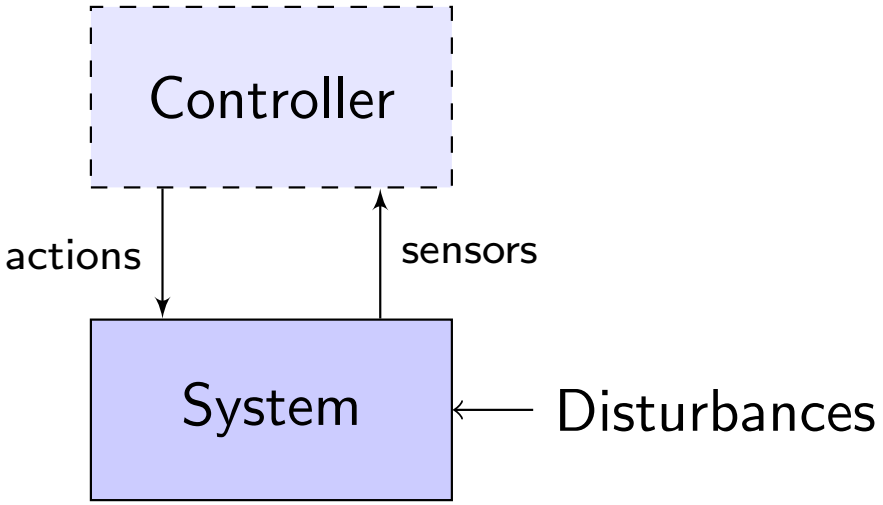
Controller Synthesis for Timed Systems

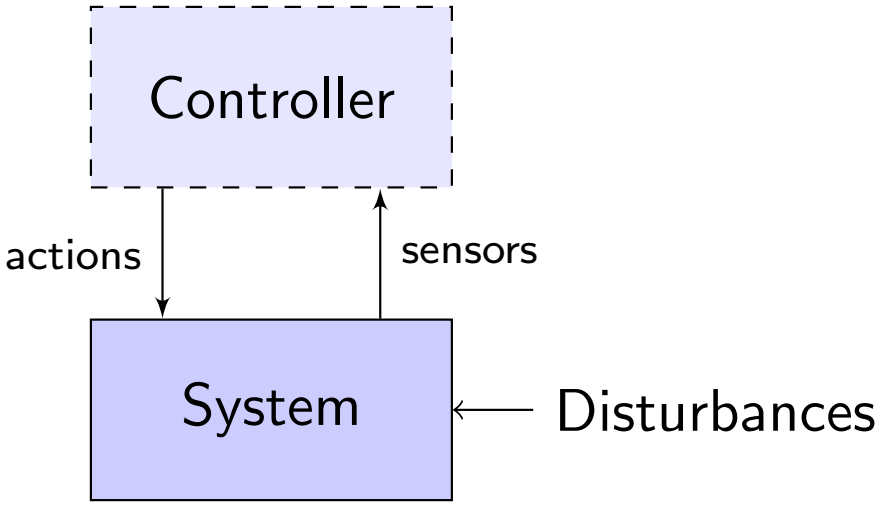
Ocan Sankur

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:

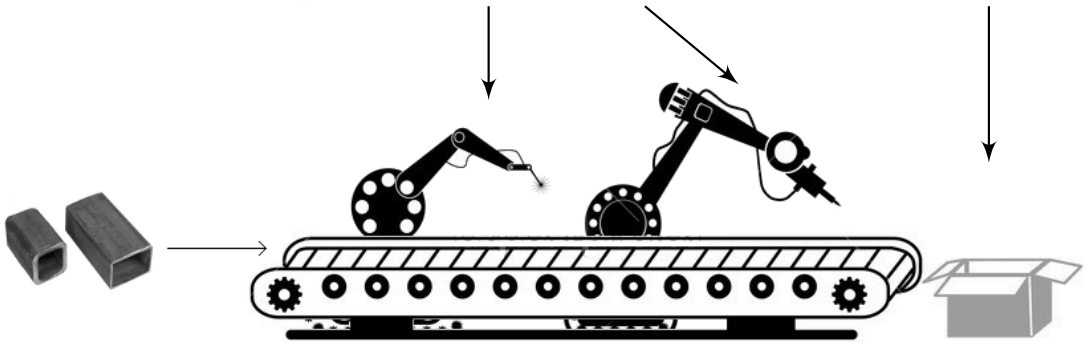
1. The system evolves with *discrete events*
2. We make the passage of time explicit, e.g. we consider execution time, deadlines etc.

Next: example

Production Line Example

Objectives

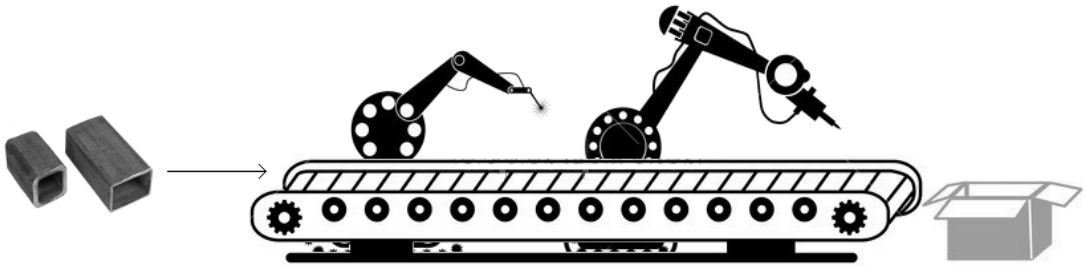
- **Small** or **Large** objects arrive every $\geq$ **10** seconds.
- Small objects must be **painted**, **drilled once**, and **finalized**.
- Large objects must be **painted** **drilled twice** **finalized**.



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.

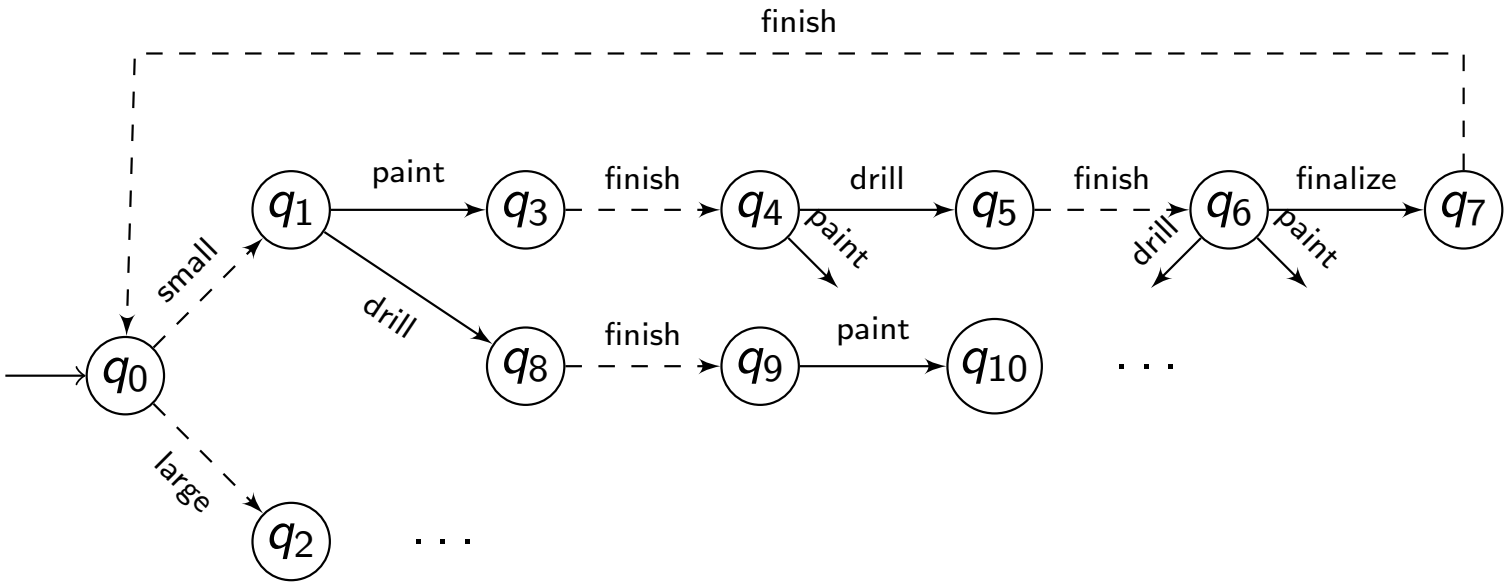
Outline

Outline

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)

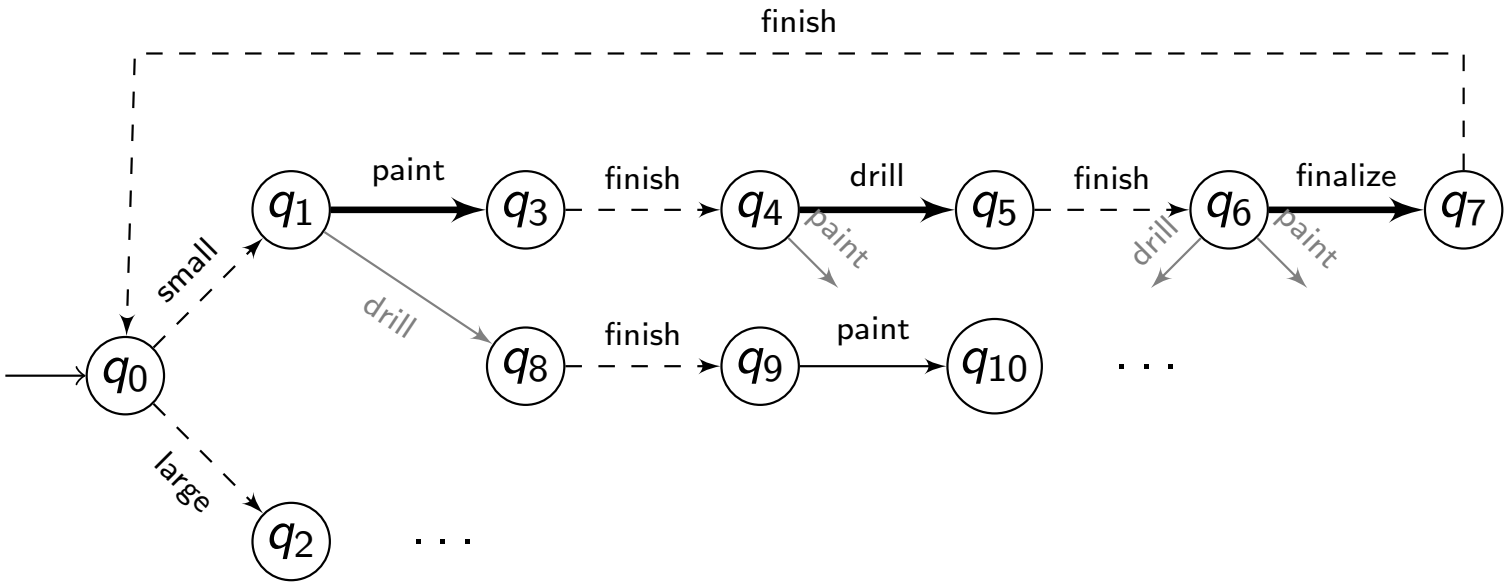


Control Strategy: A mapping from states to control actions
Control Objective: Each arriving object is processed and finalized.
Winning Strategy: All executions under the strategy satisfy the objective.

Finite Games

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

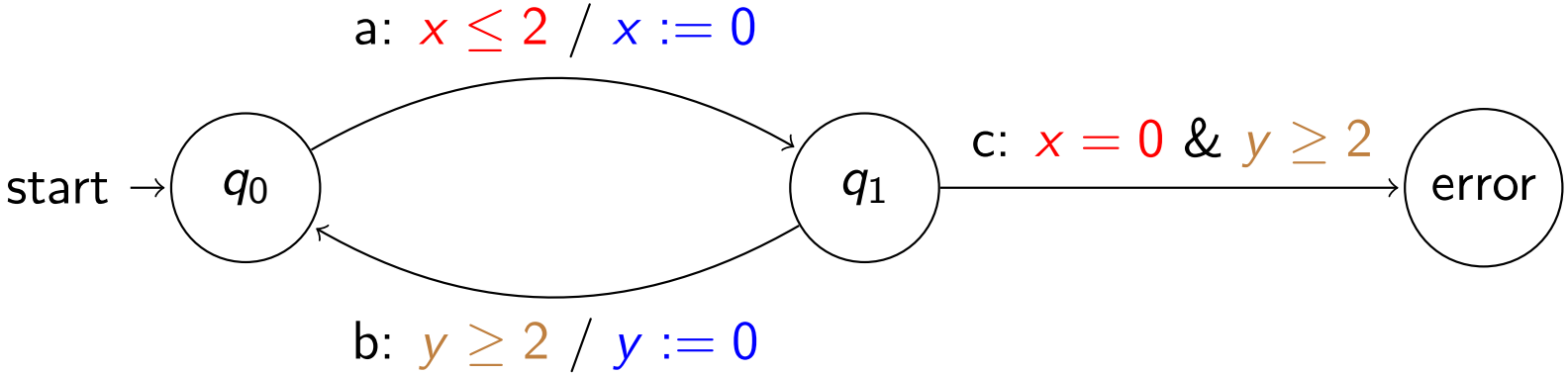
- Nodes are the configurations of the system
- Plain edges: control actions
- Dashed edges: uncontrollable events (coming from the environment)



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.

