First Lecture:
Modeling with
Finite, Timed and Hybrid Automata

Jean-François Raskin
Université Libre de Bruxelles
Belgium

Artist2 Asian Summer School - Shanghai - July 2008

Plan of the talk

- Reactive and Embedded Systems
- Modeling with Communicating Finite State Machines
- Modeling with Timed Automata
- Modeling with Hybrid Automata

Plan of the talk

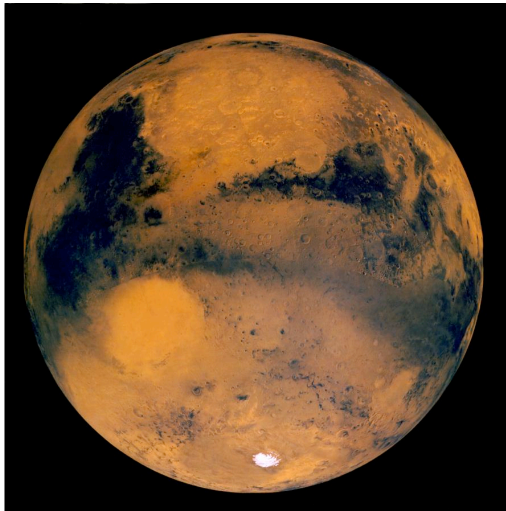
- Reactive and Embedded Systems
- Modeling with Communicating Finite State Machines
- Modeling with Timed Automata
- Modeling with Hybrid Automata

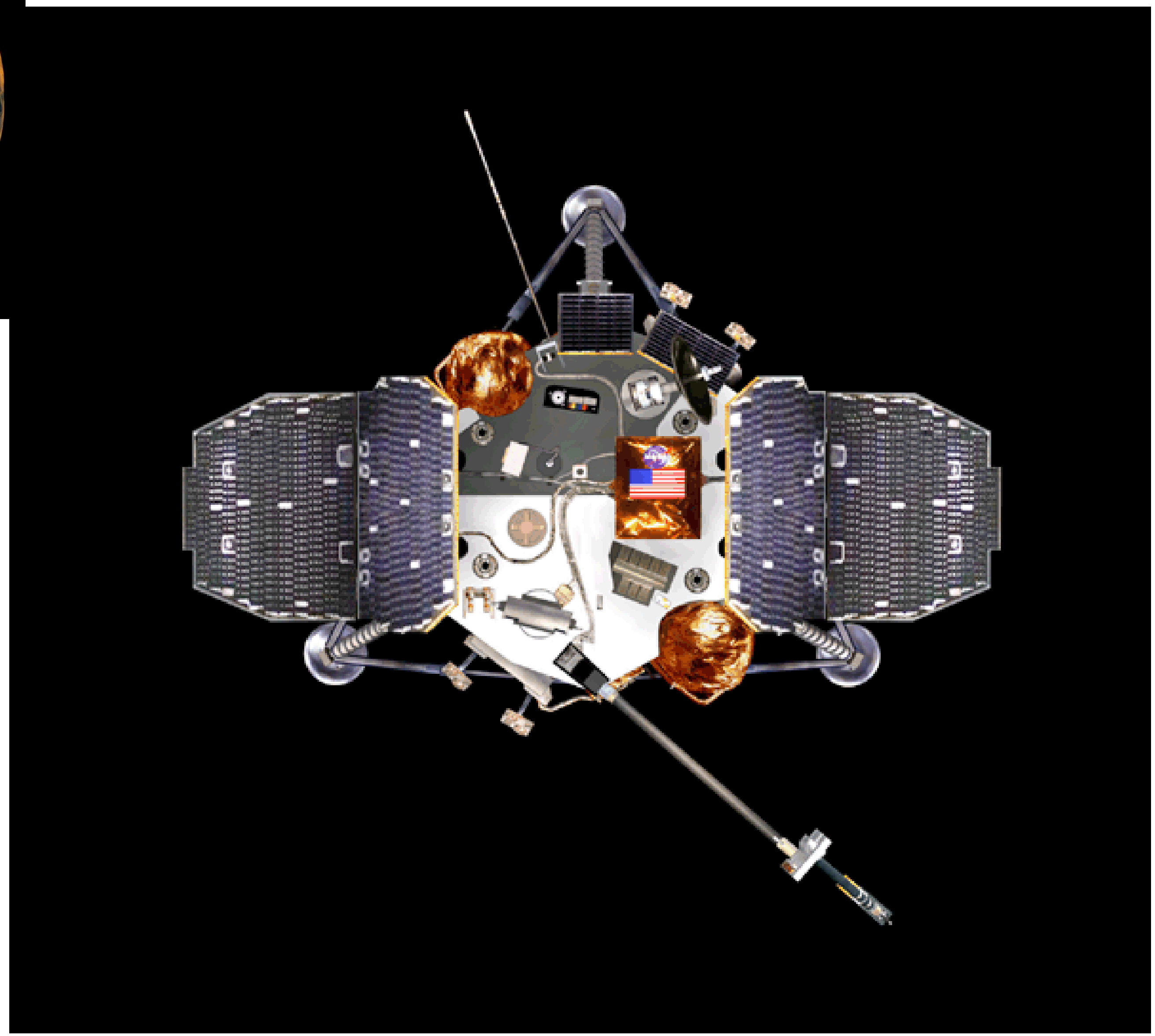
What are critical embedded systems ?


French Guniea, june 4, 1996


Mars, December 3, 1999

Crash caused by an uninitialized variable





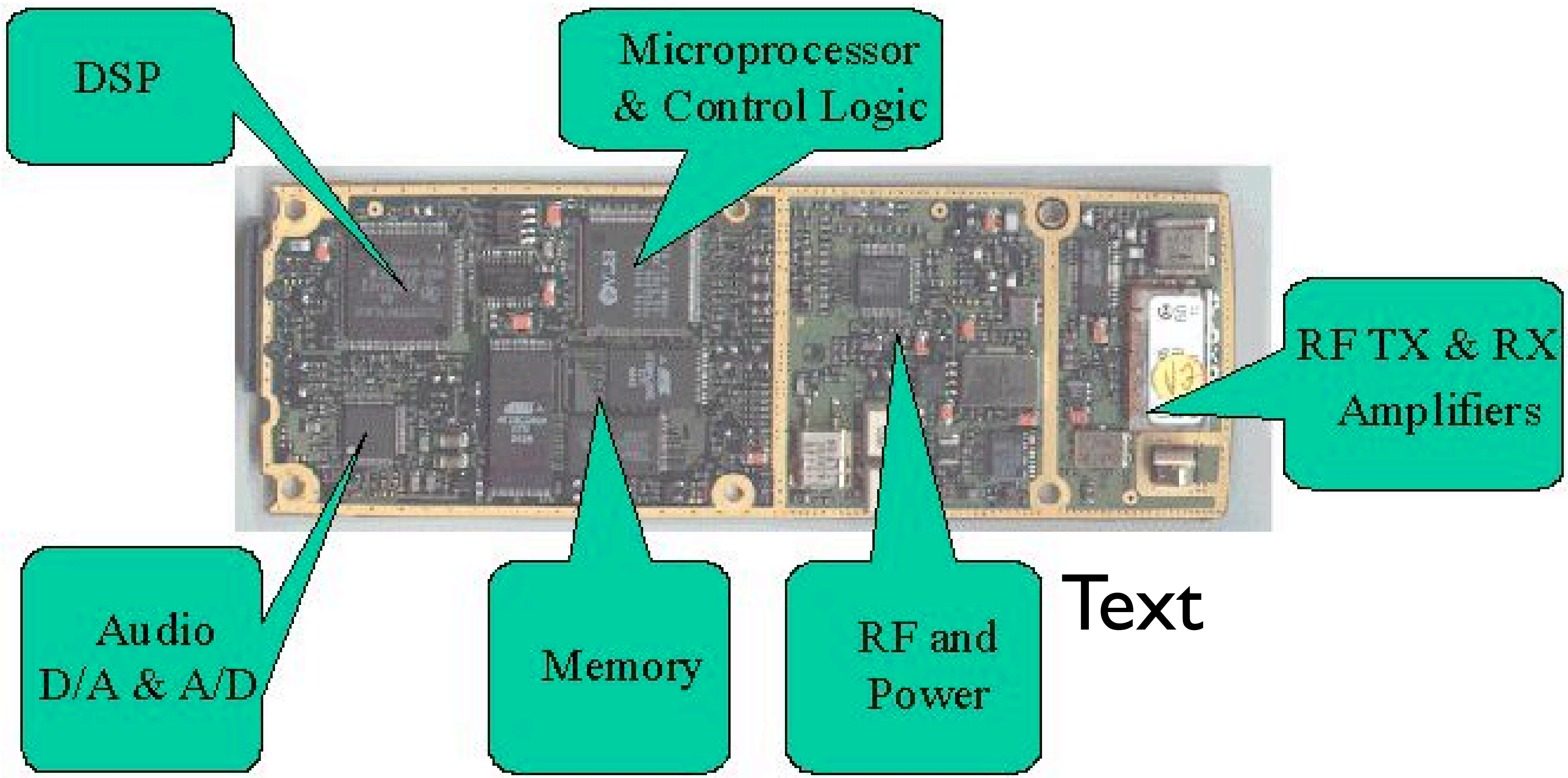

777
BOEING 777-300

300 horses power

100 processors

Cellular Phone

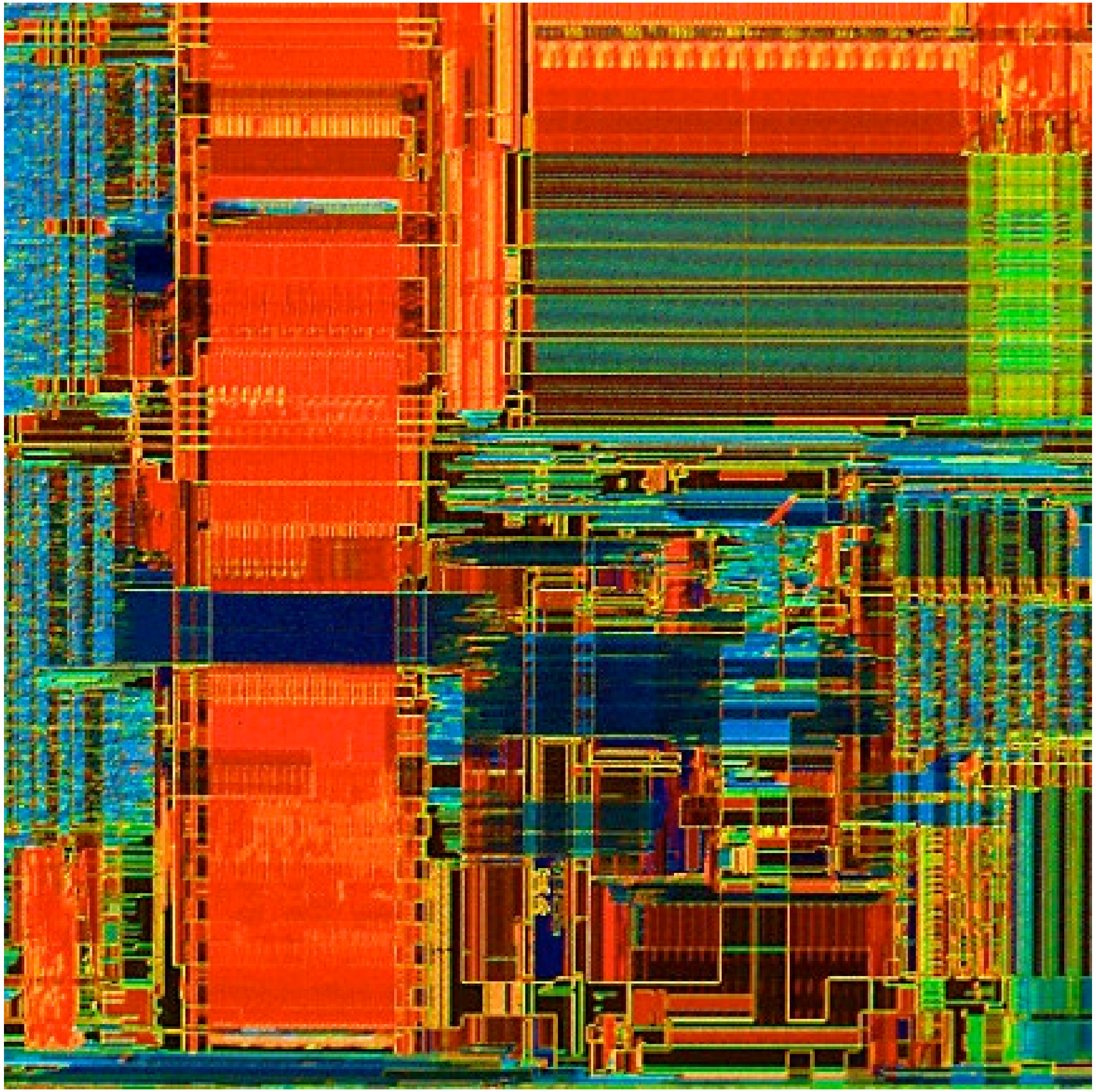
more and more software

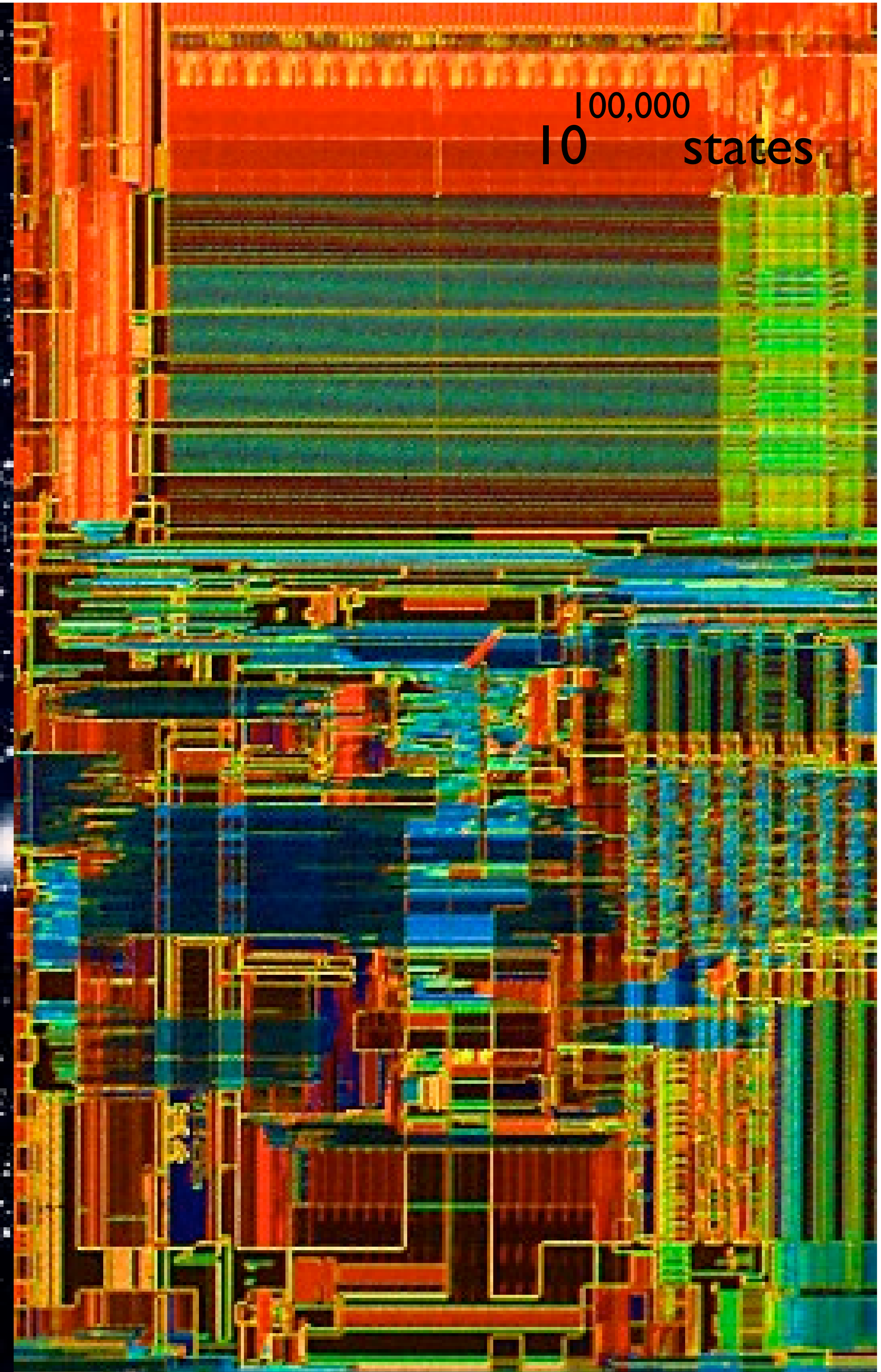


Text

Concurrency: several hardware and software components
Heterogeneity: digital (discrete time) and analog (continuous time)
Uncertainty: environment, exeptions handling

Concurrency : 300 000 logical gates




$10^{100,000}$ states

Reactive and embedded systems

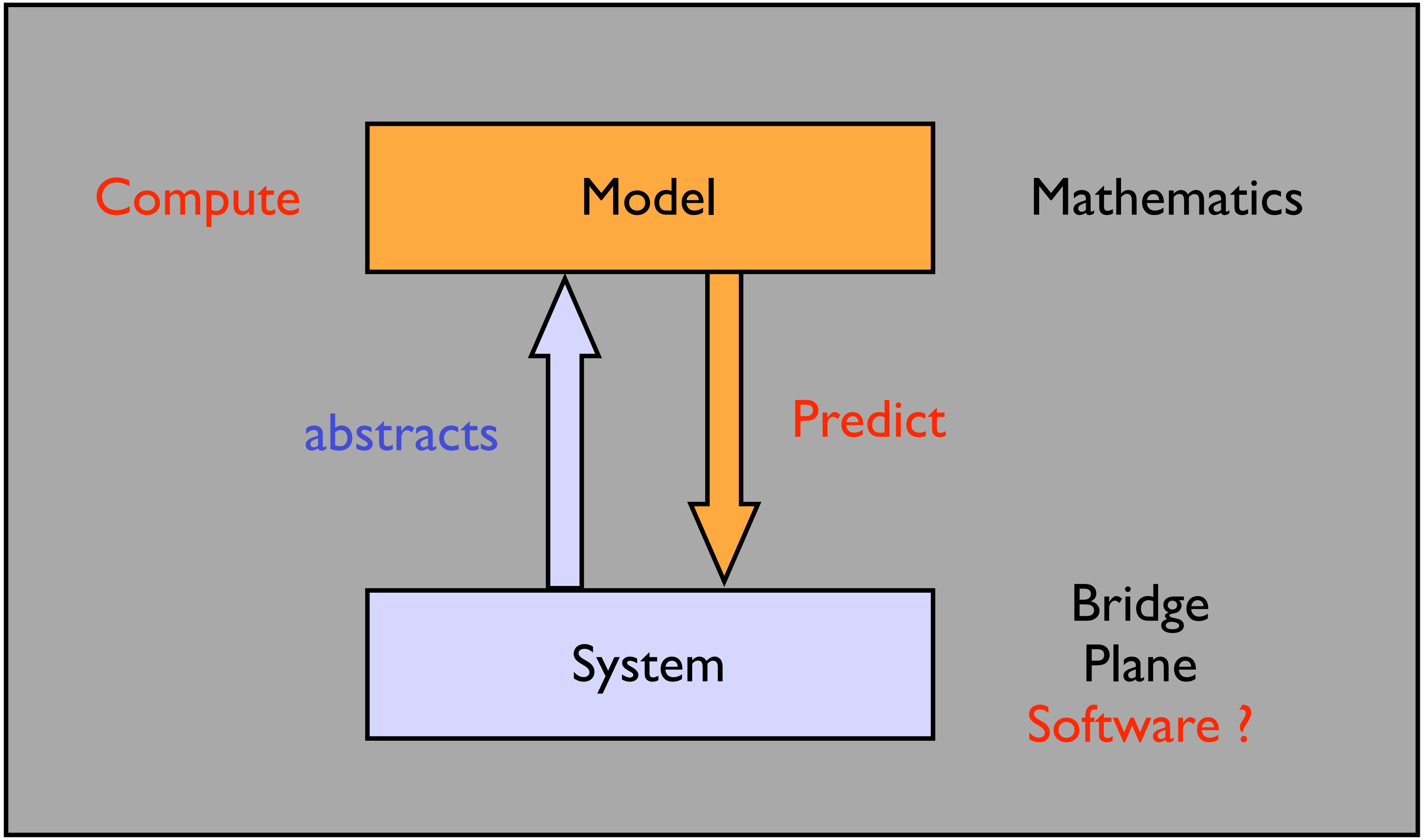
- Reactive systems are systems that maintain a continuous interaction with their environment, and they usually have several of the following properties:
  - they are non-terminating systems (processes);
  - they have to respect or enforce real-time properties;
  - they have to cope with concurrency (several processes are executing concurrently);
  - they are often embedded into an complex (continuous) and safety critical environments.
- ... as a result: the specifications that have to meet ES are often very complex and as a result ES are difficult to design correctly ! Furthermore, testing them is difficult: the environment in which they are embedded does not preexist or/and is difficult to simulate (e.g. rocket, medical equipment, ...), and even when errors are found, their diagnostic is diffult, we may not be able to replay the error.

