Course « Modelling Software-based Systems »

System Engineering and Hybrid Systems

Zheng Cheng and Dominique Méry
Telecom Nancy
Université de Lorraine

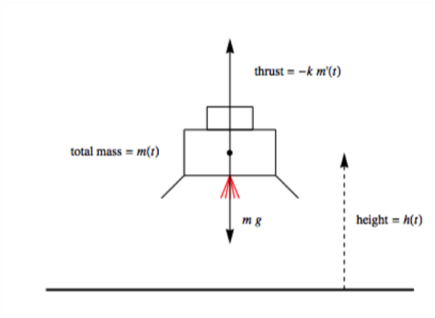
Année universitaire 2021-2022
13 décembre 2021

Summary

Current Section Summary

Current Subsection Summary

Problem of landing a spacecraft on the Moon



- Assuming that aerodynamic and gravitational forces of bodies other than the Moon are negligible, and lateral motion can be ignored.
- Accordingly, the descent trajectory is vertical, and the thrust vector is tangent to the trajectory.
- Because the spacecraft is near the Moon, we assume that the lunar acceleration of gravity has the constant value , that the relative velocity of the exhaust gases with respect to the spacecraft is constant, and that the mass rate is constrained by , where is constant and gives the maximum rate of change of the mass due to burning the fuel

Problem of landing a spacecraft on the Moon

- t is time
- m(t) is the mass of the spacecraft, which varies as fuel is burned
- m'(t) is the rate of change of mass, constrained by $-\mu \leq m'(t) \leq 0$
- $g = 12.63$ is the gravitational constant near the Moon
- k is a constant, the relative velocity of the exhaust gases with respect to the spacecraft
- $T(t) = -km'(t)$ the thrust
- $h(t)$ is the height with $h(t) \geq 0$.
- $v(t) = h'(t)$ the velocity
- $u(t) = m(t)$ the control function

Problem of landing a spacecraft on the Moon

Recalling assumptions, aerodynamic forces and gravitational forces of bodies other than the Moon are negligible and lateral motion is ignored. Thus the descent trajectory is vertical and the thrust vector is perpendicular to the ground. We also suppose that $m_0 = m(0) = M + F$ where is the mass of the spacecraft without fuel and F is the initial mass of fuel ; $m(t) > M$, since as we expect that the spacecraft will return to Earth, it needs some fuel for takeoff.
The equation of motion is given by applying Newton's law :

$$m(t) \times h''(t) = -g \times m(t) + T(t) \tag{1}$$

Current Subsection Summary

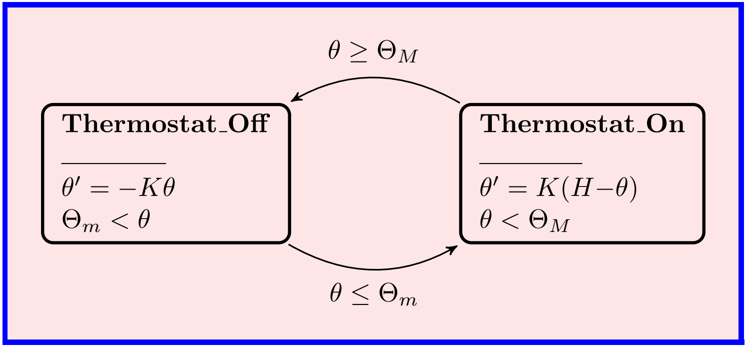
Problem of the thermostat

The temperature of a room is controlled through a thermostat, which continuously senses the temperature and turns a heater on and off. The temperature is defined by a differential equation (ODE). We used two diagrams for expressing this system and its behaviours. The behaviour is simply stated as :

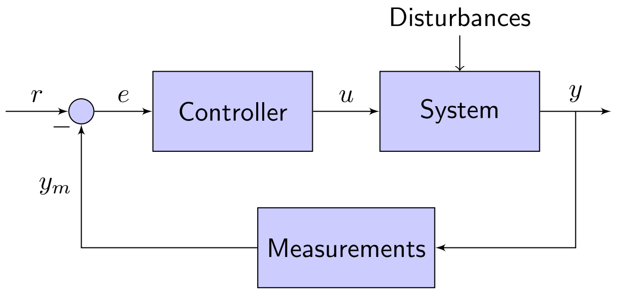
- When the heater is off, the temperature, denoted by the variable Θ, decreases according to the exponential function $\Theta(t) = \Theta_M e^{-Kt}$, where t is denoting the time, Θ_M the initial temperature, and K is a constant determined by the room.
- When the heater is on, the temperature is characterized by the function $\Theta(t) = \Theta_M e^{-Kt} + H(1 - e^{-Kt})$,where H is a constant that depends on the power of the heater.

Safety requirements for this system is that the control is such that $\forall t.t \; 0..+\infty \Rightarrow \Theta_m \leq \Theta(t) \leq \Theta_M$. The continuous variable Θ is denoting the state of the function $\Theta(t)$.

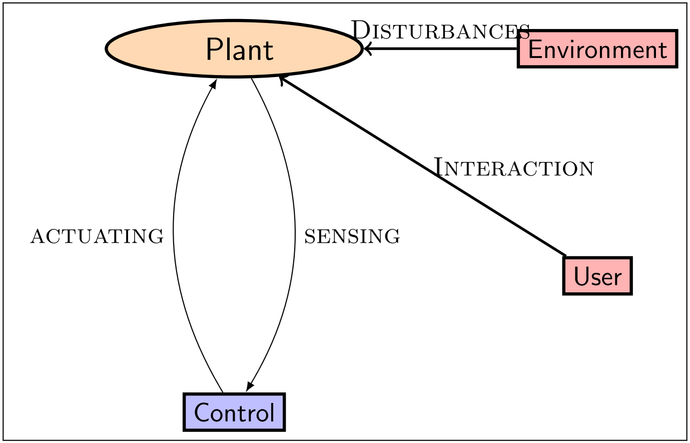
Problem of the thermostat



Current Subsection Summary


Disturbances
r
e
Controller
u
System
y
$-$
y_m
Measurements

Triptych Plant,Controller,Environment



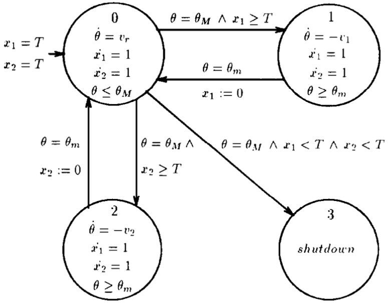
The TOULOUSE Pattern

- x_p, x_s and t are three variables which are modelling respectively the state of the plant at the current time, the control state of the discrete system and the current time.
- An event Behave updating the plant state according to some disturbances from the environment.
- An event Actuation updating the current state of the plant by integrating decision of the controller.
- An event Sensing collecting the data from the plant by delivering those values to the controller.
- An event Transition modelling the modification of the control state following the different control states identified in the problem.

Current Subsection Summary

System description

Our next example is a temperature control system for a heat producing reactor. The system controls the coolant temperature in a reactor tank by moving two independent control rods. The goal is to maintain the coolant (tank) between the temperatures Θ_M and Θ_m. When the temperature reaches its maximum value Θ_M the tank must be refrigerated with one of the rods. The temperature rises at a rate v_r, and decreases at rates v_1, and v_2 depending on which rod is being used. A rod can be moved again only if T time units have elapsed since the end of its previous movement. If the temperature of the coolant cannot decrease because there is no available rod, a complete shutdown is required. Fig. **??** shows the hybrid system of this example : variable Θ measures the temperature, and the values of clocks x_1 and x_2 represent the times elapsed since the last use of rod 1 and rod 2, respectively. The goal of is to ascertain that the reactor never reaches the critical temperature Θ_M without at least one of the rods available, or a shutdown has been initiated. An Event-B solution is given in [**?**].

