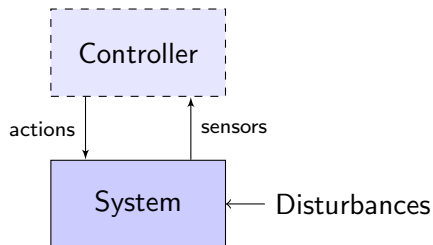


# Controller Synthesis for Timed Systems

Ocan Sankur

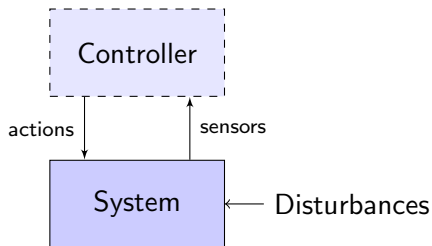
Université de Rennes, CNRS

# Controller Synthesis



**Goal:** Given a system (model), automatically compute a controller to ensure the system always has desired behavior.

# Controller Synthesis for Timed Systems



**Goal:** Given a system (model), automatically compute a controller to ensure the system always has desired behavior.

**Timed Systems:** **Discrete** systems with timing constraints / objectives:

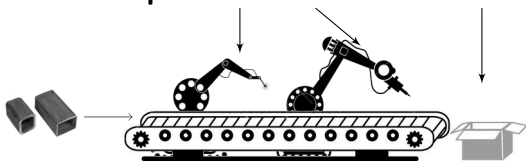
- 1 The system evolves with *discrete events*
- 2 We make the passage of time explicit, e.g. we consider execution time, deadlines etc.

Next: example

# Production Line Example

## Objectives

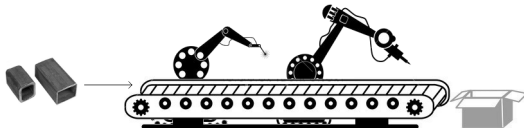
- **Small** or **Large** objects arrive every  $\geq 10$  seconds.
- Small objects must be **painted**, **drilled once**, and **finalized**.
- Large objects must be **painted** **drilled twice** **finalized**.



## Constraints

- The drill must be idle for  $\geq 3$  seconds between each drill.
- No more than one object on the line at any moment.
- Painting, drilling take **1-2 seconds**, and finalizing **1 second**.

# Production Line Example



This is a **timed system**:

- We observe **discrete events**: Object has arrived, object is small, painting starts, painting finishes, etc.
- **Timing constraints**...

**Controller Synthesis**: Find a control policy that dictates which action to take at which moment to satisfy all the constraints.

# Outline

- 1 Modeling Formalism: Finite Games and Timed Games
- 2 Controller Synthesis Algorithm
  - Finite Games
  - Timed Games - Zones
  - Details: Data Structure and Algorithms
- 3 Forward Algorithm (Sketch)
- 4 Extension: Optimal Control
- 5 Extension: Partial Observation

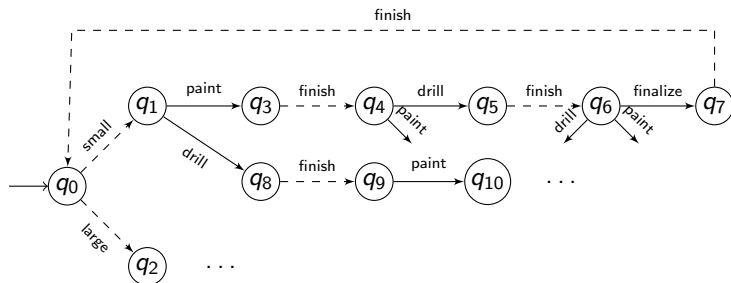
# Outline

- 1 Modeling Formalism: Finite Games and Timed Games
- 2 Controller Synthesis Algorithm
  - Finite Games
  - Timed Games - Zones
  - Details: Data Structure and Algorithms
- 3 Forward Algorithm (Sketch)
- 4 Extension: Optimal Control
- 5 Extension: Partial Observation

# Finite Games

Formalism for **discrete control problems** (no explicit time):

- Nodes are the configurations of the system
- Plain edges: **control actions**
- Dashed edges: **uncontrollable events** (coming from the environment)



**Control Strategy:** A mapping from states to control actions

**Control Objective:** Each arriving object is processed and finalized.

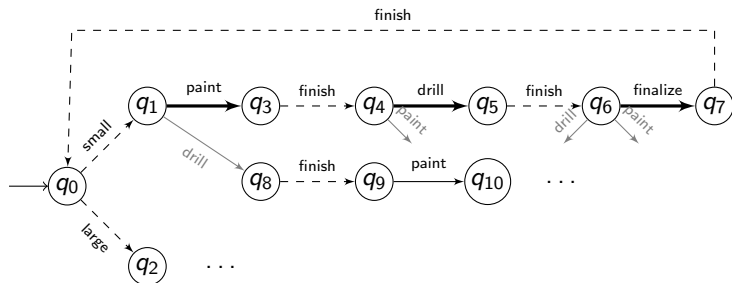
**Winning Strategy:** All executions under the strategy satisfy the objective.



# Finite Games

Formalism for **discrete control problems** (no explicit time):

- Nodes are the configurations of the system
- Plain edges: **control actions**
- Dashed edges: **uncontrollable events** (coming from the environment)



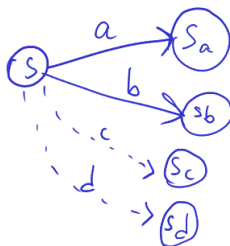
**Control Strategy:** A mapping from states to control actions

**Control Objective:** Each arriving object is processed and finalized.

**Winning Strategy:** All executions under the strategy satisfy the objective.

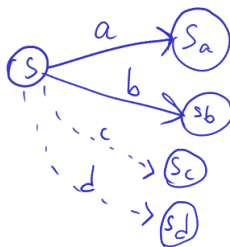
# Finite Games - Semantics

There can be controllable and uncontrollable actions from the same state



# Finite Games - Semantics

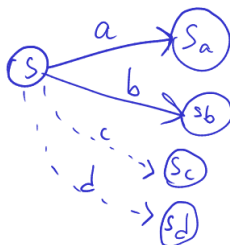
There can be controllable and uncontrollable actions from the same state



- 1 Controller chooses  $a$  or  $b$ . Say  $a$ .

# Finite Games - Semantics

There can be controllable and uncontrollable actions from the same state

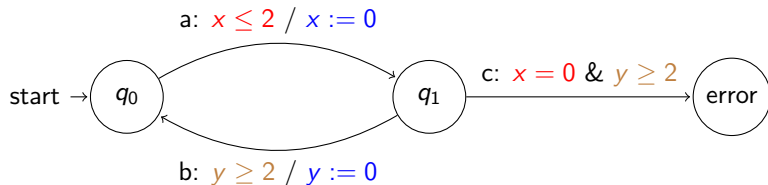


- 1 Controller chooses  $a$  or  $b$ . Say  $a$ .
- 2 The successor is chosen non-deterministically among

$$\{s_a, s_c, s_d\}$$

# Timed Automata

**Timed automata** = Finite automata + Clocks. [Alur and Dill 1994]



## Run of a TA

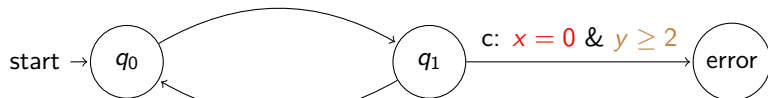
$$\begin{aligned} (q_0, (x = 0, y = 0)) &\xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7)) \\ &\xrightarrow{0.5} (q_1, (x = 0.5, y = 2.2)) \xrightarrow{b} (q_0, (x = 0.5, y = 0)) \dots \end{aligned}$$

A timed automaton is used to describe the **set of all runs**  
all possible behaviors of the system

# Timed Automata

**Timed automata** = Finite automata + Clocks. [Alur and Dill 1994]

a:  $x \leq 2$  /  $x := 0$



b:  $y \geq 2$  /  $y := 0$

c:  $x = 0$  &  $y \geq 2$

## Run of a TA

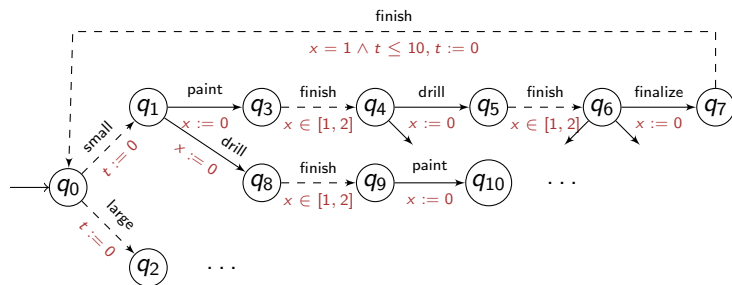
$$\begin{aligned} (q_0, (x = 0, y = 0)) &\xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7)) \\ &\xrightarrow{0.5} (q_1, (x = 0.5, y = 2.2)) \xrightarrow{b} (q_0, (x = 0.5, y = 0)) \dots \end{aligned}$$

Timed Systems can be modeled by timed automata:

- a, b, c are discrete events,
- Clocks are used for timing constraints

→ Let us define **games** played on timed automata

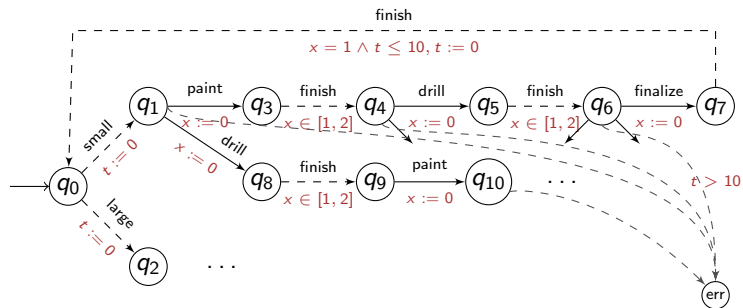
# Timed Games



**Control objective:** Process each object within 10 seconds – against **any** environment behavior (choice of objects, painting durations, etc.)

