

# Real-time models: Timed Automata

---

David Pereira   José Proença   Eduardo Tovar

FVOCA 2021/2022

Formal Verification of Critical Applications

CISTER – ISEP

Porto, Portugal

<https://cister-labs.github.io/fvoca2122>

# Motivation

---

Specifying an airbag saying that **in a car crash the airbag eventually inflates** maybe not enough, but:

in a car crash the airbag eventually inflates **within 20ms**

*Correctness in time-critical systems not only depends on the logical result of the computation, but also **on the time at which the results are produced***

[Baier & Katoen, 2008]

# Examples of time-critical systems

## Lip-synchronization protocol

Synchronizes the separate video and audio sources bounding on the amount of time mediating the presentation of a video frame and the corresponding audio frame. Humans tolerate less than 160 ms.

## Bounded retransmission protocol

Controls communication of large files over infrared channel between a remote control unit and a video/audio equipment. Correctness depends crucially on

- transmission and synchronization delays
- time-out values for times at sender and receiver

## And many others...

- medical instruments
- hybrid systems (eg for controlling industrial plants)

## 1. Real-time models

- Timed Automata and Hybrid Automata
- Temporal logic
- Static verification using UPPAAL

## 2. Program verification

- First Order Logic revisited
- Abstract Program Semantics
- Design by Contract and Hoare Logic
- Verification of annotated programs

## 3. Requirements

- SAT and SMT solvers
- *Automatic* theorem proving using Z3
- Introduction to *Interactive* theorem proving using Coq

# Table of contents

1. Motivation
2. Timed Automata
3. Semantics
4. Modelling in UPPAAL

- timed transition systems, timed Petri nets, timed IO automata, timed process algebras and other formalisms associate lower and upper bounds to transitions, but no time constraints to transverse the automaton.
- Expressive power is often somehow limited and infinite-state LTS (introduced to express dense time models) are difficult to handle in practice

## Example

Typical process algebra tools are unable to express a system which has one action  $a$  which can only occur at time point 5 with the effect of moving the system to its initial state.

This example has, however, a simple description in terms of time measured by a stopwatch:

1. Set the stopwatch to 0
2. When the stopwatch measures 5, action  $a$  can occur. If  $a$  occurs go to 1., if not idle forever.



# Motivation

This suggests resorting to an **automaton-based formalism** with an explicit notion of **clock** (stopwatch) to control availability of transitions.

## Timed Automata [Alur & Dill, 90]

- emphasis on decidability of the **reachability** problem and corresponding practically efficient algorithms
- infinite underlying timed transition systems are converted to **finitely large** symbolic transition systems where **reachability** becomes decidable (**region** or **zone** graphs)

### Associated tools

- UPPAAL [Behrmann, David, Larsen, 04]
- PRISM [Parker, Kwiatkowska, 00]
- IMITATOR [André, 09]
- KRONOS [Bozga, 98]

UPPAAL = (Uppsala University + Aalborg University) [1995]

- A toolbox for modeling, simulation and verification of real-time systems
- where systems are modeled as networks of timed automata enriched with integer variables, structured data types, channel synchronisations and urgency annotations
- Properties are specified in a subset of CTL

[www.uppaal.org](http://www.uppaal.org)

# Timed Automata

---

Finite-state machine equipped with a finite set of real-valued clock variables (**clocks**)

## Clocks

- **dense-time** model
- clocks can only be **inspected** or
- **reset to zero**, after which they start increasing their value implicitly as time progresses
- the value of a clock corresponds to time elapsed since its last reset
- all clocks proceed synchronously (at the same rate)

