The science of bio-remulation for Executable Organ on Chip

Partha Roop, Avinash Malik, Sidharta Andalam

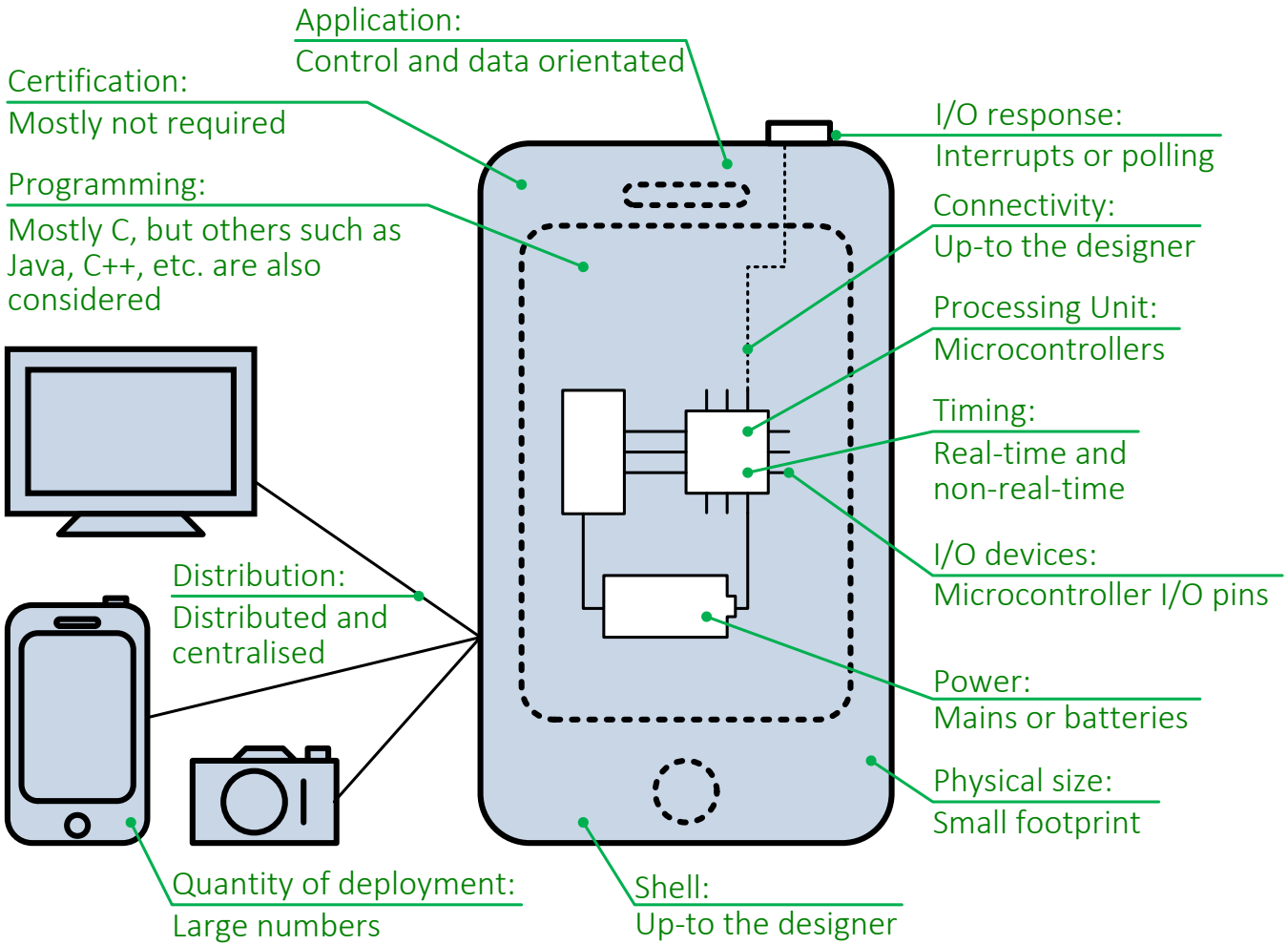
THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

BioRemulation™ Research Group
The University of Auckland

December 2016

www.pretzel.ece.auckland.ac.nz/bio

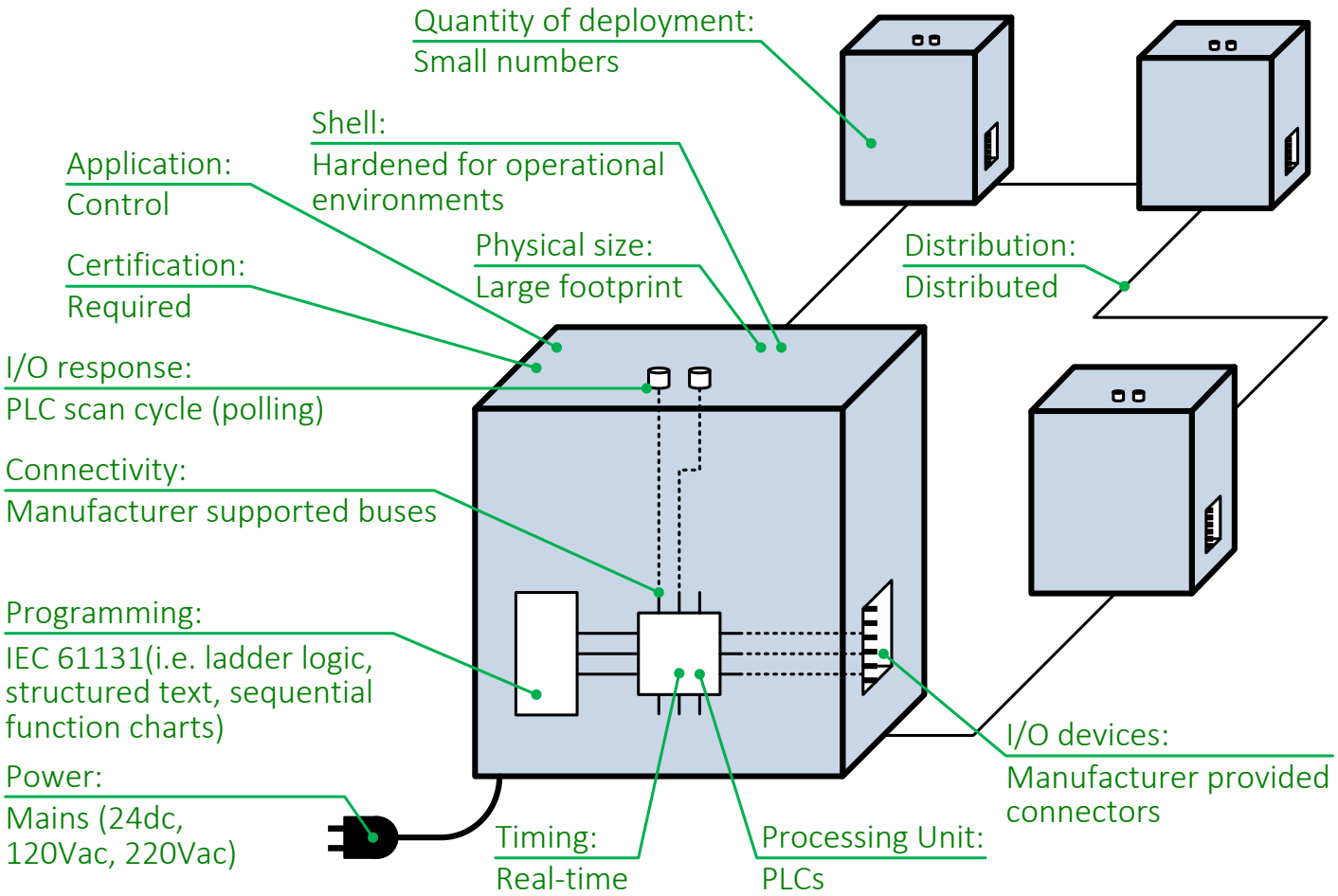
Embedded Systems



(b) Embedded systems [a]

[a] Yoong et al. Model driven design using IEC61499, Springer 2015.

Automation Systems



(a) Automation system

[a]

[a] Yoong et al. Model driven design using IEC61499, Springer 2015.

1

- Embedded systems and automation systems are *reactive* rather than *transformational*.
- Reactive systems continuously interact with their environment at a rate determined by the environment.
- Examples: lift controller, traffic light controller, digital camera, MP3 player and so on.
- Real-time systems are a subclass of reactive systems.

[1] I have based some slides on this topic from:Kopetz, Hermann. "Real-time systems: design principles for distributed embedded applications", Springer, 2011.

Real-Time Systems Versus Computer Systems

A *real- time computer system* is a system in which the correctness of system behavior depends not only on the logical results of the computations, but also on the physical instants when these results are produced.

Real–Time Systems	Conventional Computing Systems
Real– World (mobile phones, DVD players, Airplanes)	Computer World (PCs, Workstations, Mainframes)
Provides worst case timing gurantees	Provides avrage case timing guarantess
Environment is asynchronous and can't be controlled!	Computer controls the speed of operation.
Reaction if too slow will lead to safety issues.	Reaction if too slow annoyes the user!
Real–Time	Computer Time

Why real-time and not just fast?

Fast enough depends on the system and its environment [Ramamritham 2002]. Hence it is purely relative.

1. A fly is fast enough to fly away.
2. A mouse is fast enough to steal cheese.

What if the environment changes?

1. A mouse trap houses the cheese.
2. A Fly is hit with a swatter before it can fly!

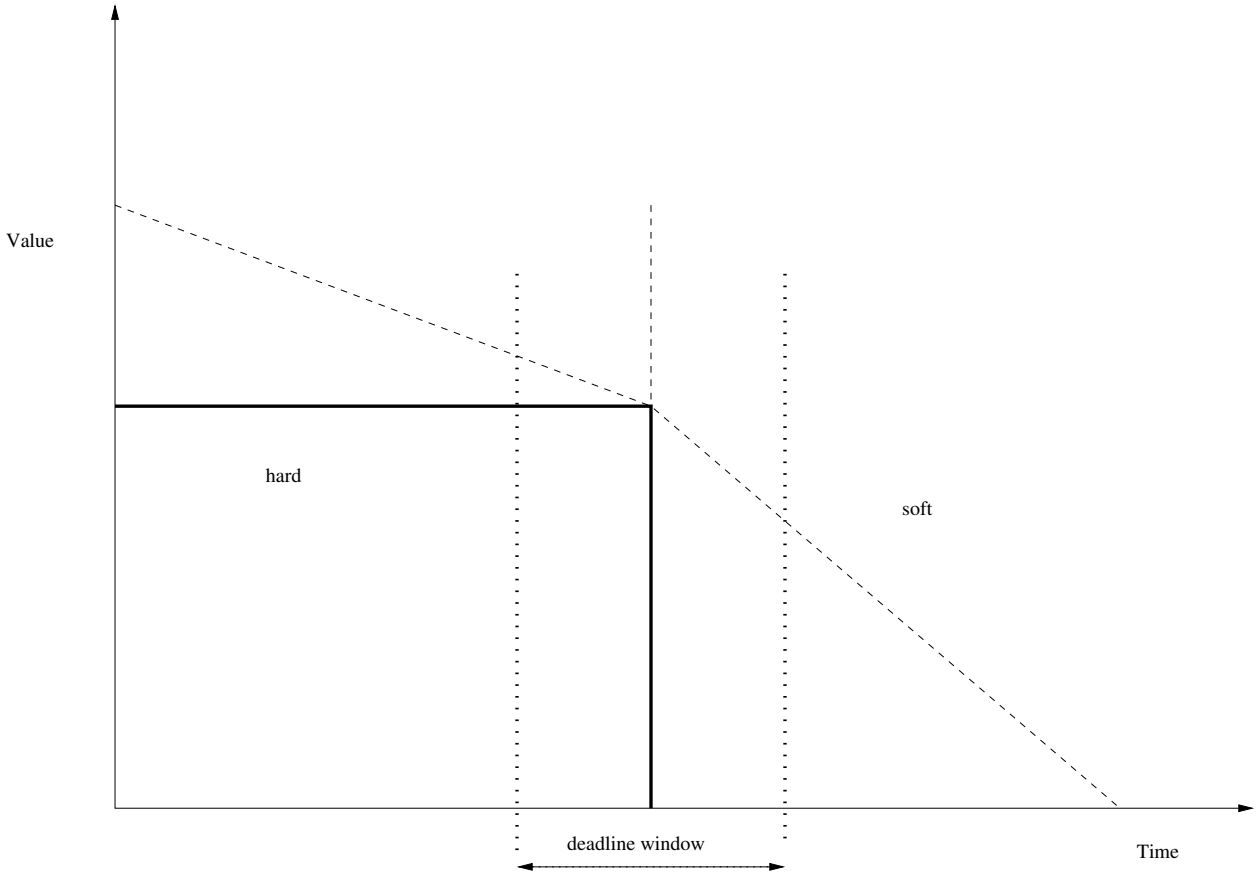
Hence, time scale is dependent on the environment (which can not be altered).

Correctness of results depends on the value produced and the time instant when this is delivered.

1. If a lift arrives at a specified floor after one day, it is no good!
2. In a digital camera, if conversion of the CCD image to JPEG takes more than a few seconds, user will not be interested in buying this camera!

Hence, in a RT System, reactions to events as they occur at their speed is critical (system and environment time scales are same).

Classification



- Hard Real-Time: result is useless or dangerous if deadline exceeded. Examples: Flight control, military, nuclear power plants, robotics, traffic light.
- Soft Real-Time: result of some lower value if deadline exceeded. Examples: multimedia, interactive games, operating systems, digital camera.
- Firm Real-Time: If the value drops to zero at deadline.

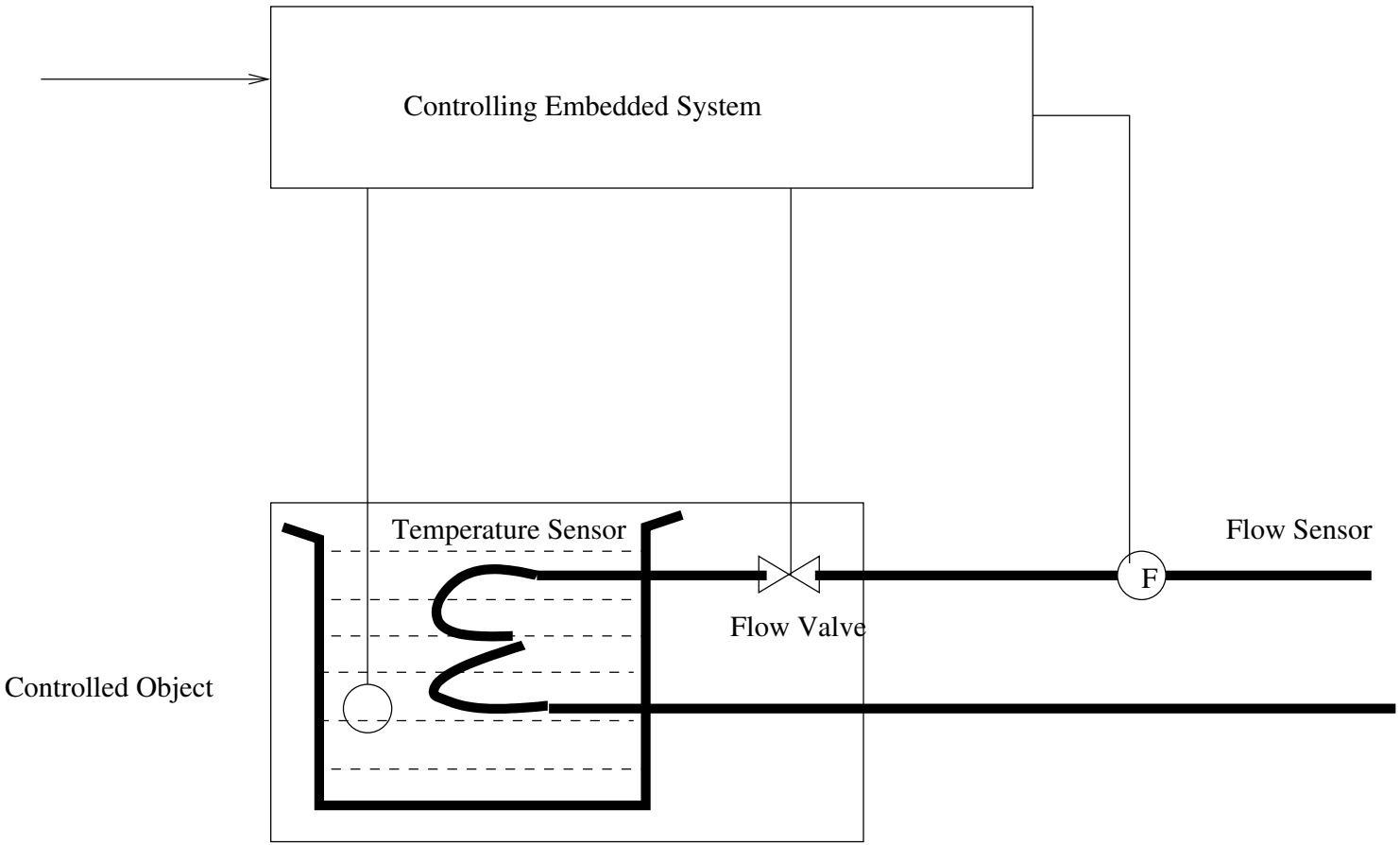
Deadline window specifies that the result must be computed within a window (lower bound, upper bound).

Deadline: An instant at which a result may be produced.

- Hard deadline: if a catastrophe could happen if a firm deadline is missed.
- Soft deadline: if the result has utility even after the deadline is missed.

An embedded system that must meet at least one hard deadline is called a hard real-time embedded system or a safety-critical embedded system [Kopetz'97].

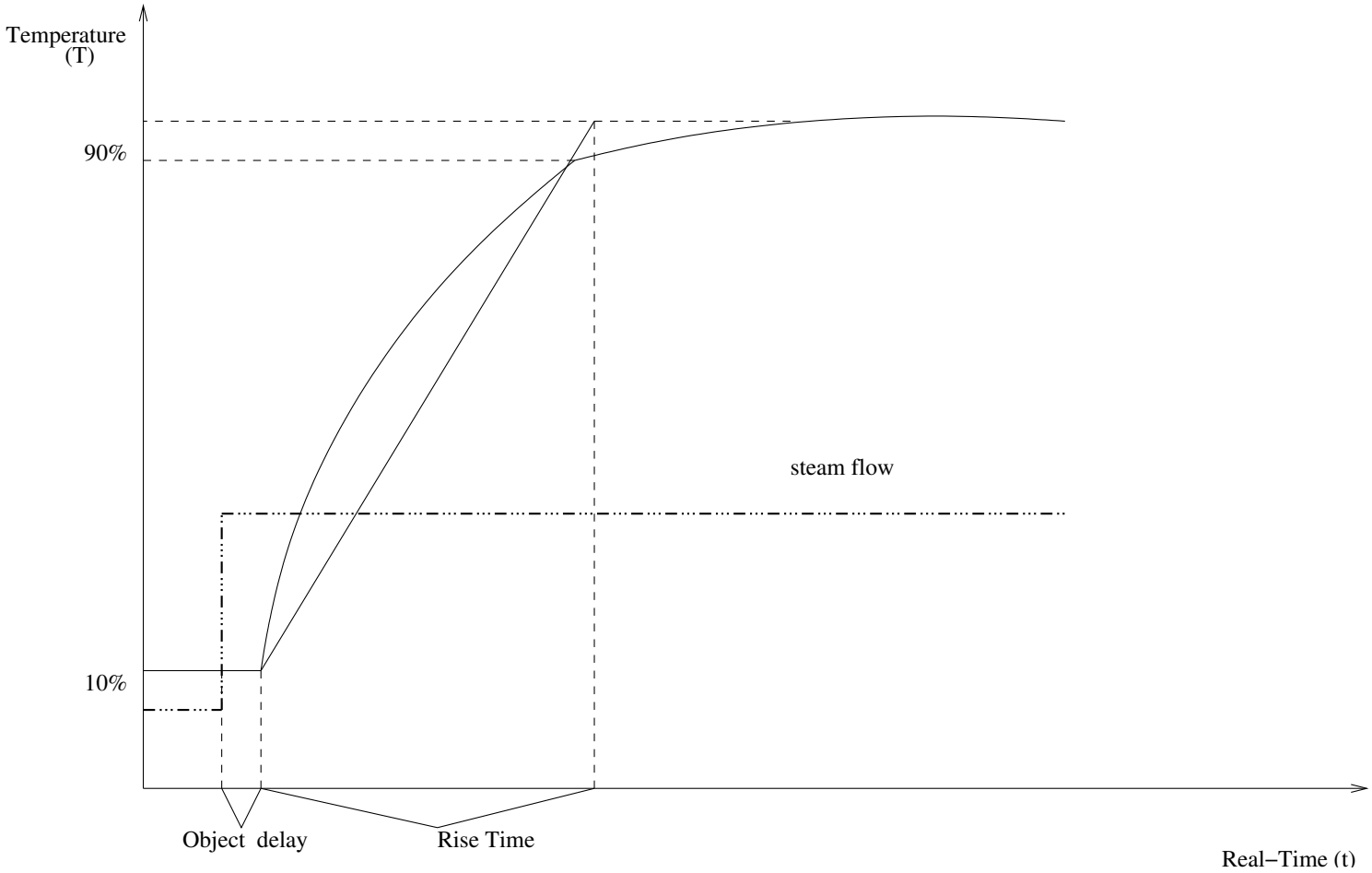
A simple control loop



[4]

Goal: To control the flow of steam so that the temperature of the liquid remains within the range of the set point set by the human operator.