**Control Strategy:** A function mapping each state to a delay and action to be taken  $(\mathcal{L} \times \mathbb{R}^C \rightarrow \mathbb{R} \times A)$

# Timed Games

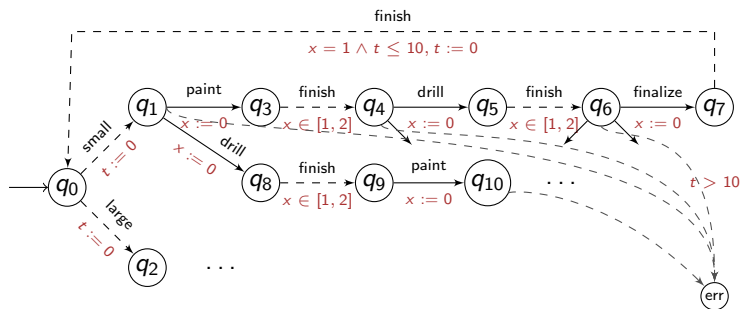


**Control objective:** Process each object within 10 seconds – against **any** environment behavior (choice of objects, painting durations, etc.)

**Control Strategy:** A function mapping each state to a delay and action to be taken  $(\mathcal{L} \times \mathbb{R}^C \rightarrow \mathbb{R} \times A)$



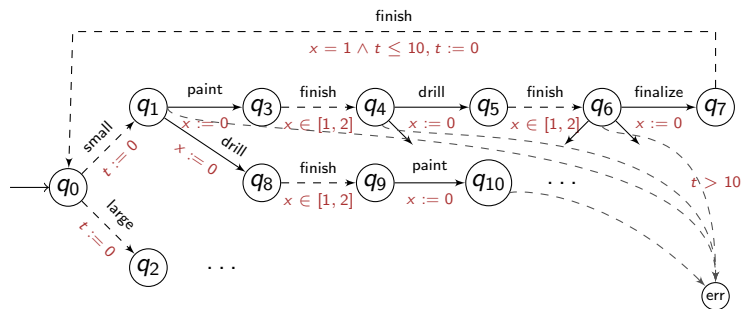
# Timed Games



**Control objective:** Process each object within 10 seconds – against **any** environment behavior (choice of objects, painting durations, etc.)

**Winning Control Strategy:** A function mapping each state to a delay and action to be taken  $(\mathcal{L} \times \mathbb{R}^C \rightarrow \mathbb{R} \times A)$   
such that all generated runs satisfy the objective

# Timed Games

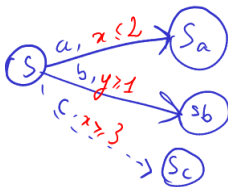


**Control objective:** Process each object within 10 seconds – against **any** environment behavior (choice of objects, painting durations, etc.)

**Winning Control Strategy:** A function mapping each state to a delay and action to be taken  $(\mathcal{L} \times \mathbb{R}^C \rightarrow \mathbb{R} \times A)$   
such that all generated runs satisfy the objective

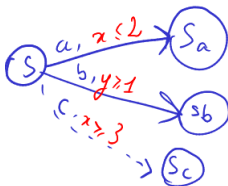
**Goal of this tutorial:** Describe algorithms to solve controller synthesis using timed games

# Timed Games: Semantics



From configuration  $(s, v)$ ,

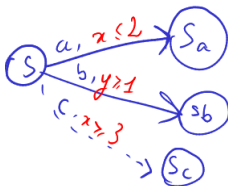
# Timed Games: Semantics



From configuration  $(s, v)$ ,

- 1 Controller chooses delay  $d \geq 0$  and a **controllable** edge  $e \in \{a, b\}$

# Timed Games: Semantics



From configuration  $(s, v)$ ,

- 1 Controller chooses delay  $d \geq 0$  and a **controllable** edge  $e \in \{a, b\}$
- 2 The observed transition is:
  - ▶ either  $(s, v) \xrightarrow{d, e} (s', v')$
  - ▶ or  $(s, v) \xrightarrow{d', e'} (s'', v'')$  where  $d' \leq d$ , and  $e'$  an **uncontrollable** edge

# Outline

- 1 Modeling Formalism: Finite Games and Timed Games
- 2 Controller Synthesis Algorithm
  - Finite Games
  - Timed Games - Zones
  - Details: Data Structure and Algorithms
- 3 Forward Algorithm (Sketch)
- 4 Extension: Optimal Control
- 5 Extension: Partial Observation

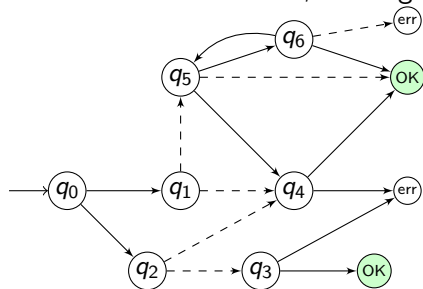
# Outline

## 2 Controller Synthesis Algorithm

- Finite Games
- Timed Games - Zones
- Details: Data Structure and Algorithms

# Algorithm for Finite Games

**Goal:** Reach the OK state, starting from  $q_0$



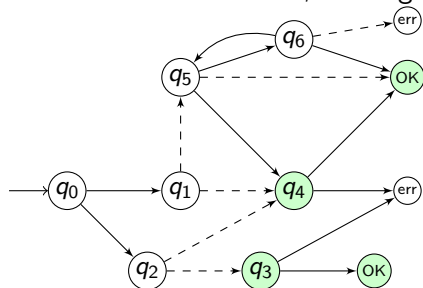
## Algorithm

Step by step: which are the states from which we can win in **1** step?



# Algorithm for Finite Games

**Goal:** Reach the OK state, starting from  $q_0$

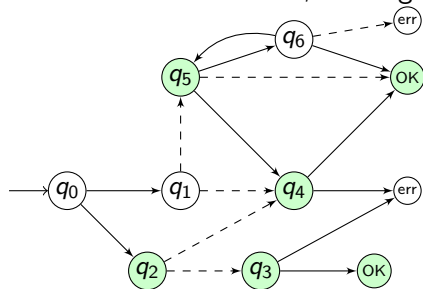


## Algorithm

Step by step: which are the states from which we can win in **1** step?

# Algorithm for Finite Games

**Goal:** Reach the OK state, starting from  $q_0$

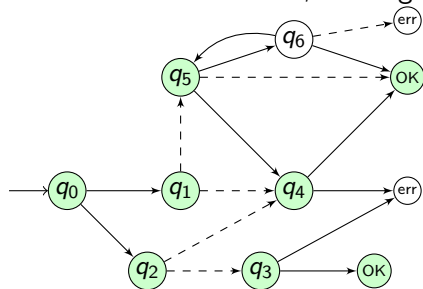


## Algorithm

Step by step: which are the states from which we can win in 2 step?

# Algorithm for Finite Games

**Goal:** Reach the OK state, starting from  $q_0$



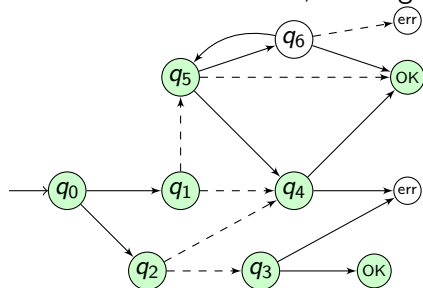
## Algorithm

Step by step: which are the states from which we can win in 3 step?

