Time-critical reactive systems (I)

Luís S. Barbosa

DI-CCTC
Universidade do Minho
Braga, Portugal

April, 2010

Motivation

Specifying an airbag saying that in a car crash the airbag eventually inflates

- in μ-calculus: $\nu Y\,.\,[\mathit{crash}](\mu X\,.\,[-\mathit{airbag}]X \wedge \langle - \rangle \mathrm{true}) \;\wedge\; [-]Y$
- in CTL: $\forall\Box(\mathit{crash} \Rightarrow \forall\Diamond \mathit{airbag})$ or $\mathrm{AG}(\mathit{crash} \Rightarrow \mathrm{AF}\mathit{airbag})$
- ...

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]

Motivation

Specifying an airbag saying that in a car crash the airbag eventually inflates

- in μ-calculus: $\nu Y \,.\, [crash](\mu X \,.\, [-airbag]X \wedge \langle - \rangle \text{true}) \;\wedge\; [-]Y$
- in CTL: $\forall\Box(crash \Rightarrow \forall\Diamond airbag)$ or $\text{AG}(crash \Rightarrow \text{AF}\, airbag)$
- ...

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)
- ...

Dealing with time in system models

Timed LTS

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

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

$s' \xleftarrow{d} s$ for $d \in \mathbb{R}^+$, are delay transitions

subject to a number of constraints, eg,

Dealing with time in system models

Timed LTS

- time additivity

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

- delay transitions are deterministic

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

- a state can only reach itself without delay

$$s \xleftarrow{0} s \text{ for all states } s$$

Dealing with time in system models

Extension of Process Algebras with time

- TCCS [Yi,90] which introduced a new prefix:

  $\epsilon(d).E$ delay d units of time and then behave as E

- TCSP [Reed& Roscoe, 88], ATP [Nicollin & Sifakis, 94], among many others

Emphasis on axiomatics, behavioural equivalences, expressivity

Dealing with time in system models

However, in general, expressive power is somehow limited and infinite-state LTS difficult to handle in practice

Example

TCCS is unable to express a system which has only 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.

Dealing with time in system models

However, in general, expressive power is somehow limited and infinite-state LTS difficult to handle in practice

Example

TCCS is unable to express a system which has only 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.

Dealing with time in system models

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 model-checking problem and corresponding practically efficient algorithms

Associate tools

- UPPAAL [Behrmann, David, Larsen, 04]
- KRONOS [Bozga, 98]

Timed automata

Program graph equipped with a finite set of real-valued clock variables (clocks)

Clocks

- 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 at the same rate

Timed automata

Definition

$$\langle L, L_0, Act, C, Tr, Inv \rangle$$

where

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

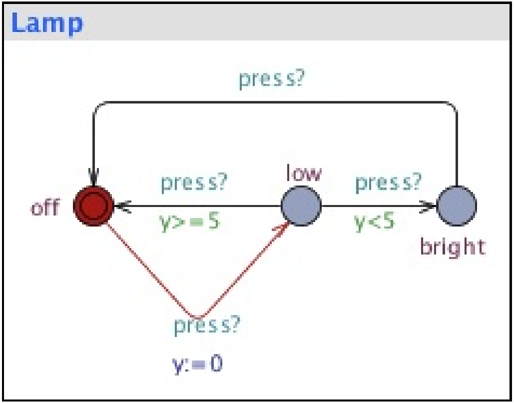
  $$l_1 \overset{g,a,U}{\longleftarrow} l_2$$

  denotes a transition from location l_1 to l_2, labelled by a, enabled if guard g is valid, which, when performed, resets the set U of clocks
- $Inv : \mathcal{C}(C) \longleftarrow L$ is the invariant assignment function

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

Example: the lamp interrupt

(extracted from UPPAAL)



Clock constraints

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

$$g \; ::= \; x \,\square\, n \;\mid\; x - y \,\square\, n \;\mid\; g \wedge g$$

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

Clock constraints

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

$$g \ ::= \ x \square n \ | \ x - y \square n \ | \ g \wedge g$$

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

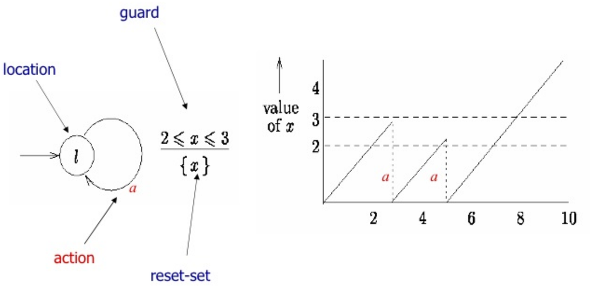
- locations as invariants (safety specifications)