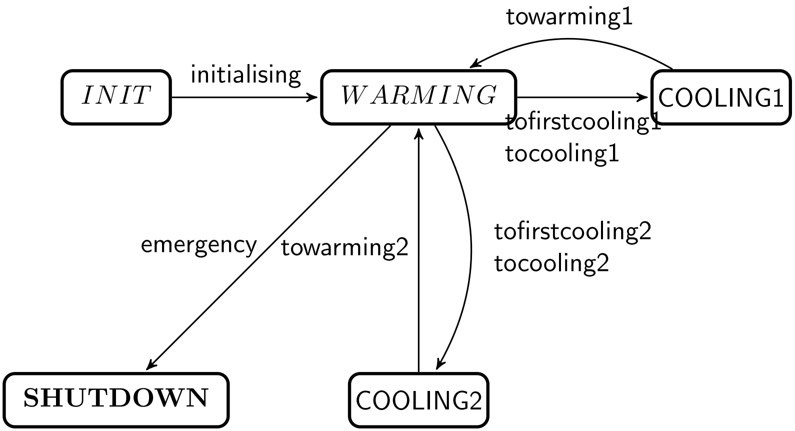


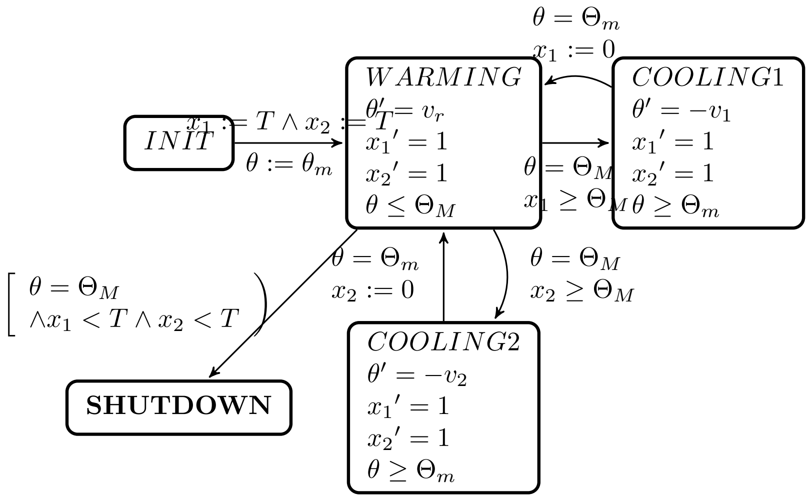
Run-time behaviours of a temperature control system for a heat producing reactor

First abstract view of the system

- initialising
- tofirstcooling1
- towarming1
- tocooling1
- tofirstcooling2
- towarming2
- tocooling2
- emergtency

First abstract view of the system

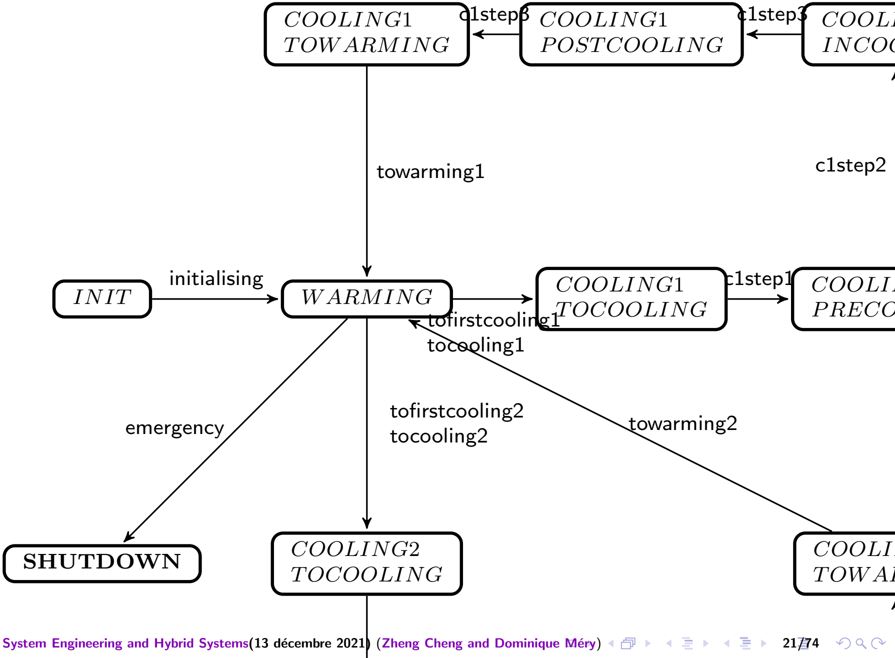



$INIT$
$x_1 := T \land x_2 := T$
$\theta := \theta_m$
$WARMING$
$\theta' = v_r$
${x_1}' = 1$
${x_2}' = 1$
$\theta \leq \Theta_M$
$\theta = \Theta_m$
$x_1 := 0$
$COOLING1$
$\theta' = -v_1$
${x_1}' = 1$
${x_2}' = 1$
$\theta \geq \Theta_m$
$\theta = \Theta_M$
$x_1 \geq \Theta_M$
$\theta = \Theta_m$
$x_2 := 0$
$\theta = \Theta_M$
$x_2 \geq \Theta_M$
$\left[\begin{array}{l} \theta = \Theta_M \\ \land x_1 < T \land x_2 < T \end{array} \right)$
SHUTDOWN
$COOLING2$
$\theta' = -v_2$
${x_1}' = 1$
${x_2}' = 1$
$\theta \geq \Theta_m$

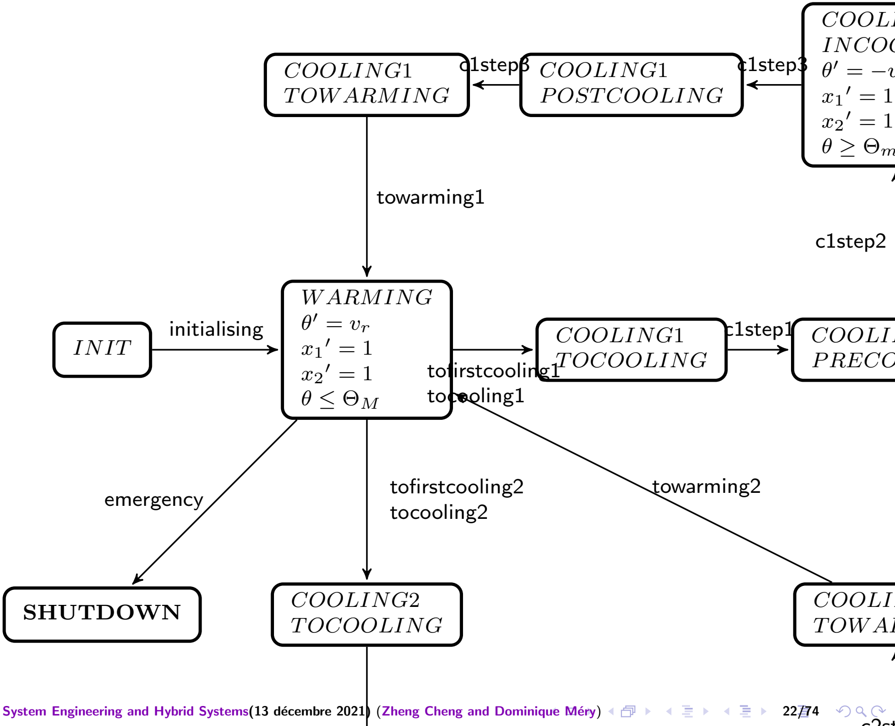
First refinement

- initialising
- tofirstcooling1 $\mathrm{REFINES}$ towarming1
  $\mathrm{REFINES}$ tocooling1
  $\mathrm{REFINES}$ tofirstcooling2
  $\mathrm{REFINES}$ tocoolin2
- emergency
- C1STEP3
- C1STEP4
- C2STEP1
- C1STEP1
- C1STEP2
- C2STEP2
- C2STEP3
- C2STEP4

Model



COOLING1 TOWARMING
c1step3
COOLING1 POSTCOOLING
c1step3
COOLI
INCOO
towarming1
c1step2
INIT
initialising
WARMING
COOLING1 TOCOOLING
c1step1
COOLI
PRECO
tofirstcooling1
tocooling1
emergency
tofirstcooling2
tocooling2
towarming2
SHUTDOWN
COOLING2 TOCOOLING
COOLI
TOWAI



$COOLING1$ $TOWARMING$
c1step3
$COOLING1$ $POSTCOOLING$
c1step3
$COOLI$ $INCOO$ $\theta' = -v$ ${x_1}' = 1$ ${x_2}' = 1$ $\theta \geq \Theta_m$
towarming1
c1step2
$INIT$
initialising
$WARMING$ $\theta' = v_r$ ${x_1}' = 1$ ${x_2}' = 1$ $\theta \leq \Theta_M$
tofirstcooling1
tocooling1
$COOLING1$ $TOCOOLING$
c1step1
$COOLI$ $PRECO$
emergency
tofirstcooling2
tocooling2
towarming2
SHUTDOWN
$COOLING2$ $TOCOOLING$
$COOLI$ $TOWA$

Current Section Summary

Systems

Systems

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

Systems

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

Interactive systems

- Interact with the environment
- Have subjective speed requirements ¿

Systems

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

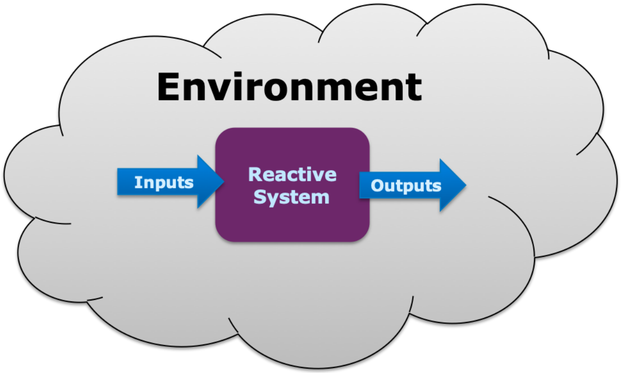
Interactive systems

- Interact with the environment
- Have subjective speed requirements ¿

Reactive Systems

- Interact with the environment
- Have subjective speed requirements

Reactive Systems



Time and Clock

Existence of a discrete clock :

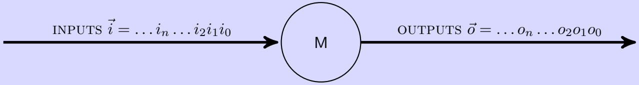
- Software cyclically activated,
- Inputs read at the cycle beginning (no inputs changes during the cycle)
- No cycle overlap
- Outputs delivered at cycle end

The cycle execution duration is considered to be null
Reasoning is possible

Reactive Systems

reactive model

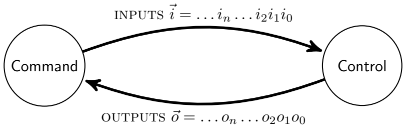
A reactive model M is receiving *inputs* and for each input it produces *outputs*.


