Lecture 5. Modelling hybrid systems: timed (and a short word on hybrid) automata

Sylvie Putot
Ecole Polytechnique
putot@lix.polytechnique.fr
Hybrid (Cyber-Physical) Systems?

Collaborating smart devices with sensing, computation, communication and control capabilities, that interact with physical entities.
Cyber-physical systems

**Embedded Systems**: special-purpose system with integrated microcontroller/software (cameras, washing machines, etc)

- software running in resource constrained systems

**Cyber Physical Systems**: computer-aided control of physical systems

- Computation: process information to make decisions
- Communication: exchange data to collaborate
- Mixing discrete/continuous dynamics: monitor and act on the physical world = **Hybrid Systems**

Embedded systems, Internet of Things/of everything, Systems of systems / Complex systems, Smart<Everything>, … = CPS!
Implanted medical devices « BioCyber systems »

WIRELESS IMPLANTABLE MEDICAL DEVICES

- Deep Brain Neurostimulators
- Cochlear Implants
- Gastric Stimulators
- Cardiac Defibrillators/Pacemakers
- Foot Drop Implants
- Insulin Pumps
Closed-loop medical devices (pacemaker, insulin pump, etc)

They present all the challenges of safe CPS design

Numerically complex modeling

- Modeling the relevant aspects of human physiology: insulin-glucose regulatory models, cardiac modeling, etc.
- Reasoning about uncertainties (modelling, sensors of limited capacity, human behaviour etc)
- Control: sophisticated algorithms to control critical physiological functions with sensing/actuation / computing limitations

Safety, security (and ethical) issues

- Closed-loop medical devices are safety-critical: malfunctions result in serious injury or death to the patient
- Security / privacy issues (August 2017 - hacking risk leads to recall of 500,000 pacemakers)
Challenges in Cyber-Physical Systems

Rapidly growing field with lots of promise/potential (driverless cars, traffic management, wearable/implanted medical devices, coordinating robots, smartgrids etc), partially due to increased connectivity ...

... often safety-critical!

Challenge = developing high-confidence CPS under uncertainties in physical systems, managing concurrency and timing issues

A currently very active field of research!
Overview

Today: mostly modeling

- Dealing with time in programs: [reactive and synchronous programs], timed automata
- Project:
  - a simple timed model of the behavior a pacemaker
  - a first word on hybrid automata and reachability analysis

Simulation and verification of hybrid systems (next 2 lectures)

- Reachability analysis, uncertainty handling
- Specification in temporal logics and introduction to model-checking
Books and references

Lee & Seshia: Introduction to Embedded Systems - A Cyber-Physical Systems Approach:

• available http://leeseshia.org


• 5 books available from the Polytechnique library
Some tools: from modelling to verification

- **Lustre, SCADE (Esterel)**: synchronous languages
  (and Zelus, its recent extension to ODE http://zelus.di.ens.fr)

- **Uppaal**: modeling and verification of real-time systems modeled as networks of timed automata, free)
  http://www.uppaal.org

- **Matlab/Simulink**: proprietary, available at X

- **Modelica**: modeling language for complex systems - free)
  https://www.modelica.org

- **Ptolemy**: modeling, simulation, and design of heterogeneous concurrent, real-time systems - free)
  http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm

- **Keymaera**: theorem prover for Hybrid Systems - free)
  http://symbolaris.com/info/KeYmaera.html

- **SpaceEx**: reachability for affine hybrid systems (free) http://spaceex.imag.fr
  **Flow** (non affine systems)

Many tools: different levels of safety requirements and types of properties (temporal, functional or not, etc)
Cyber-physical systems are embedded control-command systems = closed-loop reactive systems:

On-going reaction to an *unprecisely known* environment that *cannot wait*

Example: cruise controller, control command of a plane

- inputs = plane state (position, speed etc), outputs = actions of the pilot
- aim = stabilize the plane
- reactivity: ex update every 1ms (must compute quickly enough)
A reactive system example: insulin pump
Reactive system: simple implementation

\[
M := M_0 \quad // \quad \text{memory initialization} \\
\text{loop} \\
\quad \text{wait}(I) \quad // \quad \text{output func} \\
\quad O = f(M,I) \quad // \quad \text{transition func} \\
\quad M = g(M,I) \\
\quad \text{write}(O) \\
\text{end loop}
\]

with execution time < period

Block Diagrams representation

- Widely used in industrial design
- Tools: Scade, Simulink, Modelica, Labview, etc

What is the semantics of an execution for complex systems?