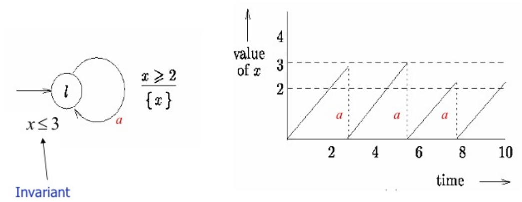
  a location must be left before its invariant becomes invalid

Note

Invariants are the only way to force transitions to occur

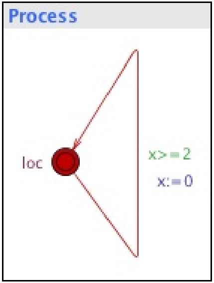
Guards, updates & invariants

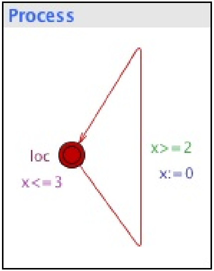


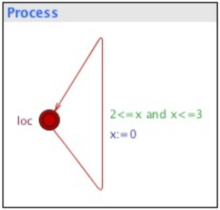


Transition guards & location invariants

Demo (in UPPAAL)







Parallel composition of timed automata

- Action labels as channel identifiers
- Communication by forced handshacking over a subset of common actions
- Can be defined as an associative binary operator (as in the tradition of process algebra) or as an automaton construction over a finite set of timed automata originating a so-called network of timed automata

Parallel composition of timed automata

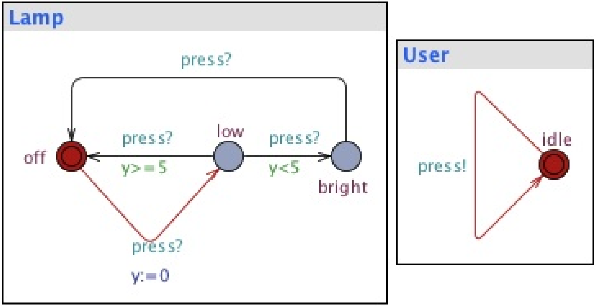
Let $H \subseteq Act_1 \cap Act_2$. The parallel composition of ta_1 and ta_2 synchronizing on H is the timed automata

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

where

- $Act_{\parallel_H} = ((Act_1 \cup Act_2) - H) \cup \{\tau\}$
- $Inv_{\parallel_H} \langle l_1, l_2 \rangle = Inv_1(l_1) \wedge Inv_2(l_2)$
- $Tr_{\parallel_H}$ is given by:
  - $\langle l_1', l_2 \rangle \xleftarrow{g,a,U} \langle l_1, l_2 \rangle$ if $a \notin H \wedge l_1' \xleftarrow{g,a,U} l_1$
  - $\langle l_1, l_2' \rangle \xleftarrow{g,a,U} \langle l_1, l_2 \rangle$ if $a \notin H \wedge l_2' \xleftarrow{g,a,U} l_2$
  - $\langle l_1', l_2' \rangle \xleftarrow{g,\tau,U} \langle l_1, l_2 \rangle$ if $a \in H \wedge l_1' \xleftarrow{g_1,a,U_1} l_1 \wedge l_2' \xleftarrow{g_2,a,U_2} l_2$ with $g = g_1 \wedge g_2$ and $U = U_1 \cup U_2$

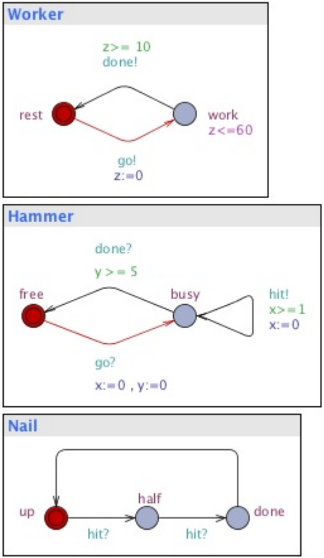
Example: the lamp interrupt as a closed system



UPPAAL:

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

Example: worker, hammer, nail



Semantics

Syntax		Semantics
Process Languages (eg CCS) Timed Automaton		LTS (Labelled Transition Systems) TLTS (Timed LTS)

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

Semantics

Syntax		Semantics
Process Languages (eg CCS) Timed Automaton		LTS (Labelled Transition Systems) TLTS (Timed LTS)

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

Clock valuations

Definition

A clock valuation η for a set of clocks C is a function

$$\eta : \mathbb{R}_0^+ \longleftarrow C$$

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

Satisfaction of clock constraints

$$\begin{aligned} \eta \models x \,\square\, n \;&\Leftrightarrow\; \eta\, x \,\square\, n \\ \eta \models x - y \,\square\, n \;&\Leftrightarrow\; (\eta\, x - \eta\, y) \,\square\, n \\ \eta \models g_1 \wedge g_2 \;&\Leftrightarrow\; \eta \models g_1 \wedge \eta \models g_2 \end{aligned}$$

