Automated Formal Methods for Embedded Systems

Bernd Finkbeiner

Universität des Saarlandes
Reactive Systems Group

2011/02/03

Automated Formal Methods

- Model Checking: automatically verify whether certain properties are guaranteed by the model; determine safe parameters
- Controller Synthesis: automatically construct control strategies that keep the system safe

Overview:

1. Intro: Analyzing FlexRay
2. Timed automata
3. Regions & zones
4. Model checking and controller synthesis

FlexRay Bus Protocol

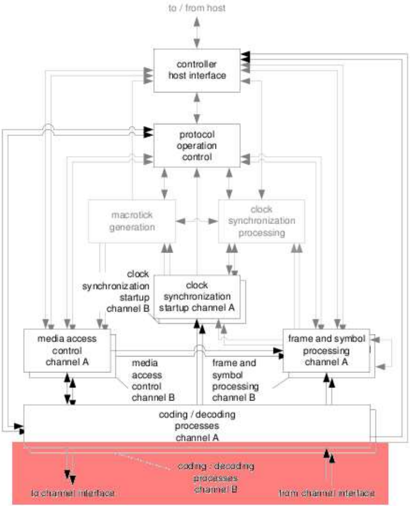
FlexRay

- communication protocol for distributed components in cars
- used in BMW X 5 and BMW's 7 series for X-by-wire
- developed by: BMW, Bosch, Daimler, Freescale, General Motors, NXP Semiconductors, Volkswagen, et al.

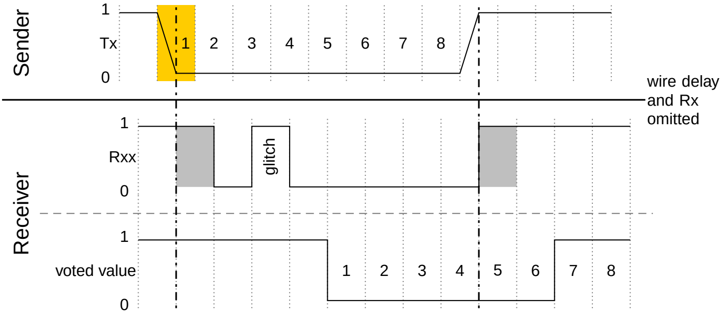
FlexRay as the Future Drive-by-Wire Standard

$\Rightarrow$ **Safety**-critical!

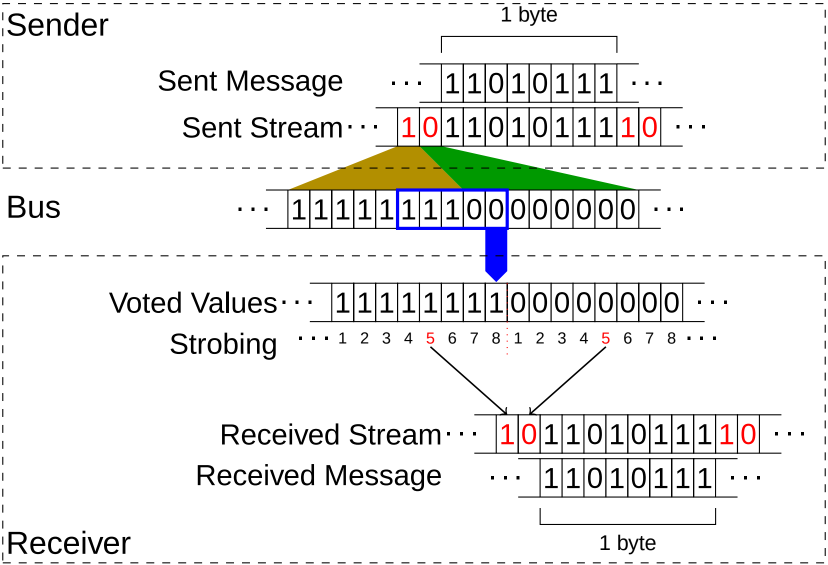
FlexRay Physical Layer



Jitter and Glitch Correction



Protocol Operation

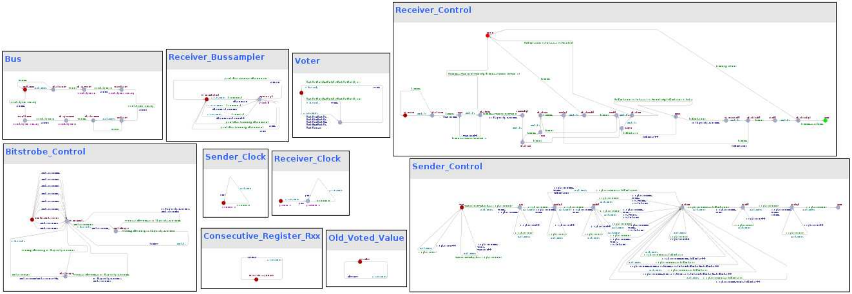


Guaranteed Error Resilience?

Newest FlexRay Specification, Version 2.1, Revision A:

> *"[FlexRay] attempts to enable tolerance of the physical layer against presence of one glitch in a bit cell [. . .]. There are specific cases where a single glitch cannot be tolerated and others where two glitches can be tolerated."*

Michael Gerke's Model of the Protocol



- protocol
- jitter (parameterized)
- glitches

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: . . .  . . .

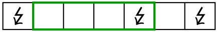
Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: . . .  . . .

Automated Analysis: Glitch Tolerance

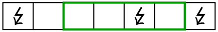
The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: . . .  . . .

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

  E.g.: . . .  . . .

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
  E.g.: . . .  . . .

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

  E.g.: …  …

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 arbitrarily placed glitches in the complete message (at most 2)

Note: one message $\approx$ 21.000 samples

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 arbitrarily placed glitches in the complete message (at most 2)
  E.g.: . . . [ ][↯][ ][↯][ ][ ][ ] . . .