Green states are **winning states**: there is a winning control strategy

# Algorithm for Finite Games

**Goal:** Reach the OK state, starting from  $q_0$



**Formally, controllable predecessors** of a set of states  $T$  is:

$$\text{CPRE}(T) = \{q \mid \exists q' \in T, (q \rightarrow q' \vee q \dashrightarrow q') \wedge \forall q \dashrightarrow q'', q'' \in T\}.$$

The set of all winning (green) states is obtained as follows:

```
X := {OK};
while CPRE(X) ⊄ X do
    X := X ∪ CPRE(X);
```

# Finite Games

Fixpoint:  $\mu X.(\{\text{OK}\} \cup X \cup \text{CPRE}(X))$ .

## Algorithm

The controller synthesis problem in finite games can be solved in polynomial time for finite games, by computing the fixpoint of CPRE.

Similar results exist of  $\omega$ -regular winning conditions, and temporal logics.

**Memoryless** strategies always exist: Control actions are a function of the current state, not the past history.

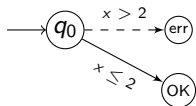
# Outline

## 2 Controller Synthesis Algorithm

- Finite Games
- **Timed Games - Zones**
- Details: Data Structure and Algorithms

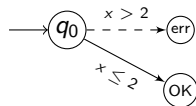
# How About Timed Games?

We want to do the same for **timed games**.



# How About Timed Games?

We want to do the same for **timed games**.



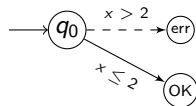
We have  $\text{CPRE}(\{\text{OK}\}) = \{(q_0, x) \mid x \in \mathbb{R}, 0 \leq x \leq 2\}$   
which is **infinite** and **uncountable!**

Enumerating successors or predecessors do not make any sense. . .



# How About Timed Games?

We want to do the same for **timed games**.



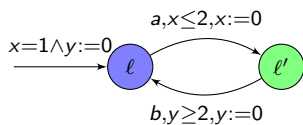
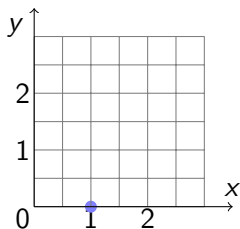
We have  $\text{CPRE}(\{\text{OK}\}) = \{(q_0, x) \mid x \in \mathbb{R}, 0 \leq x \leq 2\}$   
which is **infinite** and **uncountable!**

Enumerating successors or predecessors do not make any sense. . .

→ Data structure to represent the state space of timed games

## Zones - Intuition

**Symbolic** representation: represent **subsets** of states using **polyhedra**  
States are elements of  $\mathcal{L} \times \mathbb{R}_{\geq 0}^C$ , where  $C$  is the set of clocks.

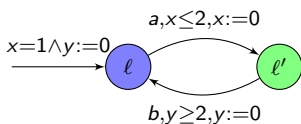
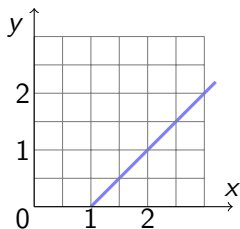


### State space exploration:

Initially, at  $l$ :  $x = 1 \wedge y = 0$

## Zones - Intuition

**Symbolic** representation: represent **subsets** of states using **polyhedra**  
States are elements of  $\mathcal{L} \times \mathbb{R}_{\geq 0}^C$ , where  $C$  is the set of clocks.



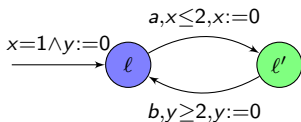
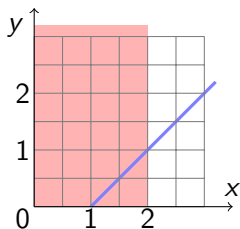
### State space exploration:

Initially, at  $l$ :  $x = 1 \wedge y = 0$

After a time delay, at  $l$ :  $\text{Post}_{\geq 0}(l, x = 1 \wedge y = 0) = (l, x - y = 1)$

## Zones - Intuition

**Symbolic** representation: represent **subsets** of states using **polyhedra**  
States are elements of  $\mathcal{L} \times \mathbb{R}_{\geq 0}^C$ , where  $C$  is the set of clocks.



### State space exploration:

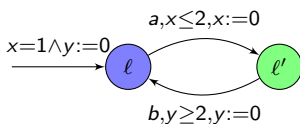
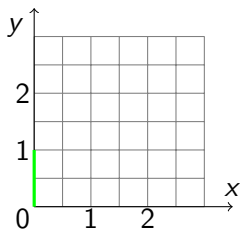
Initially, at  $l$ :  $x = 1 \wedge y = 0$

After a time delay, at  $l$ :  $\text{Post}_{\geq 0}(l, x = 1 \wedge y = 0) = (l, x - y = 1)$

If we move to  $l'$ :

## Zones - Intuition

**Symbolic** representation: represent **subsets** of states using **polyhedra**  
States are elements of  $\mathcal{L} \times \mathbb{R}_{\geq 0}^C$ , where  $C$  is the set of clocks.



### State space exploration:

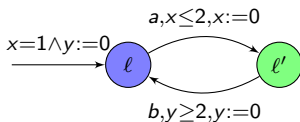
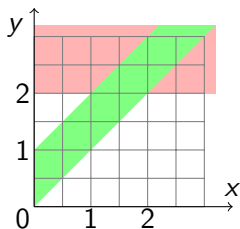
Initially, at  $l$ :  $x = 1 \wedge y = 0$

After a time delay, at  $l$ :  $\text{Post}_{\geq 0}(l, x = 1 \wedge y = 0) = (l, x - y = 1)$

If we move to  $l'$ :  $\text{Post}_a(l, x - y = 1) = (l', 0 \leq y \leq 1 \wedge x = 0)$ ,

## Zones - Intuition

**Symbolic** representation: represent **subsets** of states using **polyhedra**  
States are elements of  $\mathcal{L} \times \mathbb{R}_{\geq 0}^C$ , where  $C$  is the set of clocks.



### State space exploration:

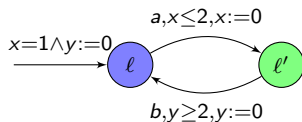
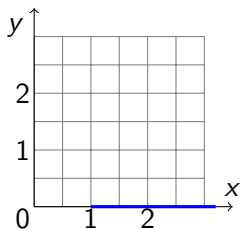
Initially, at  $l$ :  $x = 1 \wedge y = 0$

After a time delay, at  $l$ :  $\text{Post}_{\geq 0}(l, x = 1 \wedge y = 0) = (l, x - y = 1)$

If we move to  $l'$ :  $\text{Post}_a(l, x - y = 1) = (l', 0 \leq y \leq 1 \wedge x = 0)$ , and  
delay:  $0 \leq y - x \leq 1$ .

## Zones - Intuition

**Symbolic** representation: represent **subsets** of states using **polyhedra**  
States are elements of  $\mathcal{L} \times \mathbb{R}_{\geq 0}^C$ , where  $C$  is the set of clocks.



### State space exploration:

Initially, at  $l$ :  $x = 1 \wedge y = 0$

After a time delay, at  $l$ :  $\text{Post}_{\geq 0}(l, x = 1 \wedge y = 0) = (l, x - y = 1)$

If we move to  $l'$ :  $\text{Post}_a(l, x - y = 1) = (l', 0 \leq y \leq 1 \wedge x = 0)$ , and delay:  $0 \leq y - x \leq 1$ .

Move back to  $l$ :  $\text{Post}_b(l', 0 \leq y - x \leq 1) = (l, y = 0 \wedge x \geq 1)$ , etc.

# Timed Automata Verification

Given a sequence of edges  $e_1 \dots e_n$ ,  $\text{Post}_{e_1 \dots e_n}$  is expressible as a zone.

Using zones one can **exhaustively** explore the state space.

Zones were first used for verification by [Berthomieu, Menasche 1983]

Verification questions:

- Does the timed automaton have a run that visits an error state?
- Do all the runs of the timed automaton satisfy a given  $\omega$ -regular condition (e.g. a temporal logic formula)?

## Theorem

Verification of timed automata for safety properties is PSPACE-complete

Same complexity for *omega*-regular properties



# Timed Automata Verification

Given a sequence of edges  $e_1 \dots e_n$ ,  $\text{Post}_{e_1 \dots e_n}$  is expressible as a zone.

Using zones one can **exhaustively** explore the state space.

Zones were first used for verification by [Berthomieu, Menasche 1983]

Verification questions:

- Does the timed automaton have a run that visits an error state?
- Do all the runs of the timed automaton satisfy a given  $\omega$ -regular condition (e.g. a temporal logic formula)?

## Theorem

Verification of timed automata for safety properties is PSPACE-complete

Same complexity for *omega*-regular properties

**Timed games:** Can we define a controllable predecessors operator?

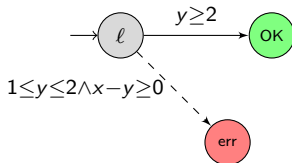
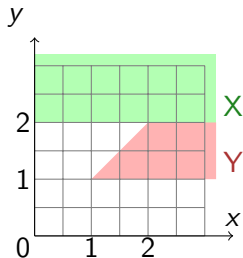
# CPRE for Timed Games

Denote  $Q = \mathcal{L} \times \mathbb{R}_{\geq 0}^C$ .

