Real Time Controller Synthesis

with

Gerd Behrmann,
Franck Cassez, Agnes Counard, Alexandre David
Emmanuel Fleury, Didier Lime

Aalborg University Denmark
BRICS Basic Research in Computer Science
CISS Center for Indlejrede Software Systemer

RELATED SITES: UPPAAL

UPPAAL TIGA

UPPAAL for Timed Games

Main Page | Download | Contact us

Welcome!

UPPAAL TIGA (Fig. 1) is an extension of UPPAAL [BDL04] and it implements the first efficient on-the-fly algorithm for solving games based on timed game automata with respect to reachability and safety properties. Though timed games for long have been known to be decidable there has until now been a lack of efficient and truly on-the-fly algorithms for their analysis.

The algorithm we propose [CDFLL05] is a symbolic extension of the on-the-fly algorithm suggested by Liu & Smolka [LS98] for linear-time model-checking of finite-state systems. Being on-the-fly, the symbolic algorithm may terminate long before having explored the entire state-space. Also the individual steps of the algorithm are carried out efficiently by the use of so-called zones as the underlying data structure. Our tool implements various optimizations of the basic symbolic algorithm, as well as methods for obtaining time-optimal winning strategies (for

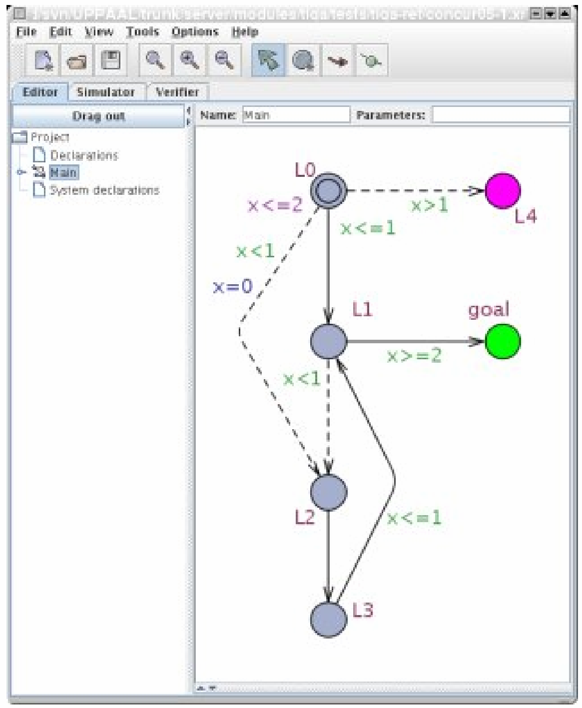


Figure 1: UPPAAL TIGA on screen.

Latest News

Versions 0.10 and 0.11 released.

7 July 2007

Versions 0.10 and 0.11 are released today. Version 0.11 contains a new concrete simulator that allows the user to play strategies from the GUI. Both versions fix the following bugs: maximal constants in the formula are now taken into account, the command line simulator is new and works better, delay when no clock was used, better user feedback, end-of-game detection fixed, other bugs involving delays in the strategy, precision problems in the simulator, and leak in the DBM library. These new versions have also the following new features: options to control the type of strategy output, better control on the search ordering (forward and backward), cooperative strategies, and time optimal strategies. The manual has been updated to reflect these new features.

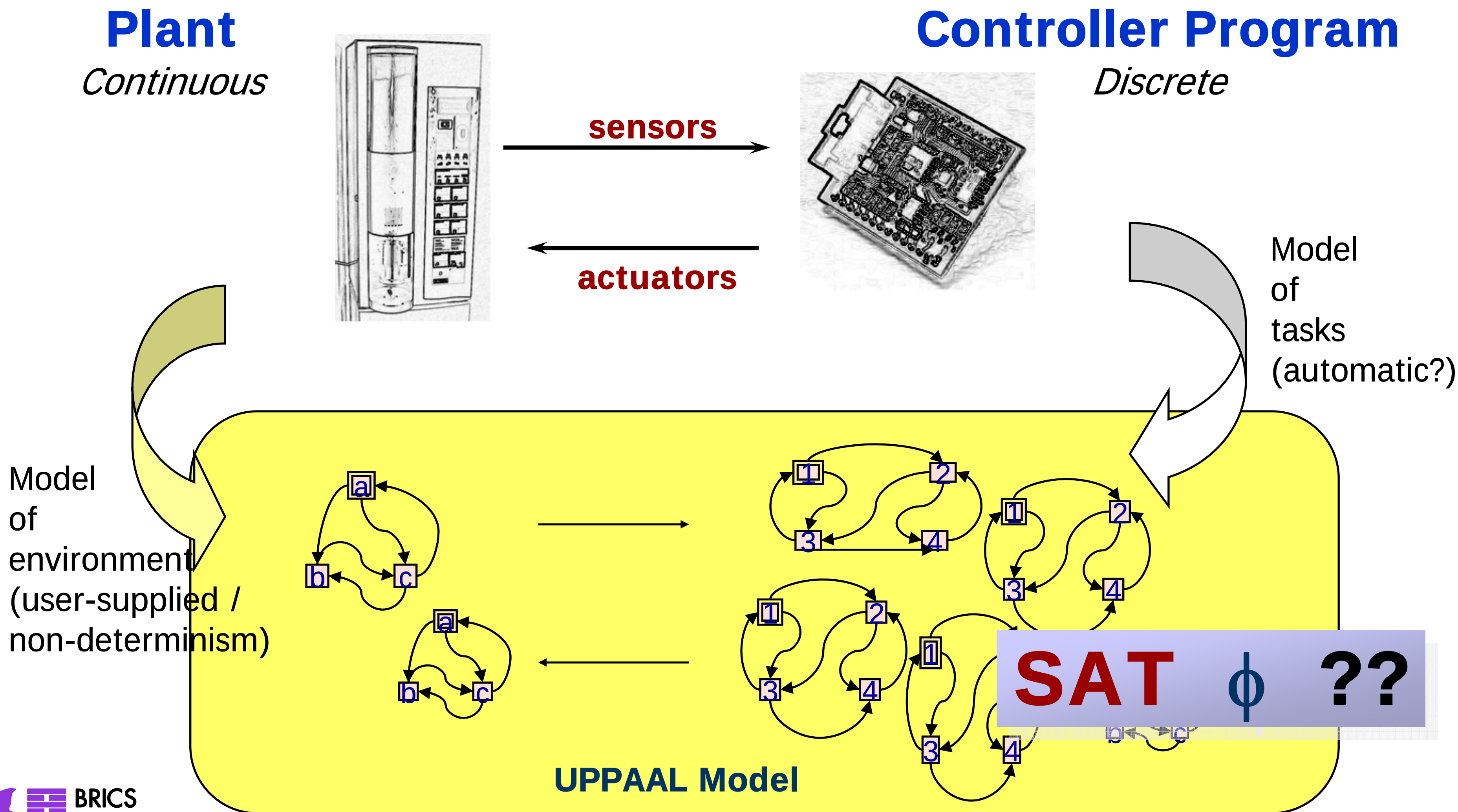
UPPSALA UNIVERSITET

AALBORG UNIVERSITY

See CAV 2007 & CONCUR 2005

BRICS Basic Research in Computer Science

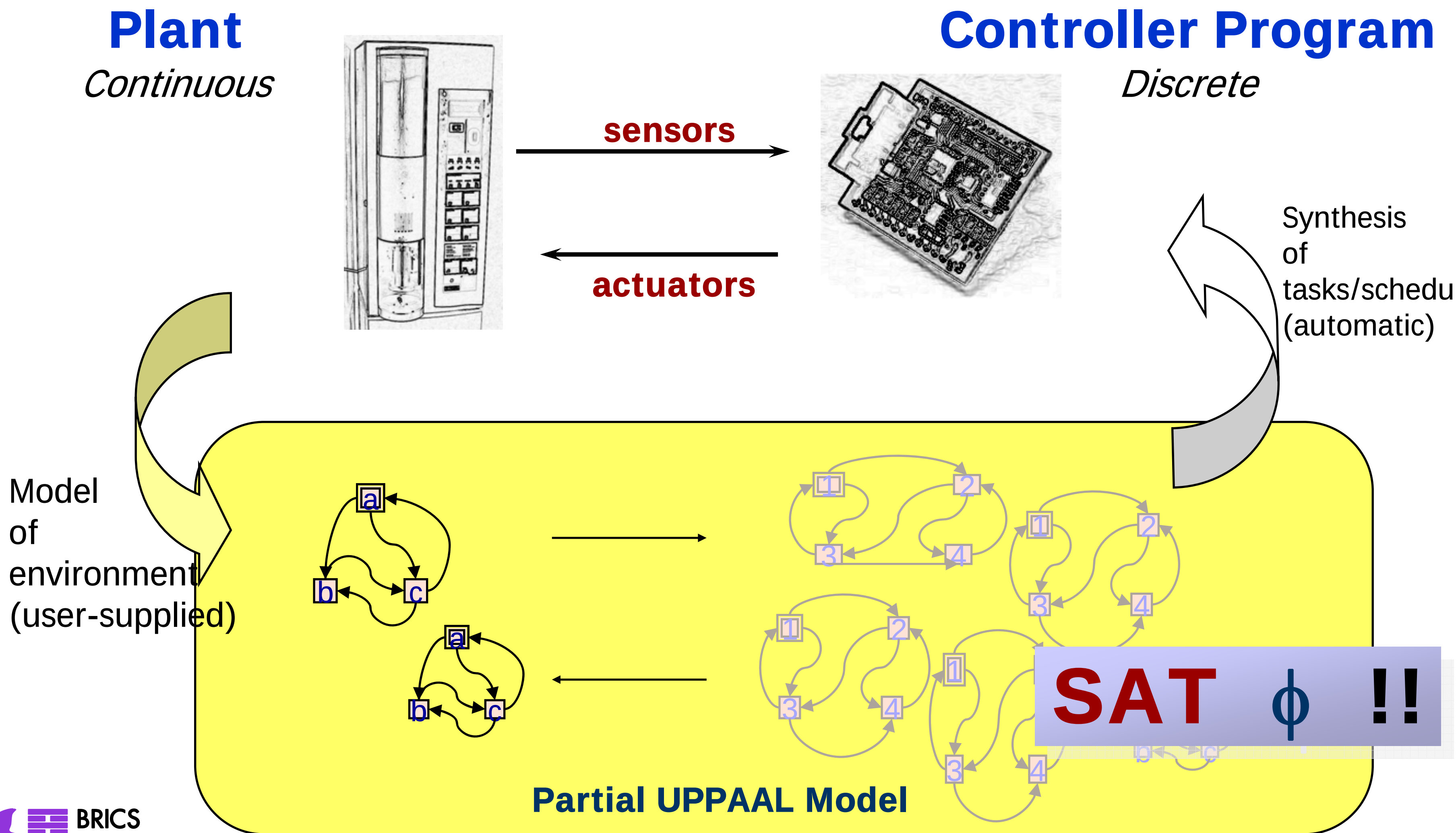
Real Time Model Checking



BRICS Basic Research in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

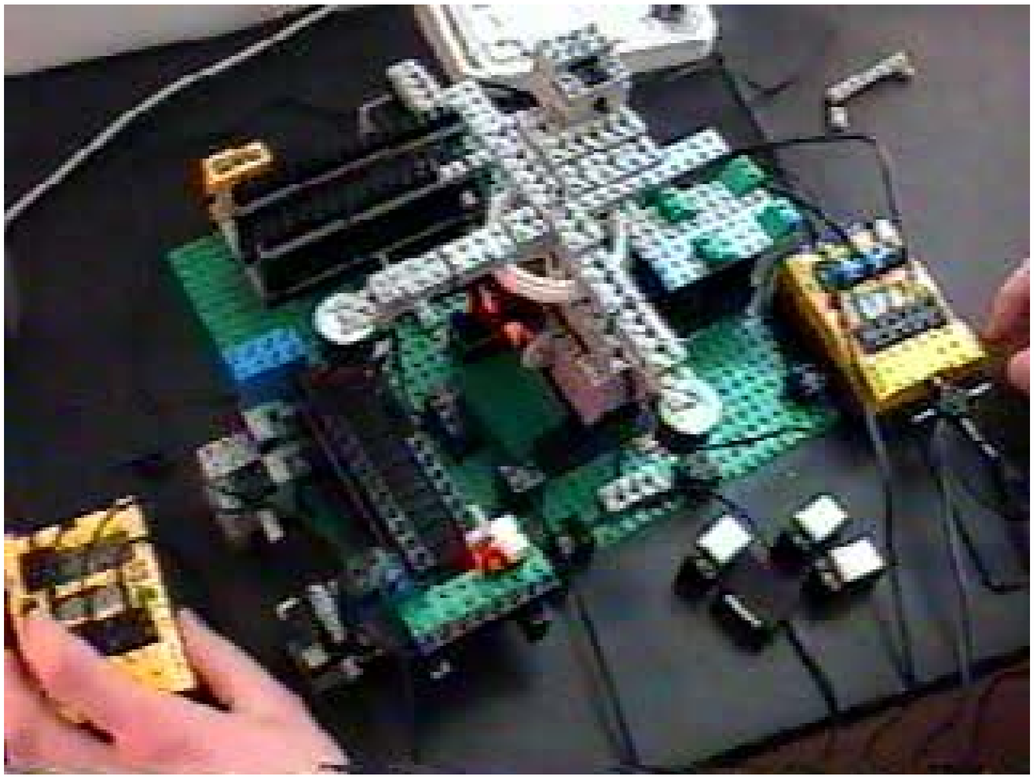
Real Time Scheduling & Control Synthesis