# Timed automata Definition

$$\langle L, L_0, \text{Act}, C, \text{Tr}, \text{Inv} \rangle$$

where

- $L$  is a set of **locations**, and  $L_0 \subseteq L$  the set of **initial** locations
- $\text{Act}$  is a set of **actions** and  $C$  a set of **clocks**
- $\text{Tr} \subseteq L \times \mathcal{C}(C) \times \text{Act} \times \mathcal{P}(C) \times L$  is the **transition relation**

$$\ell_1 \xrightarrow{g, a, U} \ell_2$$

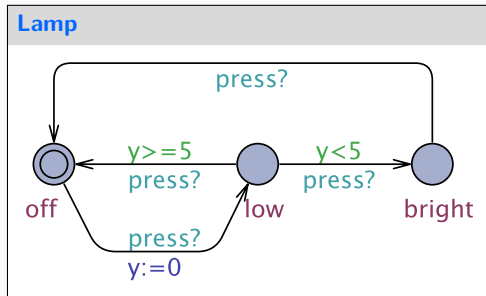
denotes a transition from location  $\ell_1$  to  $\ell_2$ , **labelled** by  $a$ , enabled if **guard**  $g$  is valid, which, when performed, **resets** the set  $U$  of **clocks**

- $\text{Inv} : L \rightarrow \mathcal{C}(C)$  is the assignment of **invariants** to locations

where  $\mathcal{C}(C)$  denotes the set of clock constraints over a set  $C$  of clock variables

## Example: the lamp interrupt

(extracted from UPPAAL)



**Ex. 1.1:** Define  $\langle L, L_0, Act, C, Tr, Inv \rangle$ .

# Clock constraints

$\mathcal{C}(C)$  denotes the set of clock constraints over a set  $C$  of clock variables. Each constraint is formed according to

$$g ::= x \square n \mid x - y \square n \mid g \wedge g \mid \text{true}$$

where  $x, y \in C, n \in \mathbb{N}$  and  $\square \in \{<, \leq, >, \geq, =\}$

used in

- **transitions** as **guards** (enabling conditions)

a transition cannot occur if its guard is invalid

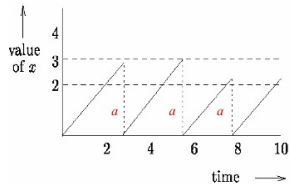
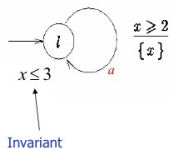
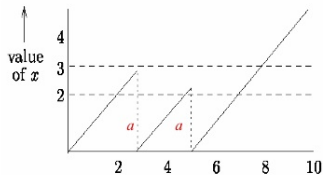
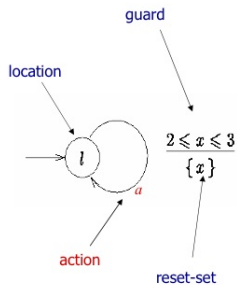
- **locations** as **invariants** (safety specifications)

a location must be left before its invariant becomes invalid

## Note

Invariants are the **only** way to force transitions to occur

# Guards, updates & invariants

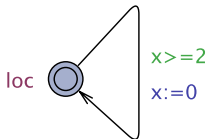




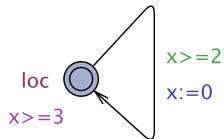
# Transition guards & location invariants

## Demo (in Uppaal)

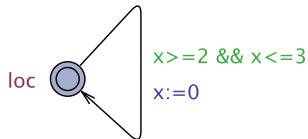
Process 1



Process 2



Process 3



# Parallel composition of timed automata

- Action labels as **channel** identifiers
- Communication by **forced handshaking** over a subset of common actions
- Is defined as an automaton construction over a finite set of timed automata originating a so-called **network** of timed automata

## Parallel composition of timed automata

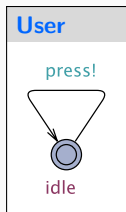
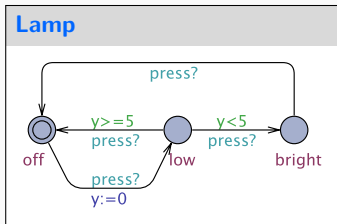
Let  $H \subseteq Act_1 \cap Act_2$ . The parallel composition of  $ta_1$  and  $ta_2$  synchronizing on  $H$  is the timed automata

$$ta_1 \parallel_H ta_2 := \langle L_1 \times L_2, L_{0,1} \times L_{0,2}, Act_{\parallel_H}, C_1 \cup C_2, Tr_{\parallel_H}, Inv_{\parallel_H} \rangle$$