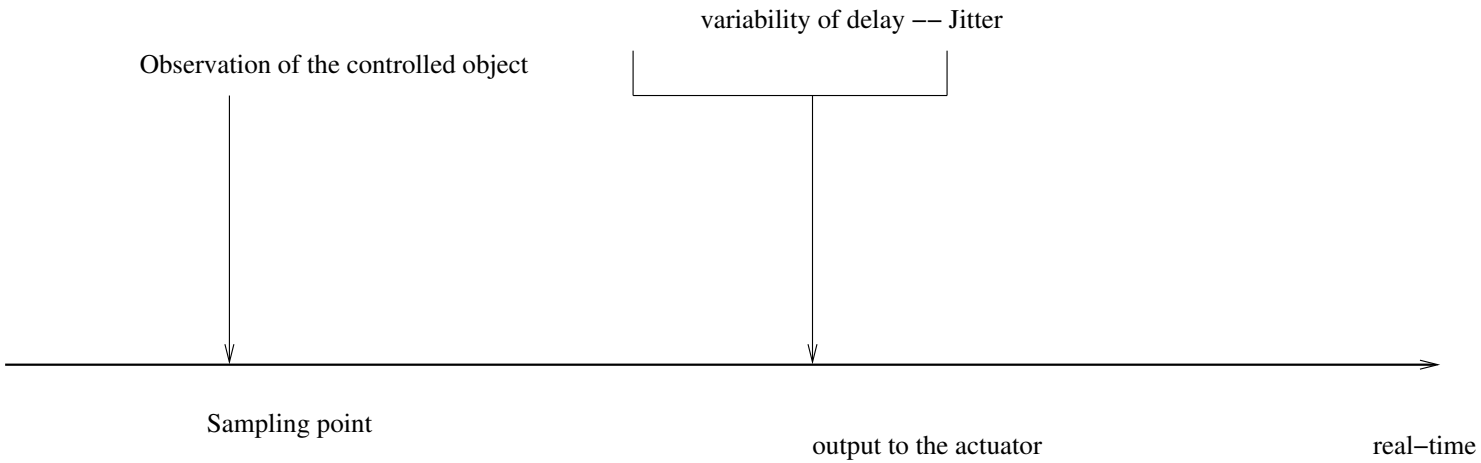
[4] **Latency**: It is the delay associated with initial startup of a process. **Throughput**: Number of tasks completely processed by a system per unit time.



Types of Delay..

- d^{object}: Delay due to inertia of the process, called process latency.
- d^{rise}: Difference between the time when the controlled variable (say temperature) has reached a steady state and the time when the control variable actually begins to rise from the previous steady state.
- d^{sample}: This is the constant duration between two sampling points. Sampling frequency is the reciprocal of this sampling period.
- $d^{computer}$: A given time interval after the sampling point, the controlling computer will output a new value of control variable. This is known as computer delay.

Delay Types..



- $\delta d^{computer}$: The difference between minimum and maximum values of delay is called *jitter* e.g, the variability of delay.
- $d^{deadtime}$: the time interval between the observation of a real-time entity and the start of a reaction of the controlled object due to computer action based on this observation. $d^{object} + d^{computer}$

Event Triggered versus Time Triggered

A *trigger* is an event that causes the start of some action such as execution of a task or transmission of some message [Kopetz'97].
Event Triggered: All communication and processing activities are initiated whenever a significant change of state i.e, an event other than the regular clock tick event is noted.
Time Triggered: All communication and processing activities are initiated at predetermined points in time.

Real-Time Operating System (OS)

- A real-time OS provides an abstraction for the underlying hardware and also provides guarantees on worst case performance for tasks with deadlines.
- Since embedded systems often have *tasks* with associated hard or sort timing constraints, a real-time OS is needed to *schedule* such tasks.
- Tasks are basic executable entities that are scheduled.

Cyclic Executives

- For simple embedded systems, a real-time operating system may be a over kill.
- They often employ a *cyclic executive* that communicates with the sensors and actuators at a single processing rate.
- A cyclic executive for the temperature controller. This loop is repeated once every period.

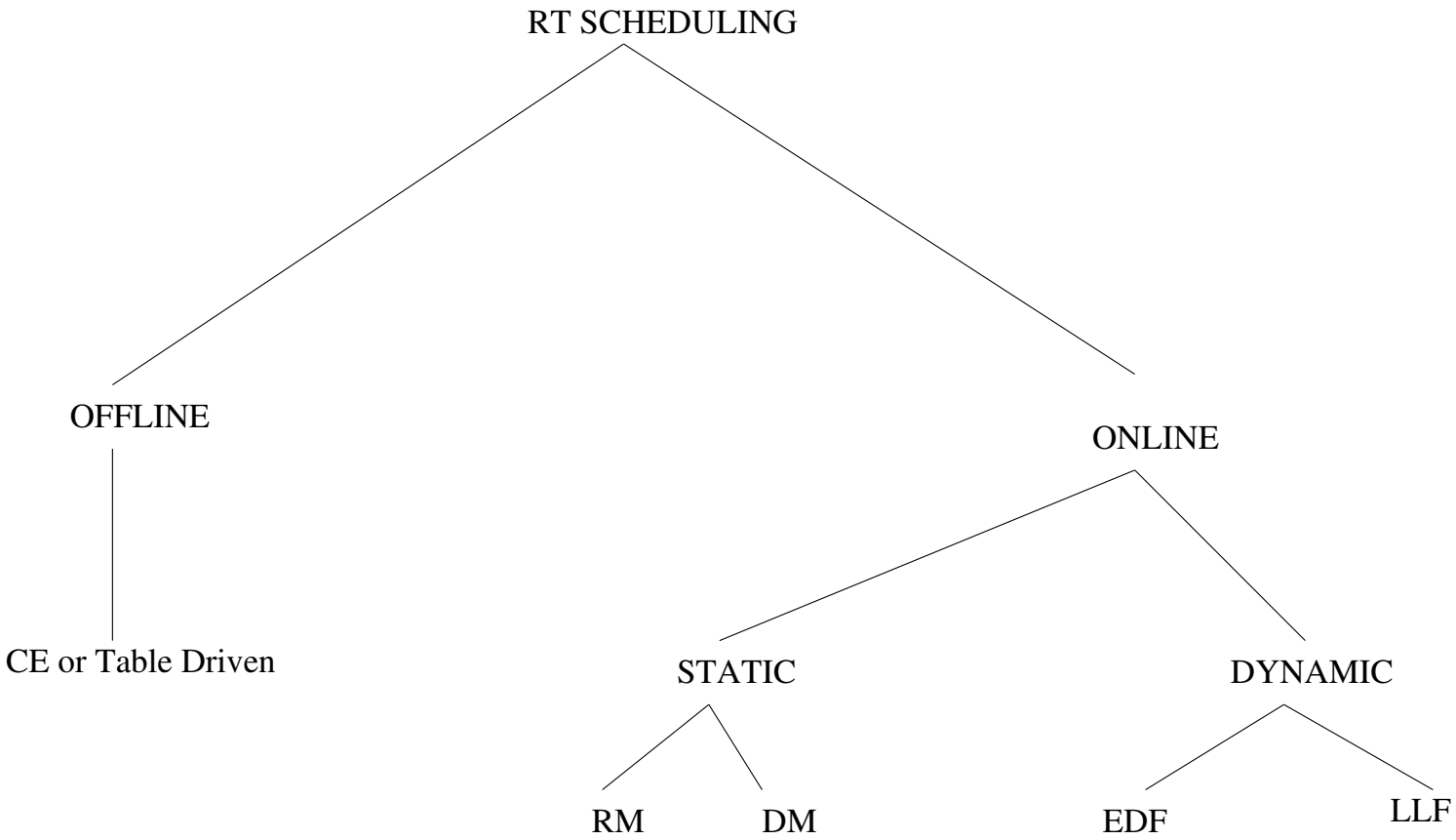
```
while(true){
    read temperature sensor;
    if (temperature > max)
      stop flowValve;
    else if (temperate < min)
      increase gasFlow;

}
```

Limitations:

1. Inputs will wait until the beginning of next period. Interrupts may be employed to avoid this.
2. If different tasks are there in the process with priorities, then only one set of static priorities can be dealt with. Also, with this model, once a task is started, it is not stopped, even though a higher priority task may be waiting.
3. Functionality and timing are mixed up and not separated.

- Preemptive multitasking needed when data arrives at sporadic instants and demands immediate attention.
- The system is described as a collection of independently running threads called processes or tasks.
- RTOS executes tasks based on a scheduling policy.

Classification



Classification..

1. Offline or Table Driven Scheduling: this approach requires the entire task set (must be known a priori) and performs schedulability analysis offline. If a schedule is possible, the start time of each task is identified and stored in a table. This is applicable to a set of periodic tasks.
2. Online Scheduling: this approach takes scheduling decisions online whenever a task finishes or a new task enters the system. These algorithms deal with tasks having *static priorities* or *dynamic priorities*.

Real-Time tasks are basic executable entities that are scheduled. Primary task parameters are:

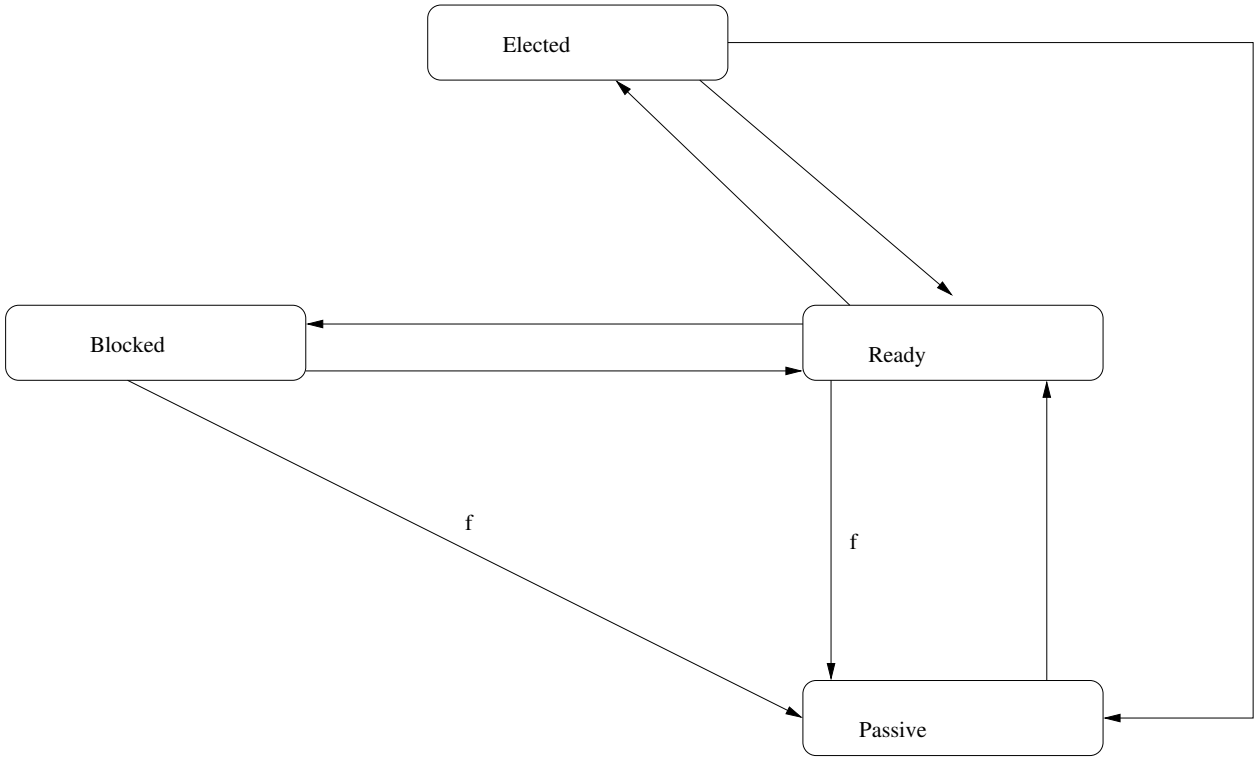
1. **r**: the task *release time* i.e., the triggering time of the task execution request.
2. **c**: the worst case *computation time*, when the processor is fully allocated to the task.
3. **D**: the task *relative deadline*, i.e., the maximum acceptable delay for its processing.
4. **T**: the task *period* (for periodic tasks).
5. **d**: the *absolute deadline* of the task (r+D). Transgression of the absolute deadline causes a timing fault.

For periodic tasks, when a task is ready, it releases a periodic request. The successive release times are $r_k = r_0 + k * T$, where r_0 is the first release and r_k is the $k + 1$th release. Similarly, the successive absolute deadlines are $d_k = r_k + D$.
A task set is said to be well formed if $0 < c \leq D \leq T$.

Derived Task Parameters

1. $u = c/T$: is the processor utilization factor (must be ≤ 1).
2. $ch = c/D$: is the processor load factor (must be ≤ 1).
3. **s**: is the start time of task execution.
4. **e**: is the finish time of task execution.
5. $D(t) = d - t$: is the residual relative deadline at time t ($0 \leq D(t) \leq D$).
6. $C(t)$: is the pending execution time at time t ($0 \leq C(t) \leq C$).

Task States



f: request abortion due to timing fault

- *Elected*: the dispatcher selects this task for execution and allocates the processor to it.
- *Blocked*: the task waits for a resource, a message or some synchronization signal.
- *Ready*: the task waits for election.
- *Passive*: The task has no current request.

Cyber-physical systems

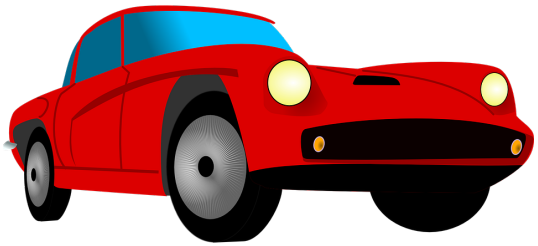
Cyber-physical systems (CPS)[a] use distributed embedded controllers to control physical processes. Examples may be found in several domains: automotive, robotics, medical devices, and smart grids.

[a] R. Alur, Principles of Cyber-Physical Systems. MIT Press, 2015.

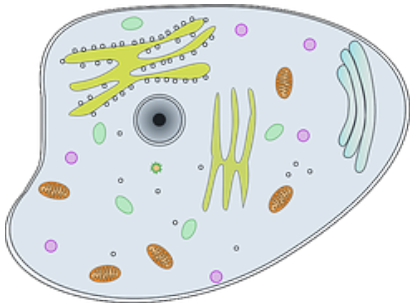
2

[2] Figure reproduced from http://icc.mtu.edu/cps/

- Hybrid automata (HA) is a major enabler for the formalization of CPS.
- A combination of ODEs to model the continuous dynamics and FSMs to model the discrete mode changes that are induced by the controller.

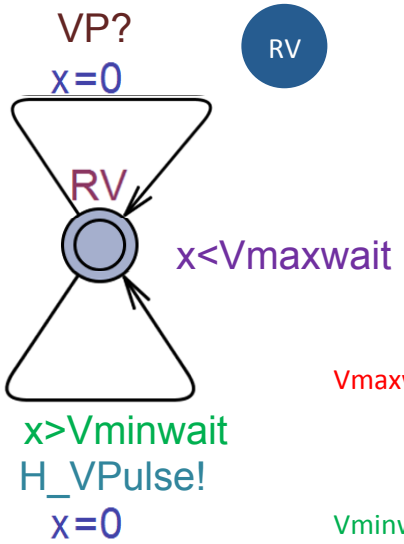


Model: Car
Discrete: Changing gears
Continuous: Throttle control

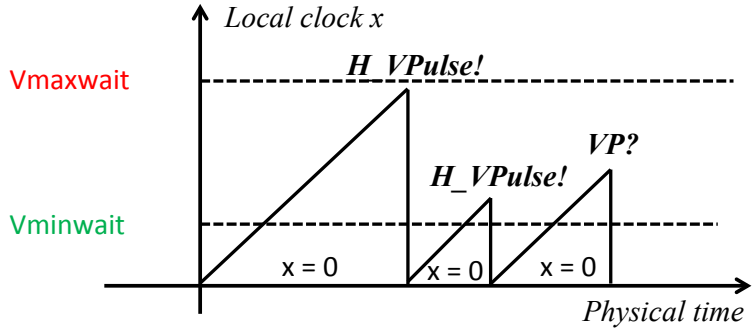


Model: Cell biology
Discrete: External stimulus
Continuous: Flow of ions

Timed Automata model for the Heart



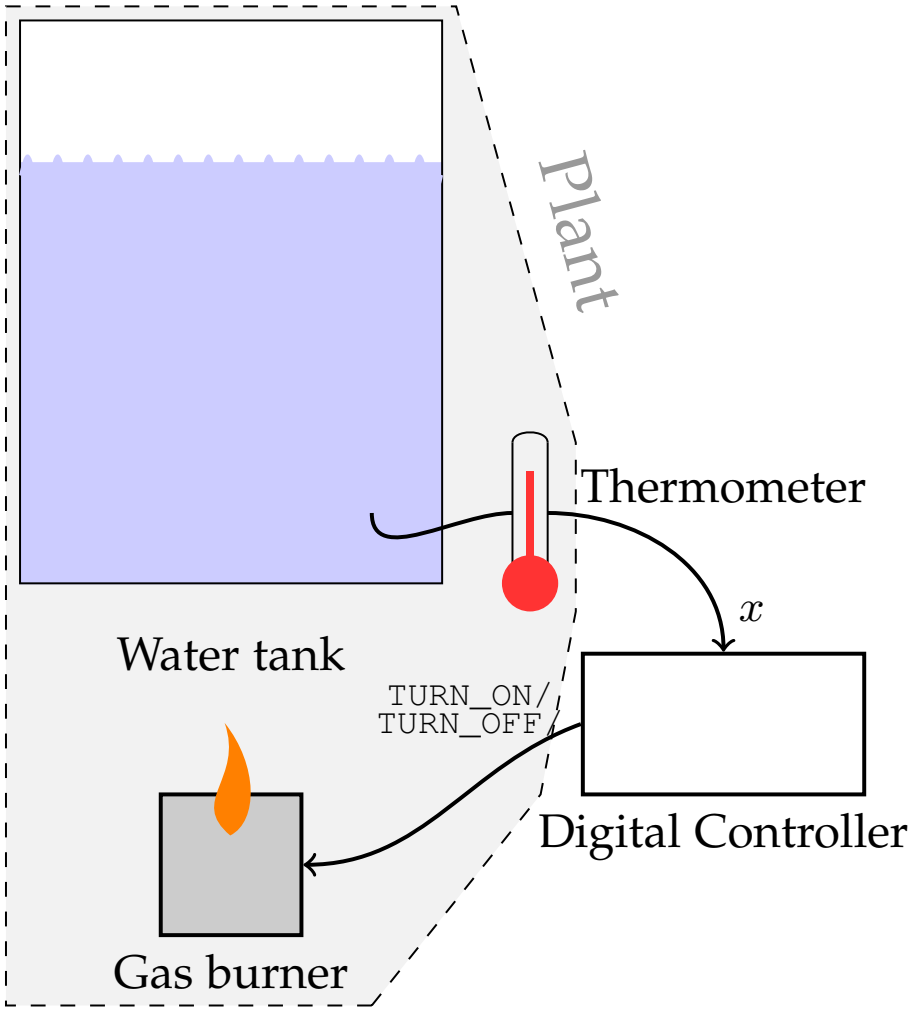
- Input: *VP*
- Output: *H_VPulse*
- Local Clock: *x*
- Clock Bound: x ∈[*Vminwait*,*Vmaxwait*]



[Ref.]: Pajic, M., Zhihao Jiang, Insup Lee, Sokolsky, O. and Mangharam, R., "From Verification to Implementation: A Model Translation Tool and a Pacemaker Case Study", *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012 IEEE 18th , pp. 173,184, 16-19 April 2012.

