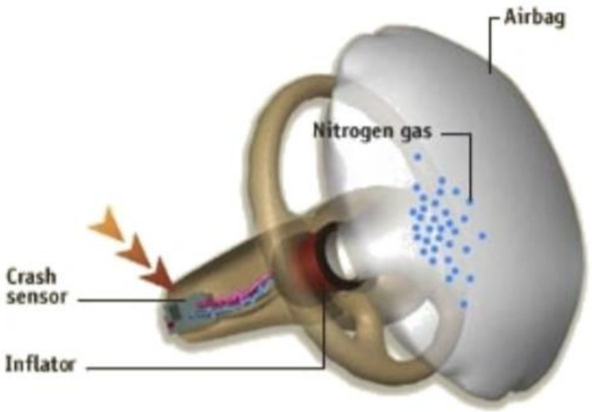
Model Checking with UPPAAL

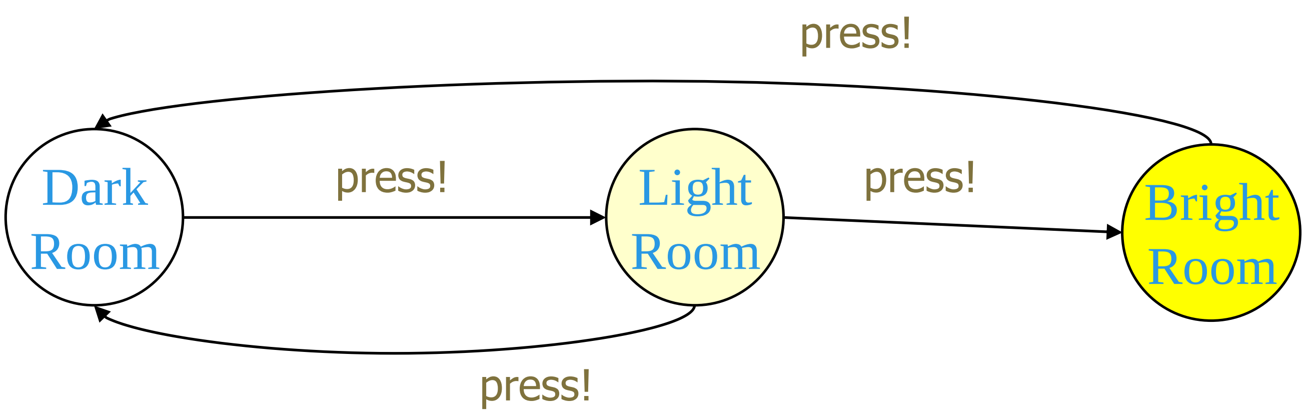
Jin Hyun Kim

Timed Correctness



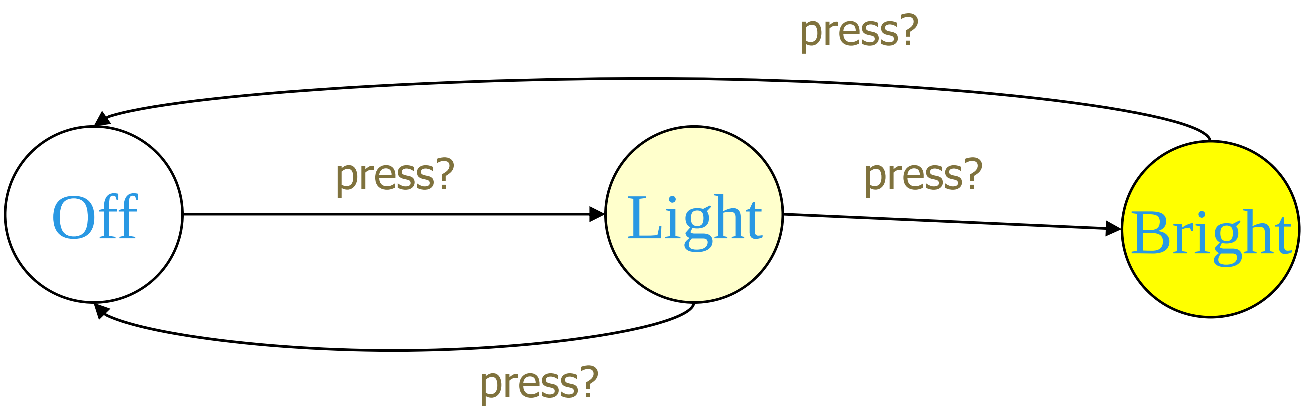
- "Will the airbag open within 5ms after the car crashes?"
- "Will the robot explore a given area without getting out of energy?"

A User



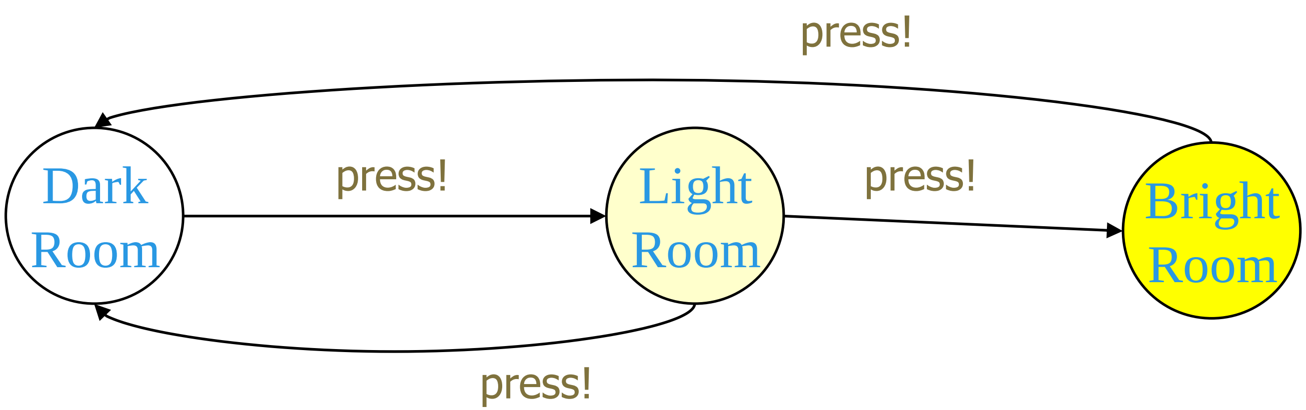
Requirement: if press is issued twice,
then the light will get brighter; otherwise the light is turned off.

A Light Controller



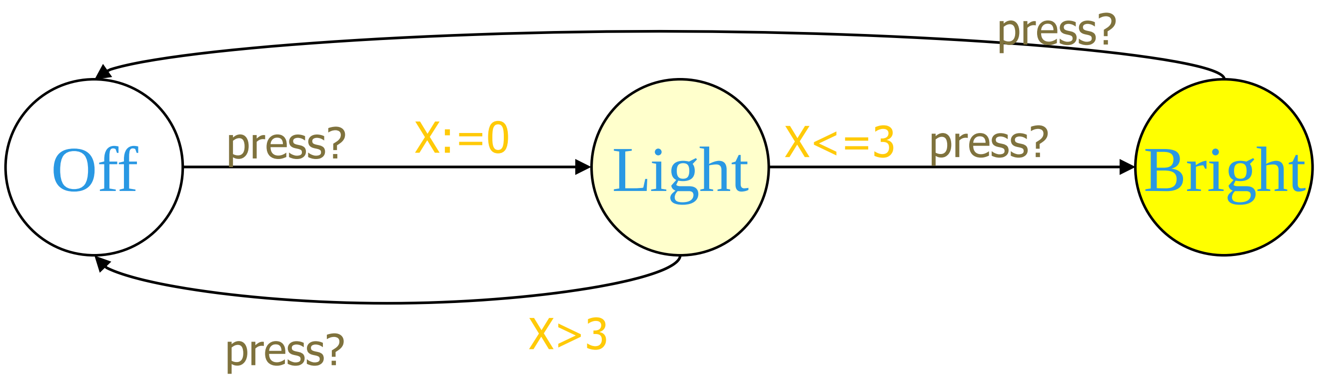
Requirement: if press is issued twice,
then the light will get brighter; otherwise the light is turned off.

A (revised) User Requirement



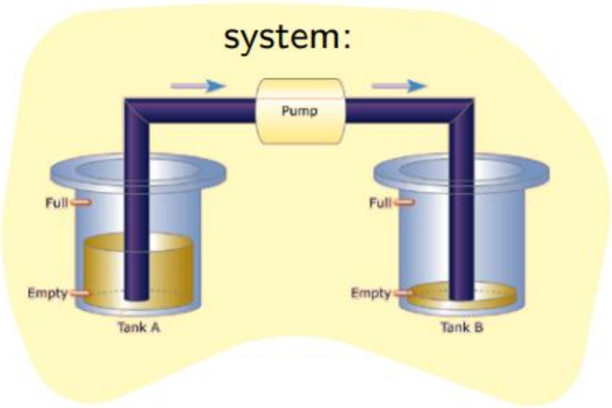
Requirement: if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.

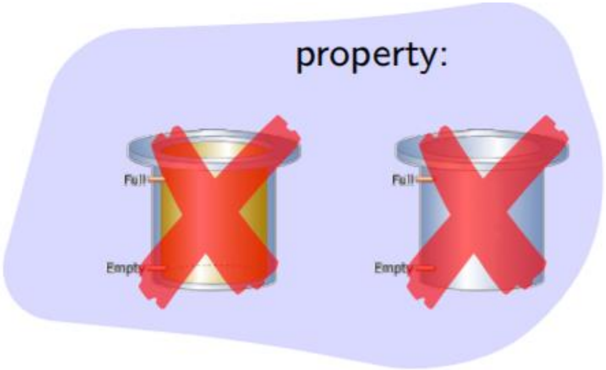
A Light Controller (with timer)



Solution: Add real-valued clock x

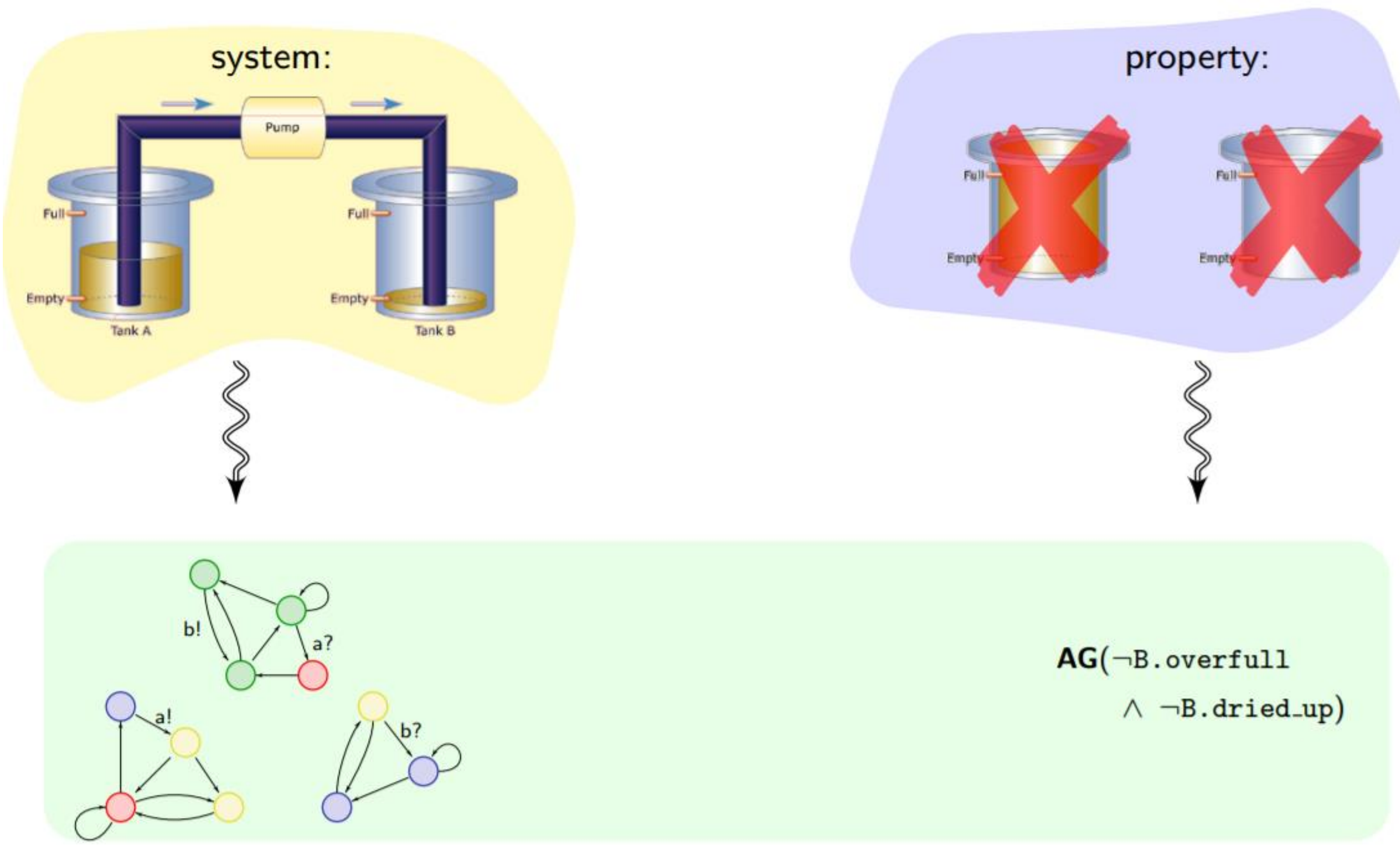
Model Checking





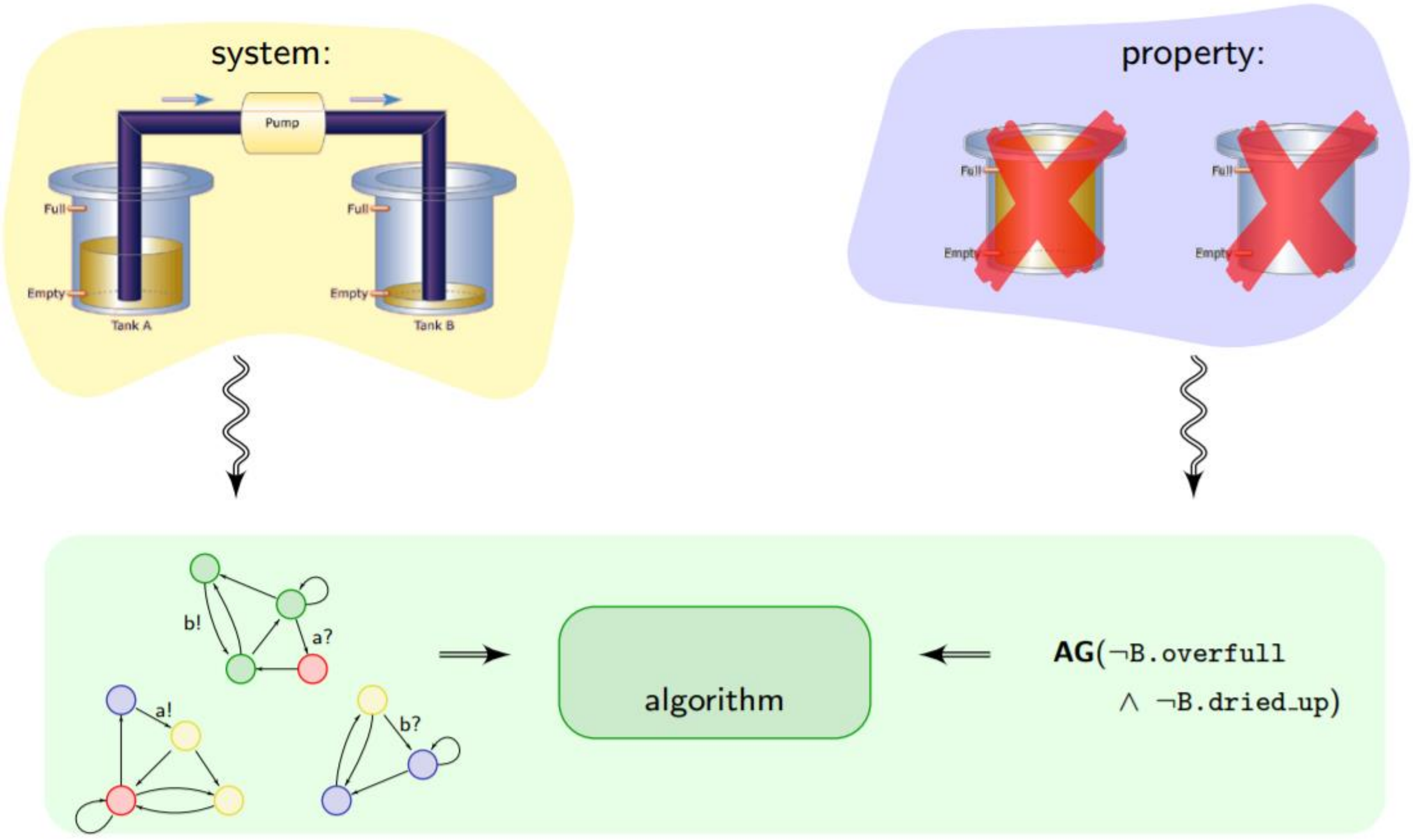
http://www.lsv.fr/~bouyer/files/movep14.pdf