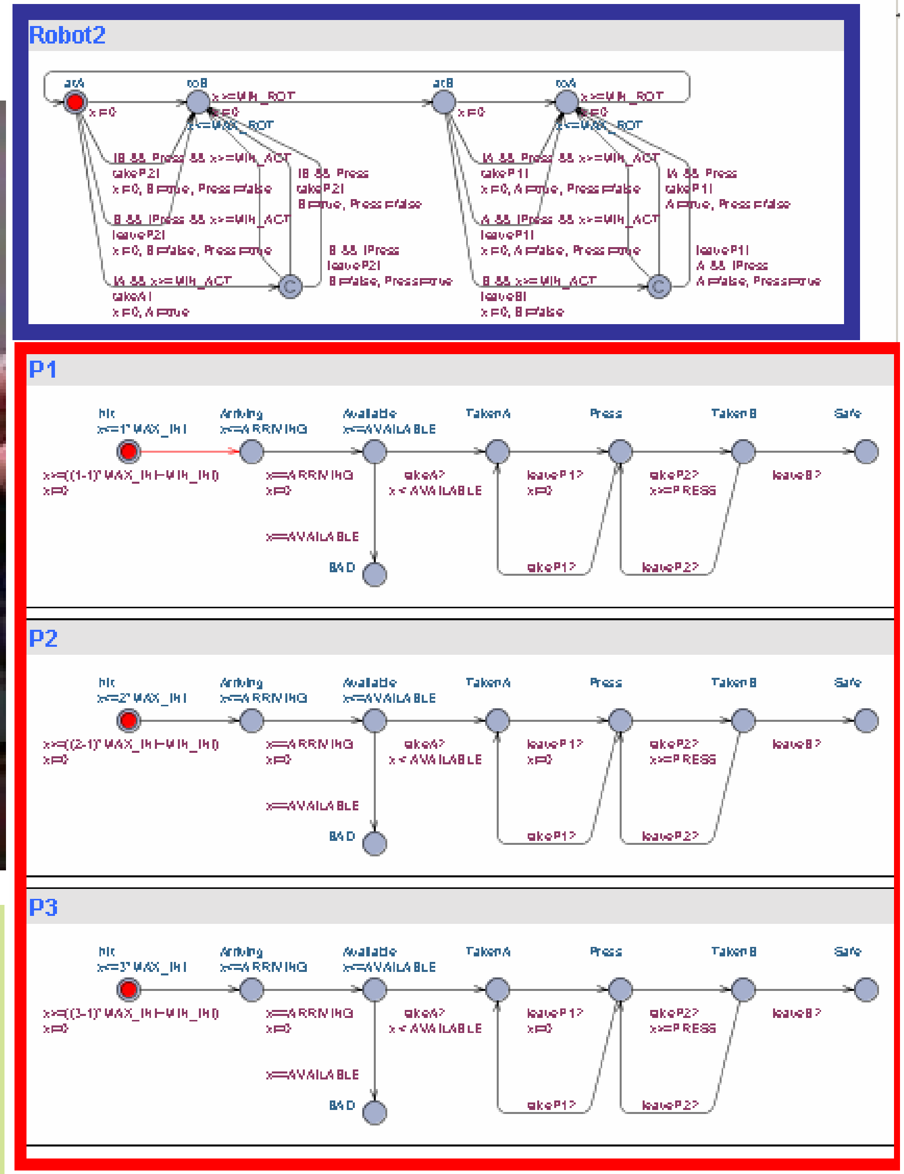


BRICS
Basic Research
in Computer Science

Controller Synthesis and Timed Games

Production Cell



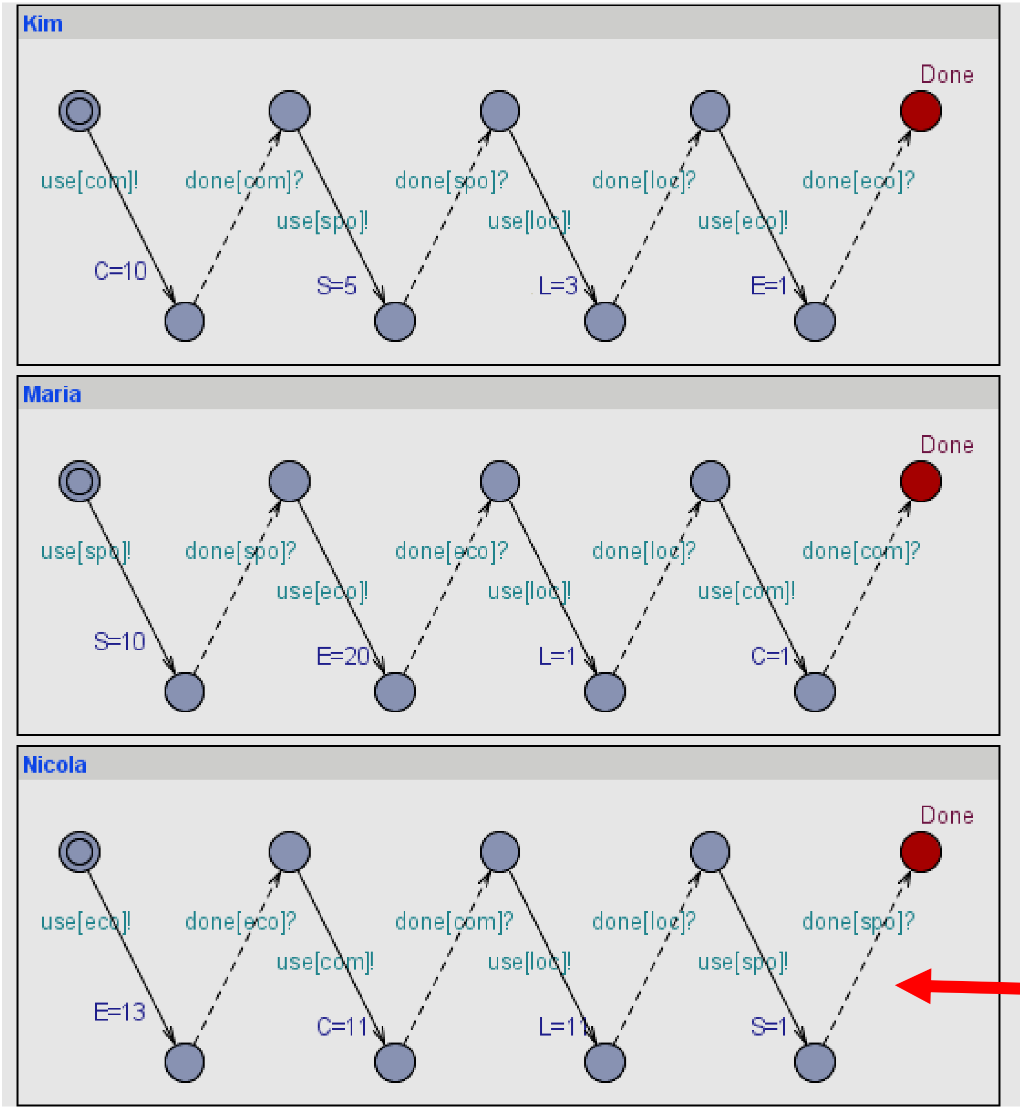


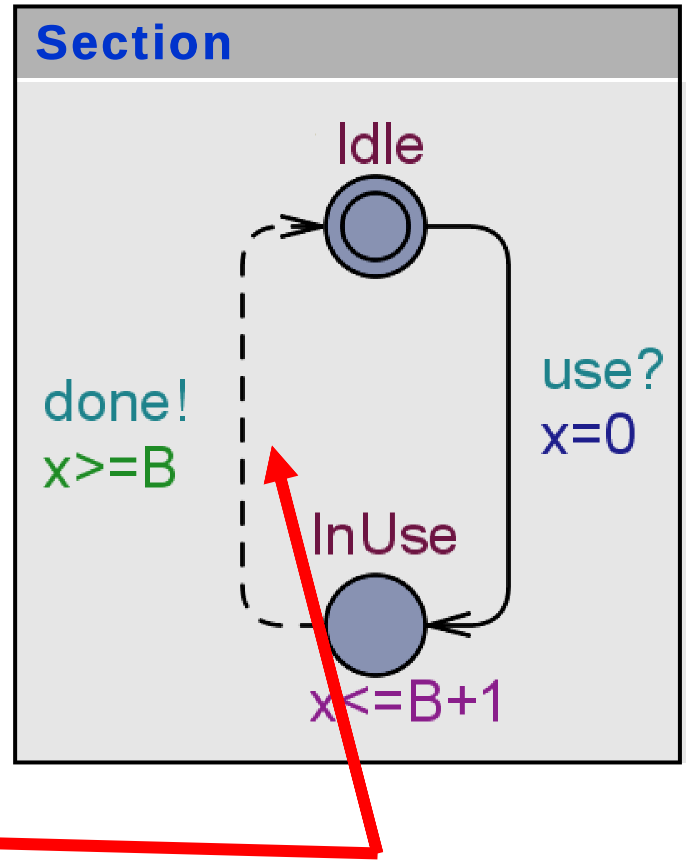
GIVEN System moves S,
Controller moves C, and property ϕ
FIND strategy s_C such that $s_C || S \models \phi$

$\rightarrow$

A Two-Player Game

Dynamic Scheduling = Controller Synthesis

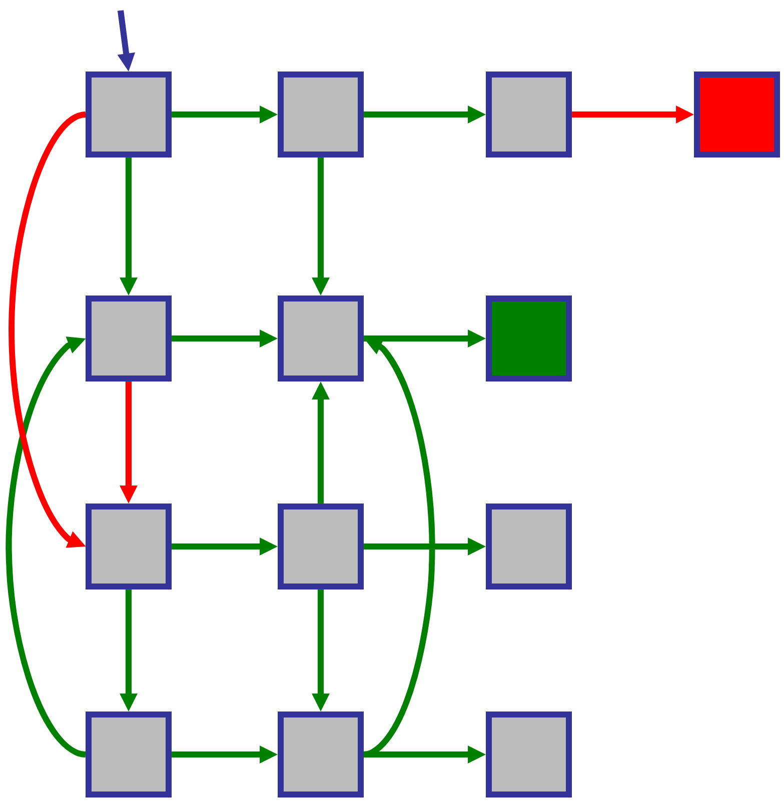


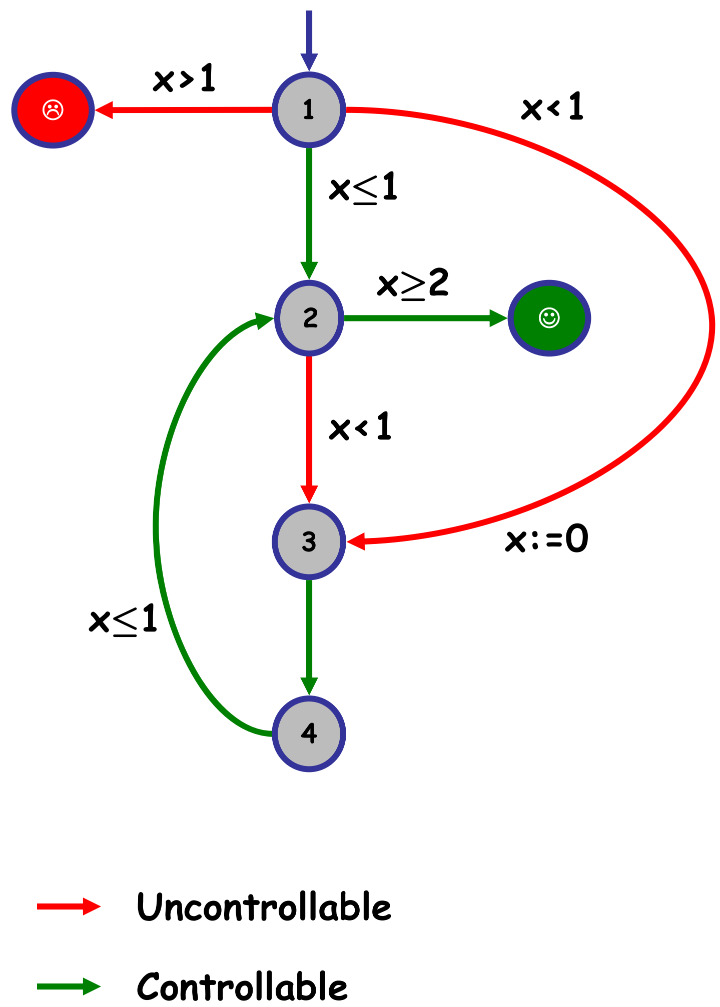


Reading time is uncontrollable

Untimed and Timed Games

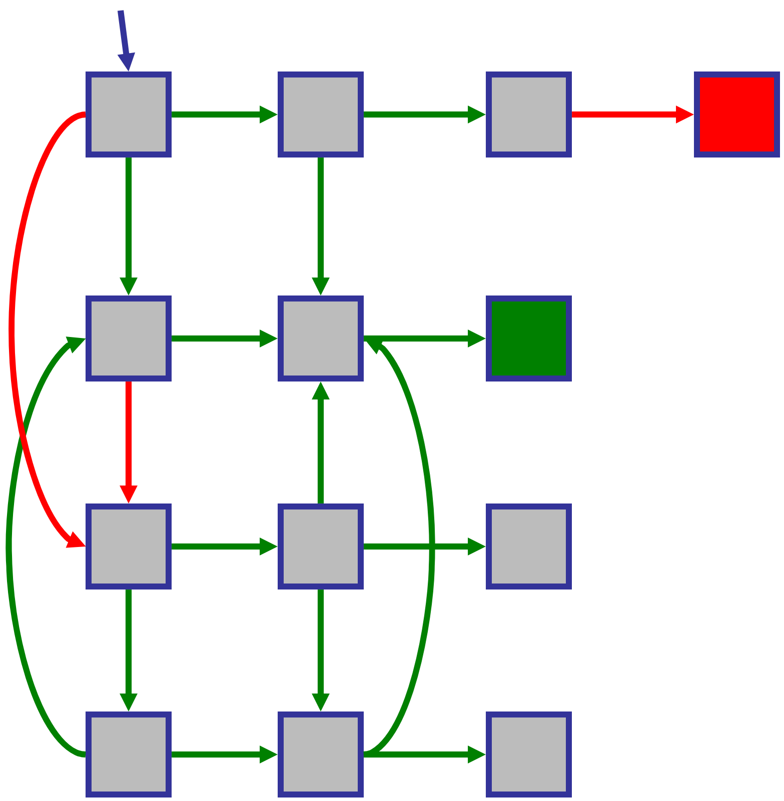
Reachability / Safety Games





Untimed Games

Reachability / Safety Games



Strategy:

$F : Run(A) \rightarrow E_c$

Memoryless strategy:

$F : Q \rightarrow E_c$

Uncontrollable

Controllable

Untimed Games

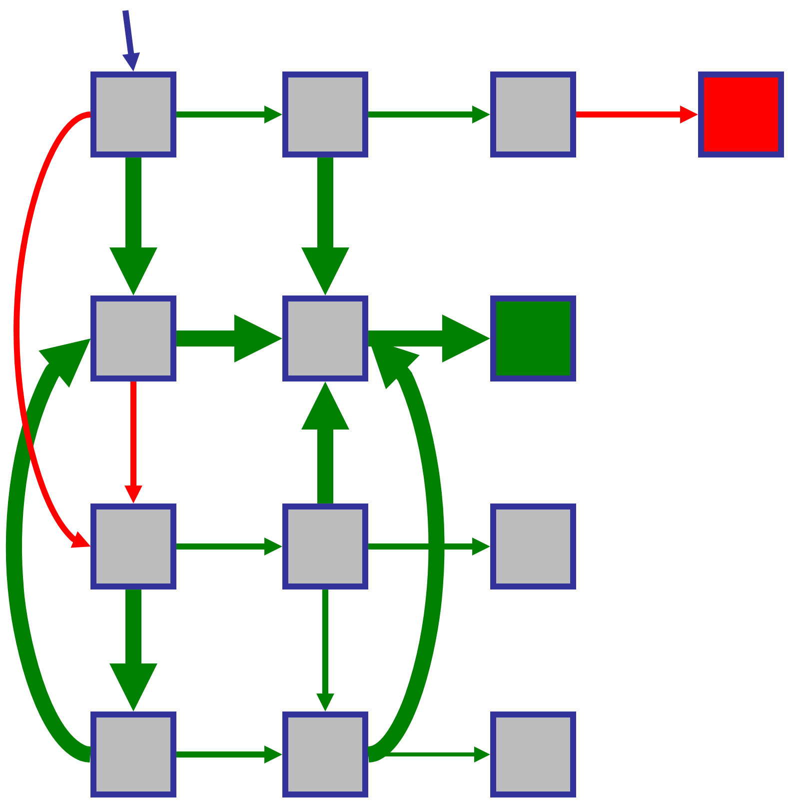
Reachability / Safety Games

Strategy:

$F : Run(A) \rightarrow E_c$

Memoryless strategy:

$F : Q \rightarrow E_c$

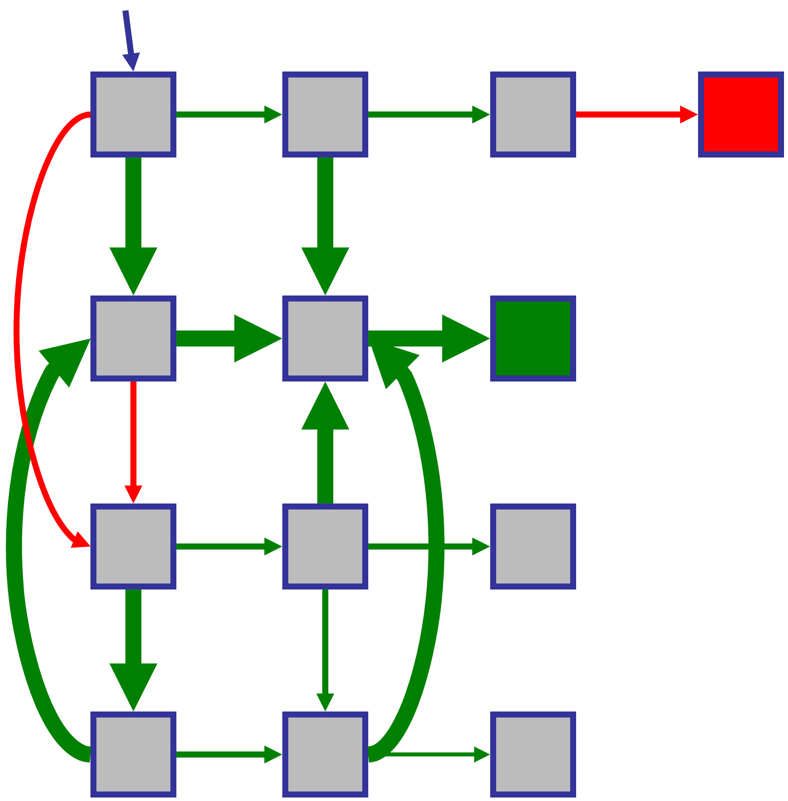


Uncontrollable

Controllable

Untimed Games

Reachability / Safety Games



Strategy:

F : Run(A) → E_c

Memoryless strategy:

F : Q → E_c

Winning Run:

$States(\rho) \cap G \neq \emptyset$

$States(\rho) \cap B = \emptyset$

Winning Strategy:

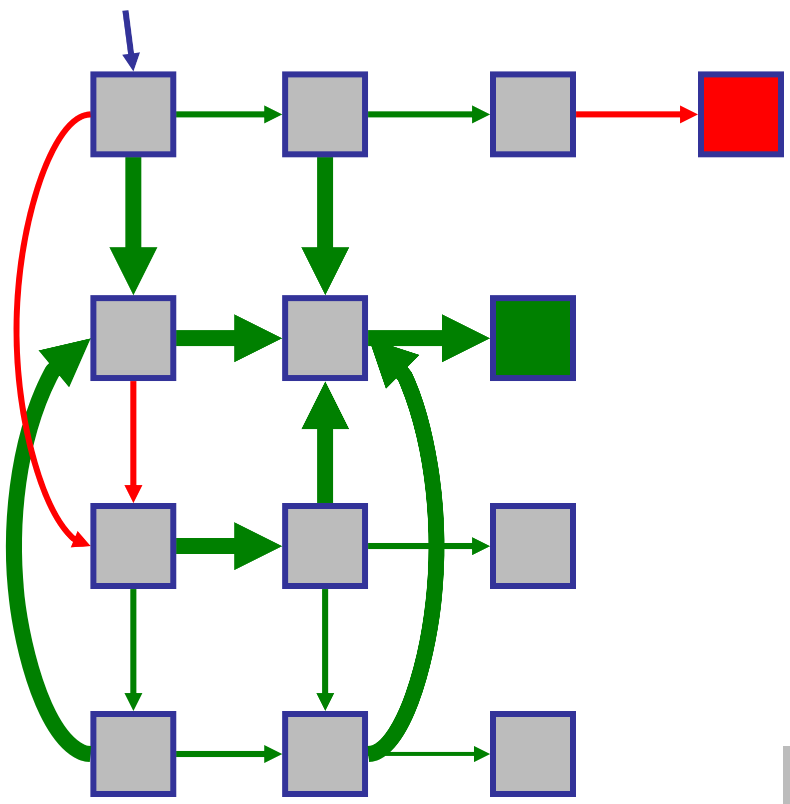
$Runs(F) \subseteq WinRuns$

Loosing (memoryless) strategy

Uncontrollable

Controllable

Untimed Games



Reachability / Safety Games

Strategy:

$F : Run(A) \rightarrow E_c$

Memoryless strategy:

$F : Q \rightarrow E_c$

Winning Run:

$States(\rho) \cap G \neq \emptyset$

$States(\rho) \cap B = \emptyset$

Winning Strategy:

$Runs(F) \subseteq WinRuns$

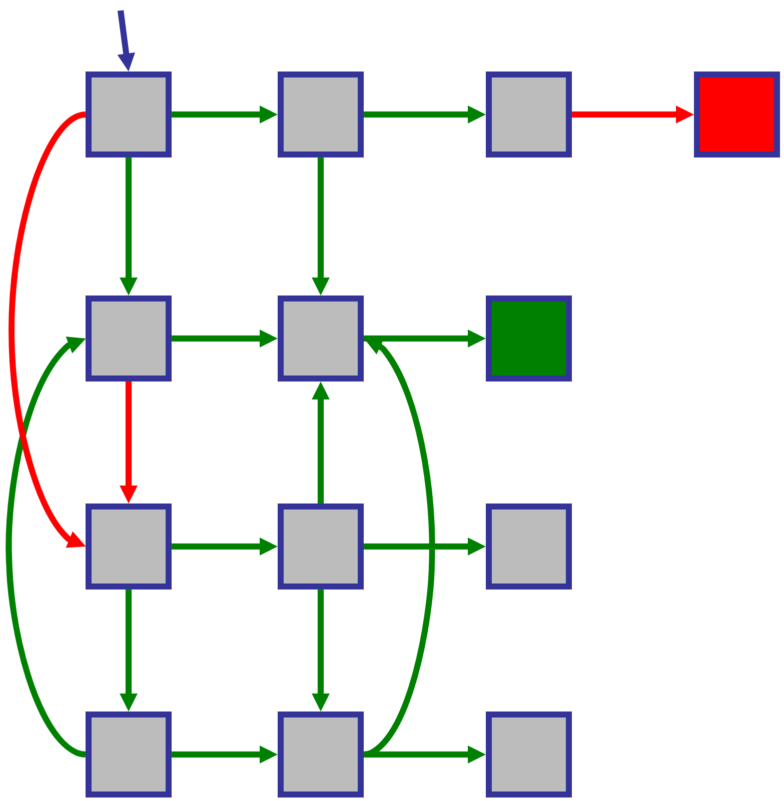
Winning (memoryless) strategy)