- The random heart model generates `H_APulse`, `H_VPulse` randomly in the interval [*Minwait*, *Maxwait*].
- Progression of time modelled using real-valued clock variables `Clock1`, `Clock2`.
- Invariants introduce *fairness* i.e. locations have to be exited before invariants become false.

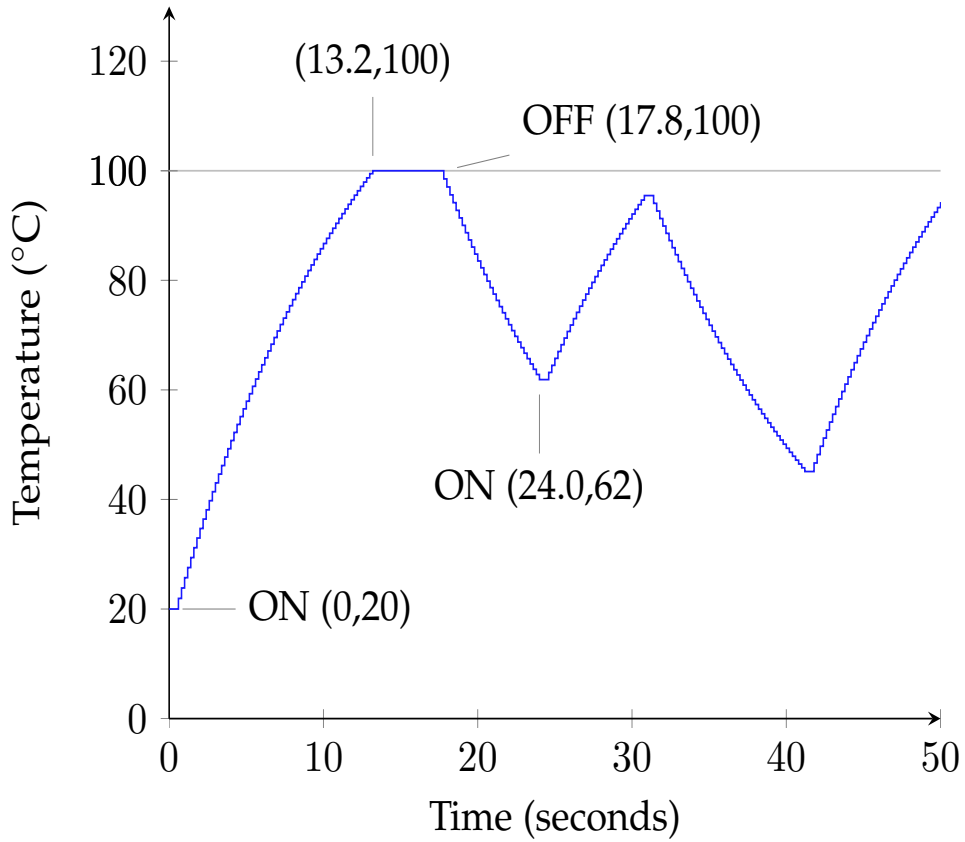
A water tank temperature controller



[a]

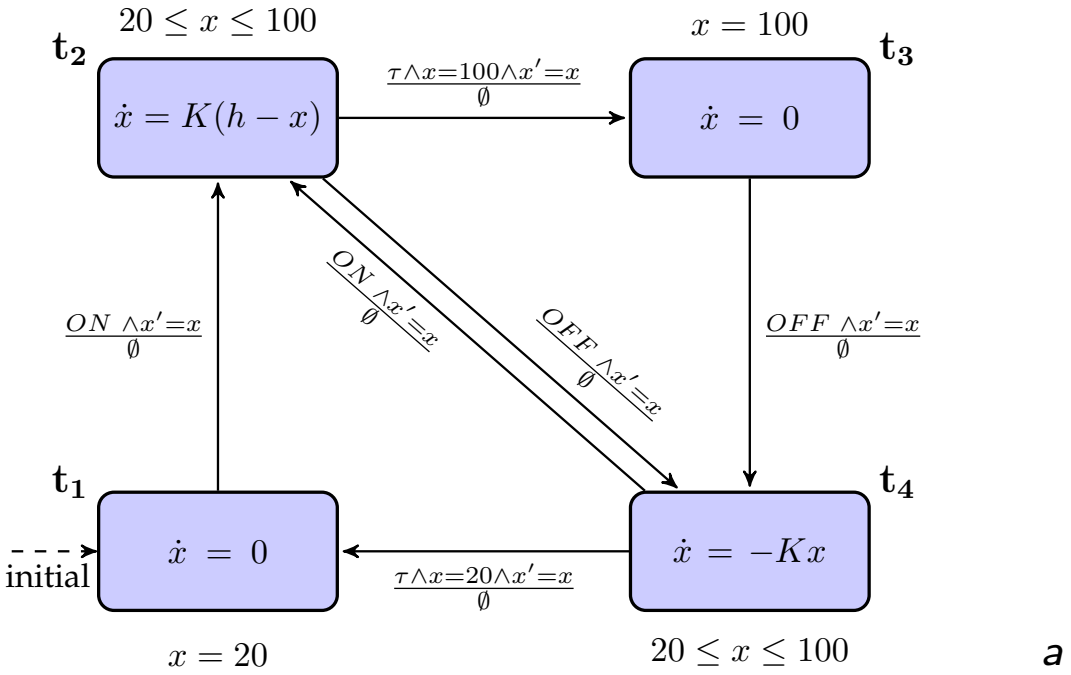
[a] J-F Raskin, "An introduction to hybrid automata", Handbook of Networked and Embedded Control Systems Control Engineering 2005, pp 491-517.

A water tank temperature controller



- Temperature of water inside a tank may be modelled as $x(t) = Ie^{-Kt} + h(1 - e^{-Kt})$ where:
- I is the initial temperature.
- K is a constant that depends on the tank conductivity.
- h is a constant that depends on the power of the gas burner.

A hybrid automata example

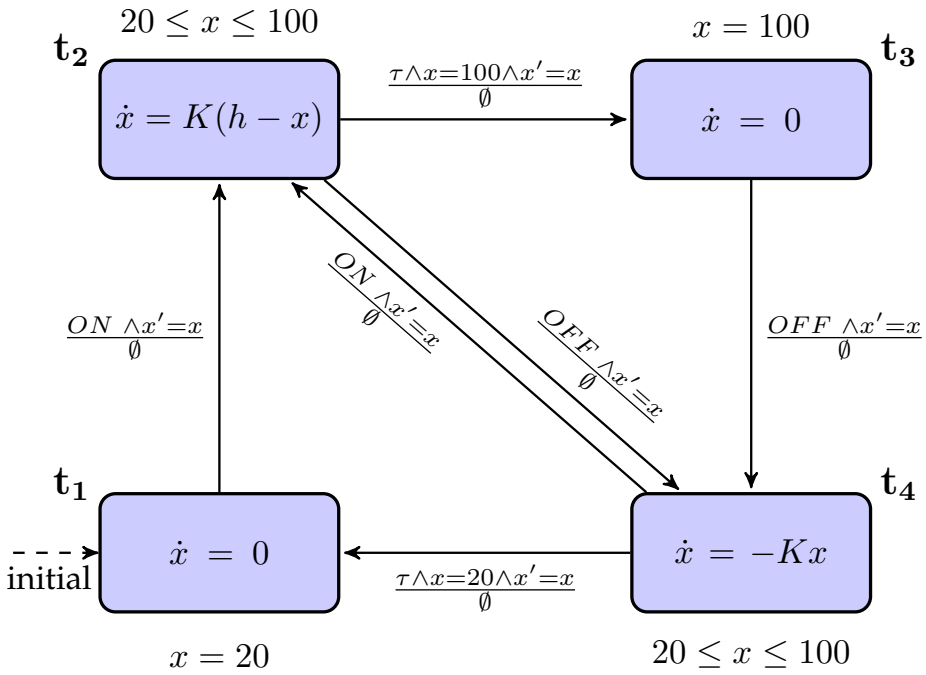


[a]

- Four locations $t1, ..t4$ that represent the discrete modes.
- Each location has some flow predicates that specify the rate of change of the continuous variables.

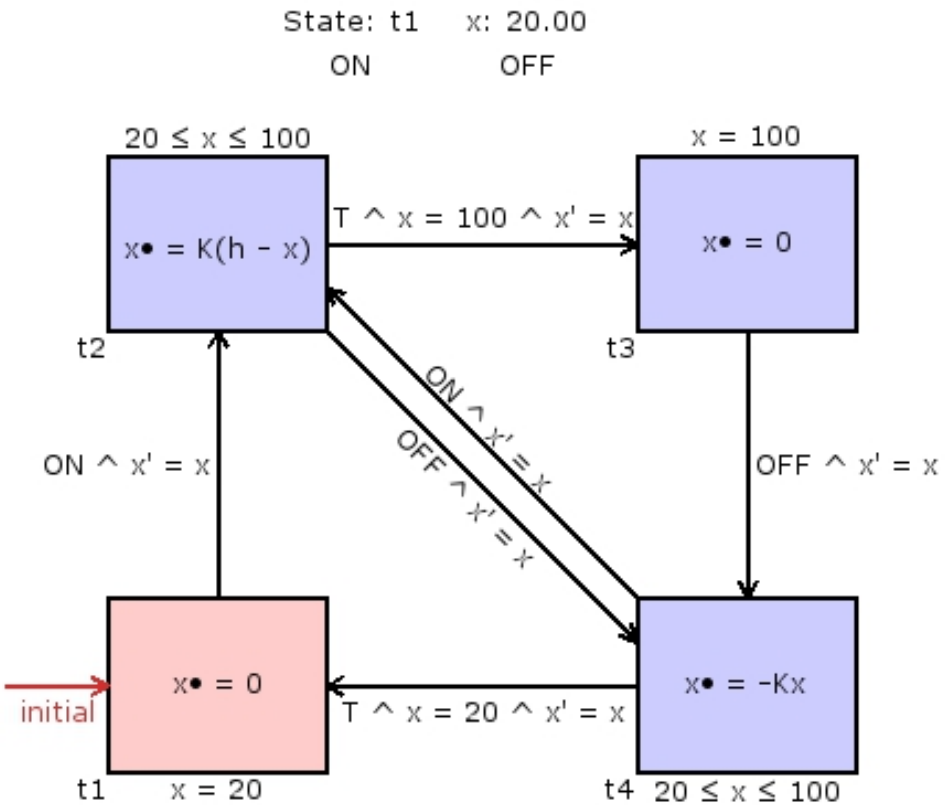
[a] J-F Raskin, "An introduction to hybrid automata", Handbook of Networked and Embedded Control Systems Control Engineering 2005, pp 491-517.

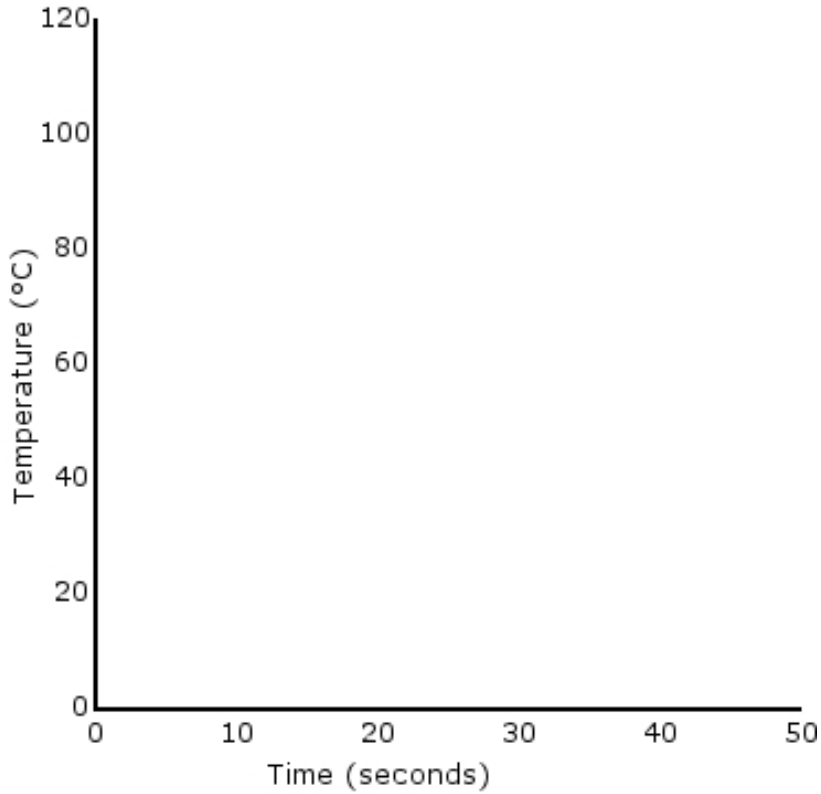
A hybrid automata example



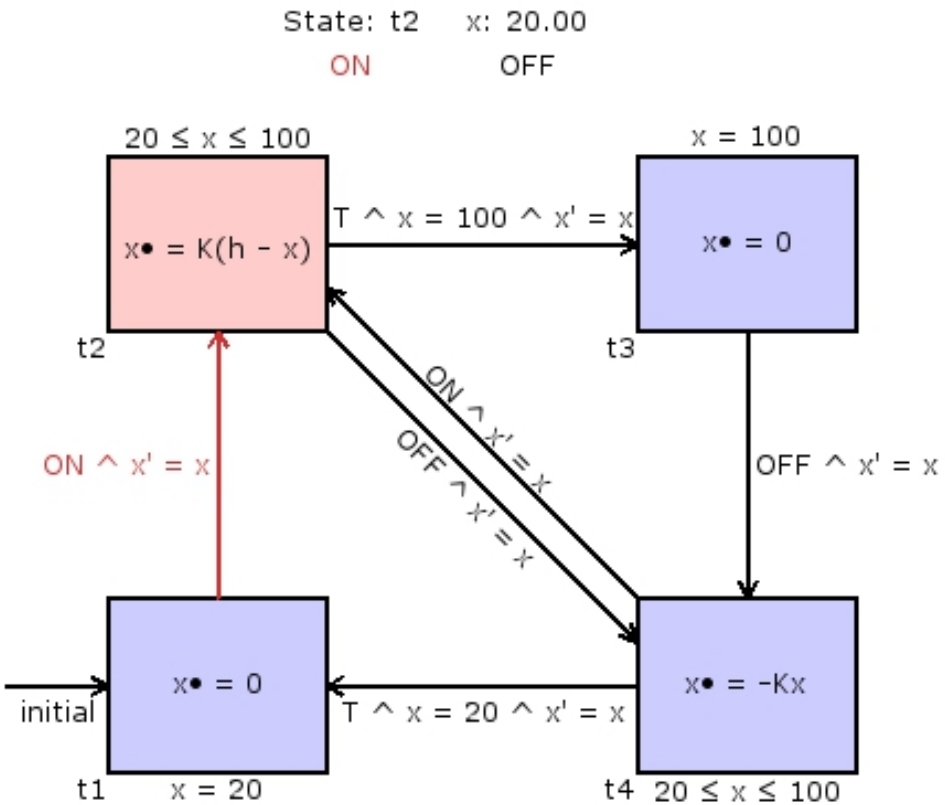
- Invariants are associated with locations e.g. $20 \leq x \leq 100$ is an invariant associated with $t1$. Execution remains in a location until the invariant holds.
- Some locations may have initialization conditions that provide the initial values of the variables.
- A transition is enabled when the input is present and the jump condition associated with the transition holds. When a given transition is taken the final value of the variables are updated.

