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

Environment || | Controller | = | Req.
Metric Temporal Logic (MTL)

\[ \varphi ::= \top \mid a \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi U_I \varphi \]

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

\[ \varphi_1 U_I \varphi_2 \]

\[ \square \varphi \]

\[ \Diamond \varphi \]
Synthesis with plant: example of lift

$$\Sigma = \Sigma_c \cup \Sigma_e$$

- **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** $P$ = Time-det. timed automaton

Specification via MTL: "lift grants the calls in reasonable time"

$$□ (\text{req} \Rightarrow ▶ \leq 2 \text{grant})$$

Play: environment and controller propose timed actions $(t, \text{req}) \rightarrow (t', \text{grant})$

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
Synthesis with plant: example of lift

\[ \Sigma = \Sigma_c \cup \Sigma_e \]

- 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 \( P = \) Time-det. timed automaton
Synthesis with plant: example of lift

\[ \Sigma = \Sigma_c \cup \Sigma_e \]

- 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”

\[ \Box (\text{req} \Rightarrow \diamond \leq_2 \text{grant}) \]
Synthesis with plant: example of lift

\[ \Sigma = \Sigma_c \cup \Sigma_e \]

- **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 \( \mathcal{P} = \text{Time-det. timed automaton} \)

Specification via MTL: "lift grants the calls in reasonable time" \( \Box (\text{req} \Rightarrow \diamond \leq 2 \text{grant}) \)

Play: environment and controller propose timed actions

\[(t, \text{req}) \quad (t', \text{grant})\]

Only action(s) with the shortest delay \( \min(t, t') \) may be played
Synthesis with plant: example of lift

\[ \Sigma = \Sigma_c \cup \Sigma_e \]

- **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** \( P = \) Time-det. timed automaton

Specification via MTL: “*lift grants the calls in reasonable time*”

\[ \square(req \Rightarrow \Diamond \leq 2\, grant) \]

Play: **environment** and **controller** propose timed actions

\[ (t, \text{ req}) \quad (t', \text{ grant}) \]

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: $\Box ( req \land \Diamond \geq 1 \text{req} \Rightarrow \Diamond = 1 \text{grant} )$
A toy example

Universal plant $\mathcal{P}$:

Specification: $\Box\left( req \land \Diamond \geq 1 req \Rightarrow \Diamond = 1 grant \right)$

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

Universal plant \( \mathcal{P} \):

\[ q_0 \]

Specification: \( \square (req \land \Diamond \geq 1 req \Rightarrow \Diamond = 1 grant) \)

