Chapter 8

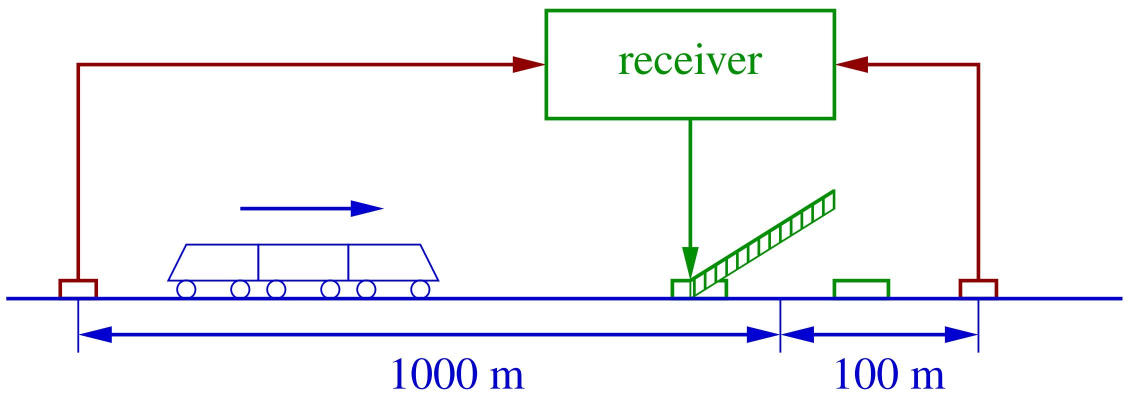
Complex timed systems

Introduction

In order to analyze the properties of a complex system, it is not always sufficient to study the individual behavior of its components.

Example: An embedded system controlling a railroad crossing is composed of the following elements:

- Two sensors located on the tracks 1000 meters before and 100 meters after the crossing, aimed at detecting (respectively) that a train approaches or has passed the crossing.
- A receiver that processes the signals emitted by the sensors, and sends orders to open or close the gate.



The following information is known:

- The speed of the approaching trains is between 48 and 52 m/s. Then, after reaching the first sensor, their speed is reduced to a value between 40 and 52 m/s.
- After it receives a signal from a sensor, the receiver waits for at most 5 seconds before sending an order to close or to open the gate. During this delay, the receiver ignores incoming signals.
- The gate is closed (resp. open) when its angle is equal to 0 (resp. 90) deg. The gate is able to move at the rate of 20 deg/s.
- Two successive trains are always separated by at least 1600 m.

Question: Is the gate always closed when a train passes the crossing?

Modeling a system

In order to analyze the properties of a system, the first step consists in building a model, i.e., an abstract representation of the system that describes its relevant properties without any ambiguity.

For embedded applications, the modeling formalism must be able to express

- operations on integer variables (used as counters, sequence numbers, identifiers, . . .), as well as on real variables (for representing positions, speeds, delays, . . .).
- discrete state transitions (e.g., incrementing a counter) as well as continuous evolution laws (e.g., constant-speed movement).
- composition of elementary systems into a more complex entity.

Hybrid systems

Hybrid systems are a modeling formalism that meets those requirements.

Syntax:

A hybrid system is composed of:

- a finite number p of processes $P_1, P_2, \ldots, P_p$,
- a finite number n of variables $x_1, x_2, \ldots, x_n$, grouped together into a vector $\vec{x} \in \mathbb{R}^n$,
- a finite set L of synchronization labels.

Each process P_i is represented by a graph (V_i, E_i), where

- V_i est a finite set of control locations,
- $E_i \subseteq V_i \times V_i$ is a finite set of transitions.

Each control location $v \in V_i$ is associated with:

- An activity *dif*(v), expressed as a conjunction of linear constraints over the variables $x_1, x_2, \ldots, x_n$ and their first temporal derivative $\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n$.
- An invariant *inv*(v), expressed as a conjunction of linear constraints over the variables $x_1, x_2, \ldots, x_n$.

Each transition $e \in E_i$ is associated with:

- A guard *guard*(e), that represents a condition that must be satisfied in order to enable this transition.
- An action *act*(e), composed of constraints that specify how the values of the variables are modified when this transition is followed.