INPUTS $\vec{i} = \ldots i_n \ldots i_2 i_1 i_0$
M
OUTPUTS $\vec{o} = \ldots o_n \ldots o_2 o_1 o_0$

- When an input i is received at time t_i, the model M is producing an output o at time t_o.
- $t_i < t_o$: the reaction takes time which is considered as small enough.

Control/Command

- Command : laws for the evolution of the dynamics as Newton's laws
  - command of actuators
  - time is continuous but only a discrete set of real values is taken into account.
- Control : laws for behaviours of the system
  - deciding the law of command to apply on the system.
  - verifying the correct behaviour of a system.
  - time is discrete


INPUTS $\vec{i} = \ldots i_n \ldots i_2 i_1 i_0$
Command
Control
OUTPUTS $\vec{o} = \ldots o_n \ldots o_2 o_1 o_0$

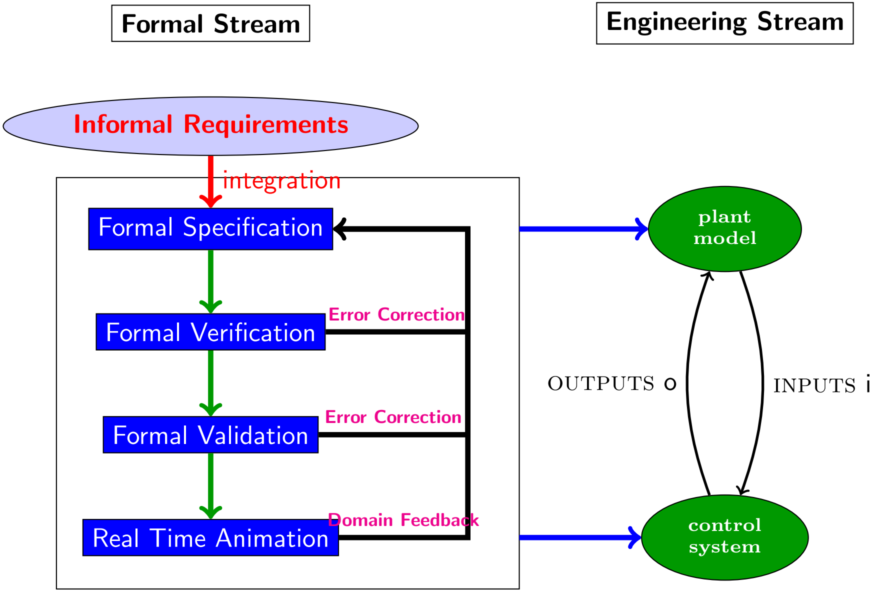
Controllers

- parallelism : the controller should take into account several hardware equipments
- determinism : when an input i is handled by the controler, the controller reacts always in the same way.
- real time : the system can not wait.
- safety ; the system is critical.

Synchronous abstraction

- facilitating the temporal reasoning
- two principles
  - simultaneity (no concurrency)
  - one time reference nothing does happen between two instants

General View of the : Formal Engineering Methodology



Current Section Summary

Hybrid Automata

Hybrid System

An hybrid system HS is a set of subsystems $\mathsf{SS}_1, \ldots \mathsf{SS}_n$ interacting through discrete and continuous variables where subsystems $\mathsf{SS}_1, \ldots \mathsf{SS}_n$ are either fully discrete systems, or fully continuous systems.

- using both *discrete* and *continuous* variables.
- assumptions on the possible transitions
- Hybrid systems are modelled by hybrid models which may have different forms.

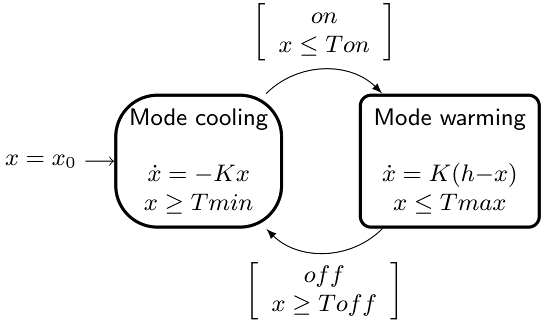
Hybrid automaton

A hybrid automaton is a collection $HA = (X, L, Init, Inv, f, E, Guard, Assign, \Sigma)$ where :

- $X \subseteq \mathbb{R}^n$: the continuous state space and $x = (x_1, x_2, \ldots, x_n)$, where $x_i \in \mathbb{R}, i \in \{1, 2, \ldots, n, \}$ represents the continuous dynamics.
- L is a finite set of locations.
- $Init \subseteq L \times X$ is a set of initial location state pairs.
- $Inv \in L \longrightarrow \mathcal{P}(X)$ assigns to each location $\ell (\in L)$ an invariant to be satisfied by the state x while in the location ℓ.
- $f \in L \longrightarrow (XL \longrightarrow \mathbb{R}^n)$ assigns to each location ℓ a continuous vector field f_ℓ such that the state $x \in X$ should satisfy $\frac{d}{dt}x = f_\ell(x)$.
- $E \subseteq L \times \Sigma \times L$ is the set of transitions, also called *switches*, where Σ is a set of transition labels.
- $Guard \in E \longrightarrow \mathcal{P}(X)$ assigns to each transition a guard that has to be satisfied by the state x if the transition is taken.
- $Assign \in E \longrightarrow (X \longrightarrow X)$ assigns to each transition an assignment that may alter the state x when the transition is taken.

hybrid systems as hybrid automata : the thermostat

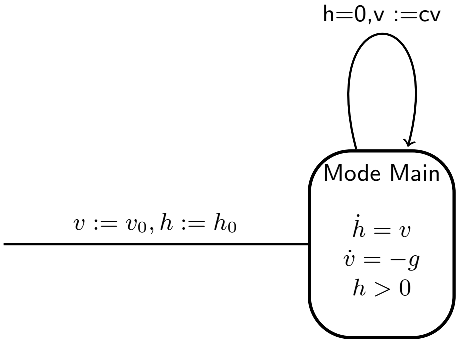
- The thermostat with two possible modes.
- The first mode is defining the warming phase of system and is characterized by a linear differential l equation.
- The second mode is defining as well the cooling phase.



$\left[\begin{array}{c} on \\ x \leq Ton \end{array}\right]$
Mode cooling
$\dot{x} = -Kx$
$x \geq Tmin$
Mode warming
$\dot{x} = K(h-x)$
$x \leq Tmax$
$x = x_0$
$\left[\begin{array}{c} off \\ x \geq Toff \end{array}\right]$

hybrid systems as hybrid automata : the bouncing ball

- The bouncing ball following the laws for the dynamics of Newton.
- The ball pushes on the floor and the floor responds by pushing back on the ball with an equal amount of force.



- The push the ball receives from the floor causes it to rebound, meaning it bounces up.
- The moving ball again has kinetic energy.
- This is an example of Newton's Third Law of Motion : Action/Reaction. The principles are translated by one mode hybrid automaton.

hybrid systems as hybrid automata : comments

- Hybrid automata are clearly structures that organise the different modes or phases of the hybrid system under consideration.
- On the two examples :
  - the nodes are encapsulating the differential equations and the piece of system
  - the transitions are actions which are modelling the instantaneous switching from one mode to another mode.
  - Each mode is describing a *behaviour* of the system between two instants.

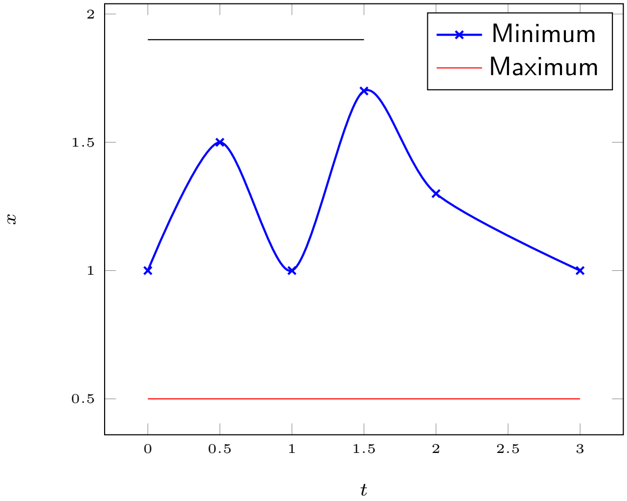
Observation of an hybrid system

- Modelling an hybrid systems requires to handle discrete features as well as continuous features.
- The interactions among the different *active* parts involve the use of *sensors* and *actuators*
- and other possible interactions are related to the possible user who is possibly executing an *operation*.
- *disturbances* are possible observable events.