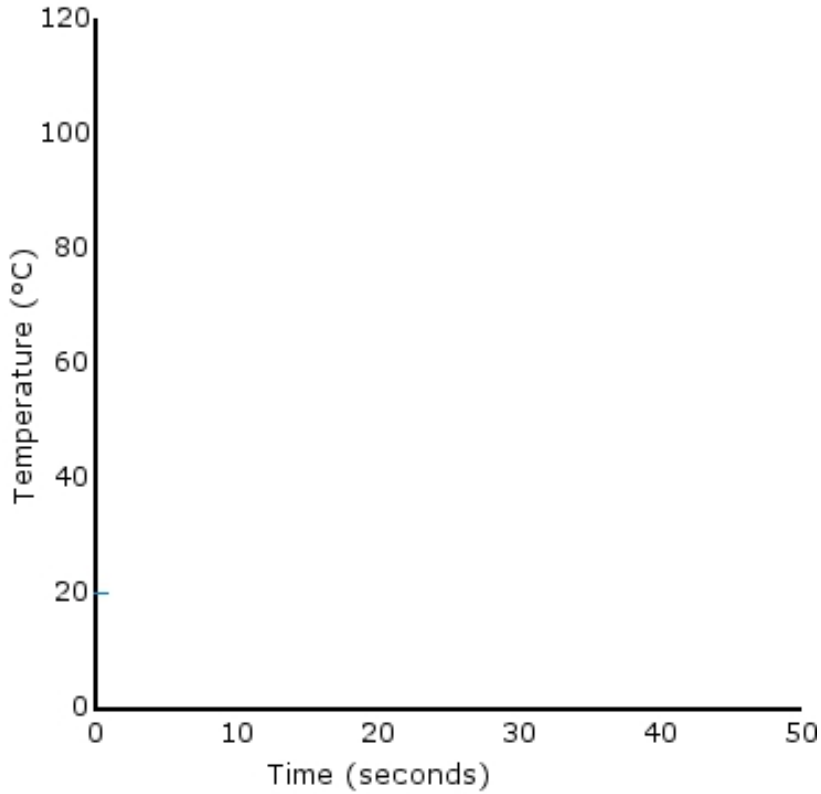
A hybrid automata example



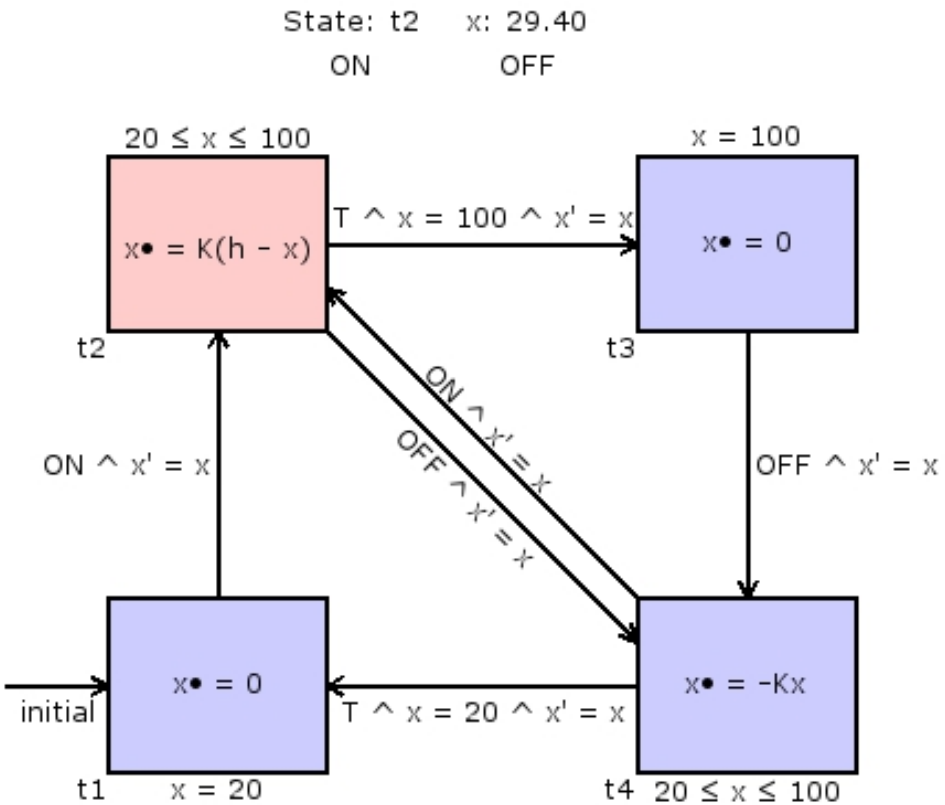


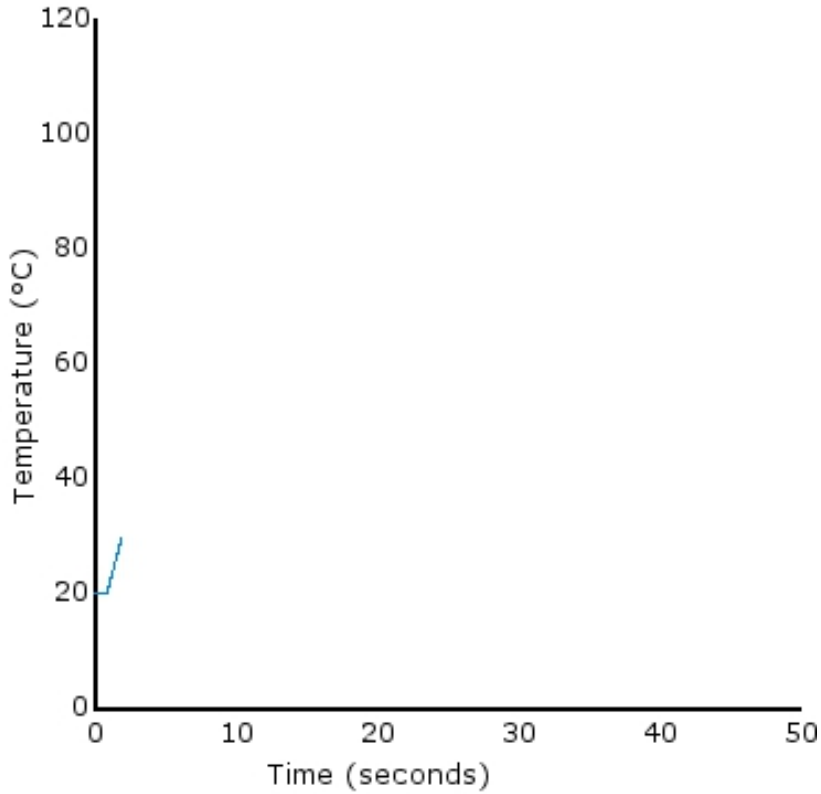
A hybrid automata example



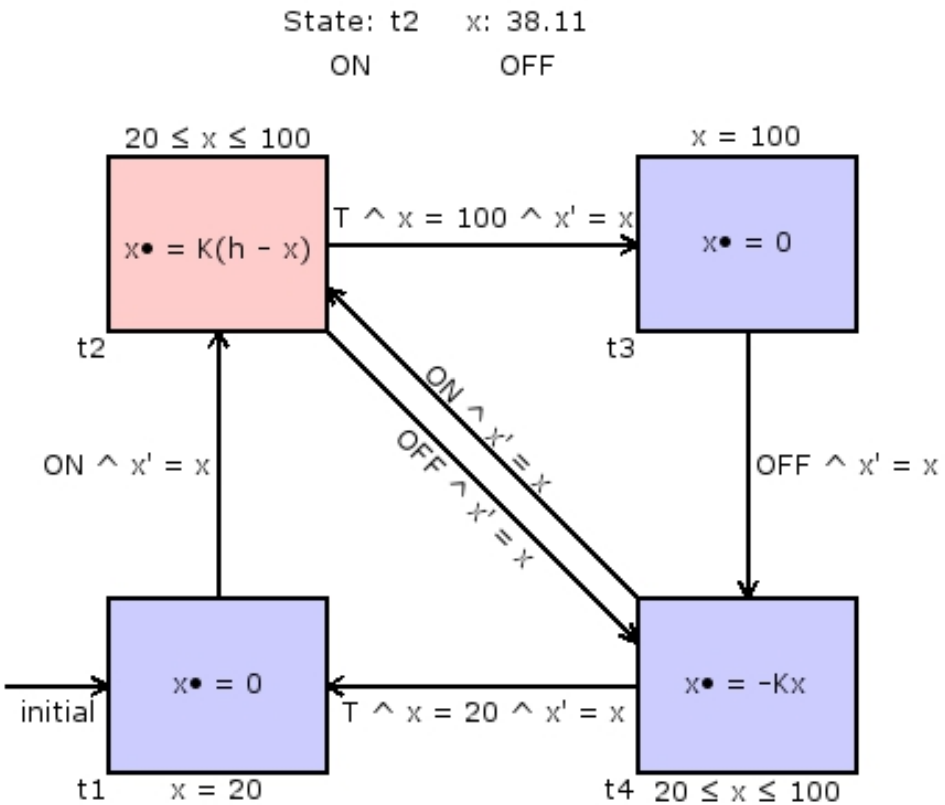


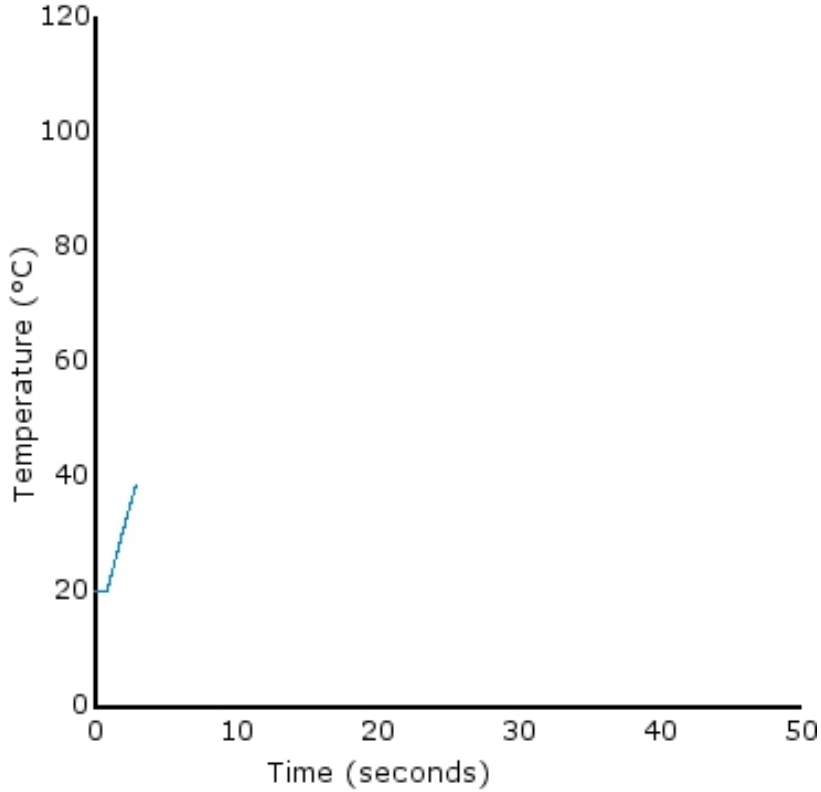
A hybrid automata example



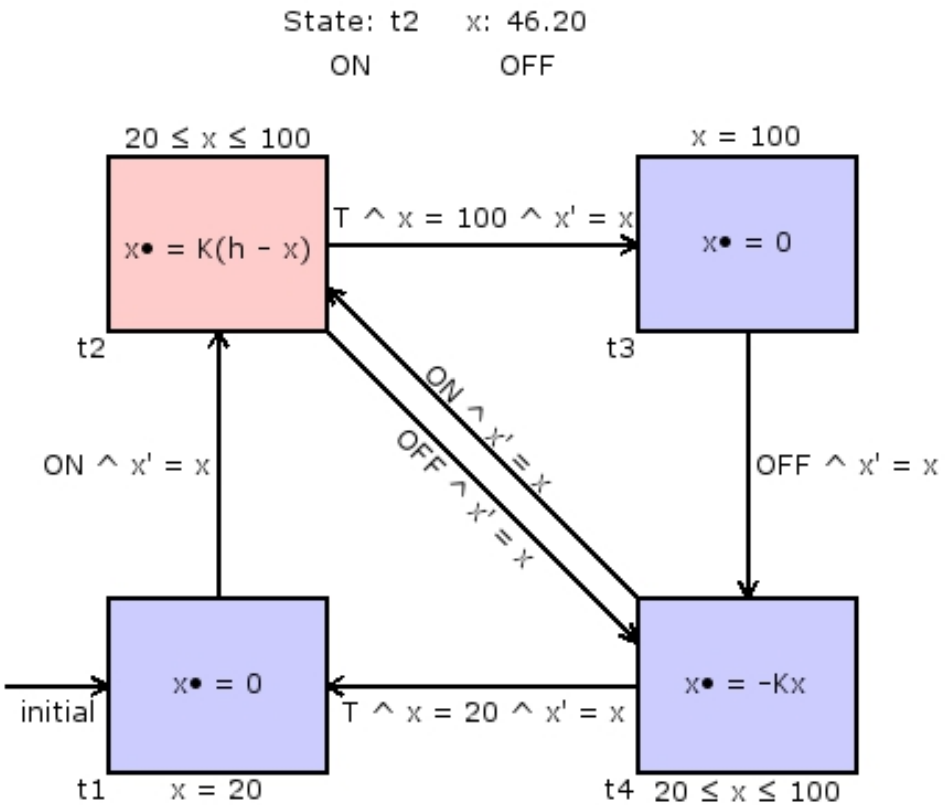


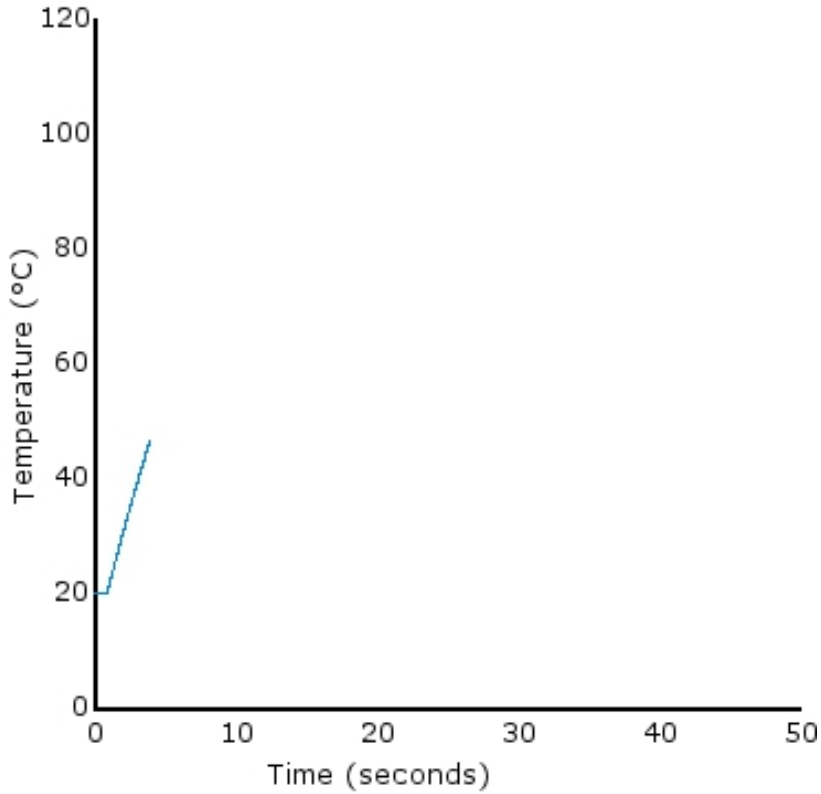
A hybrid automata example



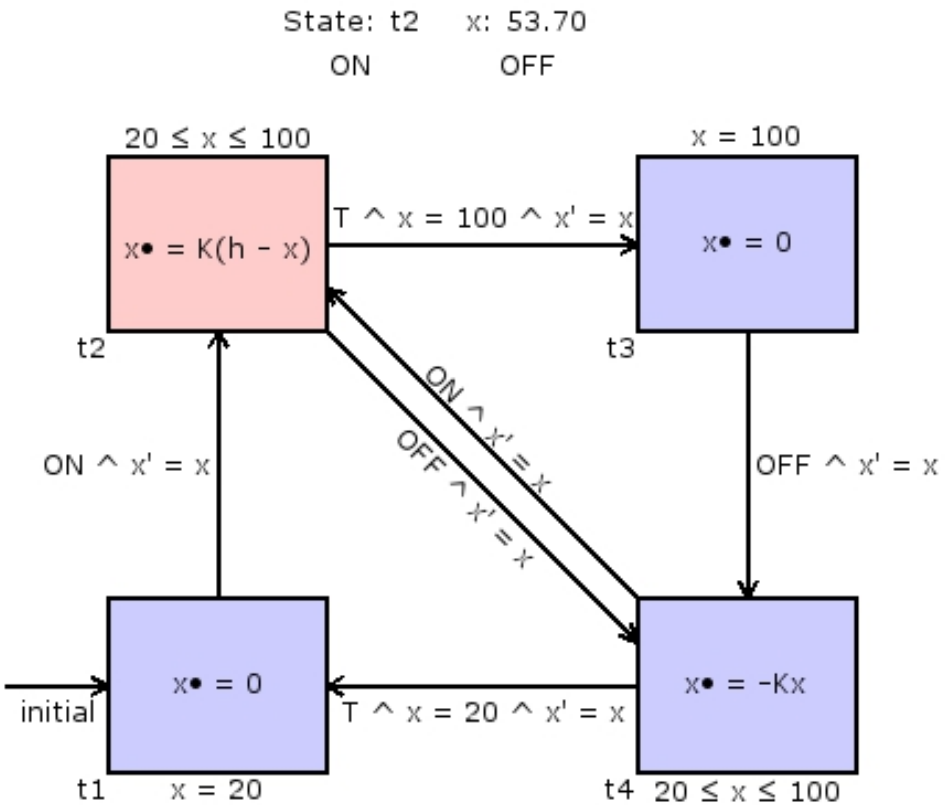


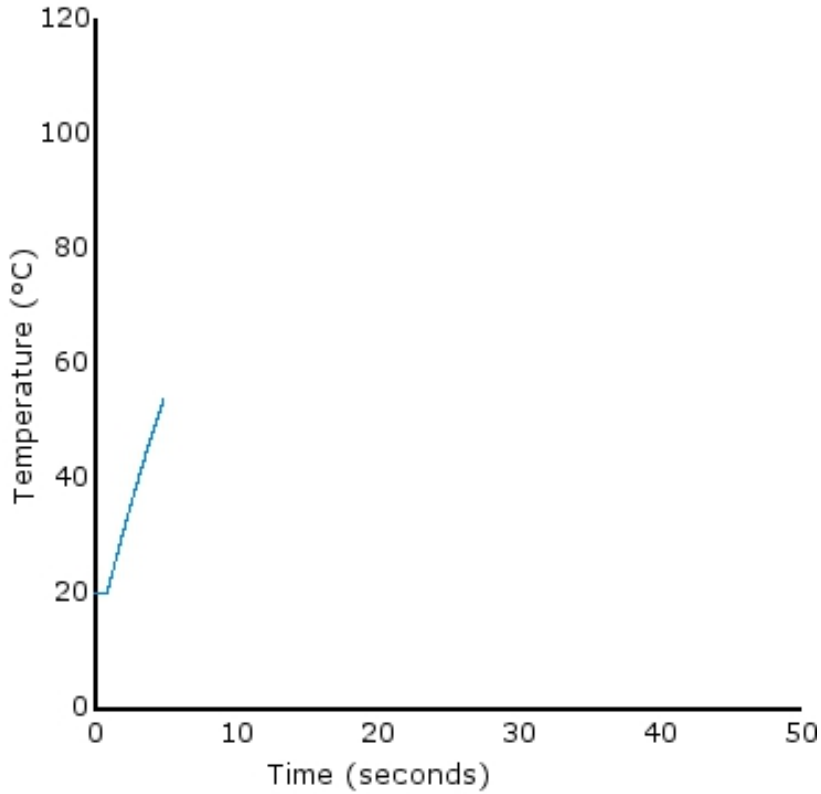
A hybrid automata example