- Hierarchical/parallel composition of components
- Need to guarantee determinism and real-time (multi-task approach with dynamic scheduling is unpredictable)
Model-based design in embedded control systems

- modelling and simulation: continuous/discrete time toolboxes (Simulink):
- model-based development tools with formal semantics, (SCADE, synchronous approach)
- code generated for execution platform

Perspectives: more expressivity with clear semantics, more formal verification
The synchronous approach

Discrete, « logical » time (not seconds, but inputs sequences)

- the components of the system are cadenced by discrete time signals
- time not uniquely observable: seconds, events, meters, heartbeats, etc.
- no notion of duration of events: execution and communication are instantaneous

The implementation must ensure timing assumption: time needed to execute the update code is negligible compared to clock rate (= delay between successive inputs or events)

- WCET analysis

Simple model with well-defined semantics:

- deterministic parallelism: parallel execution of modules when they occur at the same time instant
- makes possible verification and automatic generation of reliable code
- model not always suitable for modern CPS (smart grids etc)
Synchronous-reactive languages

New programming languages for real-time control: synchronous languages

- Three languages born in France roughly at the same time (mid 80s)
  - Lustre (Halbwachs and Caspi), Esterel (Berry and Gonthier), Signal (Benveniste and Le Guernic)

Strong industrial challenges in the 80s for embedded software:

- nuclear plants: SPIN (Nuclear Integrated Protection System), Avionics: A320, 1st fly-by-wire aircraft, Ground transportation (TGV, VAL, . . .)
- SCADE (Safety Critical Applications Development Environment): graphical language based on Lustre, initial academic/industrial consortium, then Esterel Technologies

SCADE is now a standard for safety critical development in avionics, space, railway, nuclear and defense
Input and output signals

Inputs and outputs of reactive components are signals, i.e. temporal functions $s: \mathbb{R} \to D_s$, which associate to each time instant $t$, a value in $D_s$.

- pure signals or events: $\forall t, s(t) \in D_s = \{\text{present}, \text{absent}\}$
- valued signals: $D_s = \{\text{absent}\} \cup V_s$, with $V_s \subseteq \mathbb{Z}$ or $\mathbb{B}$ or $\mathbb{R}$

A discrete signal, intuitively, is absent most of the time, and we can count, and order, the times at which it is present

- the output is absent at times when a reaction does not occur

Example:

- $\text{event(bool)}$ in means $\text{in} \in \{0,1,\text{absent}\}$
- Boolean expression $\text{in?}$ means $\text{in} \neq \text{absent}$
Guards determine when a transition *may* be taken. A guard is a predicate over inputs (Boolean formulæ made of $\land, \lor, \neg$, and comparison ops $\geq, =, >$ etc).

When it evaluates to true, the transition is enabled.

Actions specify the value of the output when the transition is taken. Any output port not mentioned in a transition is absent.
Example: simple thermostat with hysteresis

Setpoint $T = 20$ degrees, hysteresis to avoid chattering

- input = $\text{temperature} : \{\text{absent}\} \cup \mathbb{R}$, outputs = $\text{heatOn, heatOff} : \text{pure}$; states = $\{\text{heating, cooling}\}$

- if no guard evaluates to true, the state remains unchanged and no output is generated (default transition)

- pure output signal ($\text{heatOn, heatOff}$): the output is present at times when a reaction occurs, absent otherwise

- could be event-triggered (reacts whenever a temperature input is provided) or time-triggered (reacts at regular time intervals): determined by the environment
Non determinism: environment modeling

Modeling the environment: taking uncertainties into account, hiding irrelevant details.

Example: model of a pedestrian arriving at a traffic light

```
inputs: sigR, sigG, sigY : pure (color of traffic light for cars)
outputs: pedestrian : pure
```

Inputs come from the traffic light model. Two possible transitions (in red) from node none: non determinist
Modeling real-time systems: a light switch

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

Timed model: adding time passing to synchronous model
A timed model of the light switch

Solution: add real-valued clock $x$

In each state, the process is waiting for the input event $press$. Waiting for a time period $\delta$ is modeled by a timed action of duration $\delta$.

