Timed Games

Patrick Jungblut

Universität des Saarlandes

July 03rd, 2008

Talk outline

Motivation and computational model

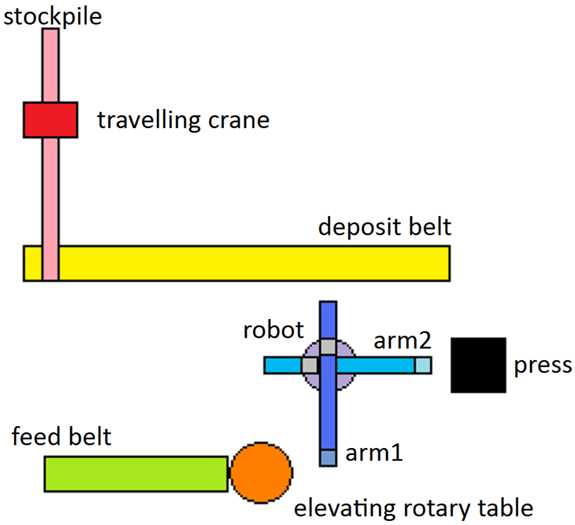
- Example
- Timed Game Automaton
- Playing and winning a Timed Game

Solving Timed Games

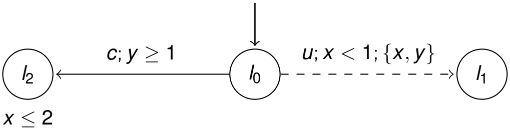
- Backward fixpoint iteration
- TiGa: An On-The-Fly algorithm

Motivation and computational model

Example: a Production Cell

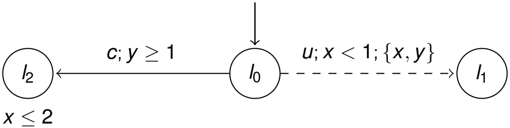


Timed Game Automata



A Timed Game Automaton TGA is a tuple (L, l_0, Inv, Act, X, T) where:

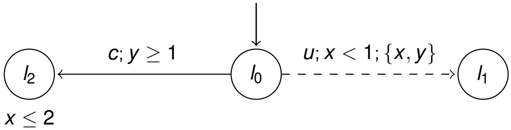
- L is a finite set of locations
- $l_0 \in L$ is the initial location
- Inv is a function, which assigns to each location its invariant.
- $Act = Act_c \cup Act_u$ is a set of actions



A Timed Game Automaton *TGA* is a tuple (L, l_0, Inv, Act, X, T) where:

- L is a finite set of locations
- $l_0 \in L$ is the initial location
- Inv is a function, which assigns to each location its invariant.
- $Act = Act_c \cup Act_u$ is a set of actions

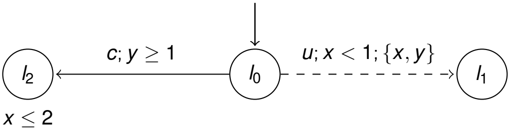
Timed Game Automata



A Timed Game Automaton *TGA* is a tuple (L, l_0, Inv, Act, X, T) where:

- L is a finite set of locations
- $l_0 \in L$ is the initial location
- Inv is a function, which assigns to each location its invariant.
- $Act = Act_c \cup Act_u$ is a set of actions

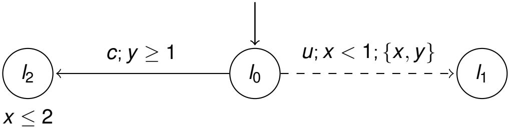
Timed Game Automata



A Timed Game Automaton *TGA* is a tuple (L, l_0, Inv, Act, X, T) where:

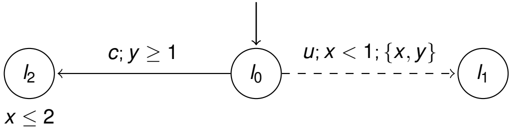
- L is a finite set of locations
- $l_0 \in L$ is the initial location
- Inv is a function, which assigns to each location its invariant.
- $Act = Act_c \cup Act_u$ is a set of actions

Timed Game Automata (continued)



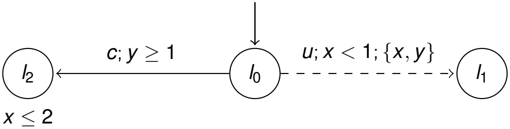
- X is a set of real-valued clocks
- $T \subseteq (L \times Act \times g \times Reset \times L)$ is a set of transitions, where
  - g is a clock constraint built by: $g = x \circ c | x_1 - x_2 \circ c | g_1 \wedge g_2$ where $x, x_1, x_2 \in X$ are clocks, $c \in \mathbb{N}$ some constant, $\circ \in \{<, \leq, =, \geq, >\}$ and g_1, g_2 clock constraints
  - $Reset \subseteq X$ is the set of clocks to reset