## Safe Time-Predecessors

Given  $X, Y \subseteq Q$ , from which states can we delay into  $X$  while avoiding  $Y$ ?

$$\text{Pred}_{\geq 0}(X, Y) = \{q \in Q \mid \exists d \geq 0, q + d \in X \wedge \forall d' \in [0, d], q + d' \notin Y\}.$$



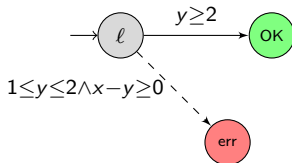
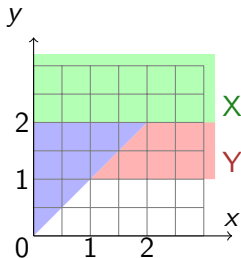
# CPRE for Timed Games

Denote  $Q = \mathcal{L} \times \mathbb{R}_{\geq 0}^C$ .

## Safe Time-Predecessors

Given  $X, Y \subseteq Q$ , from which states can we delay into  $X$  while avoiding  $Y$ ?

$$\text{Pred}_{\geq 0}(X, Y) = \{q \in Q \mid \exists d \geq 0, q + d \in X \wedge \forall d' \in [0, d], q + d' \notin Y\}.$$



# CPRE for Timed Games

Denote  $\mathcal{Q} = \mathcal{L} \times \mathbb{R}_{\geq 0}^C$ .

## Safe Time-Predecessors

Given  $X, Y \subseteq \mathcal{Q}$ , from which states can we delay into  $X$  while avoiding  $Y$ ?

$$\text{Pred}_{\geq 0}(X, Y) = \{q \in \mathcal{Q} \mid \exists d \geq 0, q + d \in X \wedge \forall d' \in [0, d], q + d' \notin Y\}.$$

## Control and Environment Predecessors

Given  $X \subseteq \mathcal{Q}$ , let us define

$$\text{Pred}_c(X) = \{q \in \mathcal{Q} \mid \exists q' \in X, \exists c, q \xrightarrow{c} q'\}, \quad (1)$$

$$\text{Pred}_u(X) = \{q \in \mathcal{Q} \mid \exists q' \in X, \exists u, q \xrightarrow{u} q'\}. \quad (2)$$

# CPRE for Timed Games

## Safe Time-Predecessors

Given  $X, Y \subseteq \mathcal{Q}$ , from which states can we delay into  $X$  while avoiding  $Y$ ?

$$\text{Pred}_{\geq 0}(X, Y) = \{q \in \mathcal{Q} \mid \exists d \geq 0, q + d \in X \wedge \forall d' \in [0, d], q + d' \notin Y\}.$$

## Control and Environment Predecessors

Given  $X \subseteq \mathcal{Q}$ , let us define

$$\text{Pred}_c(X) = \{q \in \mathcal{Q} \mid \exists q' \in X, \exists c, q \xrightarrow{c} q'\}, \quad (1)$$

$$\text{Pred}_u(X) = \{q \in \mathcal{Q} \mid \exists q' \in X, \exists u, q \xrightarrow{u} q'\}. \quad (2)$$

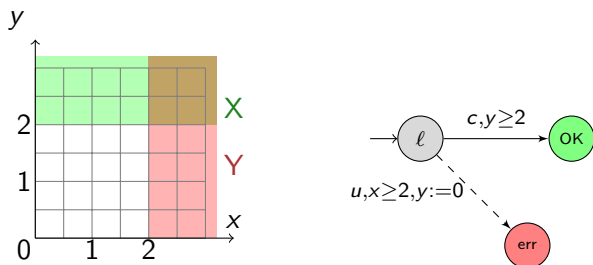
Define

$$\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(\mathcal{Q} \setminus X)).$$

“The states from which one can delay into  $\text{Pred}_c(X)$  (and go to  $X$ ), while avoiding environment’s actions into  $\mathcal{Q} \setminus X$ .”

# CPRE for Timed Games

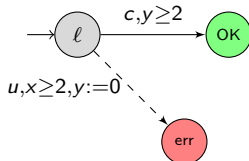
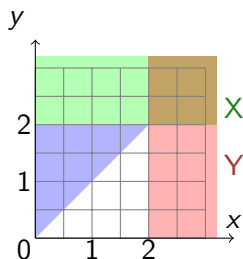
$$\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(Q \setminus X)).$$



- $X = \text{OK} \times \mathbb{R}_{\geq 0}^C$ ,
- $\text{Pred}_c(\text{OK} \times \mathbb{R}_{\geq 0}^C) = (l, y \geq 2)$ ,
- $\text{Pred}_u(Q \setminus X) = \text{Pred}_u(\{\text{err}, l\} \times \mathbb{R}_{\geq 0}^C) = (l, x \geq 2)$ .

## CPRE for Timed Games

$$\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(Q \setminus X)).$$



- $X = \text{OK} \times \mathbb{R}_{\geq 0}^C$ ,
- $\text{Pred}_c(\text{OK} \times \mathbb{R}_{\geq 0}^C) = (l, y \geq 2)$ ,
- $\text{Pred}_u(Q \setminus X) = \text{Pred}_u(\{\text{err}, l\} \times \mathbb{R}_{\geq 0}^C) = (l, x \geq 2)$ .

# CPRE for Timed Games

$$\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(Q \setminus X)).$$

## Solving Timed Games

The following fixpoint is the set of states from which Controller can ensure reaching  $T$ :  $\mu X.(T \cup X \cup \text{CPRE}(X))$ .

In other terms, the algorithm is the following.

```
X := T
while X ≠ X ∪ CPRE(X) do
  X := X ∪ CPRE(X)
```

## Complexity

Determining if Controller has a winning strategy from a given state in a timed game is EXPTIME-complete.



# CPRE for Timed Games

$$\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(Q \setminus X)).$$

## Solving Timed Games

The following fixpoint is the set of states from which Controller can ensure

How do we actually compute  $\text{CPRE}(X)$ ?

Which data structure?

Which precise algorithm?

while  $X \neq X \cup \text{CPRE}(X)$  do  
 $X := X \cup \text{CPRE}(X)$

## Complexity

Determining if Controller has a winning strategy from a given state in a timed game is EXPTIME-complete.

# Outline

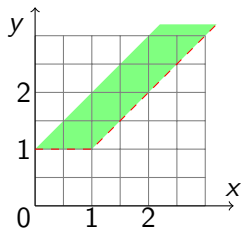
## 2 Controller Synthesis Algorithm

- Finite Games
- Timed Games - Zones
- Details: Data Structure and Algorithms

## Difference-Bound Matrices (DBM)

Consider clocks  $x, y$  and the following set of states:

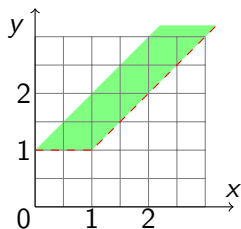
$$x < y \quad \wedge \quad y \leq x + 1 \quad \wedge \quad x \geq 0 \quad \wedge \quad y > 1$$



## Difference-Bound Matrices (DBM)

Consider clocks  $x, y$  and the following set of states:

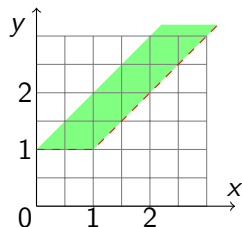
$$x - y < 0 \quad \wedge \quad y - x \leq 1 \quad \wedge \quad -x \leq 0 \quad \wedge \quad -y < -1$$



# Difference-Bound Matrices (DBM)

Consider clocks  $x, y$  and the following set of states:

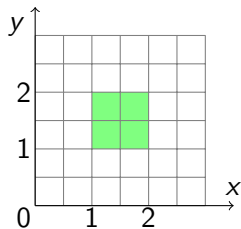
$$x - y < 0 \quad \wedge \quad y - x \leq 1 \quad \wedge \quad -x \leq 0 \quad \wedge \quad -y < -1$$



	0	x	y
0	$0, \leq$	$0, \leq$	$-1, <$
x	$\infty, <$	$0, \leq$	$0, <$
y	$\infty, <$	$1, \leq$	$0, \leq$

# Difference-Bound Matrices (DBM)- Canonical Form

$$x \leq 2 \quad \wedge \quad -x \leq -1 \quad \wedge \quad y \leq 2 \quad \wedge \quad -y \leq -1$$



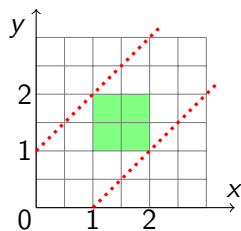
$$\begin{array}{c} 0 \\ x \\ y \end{array} \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 2, \leq & 0, \leq & \infty, < \\ 2, \leq & \infty, < & 0, \leq \end{pmatrix}$$

**DBM in canonical form:** All constraints are tight.

Any given DBM can be made canonical in time  $O(C^3)$ .