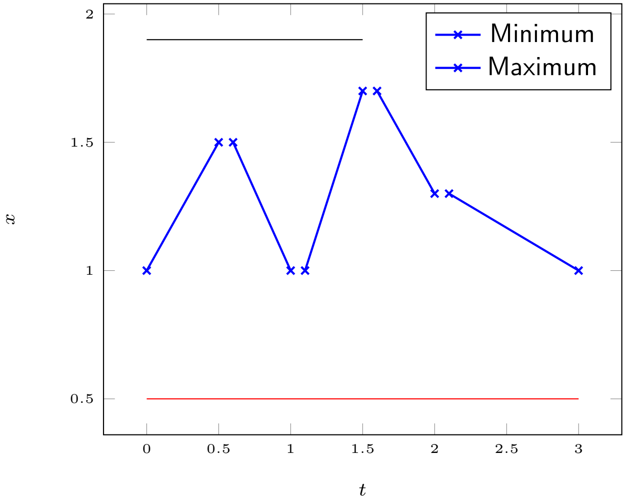
- Hybrid systems are generally related to real systems in different domains with specific laws as Newton's laws of the classical mechanics.
- The first abstraction is to consider that the phenomenon under consideration is characterized by a state variable namely $x \in Time \longrightarrow D$ where $Time$ is modelling the time and D is a set for values of the current domain.
- D is generally a Banach space (a complete normed vector space).
- The state variable x is modified by *events* observed during the time of the system and the *now* variable is modelling the current time and we consider that the past of the state variable x can not be modified but the future of x can be constrained by some events updating it.

The domain D is generally of the form $\mathbb{R}^n$ and is able to model features as temperature, level of a tank, density, pressure, . . . and it is equipped with mathematical properties that make possible to state laws using differential equations. A very important issue is the continuity of the description or more generally the fact that the phenomenon is viewed as sequence of pieces of curves with possible discontinuity regions. For instance, we use the following diagram **??** shows a blue continuous curve which is between two other curves minimum and maximum. The curve is defined in a graphical way and is build using analytic functions and it is the concatenation of several curves which are assembled to produce the general view of the evolution of a given value at time t as $x(t)$. ite

Example of a curve



Example of a disturbed curve



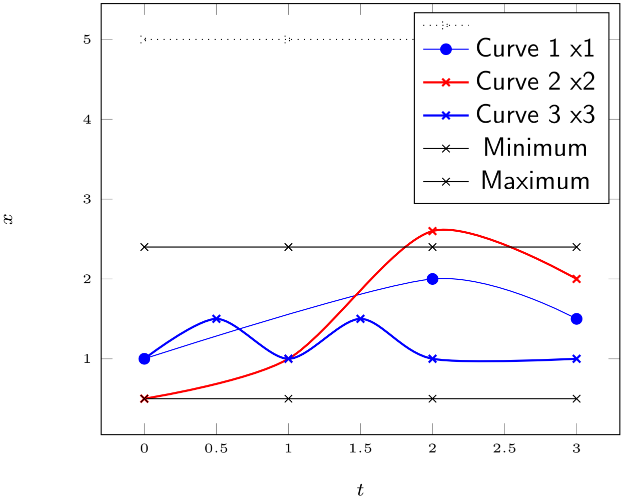
Properties for hybrid models

Different cases should be considered with respect to the required assertions :

- A safety property $safe(t1, t2, x, D, S)$ for x is prescribing that safe values of x in the interval $[t1, t2]$ are in S, a subset of D :
$$\begin{cases} S \subseteq D \\ x \in \mathbb{R}^+ \longrightarrow D \\ \forall t \in [t1, t2] : x(t) \in S \end{cases}$$
- A stabilisation property $stable(t1, t2, x, D, U, E, S)$ is prescribing that U (*Unstable)*) and S (*Stable*) are two disjunct subsets of D and that $x(t1) \in U$, $x(t2) \in S$ and the function x between $t1$ and $t2$ is continuously evolving from $t1$ to $t2$:
$$\begin{cases} U \subseteq D, S \subseteq D, E \subseteq D, U \subseteq E, S \subseteq E \\ x \in \mathbb{R}^+ \longrightarrow D \\ x(t1) \in U \\ x(t2) \in S \\ x \in \mathcal{C}_D((t1, t2]) \end{cases}$$

Examples of safety properties

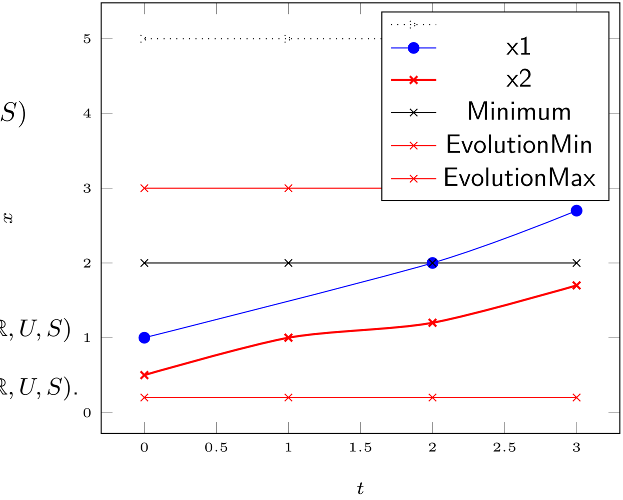
- when one define a domain $\mathbb{R}$, a safety property $S = [0.5, 2.4]$ which is stating that three possible curves between $t1 = 0$ and $t2 = 3$ should be satisfying that $x(t) \in S$.



- Curves 1 $(safe(0,3,x1,\mathbb{R},S)$) and 3 $(safe(0,3,x3,\mathbb{R},S)$) are blue and are satisfying what is the safety property.
- Curve 2 is not correct : $safe(0,3,x2,\mathbb{R},S)$ is not satisfied for $x2(2)$

Examples of stabilisation properties

- $stable(t1, t2, x, \mathbb{R}, U, S)$ with the following definitions :
  - $t1 = 0$, $t2 = 3$, $U =]-\infty, 2]$,
  - $S =]2, +\infty[$. $x1$ is satisfying $stable(t1, t2, x1, \mathbb{R}, U, S)$ but not $stable(t1, t2, x2, \mathbb{R}, U, S)$.



Current Section Summary

LUSTRE Programs

- A Lustre program or subprogram is called a node.
- Lustre is a functional language operating on streams
  - a stream is a finite or infinite sequence of values.
  - Values of a stream are of the same type called the type of the stream.
- The behavior of a LUSTRE program is cyclic
- At the nth execution cycle, all the involved streams take their nth value.
- A node defines one or several output parameters as functions of one or several input parameters and parameters are streams.

Semantical Concepts for Reactive Programming

4	4	4	...	4	...
`x`	x_0	x_1	...	x_n	...
`y`	y_0	y_1	...	y_n	...
`x+y`	x_0+y_0	x_1+y_1	...	x_n+y_n	...

`x`	x_0	x_1	...	x_n	...
`pre x`	NIL	x_0	...	x_{n-1}	...

`x`	x_0	x_1	...	x_n	...
`y`	y_0	y_1	...	y_n	...
`x->y`	x_0	y_1	...	y_n	...

- `nat = 0 -> 1 -> pre nat`

`h`	**true**	**false**	**true**	**true**	**false**
`x`	x_0	x_1	...	x_n	...
`x when h`	x_0	—	x_2	x_3	—

LUSTRE Programs

- **A LUSTRE program is called a node NODE.**
- **A LUSTRE program denotes an infinite sequence of values as** $(x_0\ x_1\ x_2\ \ldots)$
- **Two operators of programs :**
  - $\mathbf{pre}$
  - $\longrightarrow$
- $\forall n \geq 0. CUP_{n+1} = CUP_n + 1$ **is written as follow** $CUP = 0 \longrightarrow (1+\mathbf{pre}(CUP))$
- **and will produce the sequence** $(0\ 1\ 2\ \ldots)$**.**
- $FIB = 0 \longrightarrow 1 \longrightarrow (\mathbf{pre}(FIB)+\mathbf{pre}(\mathbf{pre}(FIB)))$

LUSTRE Programs

-
  ```
  node   EDGE(X : bool)   returns   (Y : bool)
  let
    Y = false → X and not pre(X)
  tel
  ```
  désigne la suite $(false, x_1 \wedge \neg x_0, x_2 \wedge \neg x_1, \ \ldots)$
- **Counter**
  $C = 0 \longrightarrow \mathbf{pre}(C){+}1$ **returnd the sequence of naturals**
  $C = 0 \longrightarrow if\ \ X\ \ then\ \mathbf{pre}(C){+}1\ else\ \mathbf{pre}(C)$
  counts the number of occurrences of X **which are true.**
  We do notvtake into account the initial value
  $PC = 0 \longrightarrow \mathbf{pre}(C)$
  $C = if\ \ X\ \ then\ \ PC{+}1\ \ else\ \ PC$

Counter

-
  ```
  $nodeCOUNTER(init, incr : int; X, reset : bool)returns(C : int)$
  $let$
    $PC = init -> \ pre\ C$
    $C = if\ reset\ then\ init$
      $else\ if\ X\ then\ (PC+incr)$
      $elsePC;$
  $tel$
  ```
- $odds = COUNTER(0, 2, true, true-> false)$ **définit les entiers impairs.**

LUSTRE Constructs

- **Two operators over programs**
  - **pre**
  - $\longrightarrow$
- $X\ when\ B$
- $current\ X$
- $assert$
  - $assert\ not\ (x\ and\ y\ )$
  - $assert\ (true- > not(x\ and\ pre(x)))$

Counter

```
node COUNTER(init,incr:int; X,reset:bool) returns (C:int)
let
   PC=init->  pre C
   C = if reset then init
       else if X then (PC+incr)
       else PC;
tel
```

- $odds = COUNTER(0, 2, true, true- > false)$ defines the odd natural numbers.