Uncontrollable

Controllable

Untimed Games

Backwards Fixed-Point Computation



$cPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_c q' \}$
$uPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_u q' \}$

$\pi(X) = cPred(X) \setminus uPred(X^c)\ ]$

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

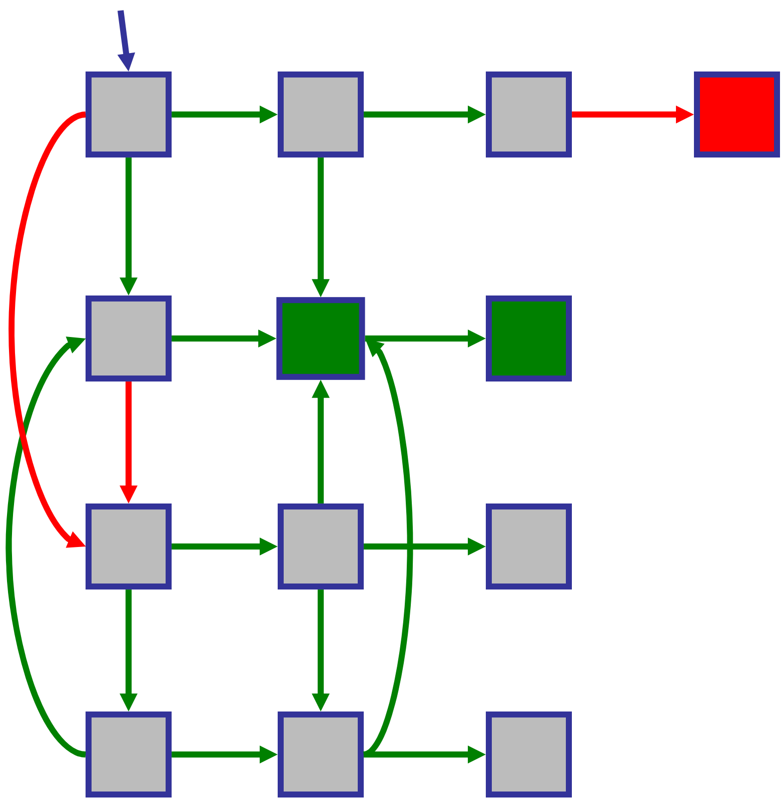
$X \mapsto \pi(X) \cup Goal$

Uncontrollable

Controllable

Untimed Games

Backwards Fixed-Point Computation

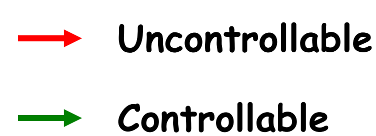


$cPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_c q' \}$
$uPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_u q' \}$

$$\pi(X) = cPred(X) \setminus uPred(X^C)]$$

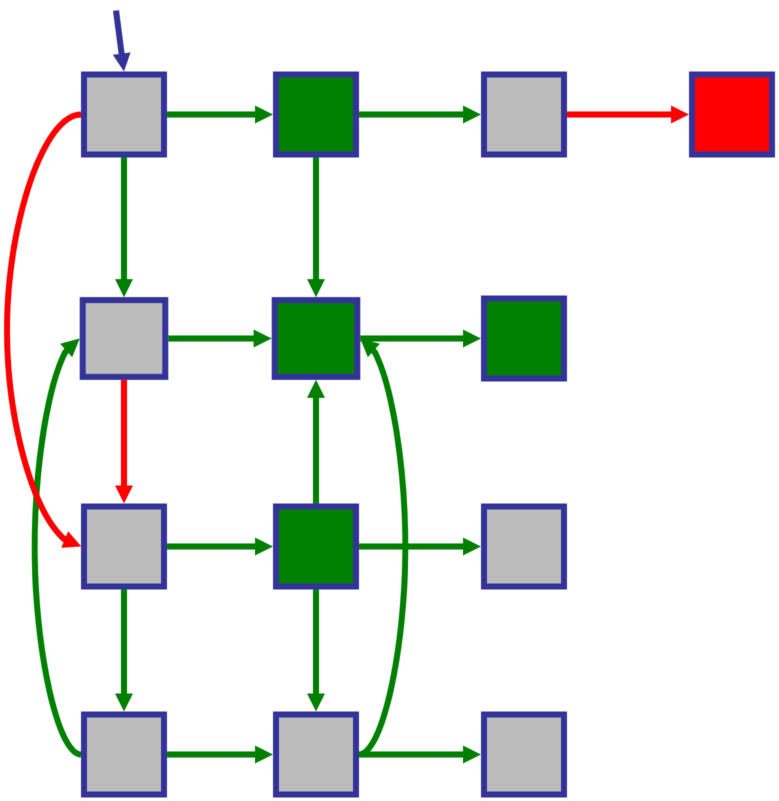
Theorem:
The set of winning states is obtained as the least fixpoint of the function:

$$X \mapsto \pi(X) \cup Goal$$



Untimed Games

Backwards Fixed-Point Computation



$cPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_c q' \}$
$uPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_u q' \}$

$$\pi(X) = cPred(X) \setminus uPred(X^C)\ ]$$

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

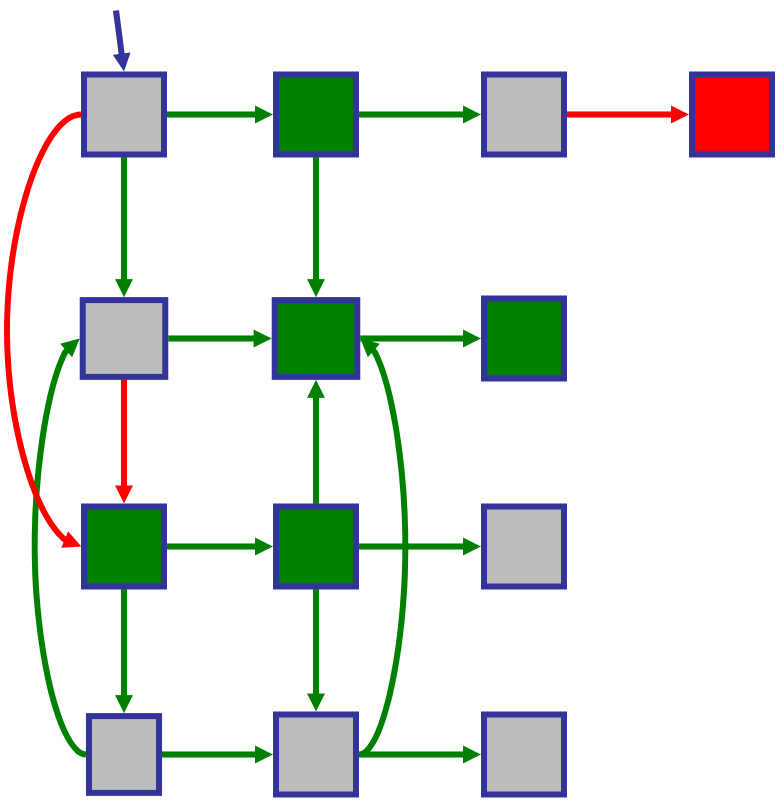
$$X \mapsto \pi(X) \cup Goal$$

Uncontrollable

Controllable

Untimed Games

Backwards Fixed-Point Computation



$cPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_c q' \}$

$uPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_u q' \}$

$\pi(X) = cPred(X) \setminus uPred(X^C)\ ]$

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

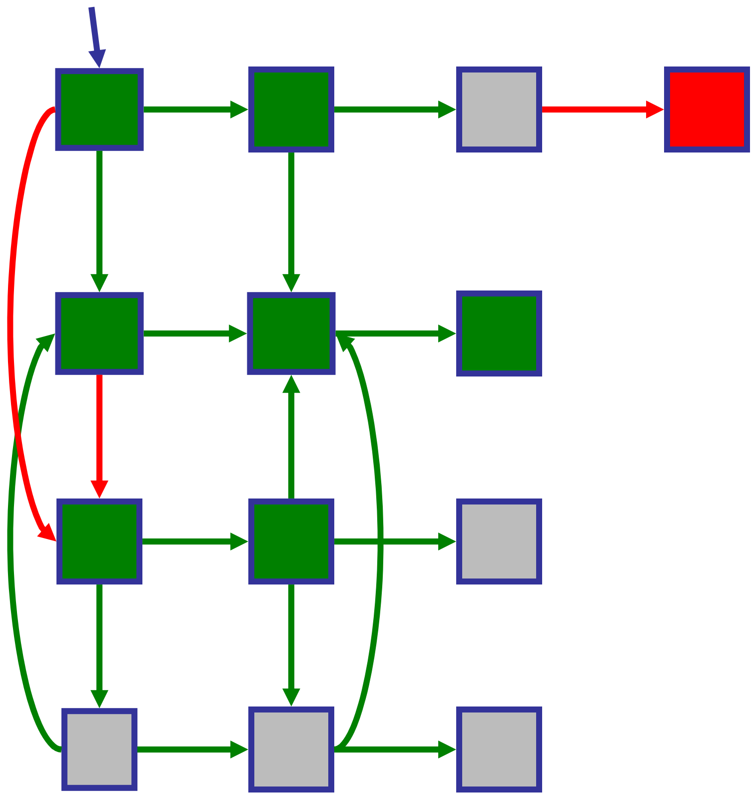
$$X \mapsto \pi(X) \cup Goal$$

Uncontrollable (red arrow)

Controllable (green arrow)

Untimed Games

Backwards Fixed-Point Computation



cPred(X) = { q∈Q | ∃ q'∈ X. q $\rightarrow_c$ q'}
uPred(X) = { q∈Q | ∃ q'∈ X. q $\rightarrow_u$ q'}

$\pi(X)$ = cPred(X) \ uPred(X^c)]

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

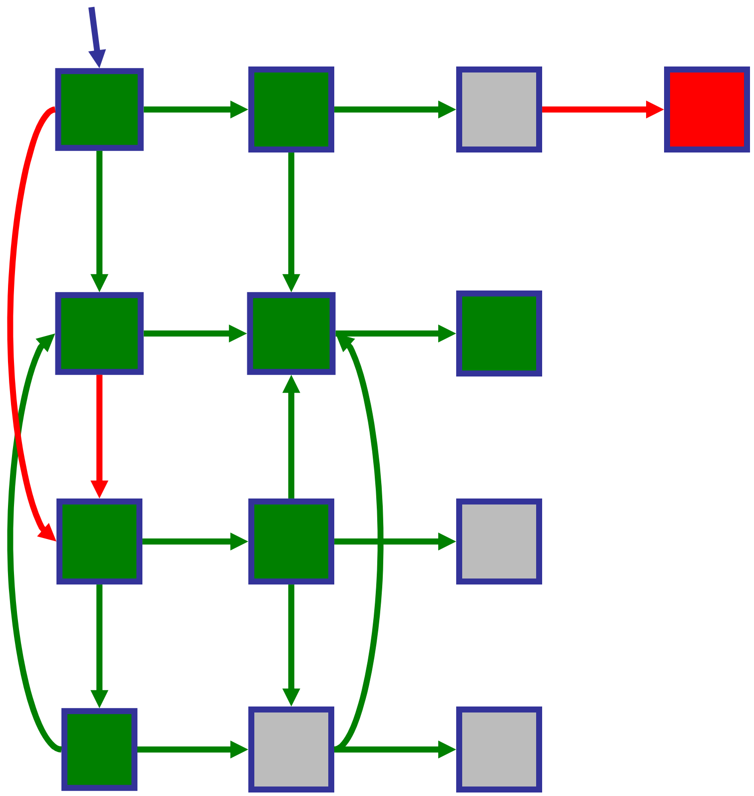
X ↦ π(X) ∪ Goal

→ Uncontrollable

→ Controllable

Untimed Games

Backwards Fixed-Point Computation



$cPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_c q' \}$
$uPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_u q' \}$

$$\pi(X) = cPred(X) \setminus uPred(X^C) \,]$$

Theorem:
The set of winning states is obtained as the least fixpoint of the function:

$$X \mapsto \pi(X) \cup Goal$$

Uncontrollable

Controllable

Timed Games

Reachability / Safety Games

Strategy:

$F : Run(A) \rightarrow E_c \cup \lambda$

Memoryless strategy:

$F : Q \rightarrow E_c \cup \lambda$

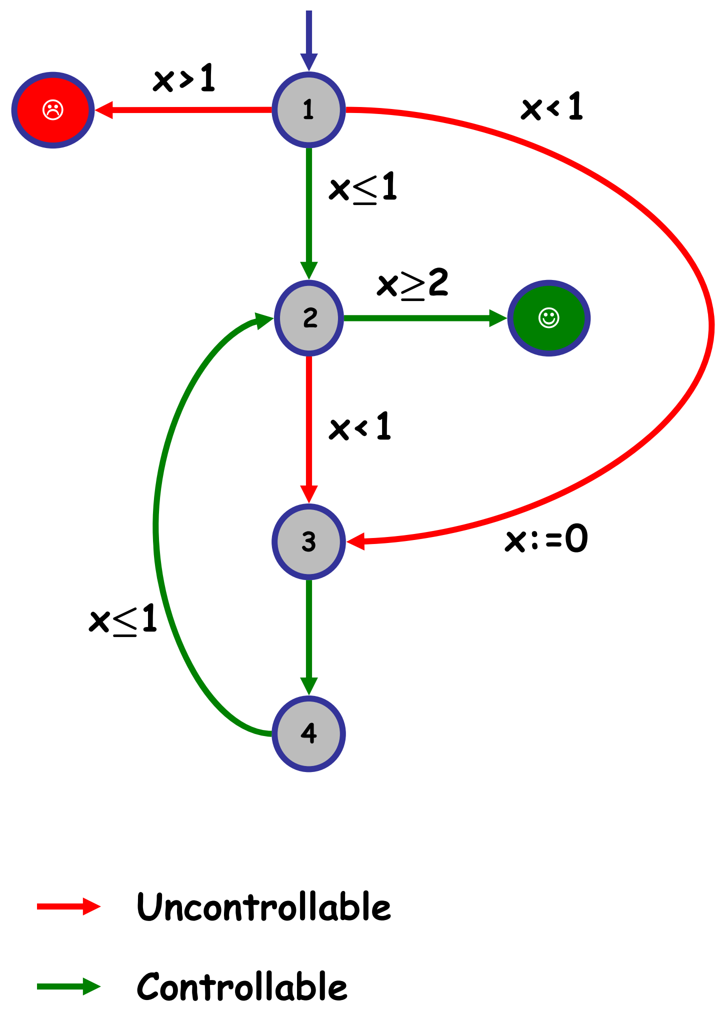
Winning Run:

$States(\rho) \cap G \neq \emptyset$

$States(\rho) \cap G = \emptyset$

Winning Strategy:

$Runs(F) \subseteq WinRuns$



Timed Games

Reachability / Safety Games

Strategy:

$F : Run(A) \rightarrow E_c \cup \lambda$

Memoryless strategy:

$F : Q \rightarrow E_c \cup \lambda$

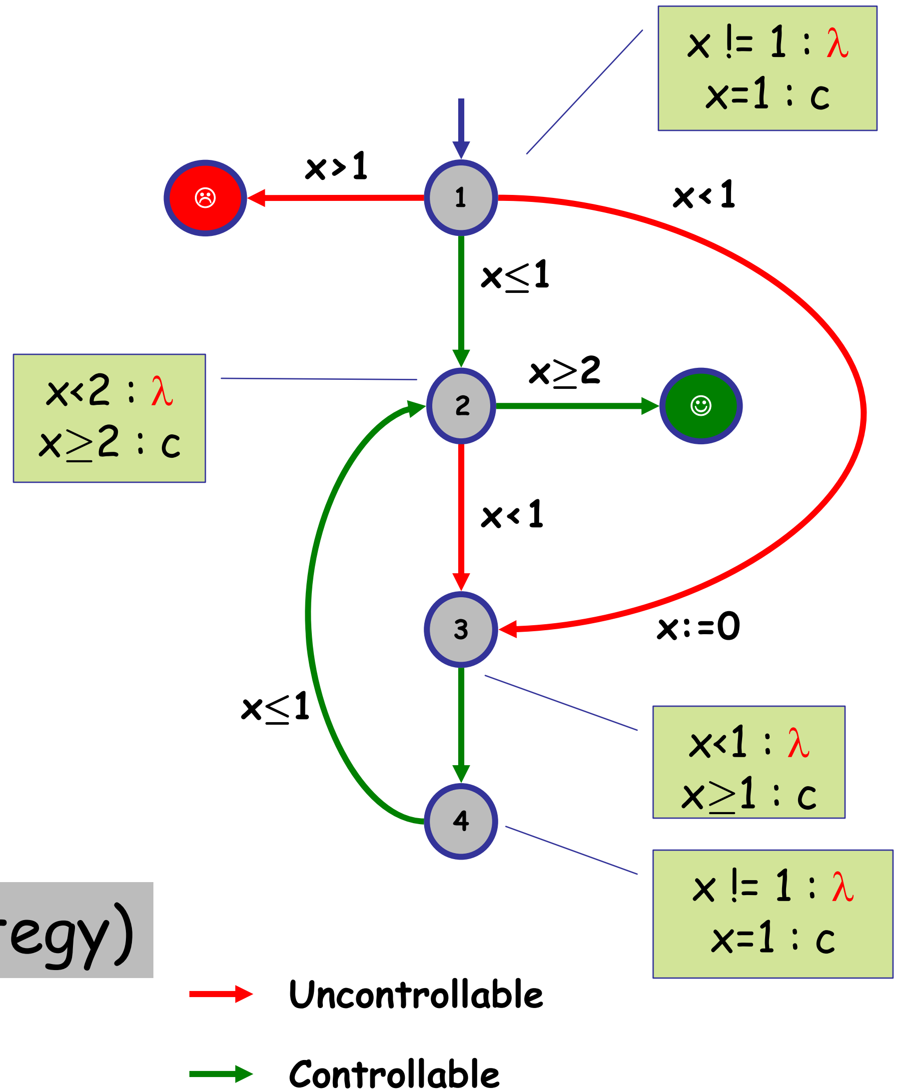
Winning Run:

$States(\rho) \cap G \neq \emptyset$

$States(\rho) \cap G = \emptyset$

Winning Strategy:

$Runs(F) \subseteq WinRuns$



Winning (memoryless) strategy)

Timed Games - State-of-the-Art

- Timed Automata + Reachability [AD94]
- Time Game Automata: Control [MPS95, AMPS98]
- Time Optimal Control (reachability) [AM99]
- "False" On-the-fly Algorithm [AT01]
- Priced Timed Automata (reachability) [LBB+01, ALTP01, LRS04, RL05]
- Price Timed Automata (safety) [BBL04]
- Price Optimal Control (reachability):
  - Acyclic PTA [LTMM02]
  - Bounded length [ABM04]
  - Strong non-zeno cost-behaviour [BCFL04]
- More to come !!