Operations on clock valuations

Delay

For each $d \in \mathbb{R}_0^+$, 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 $ta = \langle L, L_0, Act, C, Tr, Inv \rangle$

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

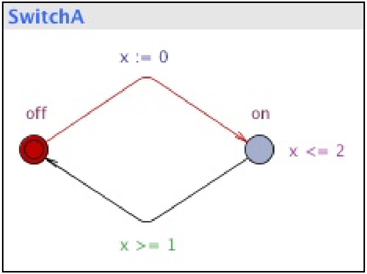
where

- $S = \{\langle l, \eta \rangle \in L \times (\mathbb{R}_0^+)^C \mid \eta \models Inv(l)\}$
- $S_0 = \{\langle l_0, \eta \rangle \mid l_0 \in L_0 \;\wedge\; \eta\, x = 0 \text{ for all } x \in C\}$
- $N = Act \cup \mathbb{R}_0^+$ (ie, transitions can be labelled by actions or delays)
- $T \subseteq S \times N \times S$ is given by:

$$\langle l', \eta' \rangle \xleftarrow{a} \langle l, \eta \rangle \;\Leftarrow\; \exists_{l' \xleftarrow{g,a,U} l \in Tr} .\; \eta \models g \;\wedge\; \eta' = \eta[U] \;\wedge\; \eta' \models Inv(l')$$

$$\langle l, \eta + d \rangle \xleftarrow{d} \langle l, \eta \rangle \;\Leftarrow\; \exists_{d \in \mathbb{R}_0^+} .\; \eta \models Inv(l) \;\wedge\; \eta + d \models Inv(l')$$

Example: the simple switch

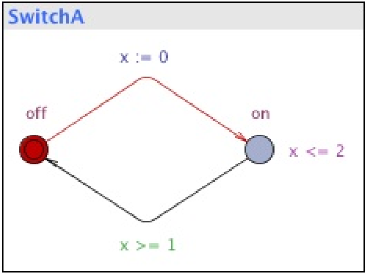


$\mathcal{T}$(SwitchA)

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

where t is a shothand for η such that $\eta\, x = t$

Example: the simple switch



$\mathcal{T}$(SwitchA)

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

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

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

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

Behaviours

- Paths in $\mathcal{T}(ta)$ are discrete representations of behaviours in *ta*
- Such paths can also be represented graphically through location diagrams
- However, as interval delays may be realized 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

Behaviours

- Paths in $\mathcal{T}(ta)$ are discrete representations of behaviours in *ta*
- Such paths can also be represented graphically through location diagrams
- However, as interval delays may be realized 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

$$\cdots \xleftarrow{d_4} \langle l, \eta + d_1 + d_2 + d_3 \rangle \xleftarrow{d_3} \langle l, \eta + d_1 + d_2 \rangle \xleftarrow{d_2} \langle l, \eta + d_1 \rangle \xleftarrow{d_1} \langle l, \eta \rangle$$

such that

$$\forall_{i \in N} \cdot , d_i > 0 \ \wedge \ \sum_{i \in N} d_i = d$$

ie, the infinite sequence of delays converges toward d

- Time-convergent path are conterintuitive and a ignored in the semantics of Timed Automata
- Time-divergent paths are the ones in which time always progresses

Time-convergent paths

Definition

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

where

$$\text{ExecTime}(\rho) = \sum_{i=0..\infty} \text{ExecTime}(\delta)$$

$$\text{ExecTime}(\delta) = \begin{cases} 0 & \Leftarrow \delta \in Act \\ d & \Leftarrow \delta \in \mathbb{R}_0^+ \end{cases}$$

for ρ a path and δ a label in $\mathcal{T}(ta)$

Timelock paths

Definition

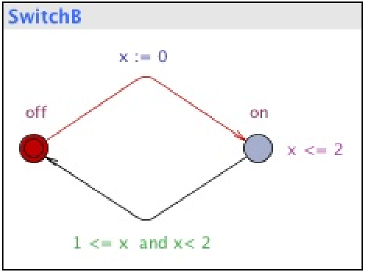
A path is timelock if it contains a state with a time lock, ie, a state from which there is not any time-divergent path

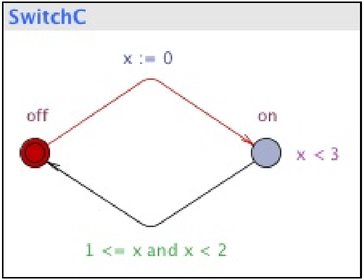
Note

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

Exercise

Identify two different types of timelocks in the following switch specifications:





Zeno

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 ρ 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 $\mathcal{T}(ta)$

Zeno

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

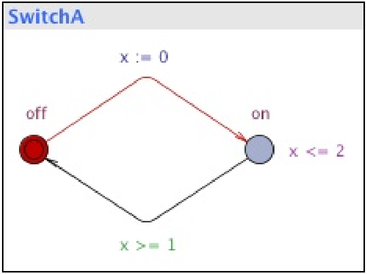
An infinite path fragment ρ is zeno if it is time-convergent and infinitely many actions occur along it

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

Zeno

Example

Suppose the user can press the *in* button when the light in *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 ...)

Zeno

Example

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

$$\cdots \xleftarrow{in} \langle on, 0\rangle \xleftarrow{in} \langle on, 0\rangle \xleftarrow{in} \langle on, 0\rangle \xleftarrow{in} \langle off, 0\rangle$$

The user presses *in* faster and faster:

$$\cdots \xleftarrow{0.125} \langle on, 0\rangle \xleftarrow{0.25} \langle on, 0\rangle \xleftarrow{0.5} \langle on, 0\rangle \xleftarrow{in} \langle off, 0\rangle$$

How can this be fixed?

Zeno

Example

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

$$\cdots \xleftarrow{in} \langle on, 0\rangle \xleftarrow{in} \langle on, 0\rangle \xleftarrow{in} \langle on, 0\rangle \xleftarrow{in} \langle off, 0\rangle$$

The user presses *in* faster and faster:

$$\cdots \xleftarrow{0.125} \langle on, 0\rangle \xleftarrow{0.25} \langle on, 0\rangle \xleftarrow{0.5} \langle on, 0\rangle \xleftarrow{in} \langle off, 0\rangle$$

How can this be fixed?

Zeno

Sufficient criterion for nonzenoness

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

$$l_n \xleftarrow{g_n, a_n, U_n} \ldots \xleftarrow{g_2, a_2, U_2} l_1 \xleftarrow{g_0, a_0, U_0} l_0$$

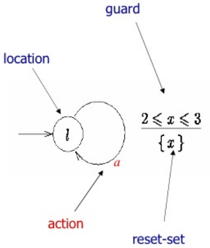
with $l_0 = l_n$,

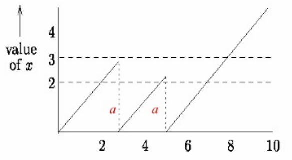
1. there exists a clock $x \in C$ such that $x \in U_i$ (for $0 \leq i \leq n$)
2. for all clock valuations η, there is a $c \in \mathbb{N}_{>0}$ such that

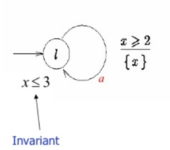
$$\eta\, x < 0 \;\Rightarrow\; (\eta \not\models g_j \;\vee\; Inv(l_j)) \text{ for some } 0 < j \leq n$$

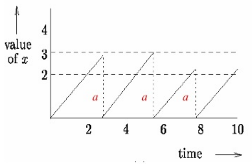
UPPAAL

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









Extensions (modelling view)

- templates with parameters and an instantiation mechanism
- data expressions over bounded integer variables (eg, `int[2..45] x`) allowed in guards, assigments 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

The toolkit

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

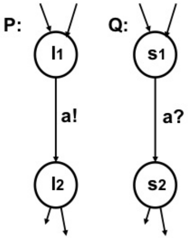
Verifier.

- `(see next session)`

Extension: broadcast synchronization

- A sender can synchronize with an arbitrary number of receivers
- Any receiver than can synchronize in the current state must do so
- Broadcast sending is never blocking.

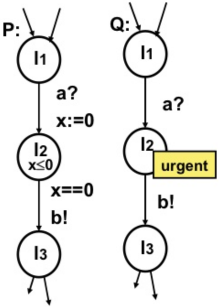
Extension: urgent synchronization



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

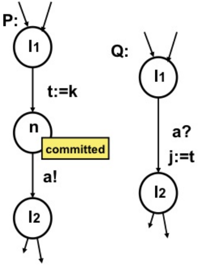
- No delay allowed if a synchronization transition on an urgent channel is enabled
- Edges using urgent channels for synchronization cannot have time constraints (ie, clock guards)

Extension: urgent location



- 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



- 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 $j := t$
- In general, a committed state cannot delay and next transition must involve an outgoing edge of at least one of the committed locations

Hints

- Modelling patterns: see the UPPAAL tutorial
- Further examples: see the `demo` folder in the standard distribution