**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 $\sigma \cdot (c, \mathcal{T})$ is possible,
and $\mathcal{T}$ may fire $(t, grant)$ currently,
then $\sigma \cdot (c, \mathcal{T}) \cdot (grant, \mathcal{T} + t)$ is possible if readable in the plant,
and $\sigma \cdot (c, \mathcal{T}) \cdot (req, \mathcal{T} + t')$ is possible if readable in the plant, with $t' \leq t$.
```
Implementable Reactive Synthesis (IRS)

Controller = time-deterministic symbolic transition system $T$
- 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, $T$ 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 $T$ may fire $(t, grant)$ currently, then $\sigma \cdot (c, T) \cdot (grant, T + t)$ is possible if readable in the plant, and $\sigma \cdot (c, T) \cdot (req, T + t')$ is possible if readable in the plant, with $t' \leq t$.

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

Universal plant $\mathcal{P}$:

Specification: $\Box (\text{req} \land \Diamond \geq 1 \text{ req} \Rightarrow \Diamond = 1 \text{ grant})$

**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: $\Box (req \land \Diamond \geq 1 \, req \Rightarrow \Diamond = 1 \, grant)$

**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

\[\begin{align*}
\text{grant}, x &= 0 \\
\text{req}, x &\leq 1 \\
\text{req}, x &> 1, x := 0
\end{align*}\]

\[\begin{array}{c}
\text{P}:
\end{array}\]

\[\begin{array}{c}
\begin{array}{c}
\text{q}_0 \\
\text{q}_1 \\
\text{q}_2
\end{array}
\end{array}\]

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

\[
\begin{align*}
\text{grant}, x &:= 0 & \text{req}, x \leq 1 & \quad \text{req}, x > 1, x := 0 \\
\end{align*}
\]

\[
\begin{array}{c}
P : \\
q_0 & \rightarrow & q_1 & \rightarrow & q_2 \\
\text{grant}, x := 0 & \quad & \text{req}, x := 0 & \quad & \text{req}, x > 1, x := 0 \\
\end{array}
\]

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

Specification: \(\square (\text{req} \Rightarrow \Diamond \leq 1 \text{grant})\)
Another example

\[
\begin{align*}
\text{grant}, x &:= 0 & \text{req}, x &\leq 1 & \text{req}, x &> 1, x := 0
\end{align*}
\]

\[P:\]

\[\begin{array}{c}
q_0 \quad \text{req}, x := 0 \quad q_1 \quad \text{req}, x > 1, x := 0 \\
\downarrow \quad \text{grant}, x := 0 \quad \downarrow \\
q_2 \quad \text{req}, x \leq 1
\end{array}\]

- 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: \(\square (\text{req} \Rightarrow \Diamond \leq 1 \text{grant})\)

**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

\[T:\]

\[\begin{array}{c}
l_0 \quad \text{req}, z := 0 \quad l_1 \\
\downarrow \quad \text{grant}, z = 1
\end{array}\]
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
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 $G_{m,K}^{\text{max}}(X)$

$$g ::= \top \mid g \land g \mid x < \alpha/m \mid x \leq \alpha/m \mid x = \alpha/m \mid x \geq \alpha/m \mid x > \alpha/m$$

with $x \in X$, and $0 \leq \alpha \leq K$. 
Bounding *a priori* the resources: set of clocks $X$ and precision $(m, K)$ of the controller. Comparisons with maximal guards in $G_{m,K}^{\text{max}}(X)$

$$g ::= \top \mid g \land g \mid x < \alpha/m \mid x \leq \alpha/m \mid x = \alpha/m \mid x \geq \alpha/m \mid x > \alpha/m$$

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

**Bounded-resources reactive synthesis problem (BRessRS):** find a td STS $T$ 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

\[ grant, x := 0 \quad req, x \leq 1 \quad req, x > 1, x := 0 \]

\[ P : \]

\[ q_0 \rightarrow q_1 \rightarrow q_2 \]

Specification: \( \square (req \Rightarrow \diamond \leq 1 \ grant) \)

CONTROLLABLE for BResRS: 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 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