Timed Automata

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



Run of a TA

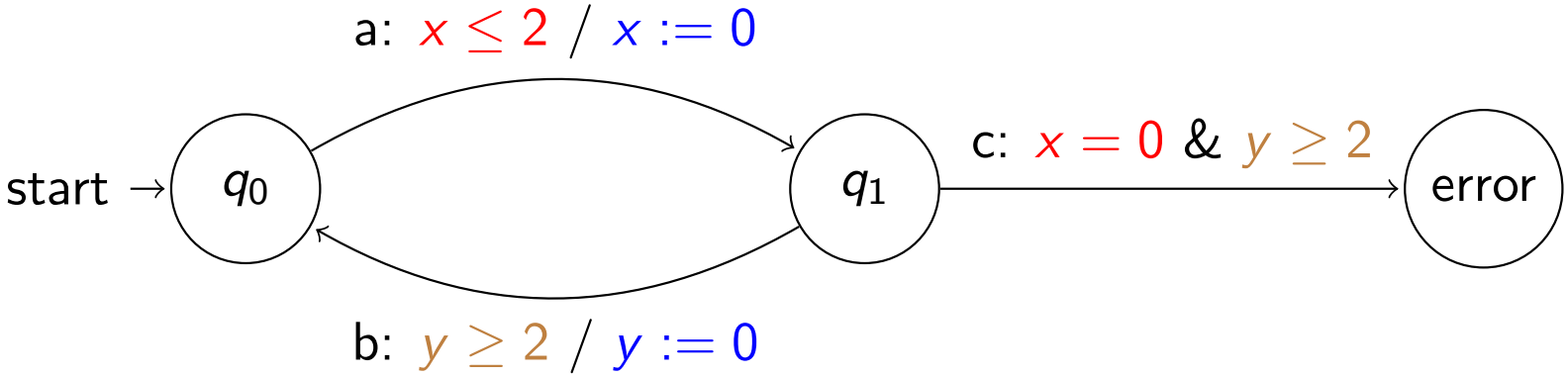
$$(q_0, (x = 0, y = 0)) \xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7))$$
$$\xrightarrow{0.5} (q_1, (x = 0.5, y = 2.2)) \xrightarrow{b} (q_0, (x = 0.5, y = 0)) \ldots$$

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]



Run of a TA

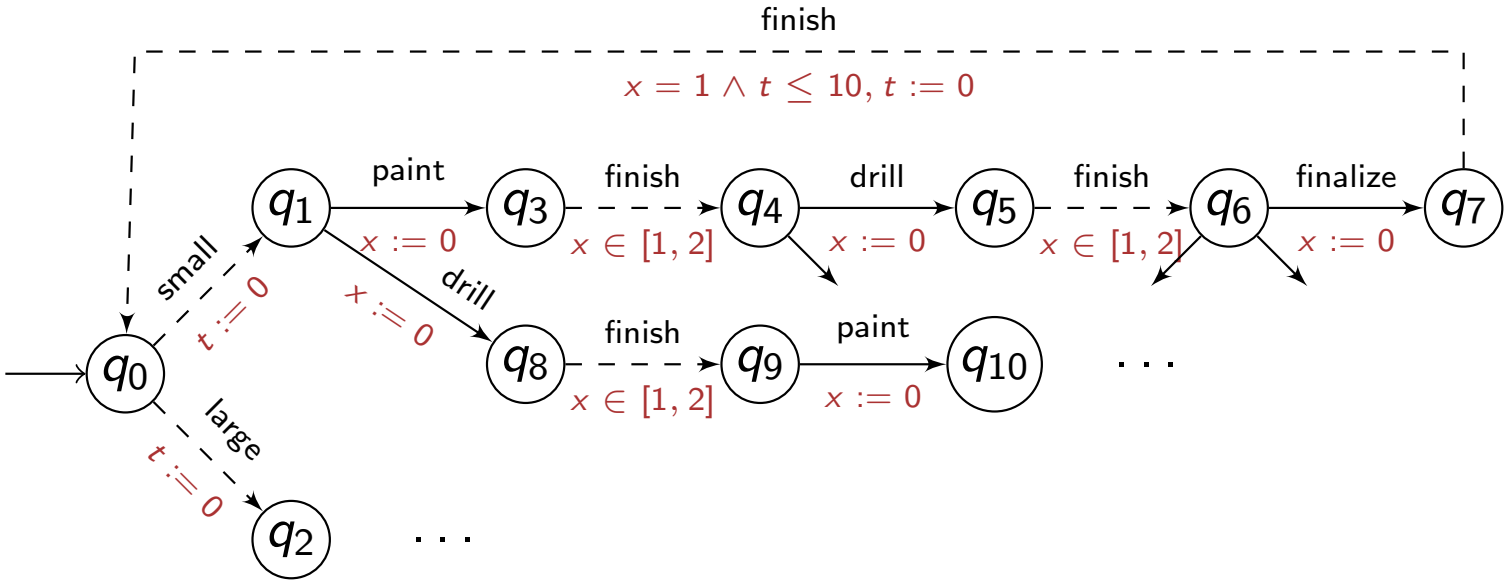
$$(q_0, (x = 0, y = 0)) \xrightarrow{1.7} (q_0, (x = 1.7, y = 1.7)) \xrightarrow{a} (q_1, (x = 0, y = 1.7))$$
$$\xrightarrow{0.5} (q_1, (x = 0.5, y = 2.2)) \xrightarrow{b} (q_0, (x = 0.5, y = 0)) \ldots$$

Timed Systems can be modeled by timed automata:

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

$\rightarrow$ Let us define **games** played on timed automata

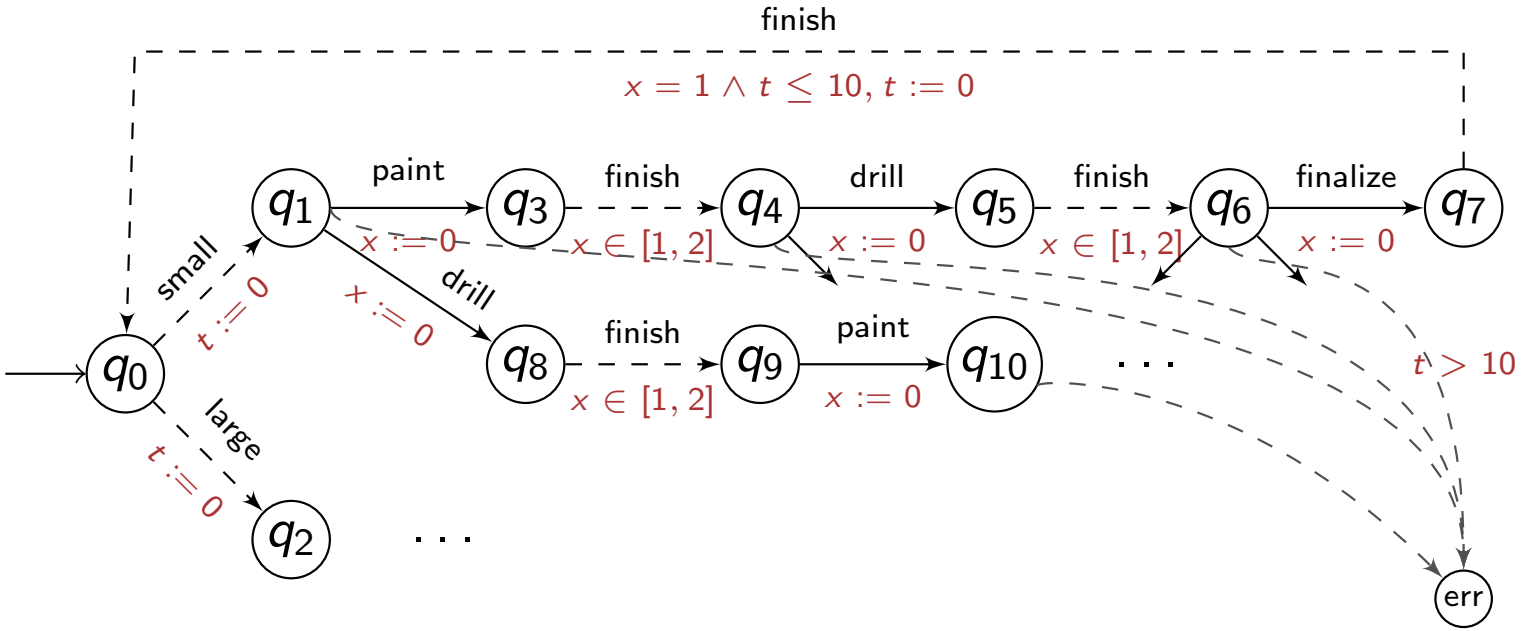
Timed Games



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 $(\mathcal{L} \times \mathbb{R}^C \to \mathbb{R} \times A)$

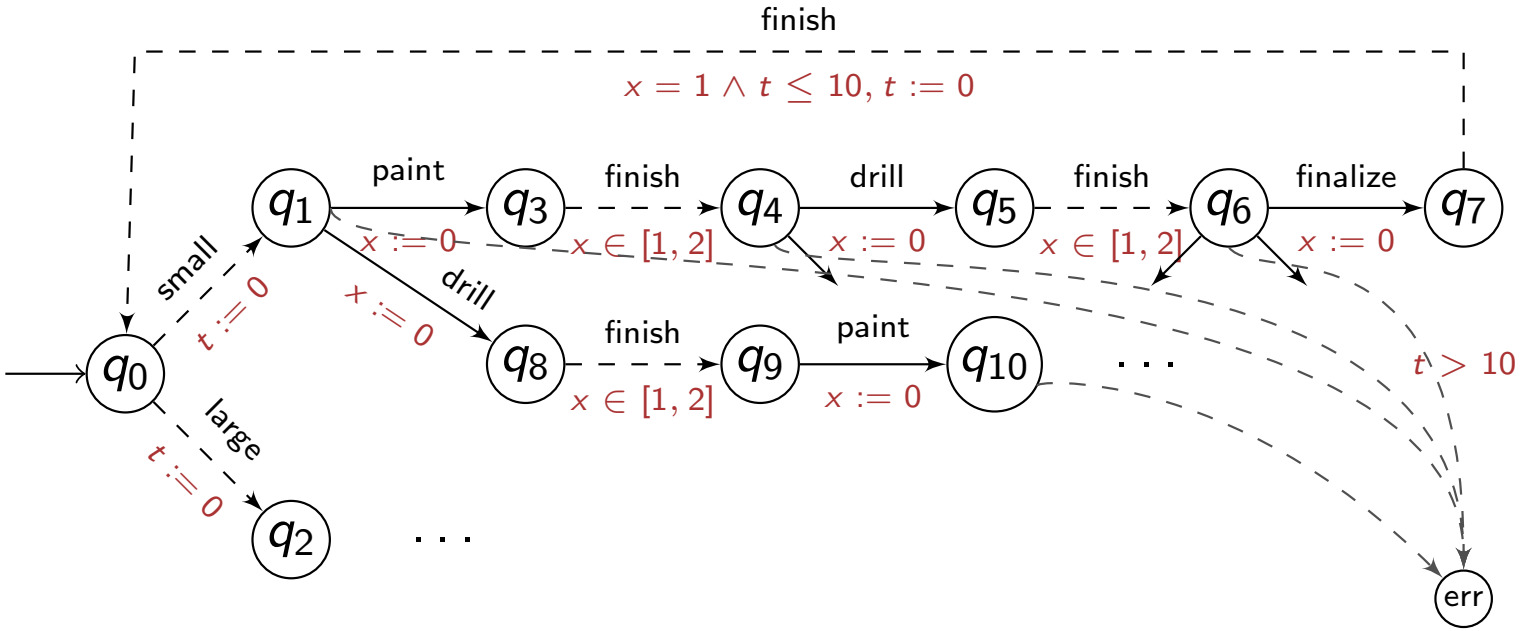
Timed Games



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 $(\mathcal{L} \times \mathbb{R}^C \to \mathbb{R} \times A)$