# Difference-Bound Matrices (DBM)- Canonical Form

$$x \leq 2 \quad \wedge \quad -x \leq -1 \quad \wedge \quad y \leq 2 \quad \wedge \quad -y \leq -1$$



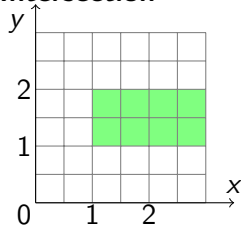
$$\begin{array}{c} 0 \\ x \\ y \end{array} \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 2, \leq & 0, \leq & 1, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix}$$

**DBM in canonical form:** All constraints are tight.

Any given DBM can be made canonical in time  $O(C^3)$ .

# Difference-Bound Matrices (DBM) - Operations

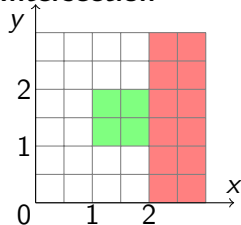
## Intersection





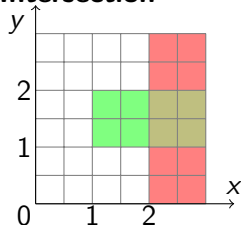
# Difference-Bound Matrices (DBM) - Operations

## Intersection



# Difference-Bound Matrices (DBM) - Operations

## Intersection



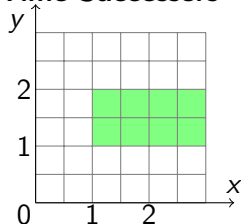
Idea:  $y \leq 2$  **and**  $y < 3 = y \leq 2$ .

Given two DBMs  $M, N$ ,  $M \cap N$  is defined by

$$\forall i, j \in \{0, 1, \dots, C\}, (M \cap N)_{i,j} = \min(M_{i,j}, N_{i,j}).$$

# Difference-Bound Matrices (DBM) - Operations

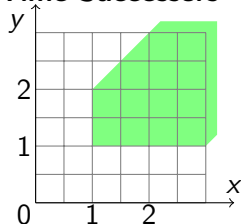
## Time Successors



$$\begin{array}{c} 0 \\ x \\ y \end{array} \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 3, \leq & 0, \leq & 2, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix}$$

# Difference-Bound Matrices (DBM) - Operations

## Time Successors

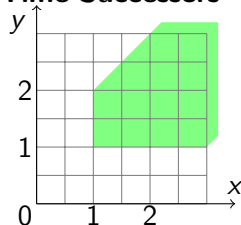


$$\begin{array}{c} 0 \\ x \\ y \end{array} \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 3, \leq & 0, \leq & 2, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix}$$

**Idea:** Remove all upper bounds on clocks:  $x \leq 3 \rightarrow x < \infty$ .

# Difference-Bound Matrices (DBM) - Operations

## Time Successors

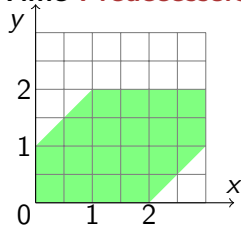


$$\begin{array}{c} 0 \\ x \\ y \end{array} \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ \infty, < & 0, \leq & 2, \leq \\ \infty, < & 1, \leq & 0, \leq \end{pmatrix}$$

**Idea:** Remove all upper bounds on clocks:  $x \leq 3 \rightarrow x < \infty$ .

# Difference-Bound Matrices (DBM) - Operations

## Time Predecessors

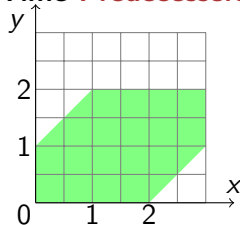


$$\begin{array}{c} 0 \\ x \\ y \end{array} \begin{pmatrix} 0, \leq & 0, \leq & 0, \leq \\ 3, \leq & 0, \leq & 2, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix}$$

**Idea:** Remove all **lower** bounds on clocks:  $x \geq 1 \rightarrow x \geq 0$ .

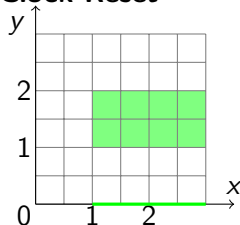
# Difference-Bound Matrices (DBM) - Operations

## Time Predecessors



$$\begin{matrix} & 0 & x & y \\ \begin{matrix} 0 \\ x \\ y \end{matrix} & \left( \begin{matrix} 0, \leq & 0, \leq & 0, \leq \\ 3, \leq & 0, \leq & 2, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{matrix} \right) \end{matrix}$$

## Clock Reset



**Idea:** Remove any constraint on reset clock  $y$ , and intersect with the DBM  $y = 0$ .

# DBM Operations

- DBMs represent **convex** subsets of states
- Inclusion and Equality can be checked as well (not shown here)
- Successor and predecessors (through a given edge) can be computed efficiently  $O(C^3)$

e.g. for successor of set  $M$  through an edge with guard  $G$ , resetting  $x$ :

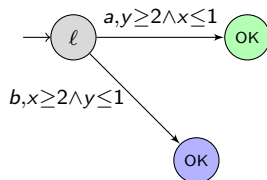
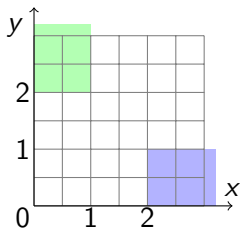
$$\text{Post}_e(M) = \text{Reset}_x(\text{Post}_{\geq 0}(M) \cap G)$$

Can we compute CPRE with DBMs?



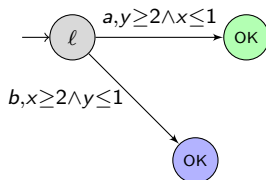
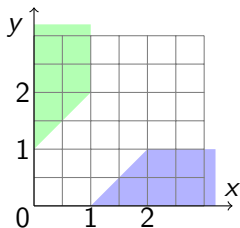
## CPRE is not convex

Computing  $\text{CPRE}(\{\text{OK}\}) = \text{Pred}_{\geq 0}(\text{Pred}_c(\{\text{OK}\}), \emptyset)$ .



## CPRE is not convex

Computing  $\text{CPRE}(\{\text{OK}\}) = \text{Pred}_{\geq 0}(\text{Pred}_c(\{\text{OK}\}), \emptyset)$ .



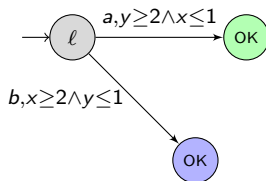
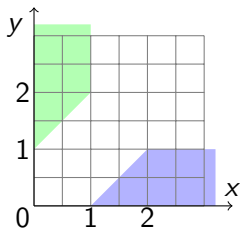
Non-convex...

Moreover we also need **complementation** which is also a non-convex operation

recall the definition  $\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(\mathcal{Q} \setminus X))$

## CPRE is not convex

Computing  $\text{CPRE}(\{\text{OK}\}) = \text{Pred}_{\geq 0}(\text{Pred}_c(\{\text{OK}\}), \emptyset)$ .



Non-convex...

Moreover we also need **complementation** which is also a non-convex operation

recall the definition  $\text{CPRE}(X) = \text{Pred}_{\geq 0}(\text{Pred}_c(X), \text{Pred}_u(\mathcal{Q} \setminus X))$

## DBM Federation

A list of DBMs which represent the **union** of each DBM

# DBM Federations – Operations

Some easy operations:

- $(M_1 \cup \dots \cup M_k) \cap (N_1 \cup \dots \cup N_l) = \cup_{i,j} M_i \cap N_j.$

# DBM Federations – Operations

Some easy operations:

- $(M_1 \cup \dots \cup M_k) \cap (N_1 \cup \dots \cup N_l) = \cup_{i,j} M_i \cap N_j$ .
- $\text{Post}_{\geq 0}(M_1 \cup \dots \cup M_k) = \text{Post}_{\geq 0}(M_1) \cup \dots \cup \text{Post}_{\geq 0}(M_k)$ .

# DBM Federations – Operations

Some easy operations:

- $(M_1 \cup \dots \cup M_k) \cap (N_1 \cup \dots \cup N_l) = \cup_{i,j} M_i \cap N_j$ .
- $\text{Post}_{\geq 0}(M_1 \cup \dots \cup M_k) = \text{Post}_{\geq 0}(M_1) \cup \dots \cup \text{Post}_{\geq 0}(M_k)$ .
- $\text{Reset}_x(M_1 \cup \dots \cup M_k) = \text{Reset}_x(M_1) \cup \dots \cup \text{Reset}_x(M_k)$ .