Model Checking



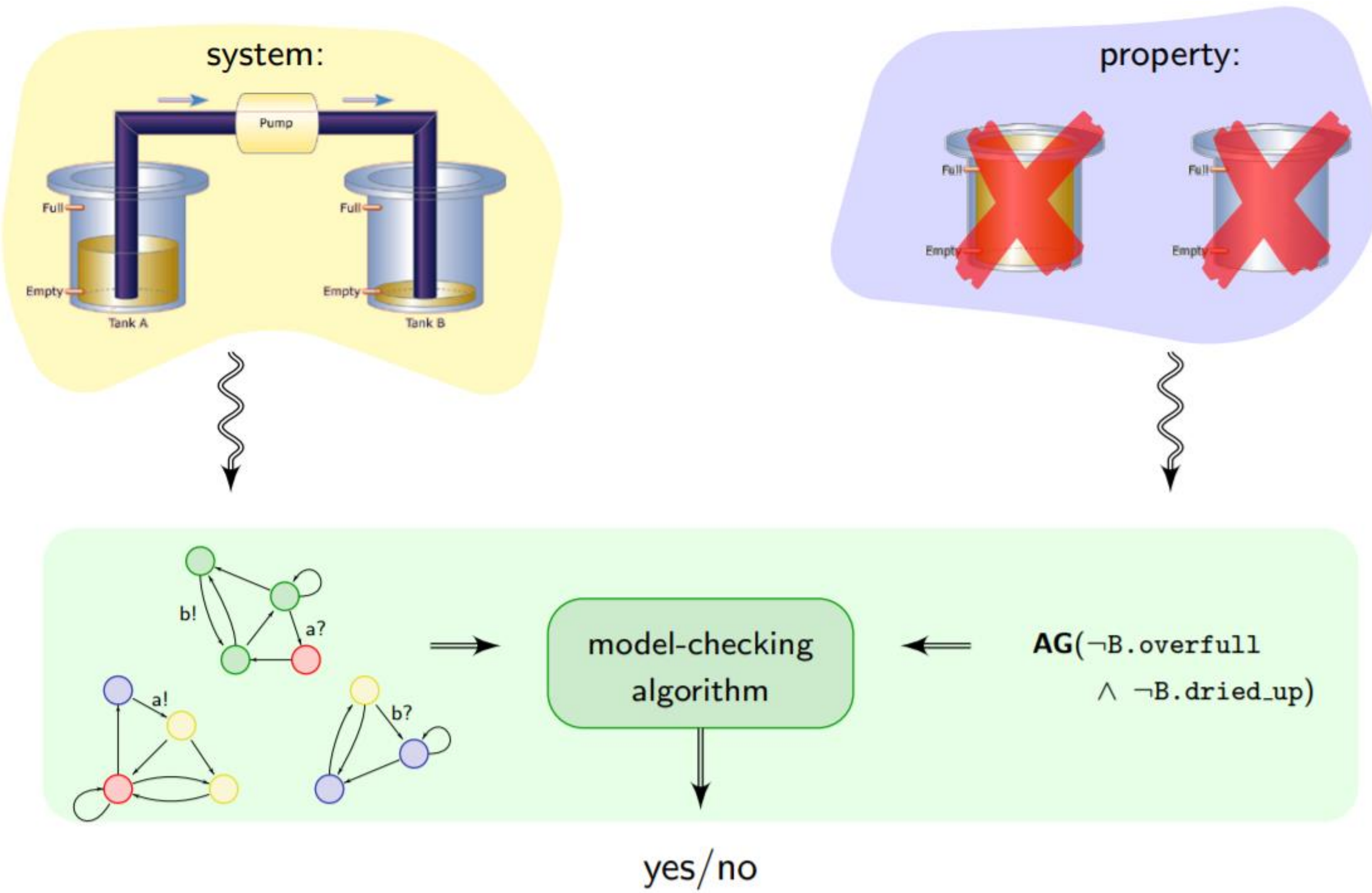
http://www.lsv.fr/~bouyer/files/movep14.pdf

Model Checking



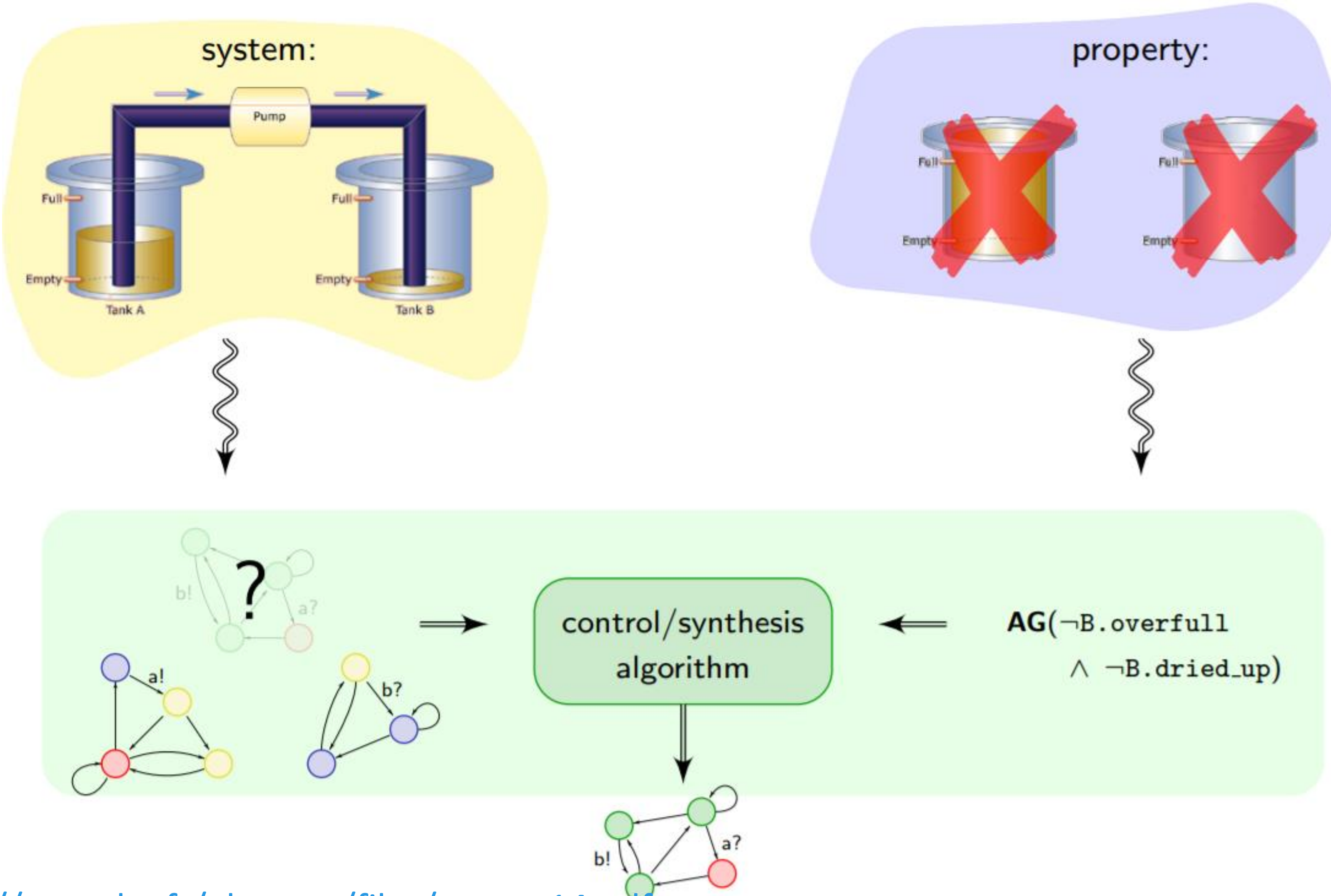
http://www.lsv.fr/~bouyer/files/movep14.pdf

Model Checking



http://www.lsv.fr/~bouyer/files/movep14.pdf

Model Checking



http://www.lsv.fr/~bouyer/files/movep14.pdf

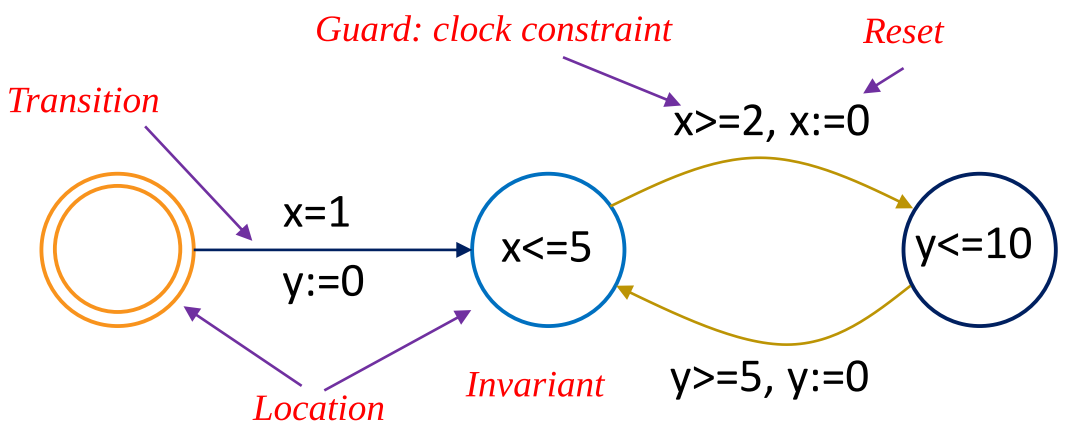
Model Checking of Timed Systems

- For a given system model, model checking checks if any timing constraints (requirements) are always satisfied by the system model

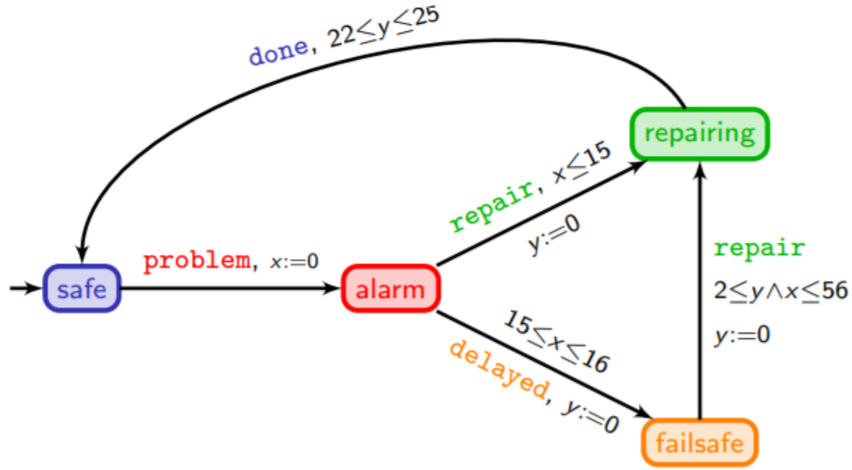
Timed Automata

Timed Automata

- In automata theory, a **timed automaton** is a finite automaton extended with a finite set of real-valued clocks.



Timed Automata



valuation

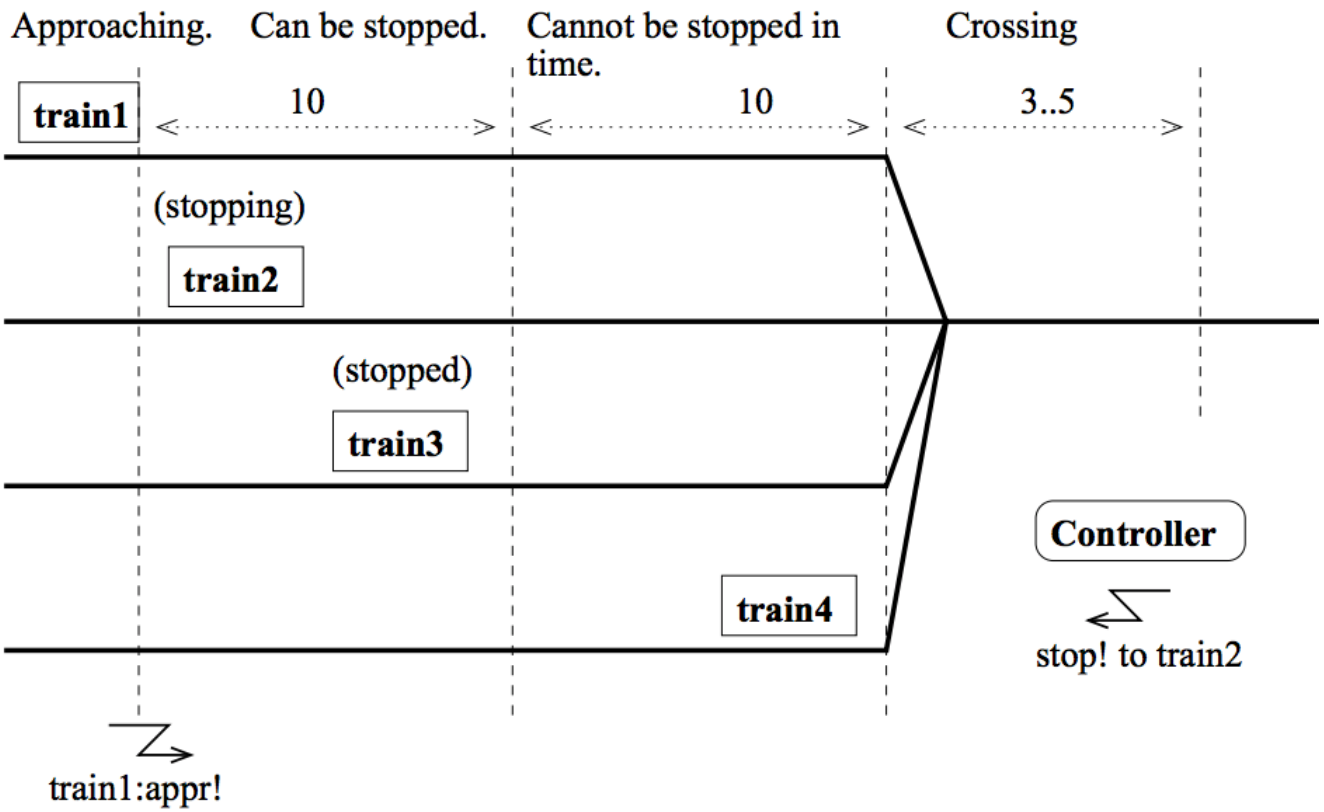
	safe	$\xrightarrow{23}$	safe	$\xrightarrow{\texttt{problem}}$	alarm	$\xrightarrow{15.6}$	alarm	$\xrightarrow{\texttt{delayed}}$	failsafe	
x	0		23		0		15.6		15.6	$\cdots$
y	0		23		23		38.6		0	

	failsafe	$\xrightarrow{2.3}$	failsafe	$\xrightarrow{\texttt{repair}}$	repairing	$\xrightarrow{22.1}$	repairing	$\xrightarrow{\texttt{done}}$	safe
$\cdots$	15.6		17.9		17.9		40		40
	0		2.3		0		22.1		22.1