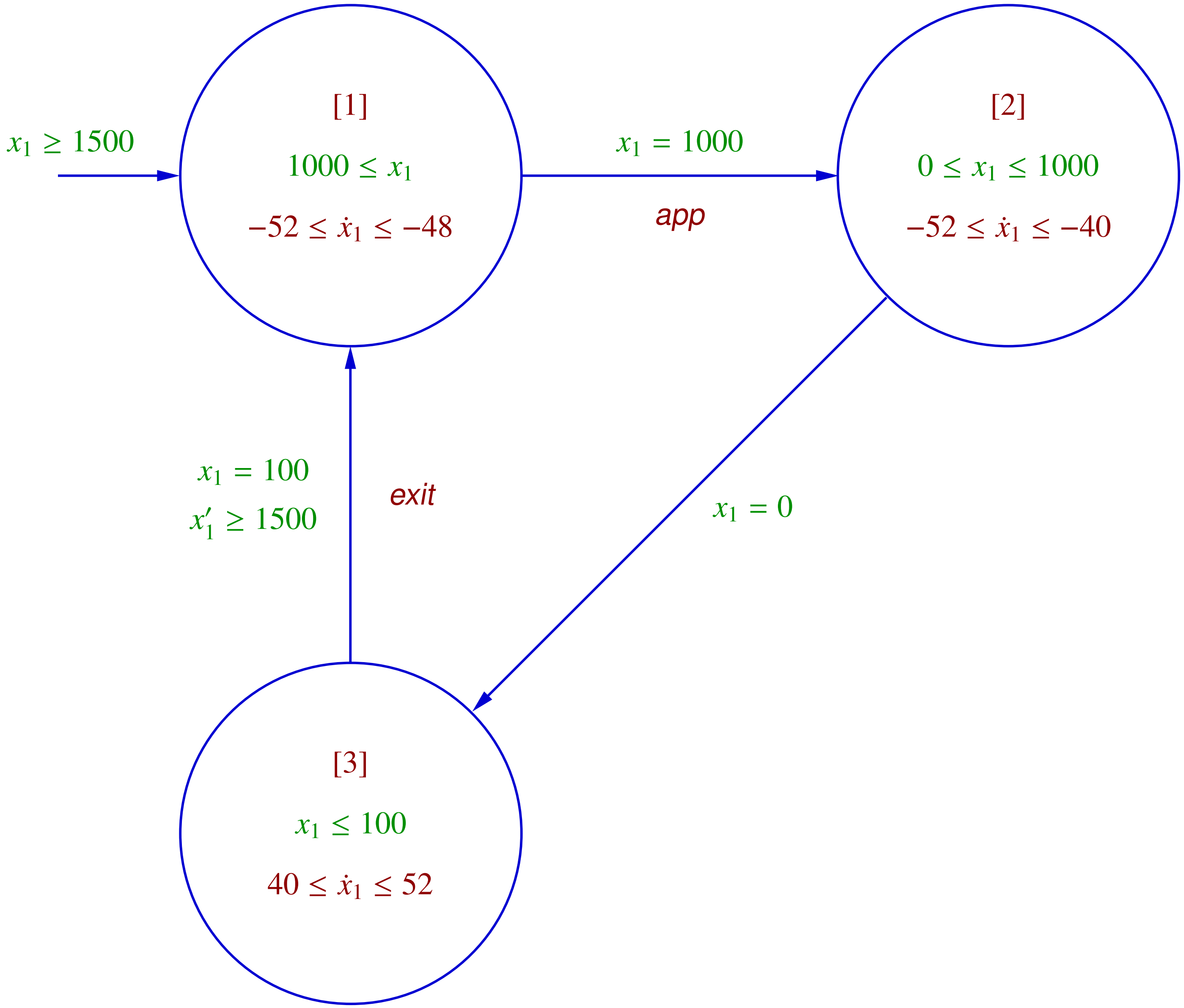
  In practice, the guard and the action can be combined into a conjunction of constraints over the values of the variables before $(x_1, x_2, x_3, \ldots)$ and after $(x'_1, x'_2, x'_3, \ldots)$ following the transition.

- An optional label *sync*$(e) \in L$ that makes it possible to synchronize this transition with one or many transitions belonging to other processes.

Finally, one defines an initial control location for each process, and assigns a set of possible initial values for each variable, specified as a conjunction of linear constraints.

Example: Process modeling the behavior of a train and the two sensors.