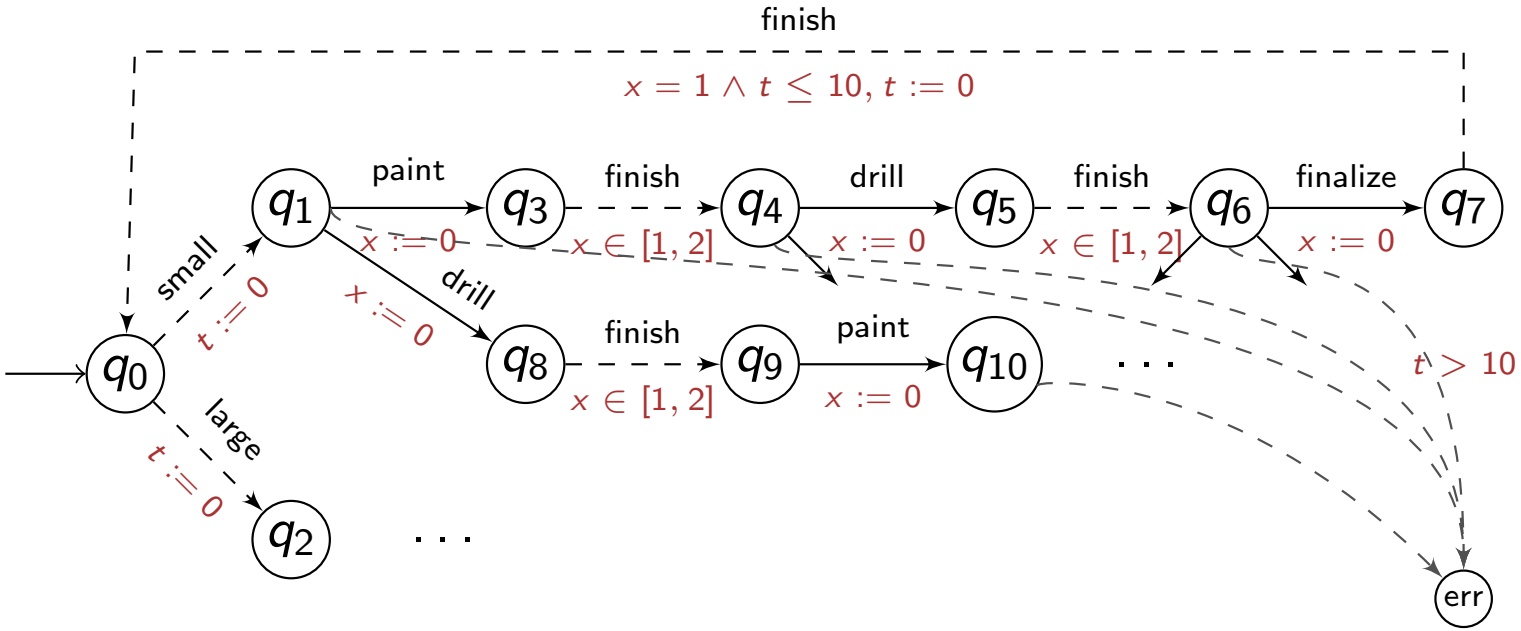
Timed Games



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 $(\mathcal{L} \times \mathbb{R}^C \to \mathbb{R} \times A)$
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 $(\mathcal{L} \times \mathbb{R}^C \to \mathbb{R} \times A)$
such that all generated runs satisfy the objective

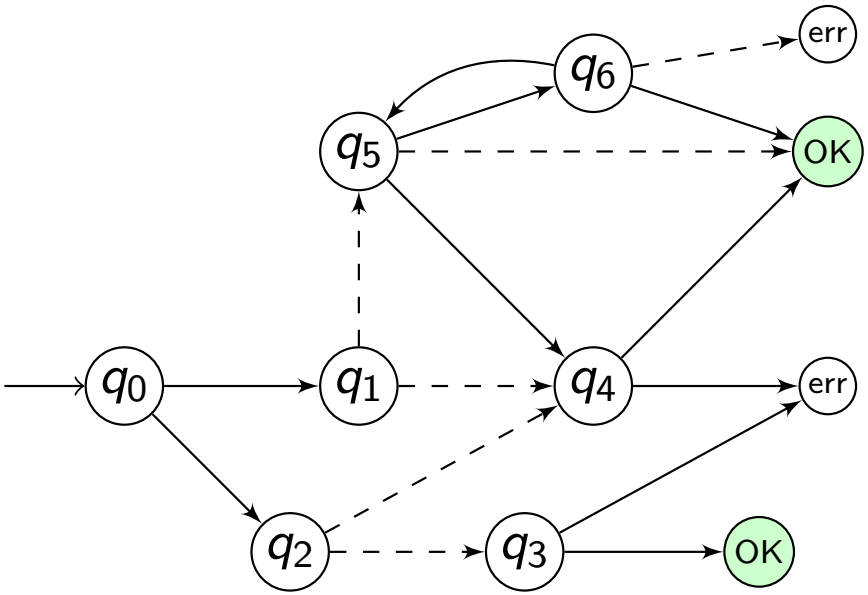
Goal of this tutorial: Describe algorithms to solve controller synthesis using timed games

Outline

Outline

Algorithm for Finite Games

Goal: Reach the `OK` state, starting from q_0

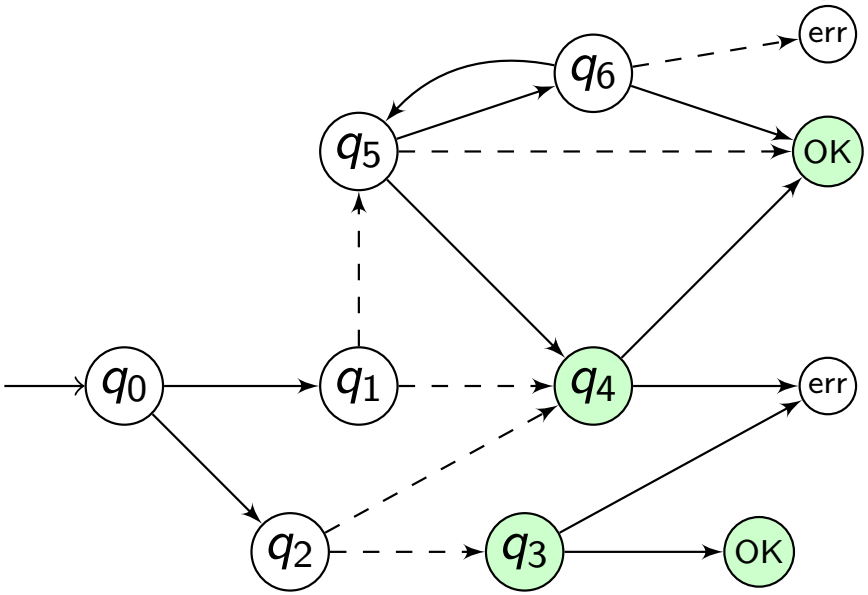


Algorithm

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

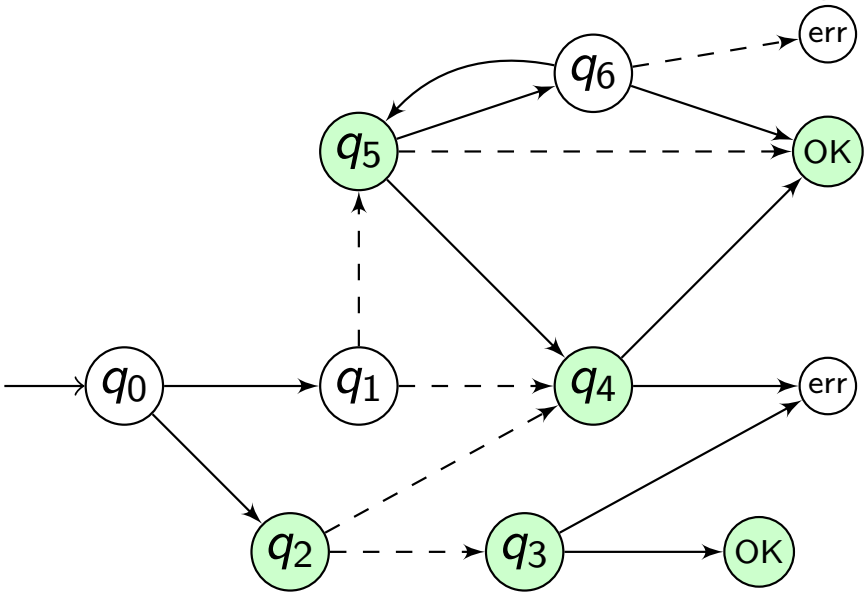


Algorithm

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

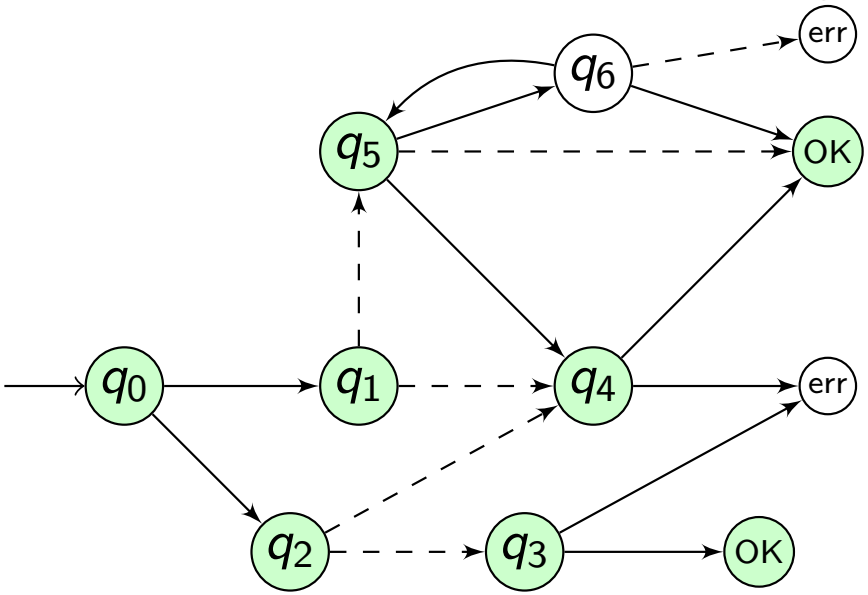


Algorithm

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



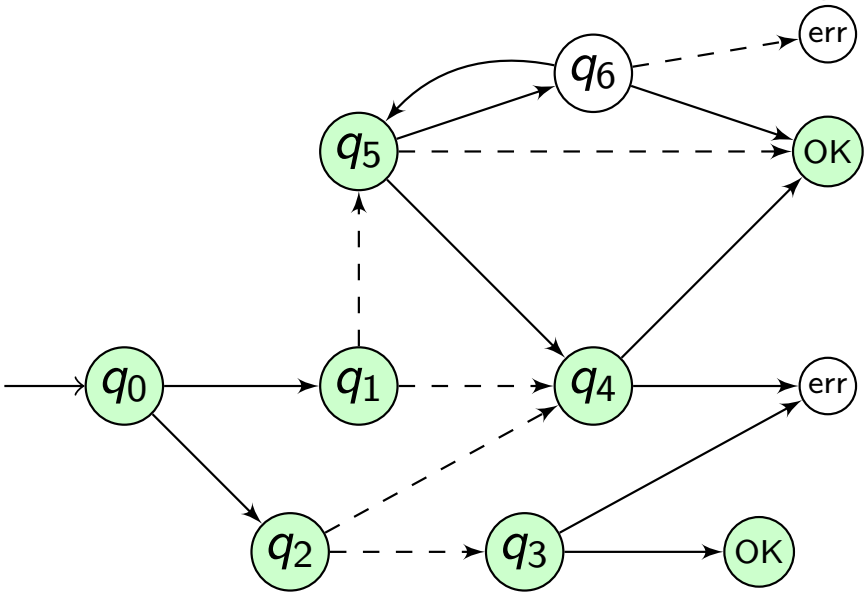
Algorithm

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:

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

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

$X := \{\mathrm{OK}\};$
while $\mathrm{CPRE}(X) \not\subseteq X$ do
    $X := X \cup \mathrm{CPRE}(X);$

Finite Games

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

Algorithm

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

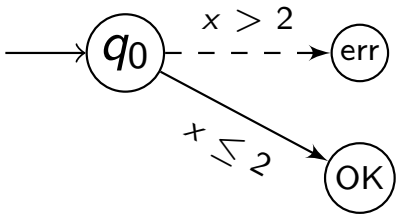
Similar results exist of ω-regular winning conditions, and temporal logics.

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

Outline

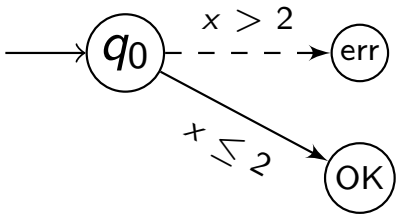
How About Timed Games?

We want to do the same for timed games.



How About Timed Games?

We want to do the same for timed games.



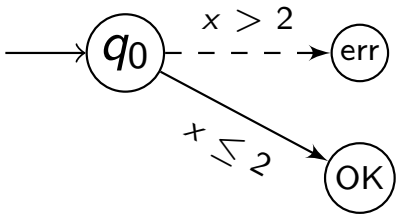
We have $\mathrm{CPRE}(\{\mathrm{OK}\}) = \{(q_0, x) \mid x \in \mathbb{R}, 0 \leq x \leq 2\}$

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 $\mathrm{CPRE}(\{\mathrm{OK}\}) = \{(q_0, x) \mid x \in \mathbb{R}, 0 \leq x \leq 2\}$

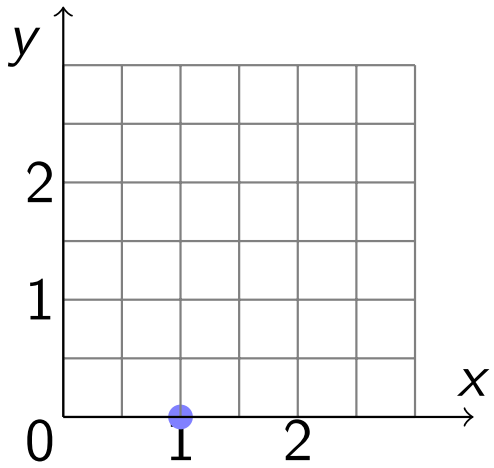
which is **infinite** and **uncountable**!