\[ \varphi ::= T \mid a \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \mathbin{U}_I \varphi \]

with **non-singular** interval of \( \mathbb{R}^+ \) with bounds in \( \mathbb{N} \cup \{+\infty\} \)
First contribution

**Theorem:** [Bouyer, Bozzeelli, 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

\[ \varphi ::= T \mid a \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \cup_I \varphi \]

with / 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:
Alternating automata combine:

- disjunctive transitions = non-determinism = the suffix of timed word must be accepted from \textbf{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

\[ \varphi = \Box (\text{req} \Rightarrow \Diamond [1,2] \text{grant}) \]
From MTL to OCATA

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

Diagram:

- Start state: req
- Transition: req \(\rightarrow\) y := 0
- Transition: y := 0 \(\rightarrow\) grant
- Grant state: grant
- Grant condition: y \(\in\) [1, 2]
From MTL to OCATA

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

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

\[ \square 0 \]
From MTL to OCATA

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

Execution on the timed word \((\text{req}, 0.5)(\text{req}, 0.6)(\text{req}, 1.2)(\text{grant}, 2.3)\):
From MTL to OCATA

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

Execution on the timed word $$(\text{req}, 0.5)(\text{req}, 0.6)(\text{req}, 1.2)(\text{grant}, 2.3)$$:
From MTL to OCATA

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

Execution on the timed word \((\text{req}, 0.5)(\text{req}, 0.6)(\text{req}, 1.2)(\text{grant}, 2.3)\):
From MTL to OCATA

\[ \varphi = \square (req \Rightarrow \Diamond [1,2] grant) \]

\[ y := 0 \]

\[ y \in [1, 2] \]

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

\[ \Diamond 0 \longrightarrow \Diamond 0.1 \longrightarrow \Diamond 0.7 \quad y = 1.8 \]

\[ \square 0 \]

\[ \Diamond 0 \longrightarrow \Diamond 0.6 \quad y = 1.7 \]

\[ \square 0.5 \]

\[ \square 0 \longrightarrow \Diamond 0.6 \quad y = 1.1 \]

\[ \square 0.6 \]

\[ \square 1.2 \longrightarrow \square 2.3 \]

\[ \square 0.5 \]

\[ \square 0 \longrightarrow \Diamond 0.1 \longrightarrow \Diamond 0.7 \quad y = 1.8 \]

\[ \square 0 \]

\[ \Diamond 0 \longrightarrow \Diamond 0.6 \quad y = 1.7 \]

\[ \square 0.6 \]

\[ \square 1.2 \longrightarrow \square 2.3 \]
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: $\varphi$ in MTL, Ressources: $(X, m, K)$
- **Convert** the MTL formula $\neg \varphi$ into an **OCATA** $A$
- **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 $A$, the controller $T$
Bounded-Ressources Reactive Synthesis for MTL

- Plant: \( \mathcal{P} \), Specification: \( \varphi \) 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: $\varphi$ in MTL, Resources: $(X, m, K)$
- **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

\[(q, \{H_1, H_2, \ldots, H_n\})\]
Bounded-Ressources Reactive Synthesis for MTL

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

\[
(q, \{H_1, H_2, \ldots, H_n\})
\]
Bounded-Ressources Reactive Synthesis for MTL

- Plant: \( P \), Specification: \( \varphi \) in MTL, Ressources: \( (X, m, K) \)
- **Convert** the MTL formula \( \neg \varphi \) into an **OCATA** \( A \)
- **Cast** the control problem into a **timed game** played on a tree
- The tree **unravels** the execution of the parallel composition of: the plant \( P \), the OCATA \( A \), the controller \( T \)
- 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 \( \lambda_i \subseteq 2^{(X_P \cup X \cup Q_A) \times \text{REG}_m, K} \): region associated to all clocks