This run reads the timed word
(problem, 23)(delayed, 38.6)(repair, 40.9)(done, 63).

http://www.lsv.fr/~bouyer/files/movep14.pdf

Running Example: The Train Gate [1]



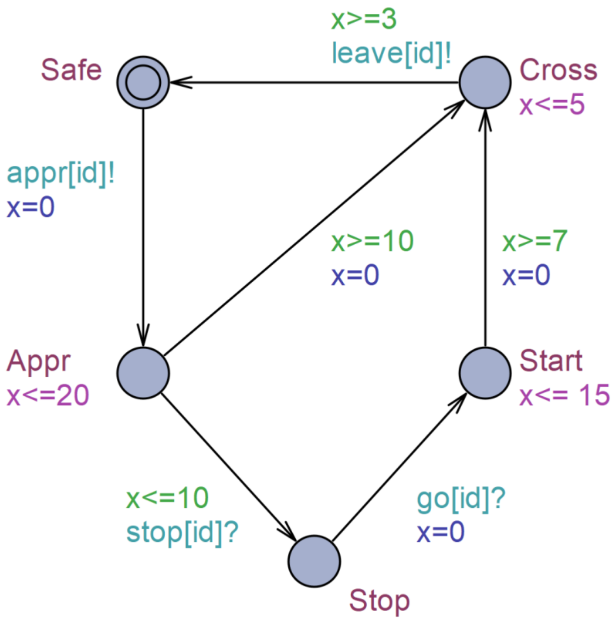
[1] Behrmann, Gerd, Alexandre David, and Kim Larsen. "A tutorial on uppaal." *Formal methods for the design of real-time s ystems* (2004): 33-35.

Examples: Train Gate Systems

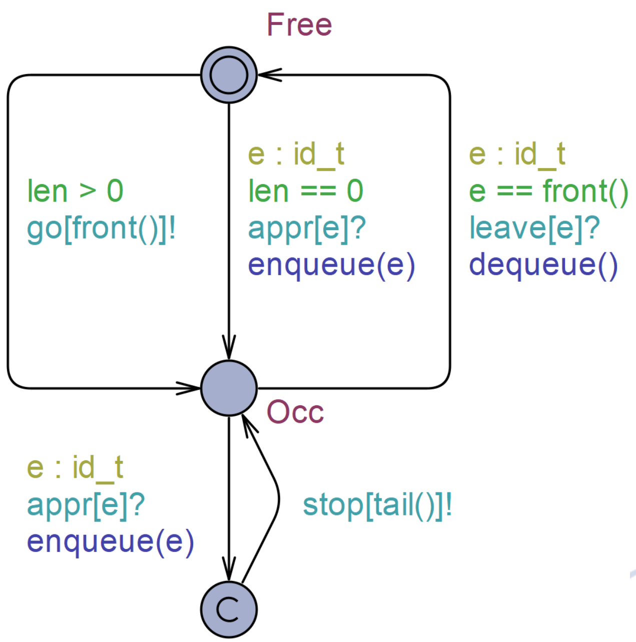
Systems

system Train, Gate;

Train

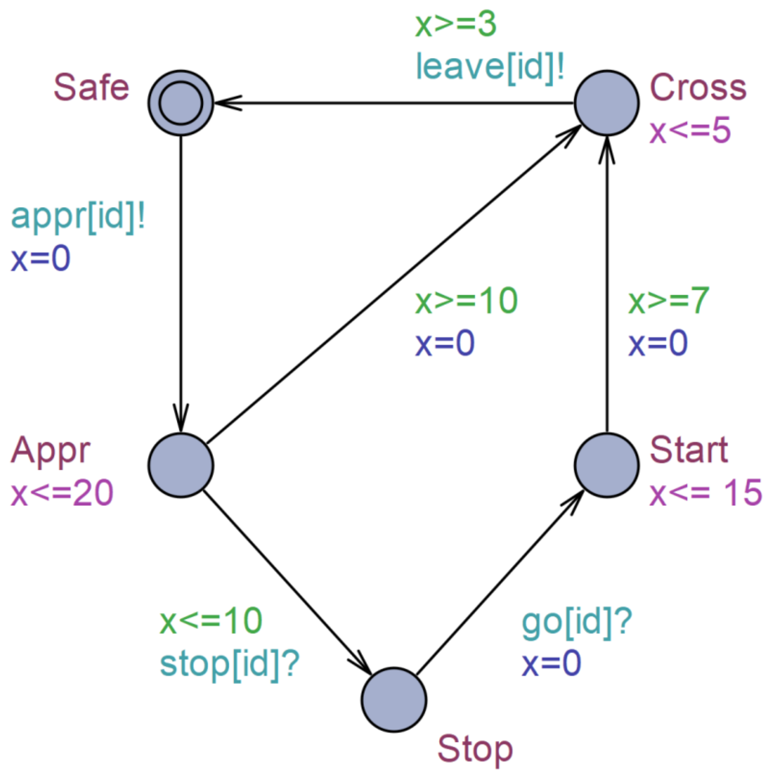


Gate Control



Examples: Train Gate Systems

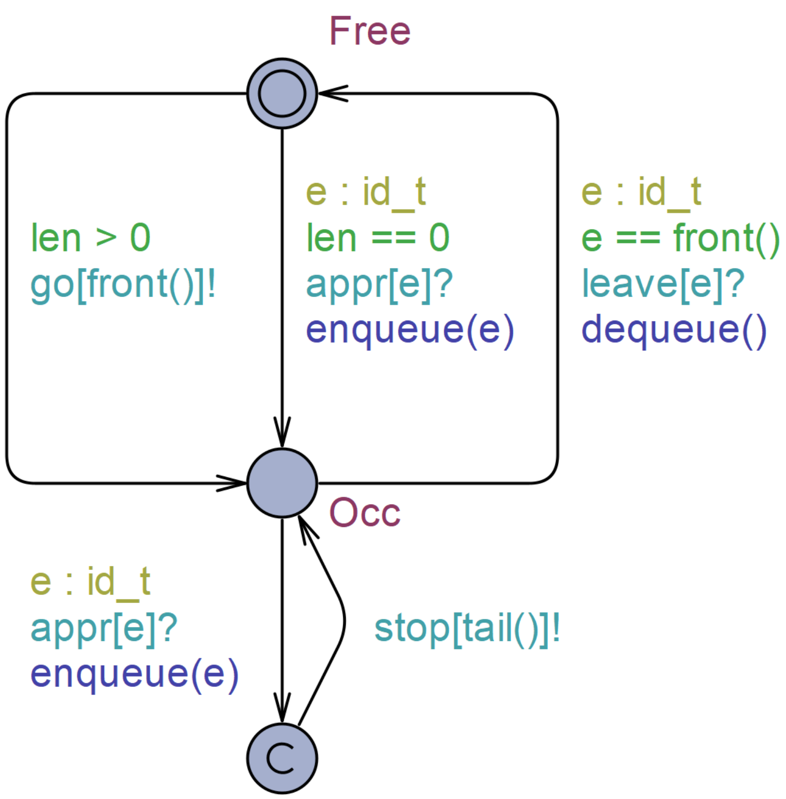
Train



clock x;

Examples: Train Gate Systems

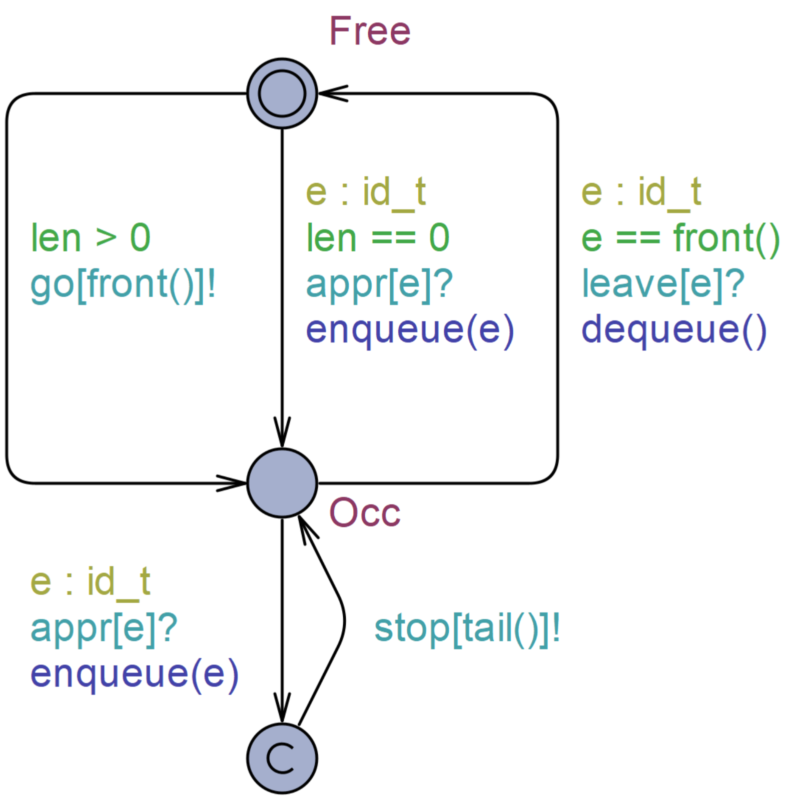
Gate Control




```
id_t list[N+1];
int[0,N] len;

// Put an element at the end of the queue
void enqueue(id_t element)
{
	list[len++] = element;
}
```

Examples: Train Gate Systems

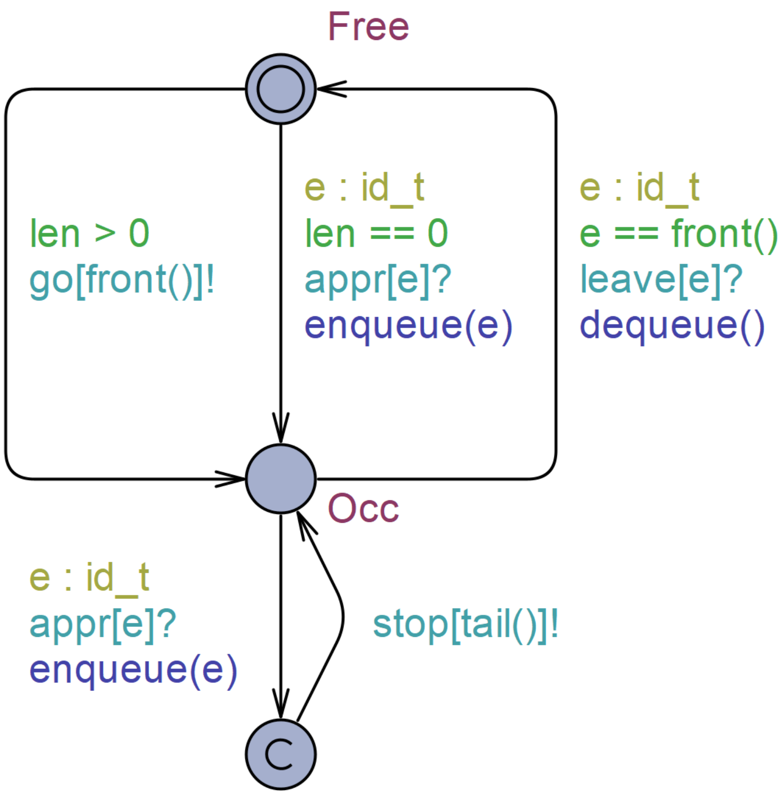
Gate Control




```
// Remove the front element of the queue
void dequeue()
{
    int i = 0;
    len -= 1;
    while (i < len)
    {
        list[i] = list[i + 1];
        i++;
    }
    list[i] = 0;
}
```

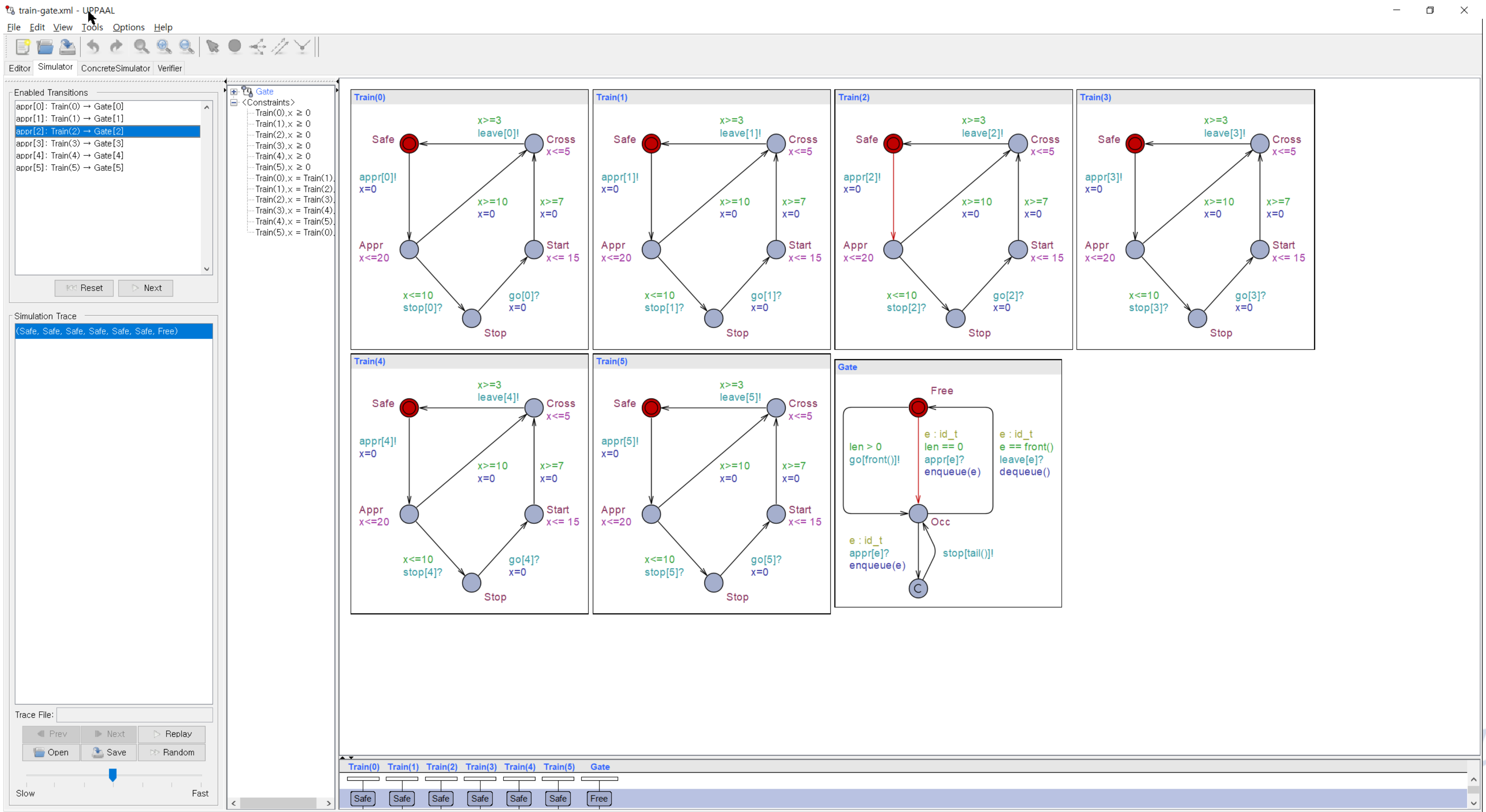
Examples: Train Gate Systems

Gate Control




```
// Returns the front element of the queue
id_t front()
{
  return list[0];
}

// Returns the last element of the queue
id_t tail()
{
  return list[len - 1];
}
```