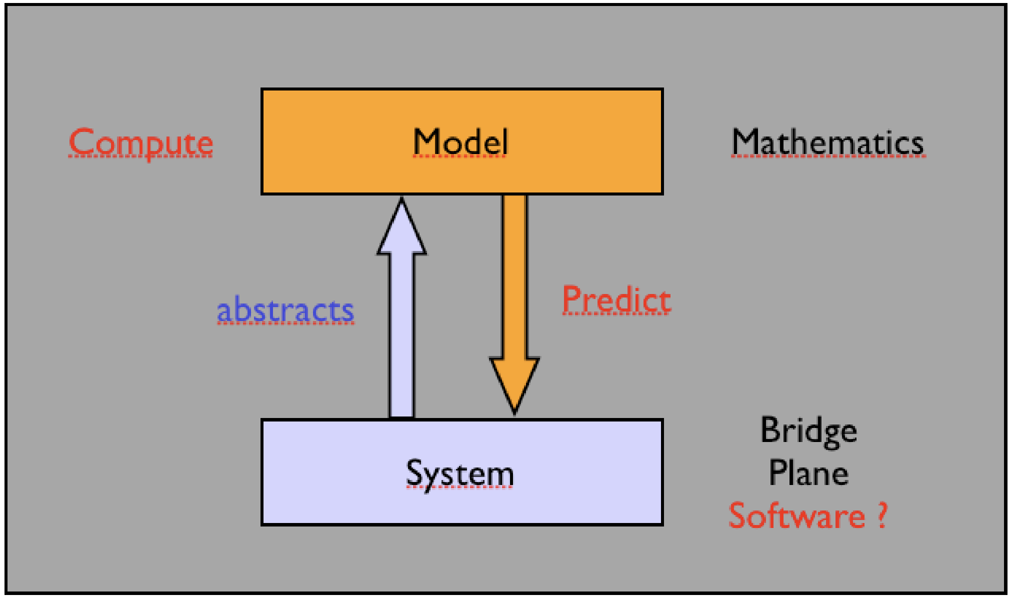
Need for verification

- ... as they are difficult to develp correctly !
- ... and often safety critical !

$\Rightarrow$ we should verify them !

How do we cope with complexity in science ?





- Model construction: capture the essential aspects of the system (sometimes automatically);
- Model verification: algorithms to analyze models.

CAV of reactive systems

- Model-Checking: does M logically entails Φ ?

  Clarke, Emerson and Sifakis received the 2008 Turing Award for their seminal works on the subject.

- M describes what happens during the (infinite) execution of the system (environment+program).

  M is usually given as a finite transition system.

- Φ is a property that refers to the entire computation: we are interested in temporal behaviors of the system.

  Φ is often expressed using a temporal logic.

- quite different from traditional (historical) approach to verification in CS where focus was on input-output behavior of programs, and as a consequence specifications were given as Pre-Post conditions.

Plan of the talk

- Reactive and Embedded Systems
- Modeling with Communicating Finite State Machines
- Modeling with Timed Automata
- Modeling with Hybrid Automata

Models for reactive systems : Communicating Finite State Machines and Temporal Logic

Models = set of traces

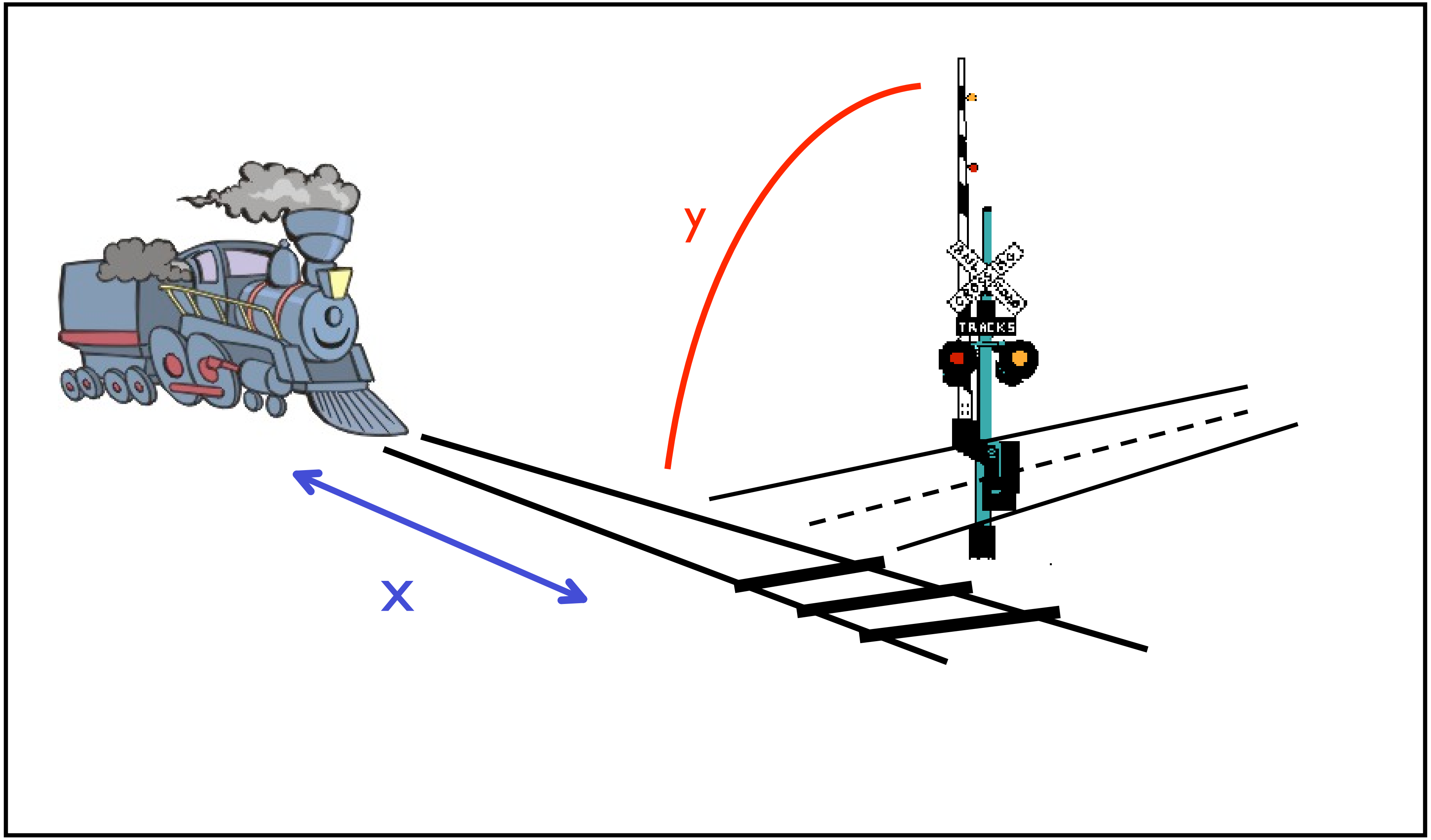
- Traces
  = infinite sequences of pairs state-event
  $s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow \ldots \rightarrow s_n \rightarrow a_n \rightarrow \ldots$
- each s_i is a subset of P (a finite set of propositions over the system);
- each a_i is an element of Σ, a finite set of events;
- Semantics of the system = (infinite) set of traces = ω-regular language.

- Properties of a reactive can also be expressed as ω-regular languages
- Verification of finite-state reactive systems = manipulate, compare, test properties of ω-regular languages;
- There exists a well-established and rich theory on which CAV is based:
  - Temporal logics [Pnu77];
  - Büchi automata [Büc62];
  - Classical theories [Kam68].

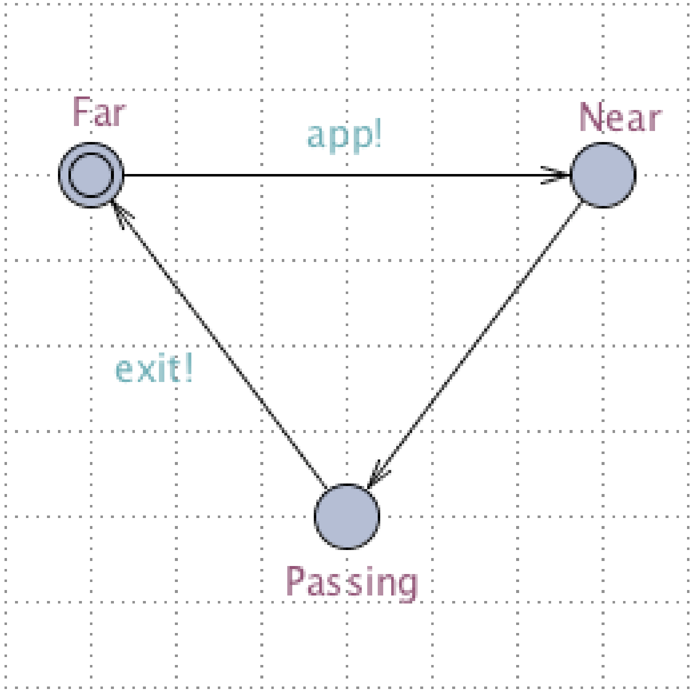
Communicating Finite State Machines (a.k.a. Büchi Automata)

CFSM are finite state machines (also called finite state automata) that communicate via shared events.

A running example

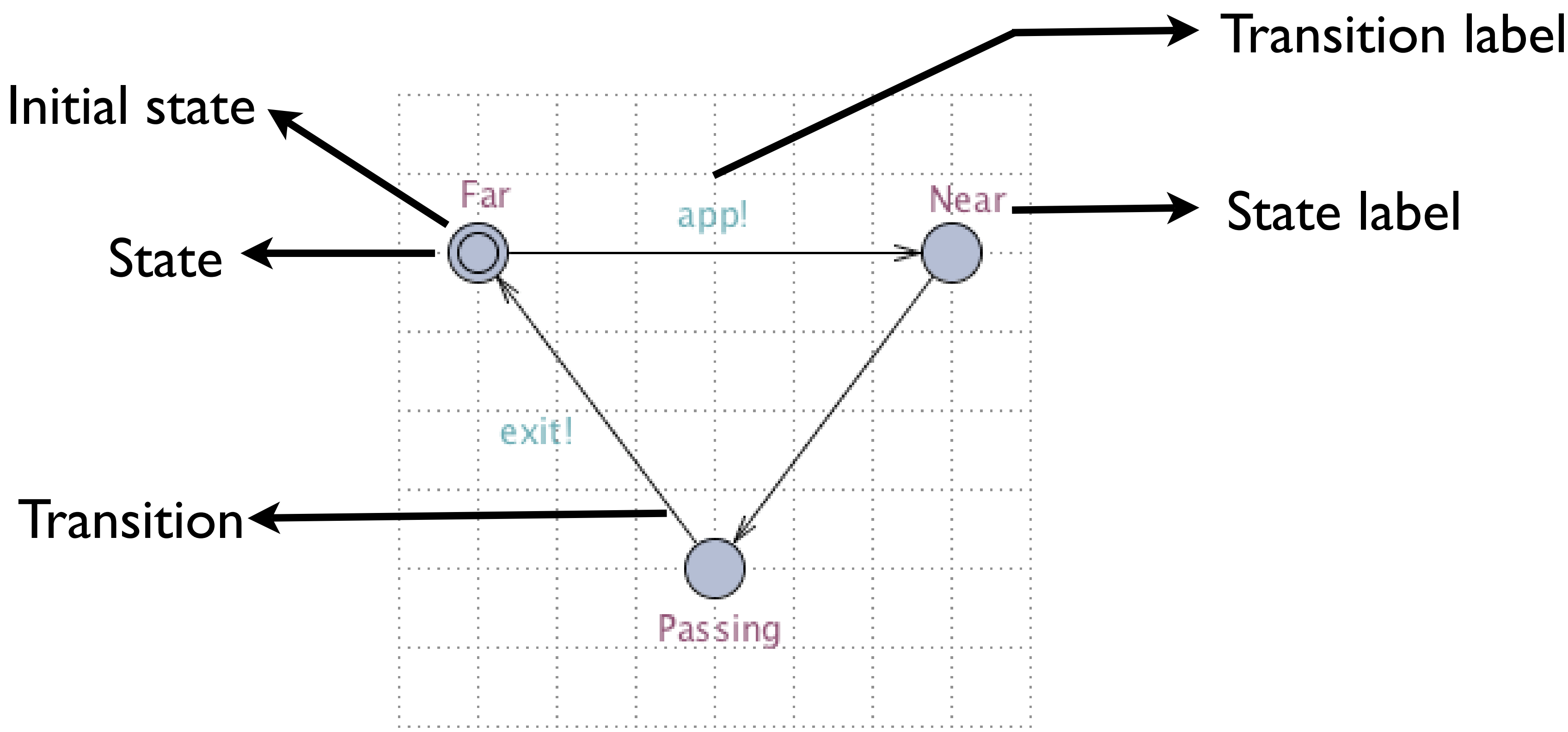


Train model



Snapshot from UppAal
http://www.uppaal.com

Train model



Snapshot from UppAal
http://www.uppaal.com

Syntax

- Formally: $A=(Q,Q_0,\Sigma,E,P,L,F)$ where:
  - Q is a finite set of states (locations); Q_0 is the subset of initial states;
  - Σ is a finite set of transition labels (events, actions); $E \subseteq Q \times \Sigma \times Q$ is the transition relation;
  - P is a finite set of propositions; $L : Q \rightarrow 2^P$ is a labelling function, this function defines state labels;
  - $F \subseteq 2^Q$ is a set of sets of accepting states (generalized Büchi condition).

Semantics

The automaton $A=(Q,Q_0,\Sigma,E,P,L,F)$ accepts the trace

$$s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow \ldots \rightarrow s_n \rightarrow a_n \rightarrow \ldots$$

iff there exists an infinite sequence of states

$$q_0 q_1 \ldots q_n \ldots$$

such that for any $i \geq 0$:

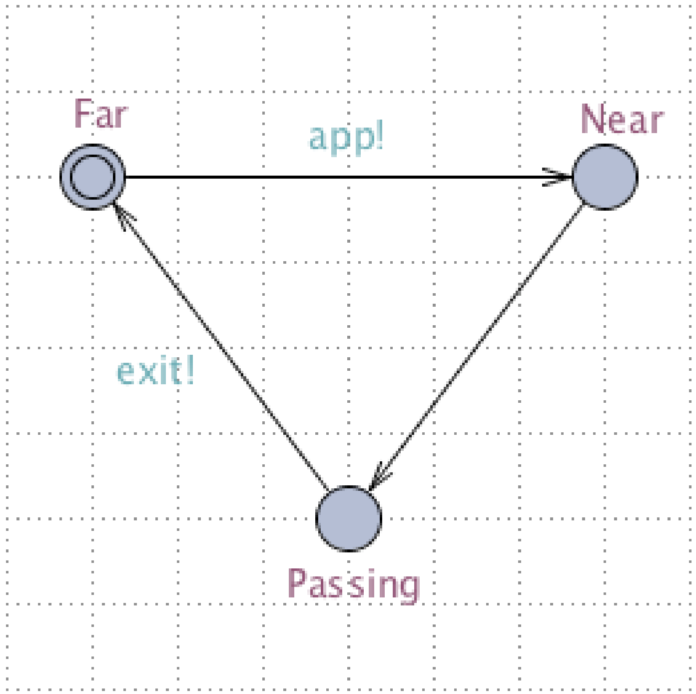
(1) $(q_i,a_i,q_{i+1}) \in E$, (2) $L(q_i)=s_i$, and

(3) for all $f \in F$ there exist infinitely many $j \geq 0$ such that

$q_j \in f$ (generalized Büchi condition).

Such a sequence is called an accepted run.

The language of a automaton A is the set of traces that A accepts. This set is noted **Lang**(A).



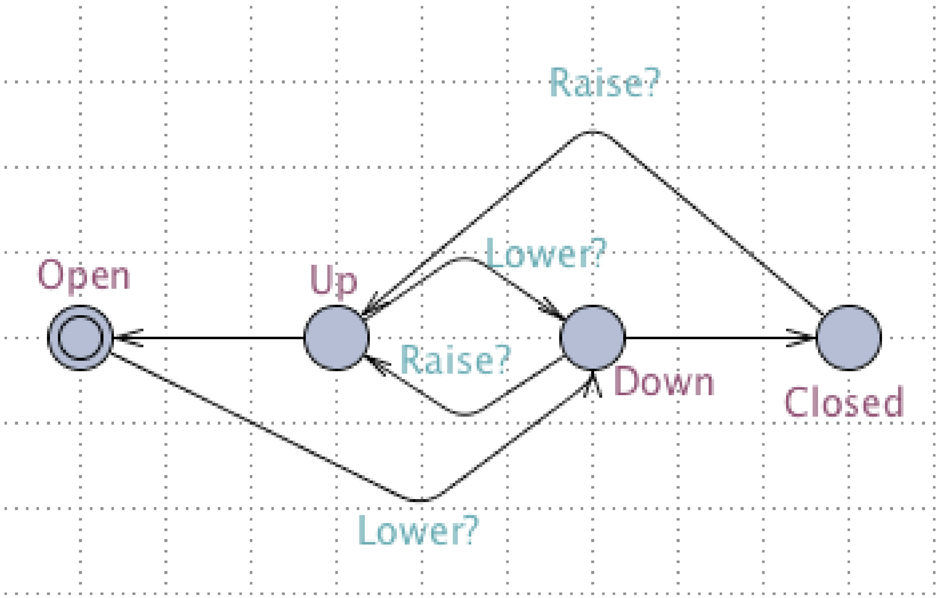
Here is one execution (the only in this special case) of the train:

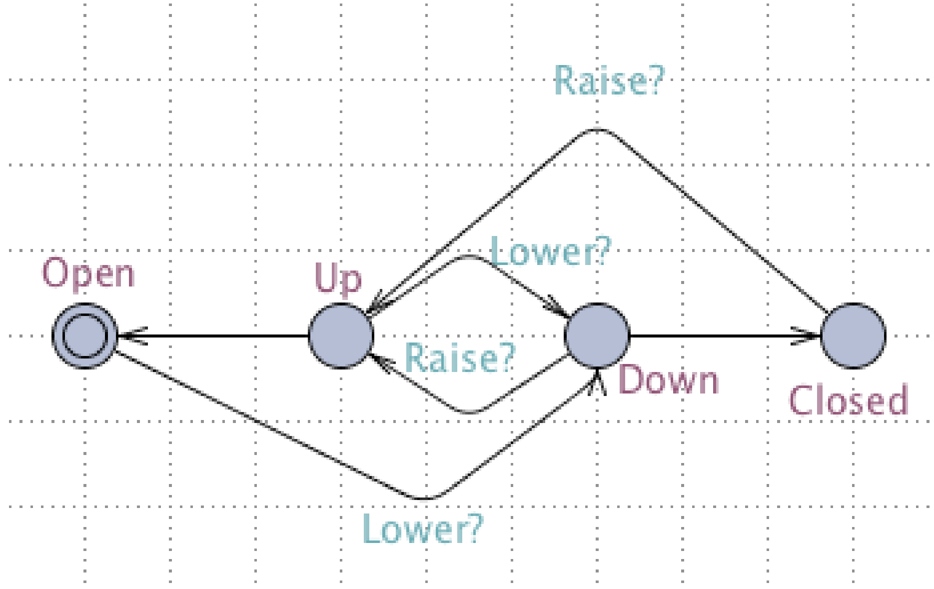
Far —App!→ Near —ε→ Passing —Exit!→ Far ...