# DBM Federations – Operations

Some easy operations:

- $(M_1 \cup \dots \cup M_k) \cap (N_1 \cup \dots \cup N_l) = \cup_{i,j} M_i \cap N_j$ .
- $\text{Post}_{\geq 0}(M_1 \cup \dots \cup M_k) = \text{Post}_{\geq 0}(M_1) \cup \dots \cup \text{Post}_{\geq 0}(M_k)$ .
- $\text{Reset}_x(M_1 \cup \dots \cup M_k) = \text{Reset}_x(M_1) \cup \dots \cup \text{Reset}_x(M_k)$ .

New: **Complementation**

- $(N_1 \cup \dots \cup N_k)^c = N_1^c \cap \dots \cap N_k^c$ .

# DBM Federations – Operations

Some easy operations:

- $(M_1 \cup \dots \cup M_k) \cap (N_1 \cup \dots \cup N_l) = \cup_{i,j} M_i \cap N_j$ .
- $\text{Post}_{\geq 0}(M_1 \cup \dots \cup M_k) = \text{Post}_{\geq 0}(M_1) \cup \dots \cup \text{Post}_{\geq 0}(M_k)$ .
- $\text{Reset}_x(M_1 \cup \dots \cup M_k) = \text{Reset}_x(M_1) \cup \dots \cup \text{Reset}_x(M_k)$ .

New: **Complementation**

- $(N_1 \cup \dots \cup N_k)^c = N_1^c \cap \dots \cap N_k^c$ .

To compute  $N^c$ ,

- Write  $N = C_1 \wedge C_2 \wedge \dots \wedge C_m$ , and  $N^c = C_1^c \vee \dots \vee C_m^c$ .
- $N^c$  is the federation:  $C_1^c \cup \dots \cup C_m^c$ .

**Example:**  $N = x \leq 2 \wedge y > 1 \rightarrow N^c = ((x > 2) \cup (y \leq 1))$ .



# DBM Federations – Operations

Some easy operations:

- $(M_1 \cup \dots \cup M_k) \cap (N_1 \cup \dots \cup N_l) = \cup_{i,j} M_i \cap N_j$ .
- $\text{Post}_{\geq 0}(M_1 \cup \dots \cup M_k) = \text{Post}_{\geq 0}(M_1) \cup \dots \cup \text{Post}_{\geq 0}(M_k)$ .
- $\text{Reset}_x(M_1 \cup \dots \cup M_k) = \text{Reset}_x(M_1) \cup \dots \cup \text{Reset}_x(M_k)$ .

New: **Complementation**

- $(N_1 \cup \dots \cup N_k)^c = N_1^c \cap \dots \cap N_k^c$ .

To compute  $N^c$ ,

- Write  $N = C_1 \wedge C_2 \wedge \dots \wedge C_m$ , and  $N^c = C_1^c \vee \dots \vee C_m^c$ .
- $N^c$  is the federation:  $C_1^c \cup \dots \cup C_m^c$ .

**Example:**  $N = x \leq 2 \wedge y > 1 \rightarrow N^c = ((x > 2) \cup (y \leq 1))$ .

Complementation and intersection are expensive...

► But there are a lot of heuristics to reduce redundant constraints, and minimize the size of federations...

# Algorithm

Computation of the winning states using DBM federations:

```
 $X := T$   
while  $X \neq X \cup \text{CPRE}(X)$  do  
   $X := X \cup \text{CPRE}(X)$ 
```

# Algorithm

Computation of the winning states using DBM federations:

```
 $X := T$   
while  $X \neq X \cup \text{CPRE}(X)$  do  
   $X := X \cup \text{CPRE}(X)$ 
```

- In this computation, federations can sometimes become too big to handle... unions of thousands of DBMs in each federation

# Algorithm

Computation of the winning states using DBM federations:

$$\begin{aligned} X &:= T \\ \text{while } X &\neq X \cup \text{CPRE}(X) \text{ do} \\ &X := X \cup \text{CPRE}(X) \end{aligned}$$

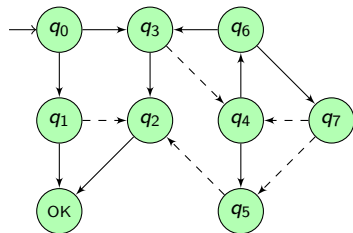
- In this computation, federations can sometimes become too big to handle. . . unions of thousands of DBMs in each federation
- This is a **backward** algorithm that computes **all** winning states

If we are interested in **some** winning strategy, do we actually need to compute all winning states?

# Outline

- 1 Modeling Formalism: Finite Games and Timed Games
- 2 Controller Synthesis Algorithm
  - Finite Games
  - Timed Games - Zones
  - Details: Data Structure and Algorithms
- 3 Forward Algorithm (Sketch)
- 4 Extension: Optimal Control
- 5 Extension: Partial Observation

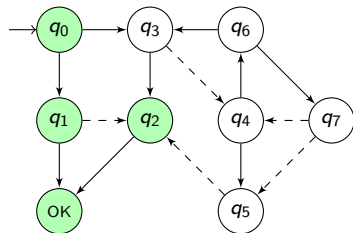
## Smaller Solutions



- Backwards algorithm computes  $X_1, \dots, X_n$  such that

$$X_1 = \{\text{OK}\}, X_2 = \text{CPRE}(X_1), X_3 = \text{CPRE}(X_1 \cup X_2), \dots$$

## Smaller Solutions



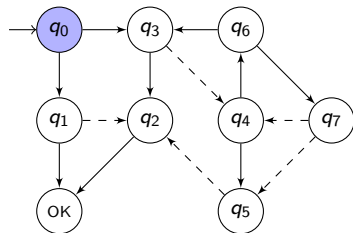
- Backwards algorithm computes  $X_1, \dots, X_n$  such that

$$X_1 = \{\text{OK}\}, X_2 = \text{CPRE}(X_1), X_3 = \text{CPRE}(X_1 \cup X_2), \dots$$

- A smaller solution could be given with  $Y_1 = \{\text{OK}\}, Y_2, Y_3, \dots$ , and

$$\forall i \geq 2, Y_i \subseteq \text{CPRE}(Y_1 \cup \dots \cup Y_{i-1}).$$

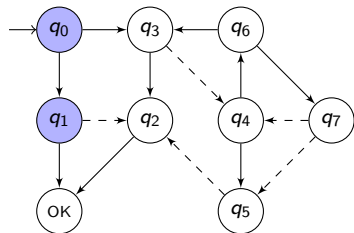
## Forward Algorithm for Finite Games



- Explore forward (e.g. depth-first search) until you hit a target state

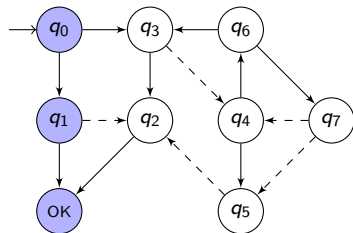


## Forward Algorithm for Finite Games



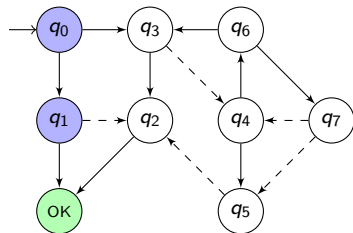
- Explore forward (e.g. depth-first search) until you hit a target state

# Forward Algorithm for Finite Games

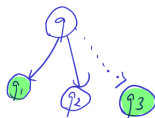


- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**

# Forward Algorithm for Finite Games

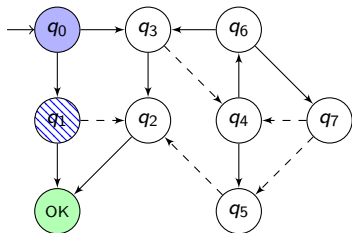


- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**

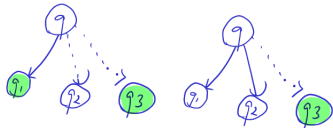


- ▶ If we are not sure, then explore its successors
- ▶ If we cannot win, then mark as **losing**

# Forward Algorithm for Finite Games

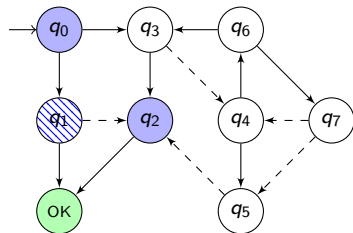


- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors



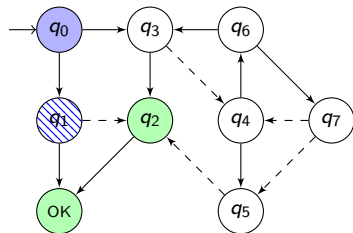
- ▶ If we cannot win, then mark as **losing**