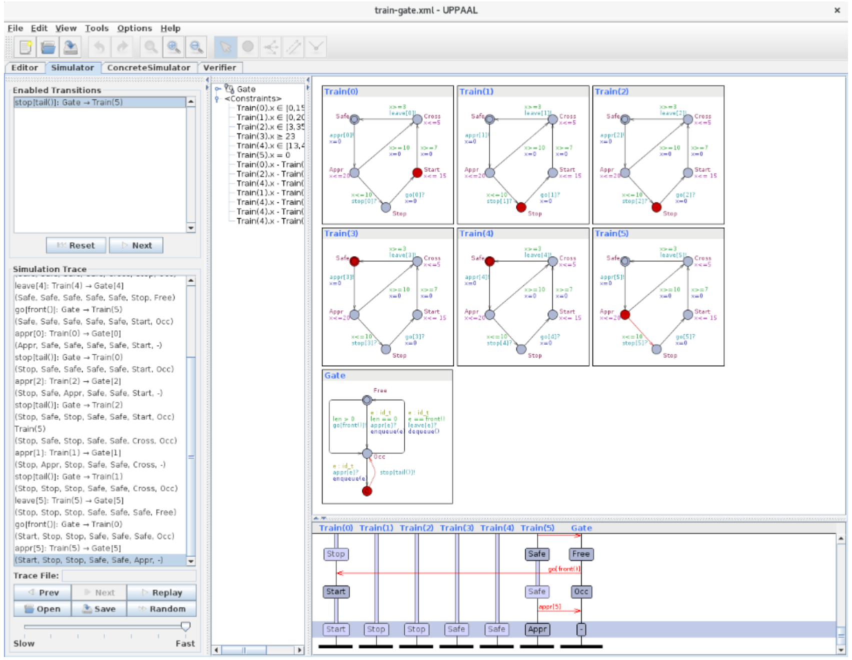
Demo



train-gate.xml - UPPAAL
File Edit View Tools Options Help
Editor Simulator ConcreteSimulator Verifier
Enabled Transitions
appr[0]: Train(0) → Gate[0]
appr[1]: Train(1) → Gate[1]
appr[2]: Train(2) → Gate[2]
appr[3]: Train(3) → Gate[3]
appr[4]: Train(4) → Gate[4]
appr[5]: Train(5) → Gate[5]
Reset
Next
Simulation Trace
(Safe, Safe, Safe, Safe, Safe, Safe, Free)
Trace File:
Prev
Next
Replay
Open
Save
Random
Slow
Fast
Gate
<Constraints>
Train(0).x ≥ 0
Train(1).x ≥ 0
Train(2).x ≥ 0
Train(3).x ≥ 0
Train(4).x ≥ 0
Train(5).x ≥ 0
Train(0).x = Train(1).
Train(1).x = Train(2).
Train(2).x = Train(3).
Train(3).x = Train(4).
Train(4).x = Train(5).
Train(5).x = Train(0).
Train(0)
Safe
x>=3
leave[0]!
Cross
x<=5
appr[0]!
x=0
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<= 15
x<=10
stop[0]?
go[0]?
x=0
Stop
Train(1)
Train(2)
Train(3)
Train(4)
Train(5)
Gate
Free
len > 0
go[front()]!
e : id_t
len == 0
appr[e]?
enqueue(e)
e : id_t
e == front()
leave[e]?
dequeue()
Occ
e : id_t
appr[e]?
enqueue(e)
stop[tail()]!
C
Train(0) Train(1) Train(2) Train(3) Train(4) Train(5) Gate
Safe Safe Safe Safe Safe Safe Free

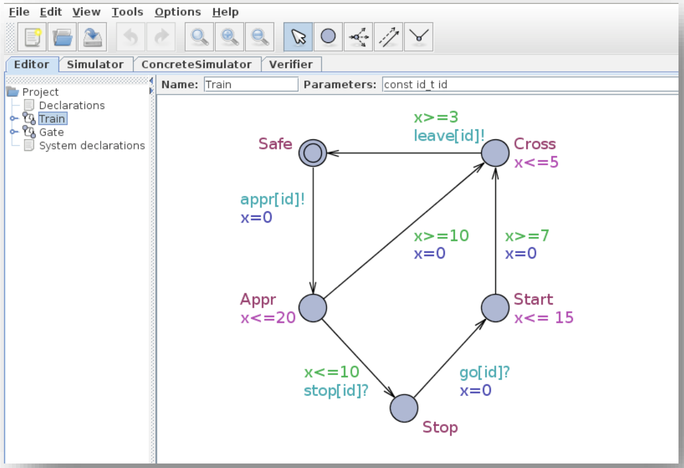
Uppaal

- An integrated tool environment for modeling, validation, and verification of real-time systems



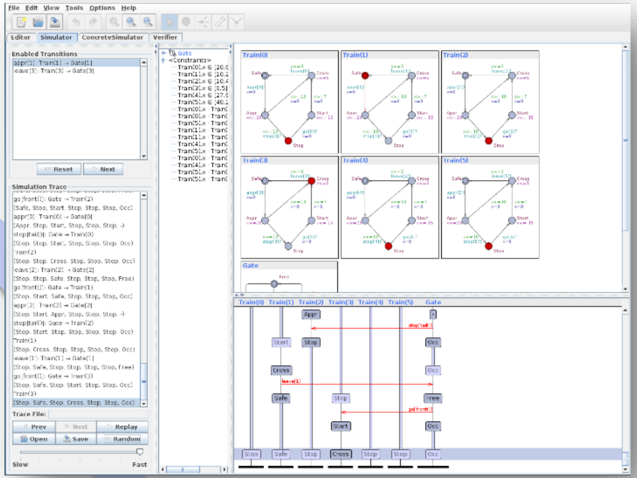
UPPAAL Analysis

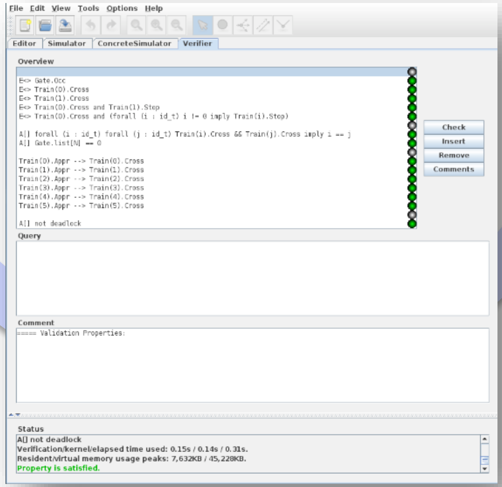
Modeling



- Timed Automata
- Declarations
- Functions

Simulation

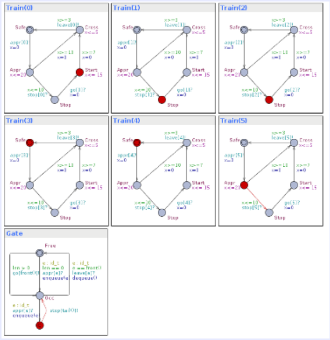




Model Checking Verification

- Timed CTL

Uppaal Tools Capability

System Specifications (Models)	Requirement Specification
• Timed Automata • Stochastic Hybrid Automata • A description language with data types	• Timed CTL • Probabilistic CTL
	• E<> Train(0).Cross • A[] (Train(0).Appr imply A<> Train(0).Cross

Model Checking

Yes/No (Counterexample)

Formal Requirement Specification for Timed Systems

Requirement Specification

- Property specification
- E.g.
  - "There is no more than one train crossing the bridge at any time (Safety)"
  - "A train can cross the bridge whenever it approaches the gate (Liveness)"
  - "The gate can receive and store messages from approaching trains in the queue"

TCTL in UPPAAL

- Quantifier
  - E – exists a path (“`E`” in UPPAAL).
  - A – for all paths (“`A`” in UPPAAL).
  - G – all states in a path (“`[]`” in UPPAAL).
  - F – some state in a path (“`<>`” in UPPAAL).
- Local and state property

$$e ::= p.l \mid g_d \mid g_c \mid e \ \mathit{and}\ e \mid e \ \mathit{or}\ e \mid \mathit{not}\ e \mid e \ \mathit{imply}\ e \mid (e)$$

  - *p* is a process name, and *l* is a location of timed automata
  - g_d is a guard on data, and g_c is a guard on clock

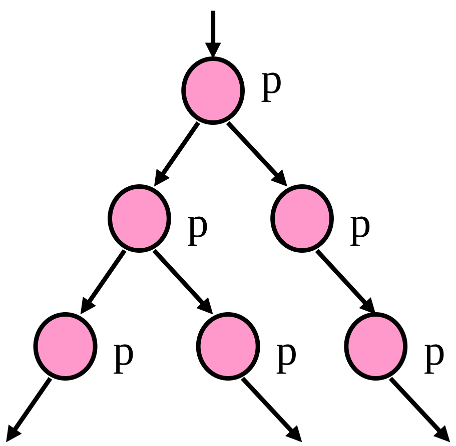
TCTL in UPPAAL

- Query composition

Name	TCTL	UPPAAL TCTL	Equivalent
Invariantly	AG p	A[] p	not E<> not p
Eventually	AF p	A<> p	not E[] not p
Potentially always	EG p	E[] p	
Possibly	EF p	E<>p	
Leads to	A[](p → A<>)	p --> q	A[] (p imply A<> q)

A[] p : “Invariantly p”

- **`A[] p`** : p holds invariantly.



“There is no more than one train crossing the brid ge at any time (Safety)”

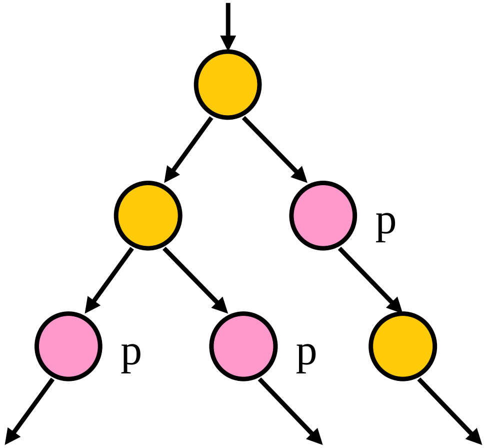
“A[] Train1.Cross + Train2.Cross + Train3.Cross + Train4.Cross <= 1”

: TrainID.Cross has a boolean value

- p is true in all reachable states.

A<> p : “Inevitable p”

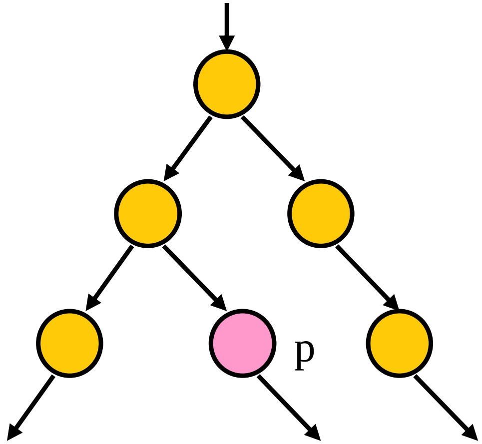
- **`A<>` p** : p will inevitable become true, the automaton is guaranteed to eventually reach a state in which p is true.



- p is true in some state of all paths.

E<> p : "p Reachable"

- **`E<>` p** : it is possible to reach a state in which p is satisfied.



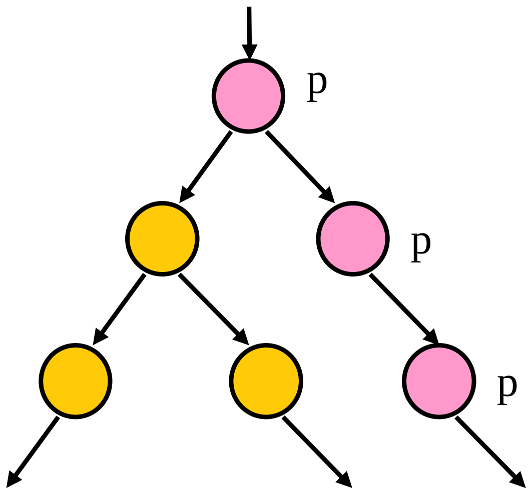
"Train 1 eventually crosses the bridge"

"E<> Train(1).Cross"

- p is true in (at least) one reachable state.

E[ ] p : “Potentially Always p”

- **`E[] p`** : p is potentially always true.

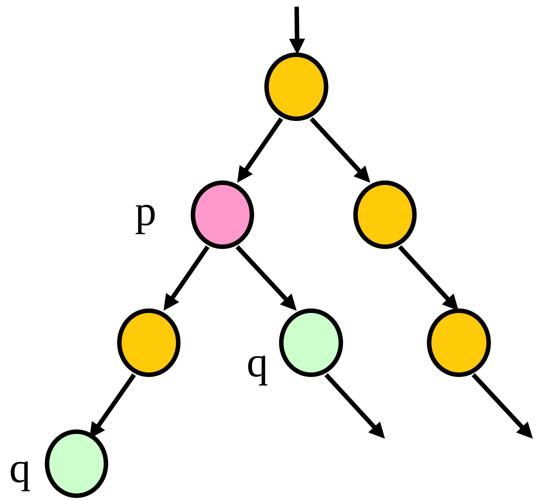


- There exists a path in which p is true in all states.

p --> q : “p lead to q”

- **p** --> **q** : if p becomes true, q will inevitably become true

  same as A[] (**p** imply A<> **q**)



“Train 1 can cross the bridge whenever it approaches the gate”

“Train(1).Appr --> Train(1).Cross”

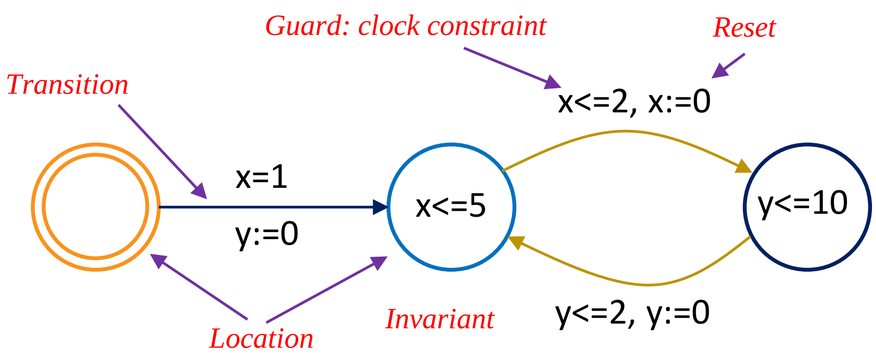
- In all paths, if p becomes true, q will inevitably become true.