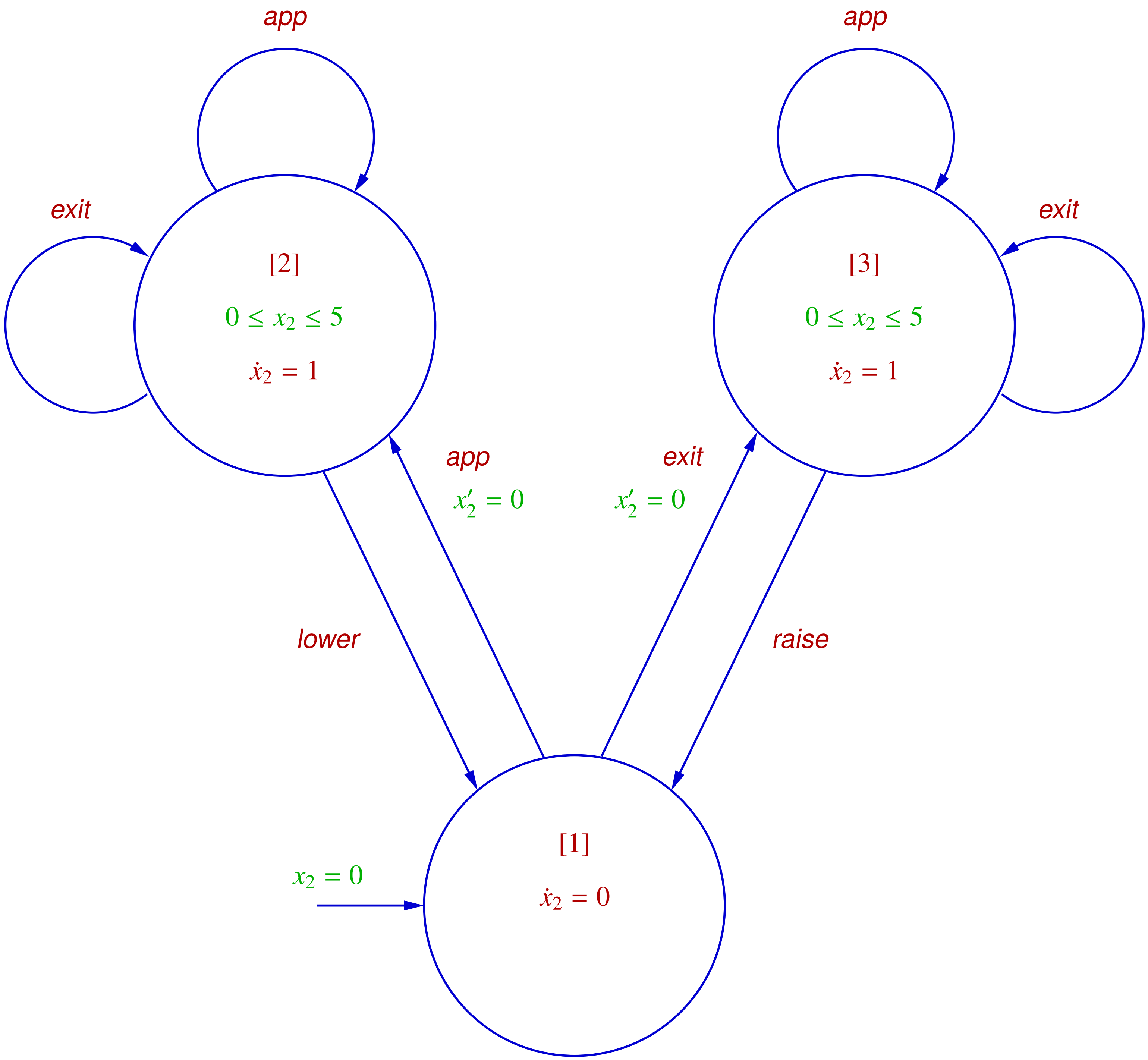
- The distance between the train and the crossing is represented by a variable x_1.
- The signals emitted by the sensors are modeled by two synchronization labels *app* and *exit*.