Timed Game Automata (continued)



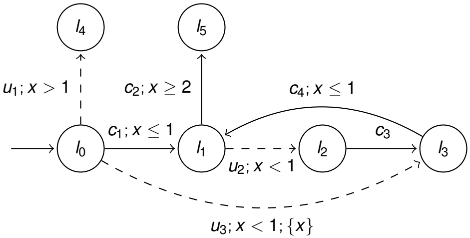
- X is a set of real-valued clocks
- $T \subseteq (L \times Act \times g \times Reset \times L)$ is a set of transitions, where
  - g is a clock constraint built by: $g = x \circ c | x_1 - x_2 \circ c | g_1 \wedge g_2$ where $x, x_1, x_2 \in X$ are clocks, $c \in \mathbb{N}$ some constant, $\circ \in \{<, \leq, =, \geq, >\}$ and g_1, g_2 clock constraints
  - $Reset \subseteq X$ is the set of clocks to reset

Timed Game Automata (continued)



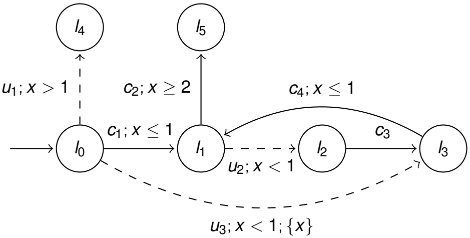
- X is a set of real-valued clocks
- $T \subseteq (L \times Act \times g \times Reset \times L)$ is a set of transitions, where
  - g is a clock constraint built by: $g = x \circ c | x_1 - x_2 \circ c | g_1 \wedge g_2$ where $x, x_1, x_2 \in X$ are clocks, $c \in \mathbb{N}$ some constant, $\circ \in \{<, \leq, =, \geq, >\}$ and g_1, g_2 clock constraints
  - $Reset \subseteq X$ is the set of clocks to reset

Playing a Timed Game



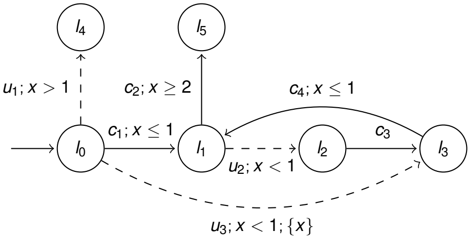
- A Timed Game is a 2 player Game
- A Timed Game Automaton is the "board" of the game
- Player *Controller* controls Act_c
- Player *Environment* controls Act_u
- *Environment* can preempt *Controller*

Playing a Timed Game



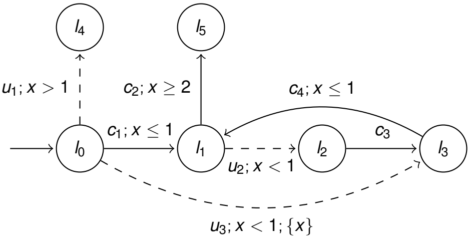
- A Timed Game is a 2 player Game
- A Timed Game Automaton is the "board" of the game
- Player *Controller* controls Act_c
- Player *Environment* controls Act_u
- *Environment* can preempt *Controller*

Playing a Timed Game



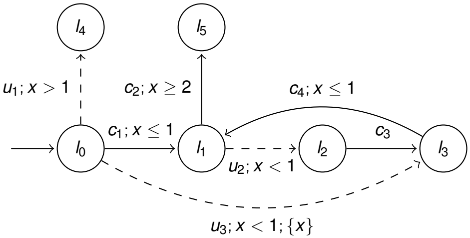
- A Timed Game is a 2 player Game
- A Timed Game Automaton is the "board" of the game
- Player *Controller* controls Act_c
- Player *Environment* controls Act_u
- *Environment* can preempt *Controller*

Playing a Timed Game



- A Timed Game is a 2 player Game
- A Timed Game Automaton is the "board" of the game
- Player *Controller* controls Act_c
- Player *Environment* controls Act_u
- *Environment* can preempt *Controller*

Playing a Timed Game



- A Timed Game is a 2 player Game
- A Timed Game Automaton is the ”board” of the game
- Player *Controller* controls Act_c
- Player *Environment* controls Act_u
- *Environment* can preempt *Controller*

Playing a Timed Game (continued)

Moves

At a location $l \in L$ at a clock-valuation $\vec{t} \in \mathbb{R}^X_{\geq 0}$ a player P has two possibilities