\[
(q, \{H_1, H_2, \ldots, H_n\})
\]
Bounded-Resources Reactive Synthesis for MTL

Action \( (a, g, R) \)
- \( a \): letter of \( \Sigma_c \cup \Sigma_e \)
- \( g \): guard over clocks of \( X \) and \( X_P \)
- \( R \): resets of clocks of \( X \)

\[
(q, \{H_1, H_2, \ldots, H_n\})
\]

\[
(q', \{H'_1, H'_2, \ldots, H'_n\})
\]

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 \( A(\equiv \neg \varphi) \)
If tree finite and winning strategy: we have a (finite) controller
Bounded-Resources Reactive Synthesis for MTL

- Action \( (a, g, R) \)
  - \( a \): letter of \( \Sigma_c \cup \Sigma_e \)
  - \( g \): guard over clocks of \( X \) and \( X_p \)
  - \( R \): resets of clocks of \( X \)

\[
(q, \{H_1, H_2, \ldots, H_n\})
\]

\[
(q', \{H'_1, H'_2, \ldots, H'_n\})
\]

- Finite abstraction is a (time-abstract) bisimulation
Bounded-Resources Reactive Synthesis for MTL

- Action \((a, g, R)\)
  - \(a\): letter of \(\Sigma_c \cup \Sigma_e\)
  - \(g\): guard over clocks of \(X\) and \(X_P\)
  - \(R\): resets of clocks of \(X\)

\[
(q, \{H_1, H_2, \ldots, H_n\})
\]

\[
(q', \{H'_1, H'_2, \ldots, H'_{n'}\})
\]

- 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 \(X_P\)
  - \(R\): resets of clocks of \(X\)

\[
(q, \{H_1, H_2, \ldots, H_n\}) \\
(q', \{H_1', H_2', \ldots, H_n'\})
\]

- 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 \(A\) \((\equiv \neg \phi)\)
- If tree finite and winning strategy: we have a (finite) controller \(T\)
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.

\[ u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4 \rightarrow u_5 \rightarrow u_6 \]
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

$u_1 \sqsubseteq u_2 \sqsubseteq u_5 \sqsubseteq u_6$

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

Complexity: non-primitive recursive due to well-quasi orderings
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

Diagram:

$u_1 \sqsubseteq u_2 \sqsubseteq u_5 \sqsubseteq u_6$

$u_2 \sqsubseteq u_3 \sqsubseteq u_4$

- 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, Estévenart, and Geeraerts, 2013]:

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

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

ϕ = □(req ⇒ ◊[1,2] grant)

Diagram:

- req req
- 0.5 0.6
- 1.2
- 1.3
- grant
- 2.3

0.3
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

ϕ = □(req ⇒ ◊[1,2] 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

\[ y := 0 \]

\[ y \in [1, 2] \]
Example run with the interval semantics

\[ y := 0 \]

\[ y \in [1, 2] \]

\[ \text{grant} \quad \text{req} \]

\[ 0.5 \quad 0.6 \quad 1.2 \quad 2.3 \]
Example run with the interval semantics

\[ y := 0 \]

\[ y \in [1, 2] \]

\[ \square 0 \rightarrow \square 0.5 \rightarrow \square 0.6 \rightarrow \square 1.2 \]

\[ \diamond 0 \rightarrow \diamond [0, 0.7] \rightarrow \diamond [0, 1] \rightarrow \diamond [0, 0.7] \]

\[ 0.5 \rightarrow 0.6 \rightarrow 1.2 \rightarrow 2.3 \]
Example run with the interval semantics

\[
\begin{align*}
\text{grant} & \quad \text{req} \quad y := 0 \\
\text{req} & \quad \text{grant} \quad y \in [1, 2]
\end{align*}
\]
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, Bozzei, 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, Estevenart, 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 \(H_i \leq H_j\)
Heuristics

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

- **Antichains**:  
  - in a label \((q, \{H_1, \ldots, H_n\})\), do not keep \(H_i\) such that \(H_i \leq H_j\)  
  - **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 $\Sigma_C = \{ m?, m!, \ldots \}$:

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 $\Sigma_C = \{ m?, m!, \ldots \}$:

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 determinstic 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 $\Sigma_C = \{m?, m!, \ldots\}$:

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 $\varphi = \Box (m! \land \Diamond \geq 1 \Sigma_C \Rightarrow \Diamond = 1 m?)$
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

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

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

Then, formula $\varphi' = \lozenge(m? \land \lozenge_{\geq 0} \text{Check}) \Rightarrow \lozenge(m! \land \lozenge_{\geq 1} \text{Check})$ checks 4.
4. every $m?$ action matched by an $m!$ action 1 t.u. before

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

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

Then, formula $\varphi' = \diamond (m? \land \diamond_{\geq 0} \text{Check}) \Rightarrow \diamond (m! \land \diamond_{\geq 1} \text{Check})$ checks 4.

**Theorem:**

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

4. every $m? \text{ action matched by an } m! \text{ action } 1 \text{ t.u. before}$

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

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

Then, formula $\varphi' = \diamond (m? \land \diamond_{\geq 0} \text{Check}) \Rightarrow \diamond (m! \land \diamond_{\geq 1} \text{Check})$ 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

\[ \Sigma_E = \{ \text{Check}, \text{Nil} \} \]

Plant \( P \): ensures a turn-based behaviour, Environment plays after 0 t.u., Check action is played only once...

Then, formula \( \varphi' = \Diamond (m? \land \Diamond_{\geq 0} \text{Check}) \implies \Diamond (m! \land \Diamond_{\geq 1} \text{Check}) \) checks 4.

Theorem:

There exists a controller \( T \) if and only if the channel machine halts.

\( \iff \): 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 $\varphi = \Box (m! \land \Diamond \geq 1 \Sigma C \Rightarrow \Diamond = 1 m?)$

4. every $m?$ action matched by an $m!$ action $1$ t.u. before
   ▶ MTL formula $\varphi' = \Diamond (m? \land \Diamond = 0 \text{Check}) \Rightarrow \Diamond (m! \land \Diamond = 1 \text{Check})$
Adaptation of proof for MITL

1. there exist $s_1, s_2, \ldots$ 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 $\varphi = \Box (m! \land \Box \geq 1 \Sigma \Rightarrow \Box = 1 m?)$

4. every $m?$ action matched by an $m!$ action 1 t.u. before
   ▶ MITL formula
   
   $\varphi' = \Diamond (m? \land (m? \cup \text{Check})) \Rightarrow \Diamond (m! \land \Diamond \leq 1 \text{Check} \land \Diamond \geq 1 \text{Check})$
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 $\Sigma_C$ action after at least 1 t.u., is matched by an $m?$ action 1 t.u. later
   ▶ MITL formula using Check again...
   $\varphi = \Diamond (m! \land \Diamond <_1 (\text{Nil} \land \text{Nil} \cup (\Sigma_C \cup \text{Check})) \land \Diamond \geq 1 \text{Check}) \Rightarrow$
   $\Diamond (m? \land (m? \cup \text{Check}))$

4. every $m?$ action matched by an $m!$ action 1 t.u. before
   ▶ MITL formula
   $\varphi' = \Diamond (m? \land (m? \cup \text{Check})) \Rightarrow \Diamond (m! \land \Diamond \leq 1 \text{Check} \land \Diamond \geq 1 \text{Check})$
Adaptation of proof for MITL

1. there exist $s_1, s_2, \ldots$ 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 $\Sigma_C$ action after at least 1 t.u., is matched by an $m?$ action 1 t.u. later
   - MITL formula using $\text{Check}$ again...
     \[ \varphi = \Diamond (m! \land \Diamond_{<1} (\text{Nil} \land \text{Nil} \lor (\Sigma_C \lor \text{Check})) \land \Diamond_{\geq 1} \text{Check}) \Rightarrow \\