[1]
1000 ≤ x1
−52 ≤ ẋ1 ≤ −48
x1 ≥ 1500
x1 = 1000
app
[2]
0 ≤ x1 ≤ 1000
−52 ≤ ẋ1 ≤ −40
x1 = 0
x1 = 100
x'1 ≥ 1500
exit
[3]
x1 ≤ 100
40 ≤ ẋ1 ≤ 52

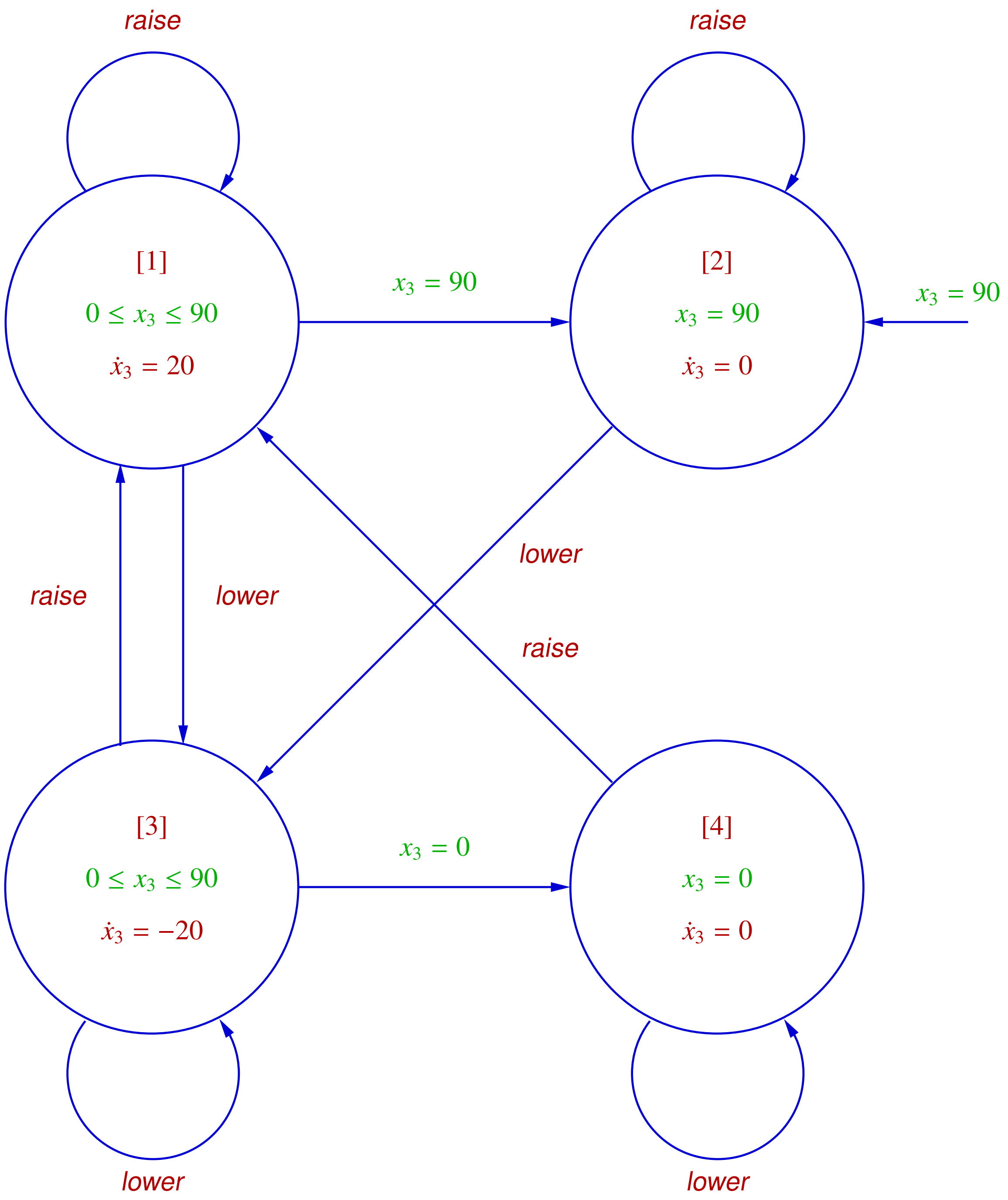
Process modeling the receiver:

- The delay between receiving a sensor signal and sending an order to the gate is represented by a variable x_2.
- The labels *raise* and *lower* model the orders sent to the gate.


app
app
exit
exit
[2]
$0 \le x_2 \le 5$
$\dot{x}_2 = 1$
[3]
$0 \le x_2 \le 5$
$\dot{x}_2 = 1$
app
$x_2' = 0$
exit
$x_2' = 0$
lower
raise
$x_2 = 0$
[1]
$\dot{x}_2 = 0$