UPPAAL

To be improved !!

UPPAAL Cora

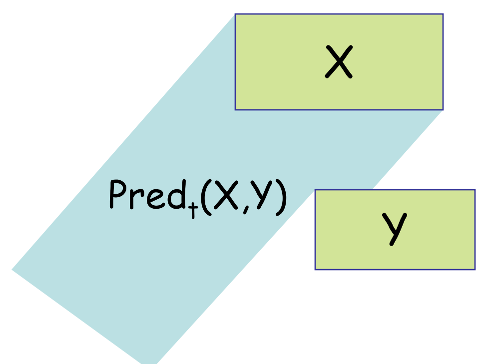
Timed Games - State-of-the-Art

Backwards Fixed-Point Computation

Definitions

$cPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_c q' \}$
$uPred(X) = \{ q \in Q \mid \exists q' \in X.\ q \rightarrow_u q' \}$
$Pred_t(X,Y) = \{ q \in Q \mid \exists t.\ q^t \in X \text{ and } \forall s \leq t.\ q^s \in Y^C \}$

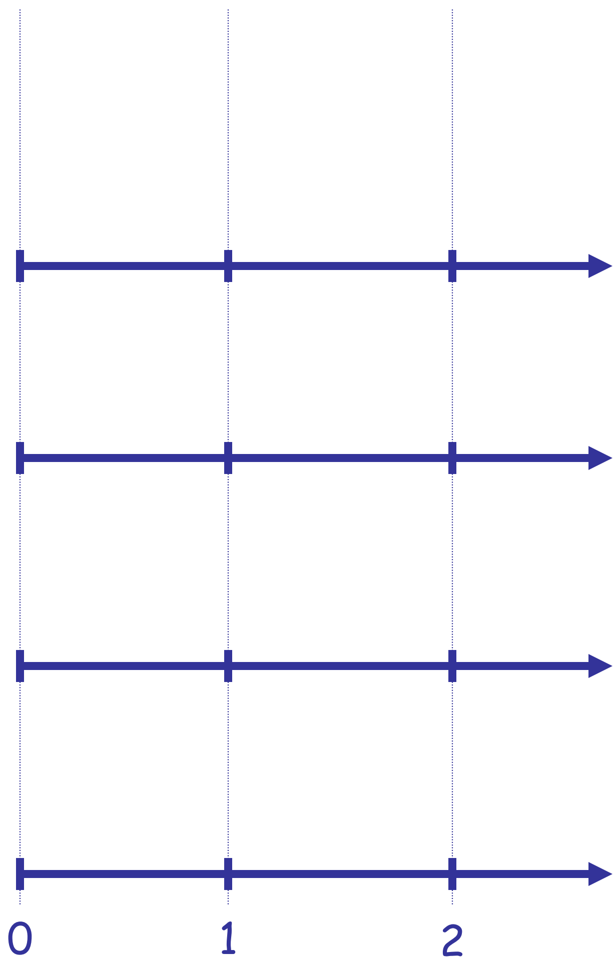
$\pi(X) = Pred_t[\, X \cup cPred(X)\, ,\, uPred(X^C)\,]$

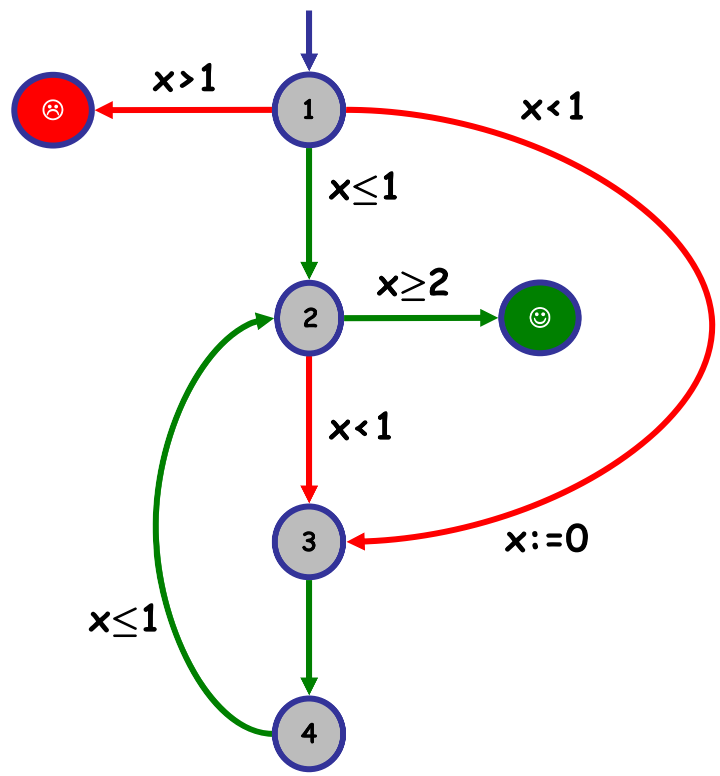


Theorem:
The set of winning states is obtained as the least fixpoint of the function: $X \mapsto \pi(X) \cup Goal$