1. Using a transition $t = (l, \alpha, g, R, l')$, if $\vec{t} \models g$, $\alpha \in Act_P$, and $\vec{t}[R] \models Inv(l')$, where $\vec{t}[R]$ is the clock-valuation resulting from $\vec{t}$ by setting all clocks in R to 0.
2. Waiting

Timed state space

- $S \subseteq L \times \mathbb{R}^X_{\geq 0}$ is the set of timed states
- $\mathbb{R}^X_{\geq 0}$ is infinite $\Rightarrow$ so is S

Playing a Timed Game (continued)

Moves

At a location $l \in L$ at a clock-valuation $\vec{t} \in \mathbb{R}^X_{\geq 0}$ a player P has two possibilities

1. Using a transition $t = (l, \alpha, g, R, l')$, if $\vec{t} \models g$, $\alpha \in Act_P$, and $\vec{t}[R] \models Inv(l')$, where $\vec{t}[R]$ is the clock-valuation resulting from $\vec{t}$ by setting all clocks in R to 0.
2. Waiting

Timed state space

- $S \subseteq L \times \mathbb{R}^X_{\geq 0}$ is the set of timed states
- $\mathbb{R}^X_{\geq 0}$ is infinite $\Rightarrow$ so is S

Playing a Timed Game (continued)

Memoryless strategy

A memoryless (state-based) strategy $f_P : S = L \times \mathbb{R}^X_{\geq 0} \rightarrow Act_P \cup \{\lambda\}$ for a player P is a partial function s.t.

- $f_P(s) = a$ for some $s \in S$ and $a \in Act_P$, if P has to use a
- $f_P(s) = \lambda$, if P has to let time pass

A strategy f_P is called winning, iff P always wins the Timed Game following f_P.

Winning a Timed Game

[MPS '95] describes 4 winning conditions for a Timed Game:

Let $G \subseteq L$ be a set of goal locations.

- Controller: $\Diamond G$
  Controller wins if he can enforce to reach G
- Controller: $\Box G$
  Controller wins if he can enforce not to leave G
- Controller: $\Diamond \Box G$
  Controller wins if he can enforce to finally stay in G
- Controller: $\Box \Diamond G$
  Controller wins if he can enforce to reach G infinitely often

Winning a Timed Game

[MPS '95] describes 4 winning conditions for a Timed Game:

Let $G \subseteq L$ be a set of goal locations.

- Controller: $\Diamond G$
  Controller wins if he can enforce to reach G
- Controller: $\Box G$
  Controller wins if he can enforce not to leave G
- Controller: $\Diamond\Box G$
  Controller wins if he can enforce to finally stay in G
- Controller: $\Box\Diamond G$
  Controller wins if he can enforce to reach G infinitely often

Winning a Timed Game

[MPS '95] describes 4 winning conditions for a Timed Game:

Let $G \subseteq L$ be a set of goal locations.

- Controller: $\Diamond G$
  Controller wins if he can enforce to reach G
- Controller: $\Box G$
  Controller wins if he can enforce not to leave G
- Controller: $\Diamond\Box G$
  Controller wins if he can enforce to finally stay in G
- Controller: $\Box\Diamond G$
  Controller wins if he can enforce to reach G infinitely often

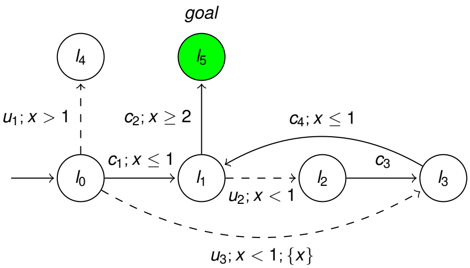
Winning a Timed Game

[MPS '95] describes 4 winning conditions for a Timed Game:

Let $G \subseteq L$ be a set of goal locations.

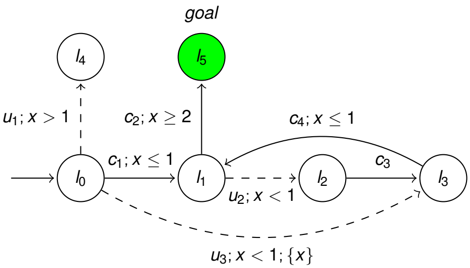
- Controller: $\Diamond G$
  Controller wins if he can enforce to reach G
- Controller: $\Box G$
  Controller wins if he can enforce not to leave G
- Controller: $\Diamond\Box G$
  Controller wins if he can enforce to finally stay in G
- Controller: $\Box\Diamond G$
  Controller wins if he can enforce to reach G infinitely often

Focus on $\Diamond G$



- *Controller* tries to reach some dedicated goal location
- *Environment* tries to prevent that

