarks algorithm for BRessRS (over MITL), and semi-algorithm for BPrecRS (over timed automata)
- Explore other timed logics: Event-Clock Logic / Event-Clock Automata?
- Semi-algorithm for BPrecRS over infinite automata?
- Decidable fragments for BPrecRS

Thank you for your attention

References

Patricia Bouyer, Laura Bozzelli, and Fabrice Chevalier. Controller synthesis for MTL specifications. In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2006.

Thomas Brihaye, Morgane Estiévenart, and Gilles Geeraerts. On MITL and alternating timed automata. In *Proceedings of the 11th international conference on Formal Modeling and Analysis of Timed Systems (FORMATS'13)*, volume 8053 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2013.

Laurent Doyen, Gilles Geeraerts, Jean-François Raskin, and Julien Reichert. Realizability of real-time logics. In *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2009.

Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *Proceedings of the 19th Annual conference on Theoretical Aspects of Computer Science (STACS'02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582. Springer, 2002.

Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.