Process modeling the gate:

- The variable x_3 represents the angular position of the gate.
- The labels *raise* and *lower* correspond to the orders received.


raise
raise
[1]
$0 \le x_3 \le 90$
$\dot{x}_3 = 20$
$x_3 = 90$
[2]
$x_3 = 90$
$\dot{x}_3 = 0$
$x_3 = 90$
lower
raise
lower
raise
[3]
$0 \le x_3 \le 90$
$\dot{x}_3 = -20$
$x_3 = 0$
[4]
$x_3 = 0$
$\dot{x}_3 = 0$
lower
lower

Semantics:

At any given time, the current state of a hybrid system is characterized by

- a control location for each process, and
- a value for each variable.

The state of a system can evolve in two ways:

- By letting time elapse (*time steps*). The control locations of processes stay unchanged, and the values of the variables evolve according to the invariants and activities associated to these locations.
- By following transitions (*transition steps*). One can either
  - follow a single unlabeled transition, or
  - follow a pair of transitions (more generally, a maximal set of at least two transitions) belonging to different processes and sharing the same synchronization label.

In both cases, a transition can only be followed provided that its guard is satisfied by the current variable values.

When a transition is followed, the variable values are modified according to the action associated to the transition. The invariant of the destination location must be satisfied by the new variable values (otherwise, the transition cannot be followed).

A state s_2 is reachable from a state s_1 if there exists a finite sequence of time steps and transition steps that lead from s_1 to s_2.

A state s is reachable if it is reachable from an initial state.

Example: The state $([2],[2],[2],800,4,90)$ of the railroad crossing controller model corresponds to

- the control location $[2]$ for each process.
- the respective values 800, 4 and 90 for the variables x_1, x_2 and x_3.

This state is reachable. Indeed, one has

$$
\begin{aligned}
& ([1],[1],[2],1500,0,90) \\
\stackrel{10}{\Longrightarrow} \quad & ([1],[1],[2],1000,0,90) \\
\stackrel{\mathit{app}}{\longrightarrow} \quad & ([2],[2],[2],1000,0,90) \\
\stackrel{4}{\Longrightarrow} \quad & ([2],[2],[2],800,4,90),
\end{aligned}
$$

where

- "$\stackrel{\lambda}{\Longrightarrow}$" denotes a time step with a delay equal to λ,
- "$\stackrel{\ell}{\longrightarrow}$" corresponds to following a pair of transitions sharing the synchronization label ℓ.

Executions of a hybrid system

An execution of a hybrid system is an infinite sequence $s_1, s_2, s_3, \ldots$ of states such that:

- s_1 is an initial state of the system.
- For each i, the state s_{i+1} is reachable from the state s_i in a time $\delta_i \geq 0$.

Note: A hybrid system generally admits several different executions (*non-determinism*). Indeed,

- The time spent at a control location may not be precisely constrained by the invariant.
- A control location can have several outgoing transitions enabled at a given time.

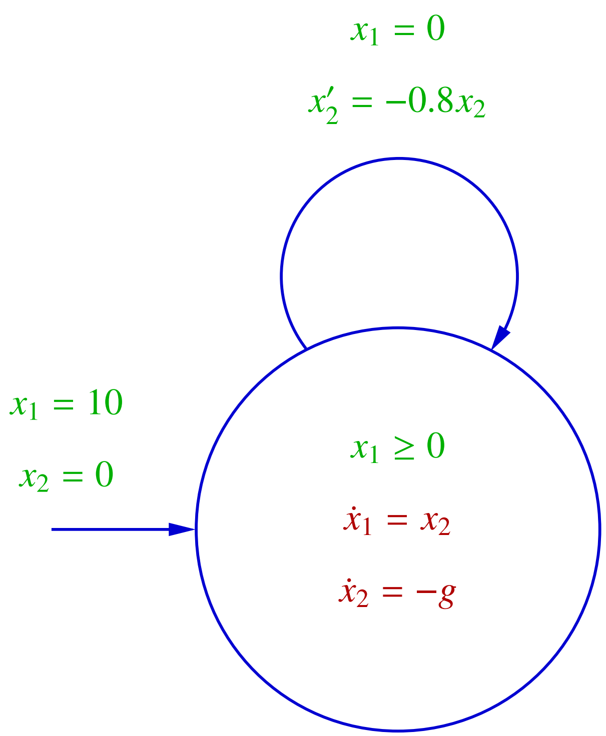
An execution $s_1, s_2, s_3, \ldots$ beginning at time $t = 0$ is said to be divergent if for every $T > 0$, there exists i such that the state s_i is reached later than time $t = T$.