Note: one message $\approx$ 21.000 samples

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 arbitrarily placed glitches in the complete message (at most 2)

Note: one message $\approx$ 21.000 samples

The protocol does not tolerate:
2 arbitr. placed glitches in every seq. of 82 consec. samples (2 out of 82)

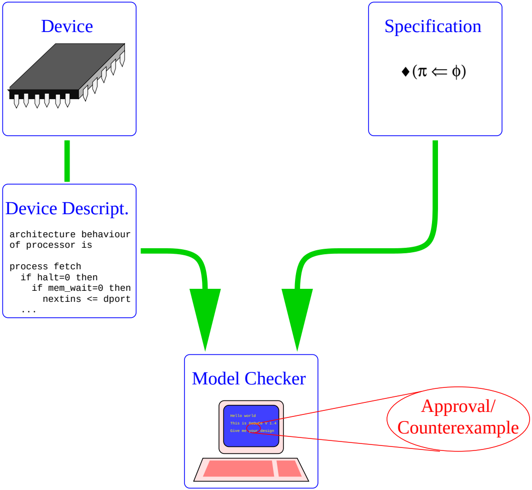
Automated Analysis: Glitch Tolerance vs. Delay Variance

Parameter exploration using binary search:
boundaries for variation of a single parameter

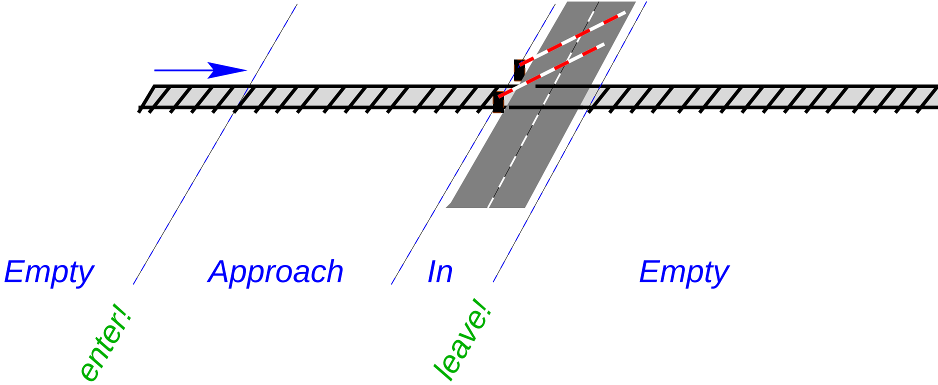
glitch tolerance	delay variance
(1 out of 4)	$1.435ns \rightarrow 7.6075ns$
(2 at most)	$1.435ns \rightarrow 7.6075ns$
(1 at most)	$1.435ns \rightarrow 12.020ns$

glitch tolerance	deviation of clock from standard rate
(1 out of 4)	$0.15\% \rightarrow 0.46\%$
(2 at most)	$0.15\% \rightarrow 0.46\%$
(1 at most)	$0.15\% \rightarrow 1.09\%$
(no glitches)	$0.15\% \rightarrow 1.74\%$

Model Checking

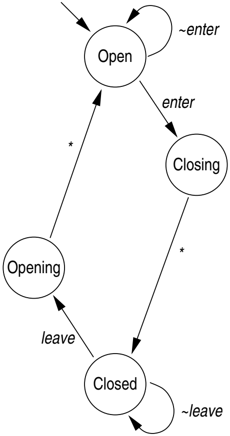


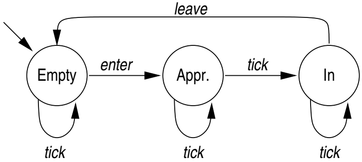
Finite-State Model-Checking



Safety requirement: Gate has to be closed whenever a train is in "In".