# Forward Algorithm for Finite Games



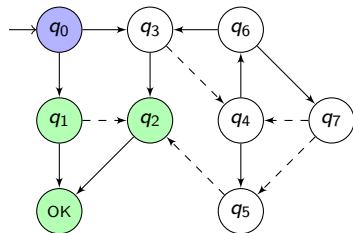
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**

# Forward Algorithm for Finite Games



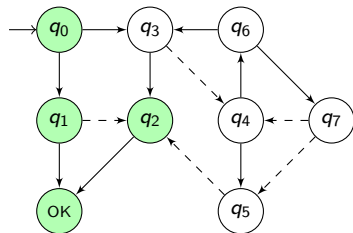
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**

# Forward Algorithm for Finite Games



- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**

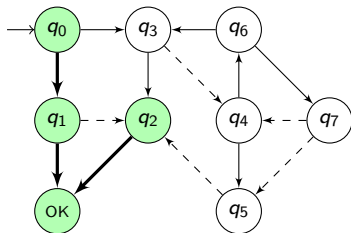
# Forward Algorithm for Finite Games



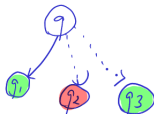
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**



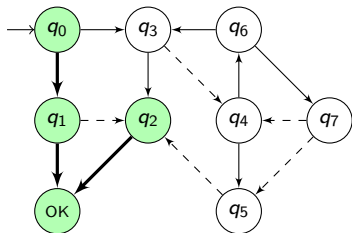
# Forward Algorithm for Finite Games



- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**



# Forward Algorithm for Finite Games



- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state  $q$ :
  - ▶ If we can conclude that  $q$  is winning, then mark as **winning**
  - ▶ If we are not sure, then explore its successors
  - ▶ If we cannot win, then mark as **losing**

**Timed games:** Apply same idea:

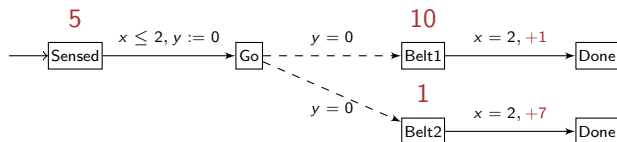
- Forward exploration using zones
- Whenever a target state is found, **backtrack**:
  - ▶ Explore all uncontrollable edges that haven't been explored
  - ▶ Use the CPRE operator to backtrack

# Outline

- 1 Modeling Formalism: Finite Games and Timed Games
- 2 Controller Synthesis Algorithm
  - Finite Games
  - Timed Games - Zones
  - Details: Data Structure and Algorithms
- 3 Forward Algorithm (Sketch)
- 4 Extension: Optimal Control
- 5 Extension: Partial Observation

# Optimal Controllers

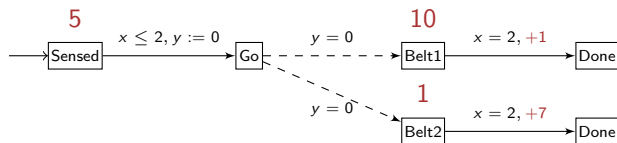
The previously presented algorithms compute an **arbitrary** controller. Assume we want to compute the cheapest, or the fastest controller.



**Weighted** timed games: the cost of spending  $t$  time units at location  $\ell$  is  $C(\ell) \cdot t$ , and each transition has a cost.

# Optimal Controllers

The previously presented algorithms compute an **arbitrary** controller. Assume we want to compute the cheapest, or the fastest controller.

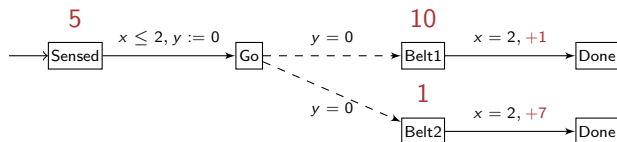


**Weighted** timed games: the cost of spending  $t$  time units at location  $\ell$  is  $C(\ell) \cdot t$ , and each transition has a cost.

**Problem:** Is there a controller that ensures reaching the target with accumulated cost at most  $b$ ?

# Optimal Controllers

The previously presented algorithms compute an **arbitrary** controller. Assume we want to compute the cheapest, or the fastest controller.



**Weighted** timed games: the cost of spending  $t$  time units at location  $\ell$  is  $C(\ell) \cdot t$ , and each transition has a cost.

**Problem:** Is there a controller that ensures reaching the target with accumulated cost at most  $b$ ?

Cost of the path through Belt1:  $5t + 10(2 - t) + 1$

Cost of the path through Belt2:  $5t + (2 - t) + 7$

The optimal cost:

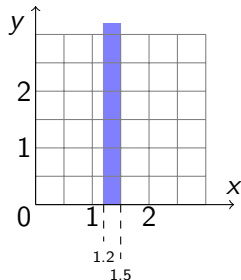
$$14 + \frac{1}{3} = \min_{t \in [0, 2]} (\max(5t + 10(2 - t) + 1, 5t + (2 - t) + 7)).$$

## Optimal Controllers - 2

$$14 + \frac{1}{3} = \min_{t \in [0,2]} (\max(5t + 10(2-t) + 1, 5t + (2-t) + 7)).$$

the min. is achieved for  $t = \frac{4}{3}$ .

Assume we want to reach the target with cost  $\leq 15$ .

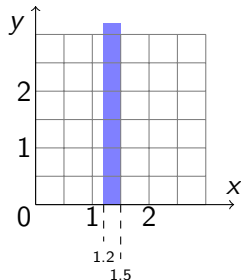


## Optimal Controllers - 2

$$14 + \frac{1}{3} = \min_{t \in [0,2]} (\max(5t + 10(2-t) + 1, 5t + (2-t) + 7)).$$

the min. is achieved for  $t = \frac{4}{3}$ .

Assume we want to reach the target with cost  $\leq 15$ .



- Zones are not precise enough to represent the state space.
- Weighted timed games require arbitrarily precise polyhedra...

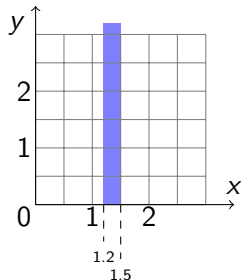


## Optimal Controllers - 2

$$14 + \frac{1}{3} = \min_{t \in [0,2]} (\max(5t + 10(2-t) + 1, 5t + (2-t) + 7)).$$

the min. is achieved for  $t = \frac{4}{3}$ .

Assume we want to reach the target with cost  $\leq 15$ .



- Zones are not precise enough to represent the state space.
- Weighted timed games require arbitrarily precise polyhedra...

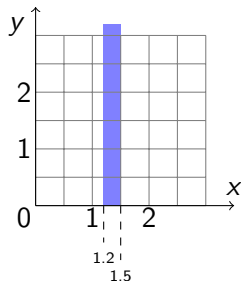
→ Fixpoint computation does not terminate in general

## Optimal Controllers - 2

$$14 + \frac{1}{3} = \min_{t \in [0,2]} (\max(5t + 10(2-t) + 1, 5t + (2-t) + 7)).$$

the min. is achieved for  $t = \frac{4}{3}$ .

Assume we want to reach the target with cost  $\leq 15$ .



- Zones are not precise enough to represent the state space.
- Weighted timed games require arbitrarily precise polyhedra...

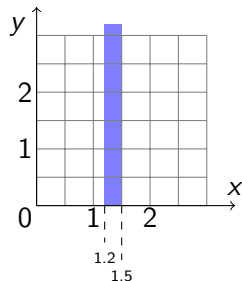
- Fixpoint computation does not terminate in general
- **Undecidable** but decidable for some classes with a single clock, or under restrictions on used weights
- Optimal cost can be approximated under some assumptions

## Optimal Controllers - 2

$$14 + \frac{1}{3} = \min_{t \in [0,2]} (\max(5t + 10(2-t) + 1, 5t + (2-t) + 7)).$$

the min. is achieved for  $t = \frac{4}{3}$ .

Assume we want to reach the target with cost  $\leq 15$ .



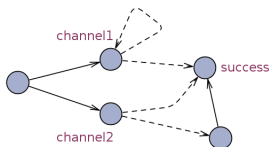
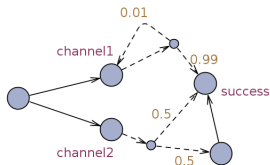
- Zones are not precise enough to represent the state space.
- Weighted timed games require arbitrarily precise polyhedra...

