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?
Mars, December 3, 1999

Crash caused by an uninitialized variable
300 horses power
100 processors
Concurrency: several hardware and software components
Heterogeneity: digital (discrete time) and analog (continuous time) environment
Uncertainty: exception handling

Cellular Phone

more and more software

---

DSP

Microprocessor & Control Logic

RF TX & RX Amplifiers

Audio D/A & A/D

Memory

RF and Power

Text
Concurrency : 300 000 logical gates
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 difficult**, we may not be able to replay the error.
Need for verification

• ... as they are difficult to develop correctly!

• ... and often safety critical!

⇒ 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 entail $\Phi$?
  
  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.

- $\Phi$ is a property that refers to the entire computation: we are interested in temporal behaviors of the system.
  
  $\Phi$ 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
    \[ s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow ... \rightarrow s_n \rightarrow a_n \rightarrow ... \]

- each \( s_i \) is a subset of \( P \) (a finite set of propositions over the system);

- each \( a_i \) is an element of \( \Sigma \), a finite set of events;

- Semantics of the system = (infinite) set of traces = \( \omega \)-regular language.
• **Properties** of a reactive can also be expressed as $\omega$-regular languages

• Verification of finite-state reactive systems = manipulate, compare, test properties of $\omega$-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;
- \( \Sigma \) 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).
The automaton $A=(Q,Q_0,\Sigma,E,P,L,F)$ accepts the trace
\[ s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow ... \rightarrow s_n \rightarrow a_n \rightarrow ... \]
iff there exists an infinite sequence of states
\[ q_0 q_1 ... q_n ... \]
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 $\text{Lang}(A)$. 

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

Far $\xrightarrow{\text{App!}}$ Near $\xrightarrow{\epsilon}$ Passing $\xrightarrow{\text{Exit!}}$ Far ...

So the language of this automaton is

$$\{ \text{Far } \xrightarrow{\text{App!}} \text{ Near } \xrightarrow{\epsilon} \text{ Passing } \xrightarrow{\text{Exit!}} \text{ Far } \ldots \}$$
Gate model
The language of the gate is:

\{ \text{Open} \rightarrow \text{Lower}? \rightarrow \text{Down} \rightarrow \varepsilon \rightarrow \text{Closed} \rightarrow \text{Raise}? \rightarrow \text{Up} \ldots, \\
\text{Open} \rightarrow \text{Lower}? \rightarrow \text{Down} \rightarrow \text{Raise}? \rightarrow \text{Up} \rightarrow \text{Lower}? \rightarrow \text{Down} \ldots \}

Controller model
Finite state machines 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 $P_a \cap P_b = \emptyset$. 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. $Q = Q_a \times Q_b$
2. $Q_0 = Q_{a0} \times Q_{b0}$
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:
   - $a \in \Sigma_a, a \not\in \Sigma_b, (q_{a1}, a, q_{a2}) \in E_a, q_{b1} = q_{b2}$
   - $a \in \Sigma_b, a \not\in \Sigma_a, (q_{b1}, a, q_{b2}) \in E_b, q_{a1} = q_{a2}$
   - $a \in \Sigma_a, a \in \Sigma_b, (q_{a1}, a, q_{a2}) \in E_a, (q_{b1}, a, q_{b2}) \in E_b$
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. $F = \{ (q_a, q_b) \mid q_a \in f_a \} \cup \{ (q_a, q_b) \mid q_b \in f_b \}$
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:
Let us consider our example with the following generalized Büchi condition: $F=\{\{\text{Open}\}, \{\text{Closed}\}\}$.

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:

\[ \Phi ::= p \mid \neg \Phi_1 \mid \Phi_1 \lor \Phi_2 \mid X \Phi \mid \Phi_1 U \Phi_2 \]

where \( \Phi_1, \Phi_2 \in \Phi \).

Formula of LTL are evaluated over states of traces.
LTL - Semantics

Let \( \eta = s_0 \rightarrow a_0 \rightarrow s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow ... \rightarrow s_n \rightarrow a_n \rightarrow ... \) be a infinite trace over the set of propositions \( P \) (and events \( \Sigma \)). 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 \( \Phi_1 \)
- \( \eta(i) \) models \( \Phi_1 \lor \Phi_2 \) iff
  \[ \eta(i) \) models \( \Phi_1 \) or \( \eta(i) \) models \( \Phi_2 \)
- \( \eta(i) \) models \( X \Phi \) iff \( \eta(i+1) \) models \( \Phi \)
- \( \eta(i) \) models \( \Phi_1 \cup \Phi_2 \) iff there exists \( j \geq i \), such that
  \( \eta(j) \) models \( \Phi_2 \) and
  for all \( k, i \leq k < j \), \( \eta(k) \) models \( \Phi_1 \)
A formula $\Phi$ is true over a trace $\eta$ iff
"$\eta(0)$ models $\Phi$"