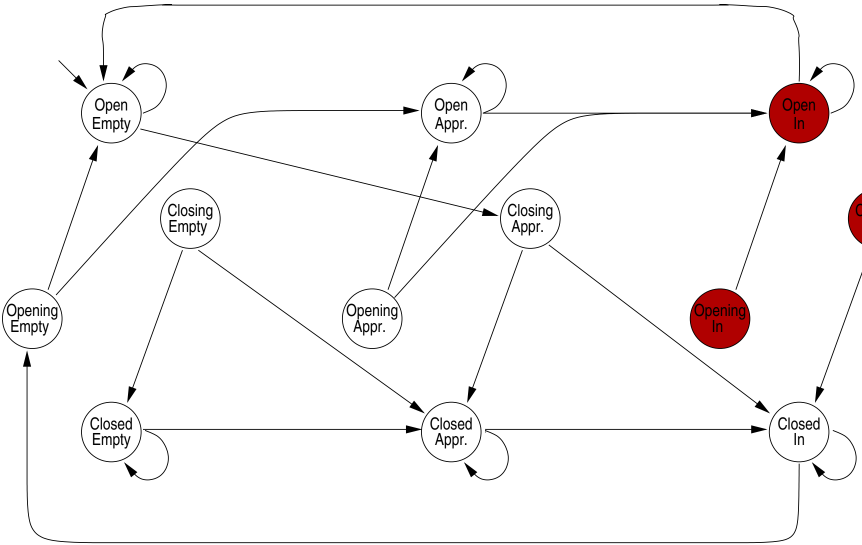
Finite-State Automata



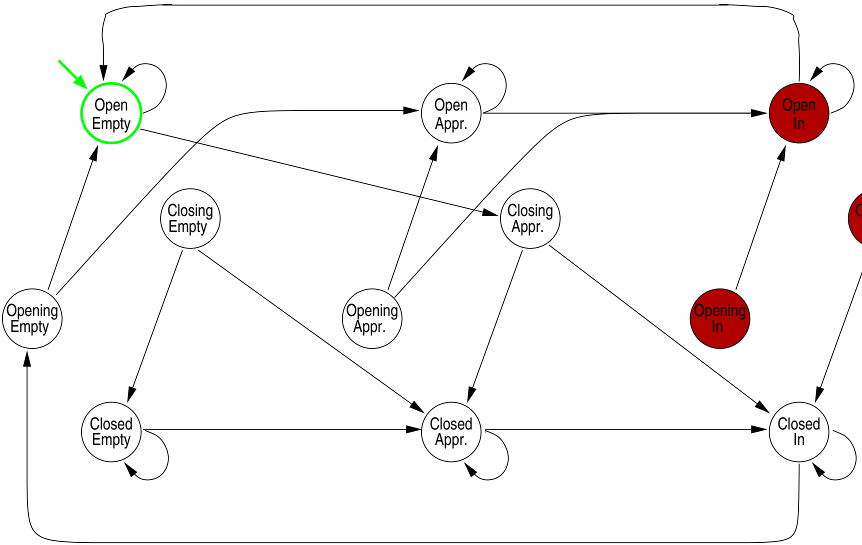


** = leave,enter,tick*
~leave = enter, tick
~enter = leave, tick

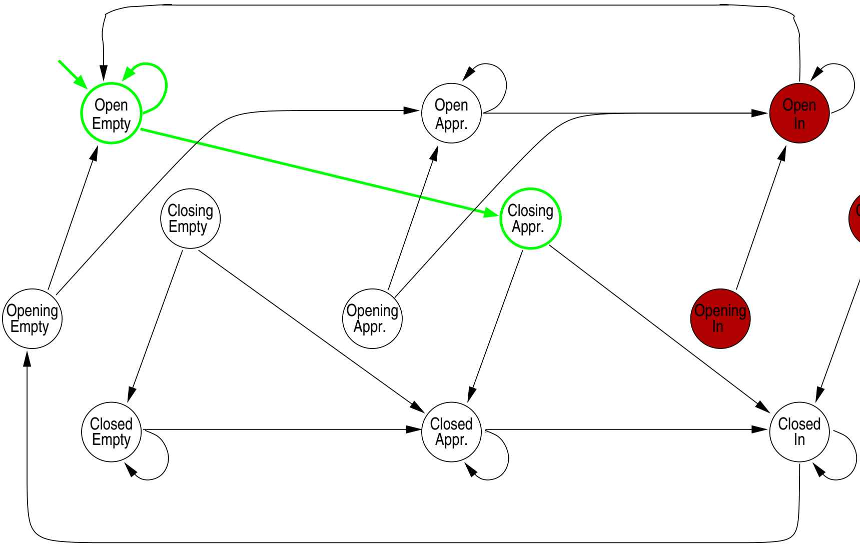
Model Checking



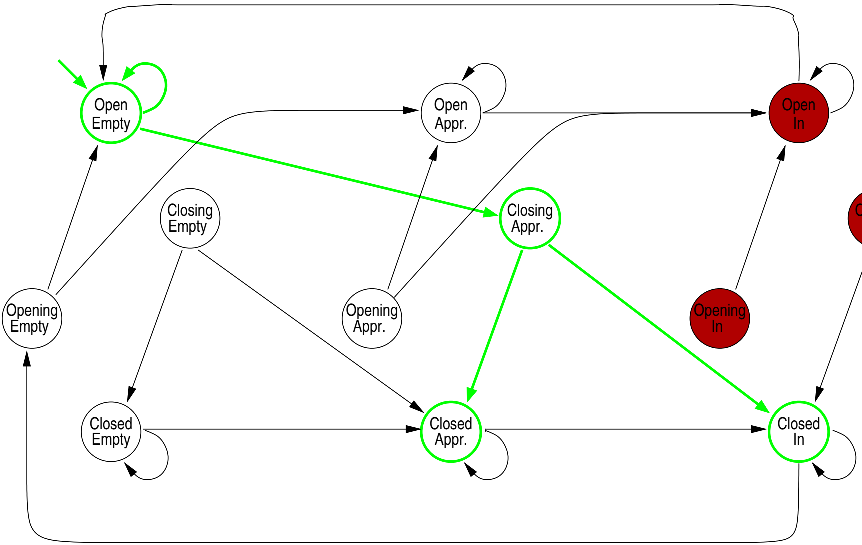
Model Checking



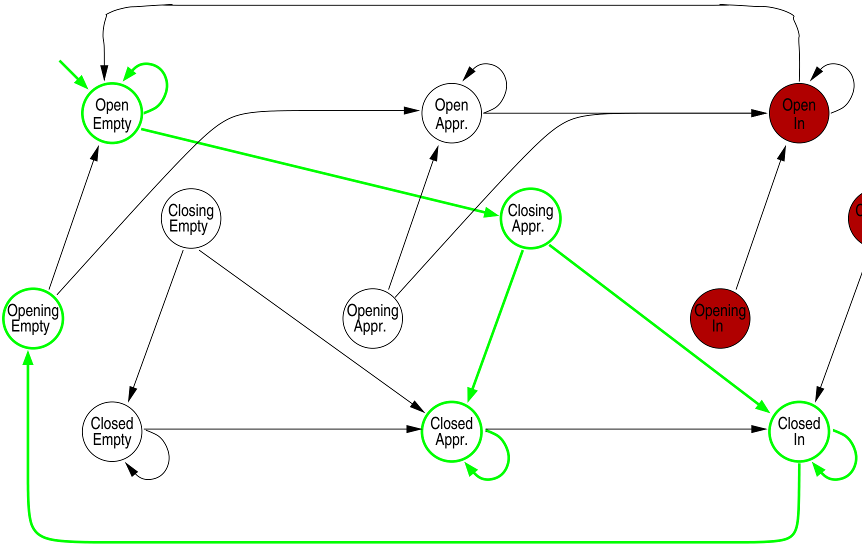
Model Checking



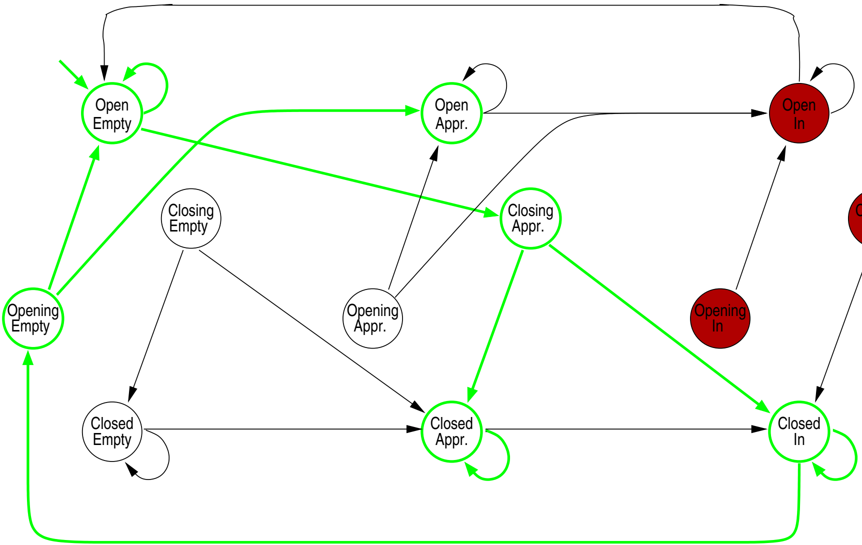
Model Checking



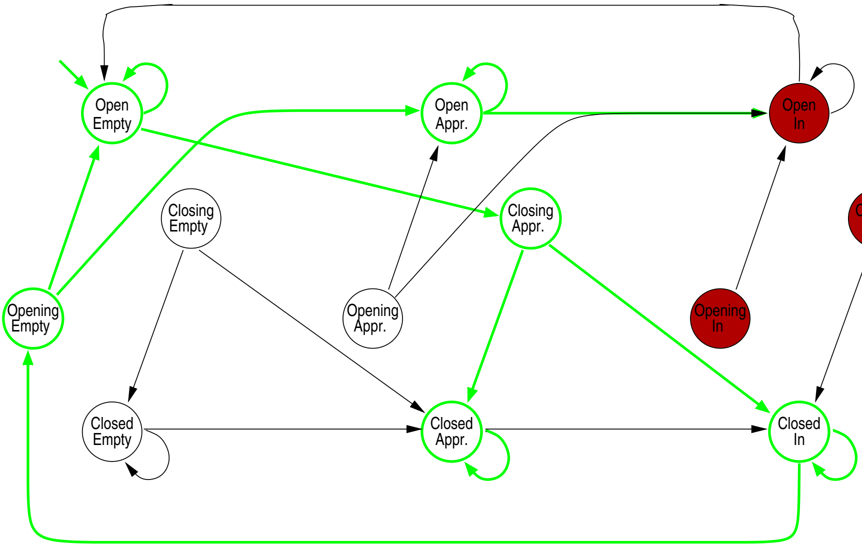
Model Checking



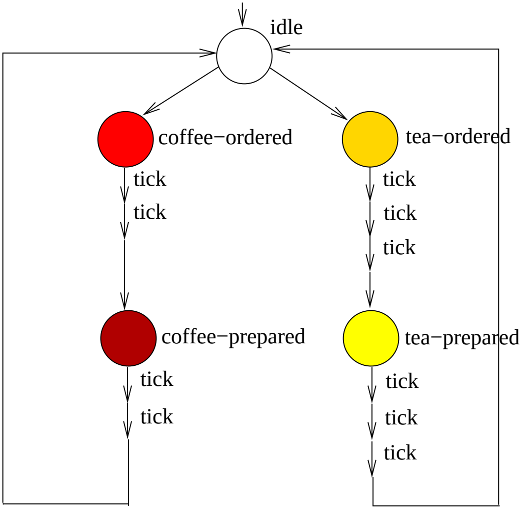
Model Checking



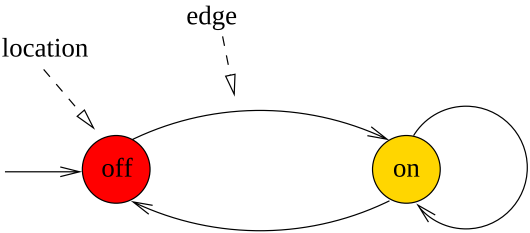
Model Checking



A Discrete-Time Coffee Machine

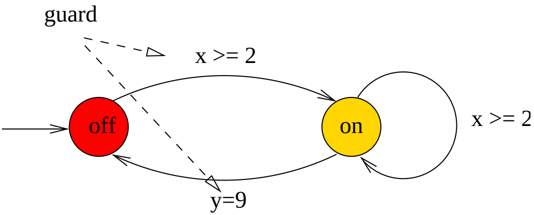


Timed Automata



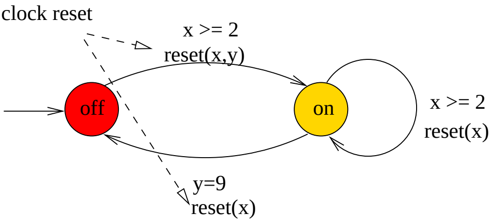
- a graph with *locations* and *edges*
- a location is labeled with the valid *atomic propositions*
- *taking an edge is instantaneous*, i.e, consumes no time