A sample execution: dated observable events

$$(off, 0) \xrightarrow{1.5} (off, 1.5) \xrightarrow{press?} (light, 0) \xrightarrow{0.2} (light, 0.2) \xrightarrow{0.5} (light, 0.7)$$

$$(press?) \xrightarrow{(bright, 0.7)} \xrightarrow{0.2} (bright, 0.9) \xrightarrow{press?} (off, 0.9)$$
Timed automata are (extended) FSMs with addition of clocks.

- A clock is a continuous variable whose value within the finite states (locations) of the automaton can be described by the differential equation $\dot{x}(t)=1$
- Timed automata are the simplest form of hybrid automata, where the continuous dynamics is the passage of time
- A popular model for reactive systems with continuous time proposed by Alur and Dill in 1991
Timed automata, informally

• Transitions are instantaneous; time elapses when the automaton is in a state
  • Time passes at the same rate for all clocks: same global time
  • States can have associated invariants, expressed as conditions on the clocks; the automaton can stay in that state only as long as the invariant is true.
    • Clock invariants on states ensure progress between states
• Transitions can have
  • guards (conditions on clock values); the transition can be taken only if the guard is satisfied.
  • input signals (events); when the signal arrives and the associated guard is satisfied, the transition will be taken.
• When a transition occurs:
  • some of the clocks can be reset
  • output signals (events) can be generated
A timed process consists of ...

- An asynchronous process, where some state variables are of type clock
  - a finite set $I$ of input channels defining inputs under the form $x?v$ with $x \in I$ and $v$ a value for $x$, a finite set of outputs under the form $y!v$
  - a finite set $S$ of state variables, and an initialisation
  - for each channel $x$, a set of input/output tasks, of internal tasks, each described by a guard condition over $S$ and an update
- A clock invariant CI (a Boolean expression over the state variables)
- Timed transition system:
  - given a state $s$ and $\delta > 0$, $s \overset{\delta}{\rightarrow} s + \delta$ is a timed action (or delay transition) if the state $s+t$ (which assigns $s(x)+t$ to all clocks $x$) satisfies CI for all $t \in [0,\delta]$: timed action executed synchronously
  - action (instantaneous) transitions: asynchronous interleaving of input/output / internal actions;
Another example: timed buffer with a bounded delay

WANTED: input received on the channel in is transmitted on the output channel after a delay of $\delta$ such that $LB \leq \delta \leq UB$ (i.e. we know lower/upper bounds on this delay)
Timed buffer with a bounded delay

- buffer of capacity 1, input channel `in`, output channel `out`, state variable `x`
- when the buffer is `Full`, it ignores (loses) further inputs until it outputs the stored value
- clock variable `y` tracks the time elapsed since the buffer is full: the guard `y ≥ LB` ensures the msg is issued only after the lower bound `LB` on the delay
- assumption on the upper bound captured by the annotation `y ≤ UB` associated to mode `Full`: clock invariant; timed transition allowed only if the clock invariant remains true

Output signal: `x` is set on channel `out`
Consider a timed process with

- input event $x?$, output events $y!$ and $z!$

Desired behavior: whenever it receives input, it produces both its output events

- time delay between $x?$ and $y!$ is in the interval $[2, 4]$
- time delay between $x?$ and $z!$ is in the interval $[3, 5]$
- input events received during this are ignored

Draw a timed state machine that captures this behavior
Exercise: a solution

clock $ck:=0$

Idle

WaitYZ
$ck \leq 4$

WaitY
$ck \leq 4$

WaitZ
$ck \leq 5$

$ck \geq 3$ / $z!$

$ck \geq 2$ / $y!$

$ck \geq 3$ / $z!$

$ck \geq 2$ / $y!$

$x?$ / $ck:=0$

$y!$

$z!$
Timed process composition

Consider timed processes $TP1 = (P1, CI1)$ and $TP2 = (P2, CI2)$

The parallel composition $TP1 \mid TP2$ is defined when the asynchronous parallel composition $P1 \mid P2$ is defined, that is, when the outputs of the two are disjoint, and $TP1 \mid TP2 = (P1 \mid P2, CI1 \land CI2)$