     \Diamond (m? \land (m? \lor \text{Check})) \]
   - assumption OK: because no loop containing only $m!$ action...

4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MITL formula
     \[ \varphi' = \Diamond (m? \land (m? \lor \text{Check})) \Rightarrow \Diamond (m! \land \Diamond_{\leq 1} \text{Check} \land \Diamond_{\geq 1} \text{Check}) \]
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 $\Sigma_C$ action after at least 1 t.u., is matched by an $m?$ action 1 t.u. later
   - MITL formula using Check again...
     $$\varphi = \Diamond (m! \land \Diamond_{<1} (\text{Nil} \land \text{Nil} \cup (\Sigma_C \cup \text{Check})) \land \Diamond_{\geq 1} \text{Check}) \Rightarrow \Diamond (m? \land (m? \cup \text{Check}))$$
   - assumption OK: because no loop containing only $m!$ action...

4. every $m?$ action matched by an $m!$ action 1 t.u. before
   - MITL formula
     $$\varphi' = \Diamond (m? \land (m? \cup \text{Check})) \Rightarrow \Diamond (m! \land \Diamond_{<1} \text{Check} \land \Diamond_{\geq 1} \text{Check})$$

Theorem:

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

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>IRS</th>
<th>BResRS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finite</strong></td>
<td>??</td>
<td>Undecidable</td>
<td><strong>on-the-fly</strong> 3-EXPTIME</td>
</tr>
<tr>
<td><strong>Infinite</strong></td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>3-EXPTIME</td>
</tr>
<tr>
<td></td>
<td>[Doyen, Geeraerts, Raskin, and Reichert, 2009]</td>
<td></td>
<td>[D'Souza and Madhusudan, 2002]</td>
</tr>
</tbody>
</table>
## Results for MITL

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>IRS</th>
<th>BRessRS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finite</strong></td>
<td>...Ackerman-hard...</td>
<td>Undecidable</td>
<td><strong>on-the-fly 3-EXPTIME</strong></td>
</tr>
<tr>
<td><strong>Infinite</strong></td>
<td>Undecidable</td>
<td>Undecidable</td>
<td><strong>3-EXPTIME</strong></td>
</tr>
<tr>
<td></td>
<td>[Doyen, Geeraerts, Raskin, and Reichert, 2009]</td>
<td></td>
<td>[D’Souza and Madhusudan, 2002]</td>
</tr>
</tbody>
</table>
An interesting sub-problem

Bounded-precision reactive synthesis problem (BPrecRS): find a finite set of clocks $X$, and a td STS $T$ 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** (BPreRS): *find a finite set of clocks $X$, and a td STS $T$ 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 $T$ 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 $T$ 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

\[grant, x := 0 \quad req, x \leq 1 \quad req, x > 1, x := 0\]

\[\mathcal{P} : \quad q_0 \rightarrow q_1 \rightarrow q_2\]

\[grant, x := 0 \quad req, x := 0 \quad req, x > 1, x := 0 \quad req, x \leq 1\]

\[grant, x := 0\]

Specification: \(\Box(\text{req} \Rightarrow \Diamond \leq_1 grant)\) equivalent to complement of

\(\{\text{req, grant}\}, y := 0 \quad \text{req, } y \leq 1 \quad \{\text{req, grant}\}\)

\(\mathcal{A} : \quad s_\Diamond \rightarrow s_\Box \rightarrow s_{\perp}\)

\(\text{req, } y := 0 \quad \{\text{req, grant}\}, y > 1\)
Running example

\[ P : \]

\begin{align*}
\text{grant, } x &:= 0 & \text{req, } x &\leq 1 & \text{req, } x &> 1, x := 0 \\
\text{req, } x &:= 0 & \text{req, } x &> 1, x := 0 & \text{grant, } x &:= 0 & \text{req, } x &\leq 1
\end{align*}

\[ q_0 \quad q_1 \quad q_2 \]

Specification: \( \square (\text{req} \Rightarrow \Diamond \leq_1 \text{grant}) \) equivalent to complement of

\[ \{ \text{req, grant} \}, y := 0 \quad \text{req, } y \leq 1 \quad \{ \text{req, grant} \} \]