LUSTRE constructs and operators

- **Two operators over programs**
  - **pre**
  - $\longrightarrow$
- $X\ when\ B$ **: filter of** X **when** B **is true.**
- $current\ X$ **: interpolation of** X
- $and, not, or, xor, \ldots$ **are boolean operators over streams.**
- $assert$
  - $assert\ not\ (x\ and\ y\ )$
  - $assert\ (true- > not(x\ and\ pre(x)))$
- **reusing nodes**

```
node FALLING_EDGE(X:bool) returns (Y:bool)
let
   Y= EDGE(not X);
tel
```

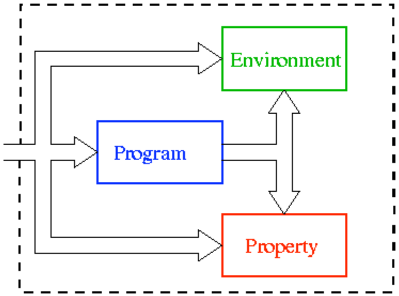
Integration

- f a function defined over time mapping to real values and we want to compute the integral of f.
- Two values are received by programs $F_n = f(x_n)$ and $x_{n+1} = x_n + STEP_{n+1}$
- Computing Y : $Y_{n+1} = Y_n + (F_n + F_{n+1}) \cdot STEPn+1/2$
- The value of Y is a data

```
node integration(F,STEP,init:real) returns (Y:real)
let
   Y= init -> pre(Y)+ ((F + pre(F))*STEP)/2.0;
tel
```

Verification by observers

- Description of the property to check and the assumptions over the environment.
- An observer of a safety property is a program using as input input/output of the program to check and decide by emitting a signal at any time if the property is violated or not.



Programming a switch

- Transforming a signal `level` by a switch used as follows :
  - two possible signals as input `set` or `reset`
  - an initial value `initial`
  - when a set signal occurs, the level is set to true.
  - when a reset signal occurs, the level is set to false.
  - when no signal occurs, the level is unchanged.pas
- a signal is modelled as a boolean

```
node SWITCH1(set,reset,initial: bool) returns (level:bool)
let
   level = initial -> if set the true
                      else if reset then false
                      else pre(level);
tel
```

- However, this program dpes not model a switch with one button.

```
state =  SWITCH1(change,change,true)
```

Programming a one button switch

-
  ```
  node SWITCH(set,reset,initial: bool) returns (level:bool)
  let
     level = initial -> if set and not pre(level)  then true
                        else if reset then false
                        else pre(level);
  tel
  ```
- **Verification :**

  ```
  node verification(set,reset,initial: bool) returns (ok:bool)
  let
     level = SWITCH(set,reset,initial);
     level1 = SWITCH(set,reset,initial);
     ok = (level = level1);
     assert  not(set  and reset)
  tel
  ```
- **the two versions are identical aq long as the two buttons are different.**

Current Section Summary

Modelling systems in Event-B

Modelling systems in Event-B

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

Modelling systems in Event-B

```
MACHINE
  $m$
SEES
  $c$
VARIABLES
  $x$
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END
```

c defines the static environment for the proofs related to m : sets, constants, axioms, theorems $\Gamma(m)$.

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m : sets, constants, axioms, theorems $\Gamma(m)$.

$$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$$

Modelling systems in Event-B

MACHINE
  m
SEES
  c
VARIABLES
  x
INVARIANT
  $I(x)$
THEOREMS
  $Q(x)$
INITIALISATION
  $Init(x)$
EVENTS
  $\ldots e$
END

c defines the static environment for the proofs related to m : sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e$:
$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

Modelling systems in Event-B

MACHINE
m
SEES
c
VARIABLES
x
INVARIANT
$I(x)$
THEOREMS
$Q(x)$
INITIALISATION
$Init(x)$
EVENTS
$\ldots e$
END

c defines the static environment for the proofs related to m : sets, constants, axioms, theorems $\Gamma(m)$.

$\Gamma(m) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e$:
$\Gamma(m) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

$\Gamma(m) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

Modelling systems in Event-B

```
MACHINE
   $m$
SEES
   $c$
VARIABLES
   $x$
INVARIANT
   $I(x)$
THEOREMS
   $Q(x)$
INITIALISATION
   $Init(x)$
EVENTS
   $\dots e$
END
```

c defines the static environment for the proofs related to m : sets, constants, axioms, theorems $\Gamma(m)$.

$$\Gamma(m) \vdash \forall x \in Values : \text{INIT}(x) \Rightarrow \text{I}(x)$$

$\forall e$:

$$\Gamma(m) \vdash \forall x, x', u \in Values : \text{I}(x) \wedge R(u, x, x') \Rightarrow \text{I}(x')$$

$$\Gamma(m) \vdash \forall x \in Values : \text{I}(x) \Rightarrow \text{Q}(x)$$

```
$e$
ANY
   $u$
WHERE
   $G(x, u)$
THEN
   $x : |(R(u, x, x'))$
END
```

or e is **observed** $x \xrightarrow{e} x'$

Event B Structure and Proofs

CONTEXT	**MACHINE**
ctxt_id_2	*machine_id_2*
EXTENDS	**REFINES**
ctxt_id_1	*machine_id_1*
SETS	**SEES**
s	*ctxt_id_2*
CONSTANTS	**VARIABLES**
c	v
AXIOMS	**INVARIANTS**
$A(s,c)$	$I(s,c,v)$
THEOREMS	**THEOREMS**
$T_c(s,c)$	$T_m(s,c,v)$
END	**VARIANT**
	$V(s,c,v)$
	EVENTS
	EVENT e
	ANY x
	WHERE $G(s,c,v,x)$
	THEN
	$v :\mid BA(s,c,v,x,v')$
	END
	END

Invariant preservation	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow I(s,c,v')$
Event feasibility	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant modelling progress	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow V(s,c,v') < V(s,c,v)$
Theorems	$A(s,c) \Rightarrow T_c(s,c)$ $A(s,c) \wedge I(s,c,v)$ $\Rightarrow T_m(s,c,v)$

Election in One Shot : Building a Spanning Tree

Election in One Shot : Building a Spanning Tree

```
MACHINE
  ELECTION
SEES  GRAPH
VARIABLES  $rt, ts, ok$
INVARIANT
    $rt \in ND$
    $ts \in ND \leftrightarrow ND$
    $ok \in BOOL$
    $ok = TRUE$
          $\Rightarrow$spanning $(rt, ts, gr)$
INITIALISATION  $Init(x)$
EVENT election  $\widehat{=}$
WHEN
  $ok = FALSE$
THEN
    $rt, ts : |($spanning $(rt', ts', gr))$
    $ok := TRUE$
  ENDEND
```

Election in One Shot : Building a Spanning Tree

MACHINE
$ELECTION$
SEES $GRAPH$
VARIABLES rt, ts, ok
INVARIANT

$rt \in ND$
$ts \in ND \leftrightarrow ND$
$ok \in BOOL$
$ok = TRUE$
$\Rightarrow \mathsf{spanning}(rt, ts, gr)$

INITIALISATION $Init(x)$
EVENT election $\widehat{=}$
WHEN
$ok = FALSE$
THEN
$rt, ts : |(\mathsf{spanning}(rt', ts', gr))$
$ok := TRUE$
END **END**

CONTEXT $GRAPH$
$(ax1)$ $gr \subseteq ND \times ND$
$(ax2)$ $gr = gr^{-1}$
$(ax3)$ $\mathsf{dom}(gr) = ND$
$(ax4)$ $\mathsf{id}(ND) \cap gr = \varnothing$

$$(ax5)\ \forall p \cdot \begin{pmatrix} p \subseteq ND \ \wedge \\ p \subseteq t^{-1}[p] \\ \Longrightarrow \\ p = \varnothing \end{pmatrix}$$

$(Th1) fn \in ND \to (ND \twoheadrightarrow ND)$
$\forall (r, t) \cdot$

$$\begin{pmatrix} r \in ND \ \wedge \\ t \in ND \twoheadrightarrow ND \\ \Longrightarrow \\ (t = fn(r) \ \Leftrightarrow\ \mathsf{spanning}(r, t, gr)) \end{pmatrix}$$

Current Section Summary

Expressing models in the event B notation

- Models are defined in two ways :
  - an abstract machine
  - a refinement of an existing model
- Models use **constants** which are defined in structures called **contexts**
- B structures are related by the three possible relations :
  - the **sees** relationship for expressing the use of constants, sets satisfying axioms and theorems.
  - the **extends** relationship for expressing the extension of contexts by adding new constants and new sets
  - the **refines** relationship stating that a B model is refined by another one.