Timed Games - State-of-the-Art

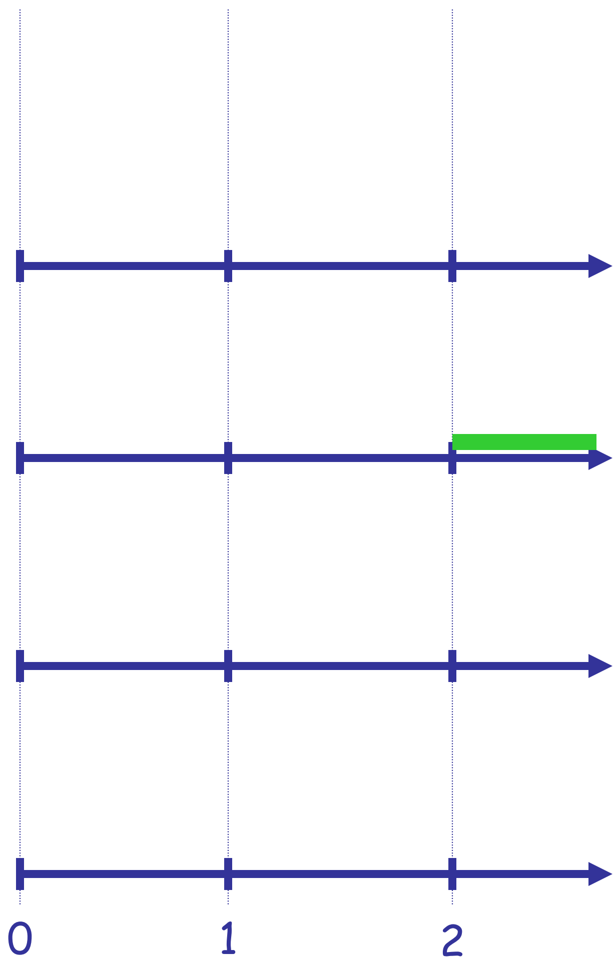
Backwards Fixed-Point Computation

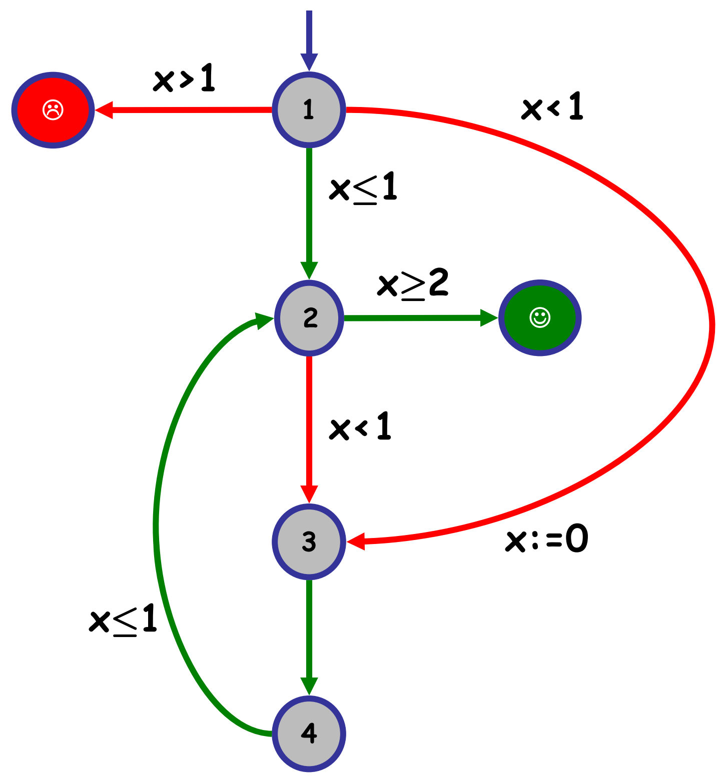




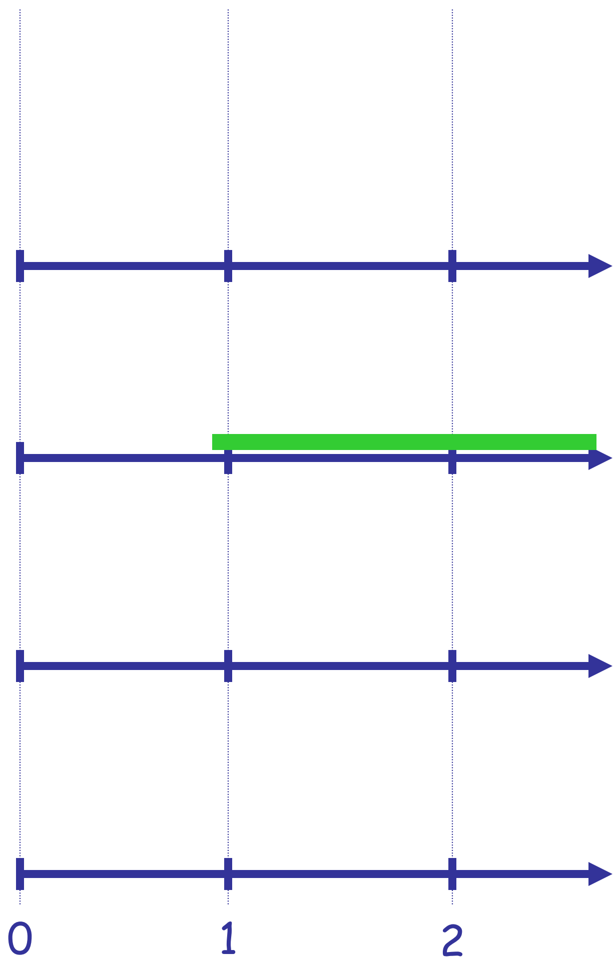
Timed Games - State-of-the-Art

Backwards Fixed-Point Computation

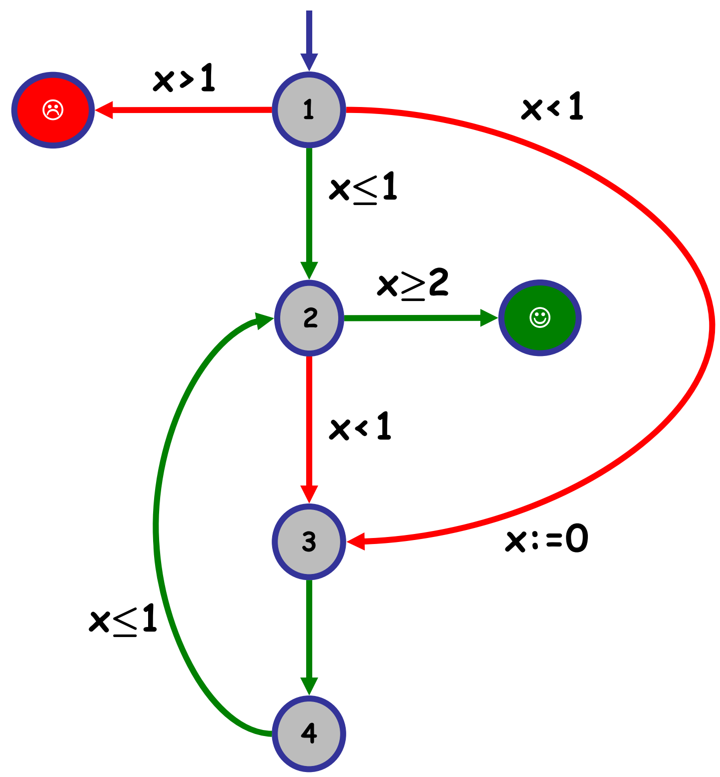




Timed Games - State-of-the-Art

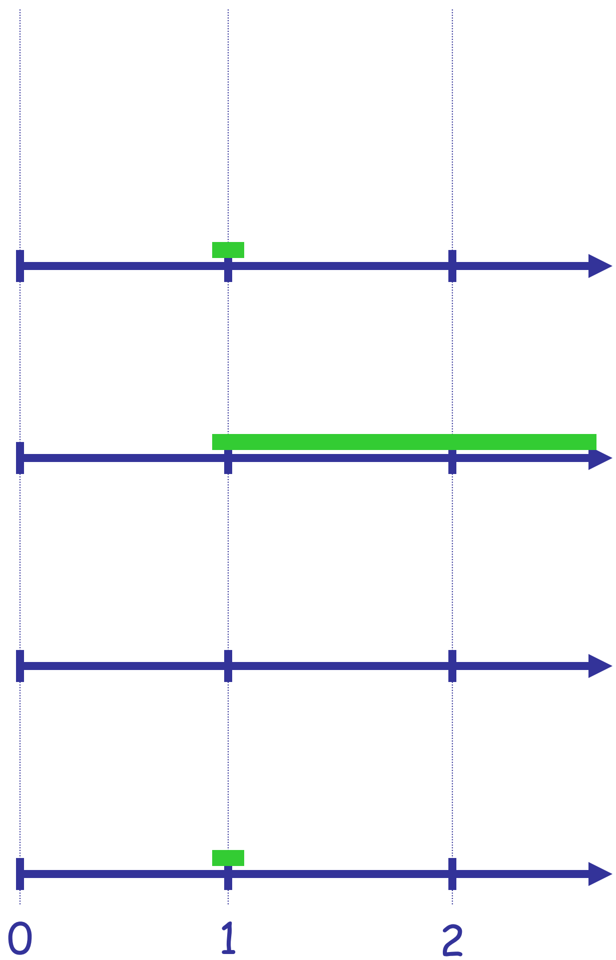


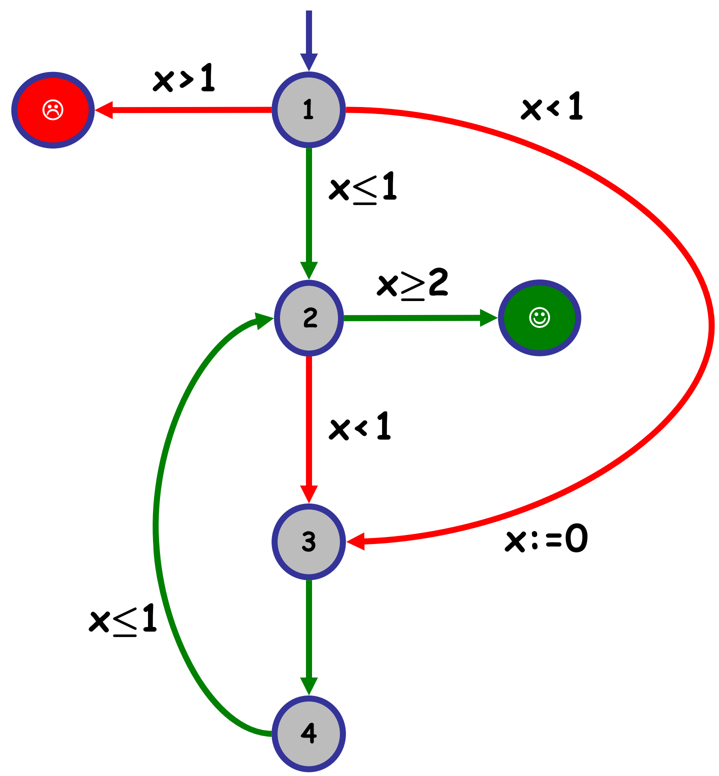
Backwards Fixed-Point Computation



Timed Games - State-of-the-Art

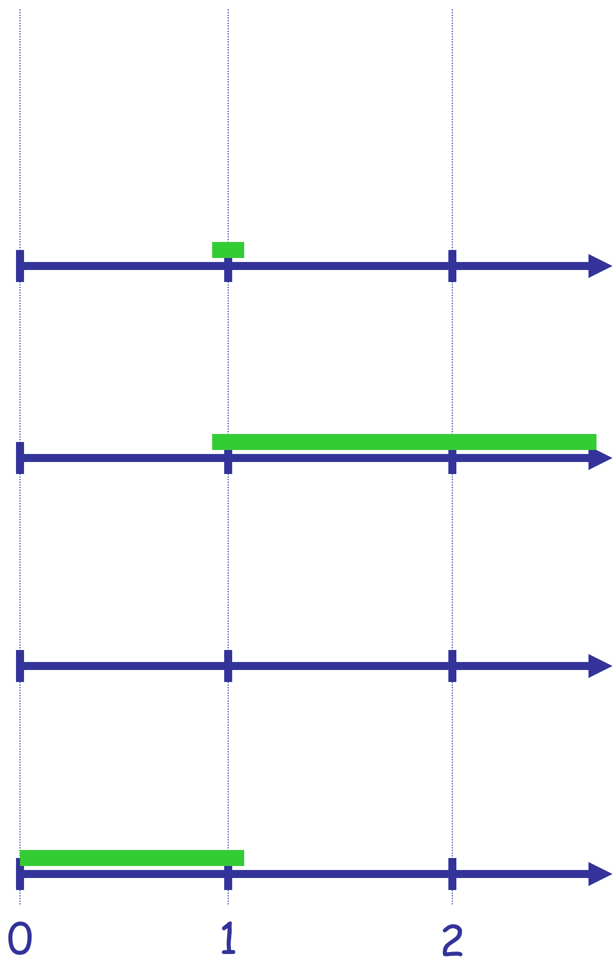
Backwards Fixed-Point Computation

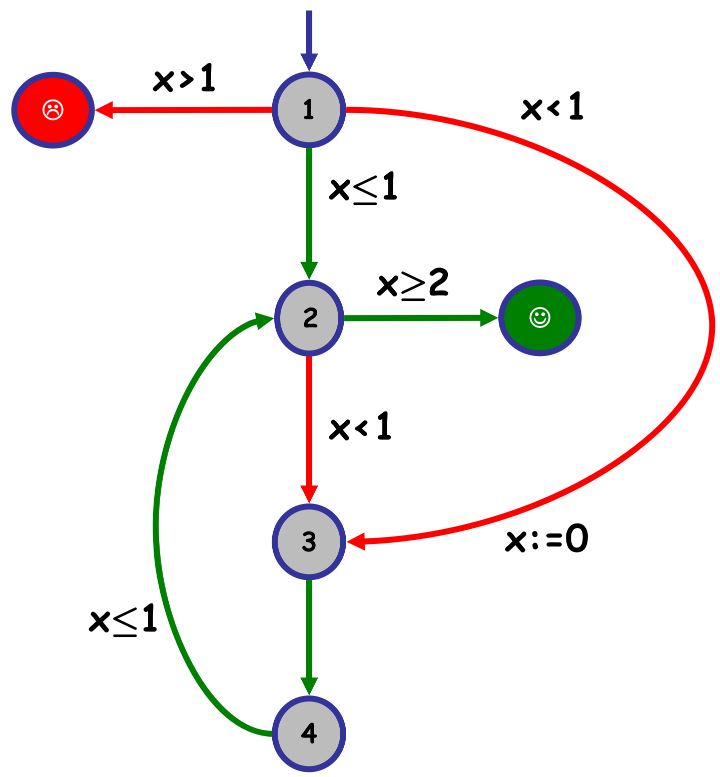




Timed Games - State-of-the-Art

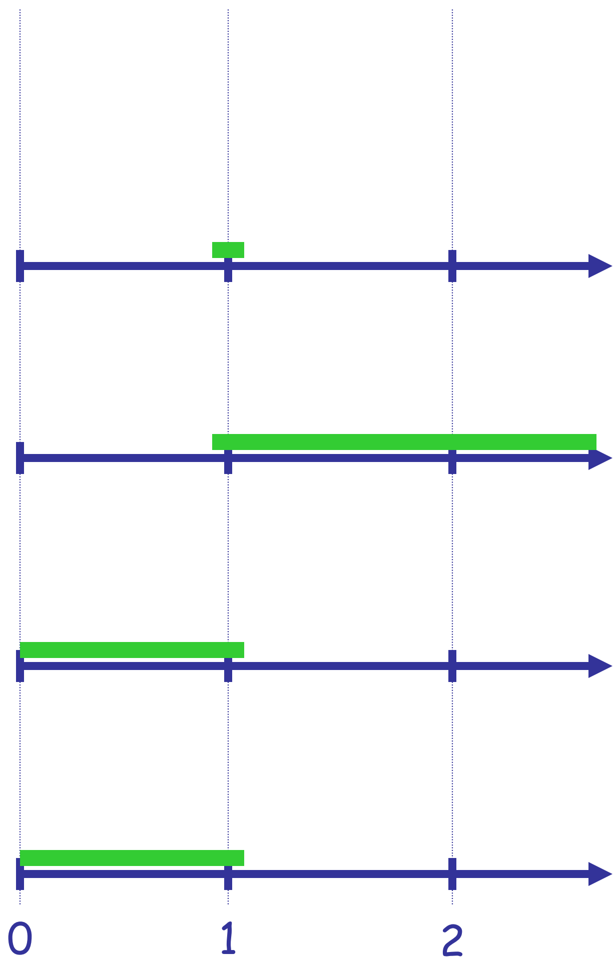
Backwards Fixed-Point Computation

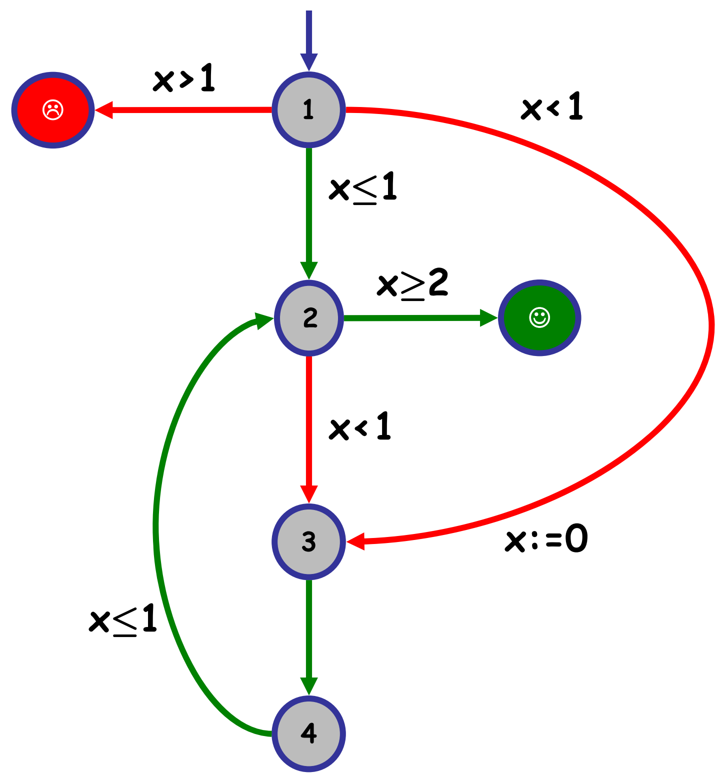




Timed Games - State-of-the-Art

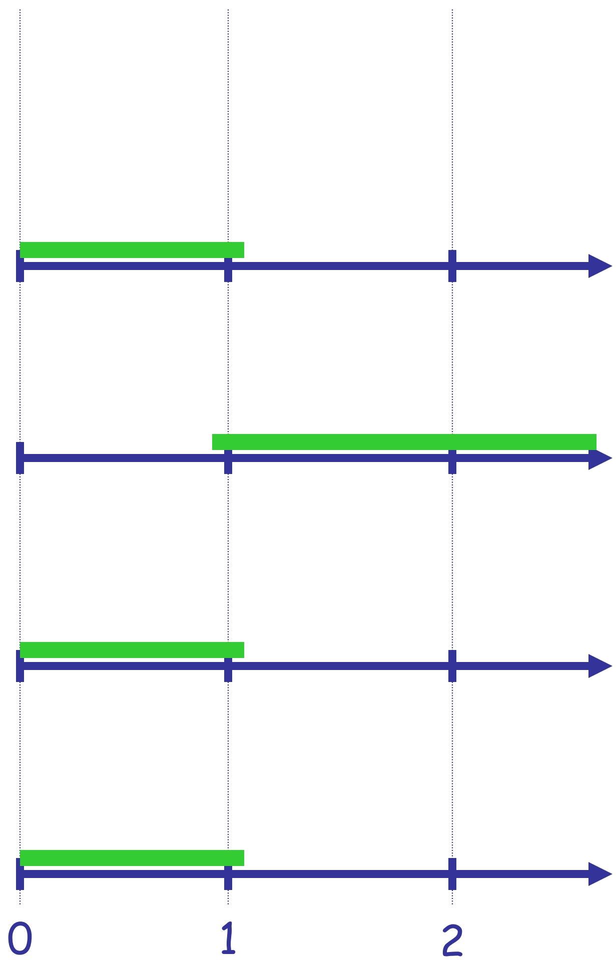
Backwards Fixed-Point Computation

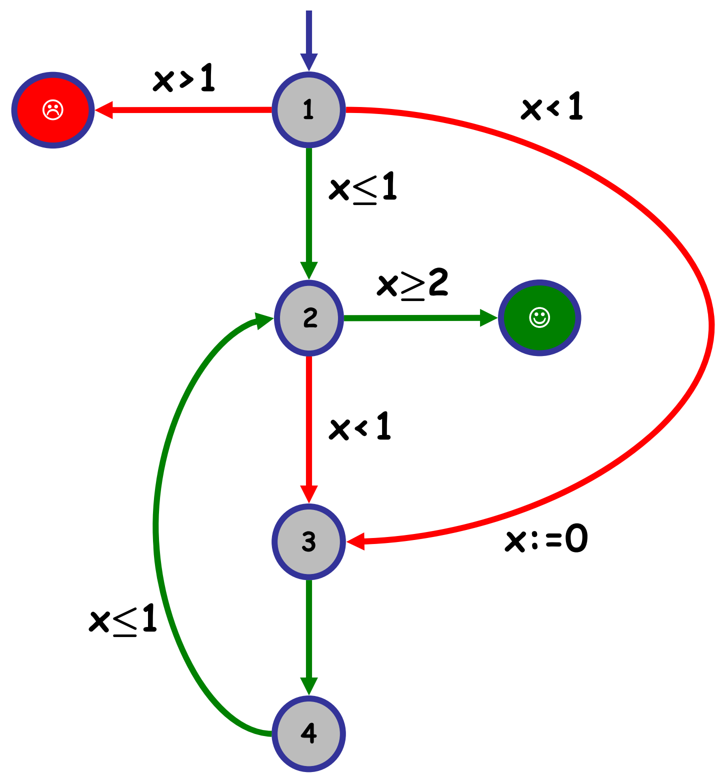




Timed Games - State-of-the-Art

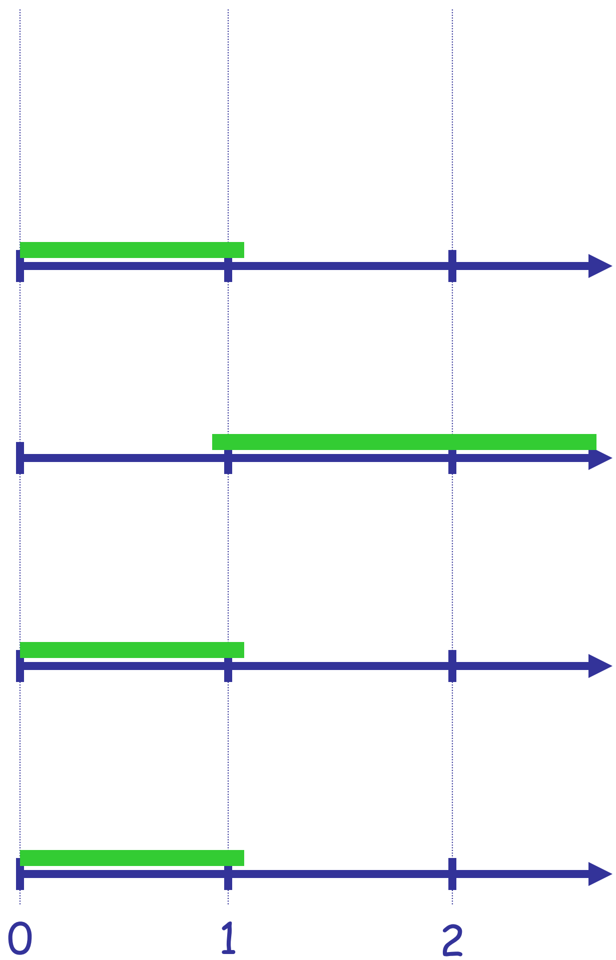
Backwards Fixed-Point Computation

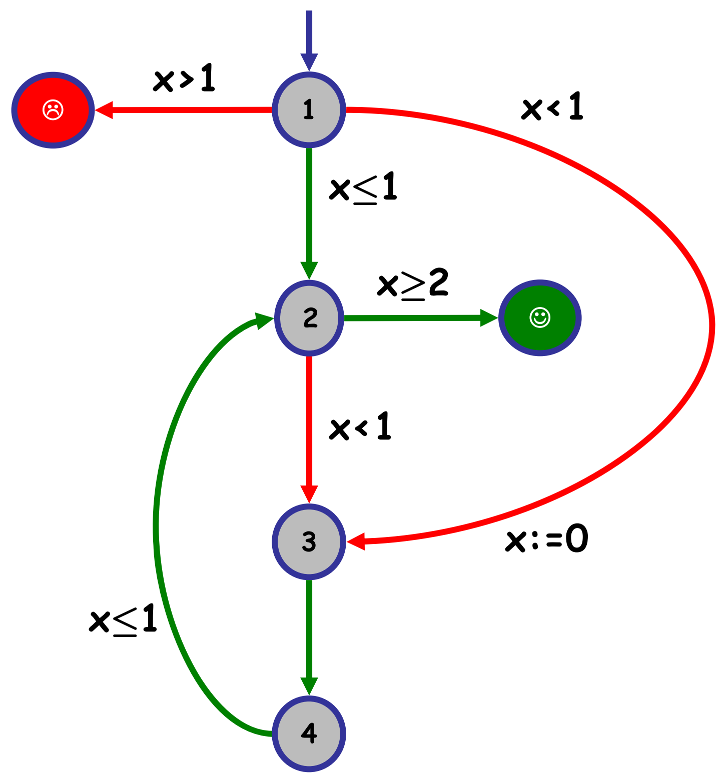




Timed Games - State-of-the-Art

Backwards Fixed-Point Computation



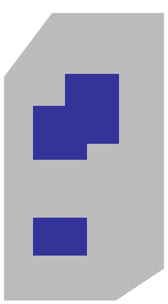


We want **Forward** and **On-The-Fly** Algorithm
in order to avoid constructing all (backwards) reachable
state-space and to allow for discrete variables (e.g. in UPPAAL)

On-the-fly Algorithms for Timed Games

- $S, S', \ldots$
  are *symbolic* states, i.e. sets of concrete s
- G
  is the set of (concrete) goal states;
- $E = \{S \xrightarrow{c} S', S \xrightarrow{u} S'\}$
  the (finite) set of symbolic transitions (c
- $Waiting \subseteq E$
  is the list of symbolic transitions waiting
- $Passed$
  is the list of the passed symbolic states;
- $Win[S] \subseteq S$
  is the subset of S currently known to be
- $Depend[S] \subseteq E$
  indicates the edges (predecessors) of S wh
  information about S is obtained.

S



Win(S)

Initialization:

$Passed \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, \vec{0})\}^{\nearrow}$;
$Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = \mathsf{Post}_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

Main:

while $((Waiting \neq \emptyset) \wedge (s_0 \notin Win[S_0]))$ **do**
  $e = (S, \alpha, S') \leftarrow pop(Waiting)$;
  if $S' \notin Passed$ **then**
    $Passed \leftarrow Passed \cup \{S'\}$;
    $Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
    $Win[S'] \leftarrow S' \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    $Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') \mid S'' = \mathsf{Post}_\alpha(S')^{\nearrow}\}$;
    if $Win[S'] \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
  else (* reevaluate *)[a]
    $Win^* \leftarrow \mathsf{Pred}_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} \mathsf{Pred}_c(Win[T]),$
      $\bigcup_{S \xrightarrow{u} T} \mathsf{Pred}_u(T \setminus Win[T])) \cap S$;
    if $(Win[S] \subsetneq Win^*)$ **then**
      $Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
    $Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
  endif
endwhile

On-the-fly Algorithms for Timed Games

– $S, S', \ldots$
are *symbolic* states, i.e. sets of concrete s
– G
is the set of (concrete) goal states;
– $E = \{S \xrightarrow{c} S', S \xrightarrow{u} S'\}$
the (finite) set of symbolic transitions (c
– $Waiting \subseteq E$
is the list of symbolic transitions waiting
– $Passed$
is the list of the passed symbolic states;
– $Win[S] \subseteq S$
is the subset of S currently
– $Depend[S]$
indicates t
informatio

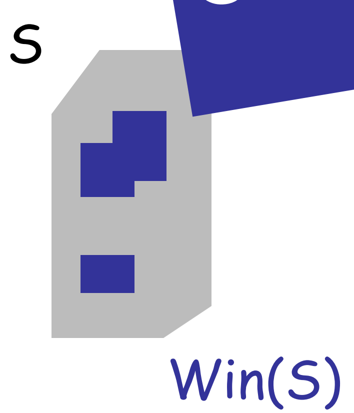
Initialization:

$Passed \leftarrow \{S_0\}$ **where** $S_0 = \{(\ell_0, \vec{0})\}^{\nearrow}$;
$Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = \mathsf{Post}_\alpha(S_0)^{\nearrow}\}$;
$Win[S_0] \leftarrow S_0 \cap (\{\mathsf{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

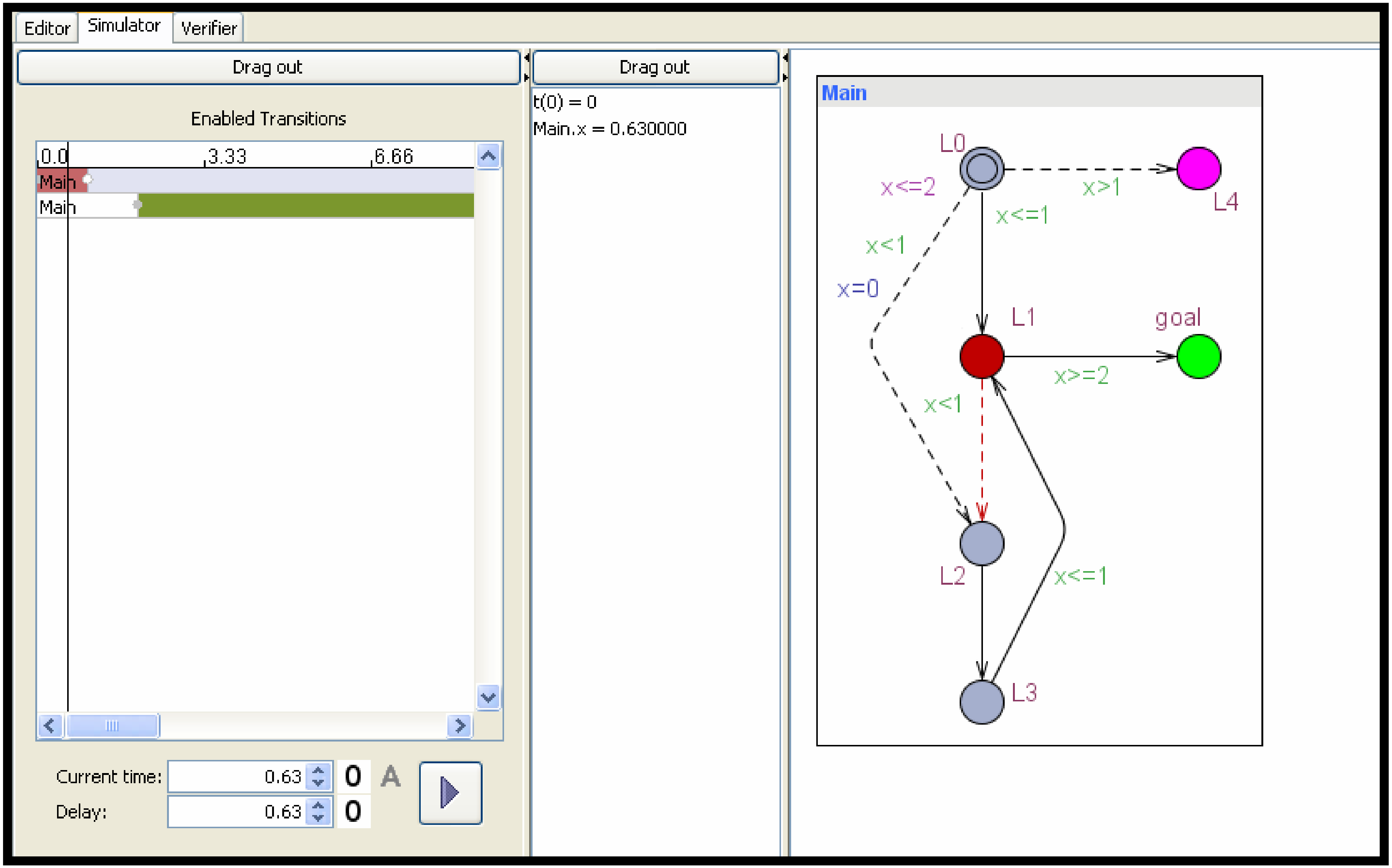
Main:

$\ldots, \alpha, S'') \mid S'' = \mathsf{Post}_\alpha(S')^{\nearrow}\}$;
$\ldots \neq \emptyset$ **then** $Waiting \leftarrow Waiting \cup \{e\}$;
else (* **reevaluate** *)[a]
$Win^* \leftarrow \mathsf{Pred}_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} \mathsf{Pred}_c(Win[T]), \bigcup_{S \xrightarrow{u} T} \mathsf{Pred}_u(T \setminus Win[T])) \cap S$;
if $(Win[S] \subsetneq Win^*)$ **then**
$Waiting \leftarrow Waiting \cup Depend[S]$; $Win[S] \leftarrow Win^*$;
$Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
endif
endwhile

UPPAAL Tiga
=
On-the-fly algorithm for timed games
[CONCUR'05]



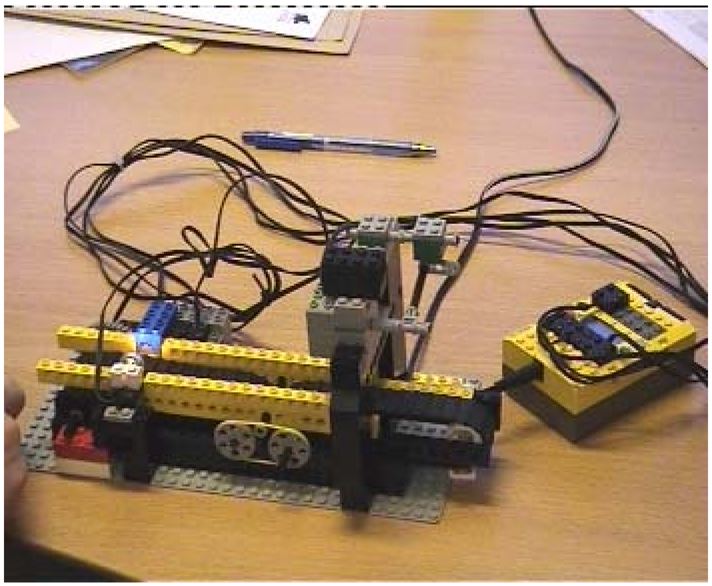
UPPAAL Tiga : New Concrete Time Simulator

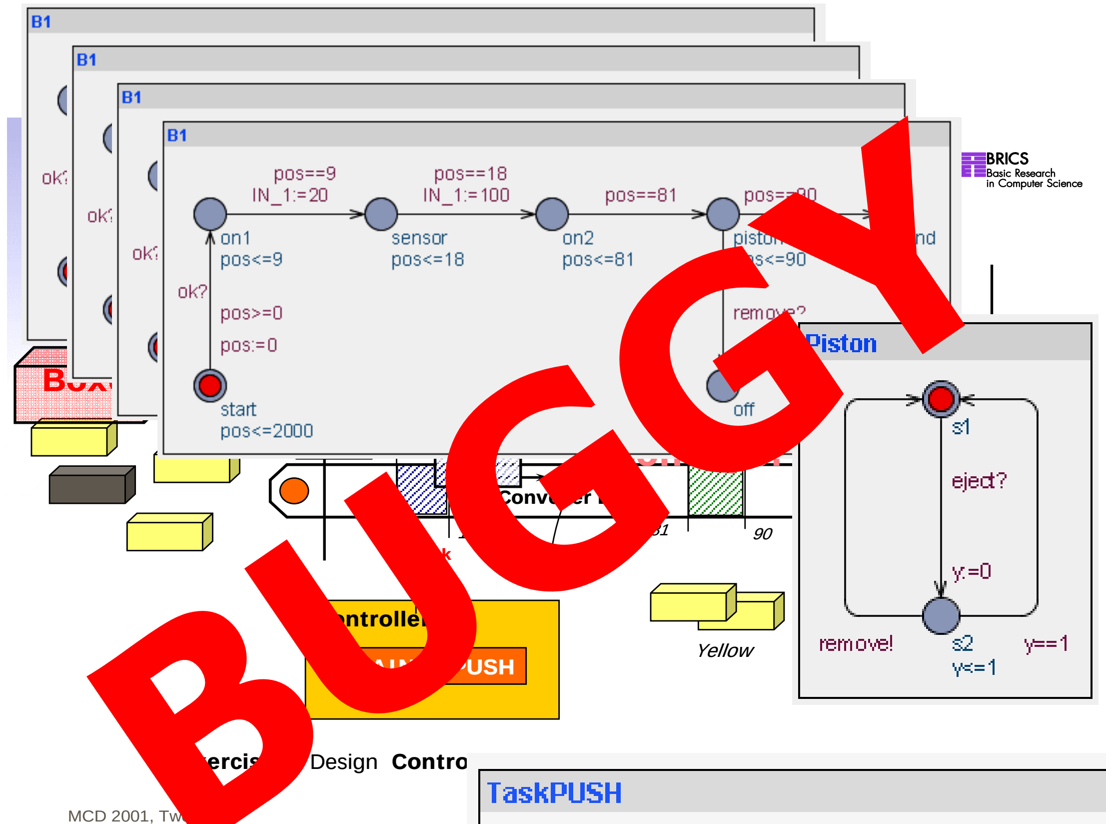


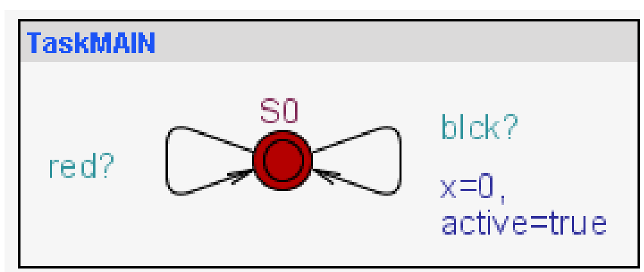
UPPAAL Tiga : CTL Control Objectives

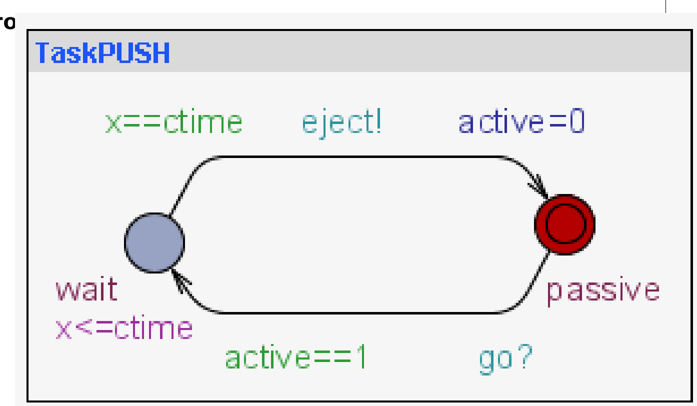
- Reachability properties:
  - control: A[p U q] *until*
  - control: A<> q ⇔ control: A[true U q]
- Safety properties:
  - control: A[p W q] *weak until*
  - control: A[] p ⇔ control: A[p W false]
- Time-optimality :
  - control_t*(u,g): A[p U q]
    - u is an upper-bound to prune the search, act like an invariant but on the path = expression on the current state.
    - g is the time to the goal from the current state (a lower-bound in fact), also used to prune the search. States with are pruned