Useful Queries

- A[] not deadlock
  - Invariantly the process is not deadlocked.
- sup: list
  - Returns the suprema (infima) of the expressions (maximal value s in case of integers, upper bounds, strict or not, for clocks).
- inf: list
  - Returns the infima of the expressions

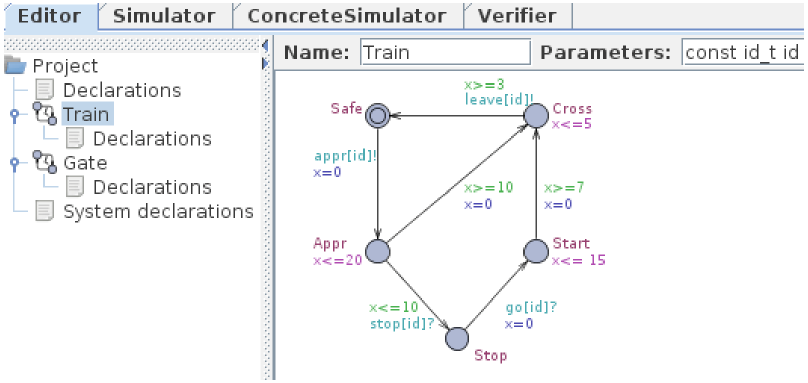
System Specification for Timed Systems with Timed Automata

Timed Automata in Uppaal

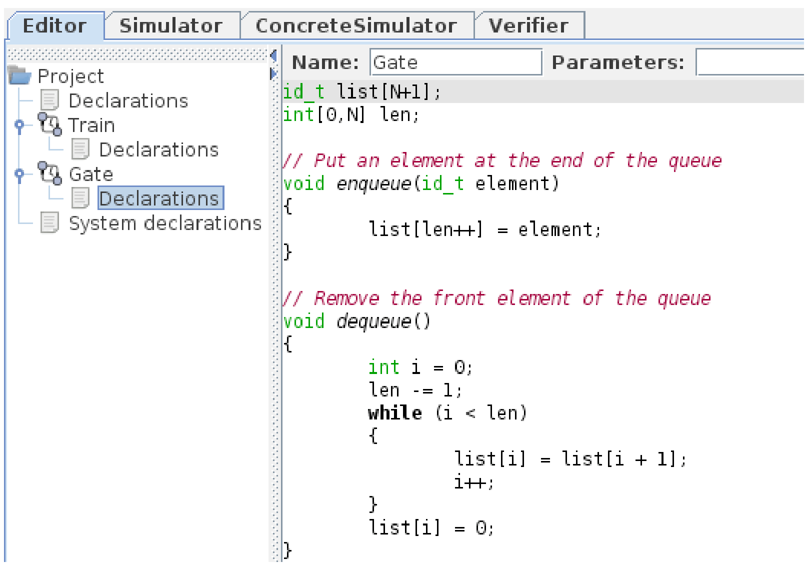


- Channel (Event)
  - Process communication and syn chronization
- Timing constraints
  - Specifying event arrivals
  - E.g., periodic and sporadic
- Data variable
  - Guards and assignments
- C-subset description
  - User-defined data type
  - Structure
  - Data array

Specification Structure

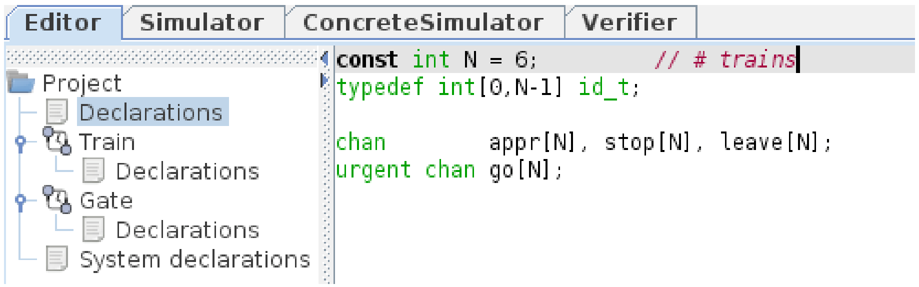


1. Definition: Timed system



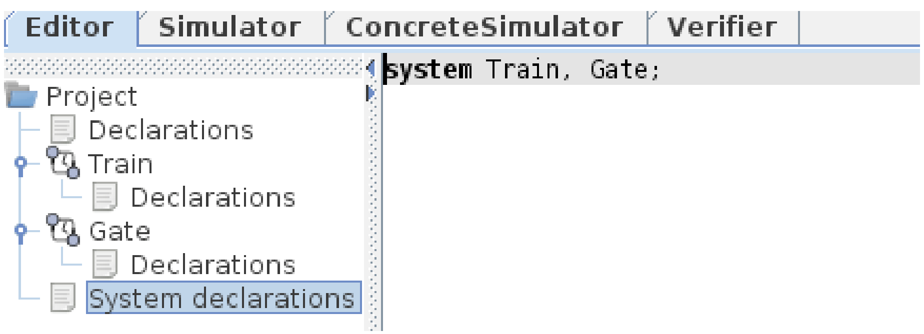
2. Declaration: Local
-Channels, Variable, Constant,
-User-defined Structure and Data type,
-Functions

Specification Structure



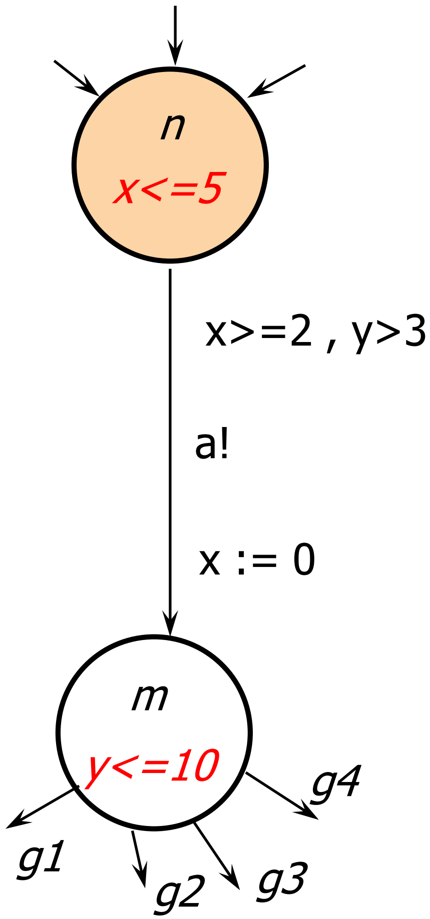
3. Declaration: Global

- Channels, Variable, Constant,
- User-defined Structure and Data type,
- Functions



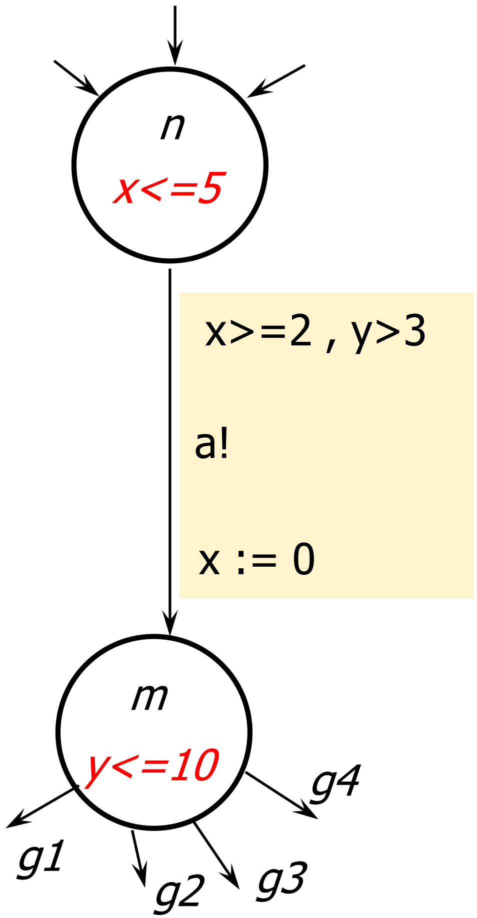
4. Declaration: Process

Syntax: Location



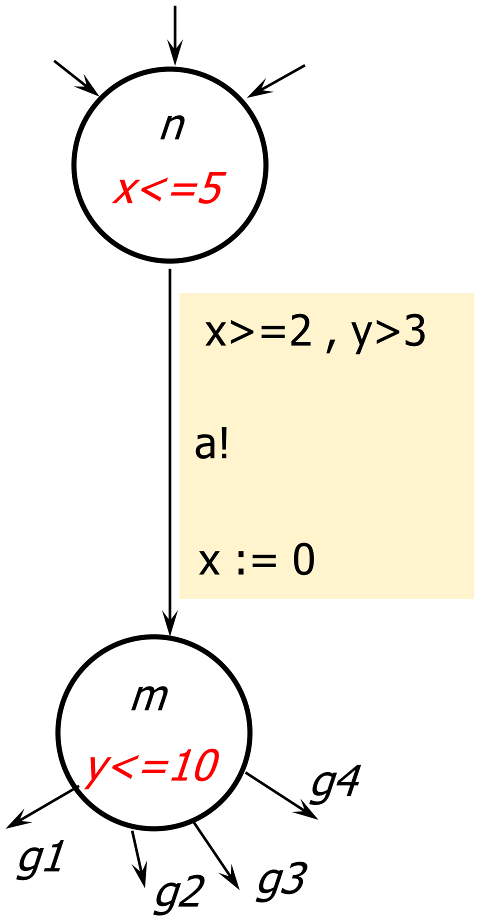
- Location name – “n”
- (Clock) Invariants
  - **x <= 5**
    - x is less than or equal to 2.
  - **x < y**
    - x is (strictly) less than y
  - “x” must be a clock

Syntax: Edge



- Selections
  - Selections non-deterministically
- Guards
  - An edge is enabled in a state if and only if the guard evaluates to true.
- Synchronisation
  - Processes can synchronize over channels.
- Updates
  - When executed, the update expression of the edge is evaluated.
- Weights
  - Probabilistic branches .

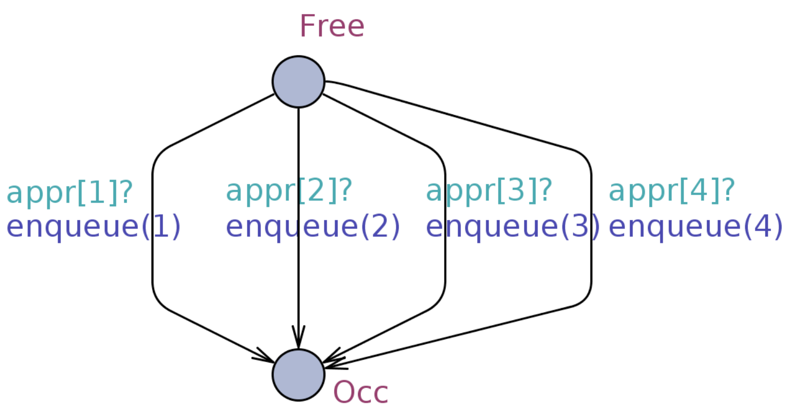
Syntax: Edge



- Selection
  - **select: i : int[0,3]**
  - synchronization: a[i]?
  - update expression: receive_a(i)

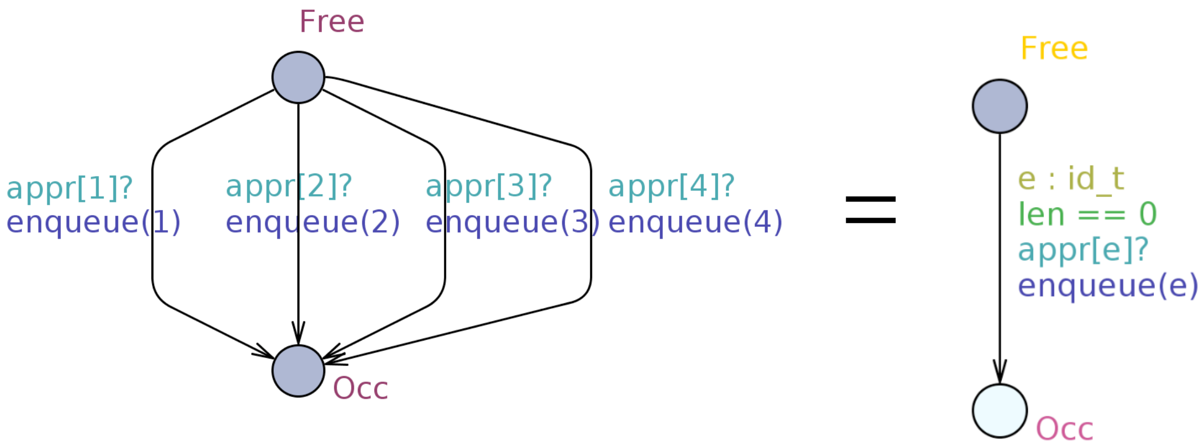
Selection

- Non-deterministic selection
  - Suppose that for the train gate systems, we have to sense one of approaches of 4 trains, we might model the selection like

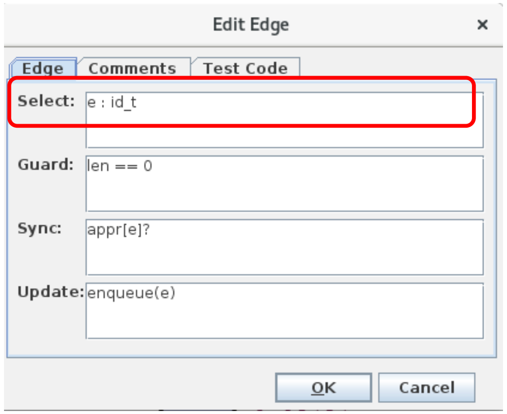


Selection

- Non-deterministic selection
  - The multiple transitions to be selected can be simplified by sele ction operator



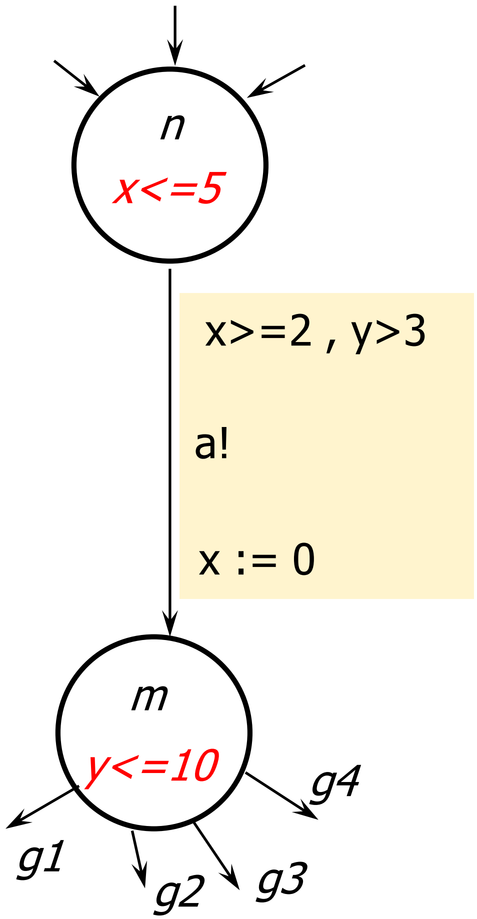
typedef int[1,4] id_t;