A formula $\Phi$ is true over a set of traces $H$ iff for all $\eta \in H$, $\Phi$ is true over $\eta$. 
LTL - Abbreviations

The following abbreviations are useful:

\[ F \Phi \equiv \text{True } U \Phi, \text{“Eventually } \Phi”. \]
\[ G \Phi \equiv \neg F \neg \Phi, \text{“Always } \Phi”. \]
Examples of properties expressed in LTL

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

\[ G ( \text{past} \rightarrow \text{closed} ) \]

At any time, the gate will *eventually* be open:

\[ G \ F \text{ open} \]
The LTL model-checking problem

Given a product of \( n \) CFSMs \( M_1 \times M_2 \times \ldots \times M_n \), given a formula of LTL \( \Phi \), determine if the set of traces defined by \( M_1 \times M_2 \times \ldots \times M_n \) satisfies the formula \( \Phi \).

There are algorithms and implementations that solve this problem but it is problem is provably hard: it is complete for \( \text{PSpace} \).
Question: are the two following formulas

\( G (\text{past} \rightarrow \text{closed}) \)

\( G \ F \text{ open} \)

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

\[ s_0 \rightarrow (a_0, t_0) \rightarrow s_1 \rightarrow (a_1, t_1) \rightarrow s_2 \rightarrow (a_2, t_2) \rightarrow \ldots \rightarrow s_n \rightarrow (a_n, t_n) \rightarrow \ldots \]

where:
- each \( s_i \) is a subset of the set of propositions \( P \);
- each \( a_i \) is an element of \( \Sigma \), 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

Clock resetting

**States:**
- Far
- App!
- Near
- Exit!
- Passing

**Transitions:**
- From Far to App! with condition $x := 0$
- From App! to Near with condition $x <= 20$
- From Near to Exit! with condition $x >= 20$
- From Exit! to Passing
- From Passing to Far

**Actions:**
- App!
TA for the train

Clock resetting

Invariants
TA for the train

Clock resetting

Invariants

Guard
TA, Syntax

- A timed automata is a tuple $A=(Q,Q_0,\Sigma,P,\mathcal{C},E,L,F,\text{Inv})$, where:
  - $Q,Q_0,\Sigma,P,L,F$ are as for CFSMs;
  - $\mathcal{C}$ is a finite set of clocks;
  - $E \subseteq Q \times \Sigma \times \text{GF}(\mathcal{C}) \times 2^\mathcal{C} \times Q$ is the set of transitions, where $\text{GF}(\mathcal{C})$ is the set of constraints of the form:
    $$\Phi ::= x \sim c \mid \Phi \lor \Phi \mid \neg \Phi$$
    where $x \in \mathcal{C}$ and $c \in \mathbb{N}$.
  - $\text{Inv} : Q \rightarrow \text{GF}(\mathcal{C})$ assigns invariants over clocks to locations.
TA, Semantics - Timed traces

TA $A=(Q,Q_0,\Sigma,E,P,\mathbf{C},L,F,\text{Inv})$ accepts the timed trace

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

iff there exists an infinite sequence

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

such that:

1. $v_0(x)=0$ for any $x \in \mathbf{C}$;
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 $t$, $0 \leq t \leq d_i$, $v_i+t \models \text{Inv}(q_i)$.
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.
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) ...
• The LTS=(S,S₀,Σ,T,C,λ) of a TA A=(Q,Q₀,Σ,P,Cl,E,L,F,Inv), is as follows:

  - S is the set of pairs (q,v) where q ∈ Q is a location of A and v : Cl → ℝ≥0 such that v ⊨ Inv(q);
  - S₀= {(q₀,<0,0,...,>) | q₀ ∈ Q₀};
  - T ⊆ S x (Σ∪ℝ≥0) x S defined by two types of transitions:

    Discrete transitions:
    (q₁,v₁)→ₐ(q₂,v₂) ∈ T iff there exists (q₁,a,Φ,Δ,q₂) ∈ E, v₁ ⊨ Φ, and v₂:=v₁[Δ:=0].

    Continuous transitions:
    (q₁,v₁)→₅(q₂,v₂) ∈ T iff q₁=q₂, δ∈ℝ≥0, v₂=v₁+δ, and ∀δ', 0≤δ'≤δ, v₁+δ ⊨ Inv(q₁).