So the language of this automaton is

{Far —App!→ Near —ε→ Passing —Exit!→ Far ...}

Gate model



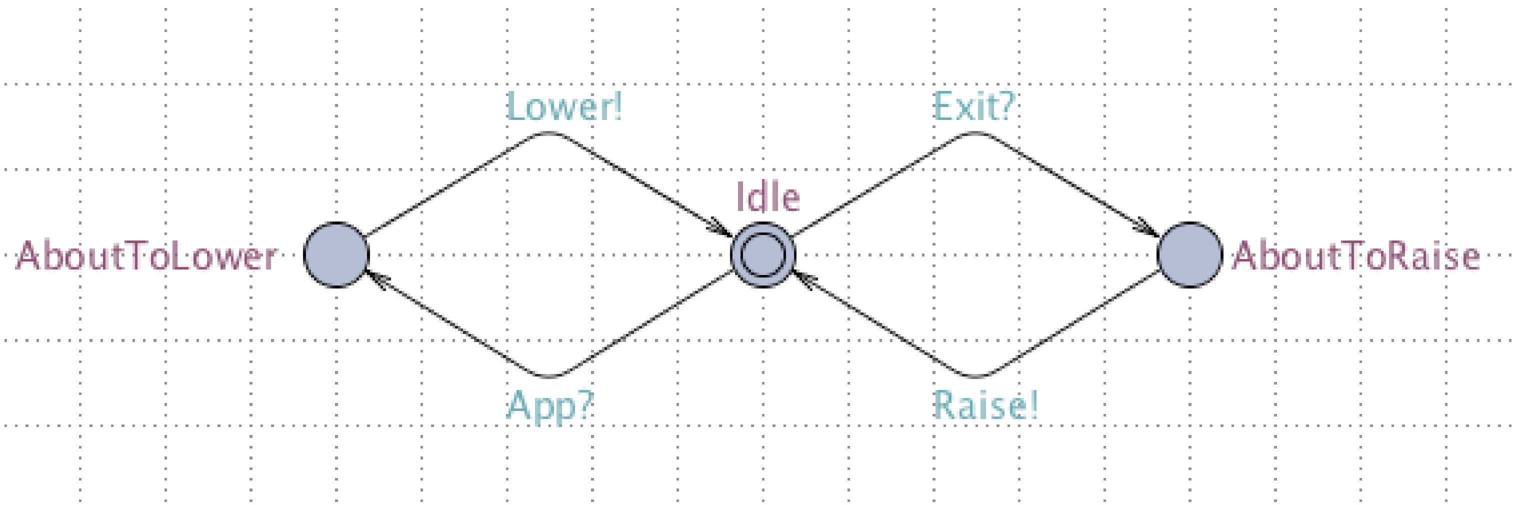


Open —Lower?→ Down —ε→ Closed —Raise?→ Up ...
Open —Lower?→ Down —Raise?→ Up —Lower?→ Down ...
...

The language of the gate is:

{Open —Lower?→ Down —ε→ Closed —Raise?→ Up ...,
Open —Lower?→ Down —Raise?→ Up —Lower?→ Down ...
...}

Controller model



Modeling reactive systems

- ➢ Finite state machine are building blocks that allow us to model components of complex systems.

- ➢ Systems are best modeled compositionally as a product of communicating finite state machines.

- ➢ We will define a synchronized product of finite state machines in which communication is performed via synchronization on common events

Product of two finite state machines

Let $A=(Q_a,Q_{a0},\Sigma_a,E_a,P_a,L_a,F_a)$ and let $B=(Q_b,Q_{b0},\Sigma_b,E_b,P_b,L_b,F_b)$ such that $Pa\cap Pb=\varnothing$. We define the product of A and B, noted $A\otimes B$, as the automaton $C=(Q,Q_0,\Sigma,E,P,L,F)$ where :

(1) $Q=Q_a x Q_b$ (2) $Q_0=Q_{a0}xQ_{b0}$ (3) $\Sigma=\Sigma_a\cup\Sigma_b$

(4) E contains $((q_{a1},q_{b1}),a,(q_{a2},q_{b2}))$ iff one of the three following conditions holds:

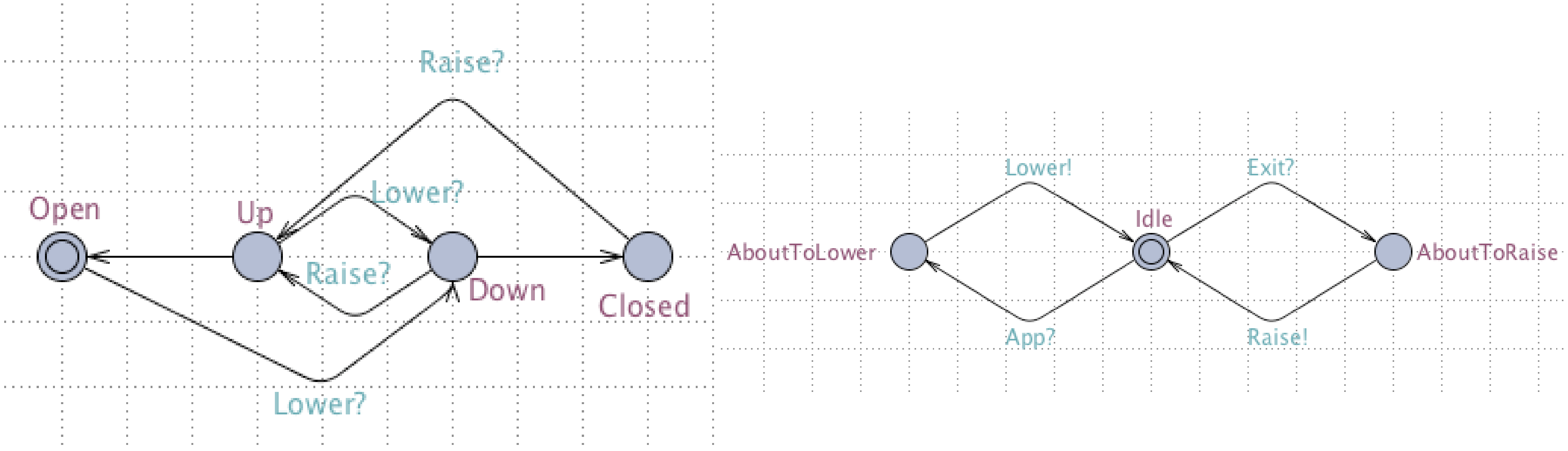
- $a\in\Sigma_a$, $a\notin\Sigma_b$, $(q_{a1},a,q_{a2})\in E_a$, $q_{b1}=q_{b2}$
- $a\in\Sigma_b$, $a\notin\Sigma_a$, $(q_{b1},a,q_{b2})\in E_b$, $q_{a1}=q_{a2}$
- $a\in\Sigma_a$, $a\in\Sigma_b$, $(q_{a1},a,q_{a2})\in E_a$, $(q_{b1},a,q_{b2})\in E_b$

(5) $P=P_a\cup P_b$

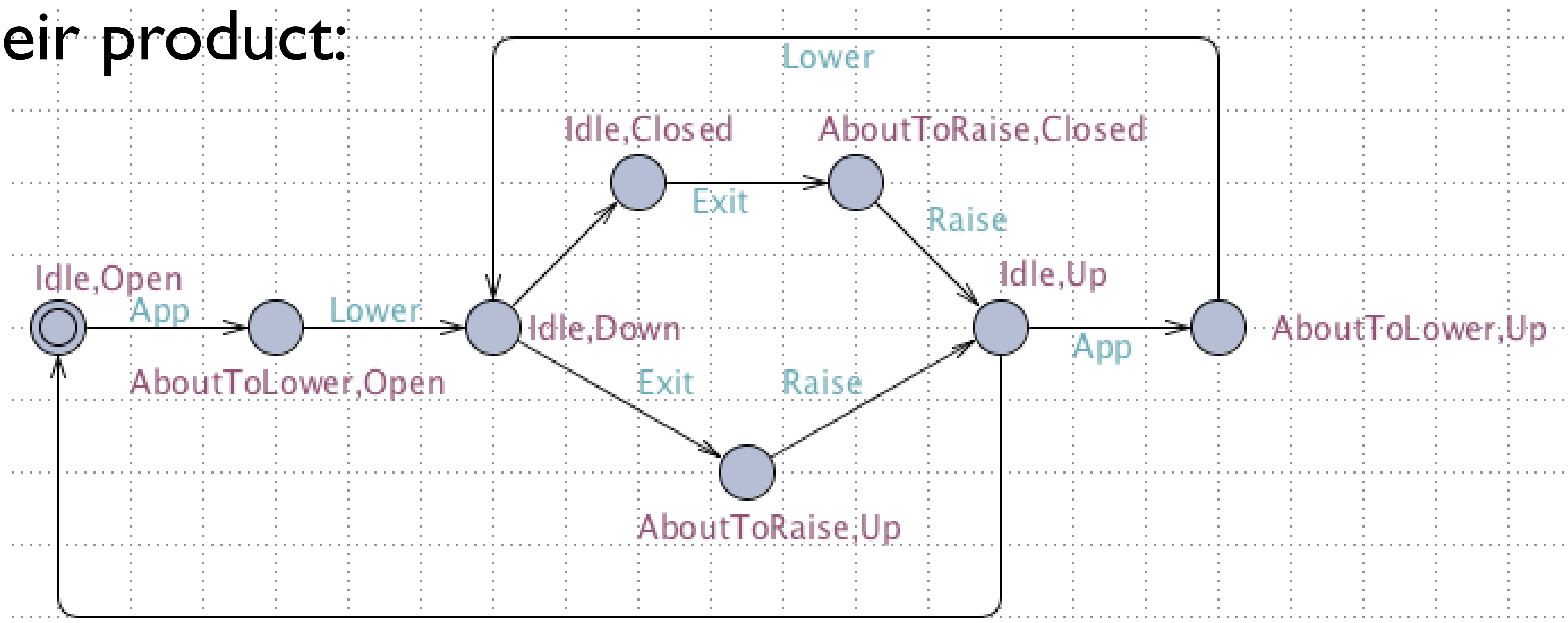
(6) for any $(q_a,q_b)\in Q$, $L((q_a,q_b))=L_a(q_a)\cup L_b(q_b)$

(7) $F=\{ \{(q_a,q_b) \mid q_a \in f_a\} \mid f_a \in F_a\} \} \cup \{ \{(q_a,q_b) \mid q_b \in f_b\} \mid f_b \in F_b\} \}$

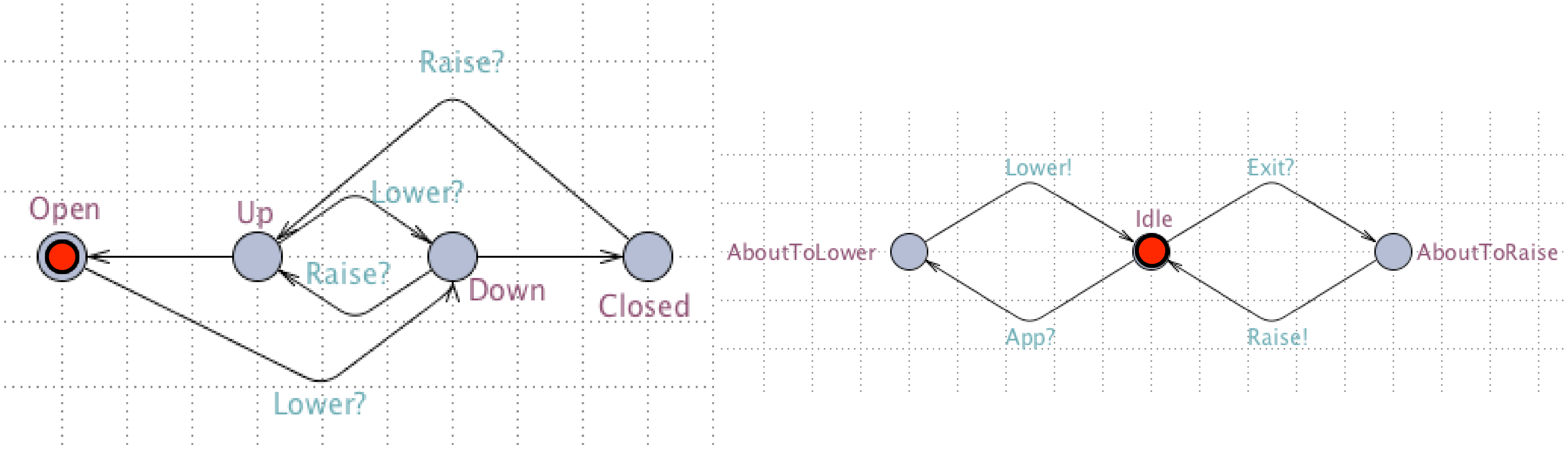
Example: the product of Gate and Controller



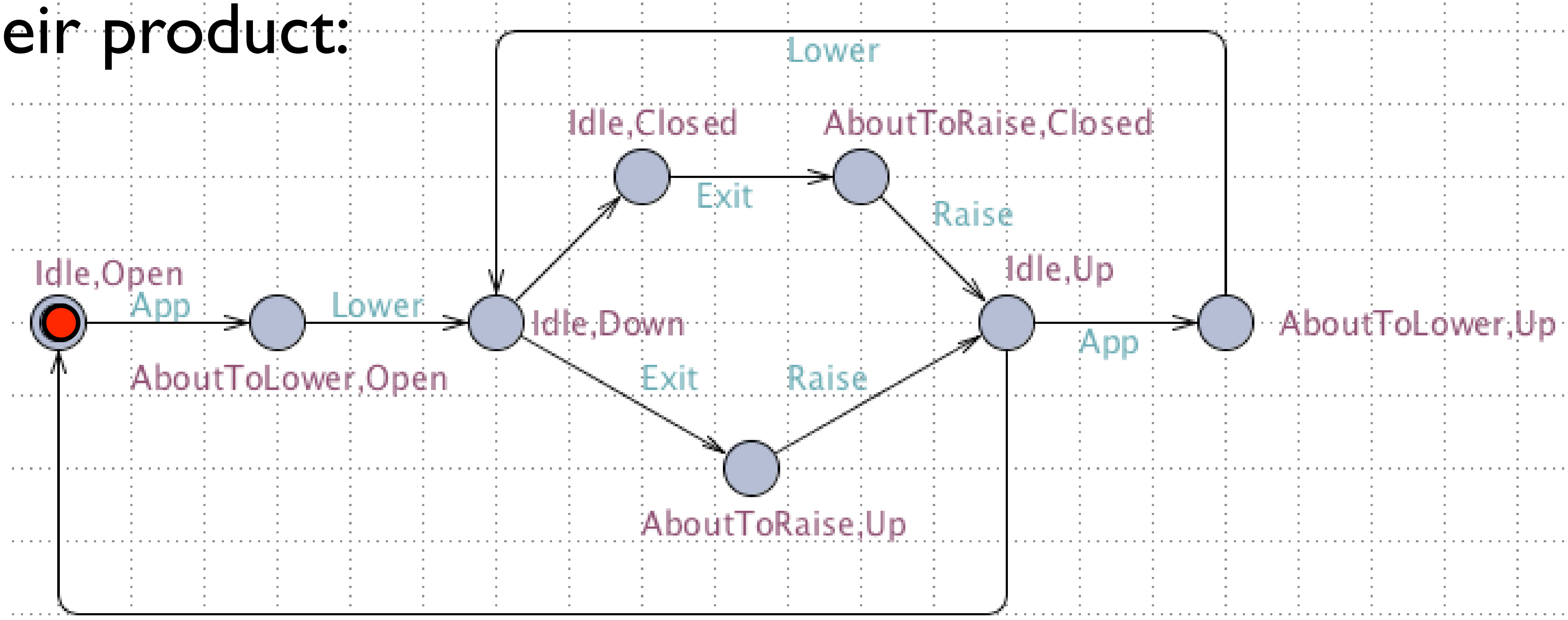
Their product:



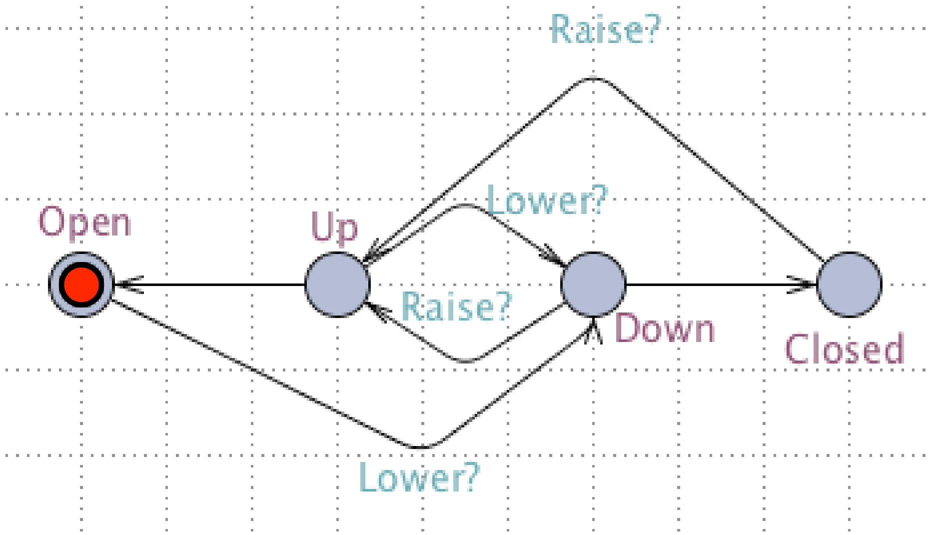
Example: the product of Gate and Controller

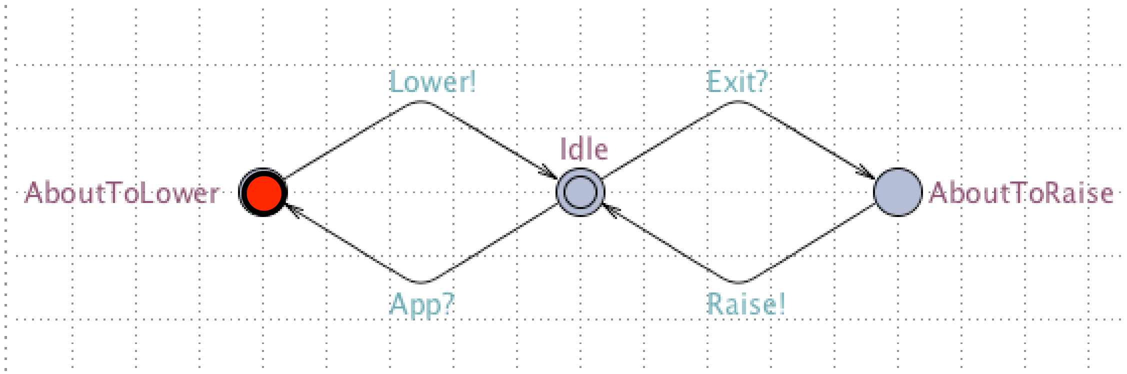


Their product:

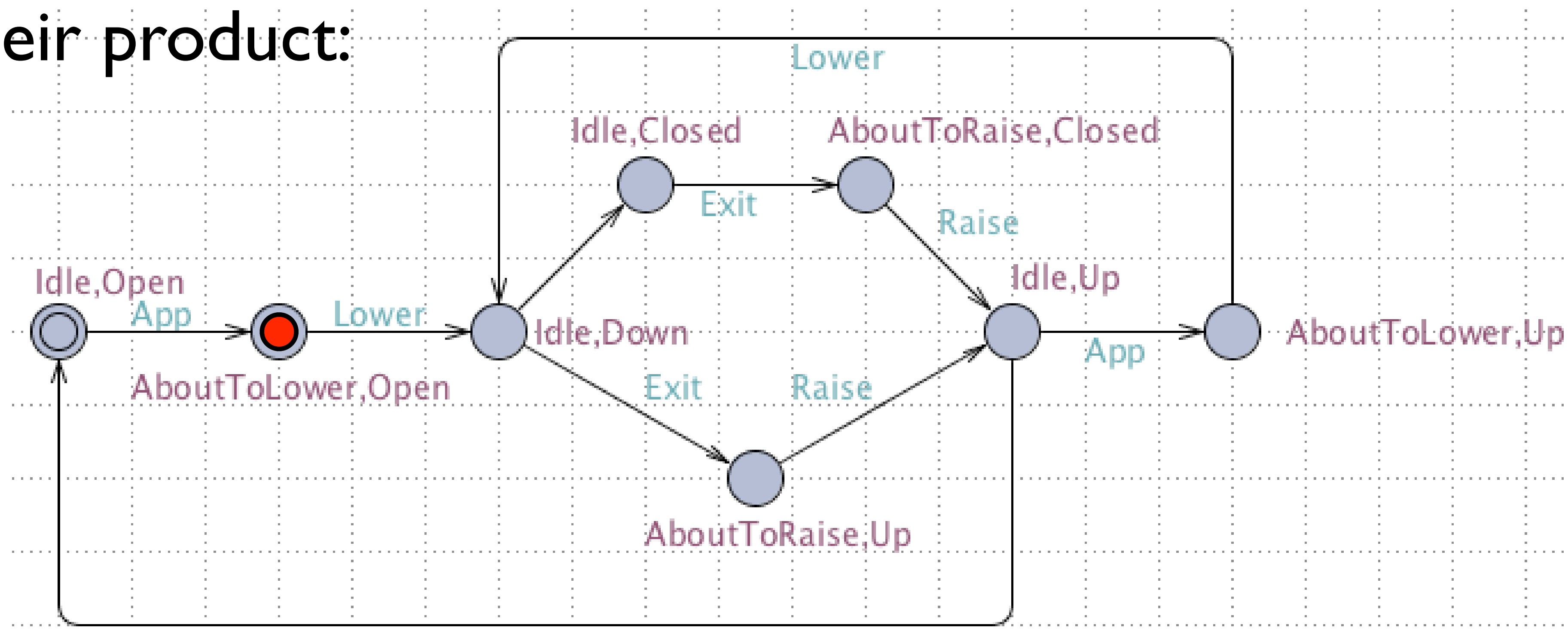


Example: the product of Gate and Controller

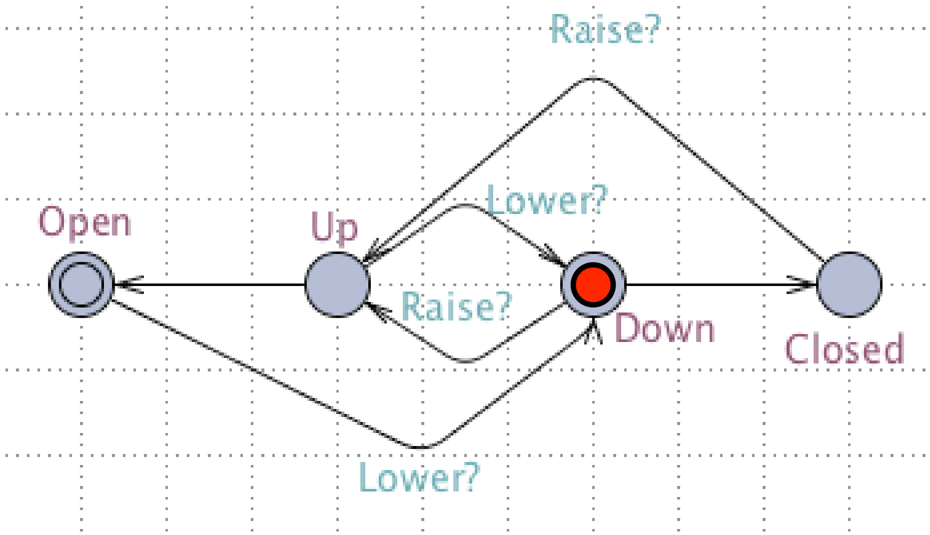


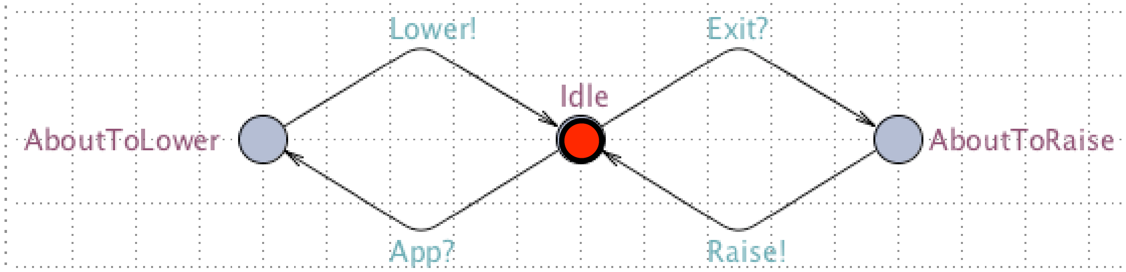


Their product:

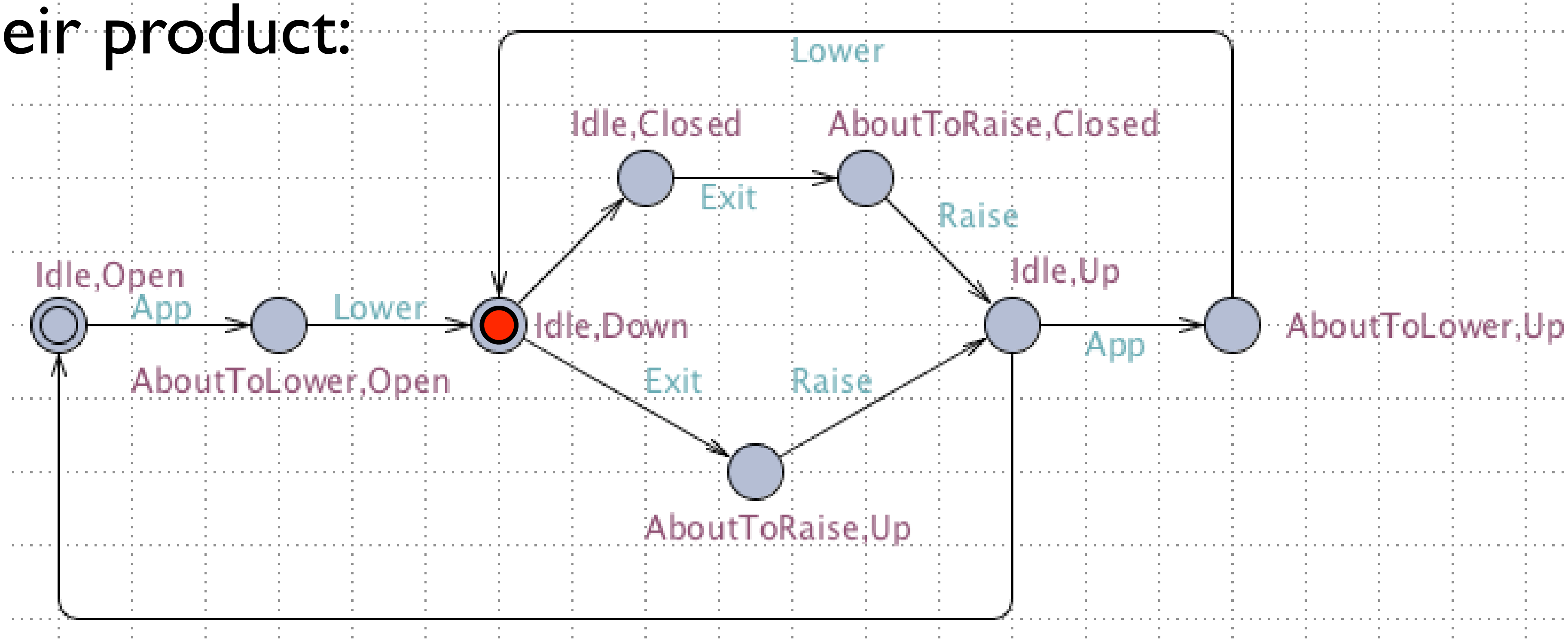


Example: the product of Gate and Controller

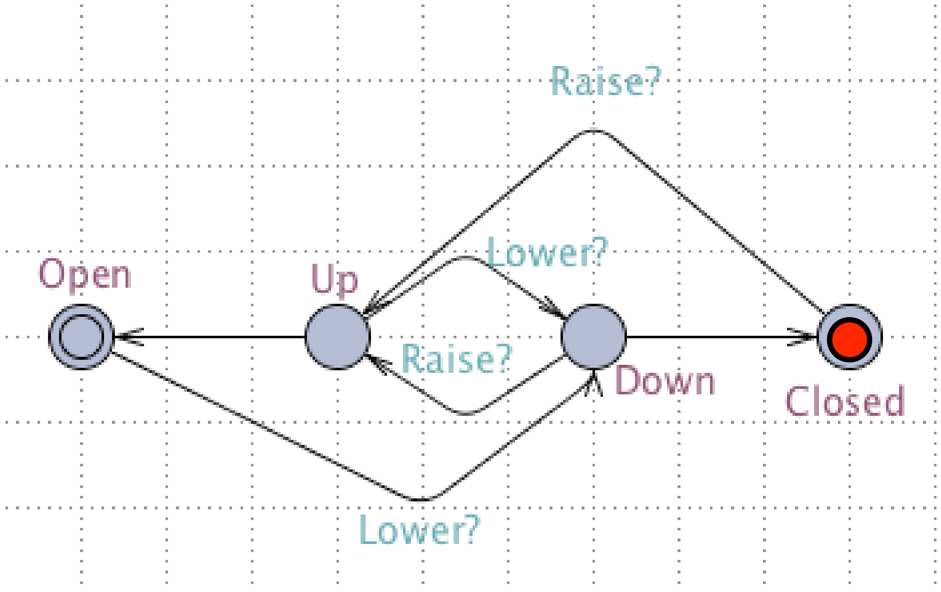


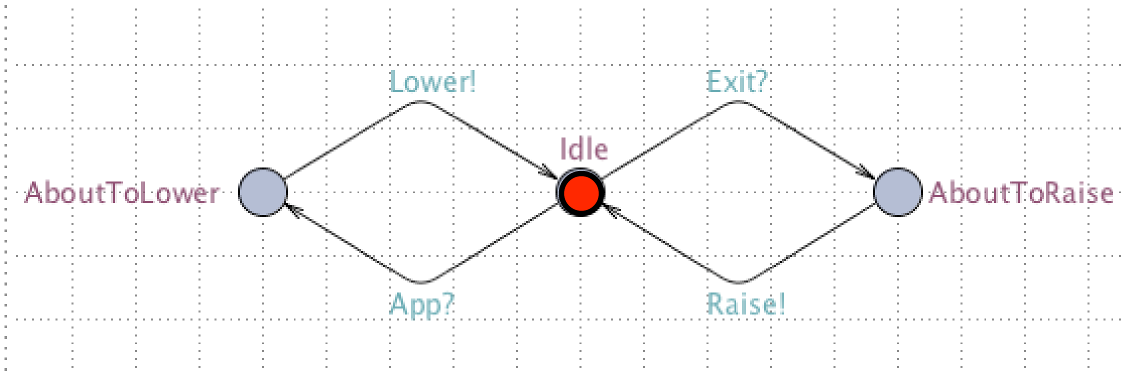


Their product:

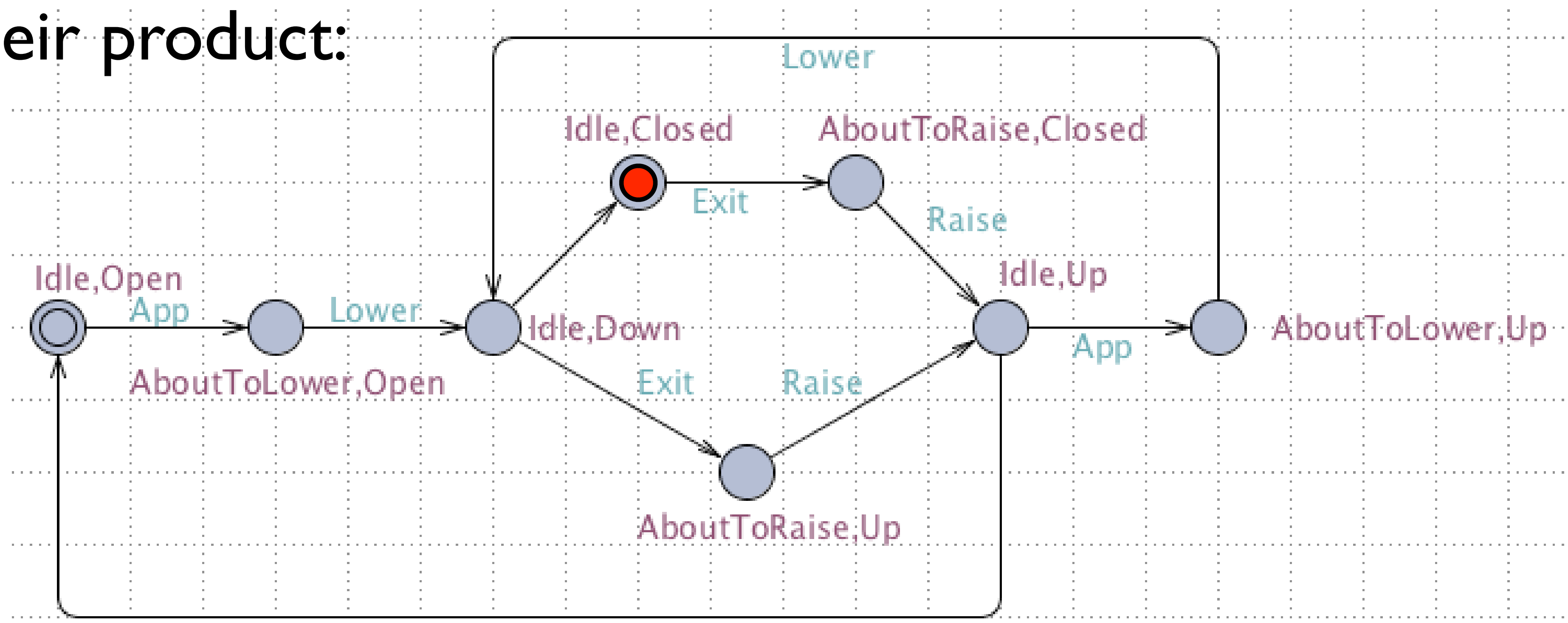


Example: the product of Gate and Controller

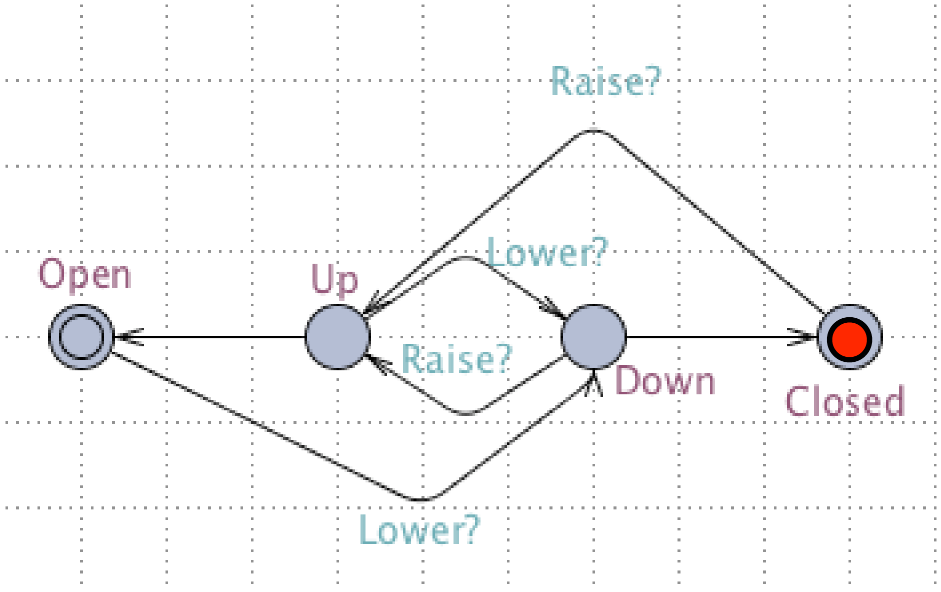


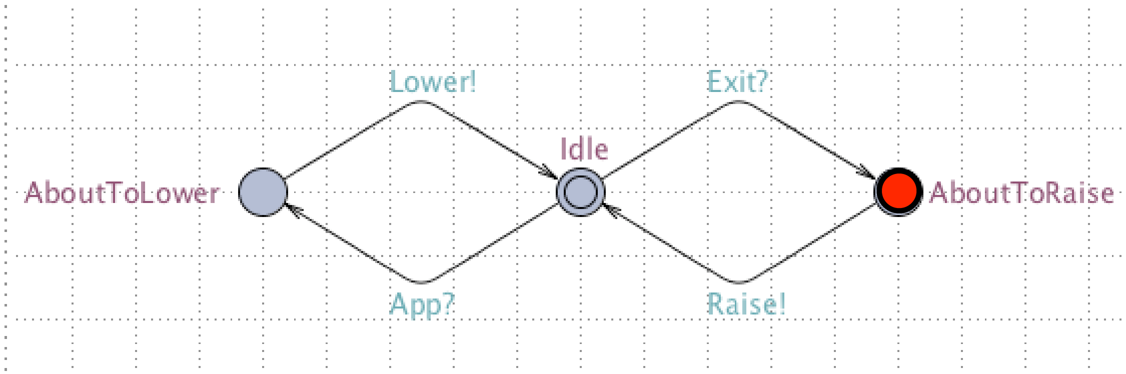


Their product:

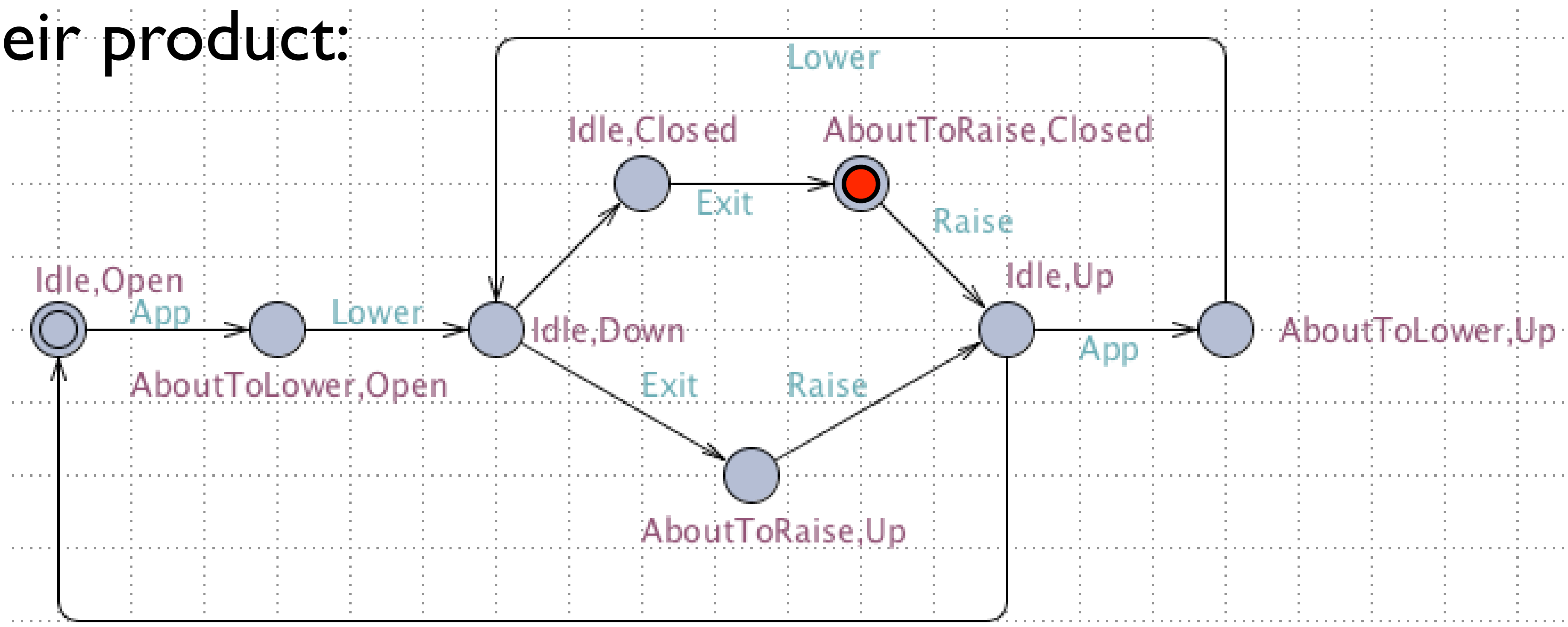


Example: the product of Gate and Controller

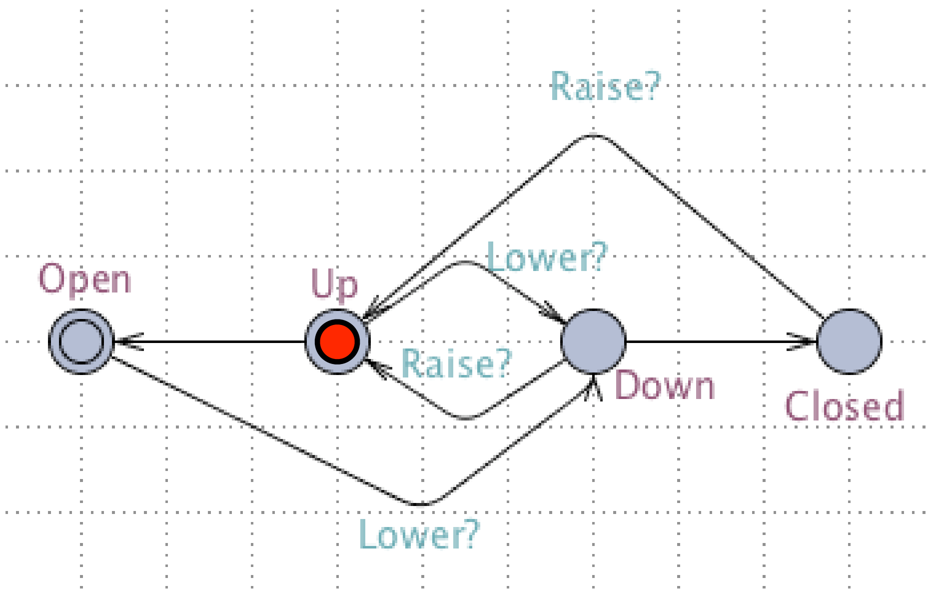


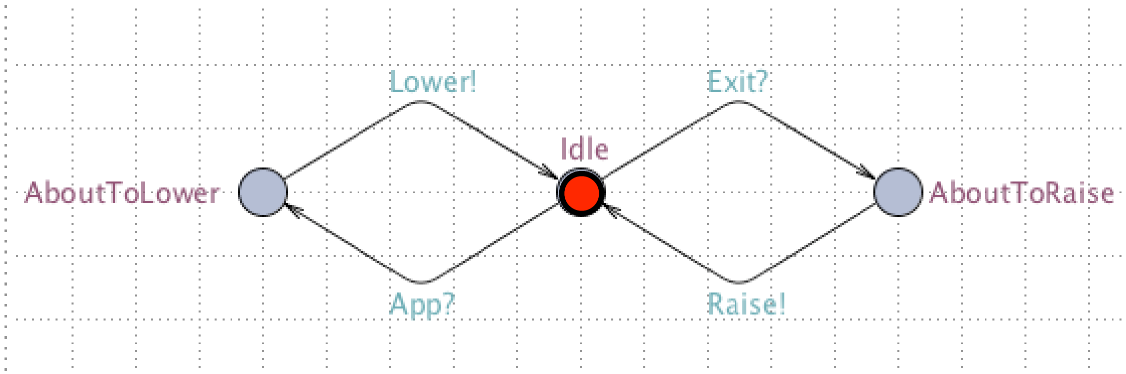


Their product:

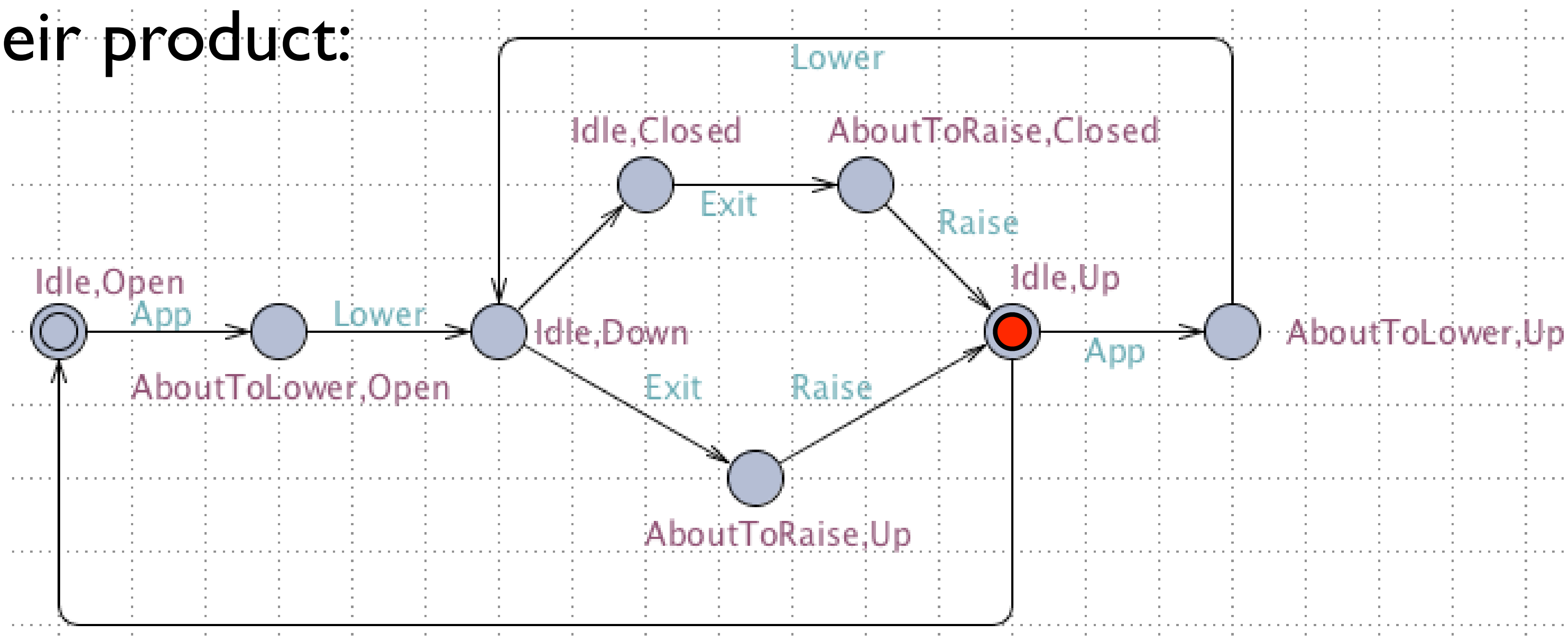


Example: the product of Gate and Controller

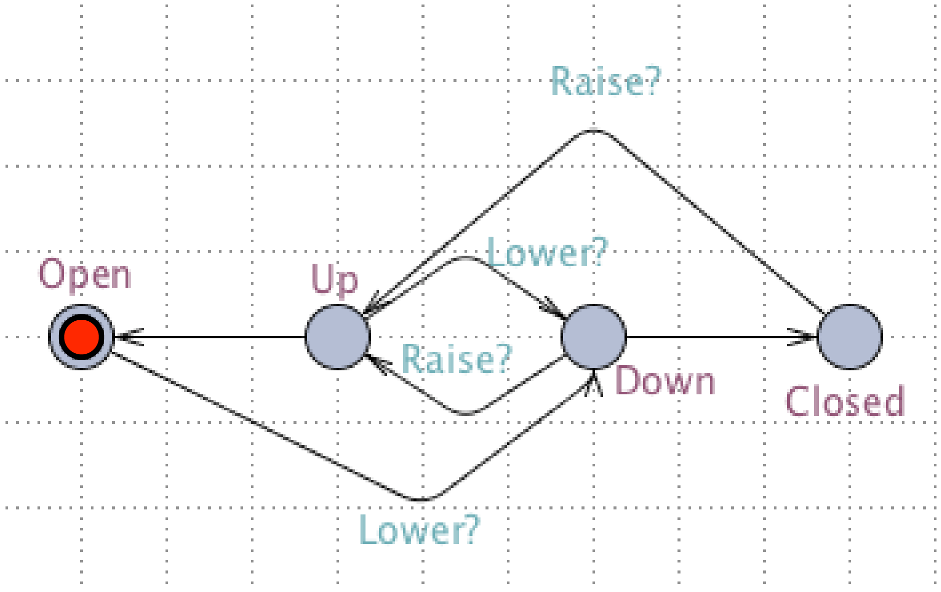


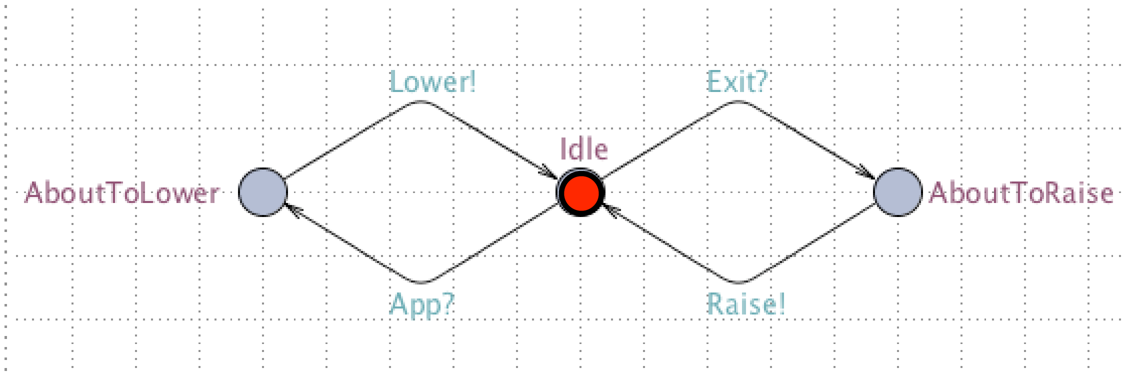


Their product:

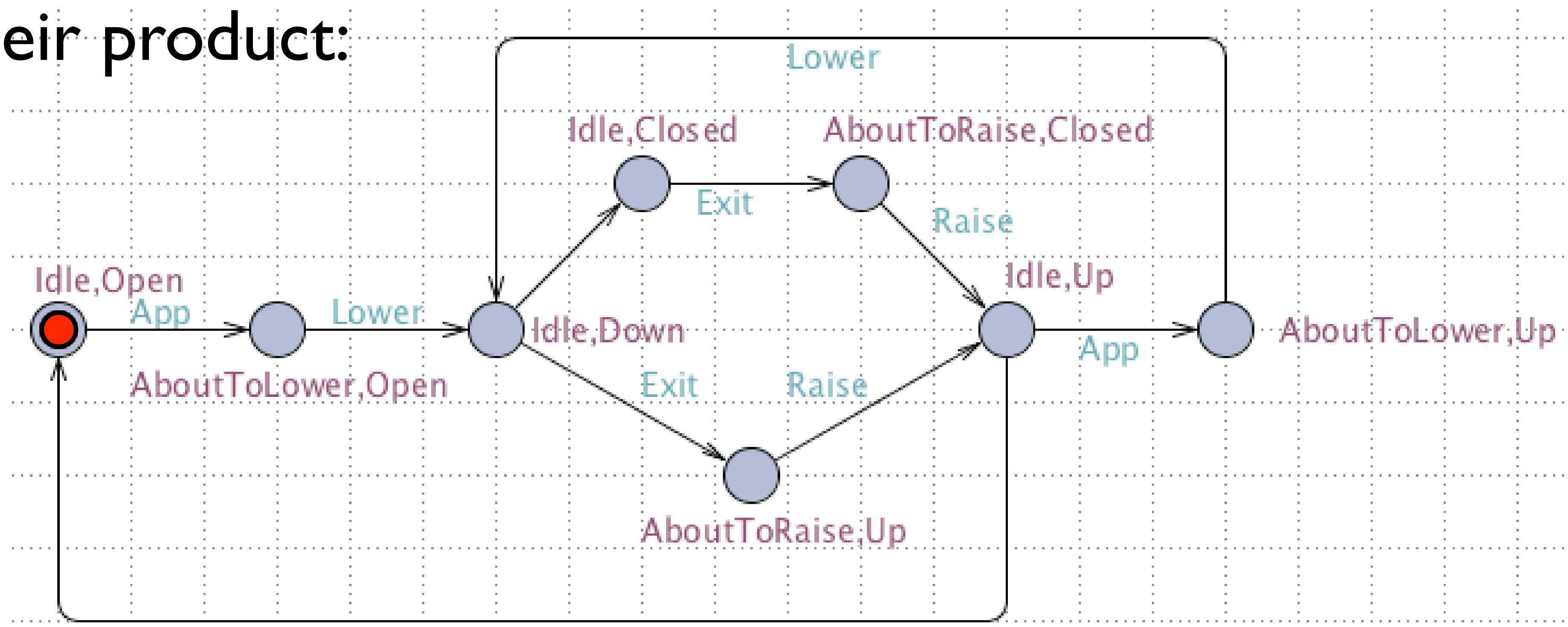


Example: the product of Gate and Controller

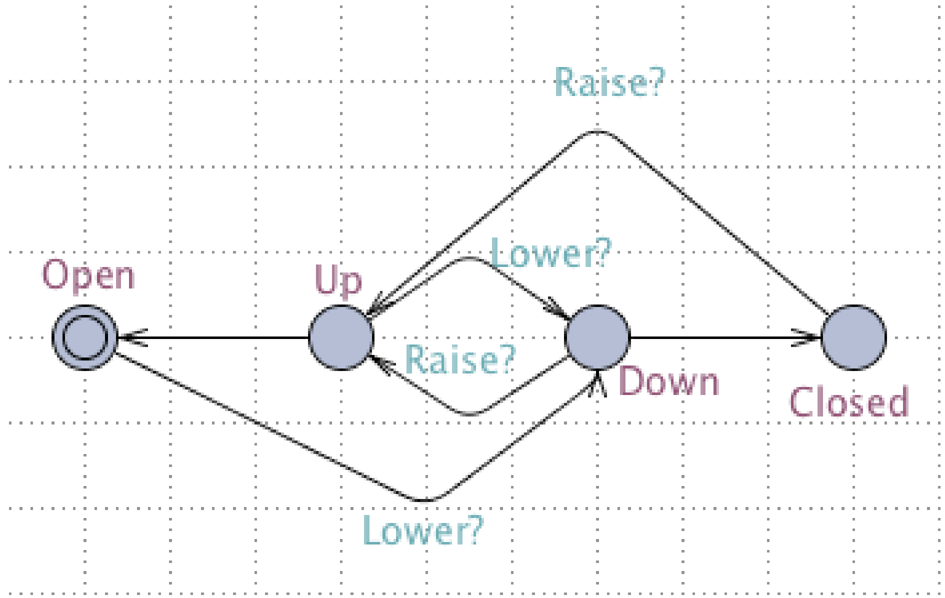




Their product:



Generalized Büchi condition
An example



Let us consider our example with the following **generalized Büchi condition**: F={{Open},{Closed}}.

This condition excludes words that imposes to its runs to loop for ever between Up and Down.

So, the word "Open Lower? (Down Raise? Up Lower?)$^\infty$" is **not** part of the language of the automaton with this generalized Büchi condition.

How to express specifications of reactive systems ?

Linear Temporal Logic

The syntax of the logic LTL is given by the following grammar:

$$\Phi ::= p \mid \neg\Phi_1 \mid \Phi_1 \vee \Phi_2 \mid X\,\Phi \mid \Phi_1 U \Phi_2$$

where $\Phi_1, \Phi_2 \in \Phi$.

Formula of LTL are evaluated over states of traces.

LTL - Semantics

Let $\eta = s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow \ldots \rightarrow s_n \rightarrow a_n \rightarrow \ldots$ be a infinite trace over the set of propositions P (and events Σ). We refer to s_i by using the notation $\eta(i)$.

For any $i \geq 0$, we have :

- $\eta(i)$ models p iff $p \in \eta(i)$
- $\eta(i)$ models $\neg\Phi_1$ iff $\eta(i)$ does not model Φ_1
- $\eta(i)$ models $\Phi_1 \vee \Phi_2$ iff
  $\eta(i)$ models Φ_1 or $\eta(i)$ models Φ_2
- $\eta(i)$ models $X\ \Phi$ iff $\eta(i+1)$ models Φ
- $\eta(i)$ models $\Phi_1 U \Phi_2$ iff there exists $j \geq i$, such that
  $\eta(j)$ models Φ_2 and
  for all k, $i \leq k < j$, $\eta(k)$ models Φ_1

LTL - Semantics (cont'd)

A formula Φ is true over a trace η

iff

"$\eta(0)$ models Φ"

A formula Φ is true over a set of traces H iff for all $\eta \in$ H, Φ is true over η.

LTL - Abbreviations

The following abbreviations are useful:

$F\ \Phi \equiv \text{True}\ U\ \Phi$, "Eventually Φ".

$G\ \Phi \equiv \neg\ F\ \neg\ \Phi$, "Always Φ".

Examples of properties expressed in LTL

The gate should always be closed when the train is within the crossing :

G (past → closed)

At any time, the gate will eventually be open:

G F open

The LTL model-checking problem

Given a product of n CFSMs $M_1 \otimes M_2 \otimes \ldots \otimes M_n$, given a formula of LTL Φ, determine if the set of traces defined by $M_1 \otimes M_2 \otimes \ldots \otimes M_n$ satisfies the formula Φ.

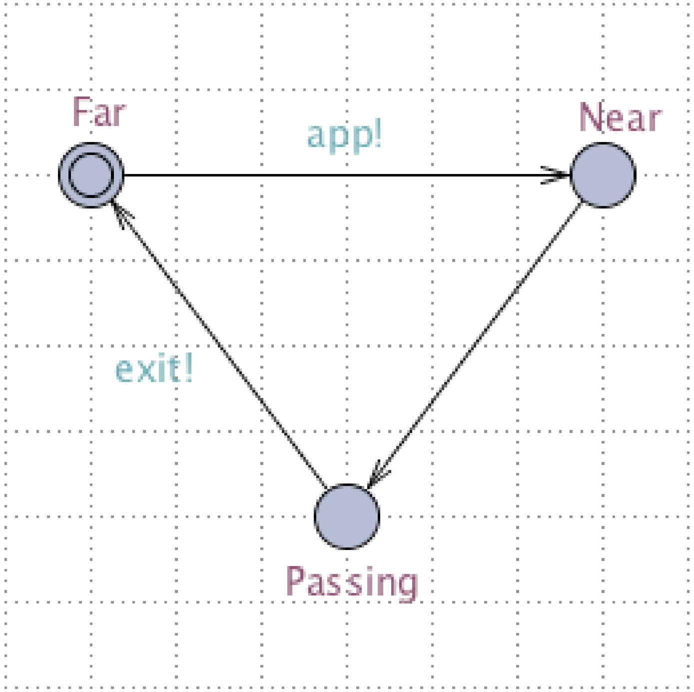
There are algorithms and implementations that solve this problem but it is problem is provably hard: it is complete for **PSpace**.