Machines and contexts

Machines

- **REFINES**
- **SEES** a context
- **VARIABLES** of the model
- **INVARIANTS** satisfied by the variables
- **THEOREMS** satisfied by the variables
- **VARIANT**
- **EVENTS** modifying the variables

Context

- **EXTENDS** another context
- **SETS** declares new sets
- **CONSTANTS** define a list of constants
- **AXIOMS** define the properties of constants and sets
- **THEOREMS** list the theorems which should be derived from axioms

Machines for Event B

```
MACHINE
  $m$
REFINES
  $am$
SEES
  $c$
VARIABLES
  $x$
INVARIANTS
  $I(x)$
THEOREMS
  $T(x)$
VARIANT
  $< variant >$
EVENTS
  $< event >$
END
```

Contexts for Event B

```
CONTEXTS
    c
EXTENDS
    ac
SETS
    c
CONSTANTS
    k
AXIOMS
    ....
THEOREMS
    T(x)
END
```

Events

Event : E	Before-After Predicate
BEGIN $x :\| P(x,x')$ **END**	$P(x,x')$
WHEN $G(x)$ **THEN** $x :\| P(x,x')$ **END**	$G(x) \quad \wedge \quad P(x,x')$

ANY t
 WHERE $G(t,x)$
 THEN $x :\| P(x,x',t)$ **END**
	$\exists t \cdot (\, G(t,x) \quad \wedge \quad P(x,x',t)\,)$

Guards of event

Event : E	Guard : grd(E)
BEGIN S **END**	$TRUE$
WHEN $G(x)$ **THEN** T **END**	$G(x)$
ANY t **WHERE** $G(t,x)$ **THEN** T **END**	$\exists t \cdot G(t,x)$

Proof obligations for a B model

	Proof obligation
(INV1)	$\Gamma(s,c) \vdash Init(x) \Rightarrow I(x)$
(INV2)	$\Gamma(s,c) \vdash I(x) \wedge BA(e)(x,x') \Rightarrow I(x')$
(DEAD)	$\Gamma(s,c) \vdash I(x) \Rightarrow (\mathsf{grd}(e_1) \vee \ldots \mathsf{grd}(e_n))$
(SAFE)	$\Gamma(s,c) \vdash I(x) \Rightarrow A(x)$
(FIS)	$\Gamma(s,c) \vdash I(x) \wedge \mathsf{grd}\,(E) \Rightarrow \exists x' \cdot P(x,x')$

Current Section Summary

Simple Form of an Event

- An event of the simple form is denoted by :

```
$< event\_name > \widehat{=}$
  WHEN
    $< condition >$
  THEN
    $< action >$
  END
```

where

- $< event\_name >$ is an identifier
- $< condition >$ is the firing condition of the event
- $< action >$ is a generalized substitution (parallel "assignment")

Non-deterministic Form of an Event

- An event of the non-deterministic form is denoted by :

```
$< event\_name > \;\widehat{=}$
  ANY $< variable >$ WHERE
    $< condition >$
  THEN
    $< action >$
  END
```

where

- $< event\_name >$ is an identifier
- $< variable >$ is a (list of) variable(s)
- $< condition >$ is the firing condition of the event
- $< action >$ is a generalized substitution (parallel "assignment")

Shape of a Generalized Substitution

A generalized substitution can be

- Simple assignment : $x := E$
- Generalized assignment : $x : P(x, x')$
- Set assignment : $x :\in S$

,

- Parallel composition : $\begin{array}{c} T \\ \cdots \\ U \end{array}$

Invariant Preservation Verification (0)

$$\begin{array}{l}\text{INVARIANT} \quad \wedge \quad \text{GUARD} \\ \Longrightarrow \\ \text{ACTION establishes INVARIANT}\end{array}$$

Invariant Preservation Verification (1)

- Given an event of the simple form :

```
EVENT EVENT  ≙
  WHEN
    $G(x)$
  THEN
    $x := E(x)$
  END
```

and invariant $I(x)$ to be preserved, the statement to prove is :

$$\boxed{I(x) \;\wedge\; G(x) \;\Longrightarrow\; I(E(x))}$$

Invariant Preservation Verification (2)

- Given an event of the simple form :

```
EVENT EVENT  $\widehat{=}$
  WHEN
    $G(x)$
  THEN
    $x :| P(x, x')$
  END
```

and invariant $I(x)$ to be preserved, the statement to prove is :

$$\boxed{I(x) \;\wedge\; G(x) \;\wedge\; P(x, x') \;\Longrightarrow\; I(x')}$$

Invariant Preservation Verification (3)

- Given an event of the simple form :

```
EVENT EVENT ≙
  WHEN
    $G(x)$
  THEN
    $x :\in S(x)$
  END
```

and invariant $I(x)$ to be preserved, the statement to prove is :

$$\boxed{I(x) \quad \wedge \quad G(x) \quad \wedge \quad x' \in S(x) \quad \Longrightarrow \quad I(x')}$$

Invariant Preservation Verification (4)

- Given an event of the non-deterministic form :

```
EVENT EVENT  $\widehat{=}$
    ANY $v$ WHERE
        $G(x, v)$
    THEN
        $x := E(x, v)$
    END
```

and invariant $I(x)$ to be preserved, the statement to prove is :

$$\boxed{I(x) \quad \wedge \quad G(x, v) \quad \Longrightarrow \quad I(E(x, v))}$$

Refinement Technique (1)

- Abstract models works with variables x, and concrete one with y
- A gluing invariant $J(x, y)$ links both sets of vrbls
- Each abstract event is refined by concrete one (see below)

Refinement Technique (2)

- Some new events may appear : they refine “skip”
- Concrete events must not block more often than the abstract ones
- The set of new event alone must always block eventually

Correct Refinement Verification (1)

- Given an abstract and a corresponding concrete event

```
EVENT EVENT  ≘
  WHEN
    $G(x)$
  THEN
    $x := E(x)$
  END
```

```
EVENT EVENT  ≘
  WHEN
    $H(y)$
  THEN
    $y := F(y)$
  END
```

and invariants $I(x)$ and $J(x, y)$, the statement to prove is :

$$\boxed{I(x) \;\wedge\; J(x,y) \wedge H(y) \Longrightarrow G(x) \;\wedge\; J(E(x), F(y))}$$

Correct Refinement Verification (2)

- Given an abstract and a corresponding concrete event

```
EVENT EVENT  ≙
   ANY $v$ WHERE
      $G(x,v)$
   THEN
      $x := E(x,v)$
   END
```

```
EVENT EVENT  ≙
   ANY $w$ WHERE
      $H(y,w)$
   THEN
      $y := F(y,w)$
   END
```

$$\boxed{\begin{array}{l} I(x) \quad \wedge \quad J(x,y) \quad \wedge \quad H(y,w) \\ \Longrightarrow \\ \exists v \cdot (\, G(x,v) \quad \wedge \quad J(E(x,v), F(y,w))\,) \end{array}}$$

Correct Refinement Verification (3)

- Given a NEW event

```
EVENT EVENT  $\widehat{=}$
  WHEN
    $H(y)$
  THEN
    $y := F(y)$
  END
```

and invariants $I(x)$ and $J(x,y)$, the statement to prove is :

$$\boxed{I(x) \;\wedge\; J(x,y) \;\wedge\; H(y) \;\Longrightarrow\; J(x,F(y))}$$

General form of an event

```
EVENT e
  ANY  $t$
  WHERE
    $G(c,s,t,x)$
  THEN
    $x :| (P(c,s,t,x,x'))$
  END
```

- c et s are constantes and visible sets by e
- x is a state variable or a list of variabless
- $G(c,s,t,x)$ is the condition for observing e.
- $P(c,s,t,x,x')$ is the assertion for the relation over x and x'.
- $BA(e)(c,s,x,x')$ is the *before-after* relationship for e and is defined by $\exists t.G(c,s,t,x) \ \wedge \ P(c,s,t,x,x')$.