  - C=2^P, λ((q,v))=L(q), for any (q,v)∈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 train is near, the gate is closed within 10 seconds”.
The logic MTL

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

$$\equiv p | \neg\Phi | \Phi_1 \lor \Phi_2 | \Phi_1 \cup_{|l} \Phi_2$$

where $l$ is an interval with rational bounds

- Example:

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

  “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 $\eta$ 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 $\eta$, we write $\eta(i, t)$ for the suffix of $\eta$ 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.
- \( \eta \) models \( \Phi_1 \cup \Phi_2 \) iff there exists a position \((i,t)\) of \( \eta \) such that:
  - \( t \in I \)
  - \( \eta(i,t) \) models \( \Phi_2 \)
  - for all positions \((0,0) \prec (i',t') \prec (i,t)\), we have that \( \eta(i',t') \) models \( \Phi_1 \)
MTL Semantics

Time $t$

Interval $t+1$

$ p \cup t \cup q $
MTL abbreviations

- “Bounded Eventually”:
  \[ F_t \Phi \equiv \text{True} U_t \Phi \]

- “Bounded Invariance”:
  \[ G_t \Phi \equiv \neg F_t \neg \Phi \]

- Examples:
  \[ G ( \text{near } \rightarrow F_{[0,10]} \text{ closed} ) \]
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

- **far** $x' \in [-50, -40]$ $x \geq 1000$
- **near** $x' \in [-50, -30]$ $x \geq 0$
- **past** $x' \in [30, 50]$ $x \leq 100$
- $x = 1000$
- $x = 0$
- $x \in [2000, \infty)$

Train

$x$: initialized rectangular variable

App!

Exit!
HA for the train

Train

$x$: initialized rectangular variable

Rectangular guards and updates

Rectangular flow constraints

Rectangular invariants
HA for the gate

- **up**: $y' = 9$, $y \leq 90$
- **open**: $y = 90$, $y' = 0$
- **down**: $y' = -9$, $y \geq 0$
- **closed**: $y = 0$, $y' = 0$

$y$: uninitialized singular variable
HA for the controller

$t' = 1$
$t \leq \alpha$
$t := 0$

app?
lower!
exit?
raise!

Controller

$t$: clock variable
$\alpha$: design parameter
HA for the controller

Parameters

- $t$: clock variable
- $\alpha$: design parameter

$t' = 1$
$t \leq \alpha$
$t := 0$

app
exit

app?
exit?
app!
exit!

lower!
raise!

idle

Controller
HA, Syntax

An hybrid automaton $A = (\text{Loc}, \text{Edge}, \Sigma, X, \text{Init}, \text{Flow}, \text{Jump})$ 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** $\subseteq \text{Loc} \times \Sigma \times \text{Loc}$ 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 $\Sigma$;

- **$X$** 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 possible 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 \( X UX^* \) 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 \( X UX^* \). **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.
The LTS=$\langle S,S_0,\Sigma,\to \rangle$ of a HA $H=(\text{Loc},\text{Edge},\Sigma,X,\text{Init},\text{Flow},\text{Jump})$ is defined as follows:

- $S$ is the set of pairs $(l,v)$ where $l \in \text{Loc}$, $v \in [X \to \mathbb{R}]$ 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 $(l,\sigma,l') \in \text{Edge}$, $(l,v) \to \sigma (l',v')$ iff $(l,v), (l',v') \in S$, and $(v,v')$ models Jump(e).
  - continuous for each nonnegative real $\delta$, we have $(l,v) \to \delta (l',v')$ iff $l = l'$ and there is a differentiable function $f:[0,\delta] \to \mathbb{R}^n$, such that the three following conditions holds: (1) $f(0)=v$, (2) $f(\delta)=v'$, (3) for all reals $\varepsilon \in (0,\delta)$: $f(\varepsilon)$ models Inv($l$), and $(f(\varepsilon), f(\varepsilon))$ models Flow($l$).
What we would like to do...

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

- \( \text{Rect}(X) \ni \Phi_1, \Phi_2 \)
  
  := \text{True, False, } x \in I, \Phi_1 \land \Phi_2 
  
  where \( x \in X \), and \( I \) is an interval with rational bounds.

- \( \text{UpdateRect}(X, X') \ni \Phi_1, \Phi_2 \)
  
  := \text{True, False, } x \in I, x' \in I, x' = x, \Phi_1 \land \Phi_2 
  
  where \( x \in X, x' \in X' \), and \( I \) is an interval with rational bounds

- An hybrid automaton \( H = (\text{Loc}, \text{Edge}, \Sigma, X, \text{Init}, \text{Flow}, \text{Jump}) \) is rectangular iff for any location \( l \in \text{Loc} \), \( \text{Init}(l), \text{Inv}(l) \) are in \( \text{Rect}(X) \), for any edge \( e \in \text{Edge} \), \( \text{Jump}(e) \) is in \( \text{UpdateRect}(X, X') \), and for any location \( l \), \( \text{Flow}(l) \) is in \( \text{Rect}(X^\bullet) \).
HA for the train

Rectangular guards and updates

Rectangular flow constraints

Rectangular invariants

Train

$x$: initialized rectangular variable
**System definition**: a set of hybrid automata

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

loc up: while y<=90 wait {dy in [9,9]}
        -- 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 down;
         when x<=10 goto error;

loc down: while y>=0 wait {dy in [-9,-9]}
         -- gate is fully down
         when y=0 goto closed;
         when True sync lower goto down;
         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 open;
          when True sync lower goto close;

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

end -- gate
```
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.