Timed Automata



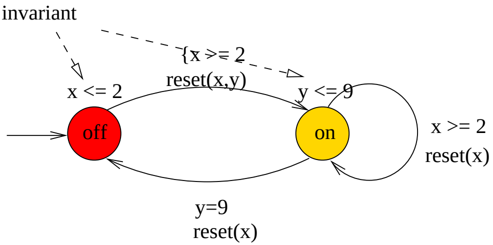
- equipped with real-valued *clocks* $x, y, z, \ldots$
- clocks advance implicitly, all at the *same speed*
- logical constraints on clocks can be used as *guards* of actions

Timed Automata



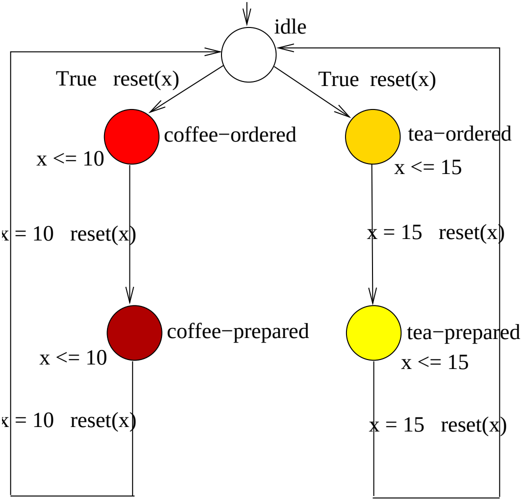
- clocks can be *reset* when taking an edge
- assumption:
  all clocks are zero when entering the initial location initially

Timed Automata



- guards indicate when an edge *may* be taken
- a location invariant specifies the *amount of time that may be spent in a location*
  - before a *location invariant* becomes invalid, an edge must be taken

A Real-Time Coffee Machine



Clock Constraints

Clock constraints over set C of clocks are defined by:

$$g ::= \quad True \;\Big|\; x < c \;\Big|\; x \leq c \;\Big|\; \neg g \;\Big|\; g \wedge g$$