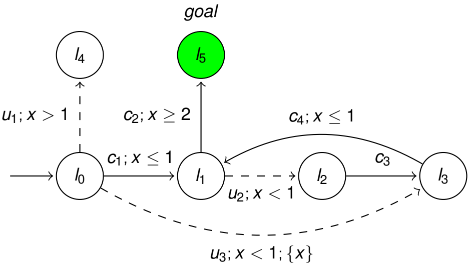
Focus on $\Diamond G$



- *Controller* tries to reach some dedicated goal location
- *Environment* tries to prevent that

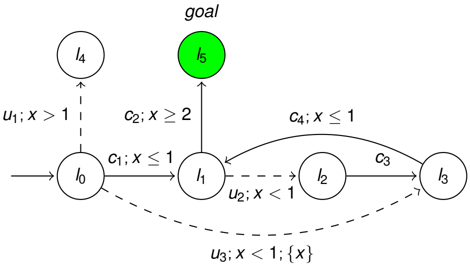
Solving Timed Games

Backward fixpoint iteration [MPS '95]



- $win_0 := \text{goal} \times \mathbb{R}^X_{\geq 0}$
- $win_{i+1} := win_i \cup Pre_{enf}(win_i)$

Backward fixpoint iteration [MPS '95]



- $win_0 := \text{goal} \times \mathbb{R}^X_{\geq 0}$
- $win_{i+1} := win_i \cup Pre_{enf}(win_i)$

The $Pre_{enf}(win)$ operator

A state $s = (l, x) \in S$ is in $Pre_{enf}(win)$ iff:

- $\exists s' = (l, x') \in win$ for some $x' > x$ and $\forall x \leq x'' \leq x'$ holds $\nexists t = (l, \alpha, g, R, l') \in T$ s.t. $\alpha \in Act_u$ and $x \models g$ and $(l', x''[R]) \notin win$ or
- $\exists s' = (l', x') \in win$ s.t. $\exists x'' > x$ and $t = (l, \alpha, g, R, l') \in T$ s.t. $x' = x''[R]$ and $\forall x \leq x''' \leq x''$ holds $\nexists t' = (l, \alpha', g', R', l'') \in T$ s.t. $\alpha' \in Act_u$ and $x \models g$ and $(l'', x'''[R]) \notin win$

Notation:

Let x, y be clock-valuations, we say $x \leq y$ if $\exists \delta \in \mathbb{R}_{\geq 0}$ s.t. $y = x + \delta\vec{1}$

The $Pre_{enf}(win)$ operator

A state $s = (l, x) \in S$ is in $Pre_{enf}(win)$ iff:

- $\exists s' = (l, x') \in win$ for some $x' > x$ and $\forall x \leq x'' \leq x'$ holds $\nexists t = (l, \alpha, g, R, l') \in T$ s.t. $\alpha \in Act_u$ and $x \models g$ and $(l', x''[R]) \notin win$ or
- $\exists s' = (l', x') \in win$ s.t. $\exists x'' > x$ and $t = (l, \alpha, g, R, l') \in T$ s.t. $x' = x''[R]$ and $\forall x \leq x''' \leq x''$ holds $\nexists t' = (l, \alpha', g', R', l'') \in T$ s.t. $\alpha' \in Act_u$ and $x \models g$ and $(l'', x'''[R]) \notin win$

Notation:

Let x, y be clock-valuations, we say $x \leq y$ if $\exists \delta \in \mathbb{R}_{\geq 0}$ s.t. $y = x + \delta\vec{1}$

The $Pre_{enf}(win)$ operator

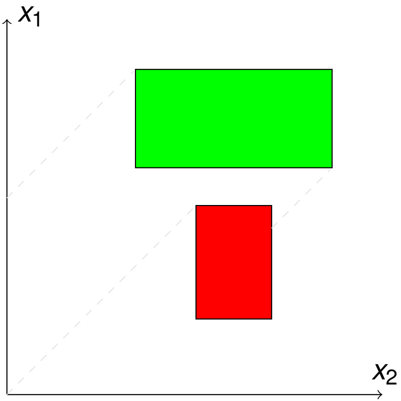
A state $s = (l, x) \in S$ is in $Pre_{enf}(win)$ iff:

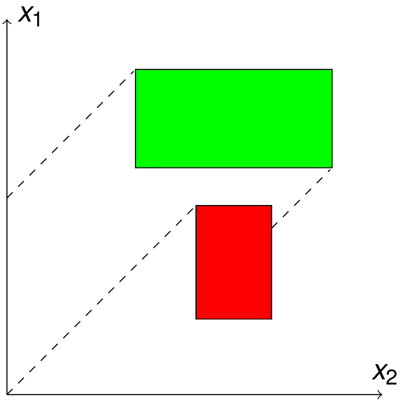
- $\exists s' = (l, x') \in win$ for some $x' > x$ and $\forall x \leq x'' \leq x'$ holds $\nexists t = (l, \alpha, g, R, l') \in T$ s.t. $\alpha \in Act_u$ and $x \models g$ and $(l', x''[R]) \notin win$ or
- $\exists s' = (l', x') \in win$ s.t. $\exists x'' > x$ and $t = (l, \alpha, g, R, l') \in T$ s.t. $x' = x''[R]$ and $\forall x \leq x''' \leq x''$ holds $\nexists t' = (l, \alpha', g', R', l'') \in T$ s.t. $\alpha' \in Act_u$ and $x \models g$ and $(l'', x'''[R]) \notin win$

Notation:

Let x, y be clock-valuations, we say $x \leq y$ if $\exists \delta \in \mathbb{R}_{\geq 0}$ s.t. $y = x + \delta\vec{1}$

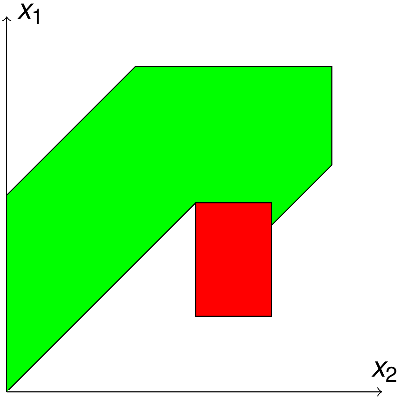
The $Pre_{enf}(win)$ operator





x_1
x_2

The $Pre_{enf}(win)$ operator



Clock Zones

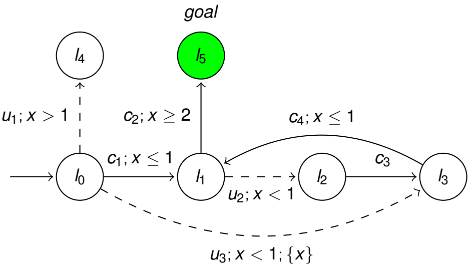
- Infinite statespace
  $\Rightarrow$ finite symbolic representation needed.
- [Alur '99] proposes a conjunction of inequalities called *Clock Zones*

$$\bigwedge x_i \circ c_i \wedge \bigwedge x_i - x_j \circ c_{ij}$$

$x_i, x_j \in X$, $c_i, c_{ij} \in \mathbb{N} \cup \{+\infty\}$, $\circ \in \{<, \leq, \geq, >\}$

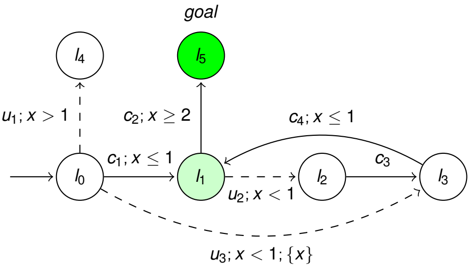
- A Clock Zone is a convex polyhedron
- Efficient matrix based data structure: DBM
- A Federation is a not necessarily convex union of Clock Zones.
- [CDFLL '05]: compute $Pre_{enf}(win)$ using Federations

Initialization of fixpoint iteration



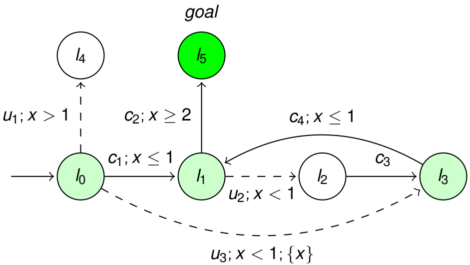
- $l_0 : \emptyset$
- $l_1 : \emptyset$
- $l_2 : \emptyset$
- $l_3 : \emptyset$
- $l_4 : \emptyset$
- $l_5 : \mathbb{R}_{\geq 0}$

Step 1:



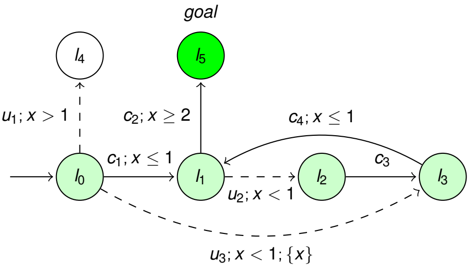
- l_0 : $\emptyset$
- l_1 : $[1, \infty[$
- l_2 : $\emptyset$
- l_3 : $\emptyset$
- l_4 : $\emptyset$
- l_5 : $\mathbb{R}_{\geq 0}$

Step 2:



- $l_0 \;:\; \{1\}$
- $l_1 \;:\; [1, \infty[$
- $l_2 \;:\; \emptyset$
- $l_3 \;:\; [0, 1]$
- $l_4 \;:\; \emptyset$
- $l_5 \;:\; \mathbb{R}_{\geq 0}$

Step 3:



- l_0 : $[0, 1]$
- l_1 : $[1, \infty[$
- l_2 : $[0, 1]$
- l_3 : $[0, 1]$
- l_4 : $\emptyset$
- l_5 : $\mathbb{R}_{\geq 0}$

Disadvantages of a pure backward approach

- Each step of the iteration is expensive
- Non-reachability of goal state will not be noticed until fixpoint is reached
- The whole statespace has to be known, but the statespace can be huge

On-the-fly Timed Game Solving [CDFLL '05]

Initialization

- Start in the initial state
- Feed a waiting queue q with the outgoing transitions of the initial state

The loop

After initialization we start the loop:

1. As long as q is not empty: take a transition t from q
2. Analyse the target state s' of t:
   - If we meet s' for the first time: start a forward step
   - If we already met s' before: start a backward step

On-the-fly Timed Game Solving [CDFLL '05]

Initialization

- Start in the initial state
- Feed a waiting queue q with the outgoing transitions of the initial state

The loop

After initialization we start the loop:

1. As long as q is not empty: take a transition t from q
2. Analyse the target state s' of t:
   - If we meet s' for the first time: start a forward step
   - If we already met s' before: start a backward step

On-the-fly Timed Game Solving [CDFLL '05]

Initialization

- Start in the initial state
- Feed a waiting queue q with the outgoing transitions of the initial state

The loop

After initialization we start the loop:

1. As long as q is not empty: take a transition t from q
2. Analyse the target state s' of t:
   - If we meet s' for the first time: start a forward step
   - If we already met s' before: start a backward step

On-the-fly Timed Game Solving [CDFLL '05]

Initialization

- Start in the initial state
- Feed a waiting queue q with the outgoing transitions of the initial state

The loop

After initialization we start the loop:

1. As long as q is not empty: take a transition t from q
2. Analyse the target state s' of t:
    - If we meet s' for the first time: start a forward step
    - If we already met s' before: start a backward step

On-the-fly Timed Game Solving [CDFLL '05]

Initialization

- Start in the initial state
- Feed a waiting queue q with the outgoing transitions of the initial state

The loop

After initialization we start the loop:

1. As long as q is not empty: take a transition t from q
2. Analyse the target state s' of t:
   - If we meet s' for the first time: start a forward step
   - If we already met s' before: start a backward step

On-the-fly Timed Game Solving [CDFLL '05]

Initialization

- Start in the initial state
- Feed a waiting queue q with the outgoing transitions of the initial state

The loop

After initialization we start the loop:

1. As long as q is not empty: take a transition t from q
2. Analyse the target state s' of t:
   - If we meet s' for the first time: start a forward step
   - If we already met s' before: start a backward step

On-the-fly Timed Game Solving (continued)

Forward step

1. s' is the goal state? If yes, add t to q
2. Add all outgoing transitions of s' to q

Backward step

1. Propagate winning information from s' back to the source s of t using Pre_{enf}
2. If the winning information of s changes by this, we add the incoming transitions to s to the queue

On-the-fly Timed Game Solving (continued)

Forward step

1. s' is the goal state? If yes, add t to q
2. Add all outgoing transitions of s' to q

Backward step

1. Propagate winning information from s' back to the source s of t using Pre_{enf}
2. If the winning information of s changes by this, we add the incoming transitions to s to the queue

On-the-fly Timed Game Solving (continued)

Forward step

1. s' is the goal state? If yes, add t to q
2. Add all outgoing transitions of s' to q

Backward step

1. Propagate winning information from s' back to the source s of t using Pre_{enf}
2. If the winning information of s changes by this, we add the incoming transitions to s to the queue

On-the-fly Timed Game Solving (continued)

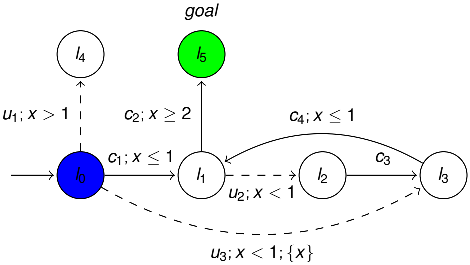
Forward step

1. s' is the goal state? If yes, add t to q
2. Add all outgoing transitions of s' to q