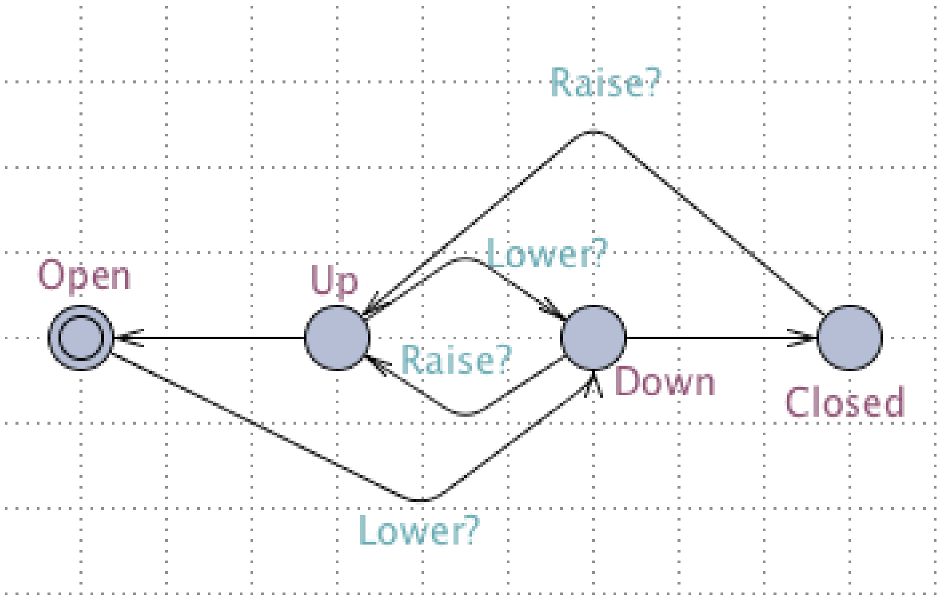
Question : are the two following formulas

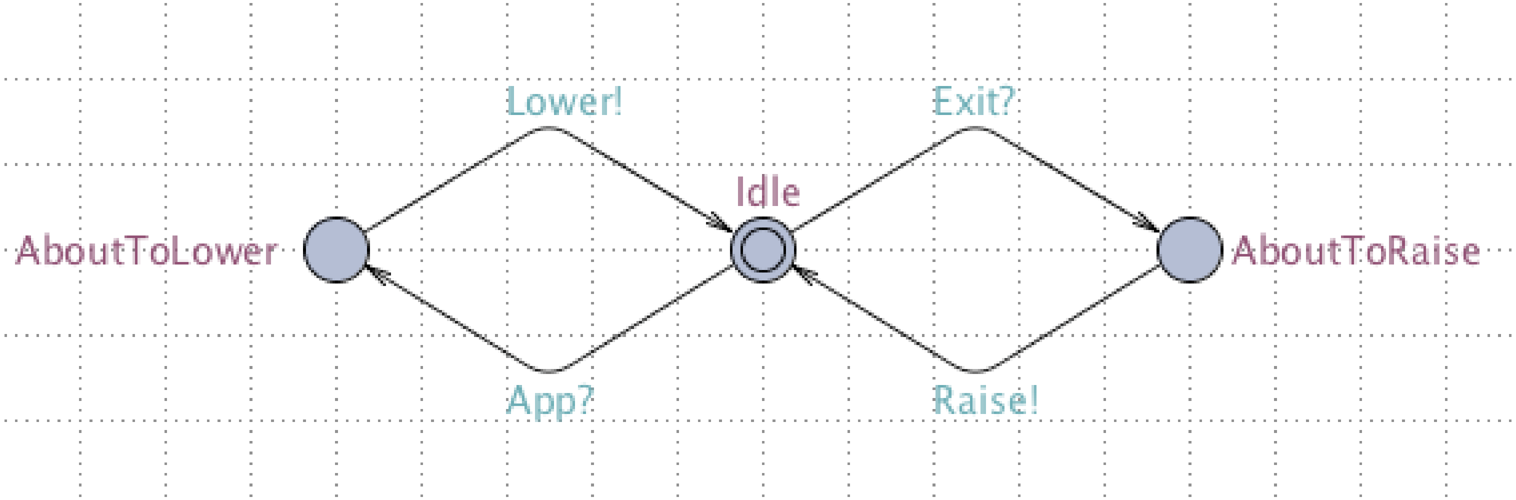
$$G\ (\ past \rightarrow closed\ )$$

$$G\ F\ open$$

true in our model of the rail-road crossing system ?

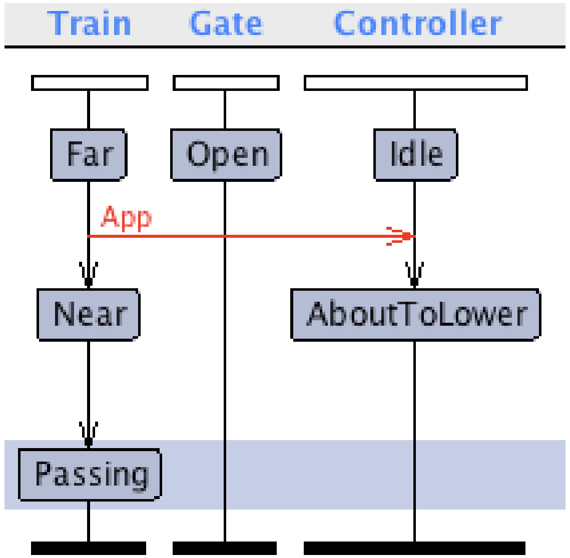






UppAal Demo (FSM-Train-Simple)

Our model of the rail-road crossing system is not correct !



Is the controller strategy that we propose flawed ?

Is your model too coarse ? not precise enough ?

Plan of the talk

- Reactive and Embedded Systems
- Modeling with Communicating Finite State Machines
- Modeling with Timed Automata
- Modeling with Hybrid Automata

Models = set of timed traces

A timed trace is an infinite sequence of the form

$$s_0 \rightarrow (a_0,t_0) \rightarrow s_1 \rightarrow (a_1,t_1) \rightarrow s_2 \rightarrow (a_2,t_2) \rightarrow \dots \rightarrow s_n \rightarrow (a_n,t_n) \rightarrow \dots$$

where :

-each s_i is a subset of the set of propositions P;

-each a_i is an element of Σ, the set of events;

-each t_i is a positive real number, and we verify :

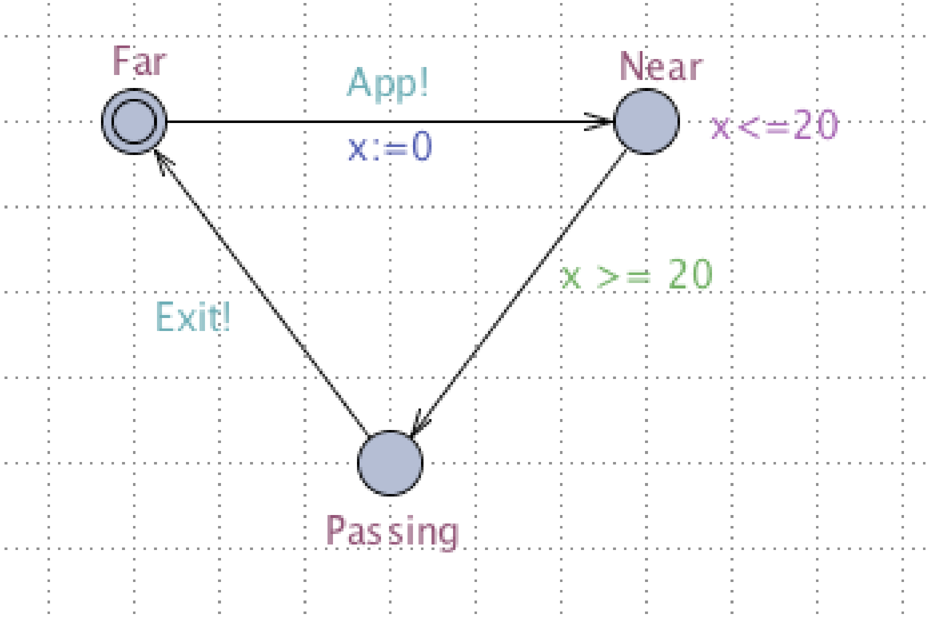
(1) for each $i \geq 0$: $t_i \leq t_{i+1}$ (monotonicity) and

(2) for any positive real r, there exists a position $i \geq 0$ such that $t_i \geq r$ (non-zenoness).

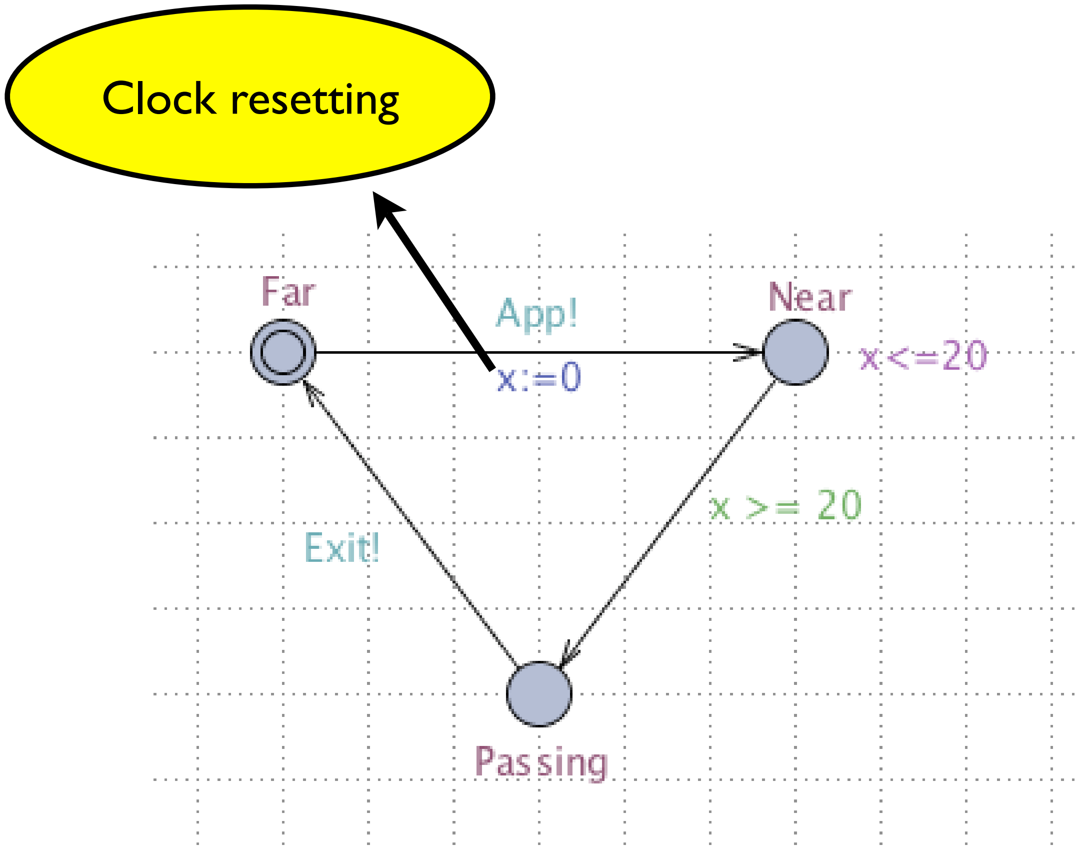
Timed Automata [AD94]

- Timed Automata = Finite State Machines + Clocks;
- Clocks = continuous variables that count time;
- Operations on clocks = resetting and comparison to constants.

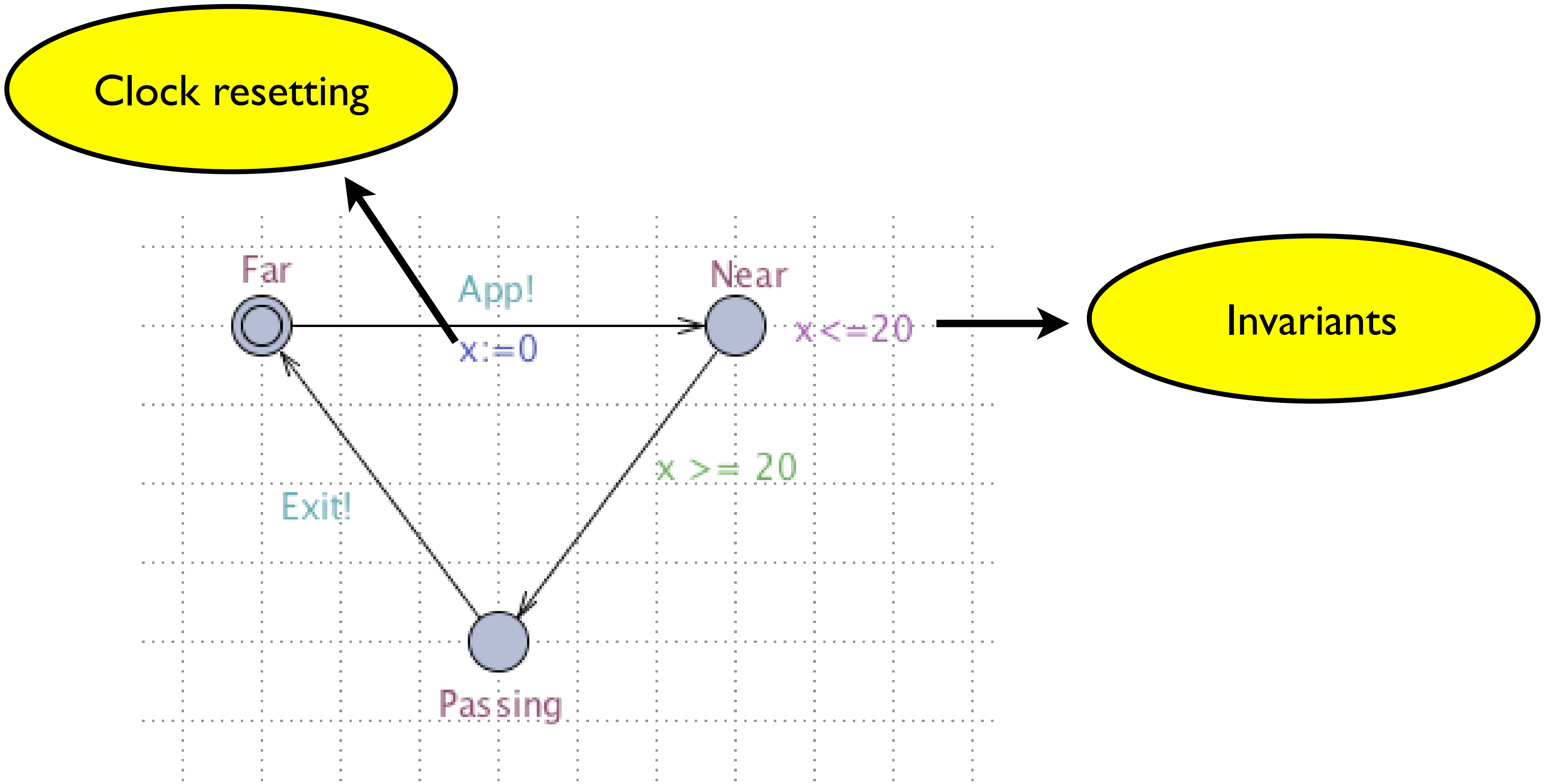
TA for the train



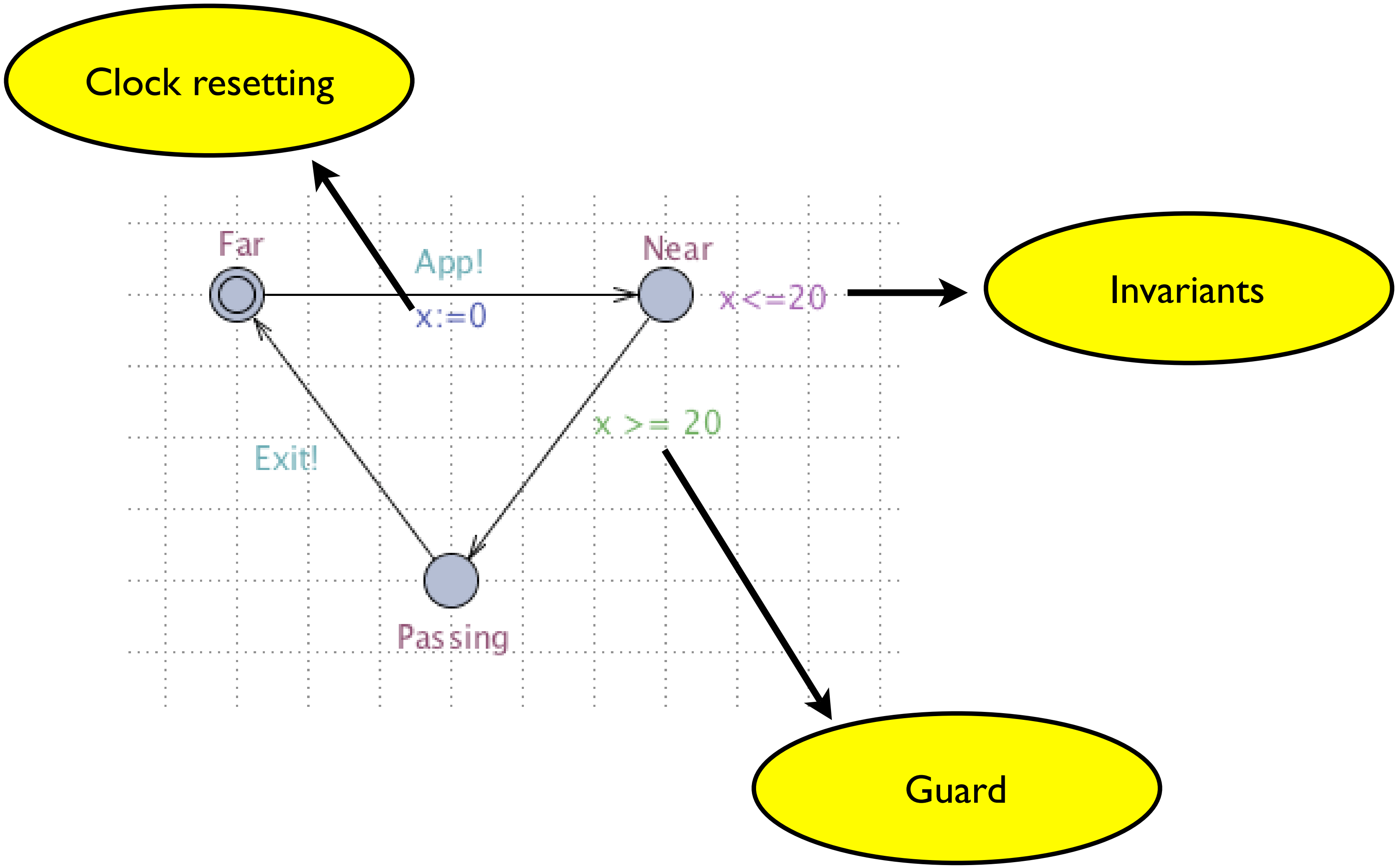
TA for the train



TA for the train



TA for the train



TA, Syntax

- A timed automata is a tuple $A=(Q,Q_0,\Sigma,P,Cl,E,L,F,Inv)$, where:
  - Q,Q_0,Σ,P,L,F are as for CFSMs;
  - Cl is a finite set of clocks;
  - $E \subseteq Q \times \Sigma \times GF(Cl) \times 2^{Cl} \times Q$ is the set of transitions, where GF(Cl) is the set of constraints of the form:

$$\Phi ::= x \sim c \mid \Phi \vee \Phi \mid \neg\Phi$$

where $x \in Cl$ and $c \in \mathbb{N}$.

  - $Inv : Q \rightarrow GF(C)$ assigns invariants over clocks to locations.

TA, Semantics - Timed traces

TA $A=(Q,Q_0,\Sigma,E,P,Cl,L,F,Inv)$ accepts the timed trace

$$s_0 \to (a_0,t_0) \to s_1 \to (a_1,t_1) \to s_2 \to (a_2,t_2) \to \ldots \to s_n \to (a_n,t_n) \to \ldots$$

iff there exists an infinite sequence

$$(q_0,v_0) \to d_0 \to (q_1,v_1) \to d_1 \to \ldots \to d_{n-1} \to (q_n,v_n) \to d_n \to \ldots$$

such that:

(1) $v_0(x)=0$ for any $x \in Cl$;

(2) $d_0=t_0$, and for any $i>0$, $d_i=t_i-t_{i-1}$;

(3) for any $i \geq 0$, there exists $(q_i,a_i,\Phi,\Delta,q_{i+1}) \in E$ such that :

- (a) $v_i \models \Phi$,
- (b) $v_{i+1}=v_i+d_i[\Delta:=0]$,
- (c) for any t, $0 \leq t \leq d_i$, $v_i+t \models Inv(q_i)$.