A Buggy Brick Sorting Program

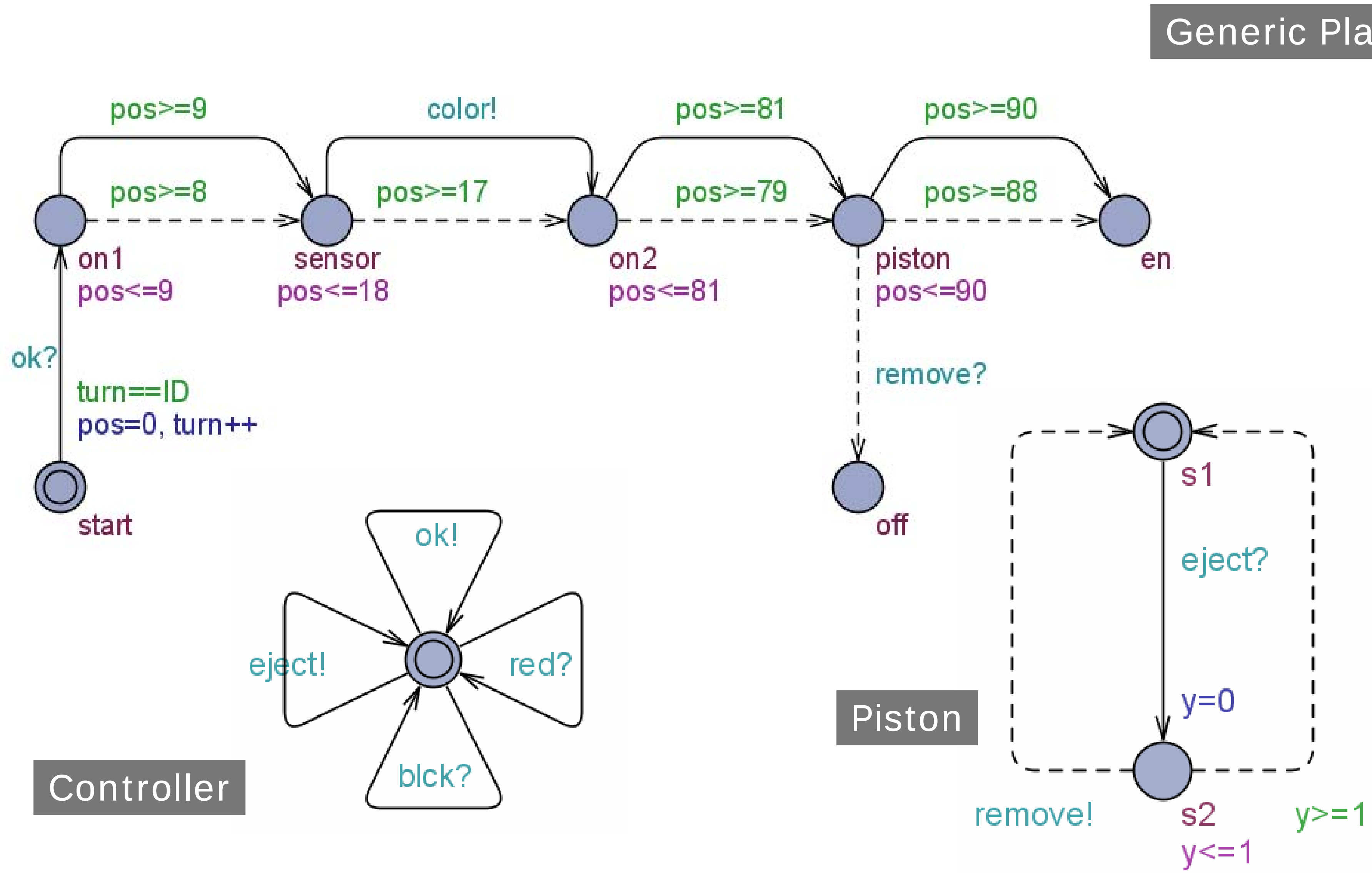




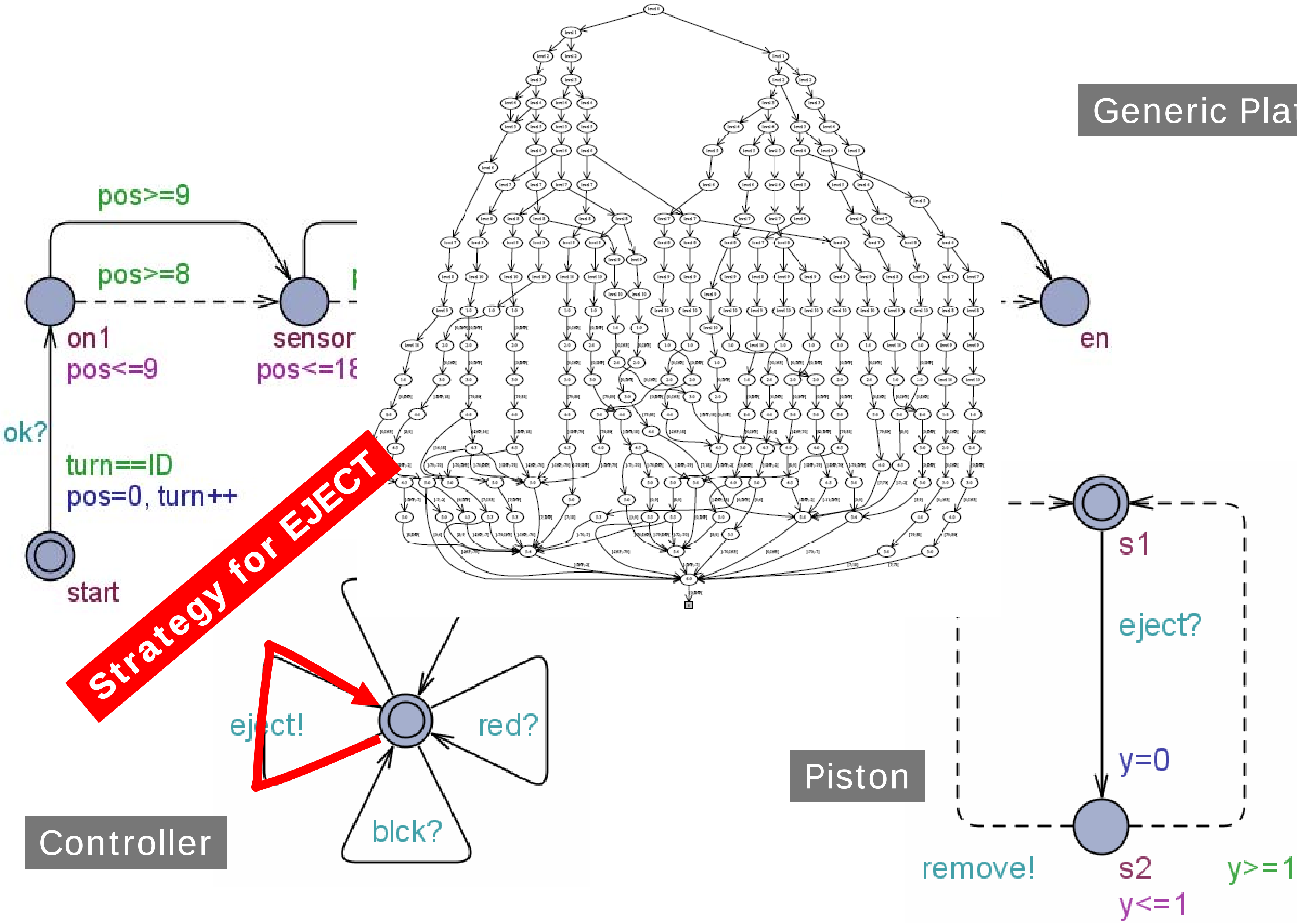




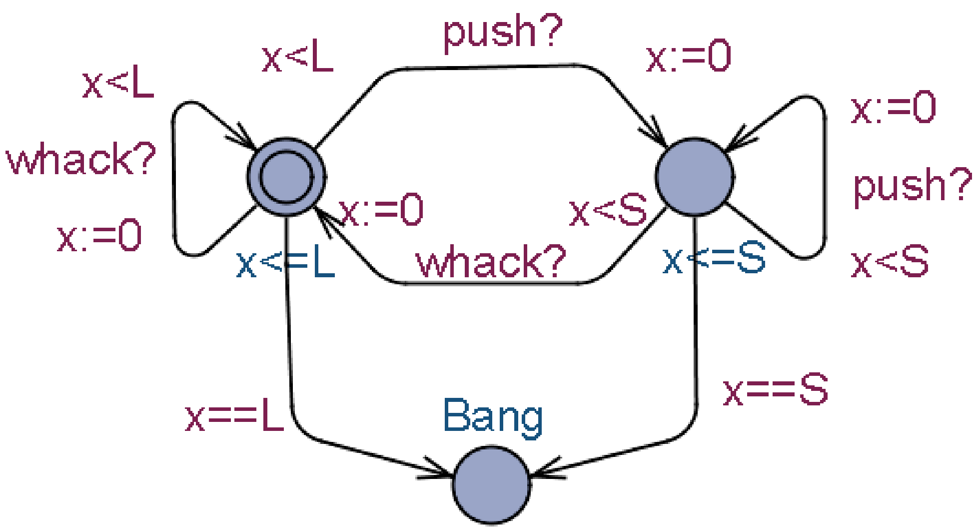
Brick Sorting



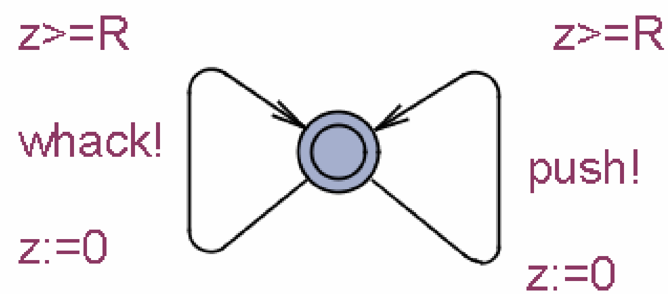
Brick Sorting



Balancing Plates / Timed Automata



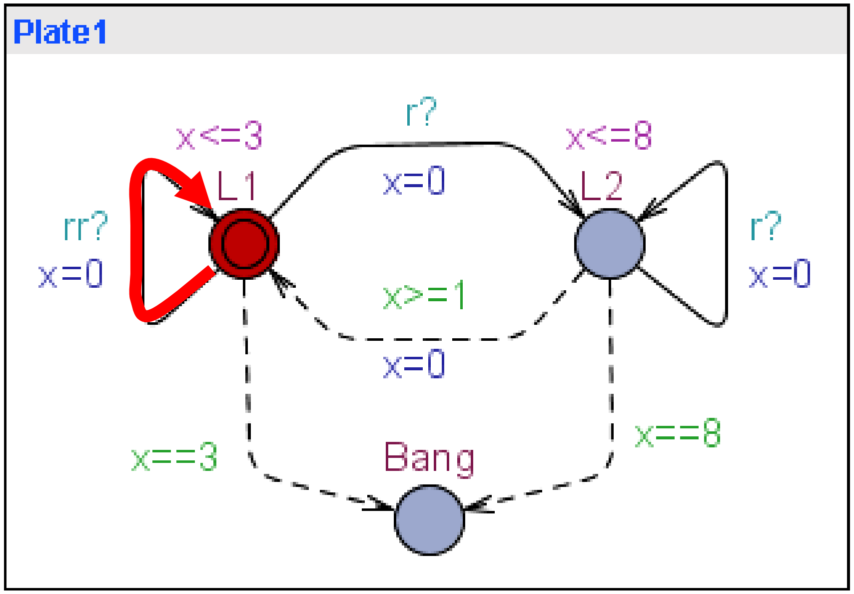
A Plate

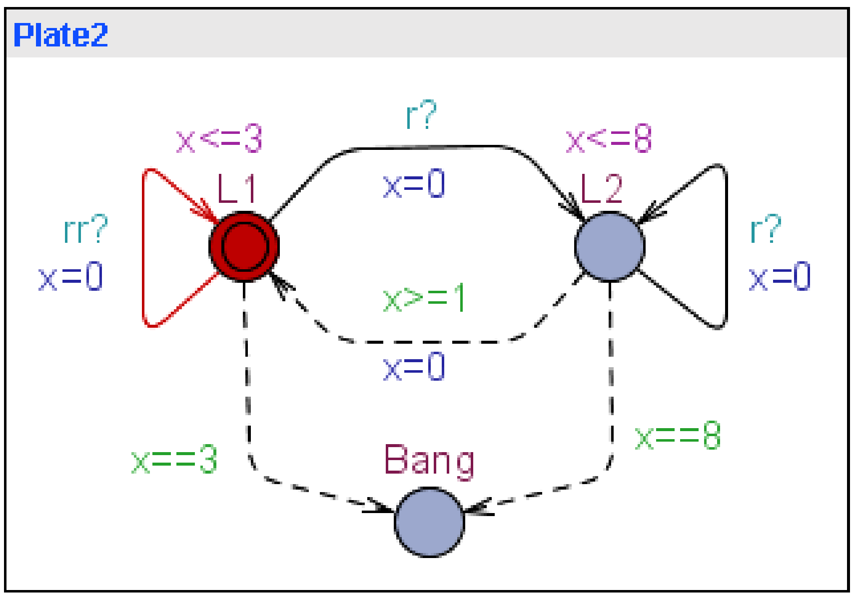


The Joggler

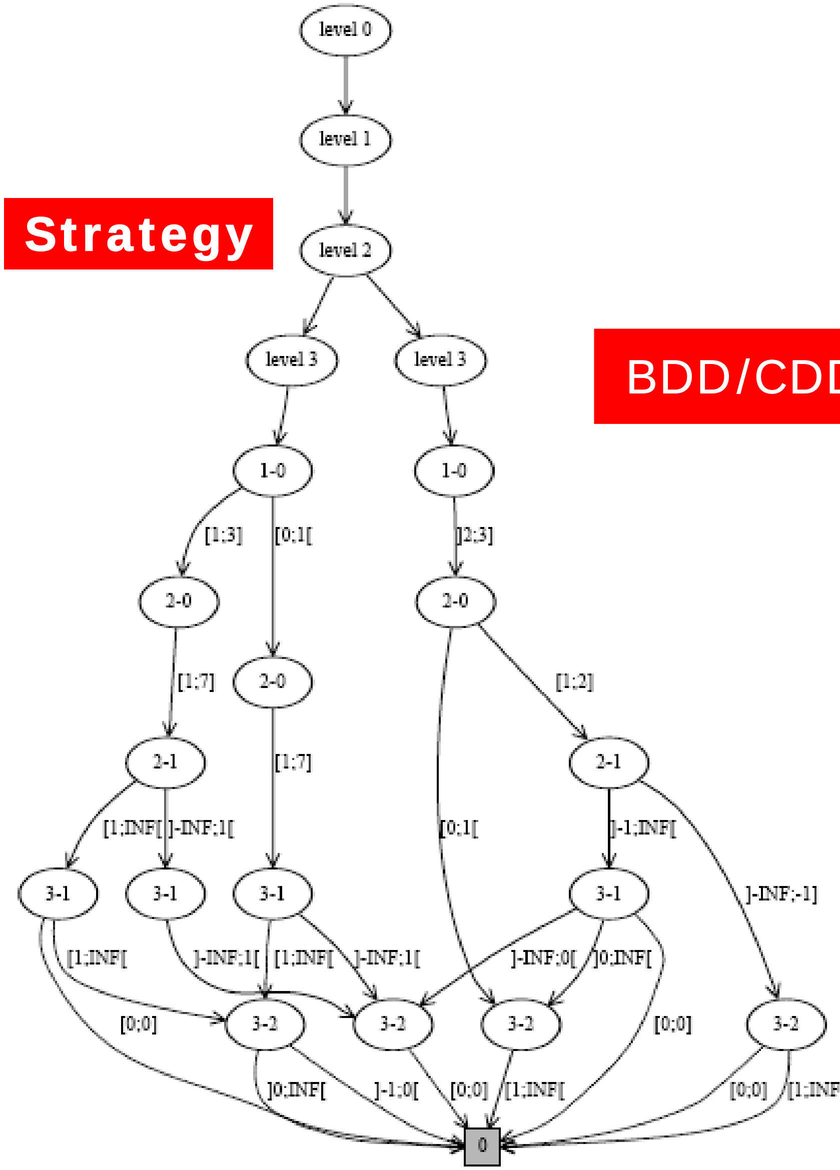
E□ ¬(Plate1.Bang or Plate2.Bang or ...)

Balancing Plates / Time Uncertainty



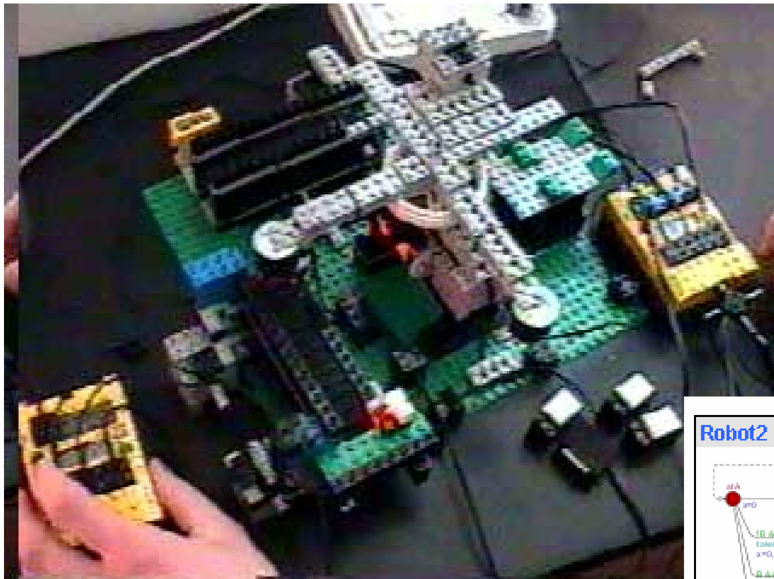


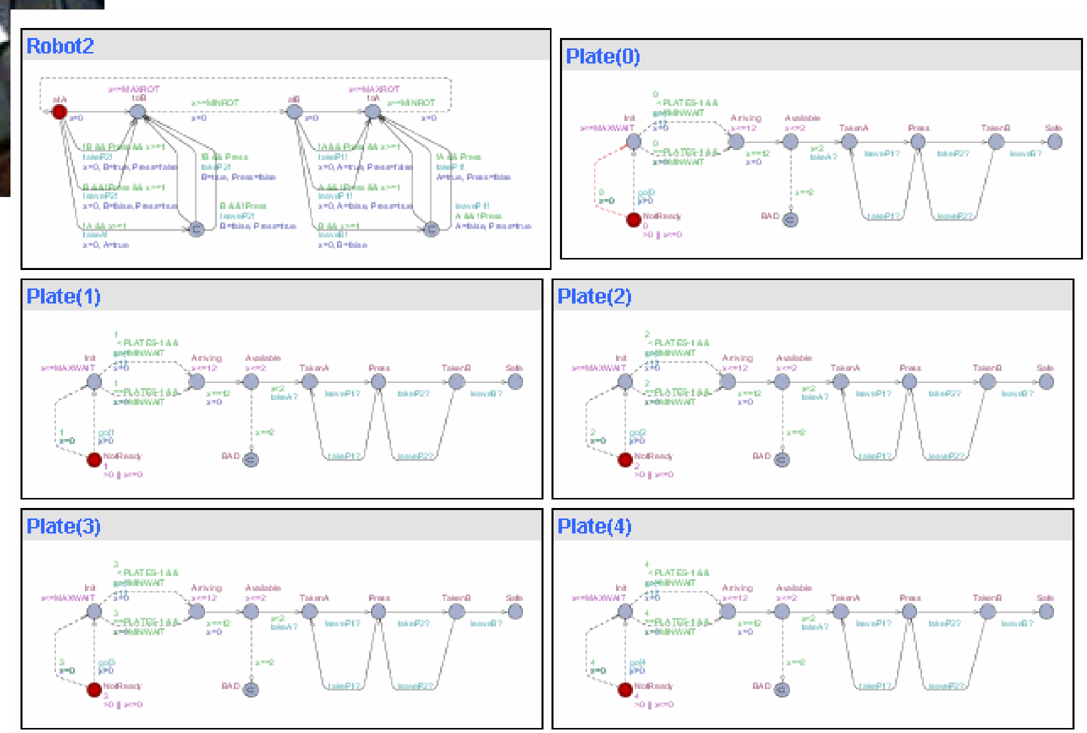
Strategy



BDD/CDD

Production Cell





Experimental Results

Plates		Basic		Basic +inc		Basic +inc +pruning		Basic+lose +inc +pruning		Basic+lose +inc +topt	
		time	mem	time	mem	time	mem	time	mem	time	mem
2	win	0.0s	1M	0.0s	1M	0.0s	1M	0.0s	1M	0.04s	1M
	lose	0.0s	1M	0.0s	1M	0.0s	1M	0.0s	1M	n/a	n/a
3	win	0.5s	19M	0.0s	1M	0.0s	1M	0.1s	1M	0.27s	4M
	lose	1.1s	45M	0.1s	1M	0.0s	1M	0.2s	3M	n/a	n/a
4	win	33.9s	1395M	0.2s	8M	0.1s	6M	0.4s	5M	1.88s	13M
	lose	-	-	0.5s	11M	0.4s	10M	0.9s	9M	n/a	n/a
5	win	-	-	3.0s	31M	1.5s	22M	2.0s	16M	13.35s	59M
	lose	-	-	11.1s	61M	5.9s	46M	7.0s	41M	n/a	n/a
6	win	-	-	89.1s	179M	38.9s	121M	12.0s	63M	220.3s	369M
	lose	-	-	699s	480M	317s	346M	135.1s	273M	n/a	n/a
7	win	-	-	3256s	1183M	1181s	786M	124s	319M	6188s	2457M
	lose	-	-	-	-	16791s	2981M	4075s	2090M	n/a	n/a

New Experimental Results Using UPPAAL 4.0 architecture

Model		3		6		12		50		100	
Old	c	0.1s	1M	12s	63M	-	-	-	-	-	-
	u	0.2s	3M	235s	273M	-	-	-	-	-	-
New	c	0.05s	3.5M	0.05s	3.5M	0.14s	55M	2.79s	104M	18.5s	426M
	u	0.02s	3.5M	0.04s	3.5M	0.12s	55M	2.32s	94M	15.6s	340M

Tricks (Alexandre):

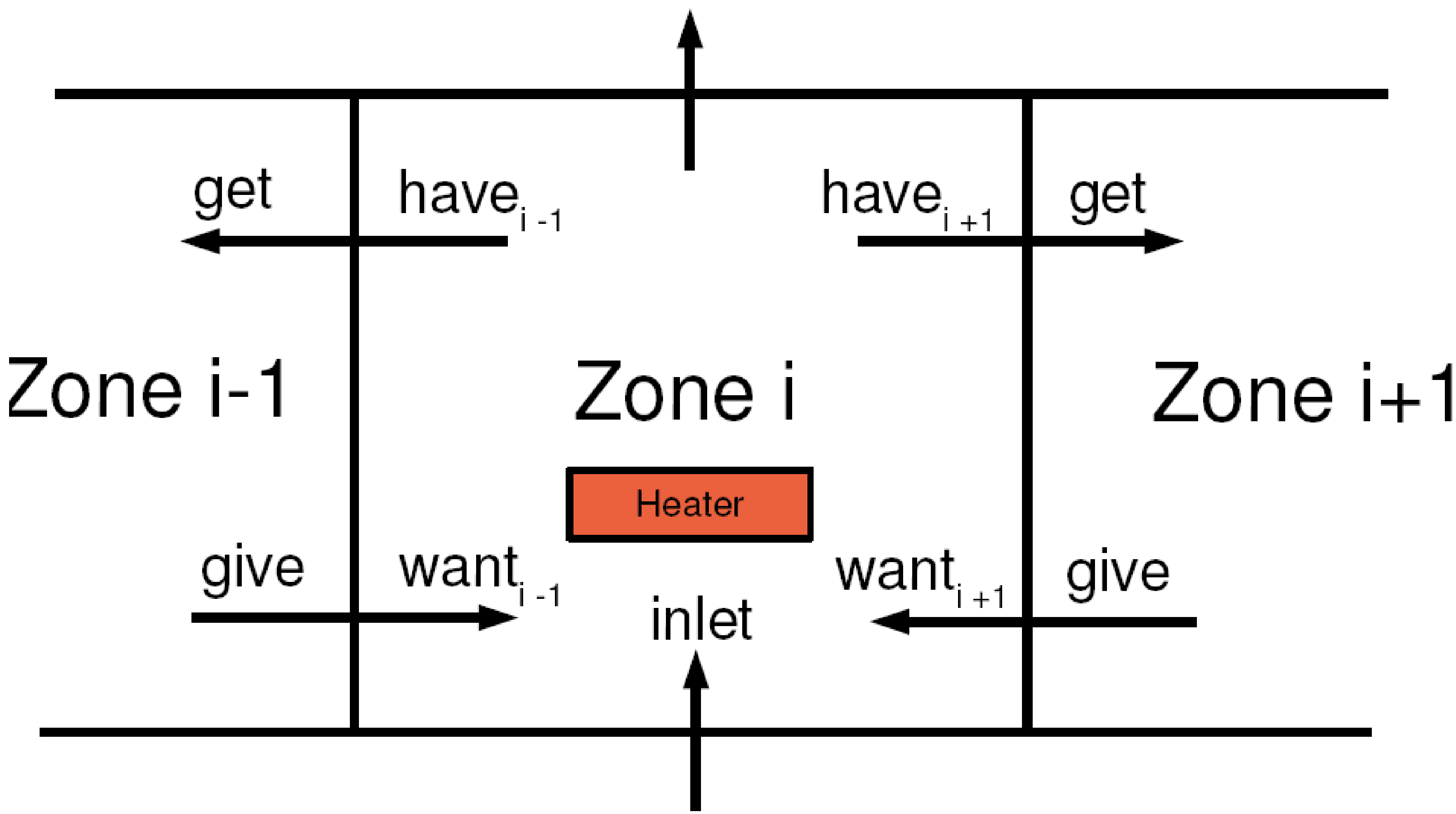
- UPPAAL pipeline architecture, which implies
  * active clock reduction
  * PW-list
  * UPPAAL optimizations (successor computation, postponed evaluation, reduced copies..)
  * improved DBM library
  * improved copy-on-write implementations
  * improved subtraction (vital)
  * enormously improved merge (between DBMs) (vital)

Climate Control

Syvsten,
Northern Jutland,
DK

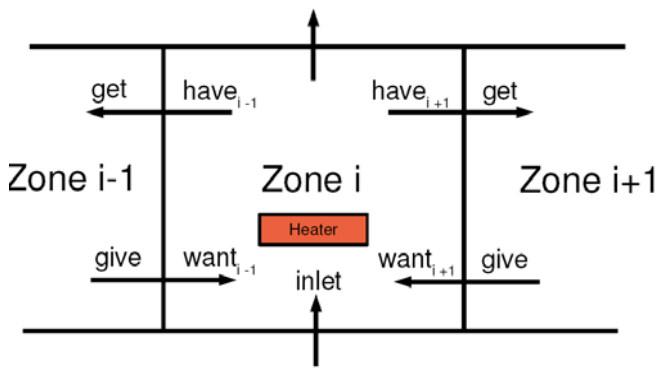
With Jan J. Jessen
Jacob I. Rasmussen

Climate Control

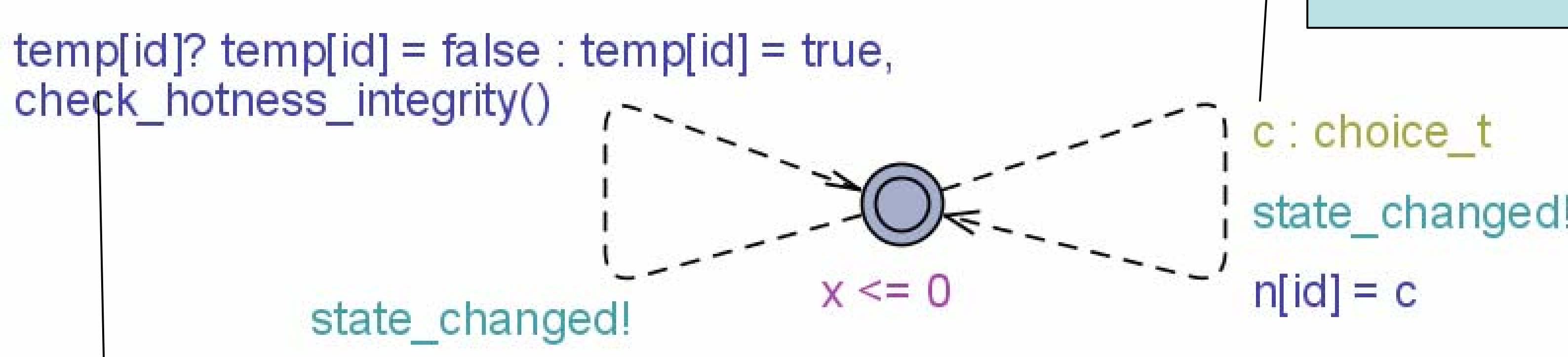


Climate Control / Neighbor

Neighboring zone

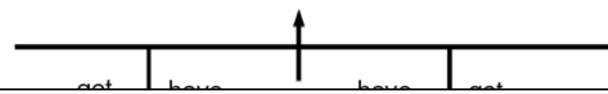


Neighbor wants to receive flow?