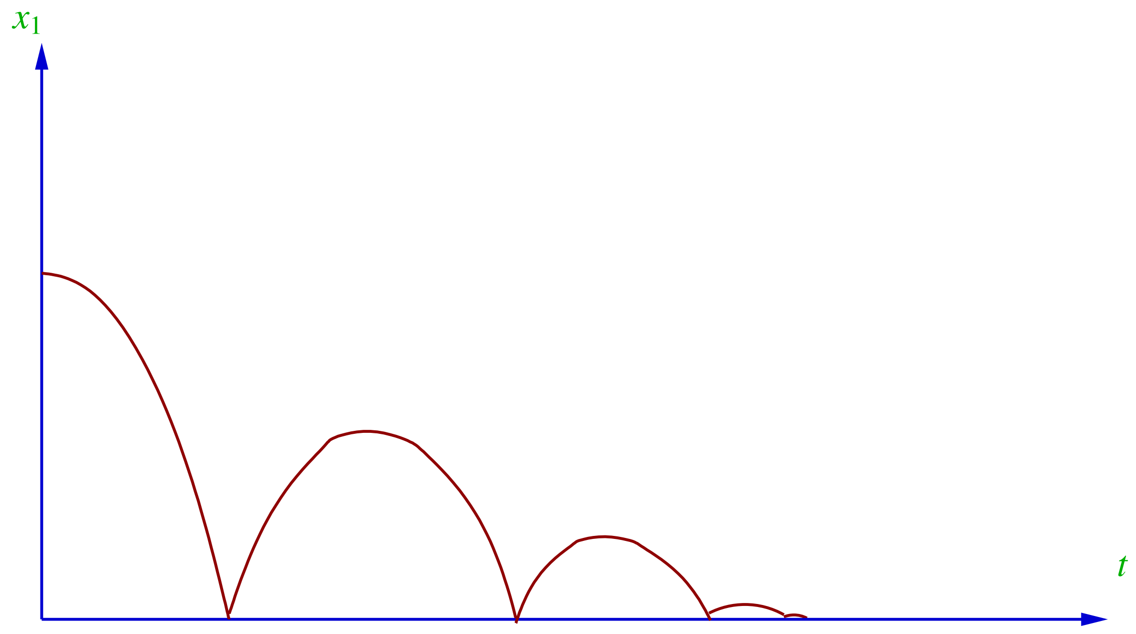
Zeno hybrid systems

A hybrid system is said to have the Zeno property if it admits an execution in which at least one finite prefix is not a prefix of a divergent execution.

In other words, in a Zeno hybrid system, there exists a reachable state from which no execution is able to get past some time bound.

Example: Hybrid system modeling a bouncing ball.





Remarks:

- Such models are inconsistent with physical reality and must be avoided!
- For some restricted classes of hybrid systems, automatic methods have been developed for transforming any given model into another one that does not have the Zeno property, and admits the same divergent executions.

State-space exploration

A large number of interesting properties of a hybrid system can be checked by computing its reachable states.

This computation can be carried out by building, from every initial state, a tree in which each node q represents a reachable state $s(q)$, and the children of q correspond to the states that are reachable from $s(q)$ by

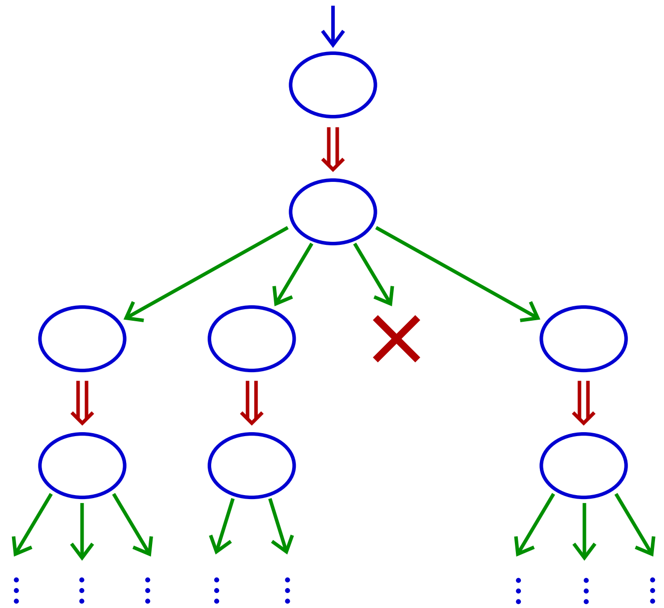
- a time step, or
- a transition step.