\[ s_{\Diamond} \quad s_{\Box} \quad s_{\bot} \]

\[ A : \]

\textbf{Question:} find a controller \( T \) with precision \( (m = 1, K = 1) \) such that \( "(P \| T) \cap A = \emptyset" \)

\textbf{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 $P, A$, and all the possible controllers: \textit{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 $\succeq$ (that is not a wqo)
Running example: finite tree

$$C_0 = (q_0, \{(s_0, \{(\langle x_1, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle)\})\})$$
Running example: finite tree

\[ C_0 = (q_0, \{(s_\Diamond, \{\langle x_1, \{0\} \rangle, \langle x, \{0\} \rangle, \langle y, \{0\} \rangle\})\}) \]
Running example: finite tree

\[ C_0 = (q_0, \{(s\diamond, \{\langle x_1, \{0\}\}, \langle x, \{0\}\}, \langle y, \{0\}\}\})\}) \]

\[ C_1 = (q_1, \{(s\diamond, \{\langle x_1, \{0\}\}, \langle x, \{0\}\}, \langle y, \{0\}\}\}), \]

\[ (s\square, \{\langle x_1, \{0\}\}, \langle x, \{0\}\}, \langle y, \{0\}\}\})) \]
Running example: finite tree

\[ \begin{align*}
C_0 &= (q_0, \{ (s_0, \{ \langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, 0 \rangle \}) \}) \\
C_1 &= (q_1, \{ (s_0, \{ \langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, 0 \rangle \}), (s_\square, \{ \langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, 0 \rangle \}) \})
\end{align*} \]
Running example: finite tree

\[ C_0 = (q_0, \{(s_\diamond, \{\langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, \emptyset \rangle\})\}) \]

\[ C_1 = (q_1, \{(s_\diamond, \{\langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, \emptyset \rangle\}), (s_{\square}, \{\langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, \emptyset \rangle\})\}) \]

\[ C_7 = (q_2, \{(s_\diamond, \{\langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, \emptyset \rangle\}), (s_{\square}, \{\langle x_1, 0 \rangle, \langle x, 0 \rangle, \langle y, \emptyset \rangle\})\}) \]
Running example: finite tree

\[ C_0 = (q_0, \{(s\diamond, \{\langle x_1, \{0\} \rangle, \langle x, \{0\} \rangle, \langle y, \{0\} \rangle\})\}) \]

\[ C_1 = (q_1, \{(s\diamond, \{\langle x_1, \{0\} \rangle, \langle x, \{0\} \rangle, \langle y, \{0\} \rangle\}),
(s\square, \{\langle x_1, \{0\} \rangle, \langle x, \{0\} \rangle, \langle y, \{0\} \rangle\})\}) \]
Running example: finite tree

\[
C_0 = (q_0, \{(s\Diamond, \{(x_1, \{0\}), (x, \{0\}), (y, \{0\})\})\})
\]

\[
C_1 = (q_1, \{(s\Diamond, \{(x_1, \{0\}), (x, \{0\}), (y, \{0\})\}), (s\Box, \{(x_1, \{0\}), (x, \{0\}), (y, \{0\})\})\})
\]
Running example: finite tree

\[ \text{grant, } x := 0 \quad \text{req, } x \leq 1 \quad \text{req, } x > 1, x := 0 \]

\[ q_0 \xrightarrow{\text{req, } x := 0} q_1 \xrightarrow{\text{req, } x > 1, x := 0} q_2 \]

\[ q_0 \xrightarrow{\text{grant, } x := 0} q_1 \xrightarrow{\text{req, } x \leq 1} q_2 \]

\[ C_0 = (q_1, \{(s\diamond, \{y, \{0\}\})\{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle\}),
(s\square, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle\}),
(s\square, \{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle, \langle y, (0, 1)\rangle\}))\}
\]

\[ C_8 = (q_1, \{(s\diamond, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle\}),
(s\square, \{\langle y, \{0\}\rangle\}\{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle\}),
(s\square, \{\langle y, (0, 1)\rangle\}\{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle\}),
(s\square, \{\langle x_1, (0, 1)\rangle, \langle x, (0, 1)\rangle, \langle y, (0, 1)\rangle\}))\}\} \]
Running example: finite tree

\[ P \quad a \quad b \]

\[ A \quad s_0 \quad q_0 \quad q_1 \quad q_2 \]