General form of proof obligations for an event e

Proofs obligations are simplified when they are generated by the module called POG and goals in sequents as $\Gamma \vdash G$:

1. $\Gamma \vdash G_1 \wedge G_2$ is decomposed into the two sequents $\begin{array}{l}(1)\Gamma \vdash G_1\\(2)\Gamma \vdash G_2\end{array}$
2. $\Gamma \vdash G_1 \Rightarrow G_2$ is transformed into the sequent $\Gamma, G_1 \vdash G_2$

Proof obligations in Rodin

- *INIT/I/INV* : $C(s,c), INIT(c,s,x) \vdash I(c,s,x)$
- *e/I/INV* : $C(s,c), I(c,s,x), G(c,s,t,x), P(c,s,t,x,x') \vdash I(c,s,x')$
- *e/act/FIS* : $C(s,c), I(c,s,x), G(c,s,t,x) \vdash \exists x'.P(c,s,t,x,x')$

Current Section Summary

Event- B versus B-System

- Modelling reactive systems : systems versus software
- Event-B $\Rightarrow$ B $\Rightarrow$ B-System
- Event-B $\Leftarrow$ B $\Leftarrow$ B-System

Components in B-System

Components in B-System

```
MACHINE
   m
SEES
   c
SETS
   s
CONSTANTS
   cst
PROPERTIES
   p
VARIABLES
   x
INVARIANT
   I(x)
ASSERTIONS
   a
INITIALISATION
   Init(x)
OPERATIONS
   ...e
END
```

Components in B-System

```
MACHINE
   m
SEES
   c
SETS
   s
CONSTANTS
   cst
PROPERTIES
   p
VARIABLES
   x
INVARIANT
   I(x)
ASSERTIONS
   a
INITIALISATION
   Init(x)
OPERATIONS
   ...e
END
```

c defines the static environment for the proofs related to **m** : sets, constants, axioms, theorems $\Gamma(\mathbf{m})$.

Components in B-System

```
MACHINE
  m
SEES
  c
SETS
  s
CONSTANTS
  cst
PROPERTIES
  p
VARIABLES
  x
INVARIANT
  I(x)
ASSERTIONS
  a
INITIALISATION
  Init(x)
OPERATIONS
  ...e
END
```

c defines the static environment for the proofs related to **m** : sets, constants, axioms, theorems $\Gamma(\mathbf{m})$.

$\Gamma(\mathbf{m}) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

Components in B-System

```
MACHINE
   m
SEES
   c
SETS
   s
CONSTANTS
   cst
PROPERTIES
   p
VARIABLES
   x
INVARIANT
   I(x)
ASSERTIONS
   a
INITIALISATION
   Init(x)
OPERATIONS
   ...e
END
```

c defines the static environment for the proofs related to **m** : sets, constants, axioms, theorems $\Gamma(\mathbf{m})$.

$\Gamma(\mathbf{m}) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(\mathbf{m}) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

Components in B-System

MACHINE
  m
SEES
  c
SETS
  s
CONSTANTS
  cst
PROPERTIES
  p
VARIABLES
  x
INVARIANT
  $I(x)$
ASSERTIONS
  a
INITIALISATION
  $Init(x)$
OPERATIONS
  $\dots e$
END

c defines the static environment for the proofs related to **m** : sets, constants, axioms, theorems $\Gamma(\mathbf{m})$.

$\Gamma(\mathbf{m}) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e$:

$\Gamma(\mathbf{m}) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x')$

$\Gamma(\mathbf{m}) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

Components in B-System

```
MACHINE
   m
SEES
   c
SETS
   s
CONSTANTS
   cst
PROPERTIES
   p
VARIABLES
   x
INVARIANT
   $I(x)$
ASSERTIONS
   $a$
INITIALISATION
   $Init(x)$
OPERATIONS
   $\dots e$
END
```

c defines the static environment for the proofs related to **m** : sets, constants, axioms, theorems $\Gamma(\mathbf{m})$.

$\Gamma(\mathbf{m}) \vdash \forall x \in Values : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$

$\forall e :$

$\Gamma(\mathbf{m}) \vdash \forall x, x', u \in Values : \mathrm{I}(x) \wedge R(u, x, x') \Rightarrow \mathrm{I}(x'$

$\Gamma(\mathbf{m}) \vdash \forall x \in Values : \mathrm{I}(x) \Rightarrow \mathrm{Q}(x)$

```
$e$
ANY
   $u$
WHERE
   $G(x, u)$
THEN
   $x :| (R(u, x, x'))$
END
```

or e is **observed** $x \xrightarrow{e} x'$

Current Section Summary

Properties for system

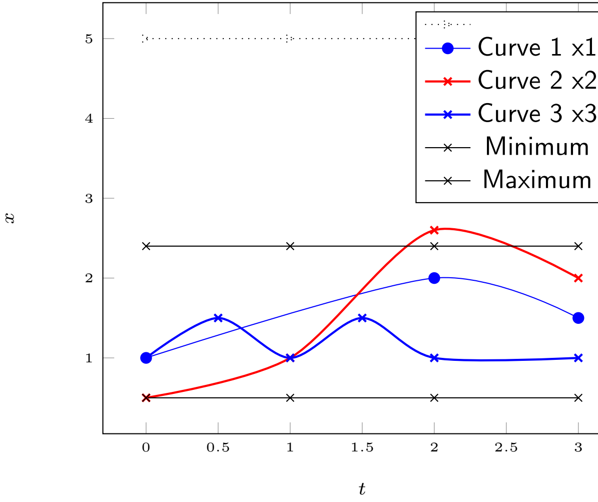
- A safety property $safe(t1, t2, x, D, S)$ for x is prescribing that safe values of x in the interval $[t1, t2]$ are in S, a subset of D :
$$\begin{cases} S \subseteq D \\ x \in \mathbb{R}^+ \longrightarrow D \\ \forall t \in [t1, t2] : x(t) \in S \end{cases}$$
- A stabilisation property $stable(t1, t2, x, D, U, E, S)$ is prescribing that U (*Unstable*)) and S (*Stable*) are two disjunct subsets of D and that $x(t1) \in U$, $x(t2) \in S$ and the function x between $t1$ and $t2$ is continuously evolving from $t1$ to $t2$:
$$\begin{cases} U \subseteq D, S \subseteq D, E \subseteq D, U \subseteq E, S \subseteq E \\ x \in \mathbb{R}^+ \longrightarrow D \\ x(t1) \in U \\ x(t2) \in S \\ x \in \mathcal{C}_D((t1, t2]) \end{cases}$$

E plays the role of an evolution of the states in a given environment which is stated by I. It means that I defines some kind of invariant which can be simply $\mathbb{R}$.

Safety properties

Example

Safety properties
In the diagram, we illustrate the possible cases when one define a domain $\mathbb{R}$, a safety property $S = [0.5, 2.4]$ which is stating that three possible curves between $t1 = 0$ and $t2 = 3$ should be satisfying that $x(t) \in S$. Curves 1 ($safe(0, 3, x1, \mathbb{R}, S)$) and 3 ($safe(0, 3, x3, \mathbb{R}, S)$) are blue and are satisfying what is the safety property ; Curve 2 is not correct : $safe(0, 3, x2, \mathbb{R}, S)$ is not satisfied for $x2(2)$

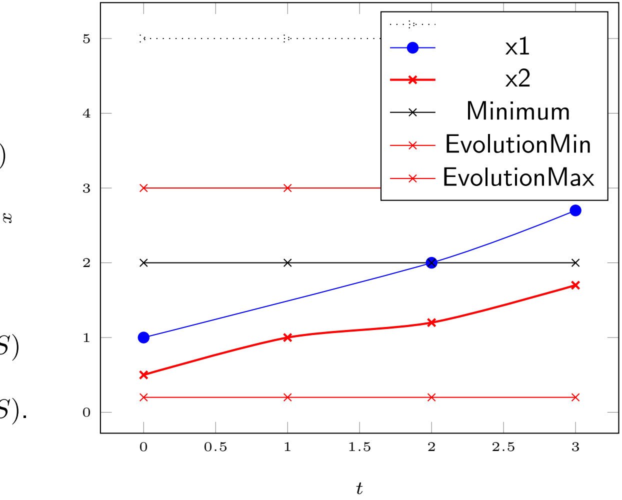


Stabilisation properties

Example

Stabilisation properties

We consider stabilisation properties $stable(t1, t2, x, \mathbb{R}, U, S)$ with the following definitions : $t1 = 0$, $t2 = 3$, $U =]-\infty, 2]$, $S =]2, +\infty[$. $x1$ is satisfying $stable(t1, t2, x1, \mathbb{R}, U, S)$ but not $stable(t1, t2, x2, \mathbb{R}, U, S)$.



According to the case study developed we have identified several *phases*, when the system is progressing :

- The system may be stable and the controller may *keep* or *control* the temperature between Tmin and Tmax ; the safety property $safe(t1, t2, \{\theta, now, \ldots\}, \mathbb{R}, Tmin..Tmax)$ and the project in[**?**] is defining the full process for the thermostat in mode nominal ; the first machine contains one event Update which is assigning to T_a a correct curve as stated by the safety property. It states that the variable T_a is representing what we have in mind when we want to get a correct observed system. The question is to prove that the function exists and the development will have as objective the progressive construction of a curve satisfying the safety property.
- The system may be entering an *unstable* state, because the user is setting *new* min and max ; the system may enter an *unstable* state and is supposed to recover from this state to reach a stable state satisfying the stability property called S in our assertion language ; another project called *tracker* is used and starts by a machine expressing the stabilisation property $stable(t1, t2, x, D, U, E, S)$ which should be possible according to the underlying constraints.