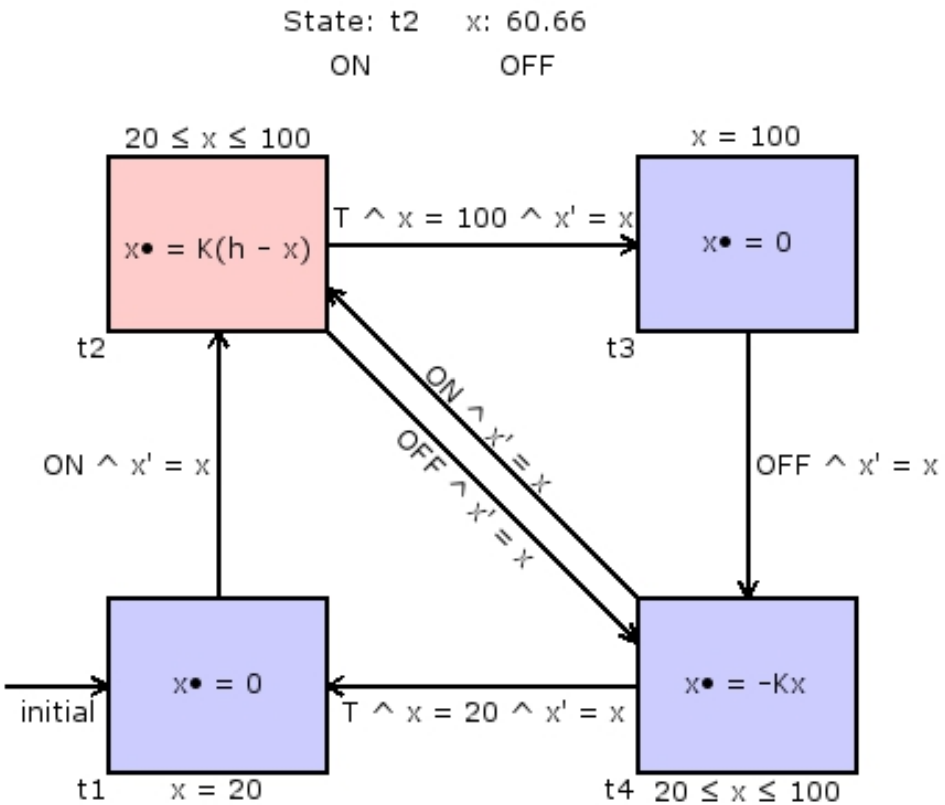


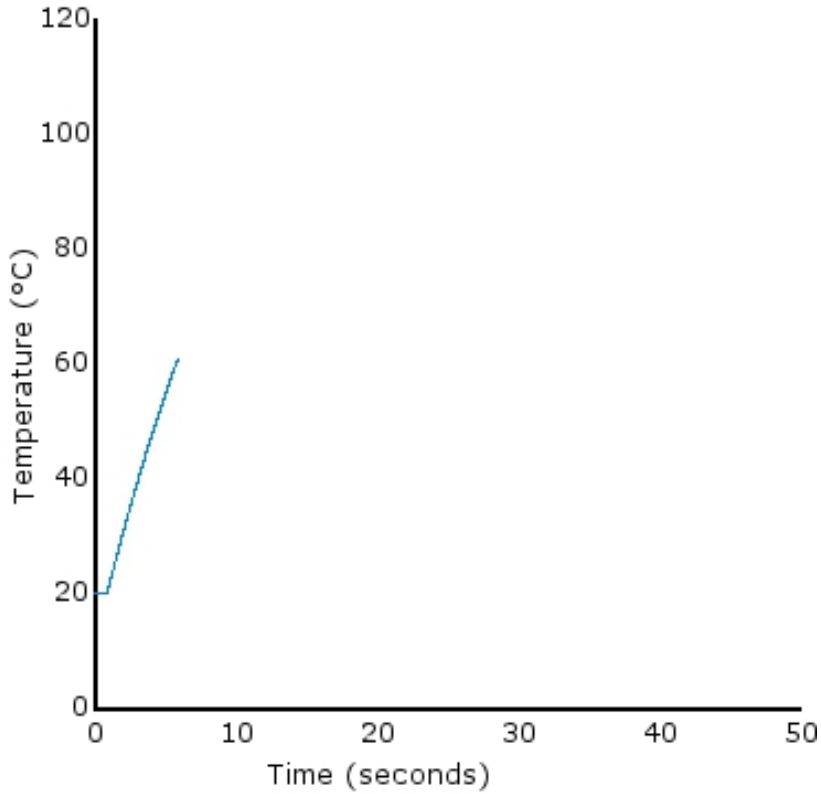
A hybrid automata example



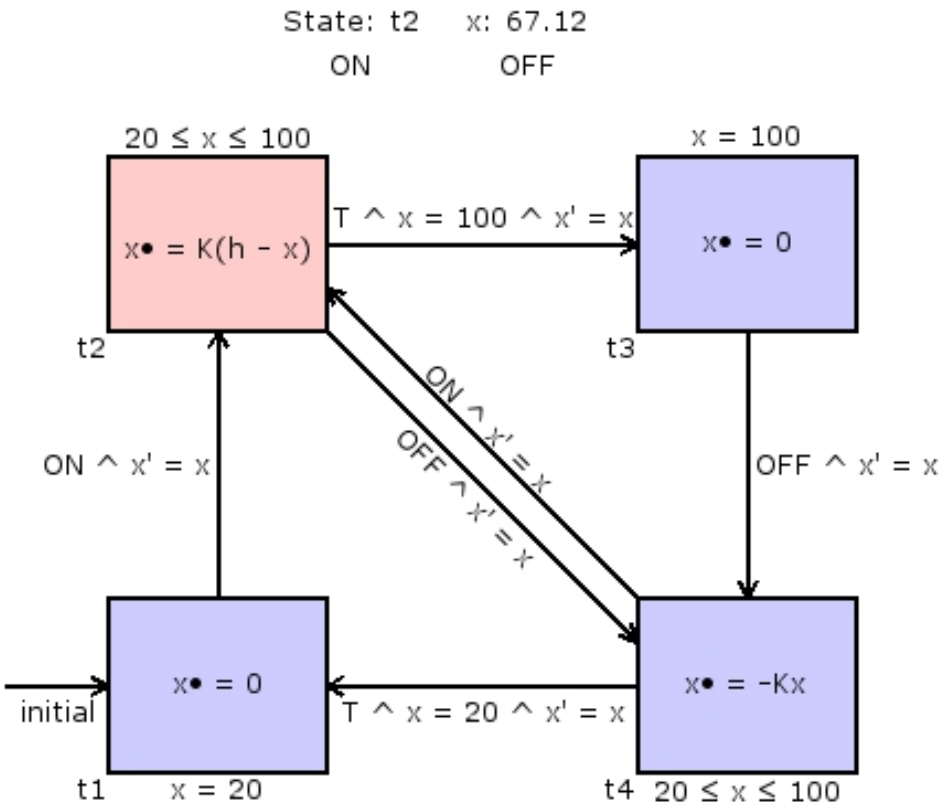


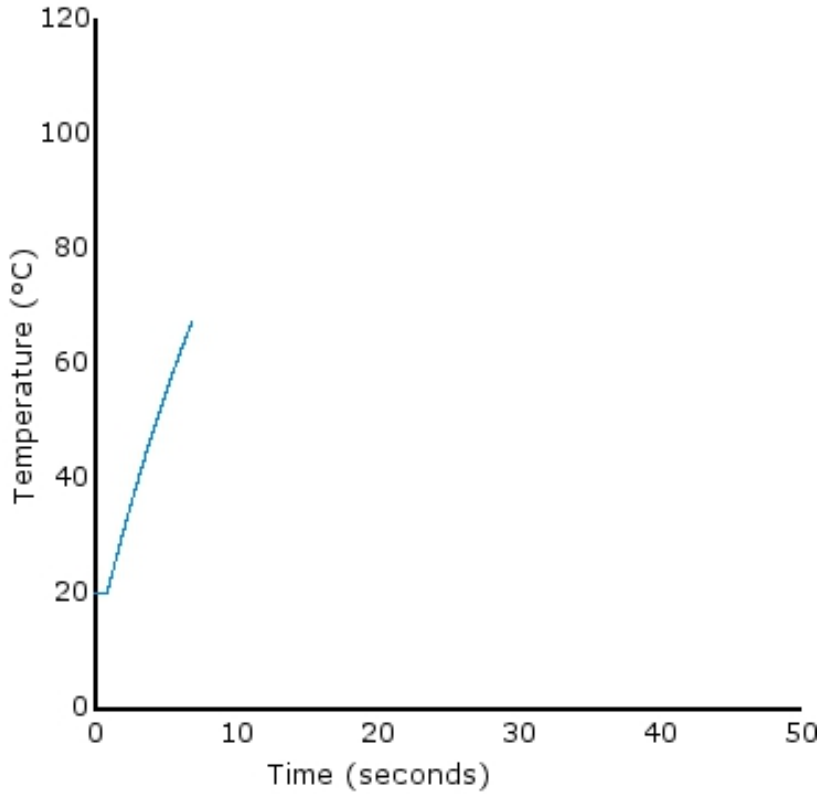
A hybrid automata example



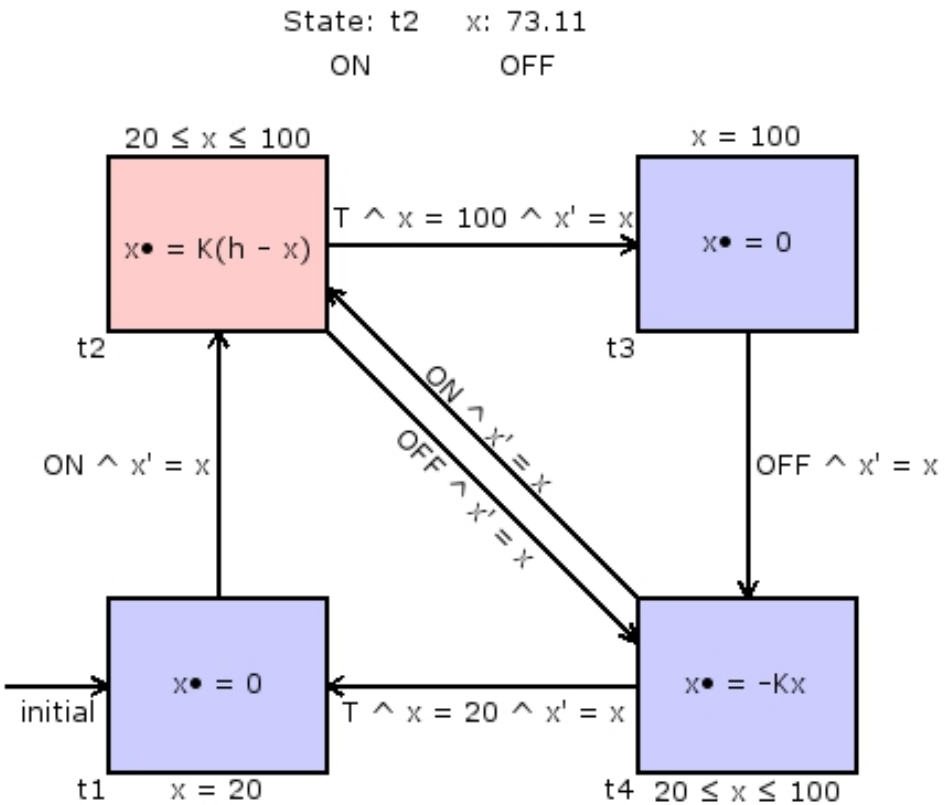


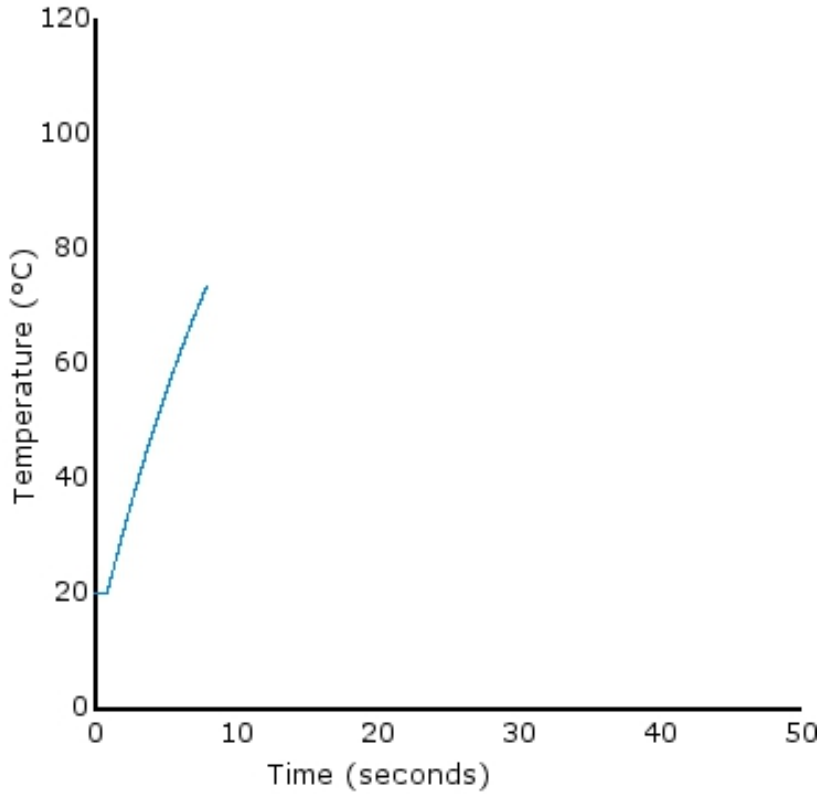
A hybrid automata example



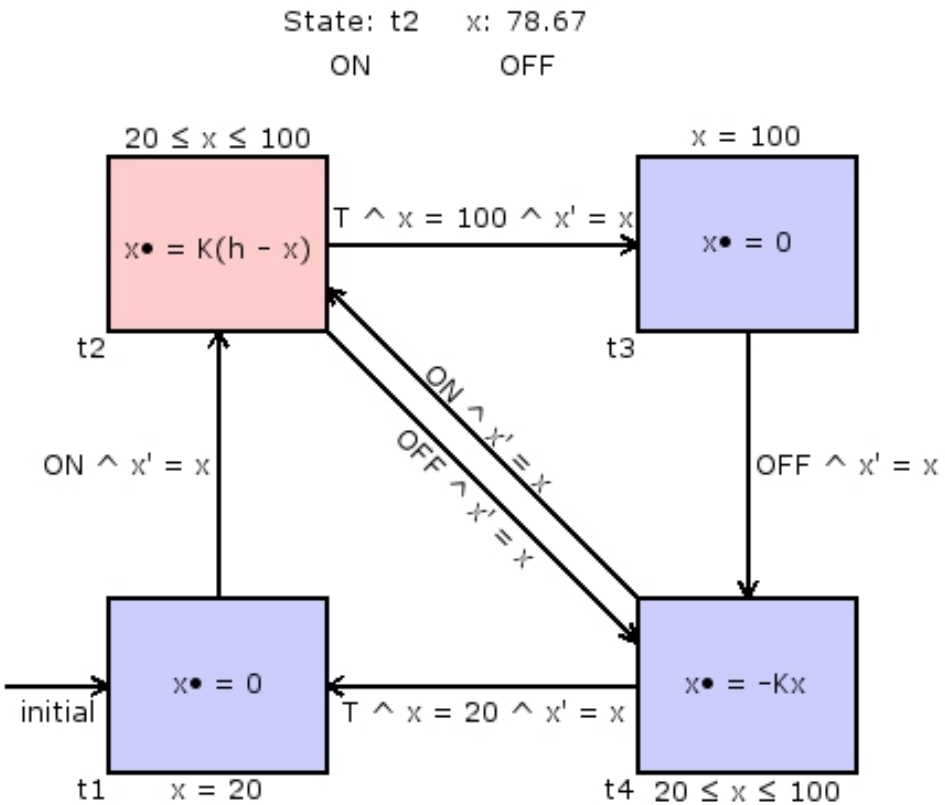


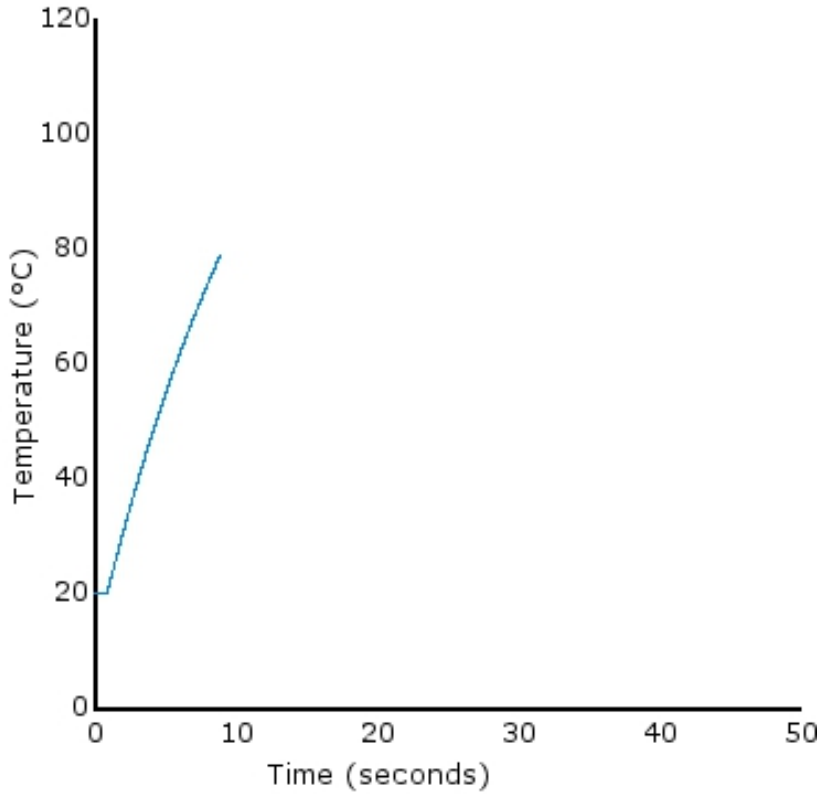
A hybrid automata example



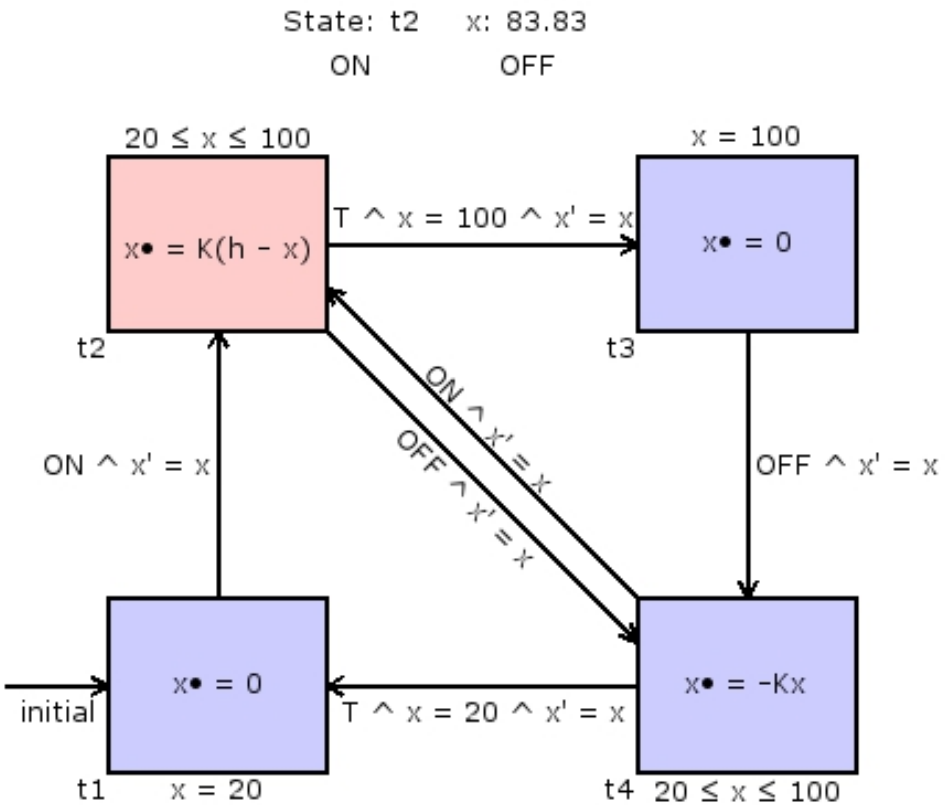


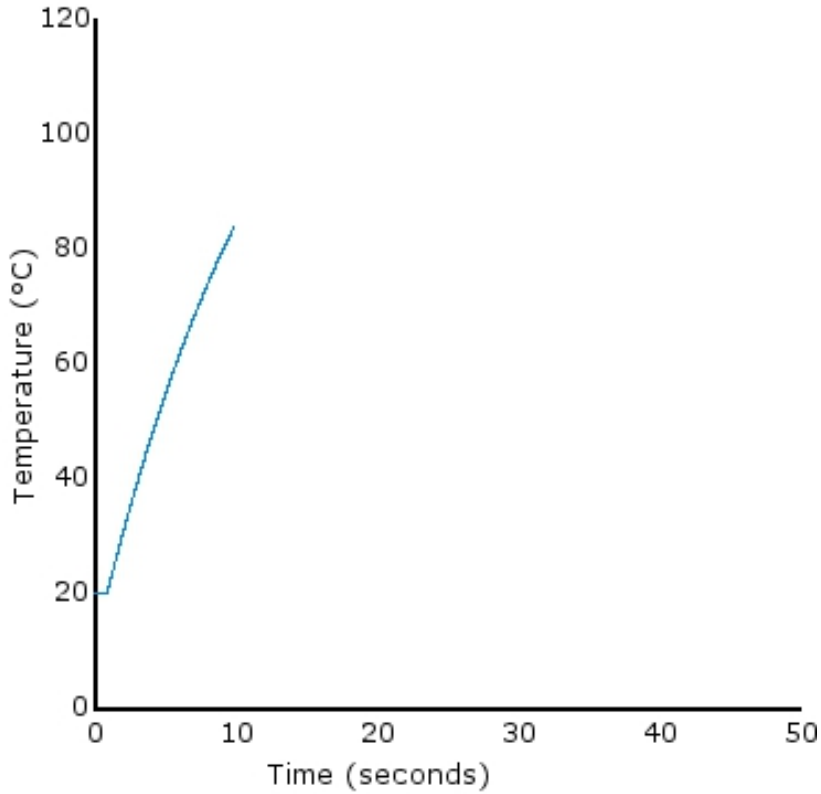