- Fixpoint computation does not terminate in general
- **Undecidable** but decidable for some classes with a single clock, or under restrictions on used weights
- Optimal cost can be approximated under some assumptions
- **Decidable** and easy with weights on edges only

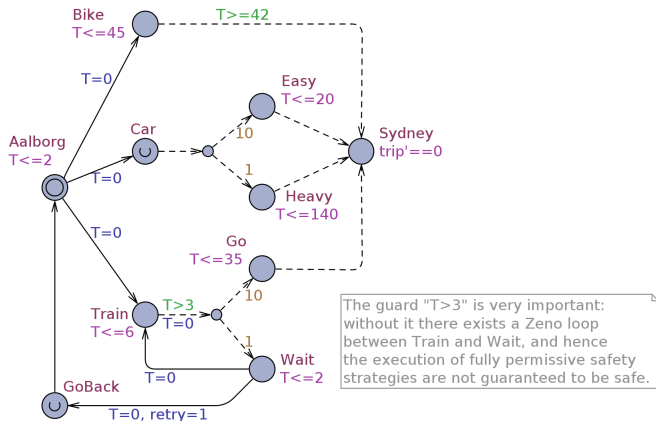
# Min. Expected Cost with Worst-case Guarantees

Assume that all durations and choice of transitions of the **opponent** follow **probability distributions**

- **Hard guarantees:** Safety property must hold when the system is seen as a timed game **against all adversaries**
- **Soft guarantees:** Minimize **expected** cost subject to hard guarantees



# Example



The opponent draws target duration from uniform distributions

# Algorithm

Two-step procedure (for safety):

- 1 Compute the **most permissive** strategy for safety
- 2 Define a new game in which **all** behaviors are **safe**
- 3 Use **reinforcement learning** techniques to optimize the expected cost

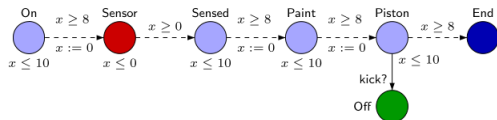
# Outline

- 1 Modeling Formalism: Finite Games and Timed Games
- 2 Controller Synthesis Algorithm
  - Finite Games
  - Timed Games - Zones
  - Details: Data Structure and Algorithms
- 3 Forward Algorithm (Sketch)
- 4 Extension: Optimal Control
- 5 Extension: Partial Observation

# Partial Observation

Previous algorithms assumed that the controller has **precise knowledge** about the system's state, at any moment.

Consider the following system in which the controller can only **sense** different colors:

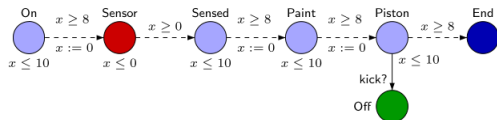




# Partial Observation

Previous algorithms assumed that the controller has **precise knowledge** about the system's state, at any moment.

Consider the following system in which the controller can only **sense** different colors:

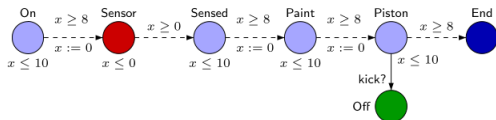


The controller has **partial information** about the system's state.

# Partial Observation



Previous algorithms assumed that the controller has **precise knowledge** about the system's state, at any moment.

Consider the following system in which the controller can only **sense** different colors:



The controller has **partial information** about the system's state.

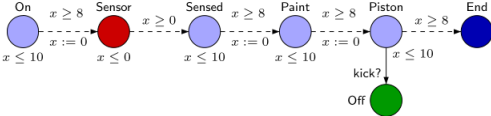
The controller only observes **changes** in the observation:

1. 
2. 

# Partial Observation

Previous algorithms assumed that the controller has **precise knowledge** about the system's state, at any moment.

Consider the following system in which the controller can only **sense** different colors:



The controller has **partial information** about the system's state.

The controller only observes **changes** in the observation:

- 1.
- 2.

► The object is definitely in the Piston area between 20-24 seconds after Sensed.

## Partial Observation: Details

Actual execution:

$$\begin{aligned} &(\text{On}, x = 0) \xrightarrow{8} (\text{On}, x = 8) \rightarrow (\text{Sensor}, x = 0) \\ &\quad \rightarrow (\text{Sensed}, x = 0) \xrightarrow{9} (\text{Sensed}, x = 9) \rightarrow (\text{Paint}, x = 0) \dots \end{aligned}$$

## Partial Observation: Details

Actual execution:

$$\begin{aligned} &(\text{On}, x = 0) \xrightarrow{8} (\text{On}, x = 8) \rightarrow (\text{Sensor}, x = 0) \\ &\quad \rightarrow (\text{Sensed}, x = 0) \xrightarrow{9} (\text{Sensed}, x = 9) \rightarrow (\text{Paint}, x = 0) \dots \end{aligned}$$



Observations:

The same observation is not repeated

Clocks are not observed

# Partial Observation: Details

Actual execution:

$$\begin{aligned} & (\text{On}, x = 0) \xrightarrow{8} (\text{On}, x = 8) \rightarrow (\text{Sensor}, x = 0) \\ & \quad \rightarrow (\text{Sensed}, x = 0) \xrightarrow{9} (\text{Sensed}, x = 9) \rightarrow (\text{Paint}, x = 0) \dots \end{aligned}$$



Observations:

The same observation is not repeated

Clocks are not observed

At each **new** observation:

- 1 Controller chooses either an edge  $e$ , or to wait ( $\lambda$ )
  - ▶ If  $e$  is chosen: Environment can let time elapse as long as  $e$  is not enabled, or take any action from  $e \cup \text{Act}_u$
  - ▶ If  $\lambda$  is chosen: Environment can let time elapse or take  $\text{Act}_u$  **until there is a new observation**
- 2 Controller chooses again at the first instant a new observation arrives

## Partial Observation: Details

One can build an equivalent **finite** game on zones

- Zone-based fixpoint algorithms can be adapted
- Forward algorithm can be used as well!

See F. Cassez FORMATS 2007

# Conclusion

## Uses / Applications

- Programmable climate control: the user sets hard temperature constraints, and desired average for each room FORMATS 2007



# Conclusion

## Uses / Applications

- Programmable climate control: the user sets hard temperature constraints, and desired average for each room FORMATS 2007
- High-level mission planning for robotics:  
e.g. find the order in which rooms are to be visited, adapting to closed doors and obstacles. Low-level controller is computed separately based on this mission.  
<https://www.verifiablerobotics.com/>

# Conclusion

## Uses / Applications

- Programmable climate control: the user sets hard temperature constraints, and desired average for each room FORMATS 2007
- High-level mission planning for robotics:  
e.g. find the order in which rooms are to be visited, adapting to closed doors and obstacles. Low-level controller is computed separately based on this mission.  
<https://www.verifiablerobotics.com/>
- Complex scheduling problems: QFM 2009
  - processes to be scheduled have internal states (such as programs),
  - uncertainties (e.g. nondeterministic durations within given intervals)
  - sporadic tasks (e.g. nondeterministic interarrival times, depending on internal states)

# Conclusion

## Uses / Applications

- Programmable climate control: the user sets hard temperature constraints, and desired average for each room FORMATS 2007
- High-level mission planning for robotics:  
e.g. find the order in which rooms are to be visited, adapting to closed doors and obstacles. Low-level controller is computed separately based on this mission.  
<https://www.verifiablerobotics.com/>
- Complex scheduling problems: QFM 2009
  - processes to be scheduled have internal states (such as programs),
  - uncertainties (e.g. nondeterministic durations within given intervals)
  - sporadic tasks (e.g. nondeterministic interarrival times, depending on internal states)

Controller Synthesis or AI Planning?

# Conclusion

## Timed Games

- Formalism to model timed systems and controller synthesis problems.
- Backward algorithm: simple and computes **all** winning states  
Forward algorithm: more efficient in general to compute **some** winning strategy
- Presented algorithms are based on zones and are **semi-symbolic**:  
Discrete states are enumerated one by one, but sets of clock valuations are efficiently represented by zones
  - ▶ Good efficiency when small number of discrete states, but complex timing constraints (e.g. big constants)
  - ▶ Poor efficiency when a large number of discrete states
- Extensions: partial-observation, weights. . .
- Tool support: Uppaal-TIGA, Uppaal-STRATEGO, TChecker ?

# References

## Timed automata and games

- Alur, Dill. A theory of timed automata. 1994.
- Asarin, O. Maler and A.Pnueli, Symbolic Controller Synthesis for Discrete and Timed Systems. 1995.
- E. Asarin, O. Maler, A.Pnueli, J. Sifakis, Controller Synthesis for Timed Automata. 1998.

## Zone-based exploration

- Berthomieu, Menasch. An Enumerative Approach For Analyzing Time Petri Nets. 1983.
- Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. 1989.
- Bellman. Dynamic Programming. 1957. **(Difference-Bound Matrices)**
- Bengtsson, Yi. Timed Automata: Semantics, Algorithms and Tools. 2004. **(Survey)**

## Controller Synthesis Algorithms using Zones

- Tripakis, Altisen. On-the-Fly Controller Synthesis for Discrete and Timed Systems. 1999.
- Cassez, David, Fleury, Larsen, Lime. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. 2005.

## Forward Algorithm

- Liu, Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points (Extended Abstract). 1998.