Problems:

- The system may have infinitely many initial states.
- The time spent at a control location may take an infinite number of possible values, which leads to trees of infinite degree.
- Since executions are infinite, the trees also have an infinite depth.

Solutions:

- Sets of states sharing the same control locations and differing only in the elapsed time in those locations can be grouped into regions. A tree can be built in which the nodes are associated with regions instead of individual states.
- At each exploration step, a first operation saturates the current region by letting time elapse during all possible delays. Then, the enabled transitions are individually followed, creating new branches.
- The branches of the exploration tree that only contain already visited states can be pruned.



Notes:

- Several exploration strategies are possible: depth-first search (DFS), breadth-first search (BFS), ...
- For general hybrid systems, the region tree can still be infinite. It is however possible to define restricted classes of models, for which a finite region tree can always be computed.

  Example: Timed automata are hybrid systems in which

  - the activities are of the form $\dot{x}_i = 1$,
  - all invariants, guards and actions are conjunctions of constraints of the form $x_i \# c$ or $x_i - x_j \# c$, where c is an integer number, and $\# \in \{<, \leq, =, \geq, >\}$.

- Some tools are available for exploring automatically the state space of hybrid systems (e.g., *HyTech*, *SpaceEx*, *Hylaa*) or timed automata (e.g., *Uppaal*).

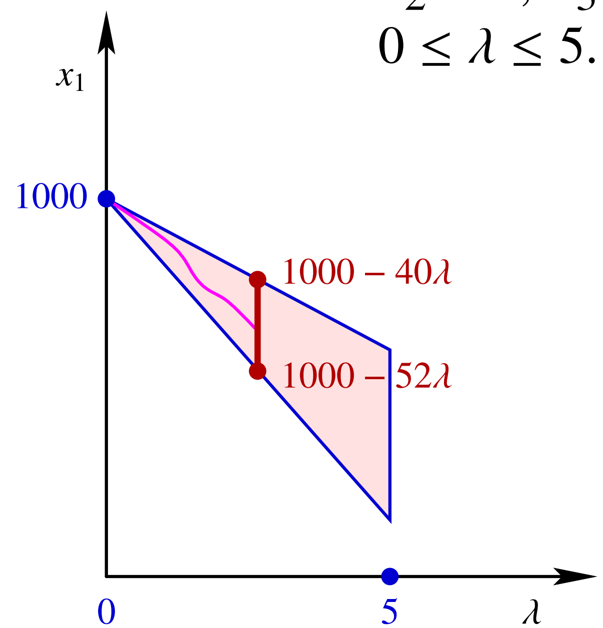
  Notes: These tools

  - represent and handle regions with the help of dedicated data structures, based on logic formulas, convex polyhedra, difference matrices, ...
  - are able to check properties that go beyond simple reachability.

Example: Railroad crossing

$([1],[1],[2]) : x_1 \geq 1500,\ x_2 = 0,\ x_3 = 90.$

$$\begin{aligned}
\Longrightarrow \quad ([1],[1],[2]) \quad &: \quad x_1 \geq 1000, \\
&\quad\ \ x_2 = 0,\ x_3 = 90. \\
\xrightarrow{app} \quad ([2],[2],[2]) \quad &: \quad x_1 = 1000, \\
&\quad\ \ x_2 = 0,\ x_3 = 90. \\
\overset{\leq 5}{\Longrightarrow} \quad ([2],[2],[2]) \quad &: \quad x_1 \geq 1000 - 52\lambda, \\
&\quad\ \ x_1 \leq 1000 - 40\lambda, \\
&\quad\ \ x_2 = \lambda,\ x_3 = 90, \\
&\quad\ \ 0 \leq \lambda \leq 5.
\end{aligned}$$