A hybrid automata example



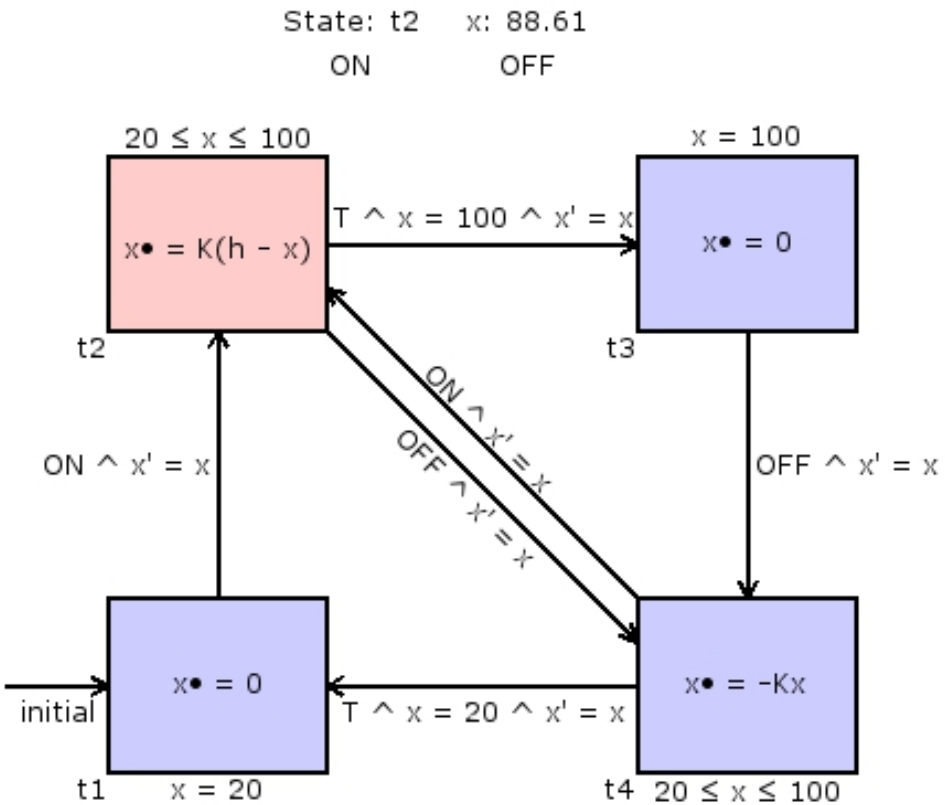


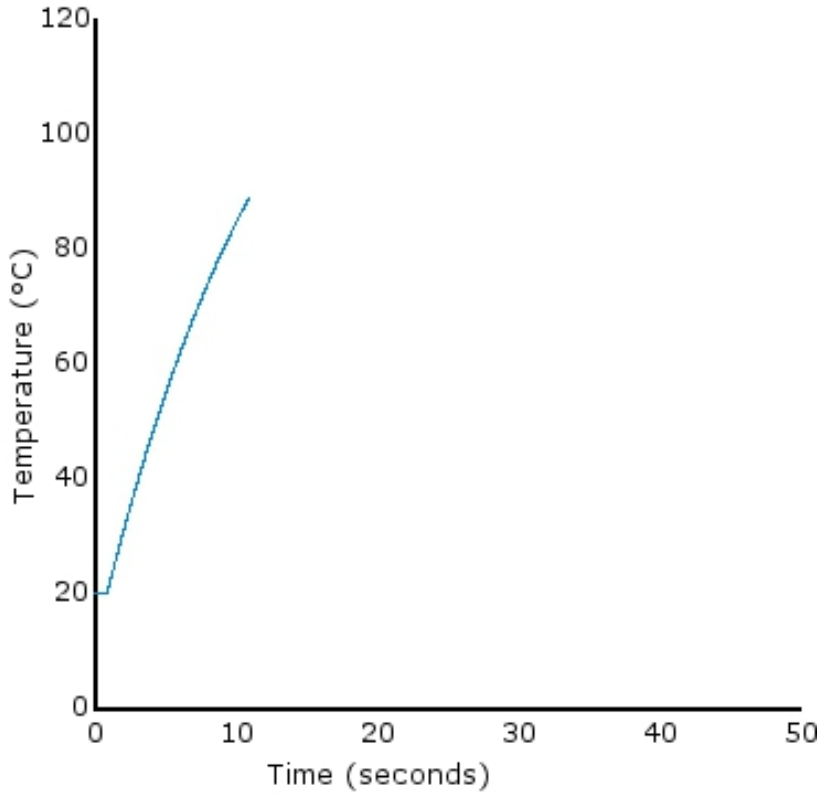
A hybrid automata example



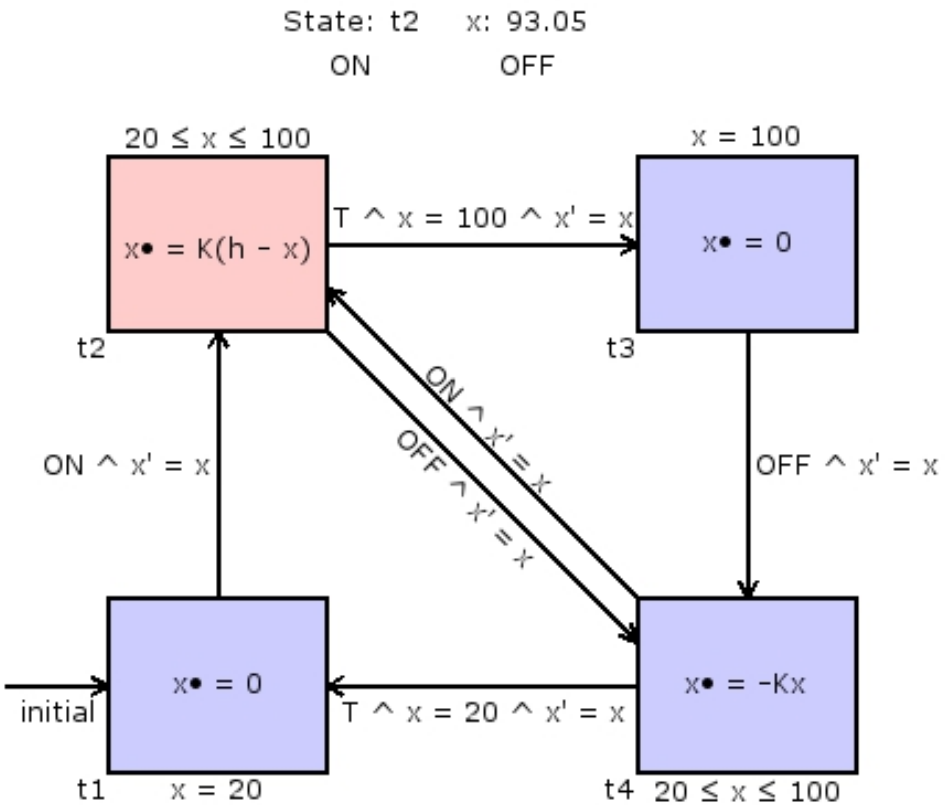


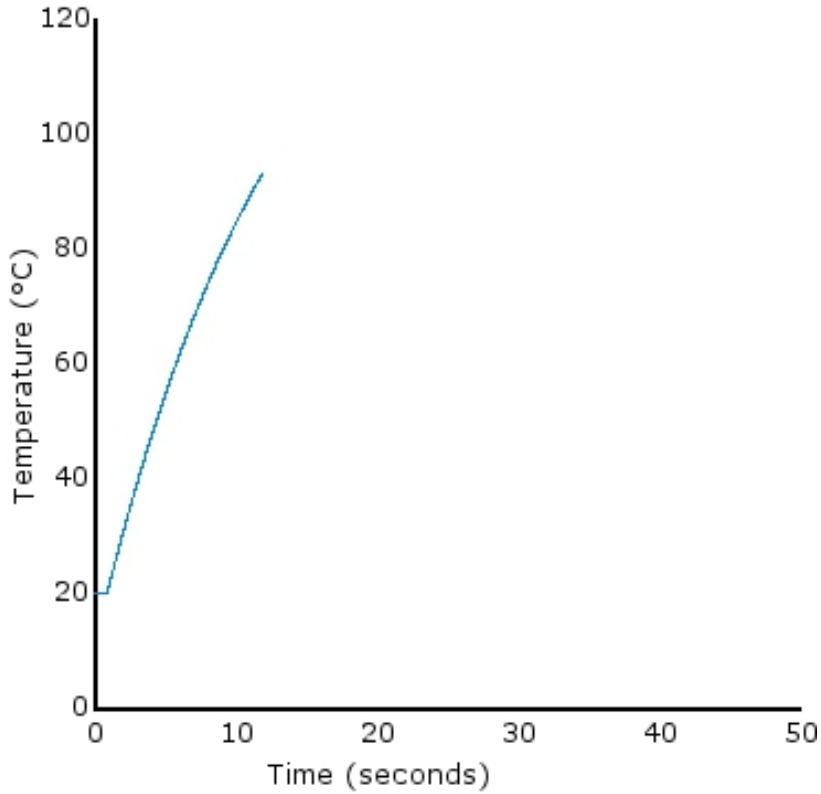
A hybrid automata example



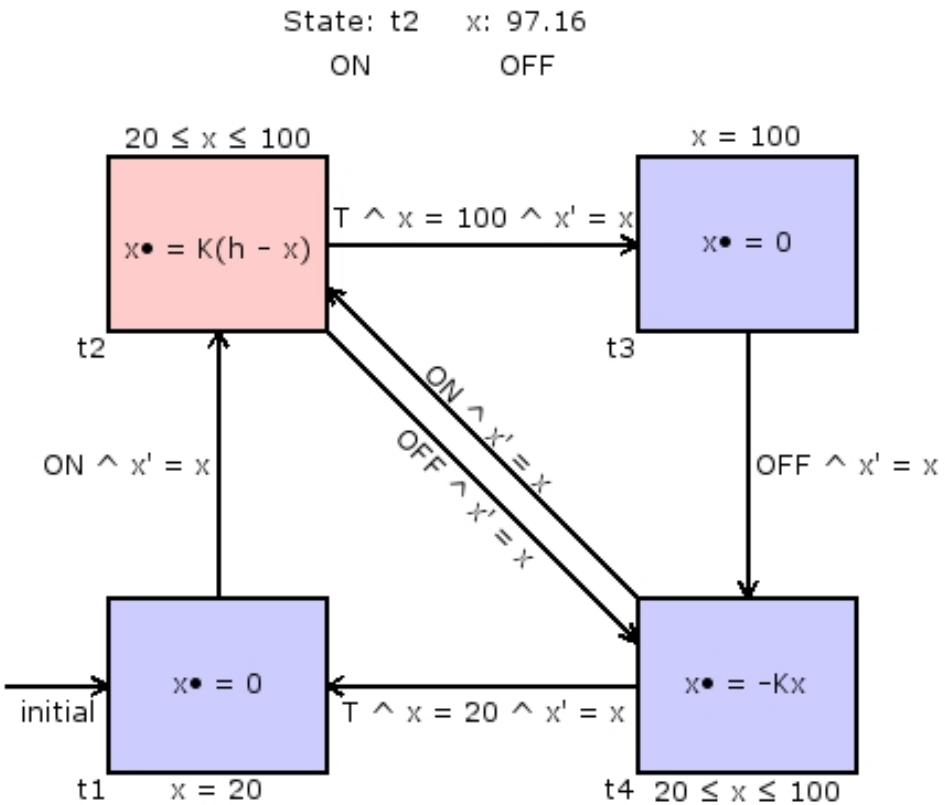


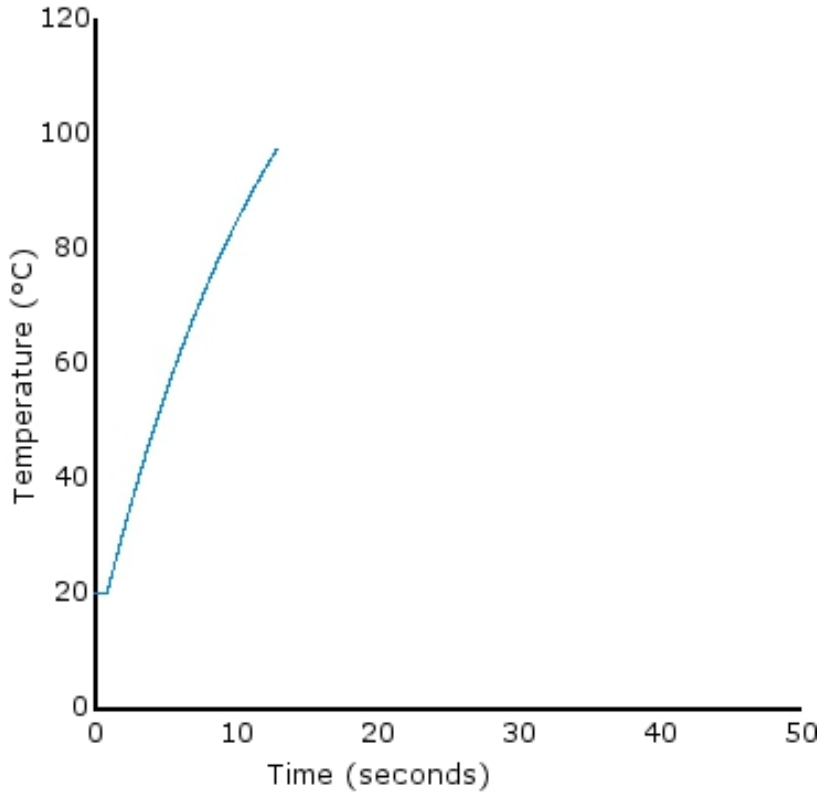
A hybrid automata example



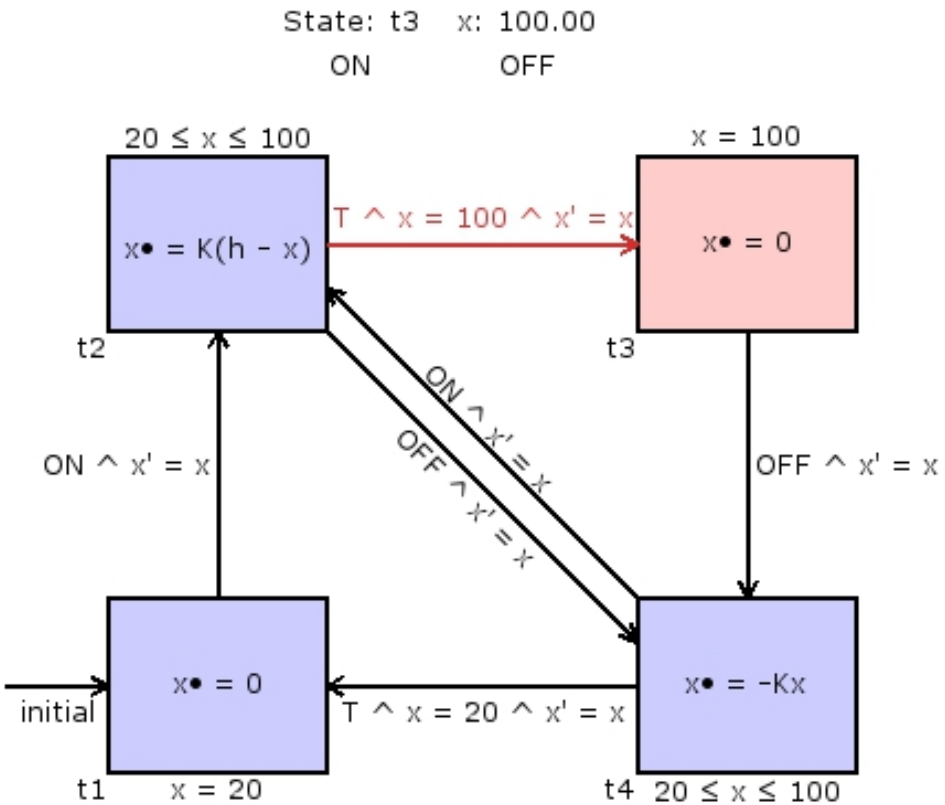


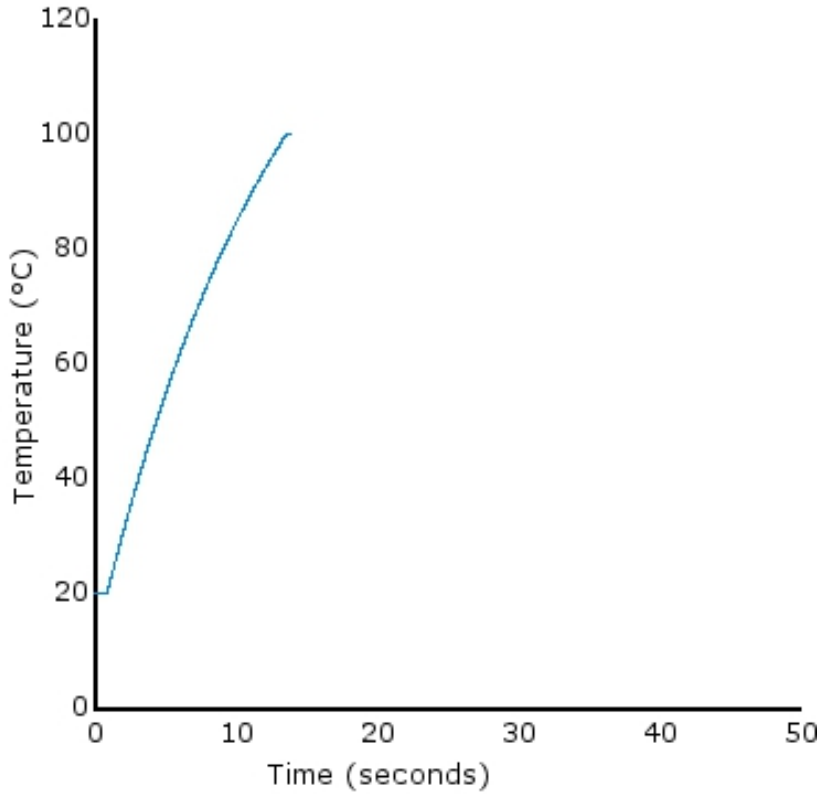
A hybrid automata example