where

- $Act_{\parallel_H} = ((Act_1 \cup Act_2) - H) \cup \{\tau\}$
- $Inv_{\parallel_H} \langle \ell_1, \ell_2 \rangle = Inv_1(\ell_1) \wedge Inv_2(\ell_2)$
- $Tr_{\parallel_H}$  is given by:
  - $\langle \ell_1, \ell_2 \rangle \xrightarrow{g, a, U} \langle \ell'_1, \ell_2 \rangle$  if  $a \notin H \wedge \ell_1 \xrightarrow{g, a, U} \ell'_1$
  - $\langle \ell_1, \ell_2 \rangle \xrightarrow{g, a, U} \langle \ell_1, \ell'_2 \rangle$  if  $a \notin H \wedge \ell_2 \xrightarrow{g, a, U} \ell'_2$
  - $\langle \ell_1, \ell_2 \rangle \xrightarrow{g, \tau, U} \langle \ell'_1, \ell'_2 \rangle$  if  $a \in H \wedge \ell_1 \xrightarrow{g_1, a, U_1} \ell'_1 \wedge \ell_2 \xrightarrow{g_2, a, U_2} \ell'_2$   
with  $g = g_1 \wedge g_2$  and  $U = U_1 \cup U_2$

## Example: the lamp interrupt as a closed system



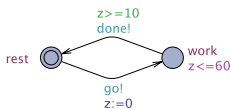
### Uppaal:

- takes  $H = Act_1 \cap Act_2$  (actually as **complementary** actions denoted by the ? and ! annotations)
- only deals with **closed** systems

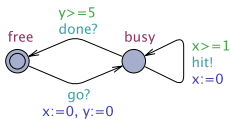
**Ex. 1.2:** Define the TA of the composition.

## Exercise: worker, hammer, nail

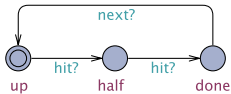
### Worker



### Hammer



### Nail



Ex. 1.3: Define the TA of the composition.

# Semantics

---

# Timed Labelled Transition Systems

Syntax	Semantics
<i>How to write</i>	<i>How to execute</i>
Process Languages	LTS (Labelled Transition Systems)
Timed Automaton	TLTS (Timed LTS)

# Timed Labelled Transition Systems

Syntax	Semantics
<i>How to write</i>	<i>How to execute</i>
Process Languages	LTS (Labelled Transition Systems)
Timed Automaton	TLTS (Timed LTS)

## Timed LTS

Introduce **delay transitions** to capture the passage of time within a LTS:

$s \xrightarrow{a} s'$  for  $a \in Act$ , are ordinary transitions due to action occurrence

$s \xrightarrow{d} s'$  for  $d \in \mathcal{R}_0^+$ , are **delay** transitions

subject to a number of constraints, eg,



## Timed LTS

- time additivity

$$(s \xrightarrow{d} s' \wedge 0 \leq d' \leq d) \Rightarrow s \xrightarrow{d'} s'' \xrightarrow{d-d'} s' \text{ for some state } s''$$

- delay transitions are deterministic

$$(s \xrightarrow{d} s' \wedge s \xrightarrow{d} s'') \Rightarrow s' = s''$$

## Semantics of TA:

Every TA  $ta$  defines a TLTS

$$\mathcal{T}(ta)$$

whose states are pairs

$$\langle \text{location}, \text{clock valuation} \rangle$$

with **infinitely**, even **uncountably** many states, and infinite branching

## Definition

A **clock valuation**  $\eta$  for a set of clocks  $C$  is a function

$$\eta : C \longrightarrow \mathcal{R}_0^+$$

assigning to each clock  $x \in C$  its current value  $\eta x$ .