- where $c \in \mathbb{N}$ and clocks $x, y \in C$
- rational constants would do; neither reals nor addition of clocks!
- let $CC(C)$ denote the set of clock constraints over C
- shorthands: $x \geq c$ denotes $\neg(x < c)$
  and $x \in [c_1, c_2)$ or $c_1 \leq x < c_2$ denotes $\neg(x < c_1) \wedge (x < c_2)$

Timed Automaton

A *timed automaton* is a tuple

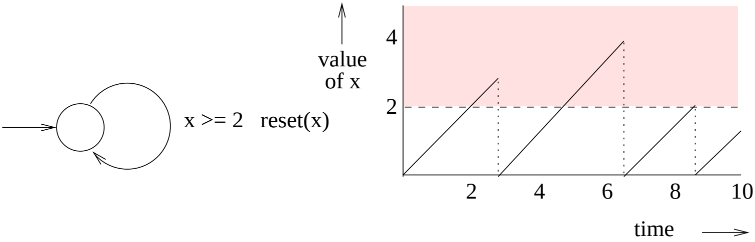
$$TA = \left(Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L\right) \quad \text{where:}$$

- Loc is a finite set of locations.
- $Loc_0 \subseteq Loc$ is a set of initial locations
- C is a finite set of clocks
- $L : Loc \rightarrow 2^{AP}$ is a labeling function for the locations
- $\rightsquigarrow \subseteq Loc \times CC(C) \times Act \times 2^{C} \times Loc$ is a transition relation, and
- $inv : Loc \rightarrow CC(C)$ is an invariant-assignment function

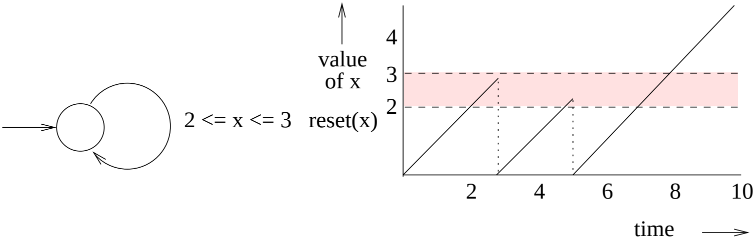
Intuitive Interpretation

- Edge $\ell \xrightarrow{g:\alpha,C'} \ell'$ means:
  - action α is enabled once guard g holds
  - when moving from location ℓ to ℓ', any clock in C' will be reset to zero
- $inv(\ell)$ constrains the amount of time that may be spent in location ℓ
  - the location ℓ must be left before the invariant $inv(\ell)$ becomes invalid

The effect of a lowerbound guard:

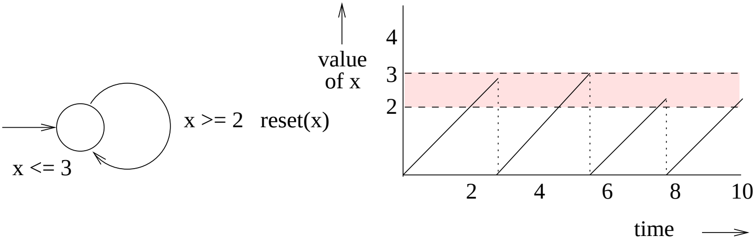


The effect of a lowerbound and upperbound guard:

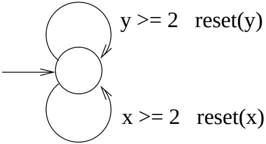


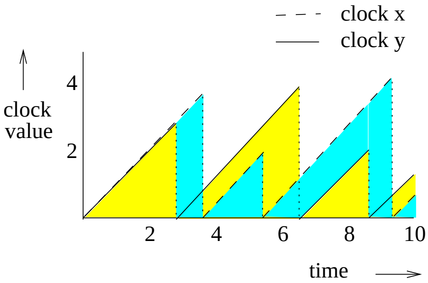
Guards vs. Location Invariants

The effect of a guard and an invariant:



Arbitrary Clock Differences





Composing Timed Automata

Let $TA_i = \big(Loc_i, Act_i, C_i, \rightsquigarrow_i, Loc_{0,i}, inv_i, AP, L_i\big)$ and H an action-set

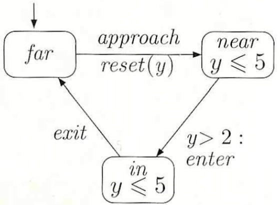
$TA_1 \;||_H\; TA_2 = \big(Loc, Act_1 \cup Act_2, C, \rightsquigarrow, Loc_0, inv, AP, L\big)$ where:

- $Loc = Loc_1 \times Loc_2$ and $Loc_0 = Loc_{0,1} \times Loc_{0,2}$ and $C = C_1 \cup C_2$
- $inv(\langle \ell_1, \ell_2 \rangle) = inv_1(\ell_1) \;\wedge\; inv_2(\ell_2)$ and $L(\langle \ell_1, \ell_2 \rangle) = L_1(\ell_1) \cup L_2(\ell_2)$
- $\rightsquigarrow$ is defined by the inference rules:

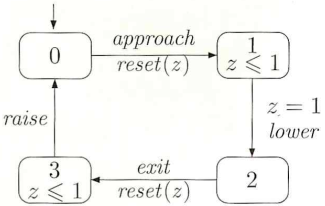
for $\alpha \in H$ $$\frac{\ell_1 \overset{g_1:\alpha,D_1}{\rightsquigarrow_1} \ell_1' \;\wedge\; \ell_2 \overset{g_2:\alpha,D_2}{\rightsquigarrow_2} \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g_1 \wedge g_2:\alpha,D_1 \cup D_2}{\rightsquigarrow} \langle \ell_1', \ell_2' \rangle}$$

for $\alpha \notin H$: $$\frac{\ell_1 \overset{g:\alpha,D}{\rightsquigarrow_1} \ell_1'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha,D}{\rightsquigarrow} \langle \ell_1', \ell_2 \rangle}$$ and $$\frac{\ell_2 \overset{g:\alpha,D}{\rightsquigarrow_2} \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha,D}{\rightsquigarrow} \langle \ell_1, \ell_2' \rangle}$$