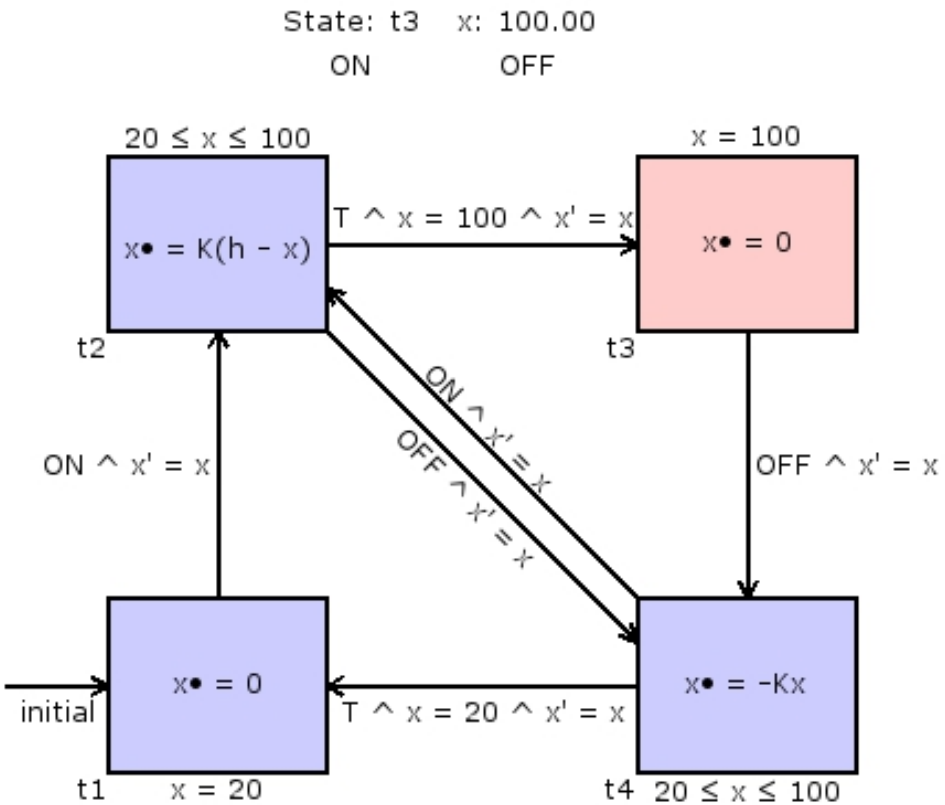


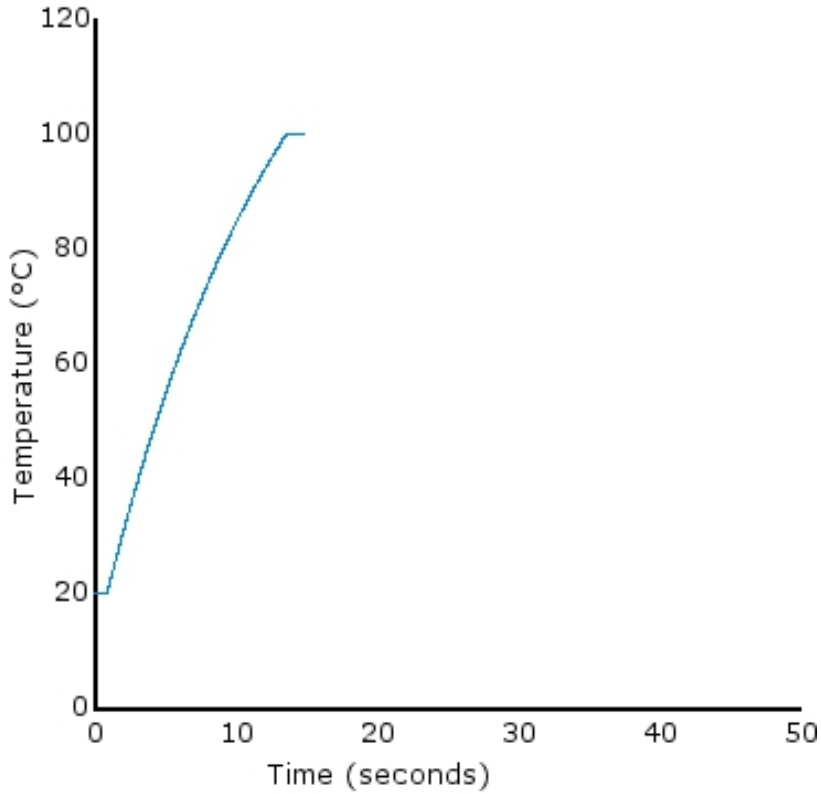
A hybrid automata example



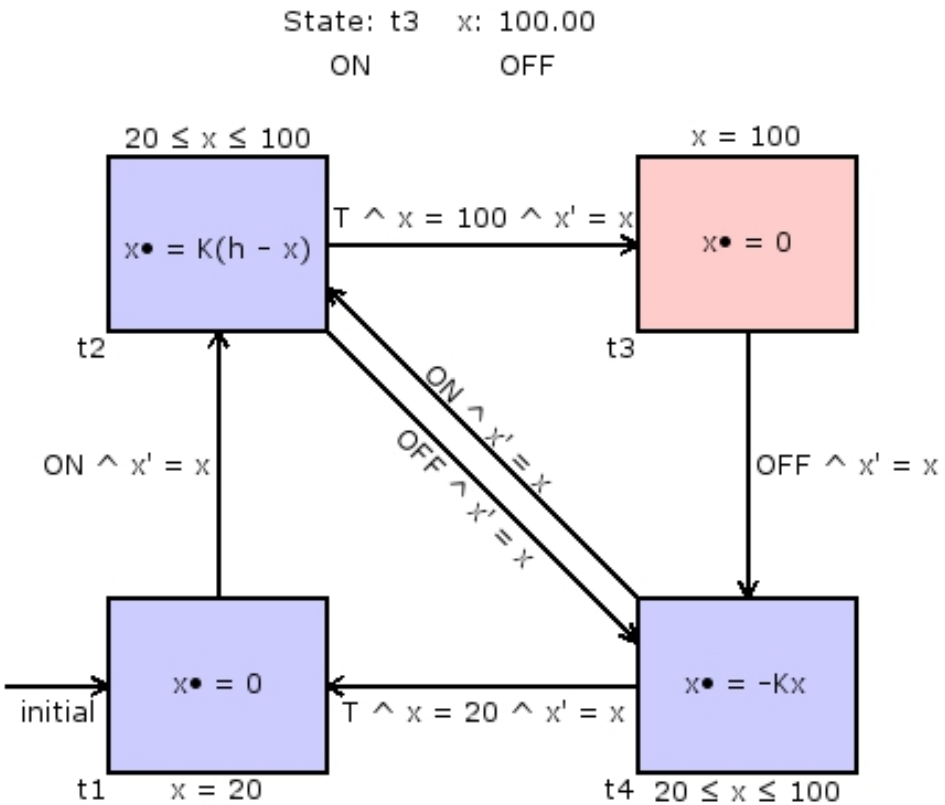


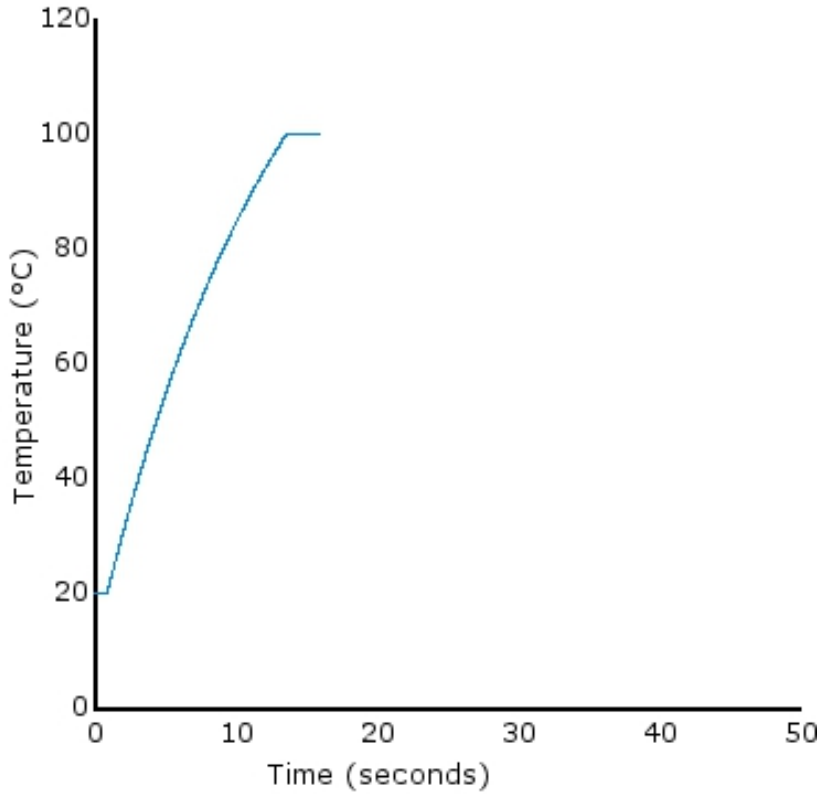
A hybrid automata example



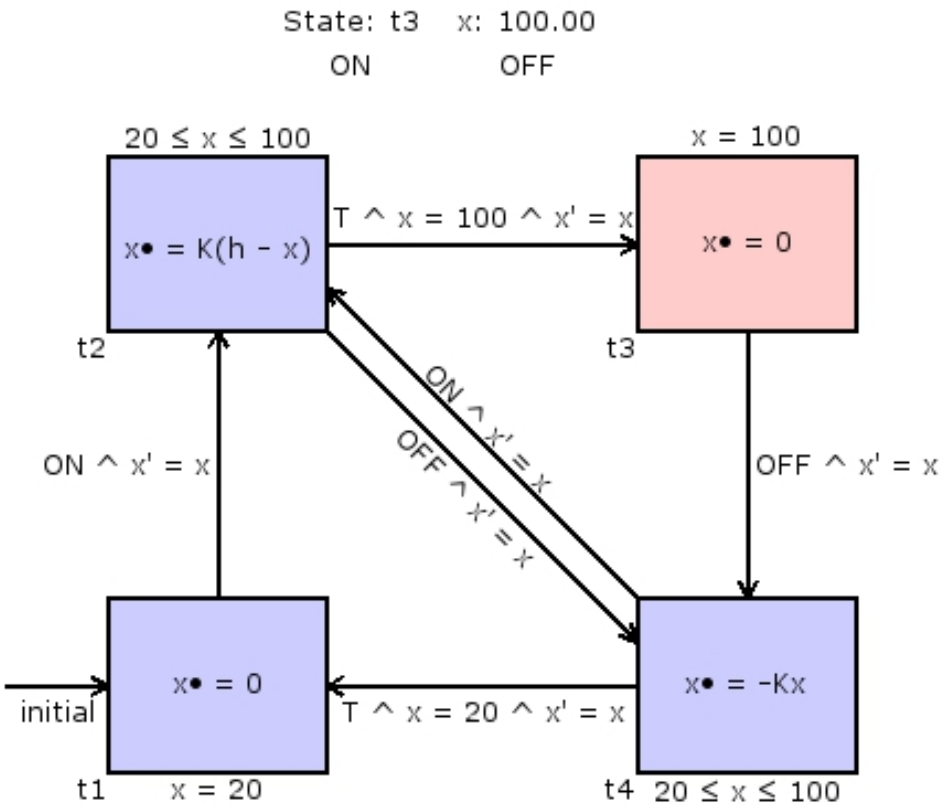


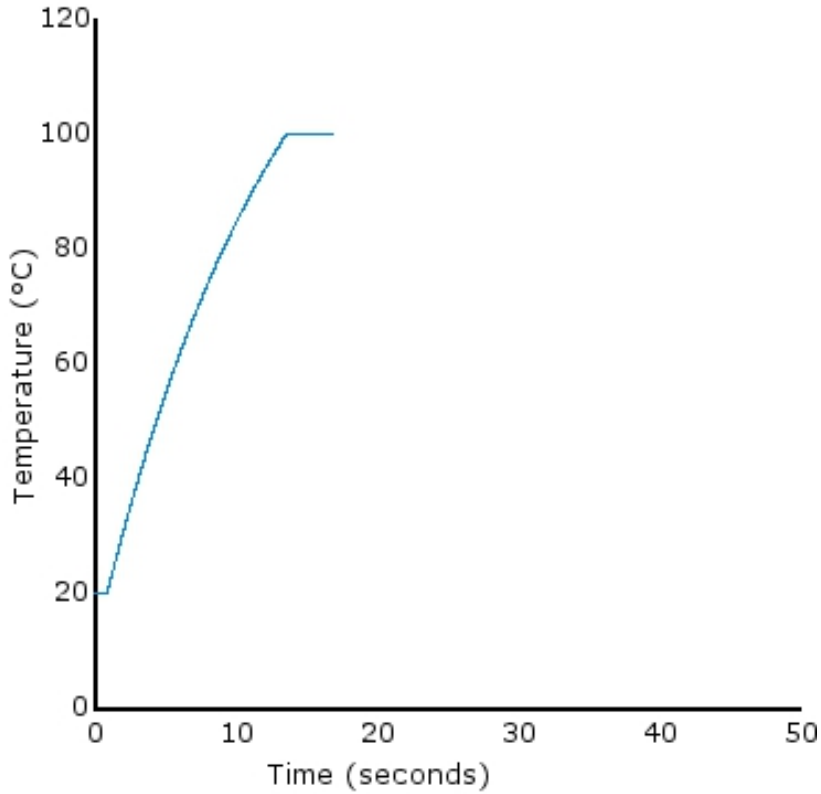
A hybrid automata example



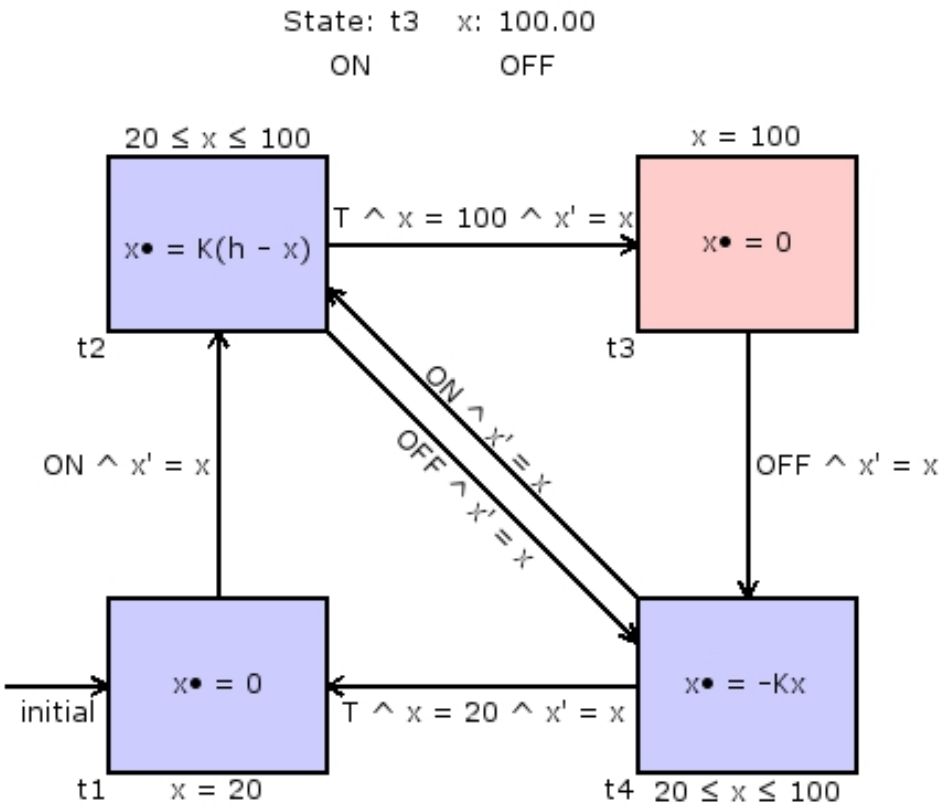


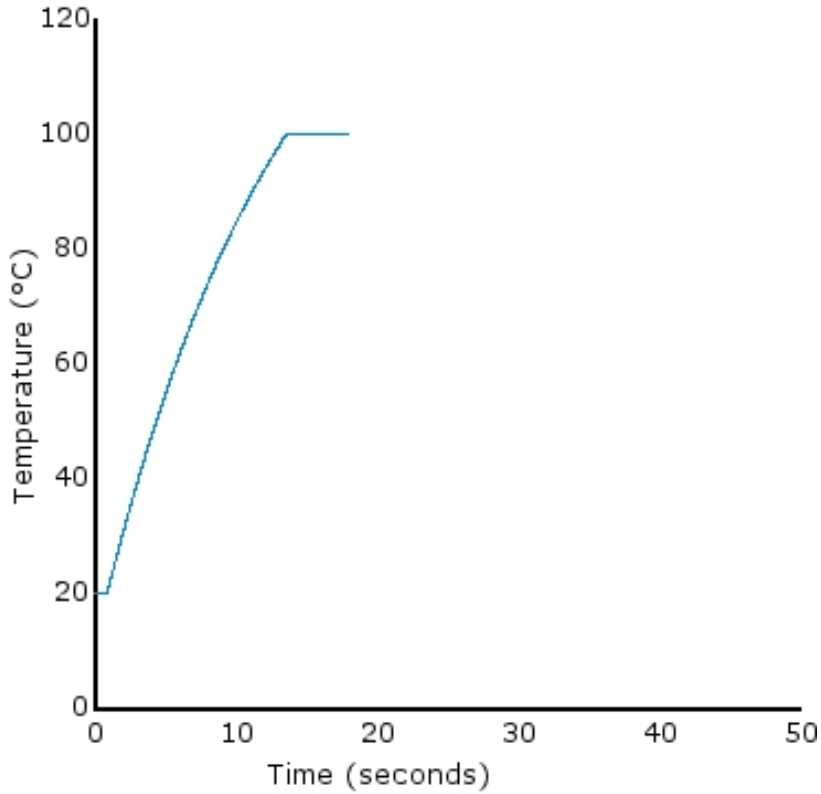
A hybrid automata example



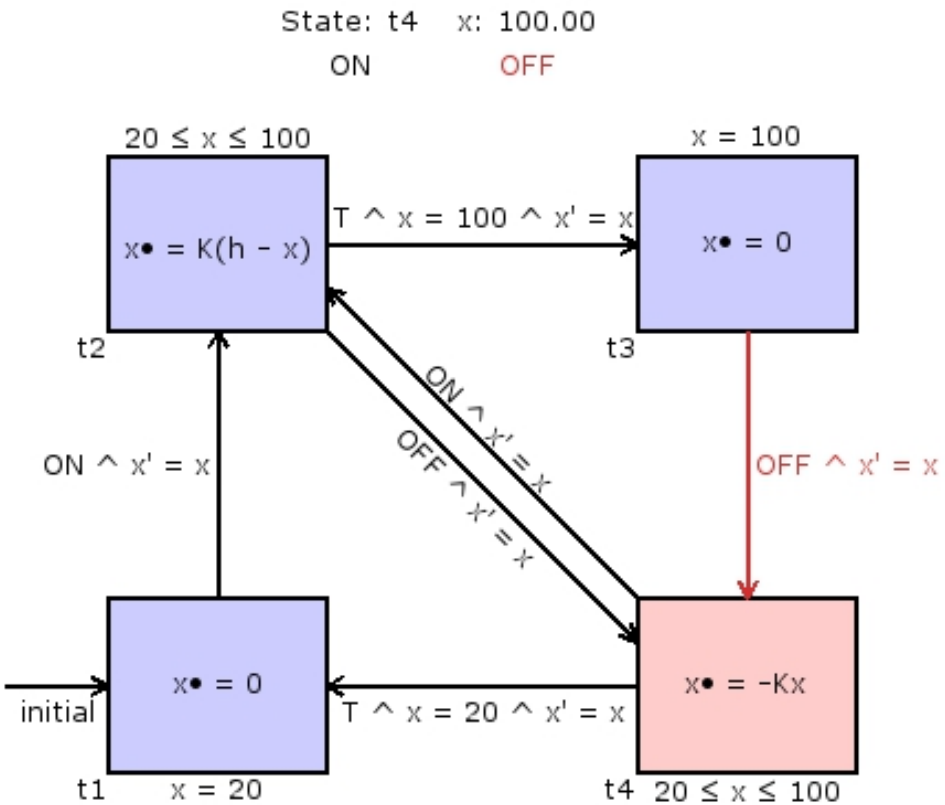


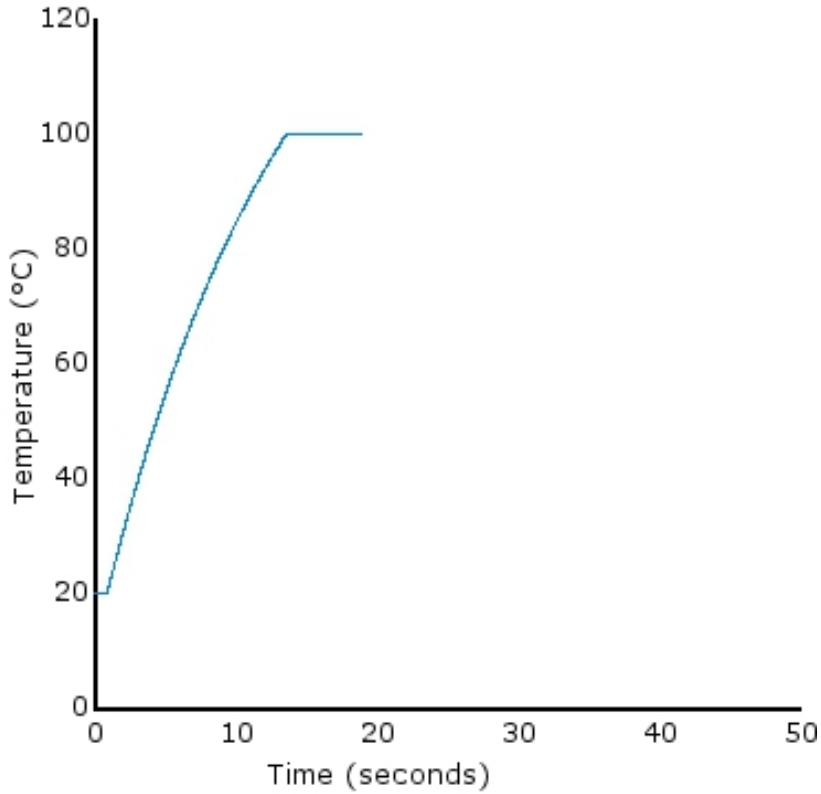