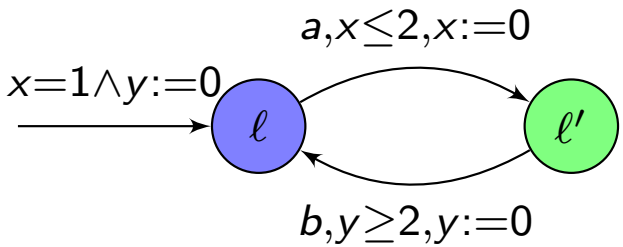
Enumerating successors or predecessors do not make any sense...

$\rightarrow$ Data structure to represent the state space of timed games

Zones - Intuition

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

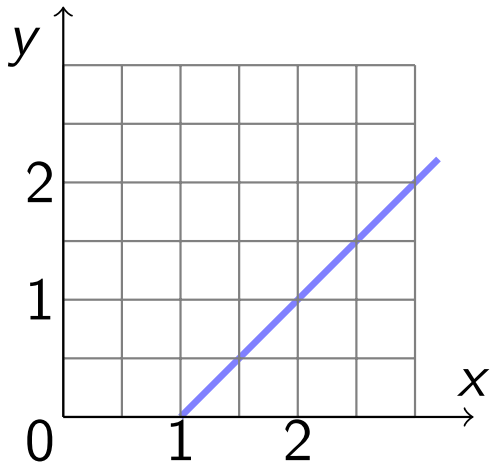


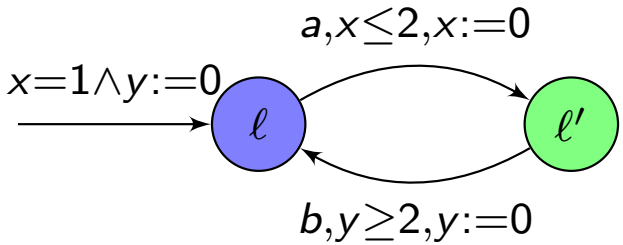


State space exploration:
Initially, at ℓ: $x = 1 \wedge y = 0$

Zones - Intuition

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

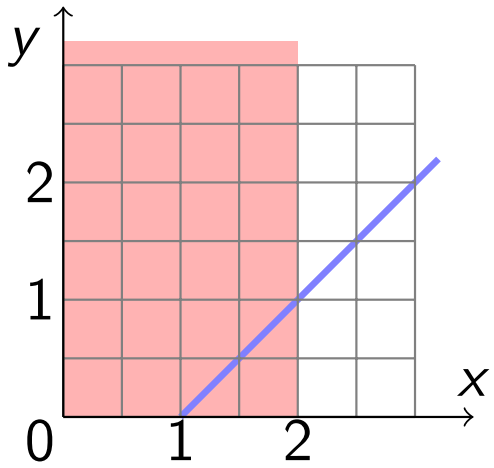


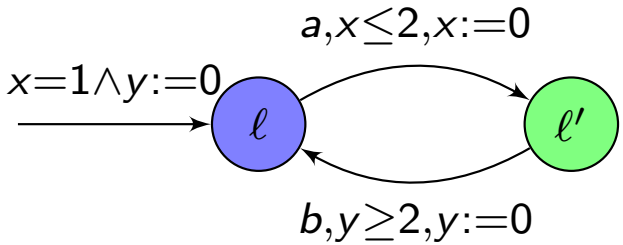


State space exploration:
Initially, at ℓ: $x = 1 \wedge y = 0$
After a time delay, at ℓ: $\text{Post}_{\geq 0}(\ell, x = 1 \wedge y = 0) = (\ell, x - y = 1)$

Zones - Intuition

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

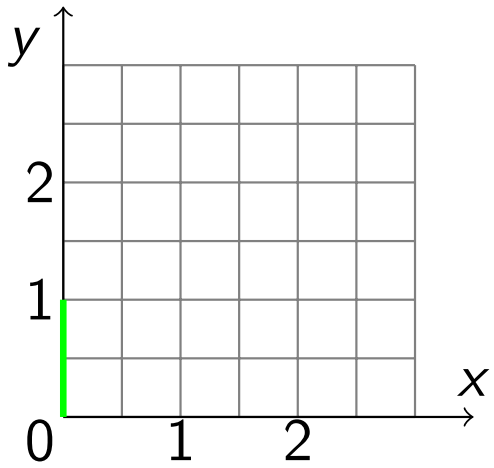


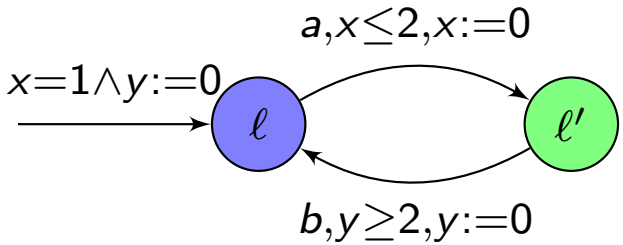


State space exploration:
Initially, at ℓ: $x = 1 \wedge y = 0$
After a time delay, at ℓ: $\text{Post}_{\geq 0}(\ell, x = 1 \wedge y = 0) = (\ell, x - y = 1)$
If we move to ℓ':

Zones - Intuition

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

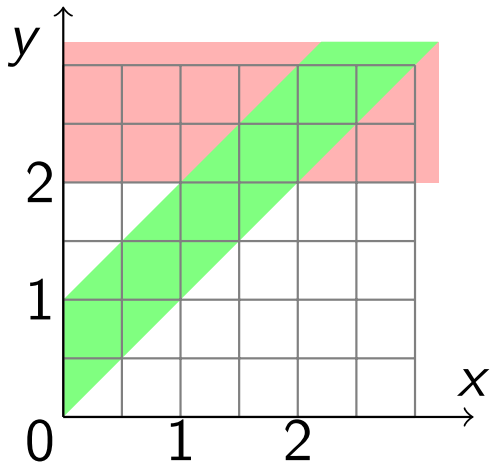


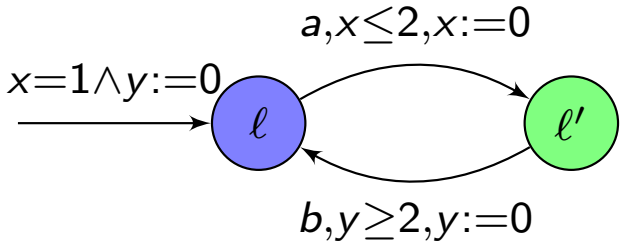


State space exploration:
Initially, at ℓ: $x = 1 \wedge y = 0$
After a time delay, at ℓ: $\mathsf{Post}_{\geq 0}(\ell, x = 1 \wedge y = 0) = (\ell, x - y = 1)$
If we move to ℓ': $\mathsf{Post}_a(\ell, x - y = 1) = (\ell', 0 \leq y \leq 1 \wedge x = 0)$,

Zones - Intuition

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

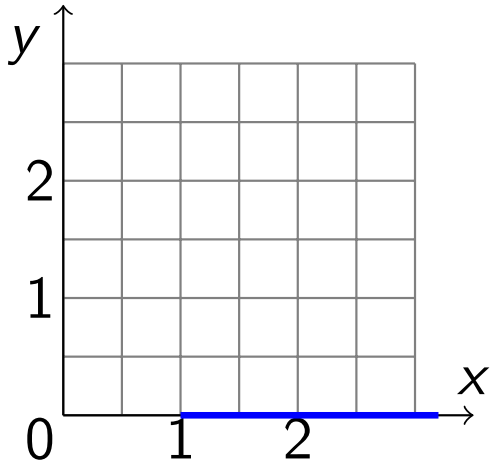


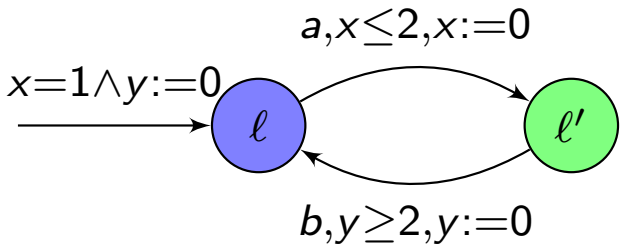


State space exploration:
Initially, at ℓ: $x = 1 \wedge y = 0$
After a time delay, at ℓ: $\mathrm{Post}_{\geq 0}(\ell, x = 1 \wedge y = 0) = (\ell, x - y = 1)$
If we move to ℓ': $\mathrm{Post}_a(\ell, x - y = 1) = (\ell', 0 \leq y \leq 1 \wedge x = 0)$, and delay: $0 \leq y - x \leq 1$.

Zones - Intuition

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





State space exploration:
Initially, at ℓ: $x = 1 \wedge y = 0$
After a time delay, at ℓ: $\mathrm{Post}_{\geq 0}(\ell, x = 1 \wedge y = 0) = (\ell, x - y = 1)$
If we move to ℓ': $\mathrm{Post}_a(\ell, x - y = 1) = (\ell', 0 \leq y \leq 1 \wedge x = 0)$, and delay: $0 \leq y - x \leq 1$.
Move back to ℓ: $\mathrm{Post}_b(\ell', 0 \leq y - x \leq 1) = (\ell, y = 0 \wedge x \geq 1)$, 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 ω-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 ω-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?

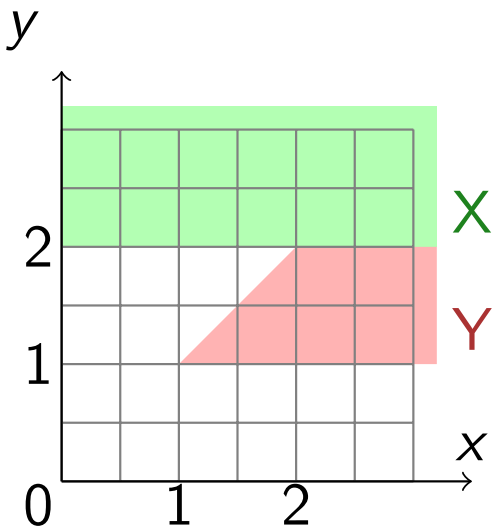
CPRE for Timed Games

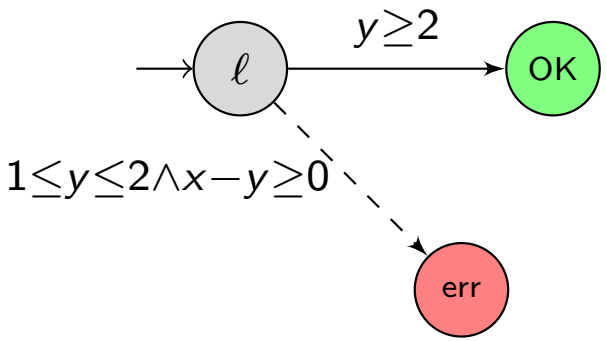
Denote $\mathcal{Q} = \mathcal{L} \times \mathbb{R}^{C}_{\geq 0}$.

Safe Time-Predecessors

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

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





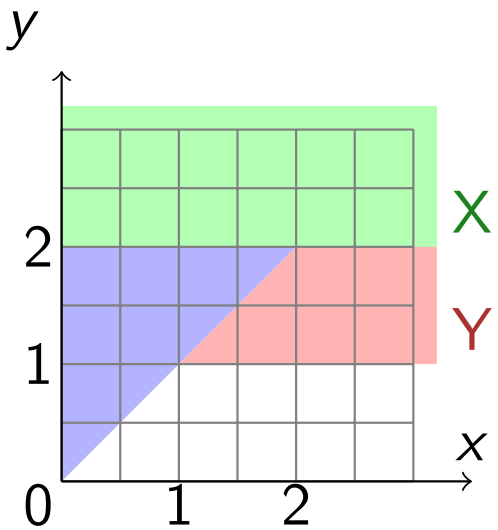
CPRE for Timed Games

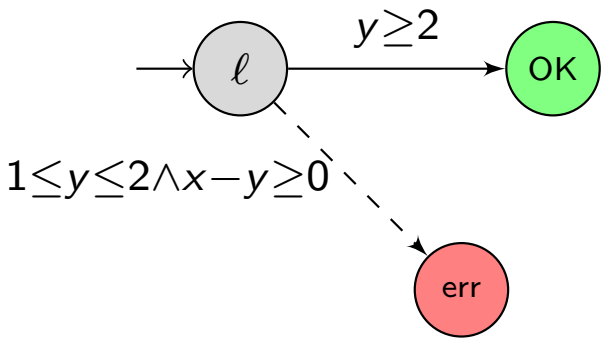
Denote $\mathcal{Q} = \mathcal{L} \times \mathbb{R}^{C}_{\geq 0}$.