$$\overset{\textsf{lower}}{\longrightarrow} \quad ([2],[1],[3]) \; : \; \begin{aligned} & x_1 \geq 1000 - 52\lambda, \\ & x_1 \leq 1000 - 40\lambda, \\ & x_2 = \lambda,\ x_3 = 90, \\ & 0 \leq \lambda \leq 5. \end{aligned}$$

$$\overset{\leq 9/2}{\Longrightarrow} \quad ([2],[1],[3]) \; : \; \begin{aligned} & x_1 \geq 1000 - 52(\lambda + \mu), \\ & x_1 \leq 1000 - 40(\lambda + \mu), \\ & x_2 = \lambda,\ x_3 = 90 - 20\mu, \\ & 0 \leq \lambda \leq 5,\ 0 \leq \mu \leq 9/2. \end{aligned}$$

$$\overset{x_3=0}{\longrightarrow} \quad ([2],[1],[4]) \; : \; \begin{aligned} & x_1 \geq 766 - 52\lambda, \\ & x_1 \leq 820 - 40\lambda, \\ & x_2 = \lambda,\ x_3 = 0, \\ & 0 \leq \lambda \leq 5. \end{aligned}$$

$$\Longrightarrow \quad ([2],[1],[4]) \; : \; \begin{aligned} & 0 \leq x_1 \leq 820 - 40\lambda, \\ & x_2 = \lambda,\ x_3 = 0, \\ & 0 \leq \lambda \leq 5. \end{aligned}$$

$$\overset{x_1=0}{\longrightarrow} \quad ([3],[1],[4]) \; : \; \begin{aligned} & x_1 = 0,\ x_2 = \lambda, \\ & x_3 = 0,\ 0 \leq \lambda \leq 5. \end{aligned}$$

$$\overset{\leq 5/2}{\Longrightarrow} \quad ([3],[1],[4]) \;:\; \begin{array}{l} 0 \leq x_1 \leq 100, \\ x_2 = \lambda,\ x_3 = 0, \\ 0 \leq \lambda \leq 5. \end{array}$$

$$\overset{\textit{exit}}{\longrightarrow} \quad ([1],[3],[4]) \;:\; \begin{array}{l} x_1 \geq 1500,\ x_2 = 0, \\ x_3 = 0. \end{array}$$

$$\overset{\leq 5}{\Longrightarrow} \quad ([1],[3],[4]) \;:\; \begin{array}{l} x_1 \geq 1500 - 52\lambda, \\ x_2 = \lambda,\ x_3 = 0, \\ 0 \leq \lambda \leq 5. \end{array}$$

$$\overset{\textit{raise}}{\longrightarrow} \quad ([1],[1],[1]) \;:\; \begin{array}{l} x_1 \geq 1500 - 52\lambda, \\ x_2 = \lambda,\ x_3 = 0, \\ 0 \leq \lambda \leq 5. \end{array}$$

$$\overset{\leq 9/2}{\Longrightarrow} \quad ([1],[1],[1]) \;:\; \begin{array}{l} x_1 \geq 1500 - 52(\lambda + \mu), \\ x_2 = \lambda,\ x_3 = 20\mu, \\ 0 \leq \lambda \leq 5,\ 0 \leq \mu \leq 9/2. \end{array}$$

$$\overset{x_3 = 90}{\longrightarrow} \quad ([1],[1],[2]) \;:\; \begin{array}{l} x_1 \geq 1266 - 52\lambda, \\ x_2 = \lambda,\ x_3 = 90, \\ 0 \leq \lambda \leq 5. \end{array}$$

$$\begin{array}{lll} \Longrightarrow & ([1],[1],[2]) & : \quad x_1 \geq 1000, \\ & & \quad x_2 = \lambda,\ x_3 = 90, \\ & & \quad 0 \leq \lambda \leq 5. \\ \overset{app}{\longrightarrow} & ([2],[2],[2]) & : \quad x_1 = 1000, \\ & & \quad x_2 = 0,\ x_3 = 90 \\ & & \quad \text{(already obtained).} \end{array}$$

Notes:

- In this example, the regions correspond to the sets of states obtained after each time-step operation (denoted by "$\Longrightarrow$").
- Checking whether the gate is always closed when a train reaches the crossing amounts to verifying that in each reachable region, $x_1 = 0$ implies $x_3 = 0$.
- This particular system shows a very deterministic behavior: In each reachable state, there is at most one transition (or a pair of synchronized transitions) that is enabled.

  (This is generally not the case!)