Example

Continuous actions as generalized substitutions

- $x :-(\lambda t.sin(t))$ is a continuous variable behaving as the sine function and is expressed as $x : |(x' = \lambda t.sin(t))$
- $z :-(v \in \mathbb{R}^+, \dot{z} = \lambda t.v \times t)$ is a continuous assignment meaning that z will behave as the x'= solution of the differential equation $(v \in \mathbb{R}^+, \dot{z} = \lambda t.v \times t)$. The variable z is not modified before the time $t < now$ and is behaving as a solution from now. The generalized substitution is leading to the following expression : $z : |(z' = y, v \in \mathbb{R}^+, \dot{y} = \lambda t.v \times t)$
- $\dot{u} :-u, u(now) = u_0$ is a continuous variable assignment meaning that u is a solution of the differential equation $\dot{u} = u$ with the constraint $u(now) = u_0$. In this case, the translation is then $u : |(u' = y, \dot{y} = y, y(now) = u_0)$.

Example

Updating the temperature

The variable x is a continuous variable defined as a function from $\mathbb{R}^+$ into $\mathbb{R}$ and is updated from now till ∞. The expression is stating that the variable x is not modified from 0 till now and from now the variable x is a solution of the differential equation $\dot{y} = -k{\cdot}y$ over the domain $[now, \infty[$. The following notation is the generalized substitution using continuous variable.

$$x : \left| \left(\begin{array}{l} x' = y \\ y \in now..\infty \longrightarrow \mathbb{R} \\ \dot{y} = -k{\cdot}y \\ y(now) = x(now) \\ \forall t < now . y(t) = x(t) \\ \forall t \geq now . y(t) = x(t) \end{array} \right) \right]$$

The variable x is behaving as the function y from now and y is an auxiliary notation which is used for the definition of the expression defining the extension of x. In fact, y is playing the role of the prime notation.

Continuous Generalized Substitution

A continuous generalized substitution over the set of continuous variables x is a expression defined by a relation between x and x' over constants c and sets s using classical set-theoretical operators, ϵ expressions, time-dependent functions defined explicitly and implicitly using differential equations.

For instance, we list examples of using this notation :

- $clock : |(clock' = \epsilon f.(\forall t \in 0..now \Rightarrow f(t) = clock(t)) \land (\forall t \in now..\infty \Rightarrow f(t) = t{-}now))$ which means that $clock$ is updated from the time *now* and is the function $\lambda t.f(t) = t{-}now))$.
- $u : |(u' = \epsilon y.(\dot{y} = y, y(now) = u_0))$.

The ϵ expressions should be proved to be feasible ; in the case of the differential equation, one has to add conditions that lead to the existence of solutions.

Definition

Hybrid Event An hybrid event e is defined by rhe notation

```
EVENT e
    ANY $t$
    WHERE
      $G(x,t)$
    THEN
      $x :|(P(x,x',t))$
    END
```

An hybrid event is defined as a classical Event-B event and the notation is extending the discrete events by allowing the use of hybrid actions. From the current syntax, we can define either purely discrete events or continuous events.

We have developed two Event-B basic models which are corresponding to *modes* when one wants to describe and control a real system :

- BM_$\diamond$(stable(t1,t2,x,D,U,E,S) is a diamond basic Event-B model and it models a mode corresponding to a value of x satisfying U at a time t1 or later and such that it exists a time later satisfying S but not later than t2.
- BM_$\square$(safe(t1,t2,x,D,S)) is a box basic Event-B model and it aims to maintain the value of x in S between t1 and t2.

We will define the notion of Event-B model in a more rigorous way later in the section. A first defintion would be that an Event-B model is an Event-B project *solving* a given problem. In the two examples, we have solved the problem of controlling the temperature between two bounds and the problem of stabilizing the temperature from a temperature out of the bounds to a temperature between the two bounds.

BM_◇(stable(t1,t2,x,D,U,E,S)

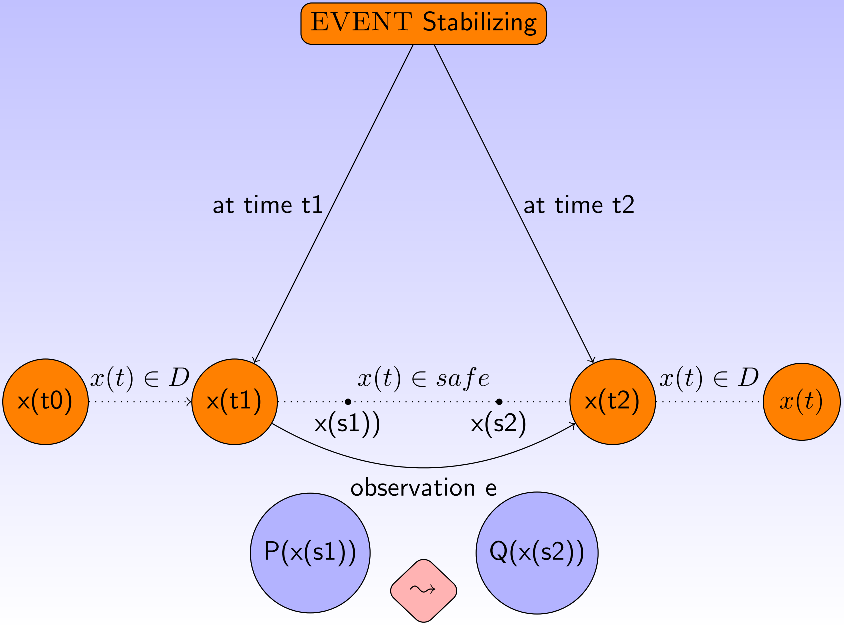
```
EVENT Stabilizing
ANY   $y, d, s$
WHERE
   $y \in R^+ \longrightarrow D$
   $d \in R^+$
   $s \in R^+$
   $t1 \le d$
   $s \le t2$
   $d \le s$
   $U(y_d)$
   $S(y_s)$
THEN
$act_1 : x := y$
END
```

The event Stabilizing is expressing that the event detects a value $y(d)$ in U and that it exists a time later s such that $y(s)$ at a time s before t2. Fig. 1 is describing the property P as the detection state and Q is the target state.

BM_□(safe(t1,t2,x,D,S))

```
EVENT Update
ANY   $y$,
WHERE
   $y \in R^+ \longrightarrow D$
   $\forall t \in t1..t2.y(t) \in S$
   $\forall t \in 0..t1.y(t) = x(t)$
THEN
$act_1 : x := y$
END
```

Previous values of x are not modified before t1 and the starting time is t1. Fig. 2 is describing the property *safe* satisfied bbetween t1 and t2.



EVENT Stabilizing
at time t1
at time t2
x(t0)
$x(t) \in D$
x(t1)
$x(t) \in safe$
x(s1))
x(s2)
x(t2)
$x(t) \in D$
$x(t)$
observation e
P(x(s1))
$\rightsquigarrow$
Q(x(s2))

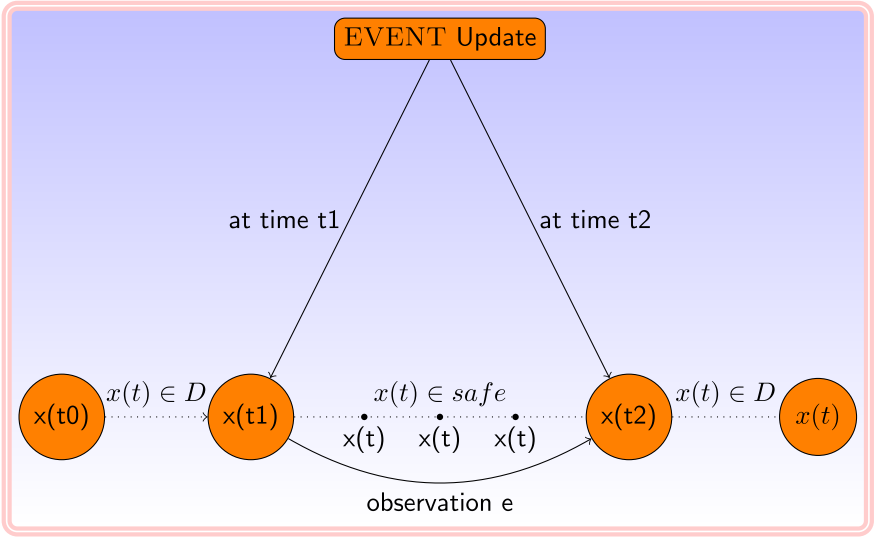


FIGURE – Schema for the box model

Conclusion and Next Lectures

- Extending the scope of the generalized substitution by differential equations.
- Defining specific patterns
- Illustrating the full chain for developing hybrid systems.