Safe Time-Predecessors

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

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





CPRE for Timed Games

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

Safe Time-Predecessors

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

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

Control and Environment Predecessors

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

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

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

CPRE for Timed Games

Safe Time-Predecessors

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

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

Control and Environment Predecessors

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

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

$$\mathrm{Pred}_u(X) = \{q \in \mathcal{Q} \mid \exists q' \in X, \exists u, q \overset{\underline{u}}{\dashrightarrow} q'\}. \quad (2)$$

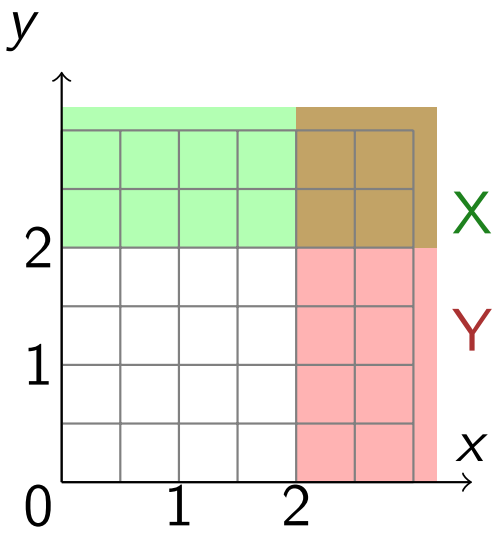
Define

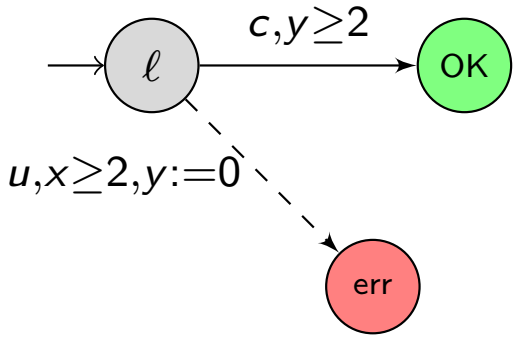
$$\mathrm{CPRE}(X) = \mathrm{Pred}_{\geq 0}(\mathrm{Pred}_c(X), \mathrm{Pred}_u(\mathcal{Q} \setminus X)).$$

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

CPRE for Timed Games

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

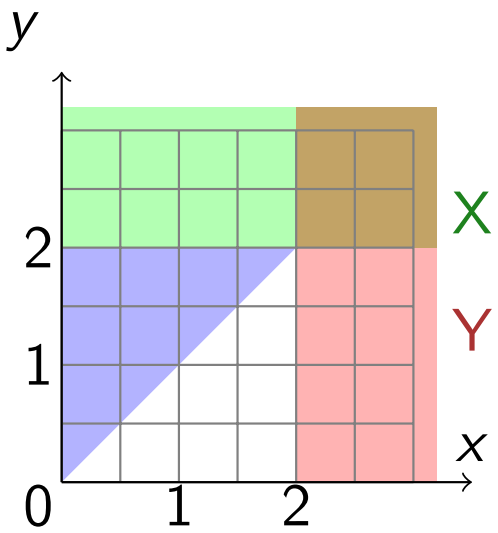


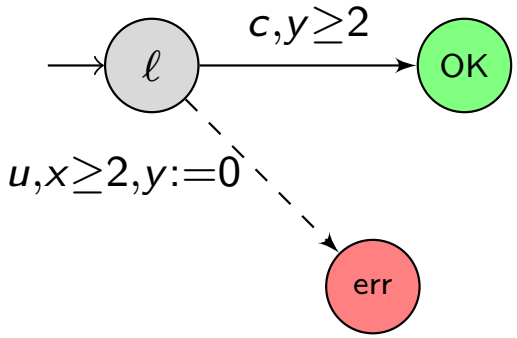


- $X = \mathit{OK} \times \mathbb{R}^C_{\geq 0}$,
- $\mathrm{Pred}_c(\mathrm{OK} \times \mathbb{R}^C_{\geq 0}) = (\ell, y \geq 2)$,
- $\mathrm{Pred}_u(\mathcal{Q} \setminus X) = \mathrm{Pred}_u((\mathrm{err} \cup \{\ell\}) \times \mathbb{R}^C_{\geq 0}) = (\ell, x \geq 2)$.

CPRE for Timed Games

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





- $X = OK \times \mathbb{R}^C_{\geq 0}$,
- $\mathrm{Pred}_c(\mathrm{OK} \times \mathbb{R}^C_{\geq 0}) = (\ell, y \geq 2)$,
- $\mathrm{Pred}_u(\mathcal{Q} \setminus X) = \mathrm{Pred}_u((\mathrm{err} \cup \{\ell\}) \times \mathbb{R}^C_{\geq 0}) = (\ell, x \geq 2)$.

CPRE for Timed Games

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

Solving Timed Games

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

In other terms, the algorithm is the following.

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

Complexity

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

CPRE for Timed Games

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

Solving Timed Games

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

How do we actually compute CPRE(X)?

Which data structure?

Which precise algorithm?

while $X \neq X \cup \mathrm{CPRE}(X)$ do

$X := X \cup \mathrm{CPRE}(X)$

Complexity

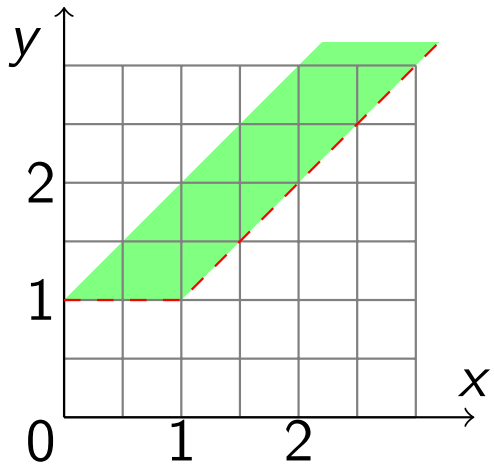
Determining if Controller has a winning strategy from a given state in a timed game is EXPTIME-complete

Outline

Difference-Bound Matrices (DBM)

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

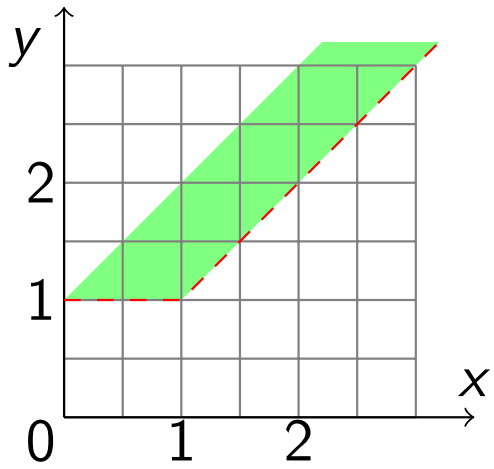
$$x < y \quad \wedge \quad y \leq x + 1 \quad \wedge \quad x \geq 0 \quad \wedge \quad y > 1$$



Difference-Bound Matrices (DBM)

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

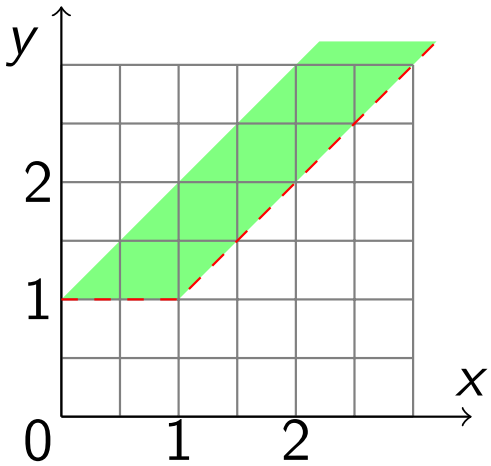
$$x - y < 0 \quad \wedge \quad y - x \leq 1 \quad \wedge \quad -x \leq 0 \quad \wedge \quad -y < -1$$



Difference-Bound Matrices (DBM)

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

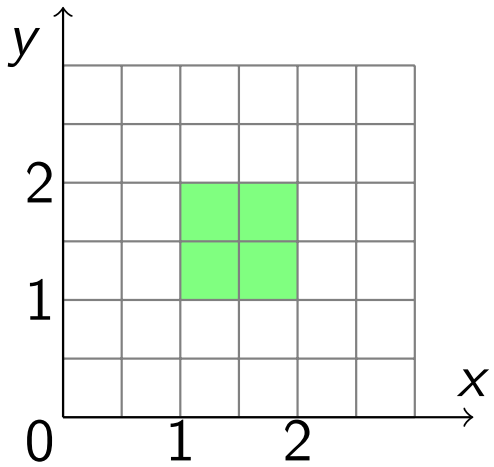
$$x - y < 0 \quad \wedge \quad y - x \leq 1 \quad \wedge \quad -x \leq 0 \quad \wedge \quad -y < -1$$



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

Difference-Bound Matrices (DBM)- Canonical Form

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



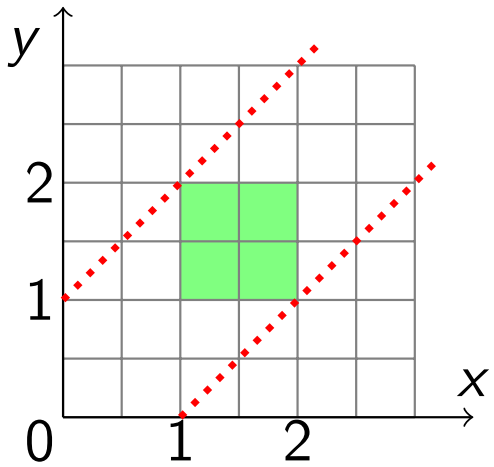
$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 2, \leq & 0, \leq & \infty, < \\ 2, \leq & \infty, < & 0, \leq \end{pmatrix} \end{array}$$

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

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



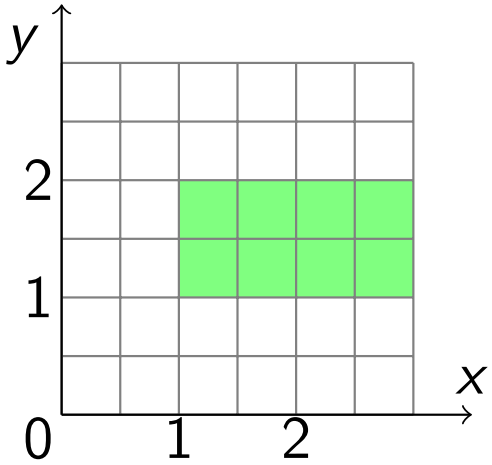
$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 2, \leq & 0, \leq & 1, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix} \end{array}$$

DBM in canonical form: All constraints are tight.

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

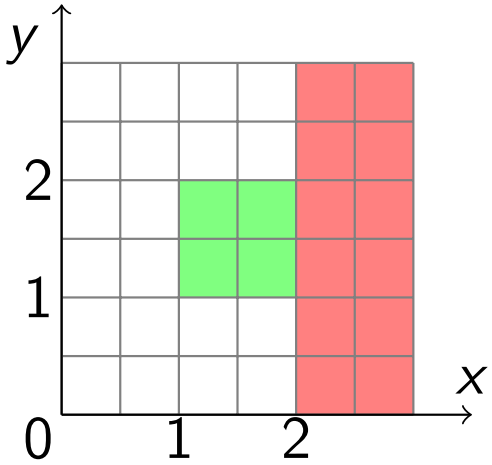
Difference-Bound Matrices (DBM) - Operations

Intersection



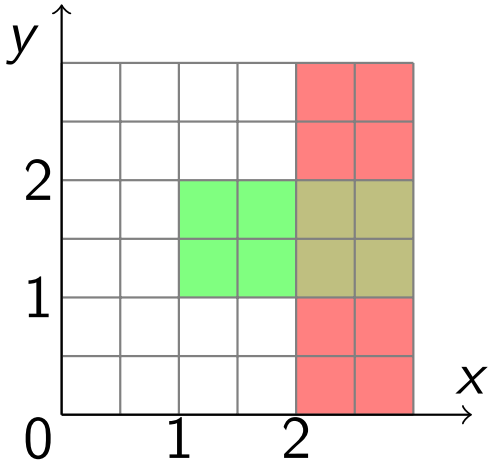
Difference-Bound Matrices (DBM) - Operations

Intersection



Difference-Bound Matrices (DBM) - Operations

Intersection



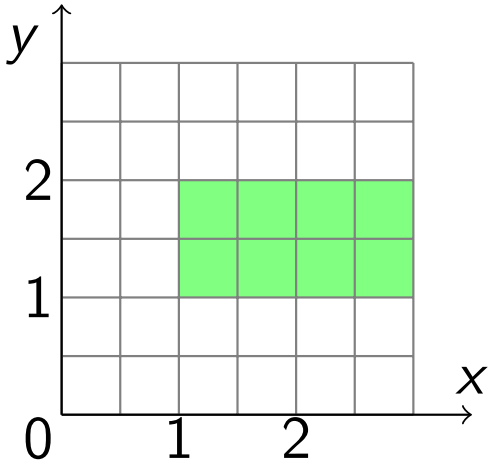
Idea: $y \leq 2$ **and** $y < 3 \;=\; y \leq 2$.