## Satisfaction of clock constraints

$$\eta \models x \Box n \Leftrightarrow \eta x \Box n$$

$$\eta \models x - y \Box n \Leftrightarrow (\eta x - \eta y) \Box n$$

$$\eta \models g_1 \wedge g_2 \Leftrightarrow \eta \models g_1 \wedge \eta \models g_2$$

# Operations on clock valuations

## Delay

For each  $d \in \mathcal{R}_0^+$ , valuation  $\eta + d$  is given by

$$(\eta + d)x = \eta x + d$$

## Reset

For each  $R \subseteq C$ , valuation  $\eta[R]$  is given by

$$\begin{cases} \eta[R]x = \eta x & \Leftarrow x \notin R \\ \eta[R]x = 0 & \Leftarrow x \in R \end{cases}$$

## From $ta$ to $\mathcal{T}(ta)$

Let  $ta = \langle L, L_0, Act, C, Tr, Inv \rangle$

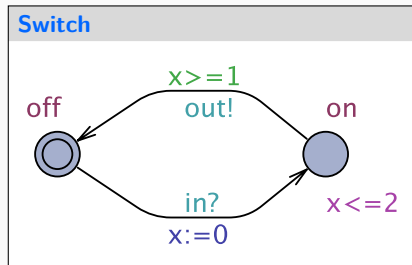
$$\mathcal{T}(ta) = \langle S, S_0 \subseteq S, N, T \rangle$$

where

- $S = \{ \langle l, \eta \rangle \in L \times (\mathcal{R}_0^+)^C \mid \eta \models Inv(l) \}$
- $S_0 = \{ \langle \ell_0, \eta \rangle \mid \ell_0 \in L_0 \wedge \eta x = 0 \text{ for all } x \in C \}$
- $N = Act \cup \mathcal{R}_0^+$  (ie, transitions can be labelled by actions or delays)
- $T \subseteq S \times N \times S$  is given by:

$$\begin{aligned} \langle l, \eta \rangle \xrightarrow{a} \langle l', \eta' \rangle &\Leftarrow \exists_{l \xrightarrow{g, a, U} l' \in Tr} \eta \models g \wedge \eta' = \eta[U] \wedge \eta' \models Inv(l') \\ \langle l, \eta \rangle \xrightarrow{d} \langle l, \eta + d \rangle &\Leftarrow \exists_{d \in \mathcal{R}_0^+} \eta + d \models Inv(l) \end{aligned}$$

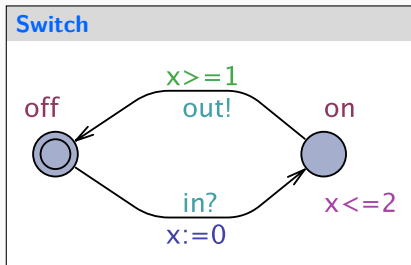
## Example: the simple switch



Ex. 1.4: Define  $\mathcal{T}(\text{SwitchA})$

$S =$

## Example: the simple switch

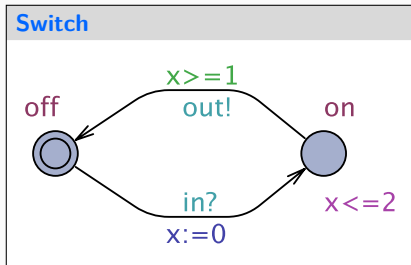


**Ex. 1.4:** Define  $\mathcal{T}(\text{SwitchA})$

$$S = \{\langle \text{off}, \bar{t} \rangle \mid t \in \mathcal{R}_0^+\} \cup \{\langle \text{on}, \bar{t} \rangle \mid 0 \leq t \leq 2\}$$

where  $\bar{t}$  is a shorthand for  $\eta$  such that  $\eta x = t$

## Example: the simple switch

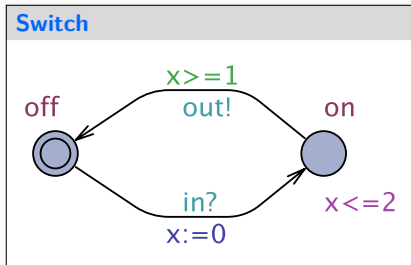


Ex. 1.4: Define  $\mathcal{T}(\text{SwitchA})$

$T = \dots$



## Example: the simple switch



**Ex. 1.4:** Define  $\mathcal{T}(\text{SwitchA})$

$$\langle \text{off}, \bar{t} \rangle \xrightarrow{d} \langle \text{off}, \bar{t} + d \rangle \text{ for all } t, d \geq 0$$

$$\langle \text{off}, \bar{t} \rangle \xrightarrow{\text{in}} \langle \text{on}, \bar{0} \rangle \text{ for all } t \geq 0$$

$$\langle \text{on}, \bar{t} \rangle \xrightarrow{d} \langle \text{on}, \bar{t} + d \rangle \text{ for all } t, d \geq 0 \text{ and } t + d \leq 2$$

$$\langle \text{on}, \bar{t} \rangle \xrightarrow{\text{out}} \langle \text{off}, \bar{t} \rangle \text{ for all } 1 \leq t \leq 2$$