Backward step

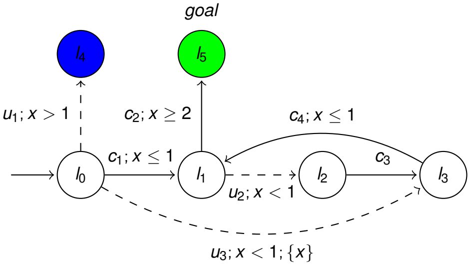
1. Propagate winning information from s' back to the source s of t using Pre_{enf}
2. If the winning information of s changes by this, we add the incoming transitions to s to the queue

Initialization



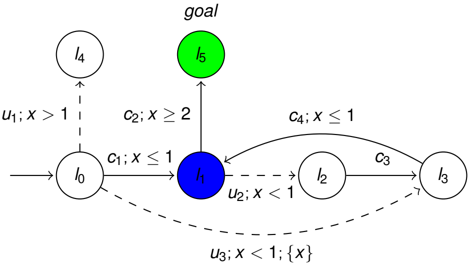
- $S = \{(l_0, \mathbb{R}_{\geq 0})\}$
- $q = \{((l_0, \mathbb{R}_{\geq 0}), u_1, (l_4,]1, \infty[)), ((l_0, \mathbb{R}_{\geq 0}), c_1, (l_1, \mathbb{R}_{\geq 0})), ((l_0, \mathbb{R}_{\geq 0}), u_3, (l_3, \mathbb{R}_{\geq 0}))\}$

Step 1:



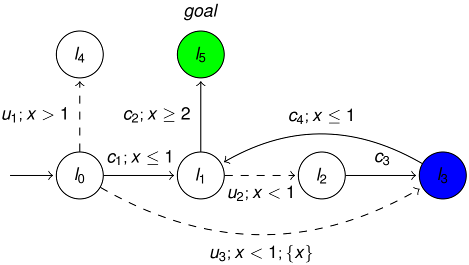
- $S = \{(l_0, \mathbb{R}_{\geq 0}), (l_4,]1, \infty[)\}$
- $q = \{((l_0, \mathbb{R}_{\geq 0}), c_1, (l_1, \mathbb{R}_{\geq 0})), ((l_0, \mathbb{R}_{\geq 0}), u_3, (l_3, \mathbb{R}_{\geq 0}))\}$

Step 2:



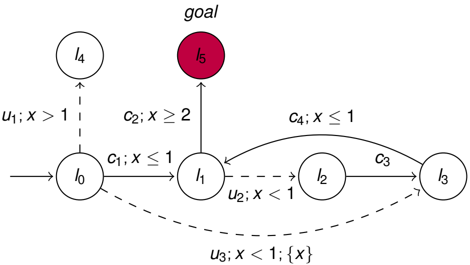
- $S = \{(l_0, \mathbb{R}_{\geq 0}), (l_4,]1, \infty[), (l_1, \mathbb{R}_{\geq 0})\}$
- $q = \{((l_0, \mathbb{R}_{\geq 0}), u_3, (l_3, \mathbb{R}_{\geq 0})), ((l_1, \mathbb{R}_{\geq 0}), c_2, (l_5, [2, \infty[)), ((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0}))\}$

Step 3:



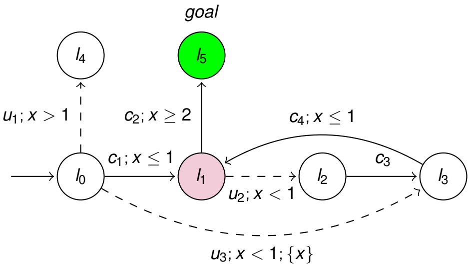
- $S = \{(l_0, \mathbb{R}_{\geq 0}), (l_4,]1, \infty[), (l_1, \mathbb{R}_{\geq 0}), (l_3, \mathbb{R}_{\geq 0})\}$
- $q = \{((l_1, \mathbb{R}_{\geq 0}), c_2, (l_5, [2, \infty[)), ((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0}))$
  $((l_3, \mathbb{R}_{\geq 0}), c_4, (l_1, \mathbb{R}_{\geq 0}))\}$

Step 4:



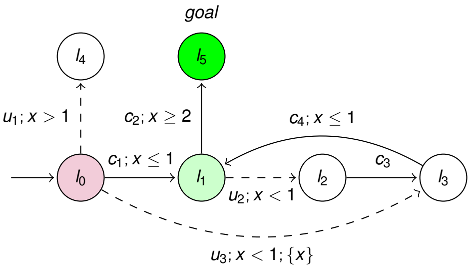
- $S = \{(l_0, \mathbb{R}_{\geq 0}), (l_4,]1, \infty[), (l_1, \mathbb{R}_{\geq 0}), (l_3, \mathbb{R}_{\geq 0}), (l_5, [2, \infty[)\}$
- $q = \{((l_1, \mathbb{R}_{\geq 0}), c_2, (l_5, [2, \infty[)), ((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0}))$ $((l_3, \mathbb{R}_{\geq 0}), c_4, (l_1, \mathbb{R}_{\geq 0}))\}$
- $win((l_5, [2, \infty[)) = [2, \infty[$