- for states $s1$ and $s2$ of $TP1$ and $TP2$, and a time duration $\delta$, $(s1, s2) \xrightarrow{\delta} (s1 + \delta, s2 + \delta)$ is a timed action of $TP1 \mid TP2$ if $s1 \xrightarrow{\delta} s1 + \delta$ is a timed action of $TP1$ and $s2 \xrightarrow{\delta} s2 + \delta$ a timed action of $TP2$

- the two tasks execute in parallel, asynchronously, but timing introduces loose coordination
Distributed coordination problems

How can we exploit the knowledge of timing delays to design protocols?

Example: asynchronous vs timed increments

Consider the composition of asynchronous increments

The interleaving semantics defining all possible executions of the asynchronous interaction yields that every possible state of the form \((i,j)\) with \(i,j\) natural numbers is reachable.
Example: asynchronous vs timed increments

Now consider the parallel composition of the timed increments:

1st task increments $x$, within a delay between 1 and 2

2nd task increments $y$, within a delay between 1 and 2

the two tasks execute in parallel, asynchronously, but timing introduces loose coordination

which values of $(x, y)$ are now reachable? $[x \leq 2y + 2$ and $y \leq 2x + 2$ are invariants of the system]
Timing-based mutual exclusion: Fisher’s protocol

Mutual exclusion problem

- safety: several processes should not be in critical section simultaneously
- absence of deadlock

Fisher’s algorithm

- one shared read/write register (variable id)
- when a process wants to enter critical section, it reads if
  - If id != 0 then try again, if id == 0 then set id=pid
  - Proceeding directly to critical section is a problem (since the other process may also have concurrently read id to 0, and updated id to its own ID)
  - Solution: wait till you are sure that concurrent writes are finished
  - Timing assumption: takes at most time k to write, is established by location invariant x≤k
The Uppaal model: networks of timed automata

Parallel composition

Binary synchronization on complementary actions on channels (such as a! and a?): example transition 
(l1,m1,x=2,y=3.5,i=3) → (l2,m2,x=0,y=3.5,i=7)

Location invariants:

\[ \text{inv} ::= x < n \mid x \leq n \mid \text{inv} \& \& \text{inv} \]
Uppaal modeling language: declarations

• clocks:

    clock x1;

• action channels for binary synchronization; broadcast channels

    chan c; // an edge labelled c! synchronises with another labelled c?. A synchronisation pair is chosen nondeterministically if several combinations are enabled.

    broadcast chan b; // one sender c! can synchronise with an arbitrary number of receivers c?. Any receiver than can synchronise must do so. If there are no receivers, then the sender can still execute the c! action (broadcast sending is never blocking)

• constants:

    const int i = 2;
We want a component that receives a signal on channel a and immediately sends it on channel b:

- P and Q have same behaviour: no delay in urgent loc

![Diagram with nodes and transitions]

- Committed location
  - no delay in location
  - next transition must involve this location (no interference from other processes)

- Urgent
  - use of such loc reduce nb of clocks and interleaving, ie complexity of analysis
Building a model in Uppaal

Automata are templates, that may be parameterized.

Locations and edges are edited on the graphical representation (initial/committed locations, invariants, synchronization, guards, updates etc).
System definition in Uppaal

```c
/*
* For more details about this example, see
* "Automatic Verification of Real-Time Communicating Systems by Constraint Solving",
* by Wang Yi, Paul Pettersson and Mats Daniels. In Proceedings of the 7th International
*/

const N 5;       // # trains + 1
int[0,N] el;
chan appr, stop, go, leave;
chan empty, notempty, hd, add, rem;

clock x;

int[0,N] list[N], len, 1;

Train1:=Train(el, 1);
Train2:=Train(el, 2);
Train3:=Train(el, 3);
Train4:=Train(el, 4);
Queue:=IntQueue(el);

system
    Train1, Train2, Train3, Train4, Gate, Queue;
```
References

• Principles of Cyber-Physical Systems, R. Alur, MIT Press 2014, Chapter 7 (Timed models)


The projects

Choice between 2 subjects:

- model and verify a (simplified) pacemaker (in Uppaal)
- implement a reachability analysis for uncertain systems of ODEs (+ optional extension to simple hybrid systems)

Practical matters:

- monomes or binomes
- report and code sent on November 11 at latest (strict deadline), oral defense on November 14 (little time: go straight to the important messages + will possibly include some questions/small exercises on the lectures)
Mini-project: implantable pacemaker modeling

- four chambers: atria (French « oreillette ») and ventricles
- electrical activation in the right atrium → heart contracts & pump blood → the ventricles do the same
- When this does not work properly, a pacemaker may be used to regulate the heart rate (not too slow/quick, bounds between atria and ventricle contractions)
  - sense signals in the (right) atrium and ventricle
  - delivers stimulations, called paces, when necessary
The pacemaker model

Heart

HeartV

HeartA

Pacemaker

PaceV

PaceA
Mini project 2: Taylor-model based reachability analysis

Hybrid systems: generalization of timed processes

- Instantaneous mode transitions enabled by guards on the state, with possible discontinuous state change
- During timed transitions, continuous evolution of state and output variables, specified using differential equations
- Much richer behaviors

\[
\begin{align*}
\dot{T} &= -k_2 \\
T &\geq 15 \\
15 \leq T \leq 23
\end{align*}
\]

\[
\begin{align*}
\dot{T} &= k_1(23-T) \\
T &\leq 23 \\
T &\geq 21
\end{align*}
\]
Mini project 2: reachable set of continuous dynamics

\[
\begin{align*}
(S) \quad & \dot{x}(t) = f(x(t), u(t)) \\
& x(t) \in \mathbb{R}^n, \ x(0) \in I, \ u(t) \in U \subseteq \mathbb{R}^p
\end{align*}
\]

- \(u \in U\) non-deterministic uncertainty: disturbance rather than control
  - imprecise models, imprecisely known parameters or external influence
- \(x_f\) is reachable at time \(t\) if \(\exists x_0 \in I, \ \exists u:[0,t] \rightarrow U,\ \text{s.t.} \ x(t) = x_f\)
- The reachable set is \(\text{Reach}(I) = \{x_f \mid x_f \text{ is reachable}\}\)
- The timed reachable set is \(\text{Reach}_{[0,T]}(I) = \{x_f \mid x_f \text{ is reachable at time } t \leq T\}\)

Simulation:
- approximate sample of single behavior
- over finite time

Reachability
- over-approximate set-valued cover of all behaviors
- over finite or infinite time
Guaranteed set integration by Taylor-based numerical method

- Taylor-Lagrange expansion of the exact solution of $\dot{x} = f(x)$, $x(t_0) \in X_0$

  $x(t_0 + h) = x(t_0) + \sum_{i=1}^{n} \frac{h^i}{i!} \frac{d^i x}{dt^i}(t_0) + \frac{h^{n+1}}{(n+1)!} \frac{d^{n+1} x}{dt^{n+1}}(t_0 + \theta h)$, $0 < \theta < 1$

- can compute all Taylor coefficients on $X_0$, for instance by automatic differentiation + interval arithmetic:
  
  $x(t_0) \in X_0,$
  $\frac{dx}{dt}(t_0) = L^1_f(x)(x_0) = \{ f(x_0), \, x_0 \in X_0 \}$
  $\frac{d^2 x}{dt^2}(t_0) = L^2_f(x)(x_0) = \{ (\frac{\partial f}{\partial x}) f)(x_0), \, x_0 \in X_0 \}$
  $\frac{d^i x}{dt^i}(t_0) = L^i_f(x)(x_0) = L_f(L^{i-1}_f(x))(x_0)$

- but the last coefficient is a priori unknown: we need an a priori enclosure for $x(t)$, $t \in [t_0, t_0 + h]$ : this is given by the Lohner theorem (see ref in the project and next lecture)

You can start the project for uncertain ODEs now, we will come back to this reachability analysis and the case of hybrid systems in next course!
TD this week

- TD5. Build a small model (network of timed automata) in Uppaal
- Start your project (subjects on the web page dedicated to lab sessions)

Next week (hybrid systems):

- modeling and simulation
- verifying safety properties: reachability analysis