- The elapse of time in timed automata **only** takes place in locations:
- ... actions take place instantaneously
- Thus, several actions may take place at a single time unit

# Behaviours

- Paths in  $\mathcal{T}(ta)$  are discrete representations of continuous-time behaviours in  $ta$
- ... at least they indicate the states immediately before and after the execution of an action
- However, as interval delays may be realised in uncountably many different ways, different paths may represent the same behaviour

# Behaviours

- Paths in  $\mathcal{T}(ta)$  are **discrete representations of continuous-time behaviours** in  $ta$
- ... at least they indicate the states immediately before and after the execution of an action
- However, as interval delays may be realised in **uncountably** many different ways, different paths may represent the same behaviour
- ... but not all paths correspond to valid (**realistic**) behaviours:

## undesirable paths:

- **time-convergent** paths
- **timelock** paths
- **zeno** paths

# Time-convergent paths

$$\langle l, \eta \rangle \xrightarrow{d_1} \langle l, \eta + d_1 \rangle \xrightarrow{d_2} \langle l, \eta + d_1 + d_2 \rangle \xrightarrow{d_3} \langle l, \eta + d_1 + d_2 + d_3 \rangle \xrightarrow{d_4} \dots$$

such that

$$\forall i \in \mathbb{N}. d_i > 0 \wedge \sum_{i \in \mathbb{N}} d_i = d$$

ie, the infinite sequence of delays converges toward  $d$

- Time-convergent paths are **counterintuitive**; as their existence cannot be avoided, they are simply **ignored** in the semantics of Timed Automata
- Time-**divergent** paths are the ones in which time always progresses

## Definition

An infinite path fragment  $\rho$  is **time-divergent** if  $\text{ExecTime}(\rho) = \infty$   
Otherwise is **time-convergent**.

where

$$\begin{aligned}\text{ExecTime}(\rho) &= \sum_{i=0.. \infty} \text{ExecTime}(\delta_i) \\ \text{ExecTime}(\delta) &= \begin{cases} 0 & \Leftarrow \delta \in \text{Act} \\ \delta & \Leftarrow \delta \in \mathcal{R}_0^+ \end{cases}\end{aligned}$$

for  $\rho$  a path and  $\delta$  a label in  $\mathcal{T}(ta)$

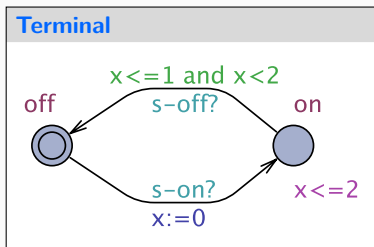
## Definition

A path is **timelock** if it contains a state with a timelock, ie, a **state from which there is not any time-divergent path**

A **timelock** represents a situation that causes time progress to halt (e.g. when it is impossible to leave a location before its invariant becomes invalid)

- any **teminal state** ( $\neq$  terminal location) in  $\mathcal{T}(ta)$  contains a timelock
- ... but not all timelocks arise as terminal states in  $\mathcal{T}(ta)$

# Timelock paths



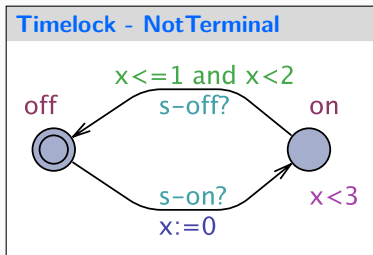
State  $\langle on, 2 \rangle$  is reachable through path

$$\langle off, 0 \rangle \xrightarrow{s-on} \langle on, 0 \rangle \xrightarrow{2} \langle on, 2 \rangle$$

and is terminal



# Timelock paths



State  $\langle on, 2 \rangle$  is not terminal but has a **convergent** path:

$\langle on, 2 \rangle \langle on, 2.9 \rangle \langle on, 2.99 \rangle \langle on, 2.999 \rangle \dots$

## In a Timed Automaton

- The elapse of time only takes place at **locations**
- Actions occur **instantaneously**: at a single time instant several actions may take place

... it may perform **infinitely** many actions in a **finite** time interval  
(non realizable because it would require infinitely fast processors)

## In a Timed Automaton

- The elapse of time only takes place at **locations**
- Actions occur **instantaneously**: at a single time instant several actions may take place

... it may perform **infinitely** many actions in a **finite** time interval  
(non realizable because it would require infinitely fast processors)

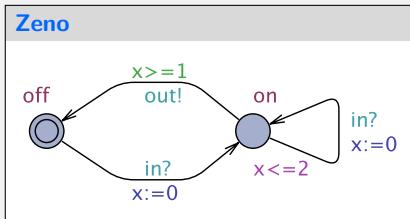
## Definition

An infinite path fragment  $\rho$  is **zeno** if it is **time-convergent** and **infinitely many actions occur along it**

A timed automaton  $ta$  is **non-zeno** if there is not an initial zeno path in  $\mathcal{T}(ta)$

## Example

Suppose the user can press the *in* button when the light is *on* in



In doing so clock  $x$  is reset to 0 and light stays *on* for more 2 time units (unless the button is pushed again ...)

## Example

Typical paths: The user presses *in* infinitely fast:

$$\langle \text{off}, 0 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{\text{in}} \dots$$

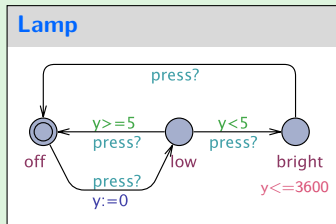
The user presses *in* faster and faster:

$$\langle \text{off}, 0 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{0.5} \langle \text{on}, 0.5 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{0.25} \langle \text{on}, 0.25 \rangle \xrightarrow{\text{in}} \langle \text{on}, 0 \rangle \xrightarrow{0.125} \dots$$

How can this be fixed?

**Time shall pass!**

## Ex. 1.5: Recall our lamp



1. Describe a **time-divergent** path, if it exists.
2. Describe a **time-convergent** path, if it exists.
3. Describe a **timelock** path, if it exists.
4. Is this automata **non-zeno**? Justify.

## Sufficient criterion for nonzenoness

A timed automaton is nonzeno if on any of its control cycles time advances with at least some **constant amount** ( $\geq 0$ ). Formally, if for every control cycle

$$\ell_0 \xrightarrow{g_0, a_0, U_0} \ell_1 \xrightarrow{g_1, a_1, U_1} \dots \xrightarrow{g_n, a_n, U_n} \ell_0$$

there exists a clock  $x \in C$  such that

1.  $x \in U_i$  (for  $0 \leq i \leq n$ )
2. for all clock valuations  $\eta$ , there is a  $c \in \mathbb{N}_{>0}$  such that

$$\eta(x) < c \Rightarrow ((\eta \not\models g_j) \vee \neg \text{Inv}(\ell_j)) \text{ for some } 0 \leq j \leq n$$

Both

- timelocks
- zenoness

are **modelling flaws** and need to be avoided.

## Example

In the example above, it is enough to impose a non zero minimal delay between successive button pushings.



# Modelling in Uppaal

---

... an editor, simulator and model-checker for TA with extensions ...