Step 5:



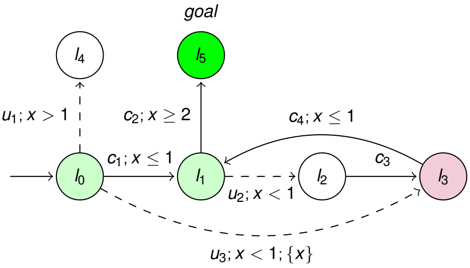
- $q = \{\big((l_0, \mathbb{R}_{\geq 0}), c_1, (l_1, \mathbb{R}_{\geq 0})\big), \big((l_3, \mathbb{R}_{\geq 0}), c_4, (l_1, \mathbb{R}_{\geq 0})\big)$ $\big((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0})\big)\}$
- $win\big((l_5, [2, \infty[)\big) = [2, \infty[$
- $win\big((l_1, \mathbb{R}_{\geq 0})\big) = [1, \infty[$

Step 6:



- $q = \{((l_3, \mathbb{R}_{\geq 0}), c_4, (l_1, \mathbb{R}_{\geq 0})), ((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0}))\}$
- $win((l_5, [2, \infty[)) = [2, \infty[$
- $win((l_1, \mathbb{R}_{\geq 0})) = [1, \infty[$
- $win((l_0, \mathbb{R}_{\geq 0})) = \{1\}$

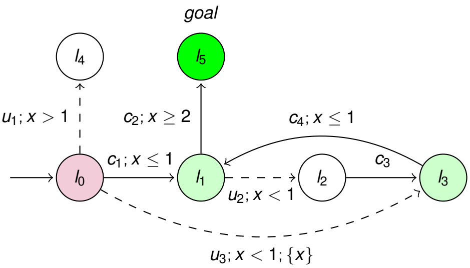
Step 7:



- $q = \{((l_0, \mathbb{R}_{\geq 0}), u_3, (l_3, \mathbb{R}_{\geq 0})), ((l_3, \mathbb{R}_{\geq 0}), c_4, (l_1, \mathbb{R}_{\geq 0})), ((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0}))\}$

- $win((l_5, [2, \infty[)) = [2, \infty[$
- $win((l_1, \mathbb{R}_{\geq 0})) = [1, \infty[$
- $win((l_0, \mathbb{R}_{\geq 0})) = \{1\}$
- $win((l_3, \mathbb{R}_{\geq 0})) = \mathbb{R}_{\geq 0}$

Step 8:



- $q = \{((l_3, \mathbb{R}_{\geq 0}), c_4, (l_1, \mathbb{R}_{\geq 0})), ((l_1, \mathbb{R}_{\geq 0}), u_2, (l_2, \mathbb{R}_{\geq 0}))\}$

- $win((l_5, [2, \infty[)) = [2, \infty[$
- $win((l_1, \mathbb{R}_{\geq 0})) = [1, \infty[$
- $win((l_0, \mathbb{R}_{\geq 0})) = [0, 1]$
- $win((l_3, \mathbb{R}_{\geq 0})) = \mathbb{R}_{\geq 0}$

Two winning conditions remain

[MPS '95] provides the following fixpoint iterations:

Controller: $\Diamond\Box G$

- $win_0 := \emptyset$, $i := 0$
- **do**
  - $help_0 := S$, $j := 0$
  - **do**
    - $help_{j+1} := Pre_{enf}(help_j) \cap (G \cup Pre_{enf}(win_i))$, $j++$
  - **while** $help_{j+1} \neq help_j$
  - $win_{i+1} := help_j$, $i++$
- **while** $win_{i+1} \neq win_i$

Two winning conditions remain (continued)

Controller: $\Box\Diamond G$

- $win_0 := S$, $i := 0$
- **do**
  - $help_0 := \emptyset$, $j := 0$
  - **do**
    - $help_{j+1} := Pre_{enf}(help_j) \cup (G \cap Pre_{enf}(win_i))$, $j++$
  - **while** $help_{j+1} \neq help_j$
  - $win_{i+1} := help_j$, $i++$
- **while** $win_{i+1} \neq win_i$

Summary

- Timed Game Automata
  - Syntax
  - Semantics
- Playing Timed Games
- Solving Timed Games for 4 types of winning conditions
- Deeper look into Reachability Games

Questions?