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

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

Difference-Bound Matrices (DBM) - Operations

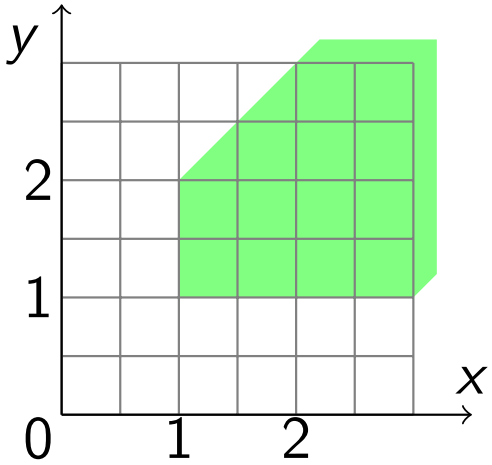
Time Successors



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

Difference-Bound Matrices (DBM) - Operations

Time Successors

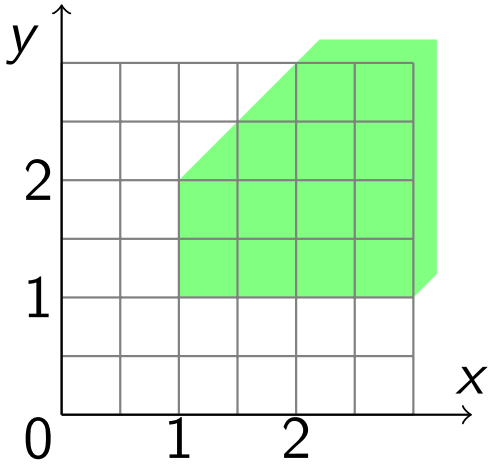


$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ 3, \leq & 0, \leq & 2, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix} \end{array}$$

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

Difference-Bound Matrices (DBM) - Operations

Time Successors

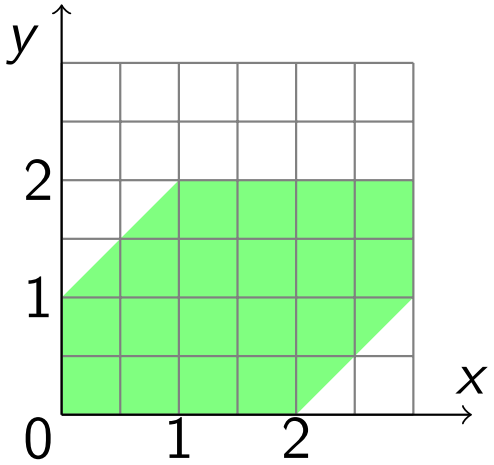


$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} 0, \leq & -1, \leq & -1, \leq \\ \infty, < & 0, \leq & 2, \leq \\ \infty, < & 1, \leq & 0, \leq \end{pmatrix} \end{array}$$

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

Difference-Bound Matrices (DBM) - Operations

Time Predecessors

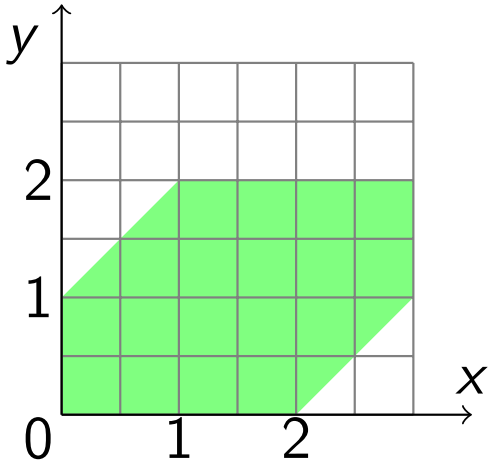


$$\begin{array}{c} \\ 0 \\ x \\ y \end{array} \begin{array}{c} \begin{array}{ccc} 0 & x & y \end{array} \\ \begin{pmatrix} 0, \leq & 0, \leq & 0, \leq \\ 3, \leq & 0, \leq & 2, \leq \\ 2, \leq & 1, \leq & 0, \leq \end{pmatrix} \end{array}$$

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

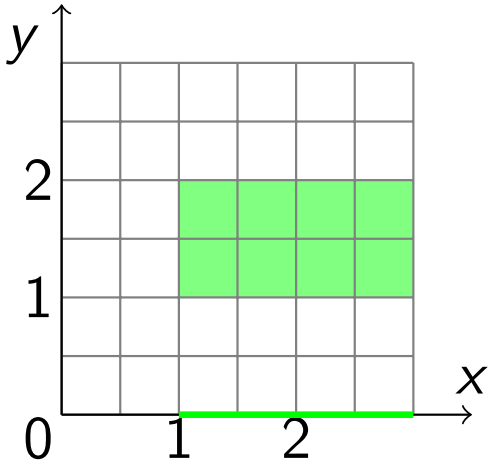
Difference-Bound Matrices (DBM) - Operations

Time Predecessors



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

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 $O(C^3)$

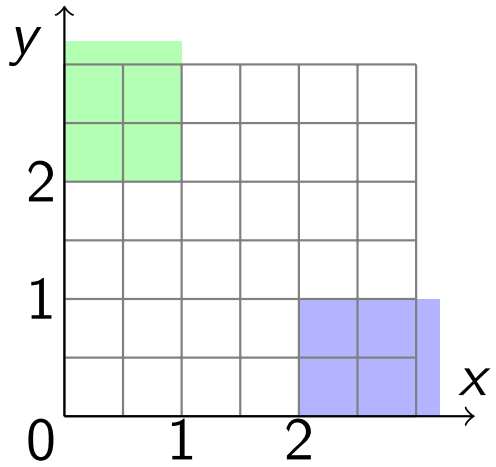
  e.g. for successor of set M through an edge with guard G, resetting x:

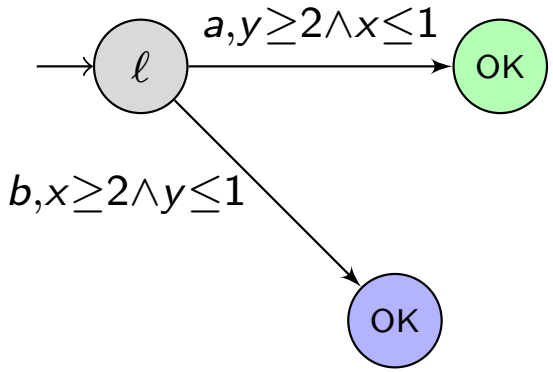
$$\mathrm{Post}_e(M) = \mathrm{Reset}_x\big(\mathrm{Post}_{\geq 0}(M) \cap G\big)$$

Can we compute CPRE with DBMs?

CPRE is not convex

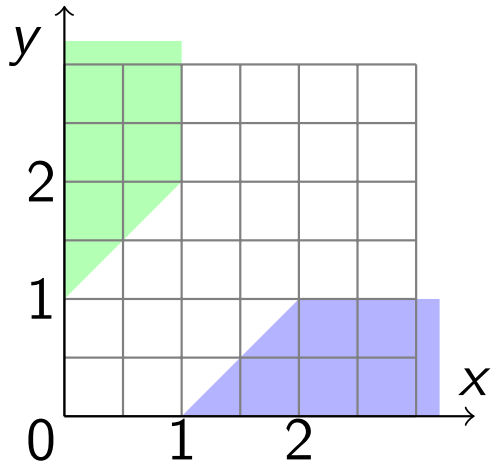
Computing $\mathsf{CPRE}(\{\mathsf{OK}\}) = \mathsf{Pred}_{\geq 0}\big(\mathsf{Pred}_c(\{\mathsf{OK}\}), \emptyset\big)$.

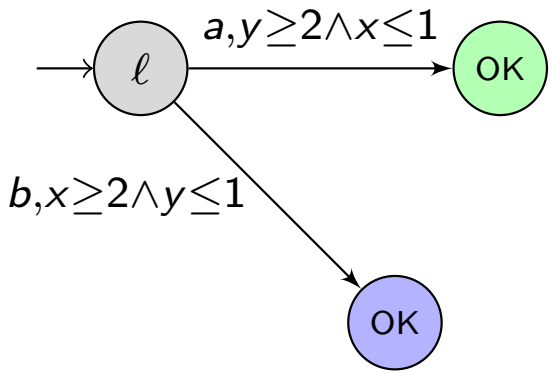




CPRE is not convex

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





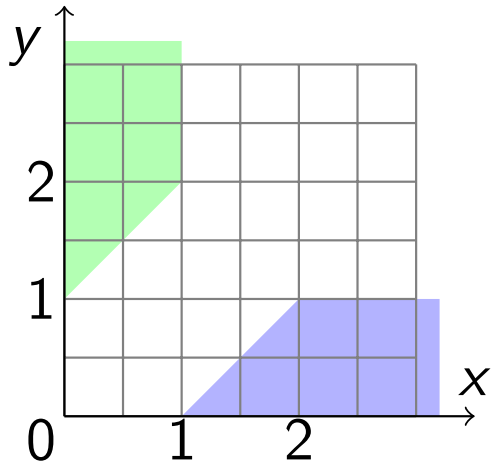
Non-convex...

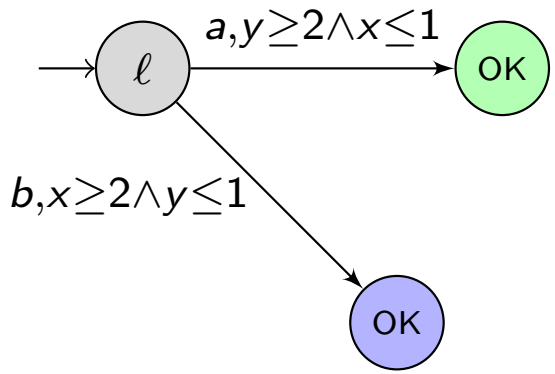
Moreover we also need **complementation** which is also a non-convex operation

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

CPRE is not convex

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





Non-convex...

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

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

DBM Federation

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

DBM Federations – Operations

Some easy operations:

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

DBM Federations – Operations

Some easy operations:

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

DBM Federations – Operations

Some easy operations:

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

DBM Federations – Operations

Some easy operations:

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

New: **Complementation**

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

DBM Federations – Operations

Some easy operations:

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

New: **Complementation**

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

To compute N^c,

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

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

DBM Federations – Operations

Some easy operations:

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

New: **Complementation**

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

To compute N^c,

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

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

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:

$$
\begin{array}{l}
X := T \\
\text{while } X \neq X \cup \mathrm{CPRE}(X) \text{ do} \\
\qquad X := X \cup \mathrm{CPRE}(X)
\end{array}
$$

Algorithm

Computation of the winning states using DBM federations:

$$
\begin{array}{l}
X := T \\
\text{while } X \neq X \cup \mathrm{CPRE}(X) \text{ do} \\
\qquad X := X \cup \mathrm{CPRE}(X)
\end{array}
$$

- 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:

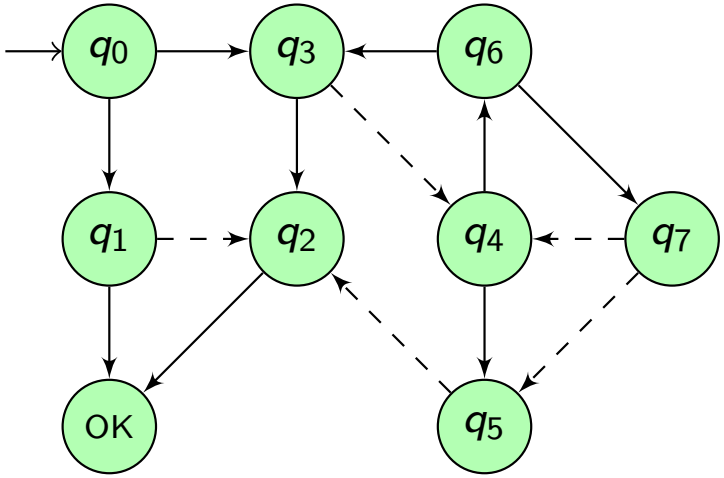
$$
\begin{array}{l}
X := T \\
\text{while } X \neq X \cup \mathrm{CPRE}(X) \text{ do} \\
\qquad X := X \cup \mathrm{CPRE}(X)
\end{array}
$$

- 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

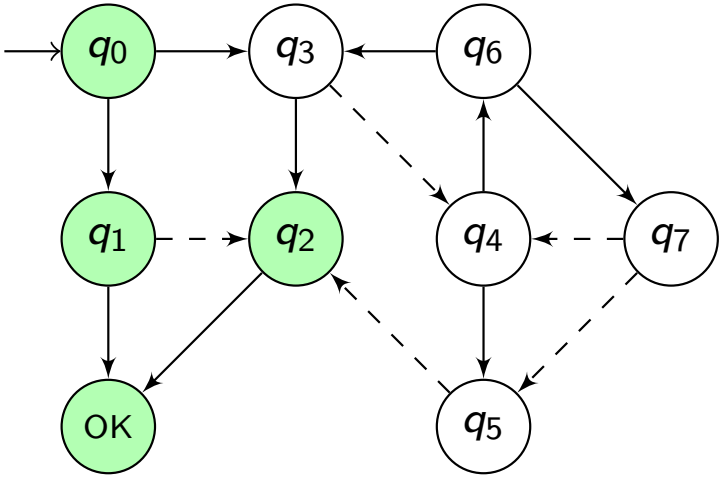
Smaller Solutions



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

$$X_1 = \{\mathsf{OK}\}, X_2 = \mathsf{CPRE}(X_1), X_3 = \mathsf{CPRE}(X_1 \cup X_2), \ldots$$