## Editor.

- Templates and instantiations
- Global and local declarations
- System definition

## Simulator.

- Viewers: automata animator and message sequence chart
- Control (eg, trace management)
- Variable view: shows values of the integer variables and the clock constraints defining symbolic states

## Verifier.

- (see next session)

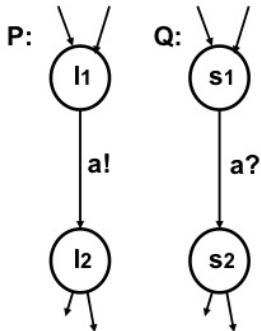
## Extensions (modelling view)

- templates with parameters and an instantiation mechanism
- data expressions over bounded integer variables (eg, `int[2..45] x`) allowed in guards, assignments and invariants
- rich set of operators over integer and booleans, including bitwise operations, arrays, initializers ... in general a whole subset of C is available
- non-standard types of synchronization
- non-standard types of locations

## Extension: broadcast synchronization

- A sender can synchronize with an arbitrary number of receivers
- Any receiver that can synchronize in the current state must do so
- Broadcast sending is never blocking (the send action can occur even with no receivers).

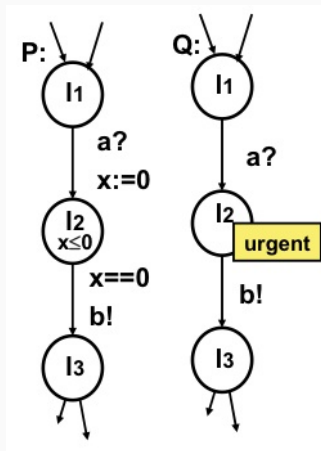
## Extension: urgent synchronization



Channel  $a$  is declared **urgent chan a** if both edges are to be taken as soon as they are ready (**simultaneously** in locations  $l_1$  and  $s_1$ ). Note the problem can **not** be solved with **invariants** because locations  $l_1$  and  $s_1$  can be reached at different moments

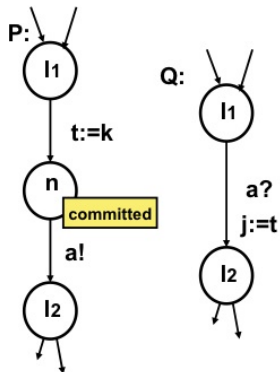
- No delay allowed if a synchronization transition on an urgent channel is enabled
- Edges using urgent channels for synchronization cannot have time constraints (ie, clock guards)

## Extension: urgent location



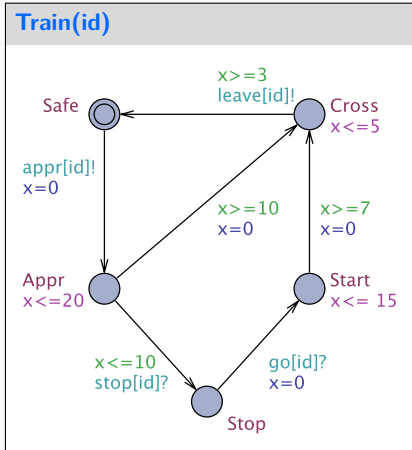
- Time does not progress but interleaving with normal location is allowed
- Both models are equivalent: no delay at an urgent location
- but the use of urgent location reduces the number of clocks in a model and simplifies analysis

## Extension: committed location



- delay is not allowed and the committed transition must be left in the next instant (or one of them if there are several), i.e., next transition must involve an outgoing edge of at least one of the committed locations
- Our aim is to pass the value  $k$  to variable  $j$  (via global variable  $t$ )
- Location  $n$  is committed to ensure that no other automata can assign  $j$  before the assignment  $j := t$

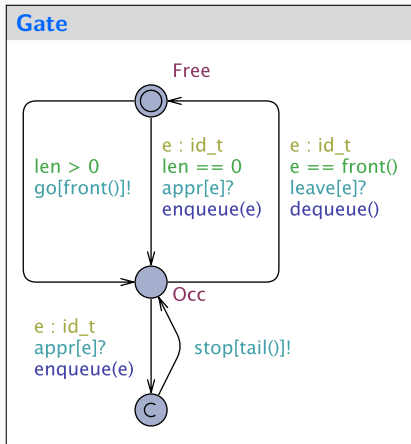
# The train gate example



- Events model approach/leave, order to stop/go
- A train can not be stopped or restart instantly
- After approaching it has 10m to receive a stop.
- After that it takes further 10 time units to reach the bridge
- After restarting takes 7 to 15m to reach the cross and 3-5 to cross



# The train gate example



- Note the use of parameters and the select clause on transitions
- Programming ...