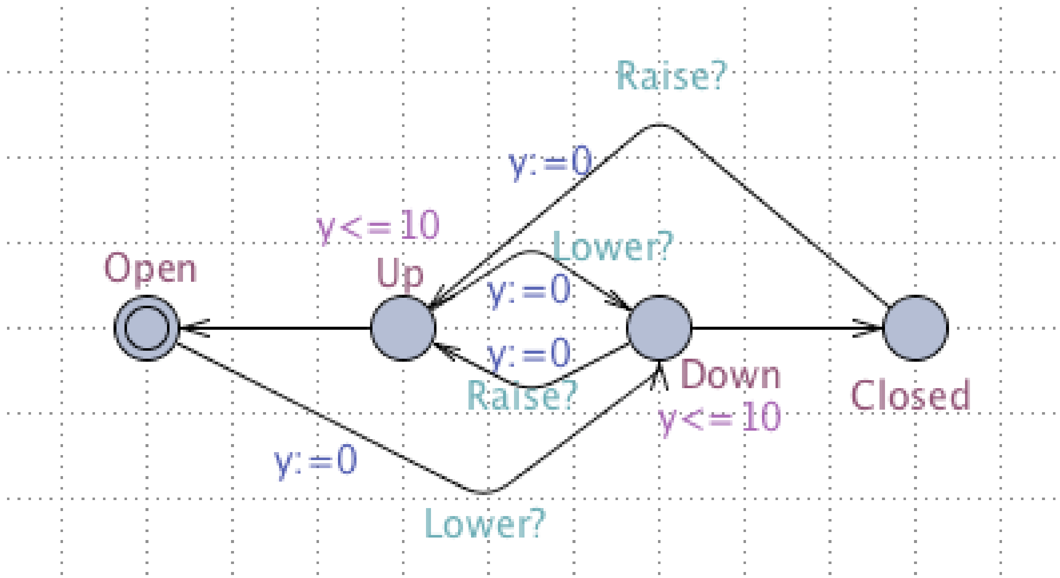
(4) for any $i \geq 0$, $L(q_i)=s_i$ and

(5) there exist infinitely many $j \geq 0$ such that $q_j \in F$ (Büchi condition).

Such a sequence is called an accepted timed run.

The set of timed traces accepted by a TA forms its timed language.

Example of timed words



Let us consider the following timed word:

Open — (1.5,Lower?) → Down — (8.75,ε) → Closed — (13,57,Raise?) → Up ...

Is it in the timed language of the Gate ?
Yes, here is a run:

(Open,0) — (1.5,Lower?) → (Down,0) — (7.25,ε) → (Closed,7.25) — (4,82,Raise?) → (Up,12,07) ...

TA, Semantics - LTS

- The $LTS=(S,S_0,\Sigma,T,C,\lambda)$ of a TA $A=(Q,Q_0,\Sigma,P,Cl,E,L,F,Inv)$, is as follows:

  - S is the set of pairs (q,v) where $q \in Q$ is a location of A and $v : Cl \rightarrow \mathbb{R}{\geq}0$ such that $v \models Inv(q)$;

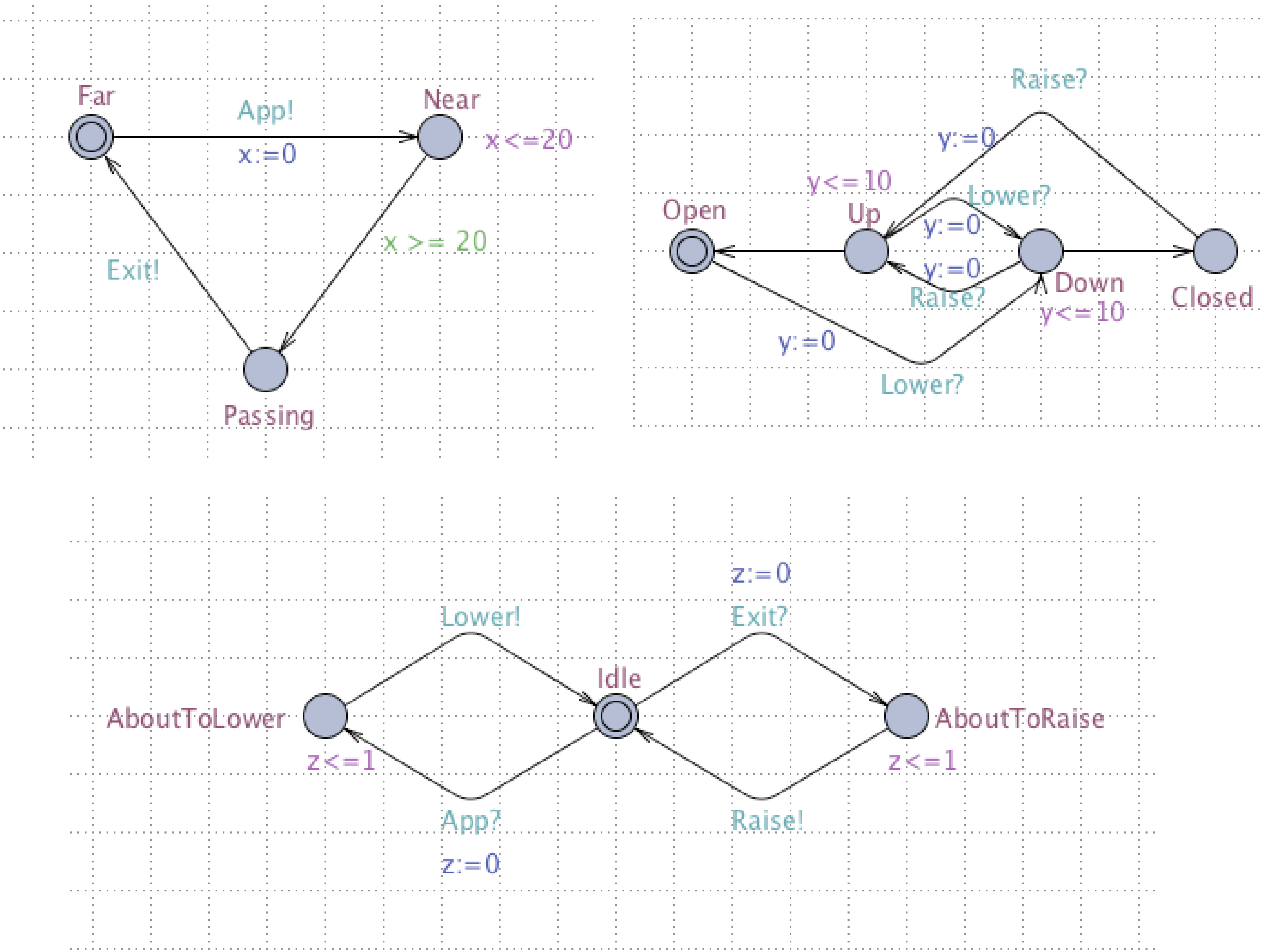
  - $S_0=\{(q_0,<0,0,\ldots 0,>) \mid q_0 \in Q_0\}$;

  - $T \subseteq S \times (\Sigma \cup \mathbb{R}{\geq}0) \times S$ defined by two types of transitions:

    Discrete transitions:
  $(q_1,v_1) \rightarrow_a (q_2,v_2) \in T$ iff there exists $(q_1,a,\Phi,\Delta,q_2) \in E$, $v_1 \models \Phi$, and $v_2:=v_1[\Delta:=0]$.
    Continuous transitions:
  $(q_1,v_1) \rightarrow_\delta (q_2,v_2) \in T$ iff $q_1=q_2$, $\delta \in \mathbb{R}{\geq}0$, $v_2=v_1+\delta$, and $\forall \delta'$, $0 \leq \delta' \leq \delta$, $v_1+\delta \models Inv(q_1)$.

  - $C=2^P$, $\lambda((q,v))=L(q)$, for any $(q,v) \in Q$.

- Clearly, this transition system has a (continuous) infinite number of states. How do we handle it ? (see second lecture)



UppAal Demo (FSM-Train-TA)

Real-time logics

- Real-time logics are extensions of temporal logics able to express real-time properties.
- Example of a real-time property:

  "it is always the case that when the the train is near, the gate is closed within 10 seconds".

The logic MTL

- MTL $\ni \Phi, \Phi_1, \Phi_2$

$$:= p \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \Phi 1 \ U_I \ \Phi 2$$

where I is an interval with rational bounds

- Example :

$$p \ U_{[2,3]} \ q$$

"p is true until q is true
within 2 to 3 time units

MTL semantics

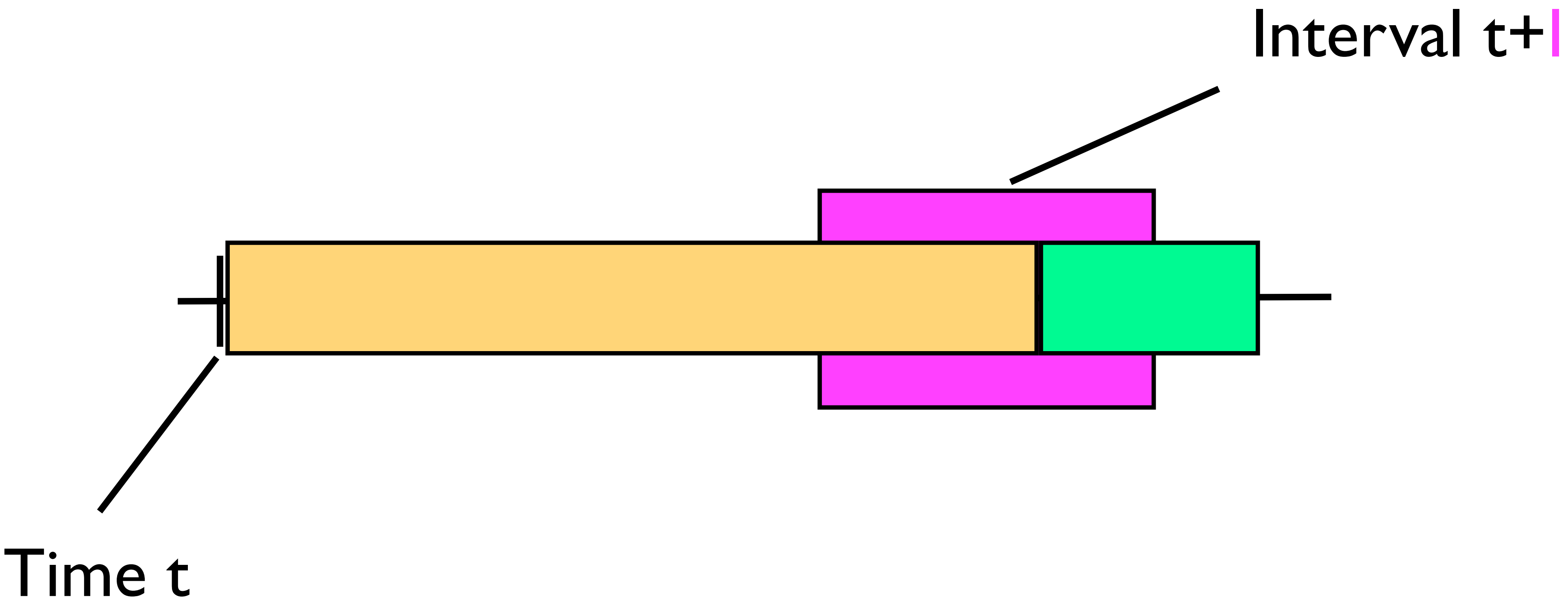
- MTL formulas are evaluated in positions along timed traces;
- Let $\eta = s_0 \rightarrow (a_0,t_0) \rightarrow s_1 \rightarrow (a_1,t_1) \rightarrow s_2 \rightarrow (a_2,t_2) \rightarrow \ldots \rightarrow s_n \rightarrow (a_n,t_n) \rightarrow \ldots$ be a timed trace:
  - a pair (i,t) is a position of η provided that $t_i \leq t \leq t_{i+1}$.
  - Given two positions (i,t), (i',t'), we have that (i,t) < (i',t') provided that i<i', or i=i' and t<t'.
  - Given a position (i,t) of η, we write η(i,t) for the suffix of η starting in (i,t), that is the trace $s_i \rightarrow (a_i,t_i-t) \rightarrow s_{i+1} \rightarrow (a_{i+1},t_{i+1}-t) \rightarrow s_2 \rightarrow (a_2,t_{i+2}-t) \rightarrow \ldots$

MTL semantics

The semantics of MTL is inductively defined as follows:

- propositional operators have their usual meaning.
- η models $\Phi_1 \, U_I \, \Phi_2$ iff there exists a position (i,t) of η such that:
    - $t \in I$
    - $\eta(i,t)$ models Φ_2
    - for all positions $(0,0) < (i',t') < (i,t)$, we have that $\eta(i',t')$ models Φ_1

MTL Semantics



$p \; U_I \; q$

MTL abbreviations

- "Bounded Eventually":

$$F_I\ \Phi \equiv \mathrm{True}\ U_I\ \Phi$$

- "Bounded Invariance":

$$G_I\ \Phi \equiv \neg\ F_I\ \neg\ \Phi$$

- Examples :

$$G\ (\ \mathrm{near} \rightarrow F_{[0,10]}\ \mathrm{closed}\ )$$

Theorem [AH96-Ras99]: the satisfiability problem for MTL is undecidable.

MITL is the subset of MTL where only non-singular intervals can be used.

Theorem [Ras99]: the satisfiability and model-checking problems for MITL are ExpSpace complete. There exists an expressively complete fragment of MITL which is PSpace complete.

Plan of the talk

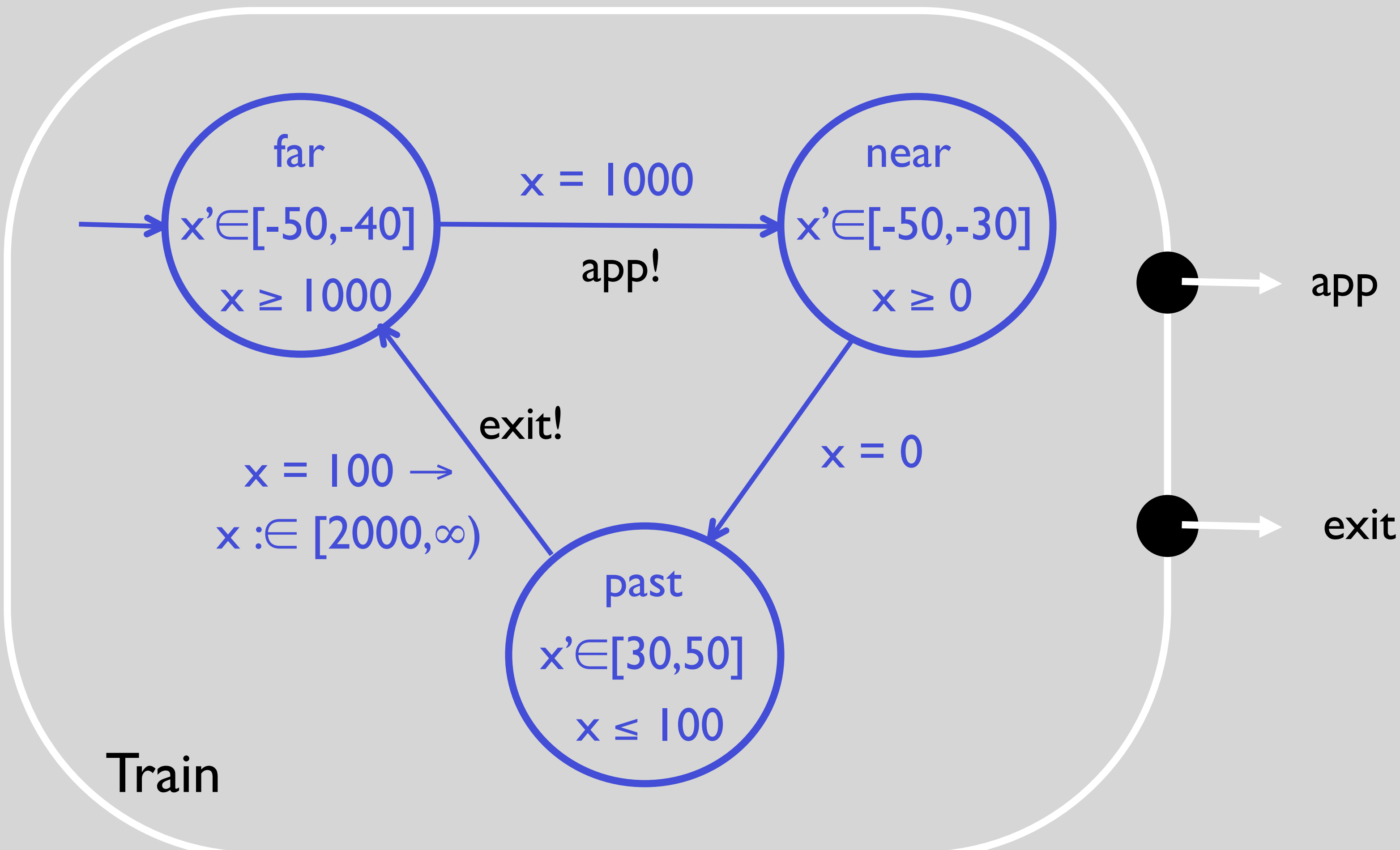
- Reactive and embedded systems
- Modeling with CFSM
- Modeling with timed automata
- Modeling with hybrid automata

Motivations

- Embedded controllers are often reacting within a complex environment with continuous components;
- We want a formalism that can naturally describe hybrid systems, that is systems with both discrete and continuous evolutions.

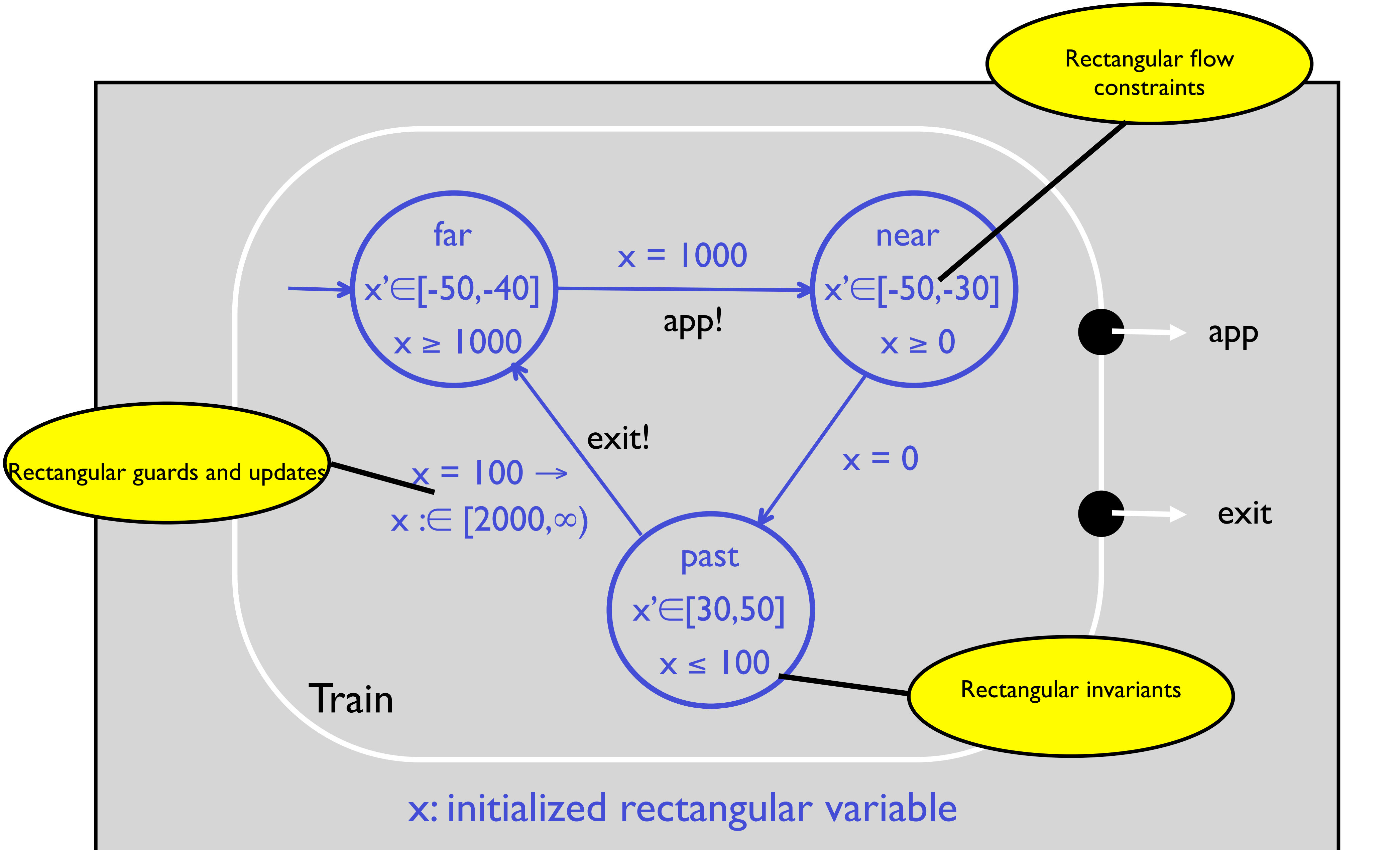
Models for reactive systems : Hybrid Automata