Smaller Solutions



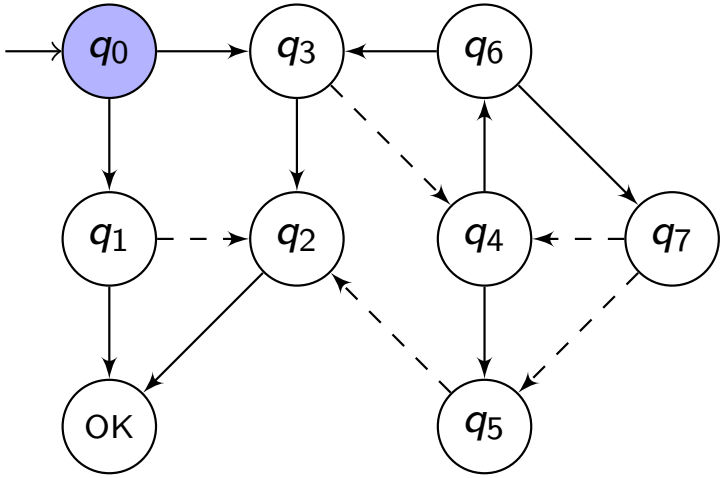
- Backwards algorithm computes $X_1, \ldots, X_n$ such that

$$X_1 = \{\mathsf{OK}\}, X_2 = \mathsf{CPRE}(X_1), X_3 = \mathsf{CPRE}(X_1 \cup X_2), \ldots$$

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

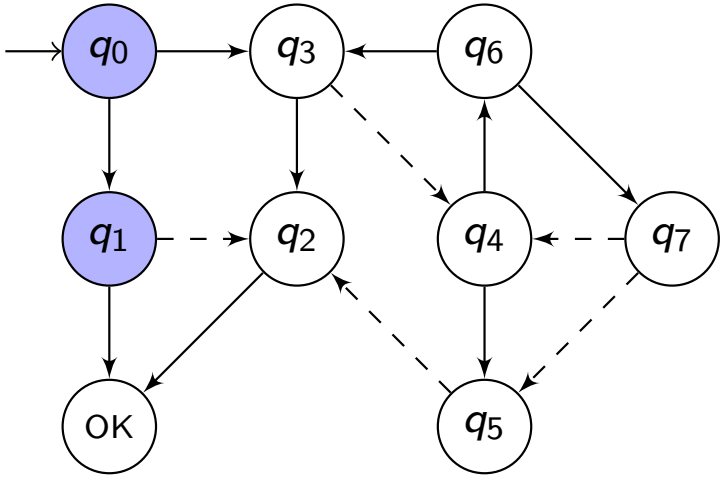
$$\forall i \geq 2, Y_i \subseteq \mathsf{CPRE}(Y_1 \cup \ldots \cup Y_{i-1}).$$

Forward Algorithm for Finite Games



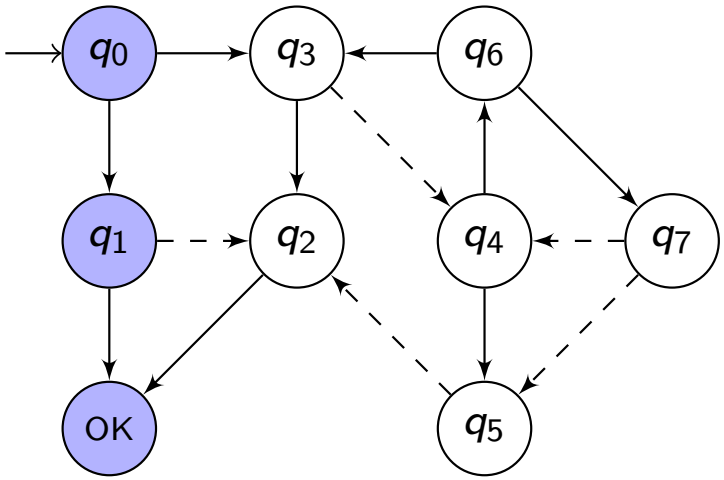
- Explore forward (e.g. depth-first search) until you hit a target state

Forward Algorithm for Finite Games



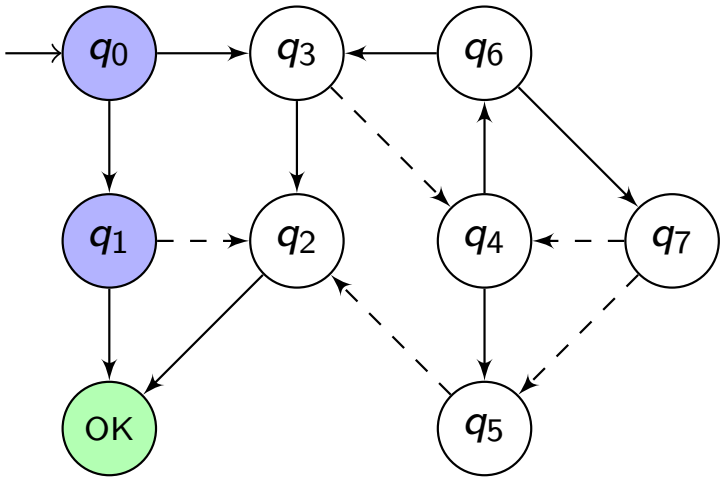
- Explore forward (e.g. depth-first search) until you hit a target state

Forward Algorithm for Finite Games



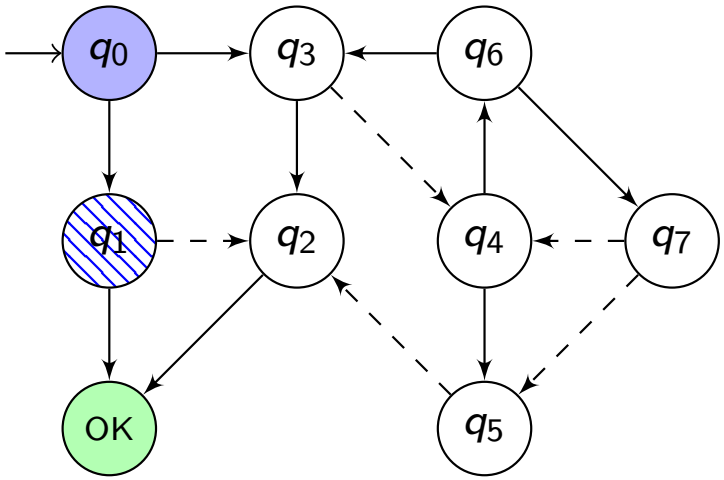
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



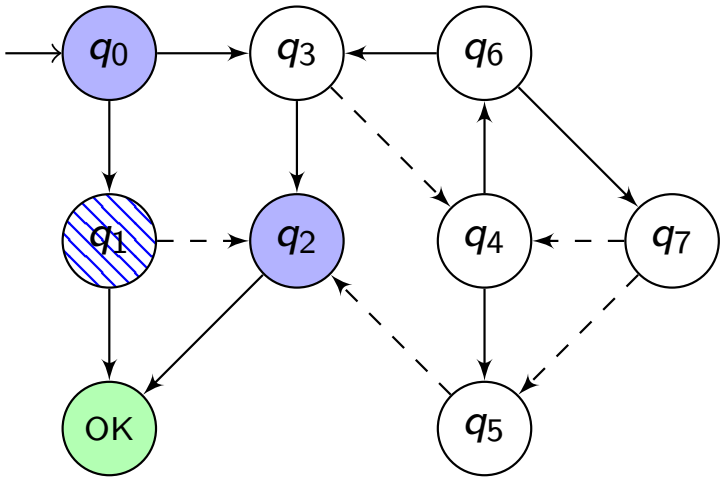
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



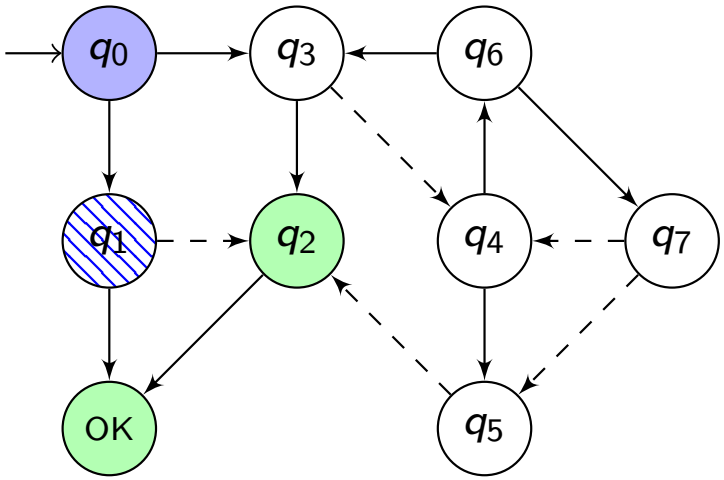
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



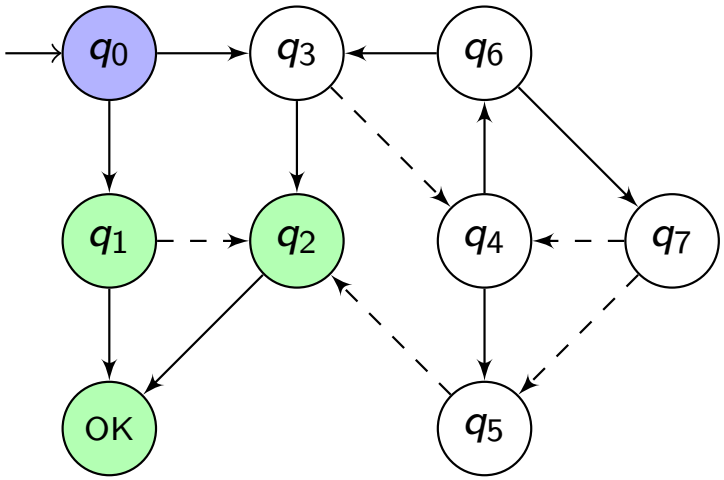
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



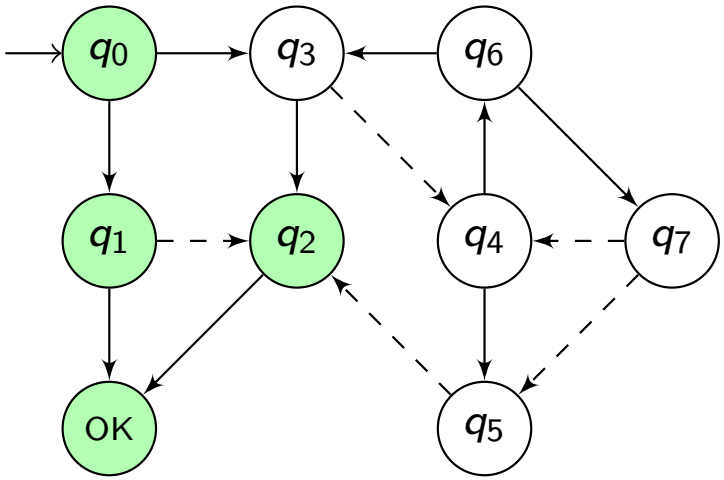
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



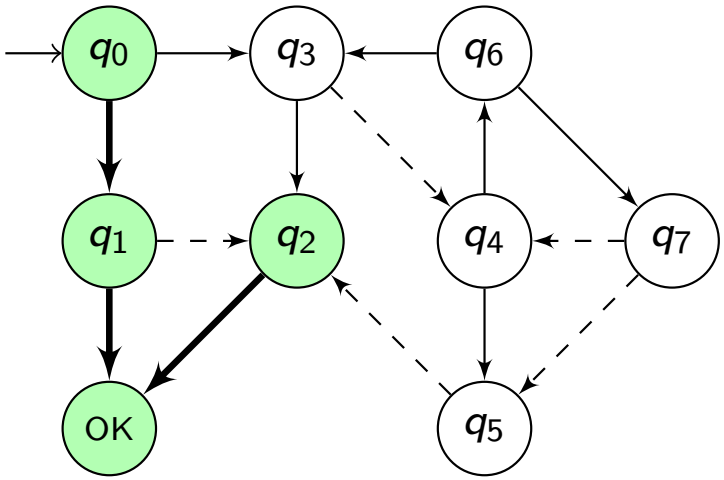
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



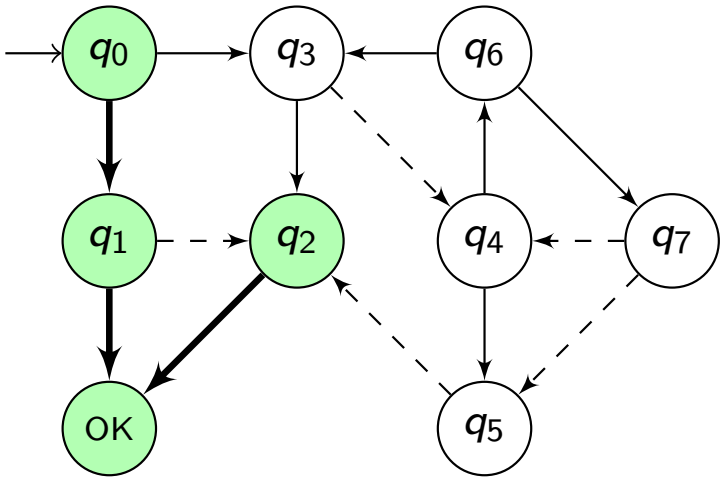
- Explore forward (e.g. depth-first search) until you hit a target state
- **Backtrack** (backwards): for each state q, check if we can conclude about winning
  - If winning, then mark as winning
  - If we are not sure, then explore its successors