Example: Railroad Crossing

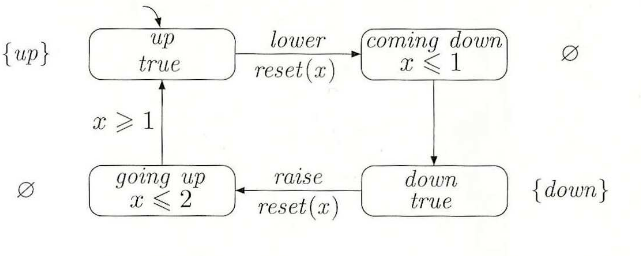


Train



Controller

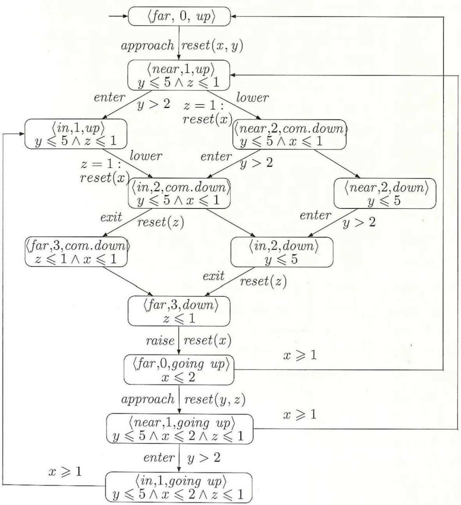
Example: Railroad Crossing



Gate

Example: Railroad Crossing

$$(Train_{\{approach,exit\}}||Controller)||_{\{lower,raise\}}Gate$$



Clock valuations

- A *clock valuation* v for set C of clocks is a function $v : C \to \mathbb{R}_{\geq 0}$
  - assigning to each clock $x \in C$ its current value $v(x)$
- Clock valuation $v+d$ for $d \in \mathbb{R}_{\geq 0}$ is defined by:
  - $(v+d)(x) = v(x) + d$ for all clocks $x \in C$
- Clock valuation reset x in v for clock x is defined by:

$$(\text{reset } x \text{ in } v)(y) = \begin{cases} v(y) & \text{if } y \neq x \\ 0 & \text{if } y = x. \end{cases}$$

  - reset x in (reset y in v) is abbreviated by reset x, y in v

Timed automaton semantics

For timed automaton $TA = \big(Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L\big)$:
state graph $S(TA) = (Q, Q_0, E, L')$ over AP where:

- $Q = Loc \times val(C)$, state $s = \langle \ell, v \rangle$ for location ℓ and clock valuation v
- $Q_0 = \{ \langle \ell_0, v_0 \rangle \mid \ell_0 \in Loc_0 \;\wedge\; v_0(x) = 0$ for all $x \in C \}$
- $L'(\langle \ell, v \rangle) = L(\ell)$
- E is the edge set defined on the next slide

Timed automaton semantics

The edge set E consist of the following two types of transitions:

- Discrete transition: $\langle \ell, v \rangle \xrightarrow{\alpha} \langle \ell', v' \rangle$ if all following conditions hold:
  - there is an edge labeled $(g : \alpha, D)$ from location ℓ to ℓ' such that:
  - g is satisfied by v, i.e., $v \models g$
  - $v' = v$ with all clocks in D reset to 0, i.e., $v' =$ reset D in v
  - v' fulfills the invariant of location ℓ', i.e., $v' \models inv(\ell')$
- Delay transition: $\langle \ell, v \rangle \xrightarrow{d} \langle \ell, v{+}d \rangle$ for positive real d
  - if for any $0 \leq d' \leq d$ the invariant of ℓ holds for $v{+}d'$, i.e. $v{+}d' \models inv(\ell)$

Time divergence

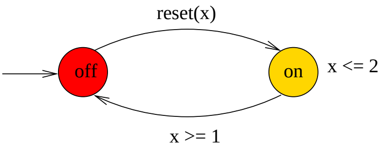
- Let for any $t < d$, for fixed $d \in \mathbb{R}_{>0}$, clock valuation $\eta{+}t \models inv(\ell)$
- A possible execution fragment starting from the location ℓ is:

$$\langle \ell, \eta \rangle \xrightarrow{d_1} \langle \ell, \eta{+}d_1 \rangle \xrightarrow{d_2} \langle \ell, \eta{+}d_1{+}d_2 \rangle \xrightarrow{d_3} \langle \ell, \eta{+}d_1{+}d_2{+}d_3 \rangle \xrightarrow{d_4} \ldots$$

  - where $d_i > 0$ and the infinite sequence $d_1 + d_2 + \ldots$ *converges* towards d
  - such path fragments are called *time-convergent*

  $\Rightarrow$ time advances only up to a certain value
- Time-convergent execution fragments are unrealistic and *ignored*

Example: light switch



The path

$$\pi = \langle \mathit{off}, 0\rangle\, \langle \mathit{off}, 1\rangle\, \langle \mathit{on}, 0\rangle\, \langle \mathit{on}, 1\rangle\, \langle \mathit{off}, 1\rangle\, \langle \mathit{off}, 2\rangle\, \langle \mathit{on}, 0\rangle\, \langle \mathit{on}, 1\rangle\, \langle \mathit{off}, 1\rangle \ldots$$

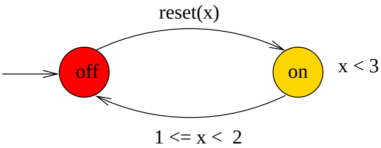
is *time-divergent*.

The path

$$\pi' = \langle \mathit{off}, 0\rangle\, \langle \mathit{off}, 1/2\rangle\, \langle \mathit{off}, 3/4\rangle\, \langle \mathit{off}, 7/8\rangle\, \langle \mathit{off}, 15/16\rangle \ldots$$

is *time-convergent*.