HA for the train

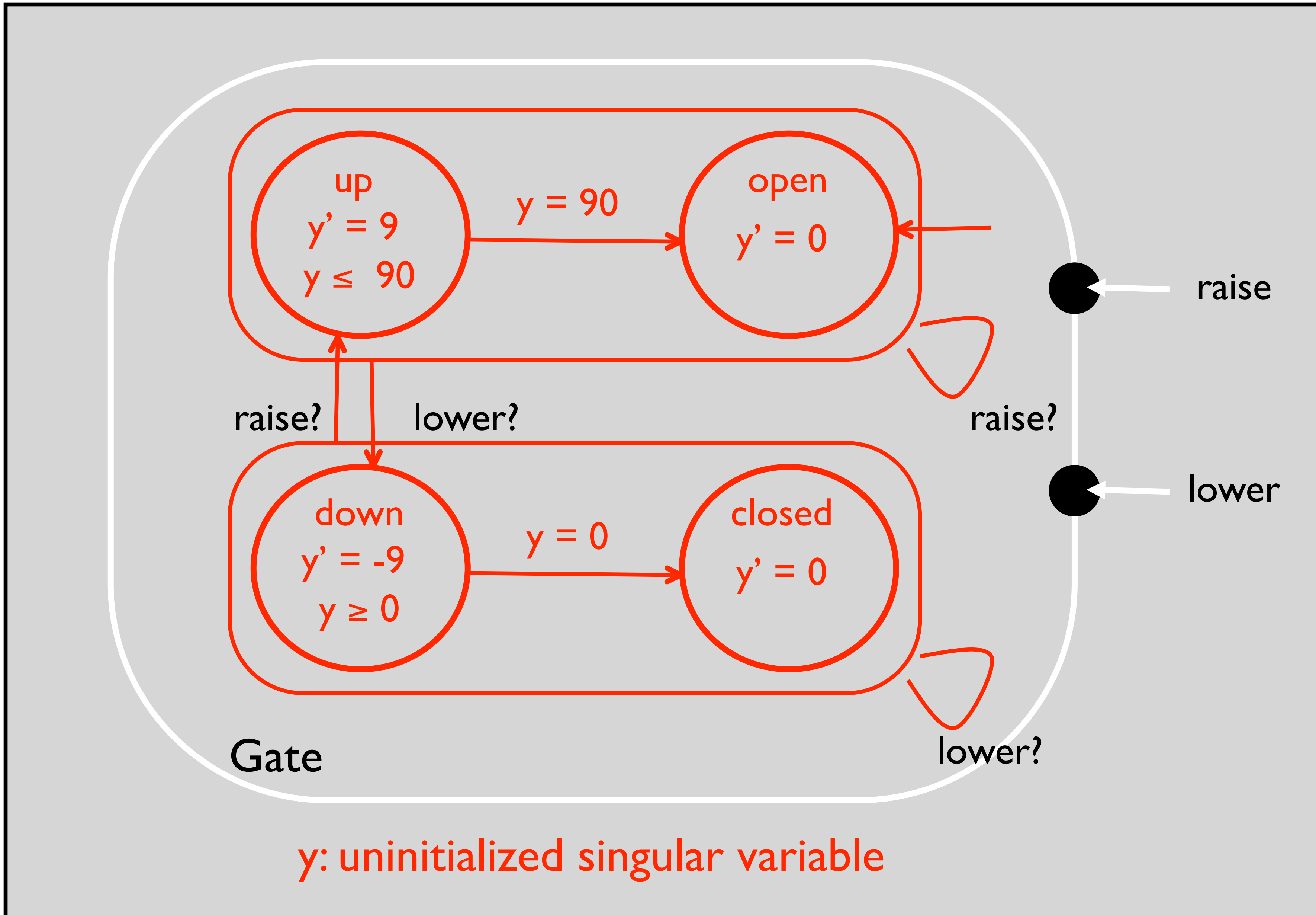


x: initialized rectangular variable

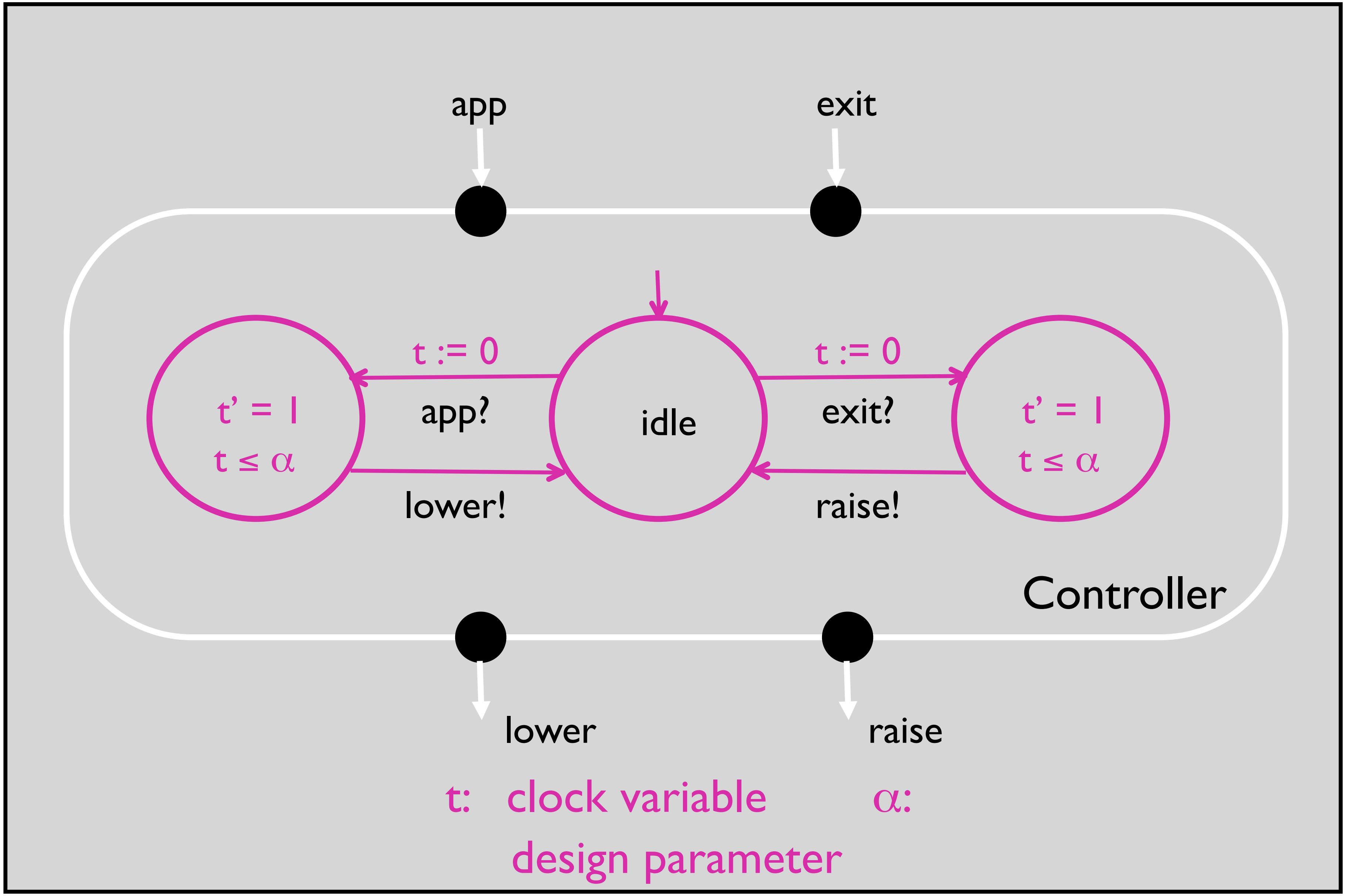
HA for the train



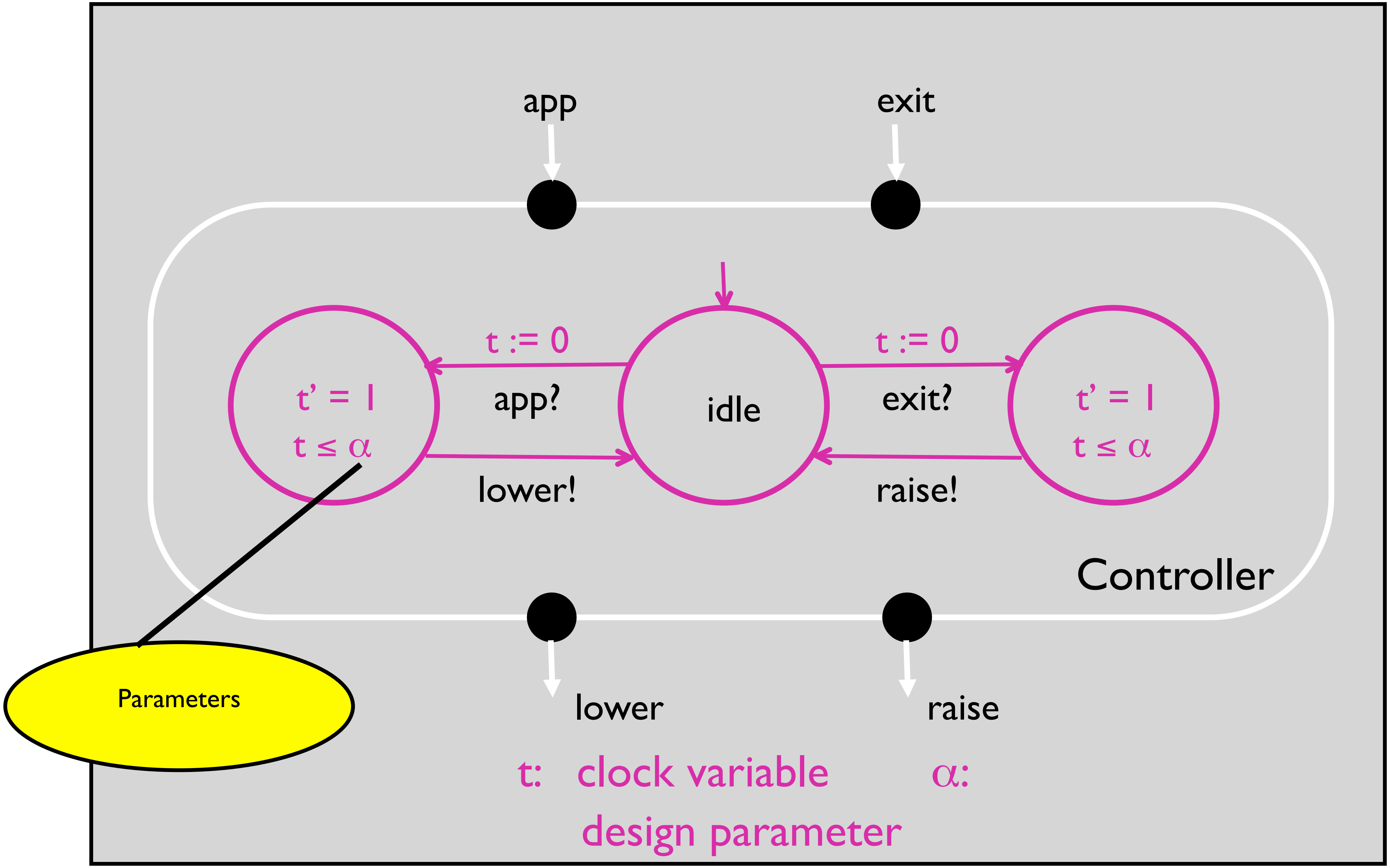
HA for the gate



HA for the controller



HA for the controller



HA, Syntax

An hybrid automaton A=(Loc,Edge,Σ,X,Init,Flow,Jump) where:

- Loc is a finite set $\{l_1,l_2,\ldots,l_m\}$ of control locations that represent control modes of the hybrid system;

- Edge ⊆ Loc x Σ x Loc is a finite set of labelled edges that represent discrete changes of control mode in the hybrid system. Those changes are labelled by event names taken from the finite set of labels Σ;

- X is a finite set $\{x_1,x_2,\ldots,x_n\}$ of real numbered variables. We write X' for the primed version of those variables X' for the first derivative of those variables.

- Init, Inv, Flow are functions that assign to each location l three predicates:

  -Init(l) is a predicate whose free variables are from X and which states what are the possibles valuations for those variables when the hybrid system starts in l.
  -Inv(l) is a predicate whose free variables are from X and which states what are the possible valuations for those variables when the control of the hybrid system is in l;
  -Flow(l) is a predicate whose free variables are from X∪X• and which states what are the possible continuous evolutions when the control of the hybrid system is in location l.

- Jump is a function that assigns to each labelled edge a predicate whose free variables are from X∪X'. Jump(e) states when the discrete change modeled by e is possible and what are the possible updates of the variables when the hybrid system makes the discrete change.

HA, Semantics

- The LTS=(S,S0,Σ,→) of a HA H=(Loc,Edge,Σ,X,Init,Flow,Jump) is defined as follows:
  - S is the set of pairs (l,v) where l∈Loc, v∈[X→R] such that v models Inv(l);
  - $S_0 \subseteq S$ such that $(l,v) \in S_0$ if v models Init(l);
  - the transitions are either:
    - discrete: for each edge (l,σ,l')∈Edge, $(l,v) \rightarrow^{\sigma} (l',v')$ iff (l,v), (l',v') ∈S, and (v,v') models Jump(e).
    - continuous for each nonnegative real δ, we have $(l,v) \rightarrow^{\delta} (l',v')$ iff l=l' and there is a differentiable function $f:[0,\delta] \rightarrow R^n$, such that the three following conditions holds: (1) f(0)=v, (2) f(δ)=v', (3) for all reals $\varepsilon \in (0, \delta)$: $f(\varepsilon)$ models Inv(l), and $(f(\varepsilon), \dot{f}(\varepsilon))$ models Flow(l).

What we would like to do...

HA Model

Property

Model Checker

Exhaustive search of the state space

Conditions under which a property is verified

- Difficult problem:
  - we do not have general methods to solve differential equations;
  - the interplay between discrete and continuous transitions make the analysis of those systems difficult (problems are usually undecidable);
  - the number of reachable states is uncountable, we must use symbolic methods.
- ...We concentrate on subclasses that are interesting in practice (reactangular HA, for example).
- ...We define approximated analysis methods (abstract interpretation).

Rectangular HA

- Rect(X) $\ni \Phi_1, \Phi_2$

  $:=$True, False, $x \in I$, $\Phi_1 \wedge \Phi_2$

  where $x \in X$, and I is an interval with rational bounds.

- UpdateRect(X,X') $\ni \Phi_1, \Phi_2$

  $:=$True, False, $x \in I$, $x' \in I$, $x'=x$, $\Phi_1 \wedge \Phi_2$

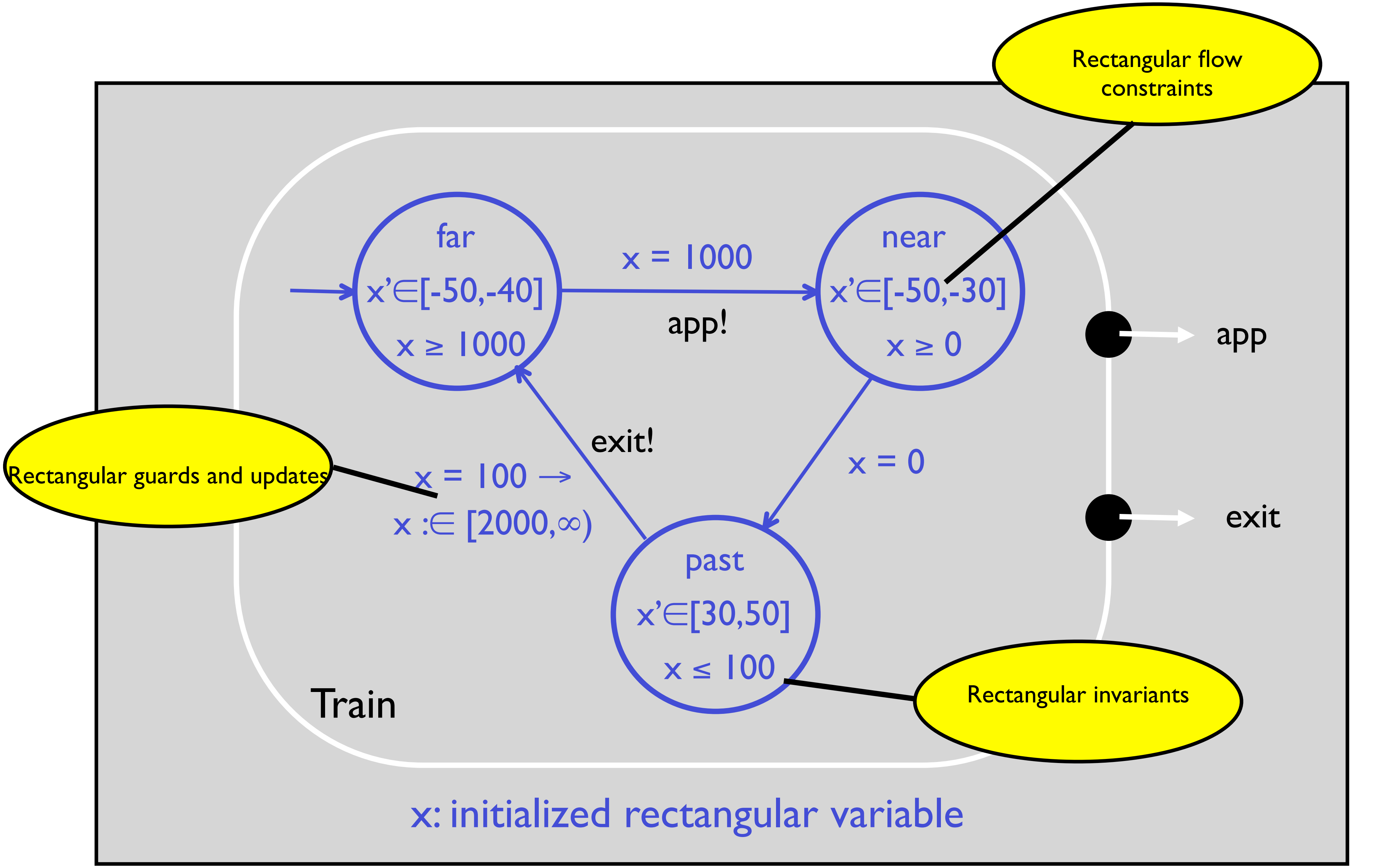
  where $x \in X$, $x' \in X'$, and I is an interval with rational bounds

- An hybrid automaton H=(Loc,Edge,Σ,X,Init,Flow,Jump) is rectangular iff for any location $l \in$ Loc, Init(l), Inv(l) are in

  Rect(X), for any edge $e \in$ Edge, Jump(e) is in UpdateRect(X,X'),

  and for any location , Flow(l) is in Rect(X•).

HA for the train



Demo HyTech

System definition: a set of hybrid automata

```
synclabs: raise, lower;
initially open & y=90;

loc up: while y<=90 wait {dy in [9,9]}       -- gate is being raised
        -- gate is fully raised
        when y=90 goto open;
        -- selfloops for receptiveness
        when True sync raise goto up;
        when True sync lower goto down ;
        when x<=10 goto error;

loc open: while True wait {dy in [0,0]}
        when True sync raise goto open;
        when True sync lower goto dowr
        when x<=10 goto error;

loc down: while y>=0 wait {dy in [-9,-
        -- gate is fully down
        when y=0 goto closed;
        when True sync lower goto dowr
        when True sync raise goto up;
        when x<=10 goto error;

loc closed: while True wait {dy in [0,0]
        when True sync raise goto up;
        when True sync lower goto close

loc error: while True wait {dy in [0,0]}

end -- gate
```

Requirements analysis: a script of verification commands

```
var init_reg, final_reg, reached: region;

init_reg := loc[train]=far & x>=2000 &
            loc[controller]=idle &
            loc[gate]=open & y=90;
final_reg := loc[gate] = up & x<=10 | loc[gate]=open & x<=10 |
             loc[gate] = down & x<=10;

reached := reach forward from init_reg endreach;


prints "Conditions under which system violates safety requirement";
print omit all locations
        hide non_parameters in    reached & final_reg endhide;
```

Execute HyTech

Conclusion

- Timed and hybrid automata are well-suited models for embedded systems;
- Towards a model based methodology for the development of safety critical embedded controllers
- In the second lecture, we will see the foundations for the analysis of timed models.