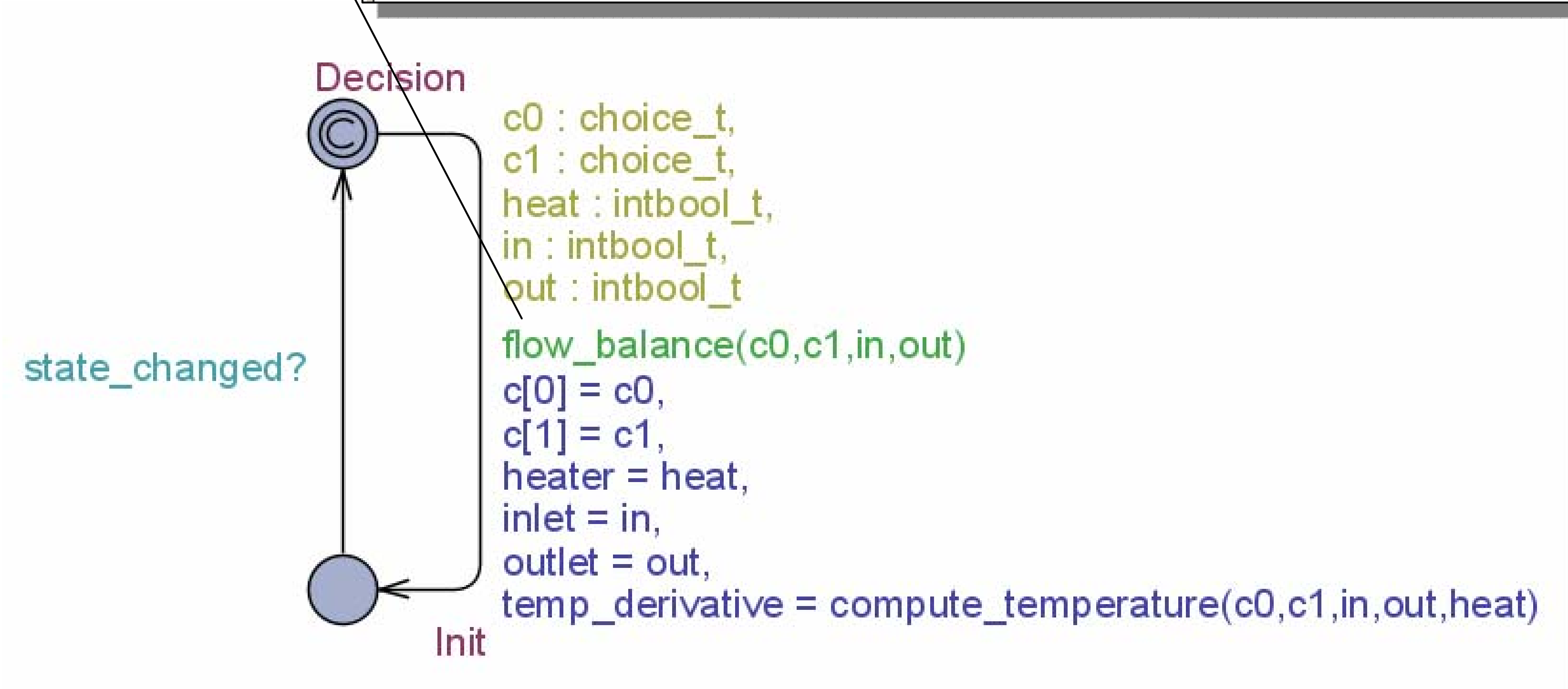
Temperature in neighbor zone (lower/higher)

Climate Control / Controller

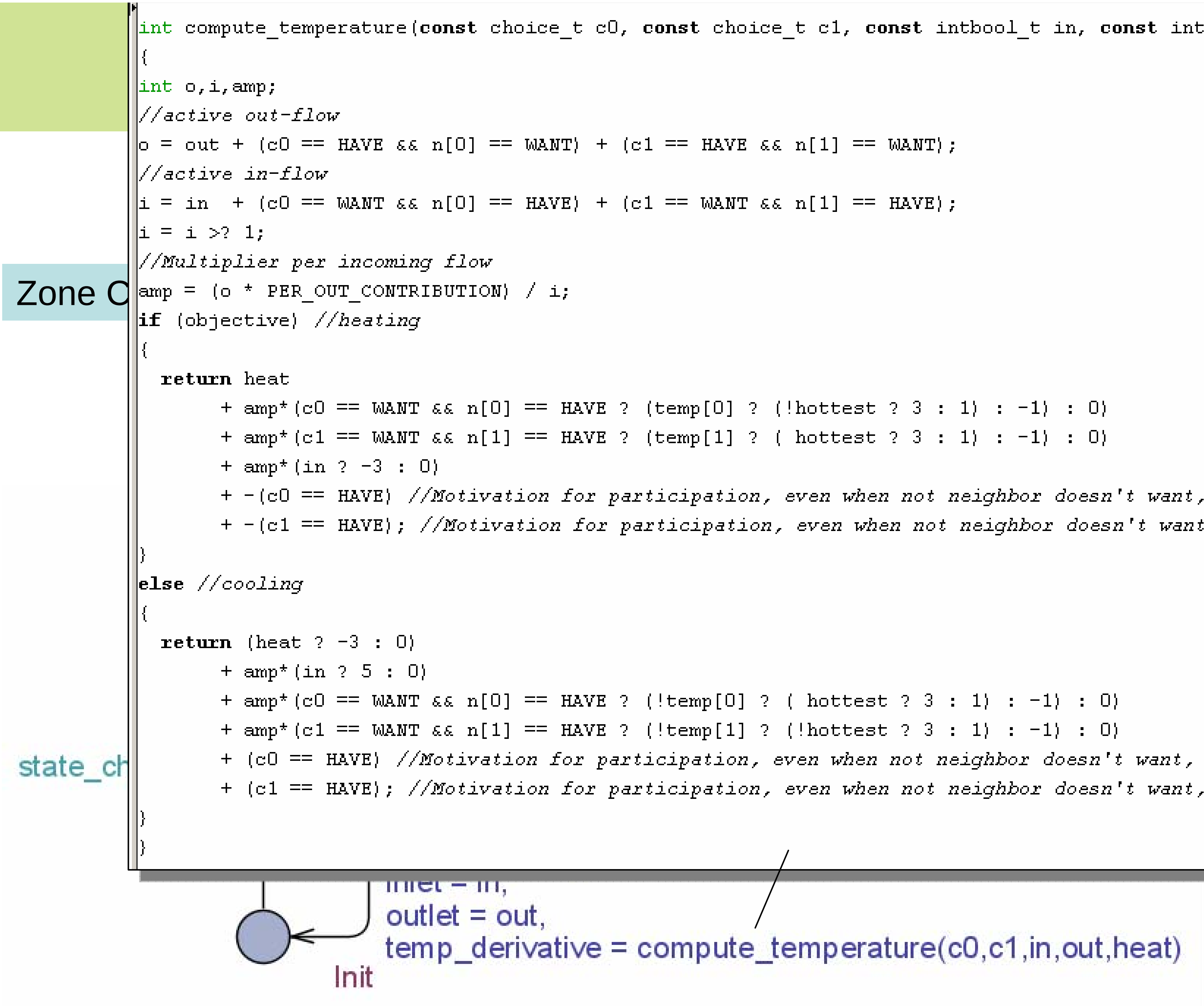


Zone Controller

```
bool flow_balance(const choice_t n0, const choice_t n1, bool in, bool out)
{
  bool o = out || (n[0] == WANT  && n0 == HAVE) || (n[1] == WANT && n1 == HAVE);
  bool i = in  || (n[0] ==HAVE   && n0 == WANT) || (n[1] == HAVE && n1 == WANT);
  return o == i;
}
```



```
int compute_temperature(const choice_t c0, const choice_t c1, const intbool_t in, const intbool_t out,
{
int o,i,amp;
//active out-flow
o = out + (c0 == HAVE && n[0] == WANT) + (c1 == HAVE && n[1] == WANT);
//active in-flow
i = in  + (c0 == WANT && n[0] == HAVE) + (c1 == WANT && n[1] == HAVE);
i = i >? 1;
//Multiplier per incoming flow
amp = (o * PER_OUT_CONTRIBUTION) / i;
if (objective) //heating
{
  return heat
        + amp*(c0 == WANT && n[0] == HAVE ? (temp[0] ? (!hottest ? 3 : 1) : -1) : 0)
        + amp*(c1 == WANT && n[1] == HAVE ? (temp[1] ? ( hottest ? 3 : 1) : -1) : 0)
        + amp*(in ? -3 : 0)
        + -(c0 == HAVE) //Motivation for participation, even when not neighbor doesn't want, og has air
        + -(c1 == HAVE); //Motivation for participation, even when not neighbor doesn't want, og has ai
}
else //cooling
{
  return (heat ? -3 : 0)
        + amp*(in ? 5 : 0)
        + amp*(c0 == WANT && n[0] == HAVE ? (!temp[0] ? ( hottest ? 3 : 1) : -1) : 0)
        + amp*(c1 == WANT && n[1] == HAVE ? (!temp[1] ? (!hottest ? 3 : 1) : -1) : 0)
        + (c0 == HAVE) //Motivation for participation, even when not neighbor doesn't want, og has air
        + (c1 == HAVE); //Motivation for participation, even when not neighbor doesn't want, og has air
}
}
```

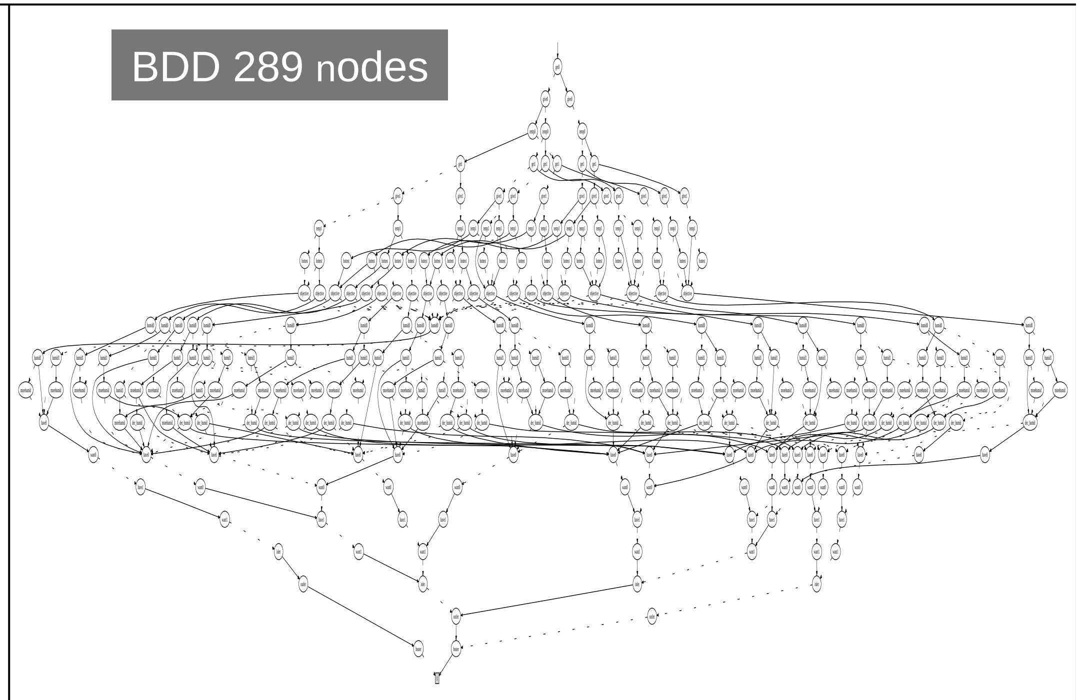


Obtaining executable code

```
-------------------
Strategy for state:
   Zone i-1: (Temp. lower/equal, wants flow)
   Zone i+1: (Temp. lower/equal, no interaction)
   Hottest neighbor: i-1
   Objective: heat
is:
   Wants flow from i-1
   Wants flow from i+1
   inlet closed
   outlet off
   heater on
-------------------
Strategy for state:
   Zone i-1: (Temp. greater, offers flow)
   Zone i+1: (Temp. greater, offers flow)
   Hottest neighbor: i+1
   Objective: cool
is:
   Has flow for i-1
   Has flow for i+1
   inlet open
   outlet on
   heater off
-------------------
Strategy for state:
   Zone i-1: (Temp. lower/equal, no interaction)
   Zone i+1: (Temp. greater, no interaction)
   Hottest neighbor: i+1
   Objective: cool
```

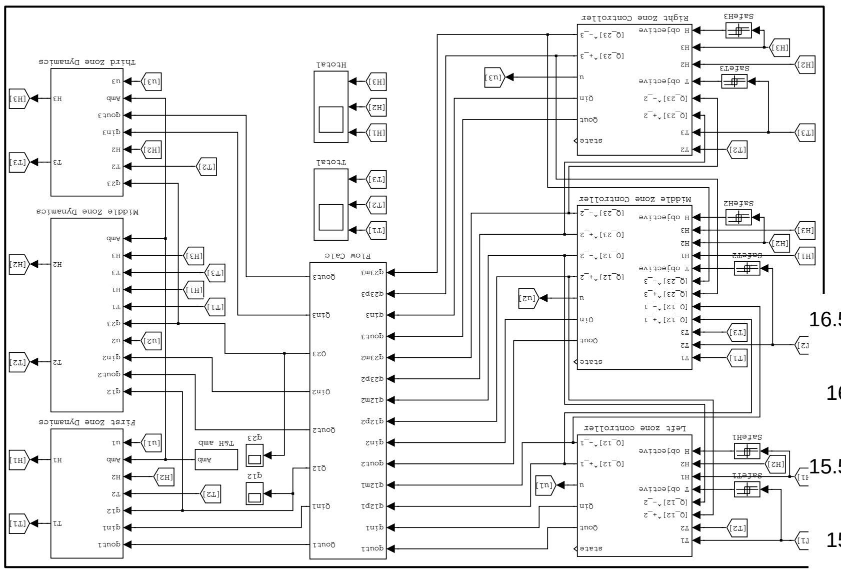
Stragegy

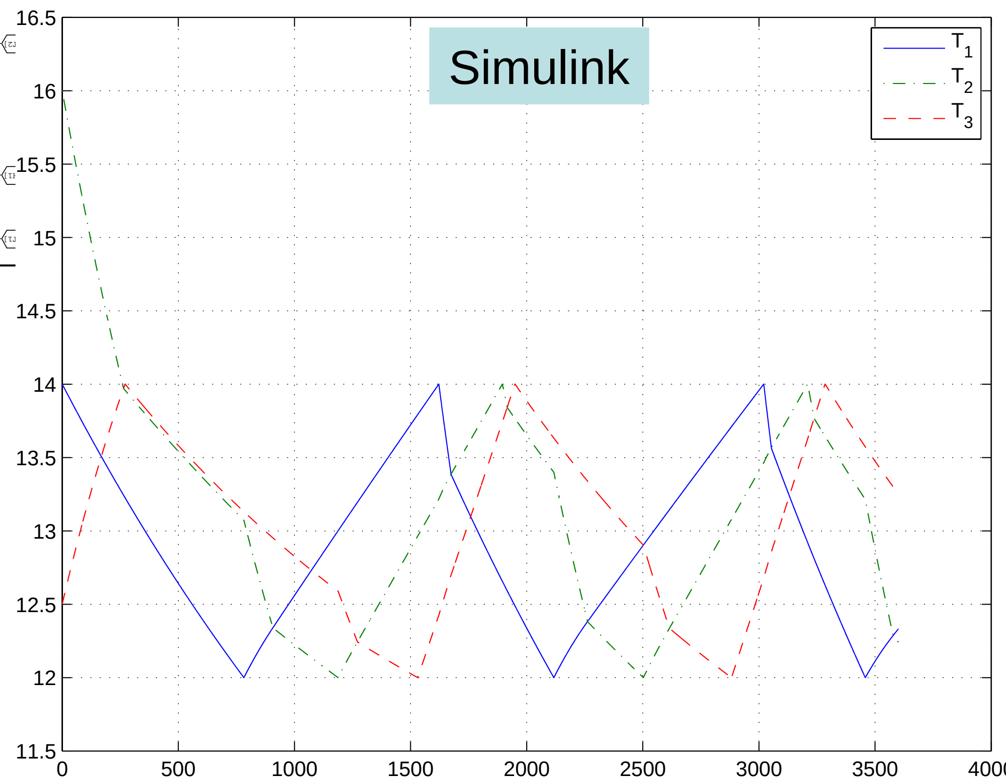
1296 cases



control : A[]
((ZC.Init && objective) imply temp_derivative > 0) &&
((ZC.Init && !objective) imply temp_derivative < 0)

Obtaining executable code





Conclusion & Future Work

- More Applications - we need you !
- Efficient Algorithms for Optimal Infinite Scheduling
- Multipriced Timed Automata
- Priced Timed Games
  - Optimal strategies undecidable in general [Raskin ao]
  - Decidability in setting of 1 clock or strong non-zenoness.
- Timed Games with Imperfect information.
- Distributed and parallel implementations (PC clusters, GRID, Shared Memory Machines)

Reading material

- **UPPAAL Cora (Priced Timed Automata):**
  - Buhrmann, Larsen, Rasmussen: Optimal Scheduling using Priced Timed Automata, ACM SIGMETRICS Performance Evaluation Review, vol. 32, nb. 4, 2005, pp. 34-40, ACM Press.
  - Priced Timed Automata: Algorithms, and Applications G. Behrmann, K. G. Larsen, J. I. Rasmussen. In proc. of FMCO'04, LNCS vol. 3657, pp. 162-186, Springer Verlag, 2005.
  - Patricia Bouyer. Weighted Timed Automata: Model-Checking and Games. In Proceedings of the 22nd Conference on Mathematical Foundations of Programming Semantics (MFPS'06), Genova, Italy, May 2006, ENTCS 158, pages 3-17. Elsevier Science Publishers.
- **UPPAAL Tiga (Timed Games & Controller Synthesis):**
  - Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. CONCUR05, volume 3653 of Lecture Notes in Computer Science.
  - Patricia Bouyer and Fabrice Chevalier. On the Control of Timed and Hybrid Systems. EATCS Bulletin 89, pages 79-96, 2006.
  - J.J. Jessen, J.I.Rasmussen, K.G.Larsen, A.David: Guided Controller Synthesis for Climate Controller using UPPAAL TIGA. Under Submission.