Forward Algorithm for Finite Games



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

Forward Algorithm for Finite Games



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

Timed games: Apply same idea:

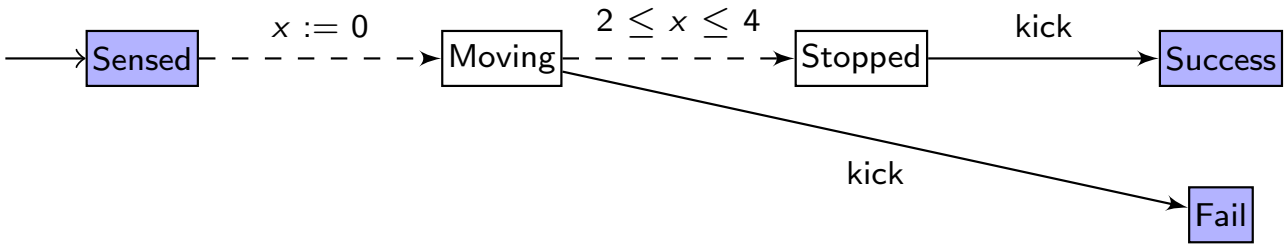
- Forward exploration using zones
- Whenever a target state is found, **backtrack:**
  - Explore all uncontrollable edges that haven't been explored
  - Use the CPRE operator to backtrack

Outline

Partial Observation

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

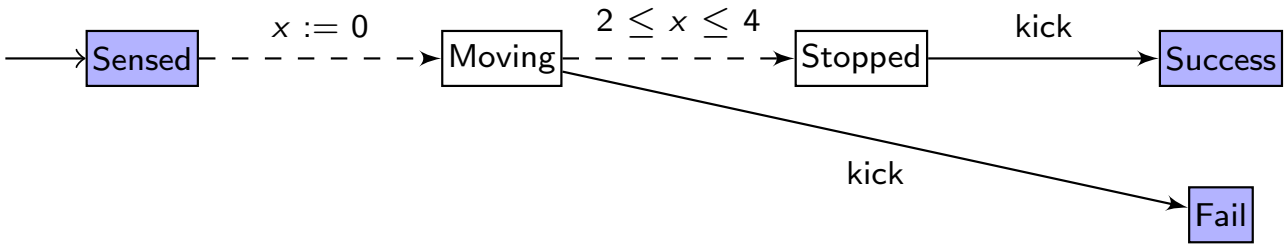
Consider the following system in which the controller can only **sense** when the system enters a blue state:



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** when the system enters a blue state:



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

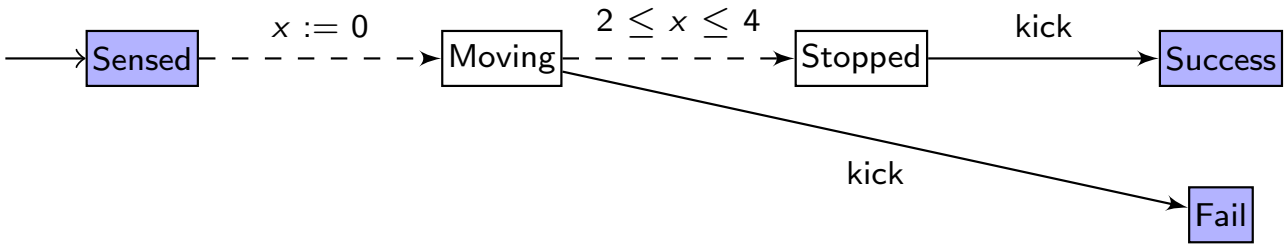
Control under Partial Information

Is there a controller whose decisions only depend on the obtained observations (but not on the internal states and clocks of the system)?

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** when the system enters a blue state:



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

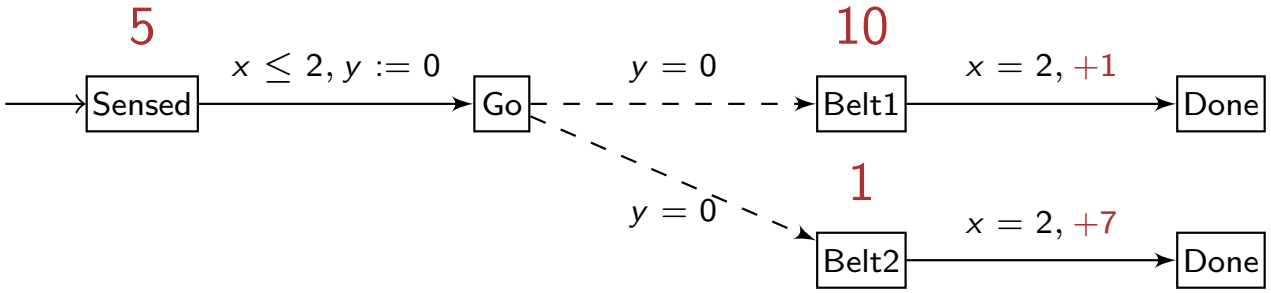
Control under Partial Information

Is there a controller whose decisions only depend on the obtained observations (but not on the internal states and clocks of the system)?

▶ Under some assumptions on the controller type, the problem is decidable, and the previous forward algorithm can be generalized.

Optimal Controllers

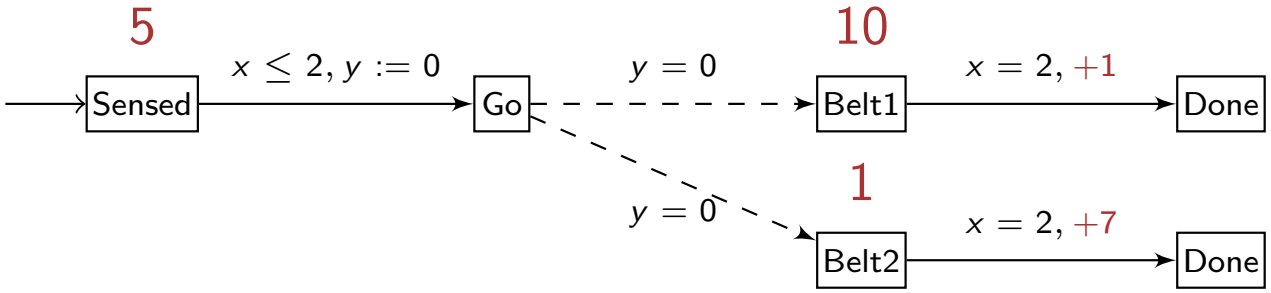
The previously presented algorithms compute an **arbitrary** controller.
Assume we want to compute the cheapest, or the fastest controller.



Weighted timed games: the cost of spending t time units at location ℓ 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.

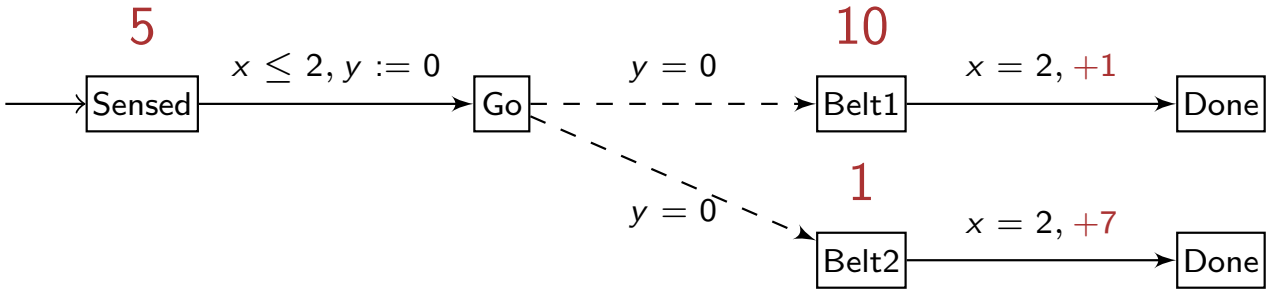


Weighted timed games: the cost of spending t time units at location ℓ is $C(\ell) \cdot t$, and each transition has a cost.

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 ℓ 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:

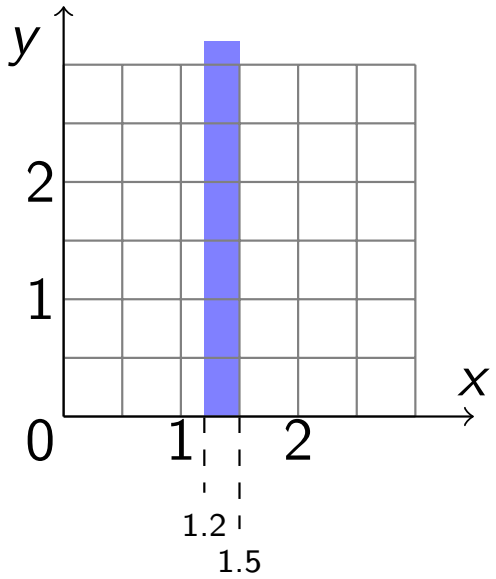
$$14 + \frac{1}{3} = \min_{t \in [0,2]} (\max(5t + 10(2 - t) + 1, 5t + (2 - t) + 7)).$$

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 ≤ 15.

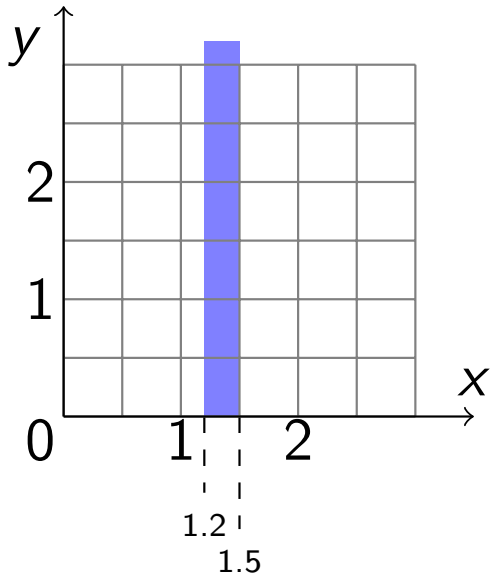


Optimal Controllers - 2

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

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

Assume we want to reach the target with cost ≤ 15.



– Zones are not precise enough to represent the state space.

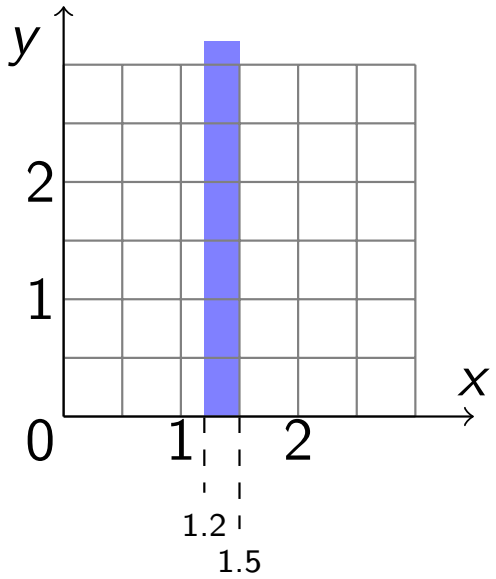
– Weighted timed games require arbitrarily precise polyhedra...

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 ≤ 15.



– Zones are not precise enough to represent the state space.

– Weighted timed games require arbitrarily precise polyhedra...

$\rightarrow$ Successor computation more complicated

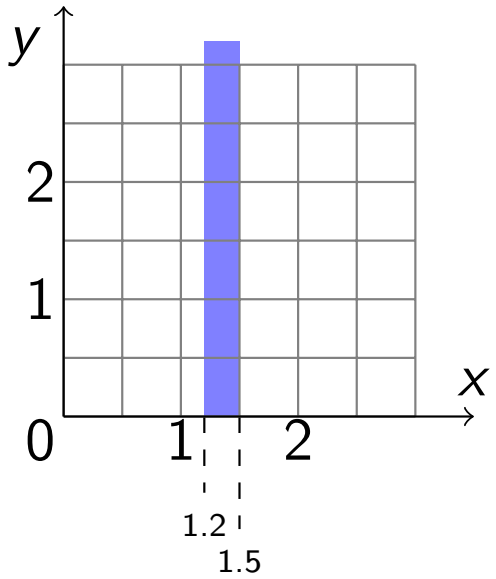
$\rightarrow$ Fixpoint computation does not terminate in general

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 ≤ 15.



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

→ Successor computation more complicated
→ Fixpoint computation does not terminate in general

→ Undecidable but decidable for some classes with a single clock
→ Optimal cost can be approximated under some assumptions

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 (see www.uppaal.org)

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.