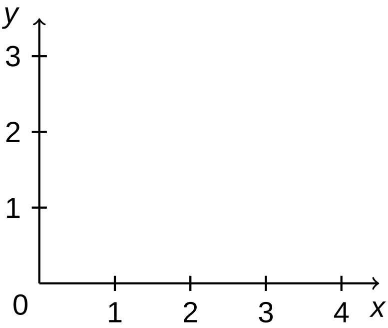
Timelock



- State $s \in S(TA)$ contains a *timelock* if there is a reachable state s where there is no time-divergent path from s
- Timelocks are considered as *modeling flaws* that should be avoided

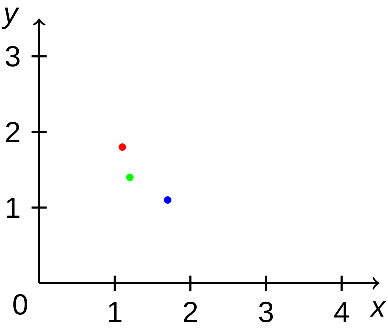
Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.



Region Abstraction

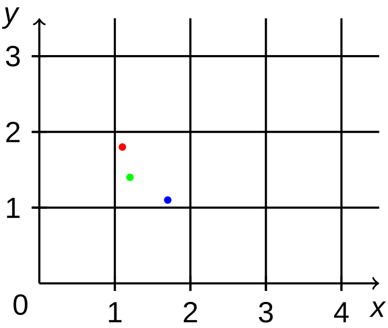
- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.



Equivalence relation $\simeq_R$

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

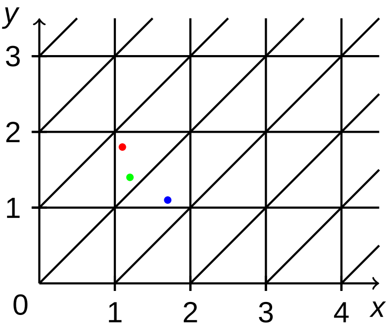


Equivalence relation $\simeq_R$

1. constraints

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

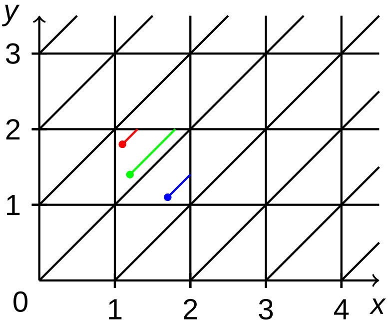


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

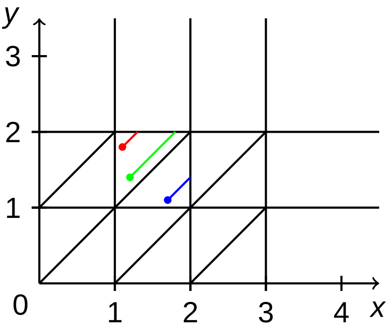


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

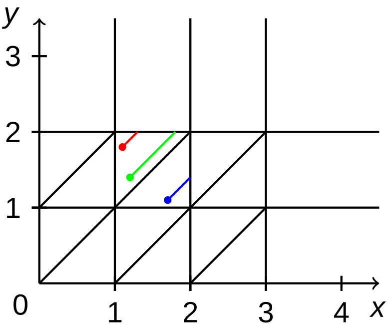


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing
3. maximal constants

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

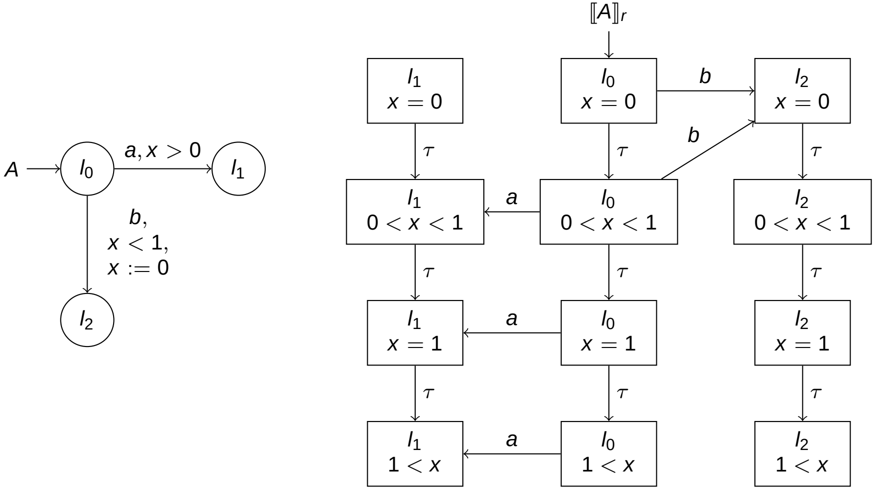


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing
3. maximal constants

$\Longrightarrow$ finite index!

Finite Semantics: Region Automaton



Timed Analysis

Reachability is decidable

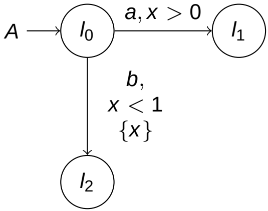
Theorem [Alur, 1994]:

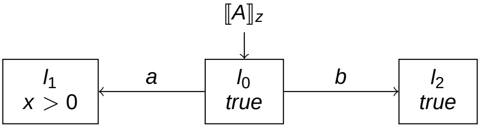
$$\exists \text{ path } (l, \vec{t}) \longrightarrow (l', \vec{t}')$$
$$\text{iff}$$
$$\exists \text{ path } (l, [\vec{t}]_R) \longrightarrow (l', [\vec{t}']_R)$$

Symbolic data structures

- Clock Region = Finest integral unit
- Clock Zone = Convex union of clock regions
- Federation = (Non-convex) union of clock zones