Selection

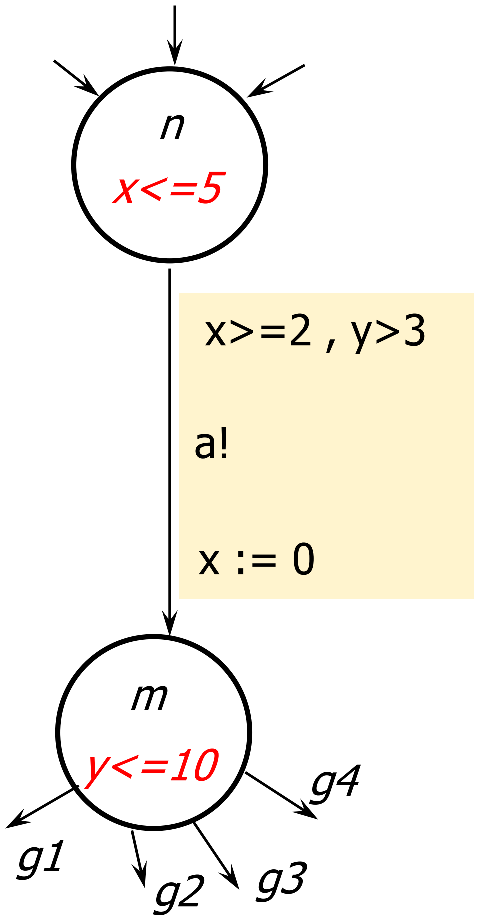
- Forall / Exists Predicates
  - forall (x:int[0,42]) expr
    - true if expr is true for all values in [0,42] of x
  - exists (x:int[0,4]) expr
    - true if expr is true for some values in [0,42] of x
  - Example
    - forall (x:int[0,4]) array[x];

Syntax: Edge



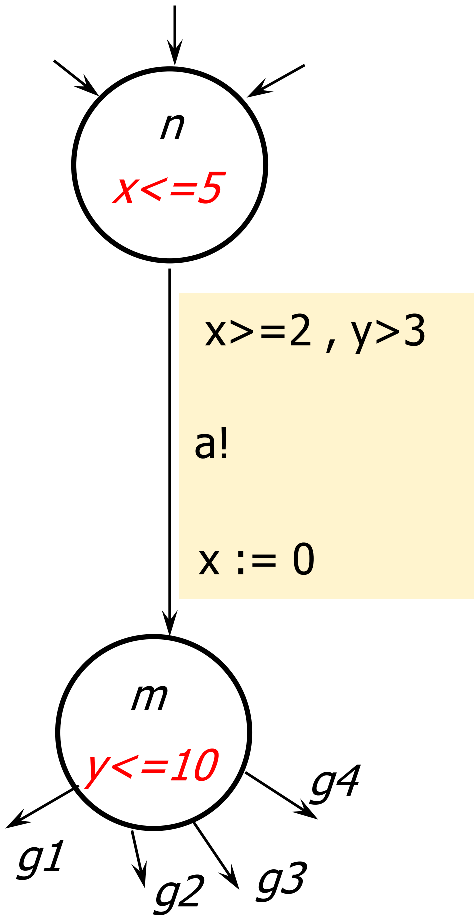
- Guard
  - **x >= 1 && x <= 2**
    - x is in the interval [1,2].
  - **x < y**
    - x is (strictly) less than y.

Syntax: Edge



- Synchronization
  - **e!**
    - Talk to the channel e
  - **e?**
    - Listen to the channel e
  - Both edges are fired at the same time

Syntax: Edge



- Update
  - Update ::= [Expression (',' Expression)*]
  - **x = 0**
    - clock (or integer variable) x is reset.
  - **j = (i[1]>i[2] ? i[1] : i[2])**
    - Conditional assignments
  - **x = 1, y=2*x**

Parameters

- Templates and functions are parameterised.
  - Parameters ::= [Parameter (',' Parameter)*]
  - Parameter ::= Type ['&'] ID ArrayDecl*
  - **P(clock &x, bool bit)**
    - Process template P has two parameters: the clock x and the Boolean variable bit.
  - **Q(clock &x, clock &y, int i1, int &i2, chan &a, chan &b)**
    - Process template Q has six parameters: two clocks, two integer variables (with default range), and two channels. All parameters except i1 are reference parameters.

System Declaration

- Constants
  - Integers, booleans, and arrays and records over integers and booleans can be marked constant by prefixing the type with the keyword const.
- Arrays
  - typedef scalar[3] s_t;
  - int a[s_t];
- Record Variables

```
struct
{
  int a;
  int b;
} s;
```

- Scalars
  - typedef scalar[3] mySet;
  - mySet s;
  - int a[mySet];

Declarations in UPPAAL

- The syntax used for declarations in UPPAAL is similar to t he syntax used in the C programming language.
- **Clocks**:

Syntax	Example
clock x1, …, xn;	clock x, y; //Declares two clocks: x and y.

Declarations in UPPAAL (cont.)

- **Data variables**

Syntax	Description
int n1, …	Integer with "default" domain.
Int[l,u] n1,…	Integer with domain from "l" to "u".
Int n1[m], …	Integer array w. elements n1[0] to n1[m-1].

- **Example**

```
int a, b;
int[0,1] a, b[5];
```

Declarations in UPPAAL (cont.)

- **Synchronization** (or channels):

Syntax	Description
chan a, … ;	Ordinary channels.
urgent chan b, … ;	Urgent channels (described later)
broadcast c;	Broadcasting channels
urgent broadcast chan d;	Urgent broadcasting channels

- **Example**:

```
chan a, b[2];
urgent chan c;
```

Declarations in UPPAAL (const.)

- **Constants**

Syntax	Description
const int c1 = n1;	Create a constant cl with default value nl

- **Example**:

```
const int[0,1] YES = 1;
const bool NO = false;
```

Declarations in UPPAAL (const.)

- Type definition

Syntax	Description
typedef int[l,u] id_t;	Define a new type id_t with interger with bound of [l,u]

typedef struct {
    int len;
    int entry[id_t];
} queue_t ;
	Define a structure with two variables: *len* of integer type and *entry* of id_t type

- Example

```
queue_t que;    // que.len ; que.entry[0]; que.entry[1]
queue_t que[0,2]
```

Declarations in UPPAAL (const.)

- Function

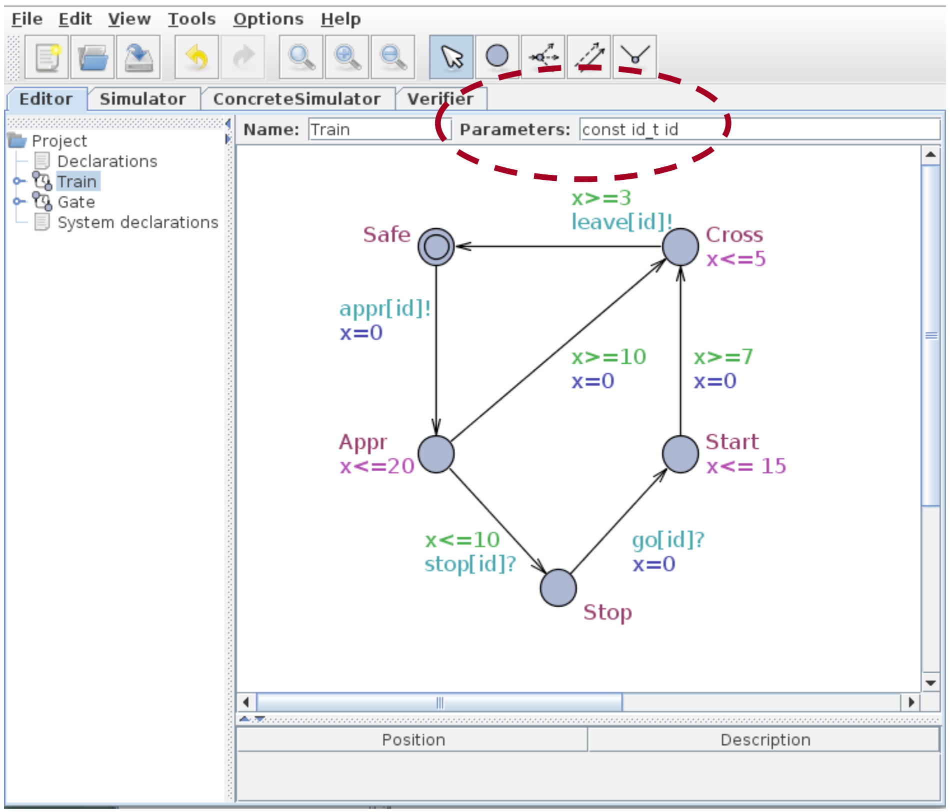
Syntax	Description

int add(int a, int b){
    return a + b;
}
	Define a function with call-by-value interger parameters.

int add(int & a, int & b){
    return a + b;
}
	Define a function with call-by-reference integer parameters.

Declaration in UPPAAL

- Template Parameters



const int N = 6;
// # trains
typedef int[0,N-1] id_t;

Declaration in UPPAAL

- Process Declaration and Instantiation

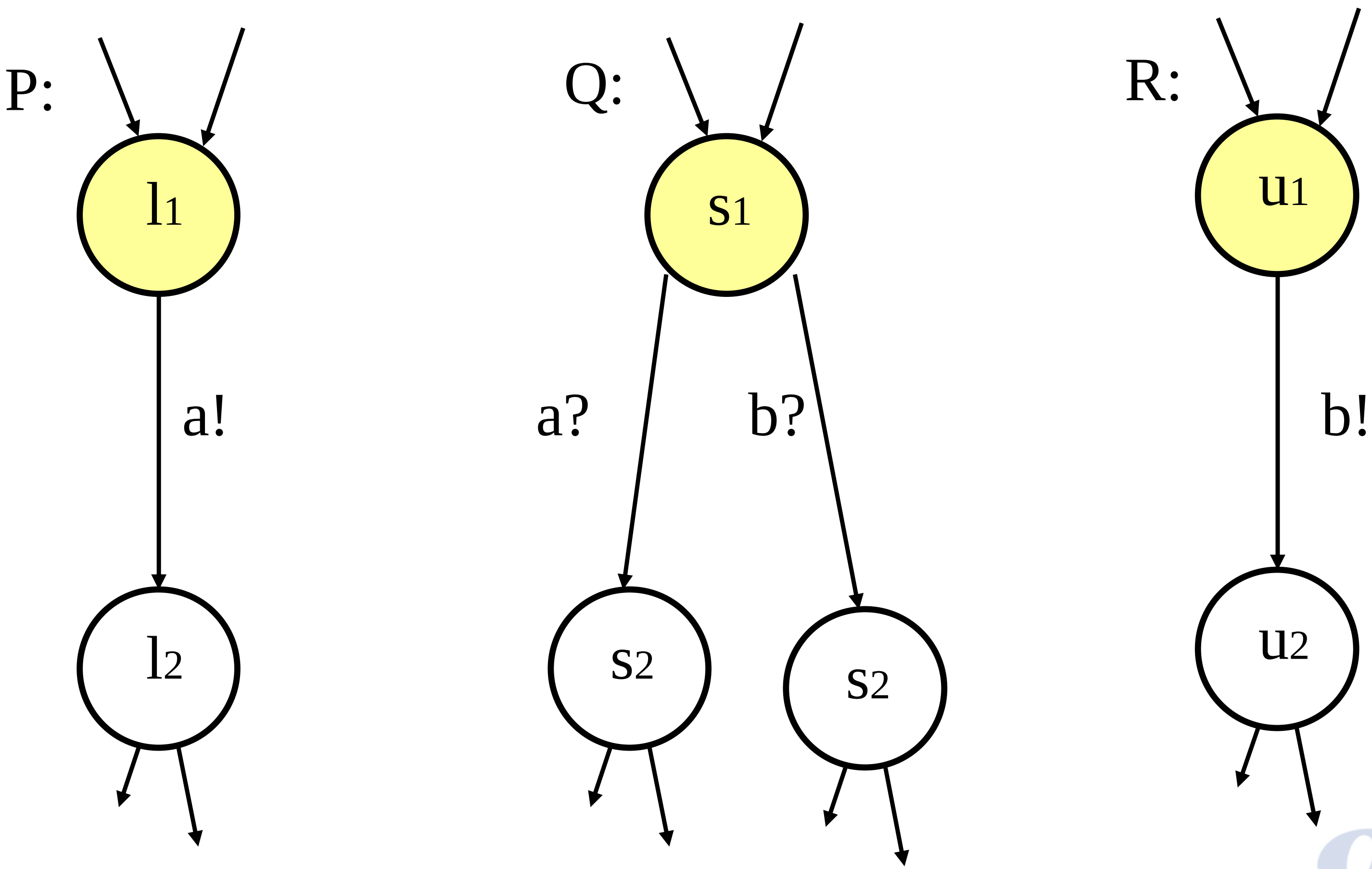
Syntax	Description
Train2 = Train(2) ;	Declare a train process of id 2 in name of Train2
system Train1, Train2, … ;	Instantiate processes of Train1, Train2 ,…

Function Definitions

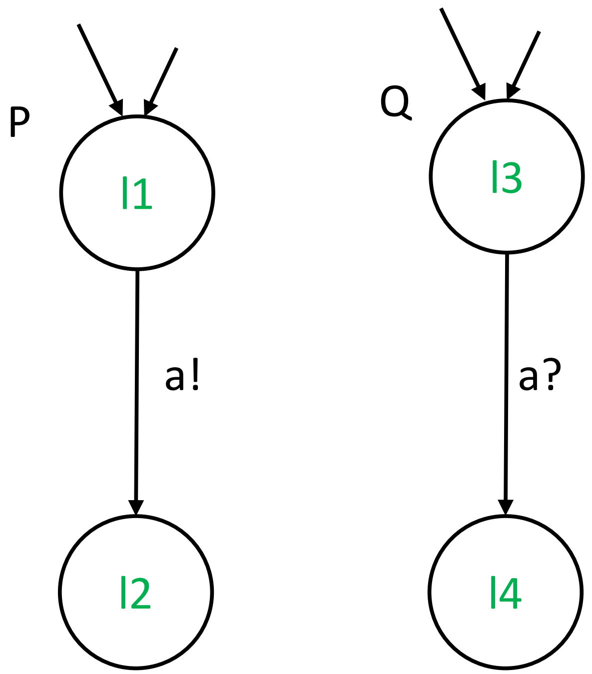
```
int add(int a, int b)
{
   return a + b;
}

void initialize(int& a[10])
{
  for (i : int[0,9])
  {
    a[i] = i;
  }
}
```

```
void swap(int &a, int &b)
{
  int c = a;
  a = b;
  b = c;
}
```