\[ a, 0 < x, x_1 < 1 \]
\[ b, 0 < x, x_1 < 1, x_1 := 0 \]

\[ C_0 \quad u_0 \quad u_2 \quad u_4 \quad u_5 \quad u_6 \quad u_7 \quad u_8 \]

\[ C_1 \quad C_2 \quad C_3 \quad C_4 \quad C_5 \quad C_6 \quad C_7 \quad C_8 \]

\[ C_0 = (q_1, \{ (s_\square, \{(y, \{0\})\}, \{(x_1, (0, 1)), (x, (0, 1))\}) \}) \]

\[ C_8 = (q_1, \{ (s_\square, \{(y, \{0\})\}, \{(x_1, (0, 1)), (x, (0, 1))\}) \}) \]
Running example: finite tree

\[ C_0 = (q_0, \{ (s_\Diamond, \{ \langle x_1, \{0\} \}, \langle x, \{0\} \}, \langle y, \{0\} \}) \}) \]

\[ C_6 = (q_1, \{ (s_\Diamond, \{ \langle y, \{0\} \}, \langle x_1, \{1\} \}, \langle x, \{1\} \} \}),
(s_\Box, \{ \langle y, \{0\} \}, \langle x_1, \{1\} \}, \langle x, \{1\} \}),
(s_\Box, \{ \langle x_1, \{1\} \}, \langle x, \{1\} \}, \langle y, \{1\} \})) \]
Running example: finite tree

\[ C_0 = (q_0, \{(s_\Diamond, \langle x_1, \{0\}\rangle), \langle x, \{0\}\rangle, \langle y, \{0\}\rangle\}) \]

\[ C_3 = (q_1, \{(s_\Diamond, \langle x_2, \{0\}\rangle), \langle x, \{0\}\rangle, \langle y, \{0\}\rangle \}
\]

\[ \{(x_1, (0, 1))\})],

\[ (s_\Box, \langle x_2, \{0\}\rangle, \langle x, \{0\}\rangle, \langle y, \{0\}\rangle \}
\]

\[ \{(x_1, (0, 1))\})]) \]
Running example: finite tree

\[ C_0 = (q_0, \{(s_\diamond, \{\langle x_1, 0 \rangle\}, \langle x, 0 \rangle, \langle y, 0 \rangle\})\}) \]

\[ C_3 = (q_1, \{(s_\diamond, \{\langle x_2, 0 \rangle, \langle x, 0 \rangle, \langle y, 0 \rangle\})\}) \]

\[ C_9 = (q_0, \{(s_\diamond, \{\langle x_2, 0 \rangle, \langle x, 0 \rangle, \langle y, 0 \rangle\})\}) \]
Running example: finite tree

\[
C_0 = (q_0, \{(\mathbf{s}, \{\langle x_1, \{0\}\}, \langle x, \{0\}\}, \langle y, \{0\}\})\})
\]

\[
C_9 = (q_0, \{(\mathbf{s}, \{\langle x_2, \{0\}\}, \langle x, \{0\}\}, \langle y, \{0\}\})
\}
\}
\]
## Conclusion

Reactive synthesis with plant for MITL specifications

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>IRS</th>
<th>BPrecRS</th>
<th>BRessRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite</td>
<td>...Ackerman-hard...</td>
<td>Undecidable</td>
<td>Undecidable + semi-algo</td>
<td>on-the-fly 3-EXPTIME</td>
</tr>
<tr>
<td>Infinite</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>3-EXPTIME</td>
</tr>
<tr>
<td></td>
<td>[Doyen, Geeraerts,</td>
<td></td>
<td>[D’Souza and Madhusudan, 2002]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Raskin, and Reichert, 2009]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Reactive synthesis with plant for MITL specifications

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>IRS</th>
<th>BPrescRS</th>
<th>BResRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite</td>
<td>...Ackerman-hard...</td>
<td>Undecidable</td>
<td>Undecidable + semi-algo</td>
<td>on-the-fly 3-EXPTIME</td>
</tr>
<tr>
<td>Infinite</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>3-EXPTIME</td>
</tr>
<tr>
<td></td>
<td>[Doyen, Geeraerts, Raskin, and Reichert, 2009]</td>
<td></td>
<td></td>
<td>[D'Souza and Madhusudan, 2002]</td>
</tr>
</tbody>
</table>

Future works:

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

Reactive synthesis with plant for MITL specifications

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>IRS</th>
<th>BPrecRS</th>
<th>BRessRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite</td>
<td>...Ackerman-hard...</td>
<td>Undecidable</td>
<td>Undecidable + semi-algo</td>
<td>on-the-fly 3-EXPTIME</td>
</tr>
<tr>
<td>Infinite</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>Undecidable</td>
<td>3-EXPTIME [D'Souza and Madhusudan, 2002]</td>
</tr>
</tbody>
</table>

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