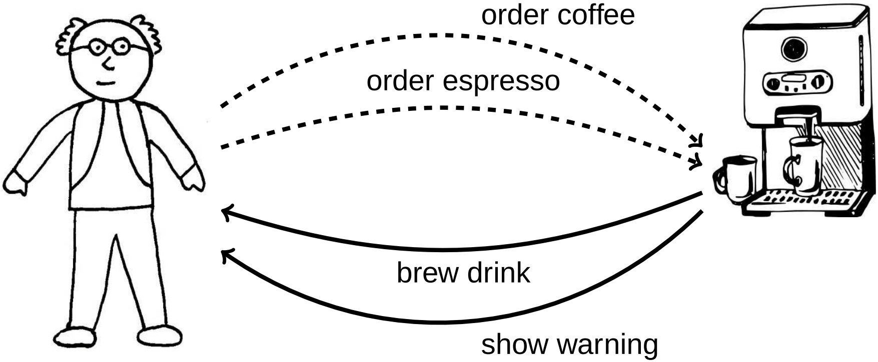
Finite Semantics: Zone Graph





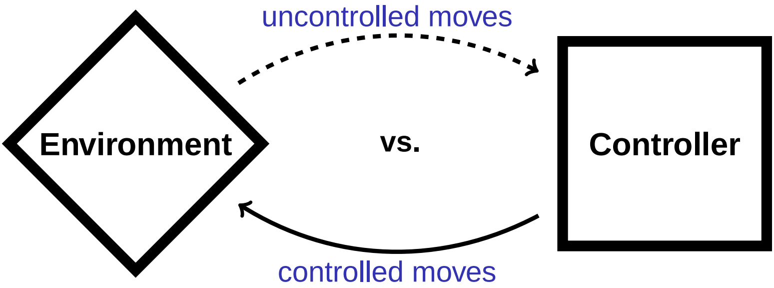
Controller Synthesis

We distinguish between external (uncontrolled) and internal (controlled) nondeterminism



Games

Game between two players

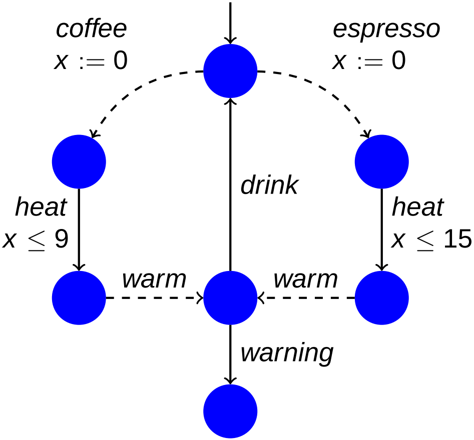


"wants to violate the spec."

"wants to satisfy the spec."

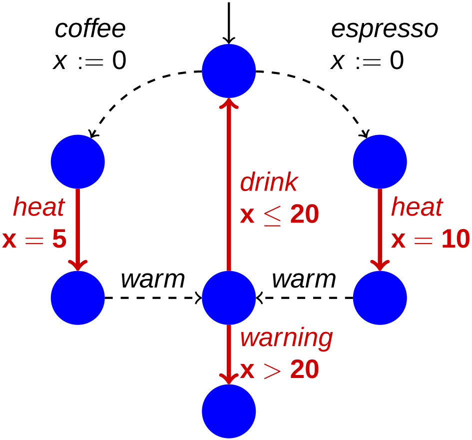
Games

Plants are modeled as timed game automata (TGA)



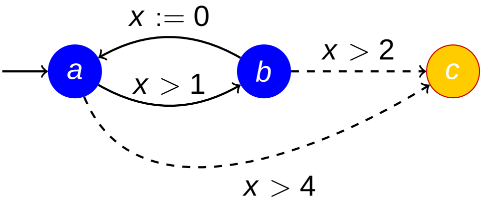
Timed Controllers

Controller = subautomaton representing winning strategies



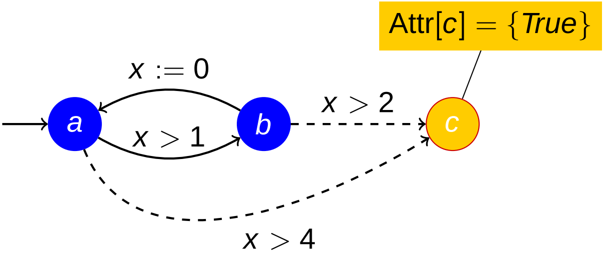
Reachability Games Played on Timed Automata

From where can $\dashrightarrow$ enforce a run to c ?



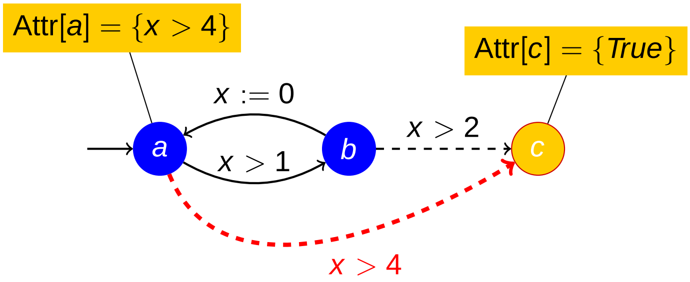
Zone-based Timed Game Solving

From where can $\dashrightarrow$ enforce a run to c ?



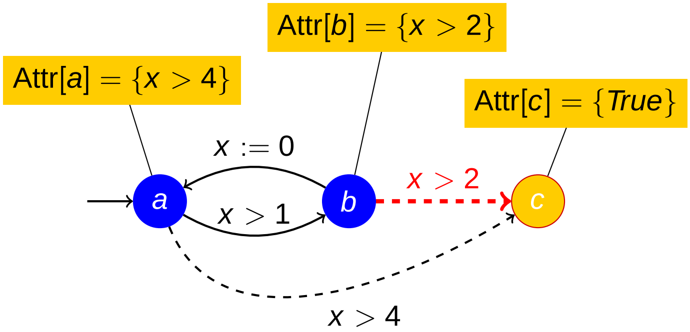
Zone-based Timed Game Solving

From where can ⇢ enforce a run to c ?



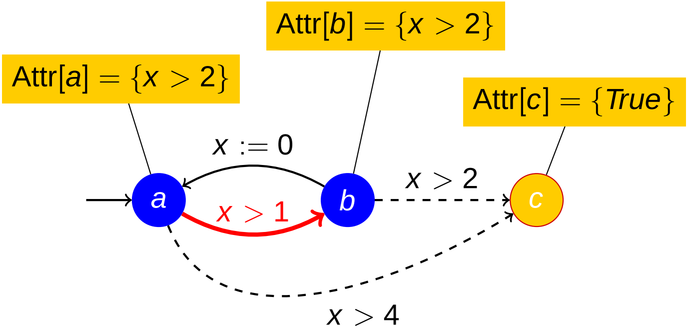
Zone-based Timed Game Solving

From where can ⇢ enforce a run to c ?



Zone-based Timed Game Solving

From where can $\dashrightarrow$ enforce a run to c ?



Summary

- Timed automata
- Automatic verification
- Automatic controller synthesis