Non-determinism

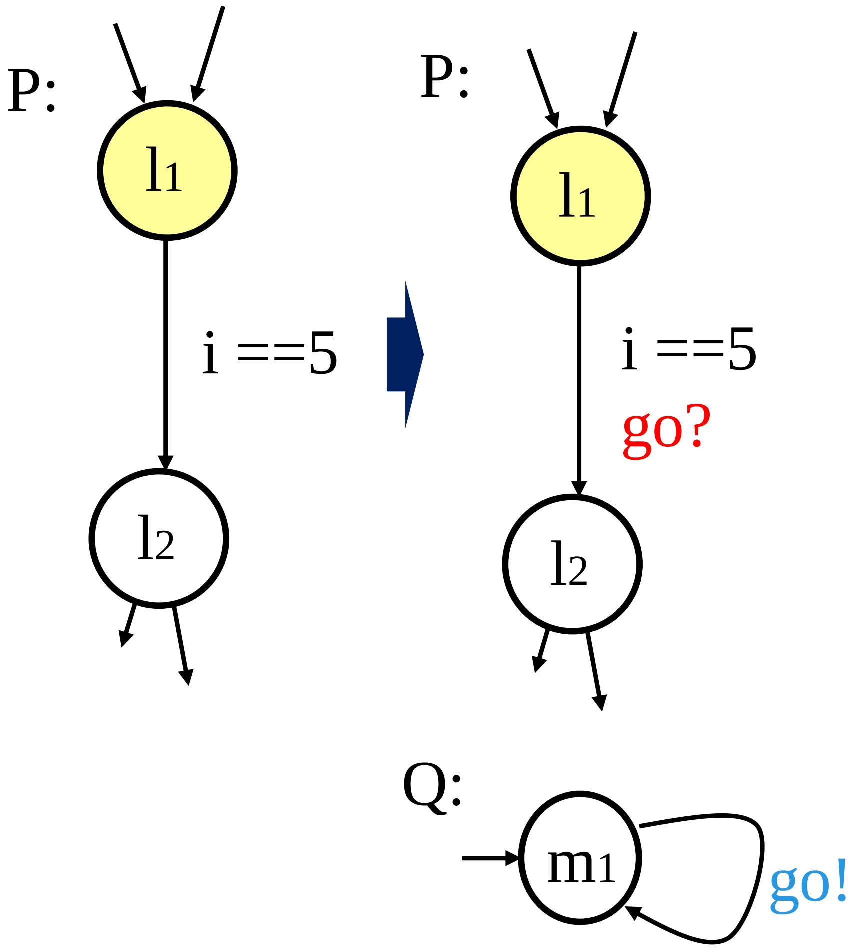


Urgent Channels



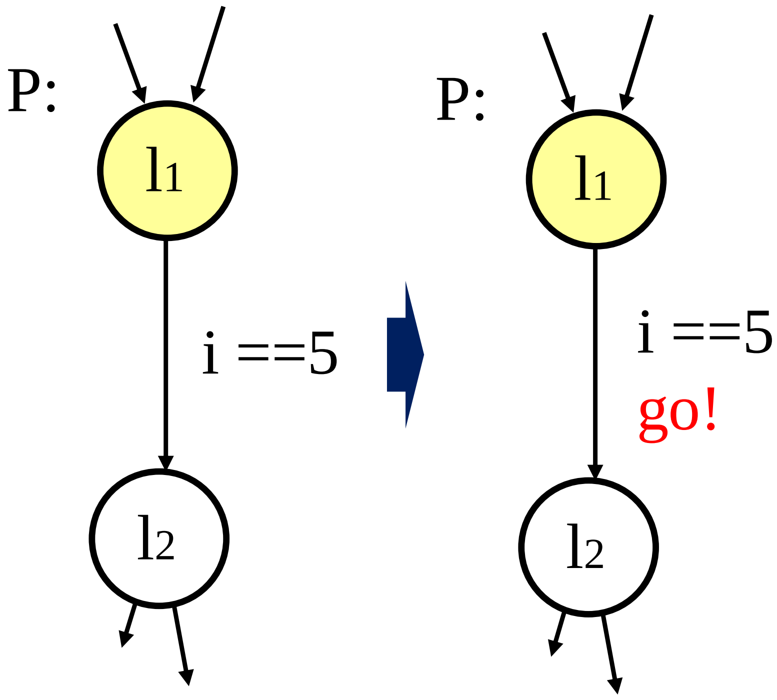
- Supposed that l1 and l3 have no invariants, when the channel a is synchronized?
- Solution
  - Set the channel to Urgent so that it is immediately synchronized whey it is ready to be fired
  - **urgent channel a;**
- No delay when the channel is rea dy to be synchronized,
- Restrictions
  - No clock guard allowed on transition s with urgent channels.

Urgent Channels



- Assume i is a data variable.
- We want P to take the transition from l_1 to l_2 as soon as i==5.
- Solution 1
  - Add a template like "Q" synchronizing the urgent channel "go!"

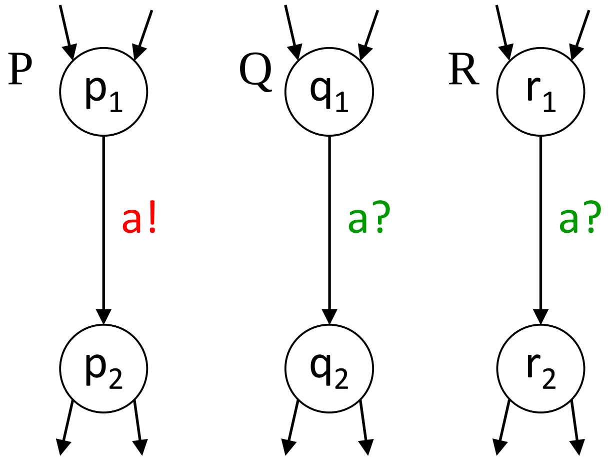
Urgent Channels



- Solution 2 (simpler than Solution 1)
  - Add a channel "go" declared as urgent broadcast
  - **urgent broadcast chan go;**

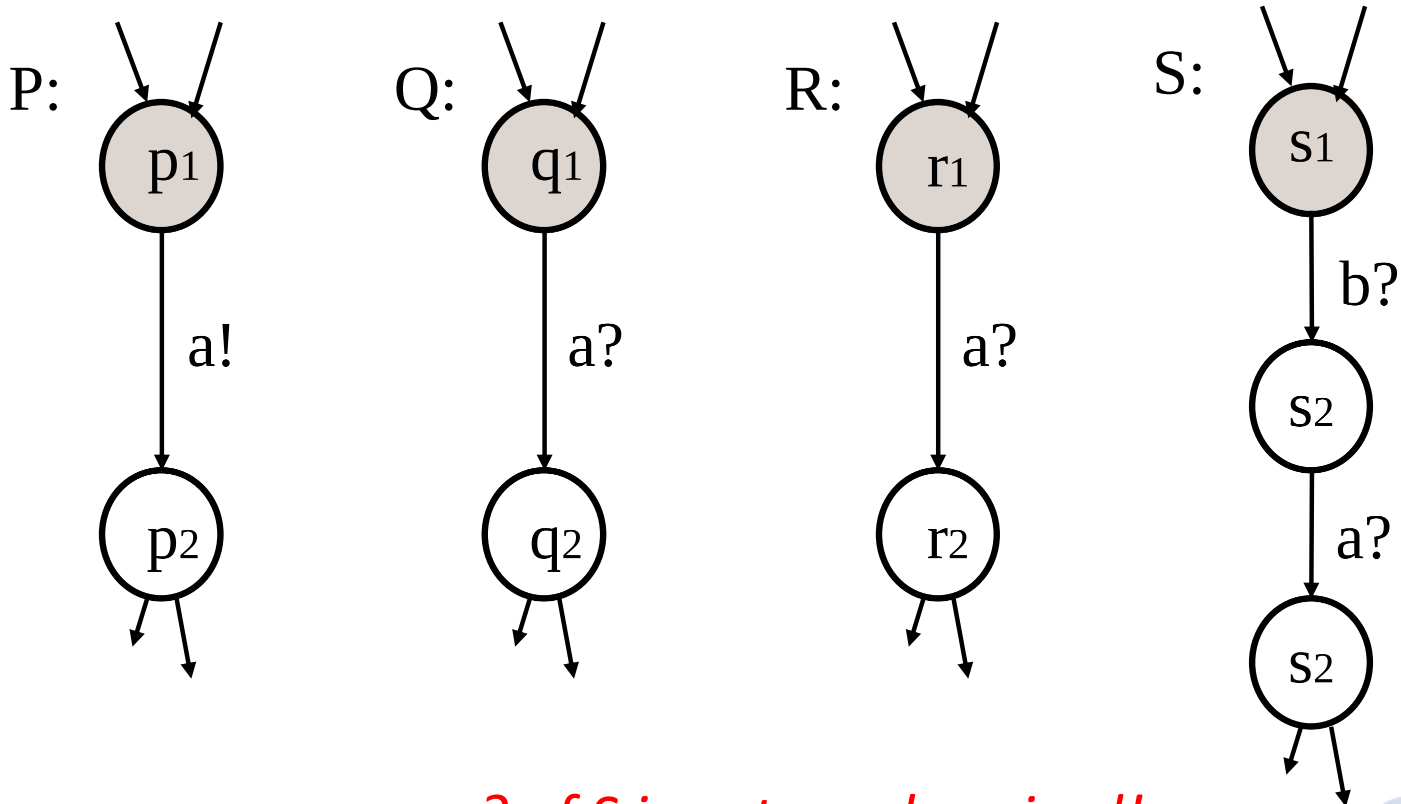
Broadcasting Channel

Syntax	Description
broadcast chan a;	Declare a broadcast channel ”a” that allows one or more synchronizations simultaneously
urgent broadcast chan b;	No delay to synchronize with b



- A set of edges in different processes can synchronize if one is emitting and the others are receiving on the same broadcast channel.
- *P* always emit a! when it is ready
- Receivers *Q* and *R* must synchronize if they can. *No blocking!*

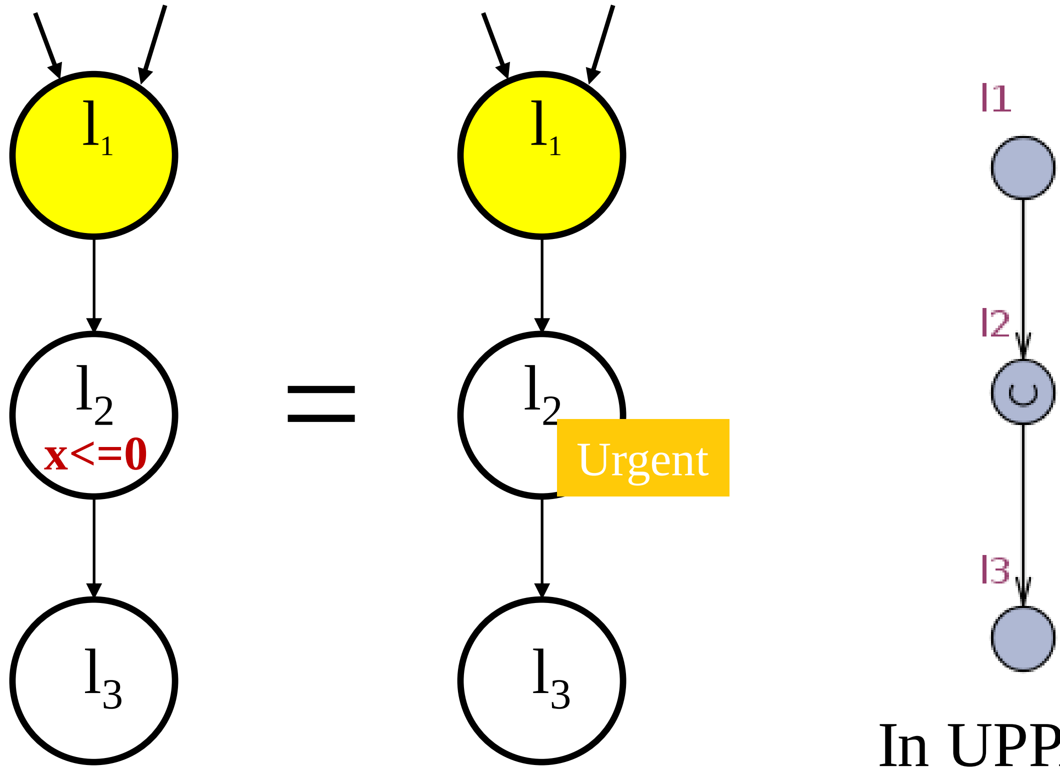
Broadcast Synchronization: Example



a? of S is not synchronized!

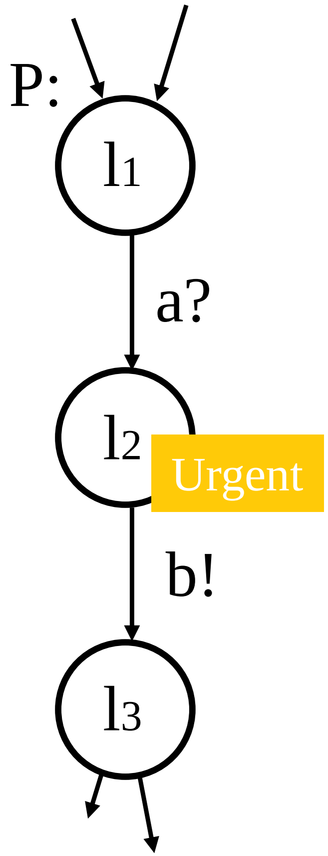
Urgent Location

- Stop at a location with *no time progress*



In UPPAAL

Urgent Location: Example

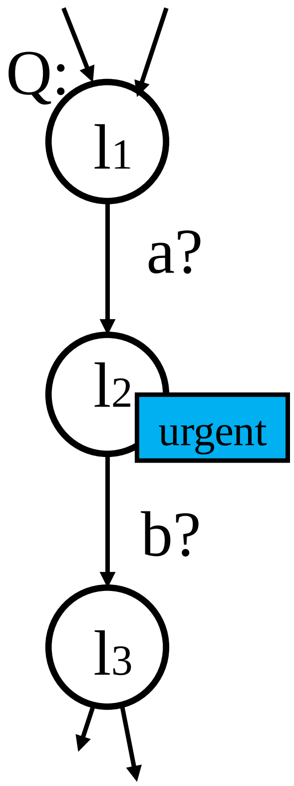


- *P* is synchronized with *b* as soon as it is synchronized with *a*
- There is *no time progress* between *a* and *b*

Urgent Location: Example

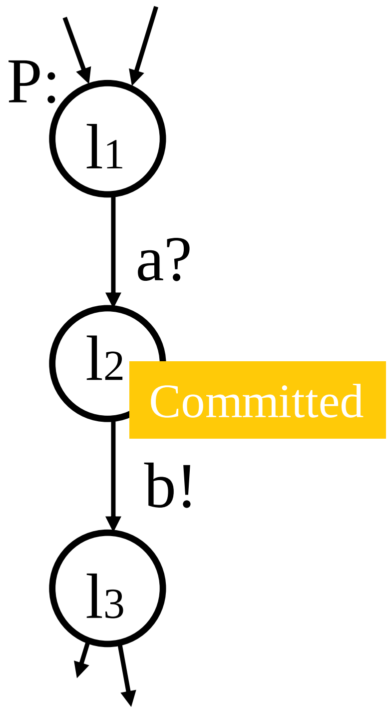
- Discussion: what will happen in this case, if “b!” is not ready?
  - Not allowed?
  - Wait for “b!”?

- Solution: Not allowed!
  - Deadlock



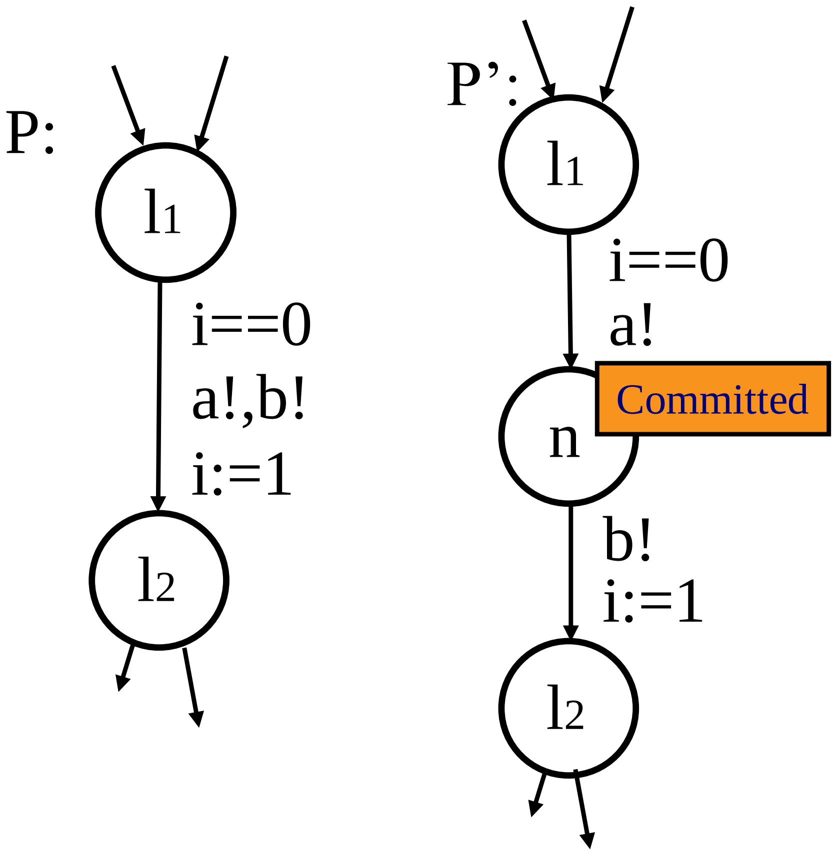
Committed Location

- Instantaneous step
- Next transition must involve automata in committed l ocation.



In UPPAAL

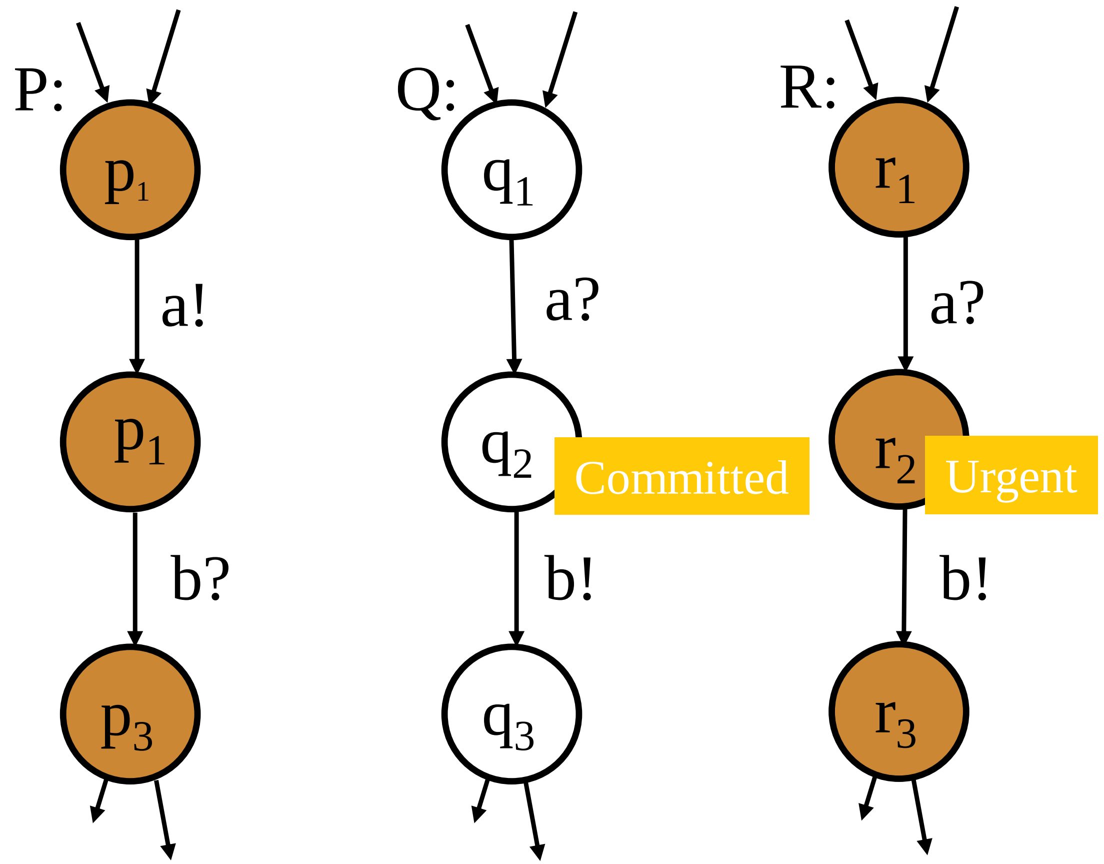
Committed Location: Example 1



- Assume: we want to model a process (P) simultaneously sending messages “a!” and “b!” (when i==0).
  - Not allowed in UPPAAL.
- Solution
  - Use the Committed location

Committed vs Urgent Location

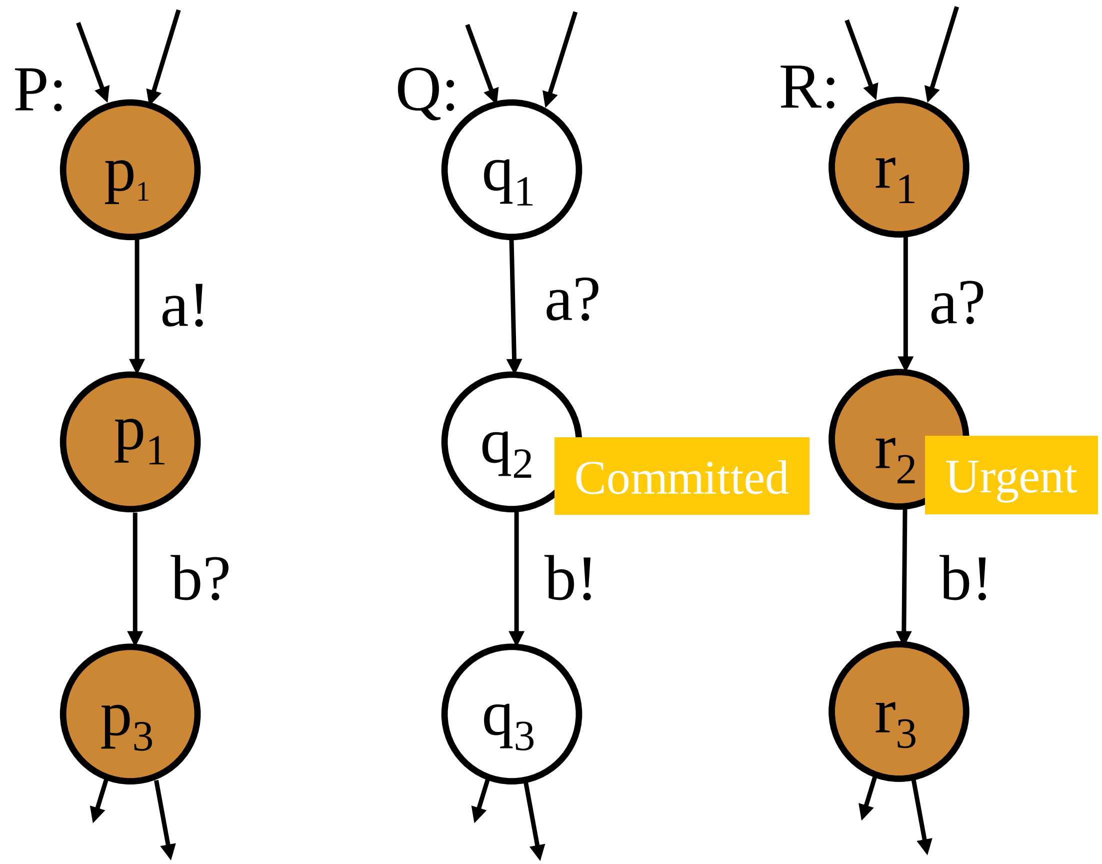
broadcast chan a;
chan b;



Which can make synchronization with P, Q or R?

Committed vs Urgent Location

broadcast chan a;
chan b;



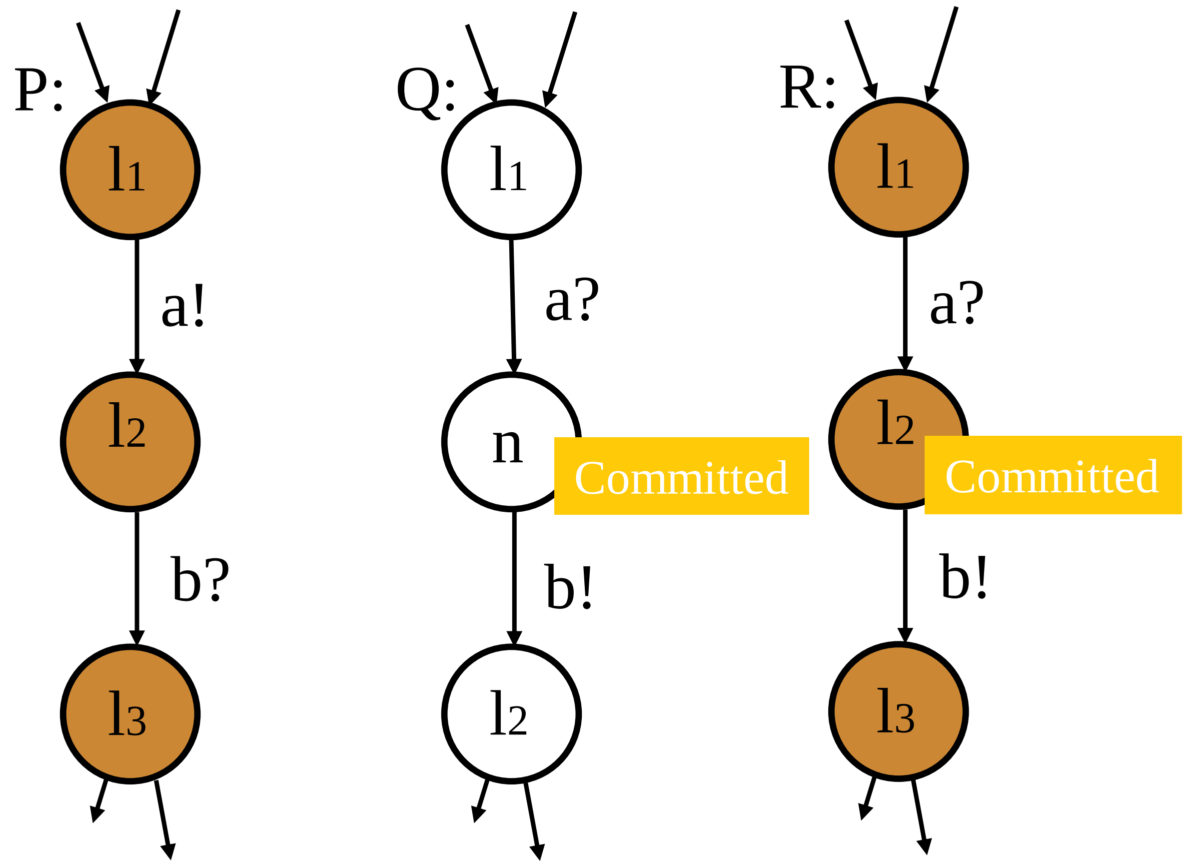
Answer: Q can make synchronization with P

Committed vs Urgent Location

- Urgent location takes *a timed step*, and *no time progress* on the location,
- Committed location takes *an instantaneous step*
- *In UPPAAL, the instantaneous step always has the priority over the timed step,*
  - *The outgoing transition from the committed location is taken rather than the one from the urgent location.*

Committed vs Committed Locations

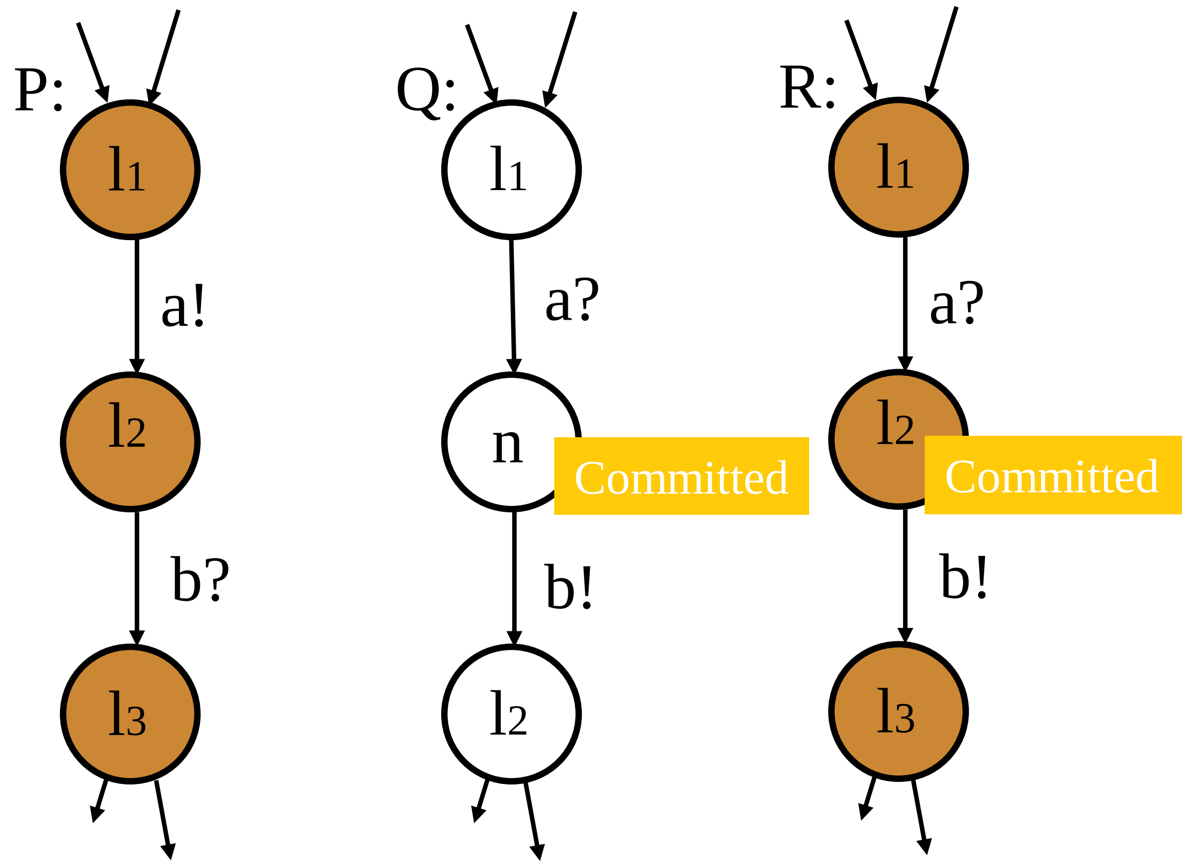
broadcast chan a;
chan b;



Which can make synchronization with P, Q or R?

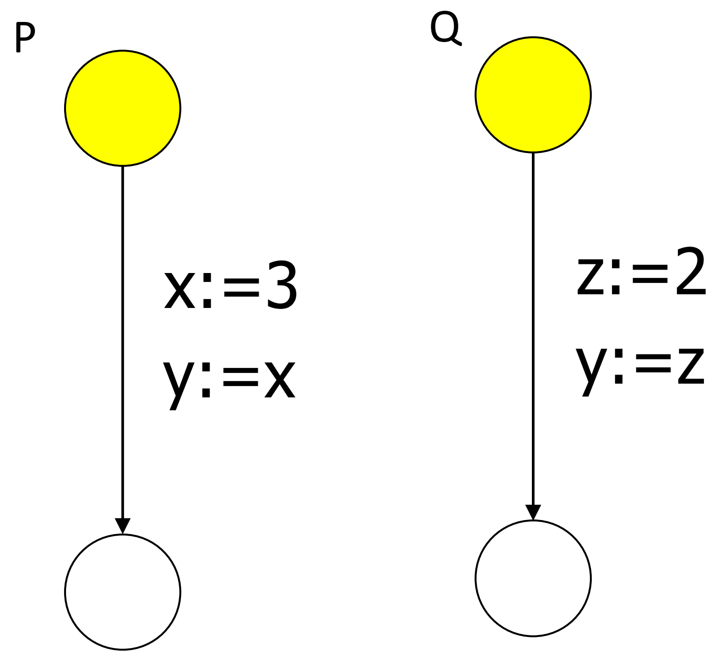
Committed vs Committed Locations

broadcast chan a;
chan b;



Solution: Either of Q or R is non-deterministically taken.

Committed vs Urgent Locations



- Assume: we models P and Q that access the shared variable y at the same time?
- We want that P has the priority over Q to assign to y.

Committed vs Urgent Locations

- Solution