A hybrid automata example



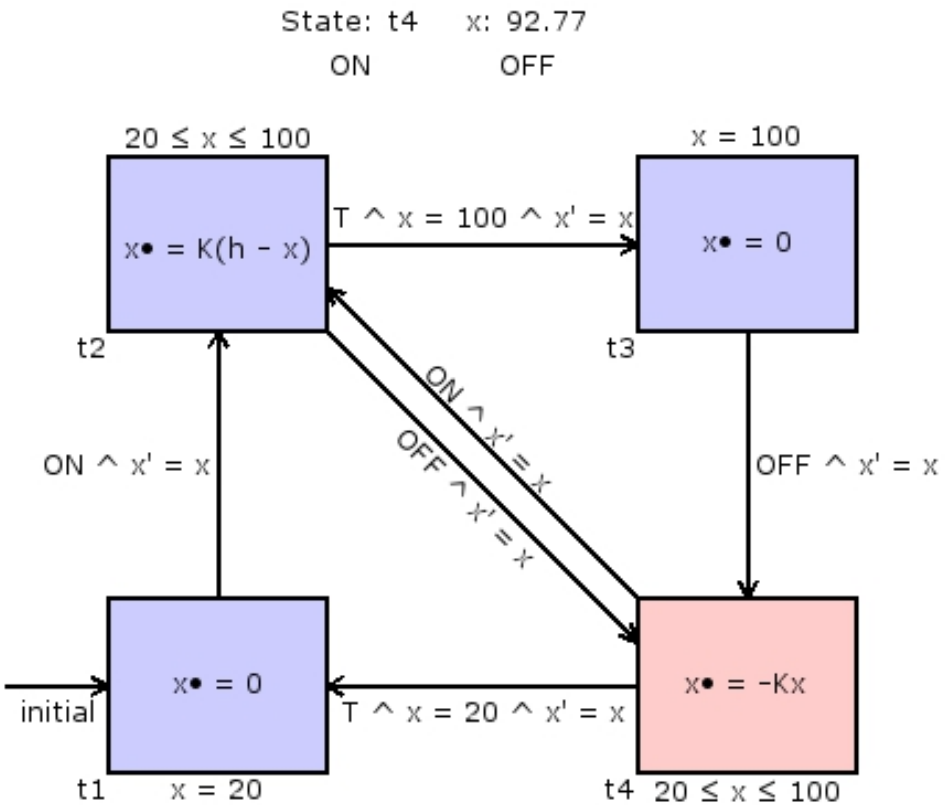


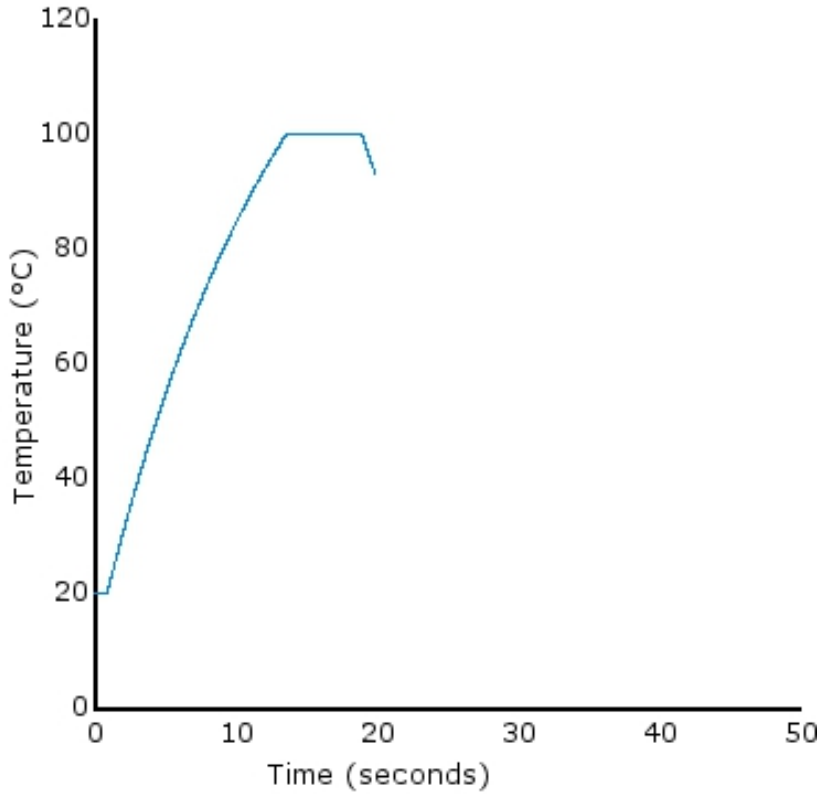
A hybrid automata example



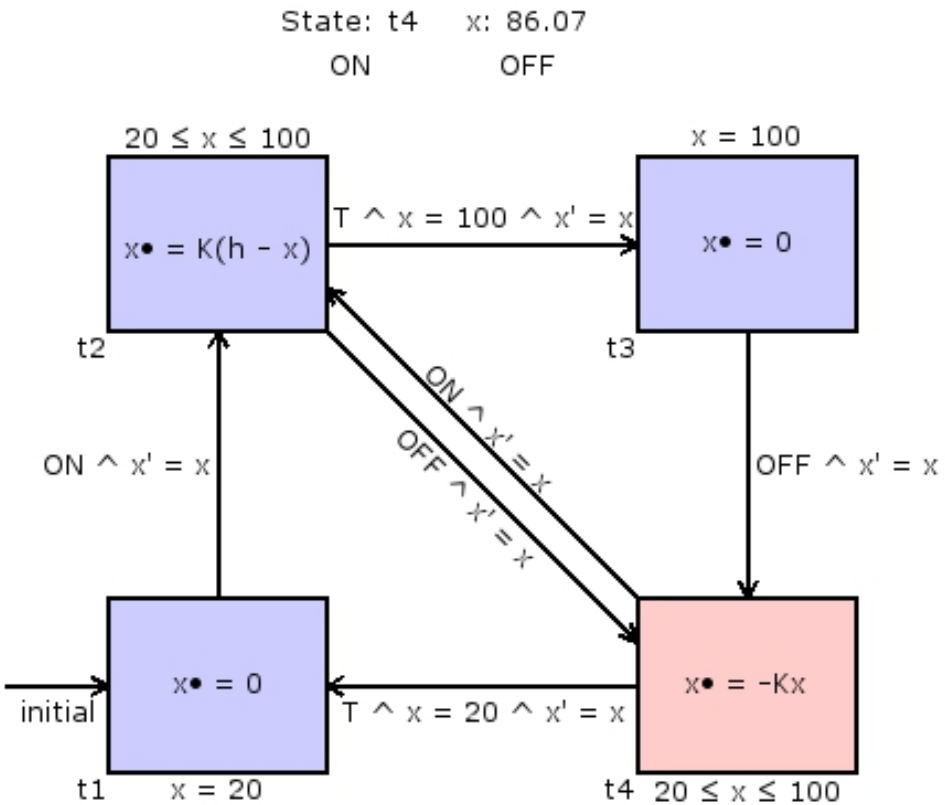


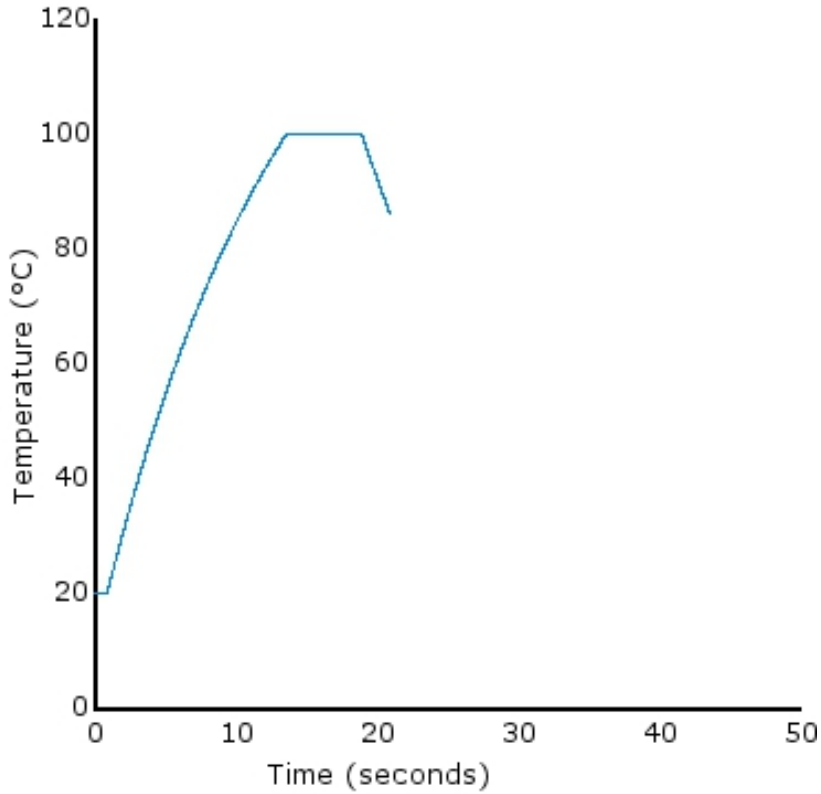
A hybrid automata example



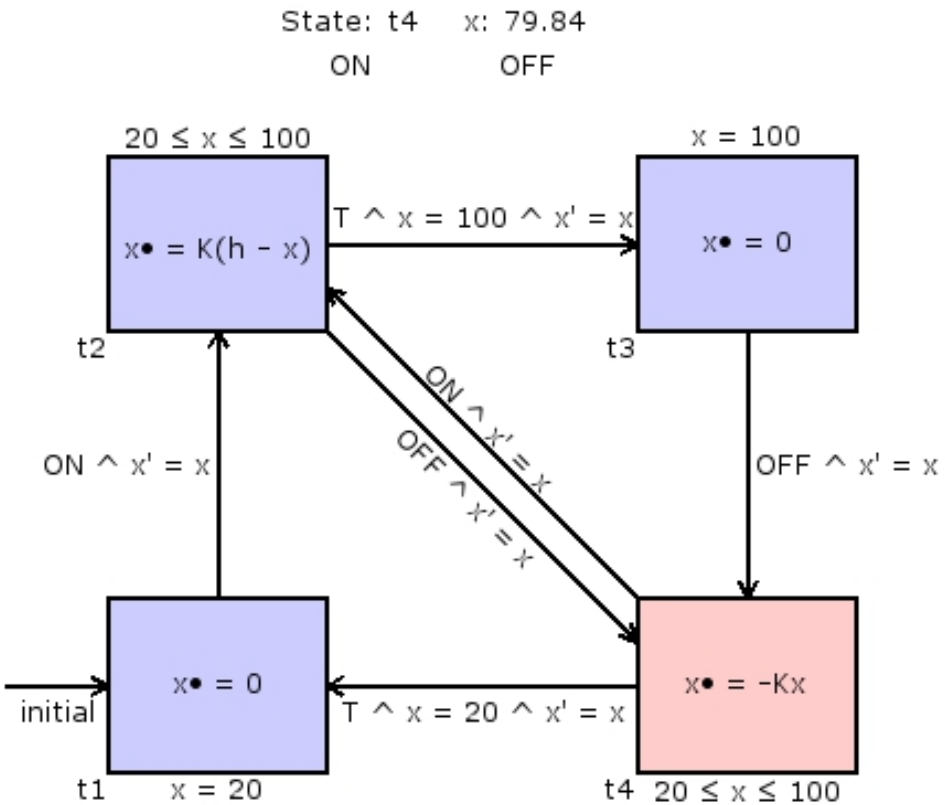


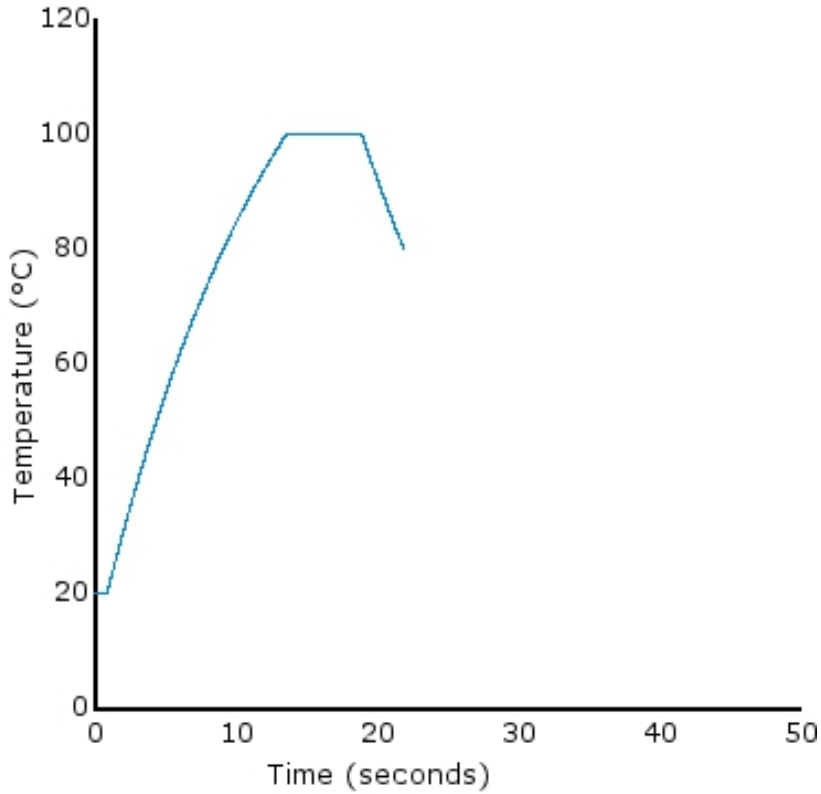
A hybrid automata example



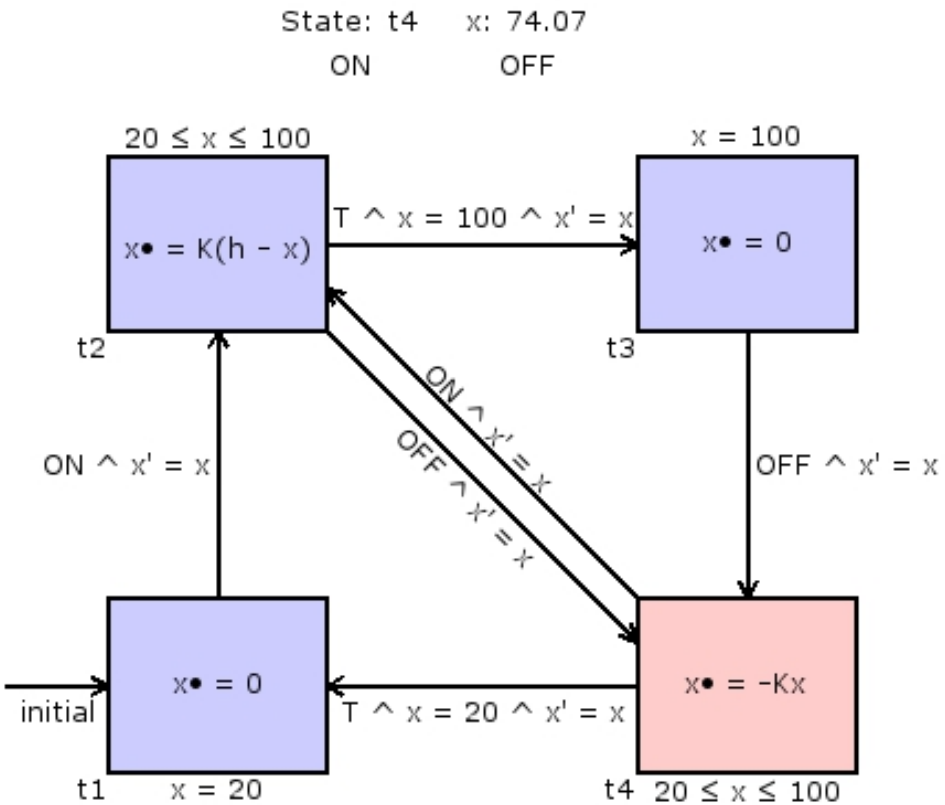


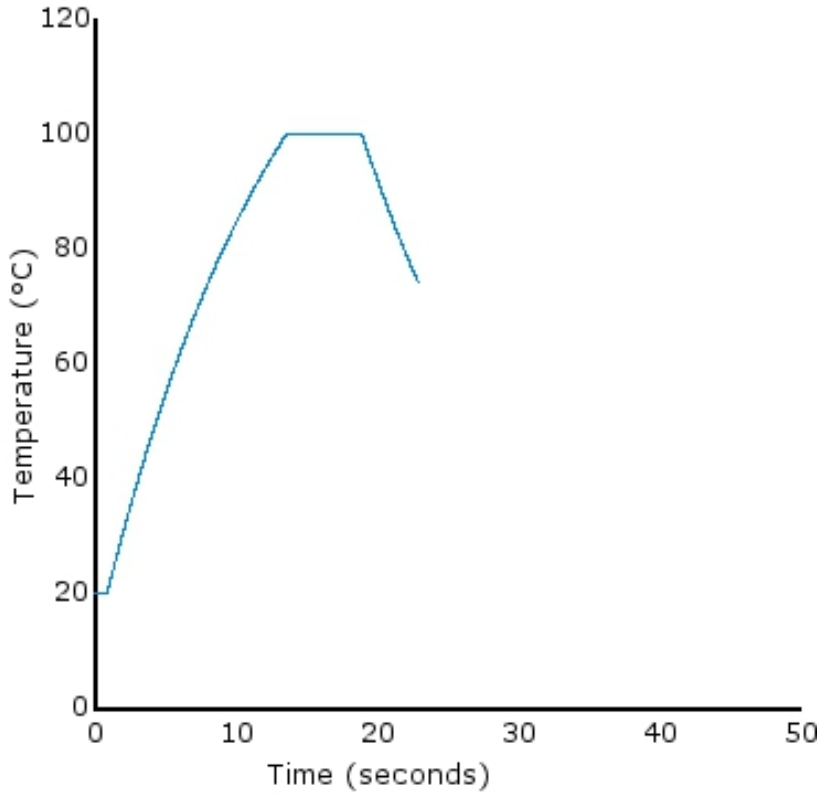
A hybrid automata example



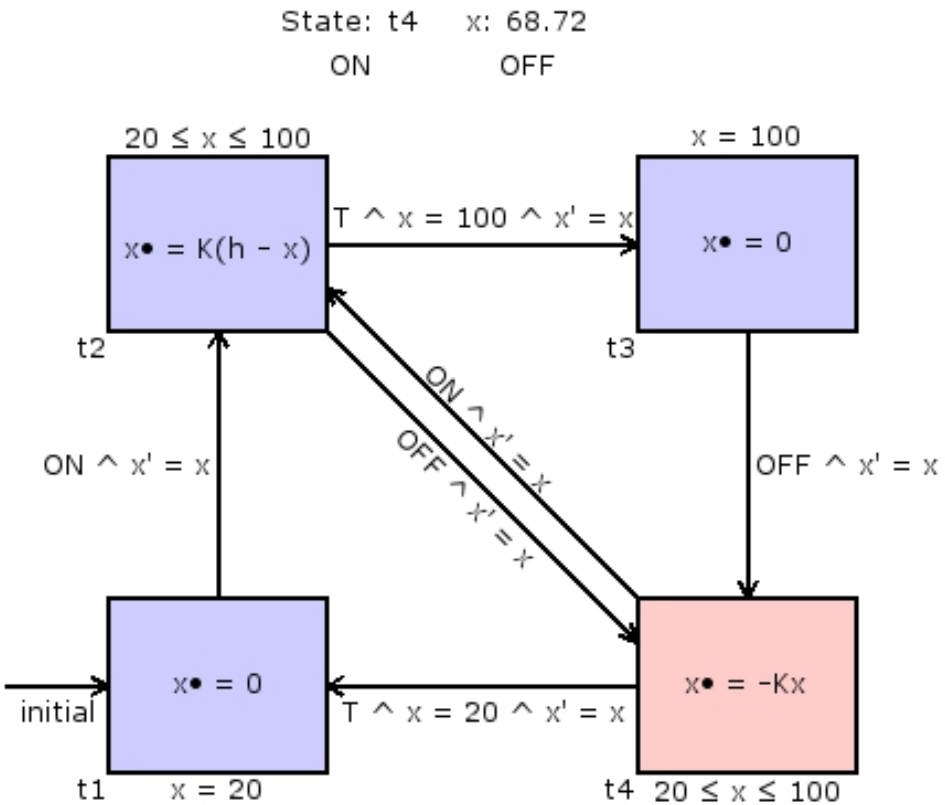


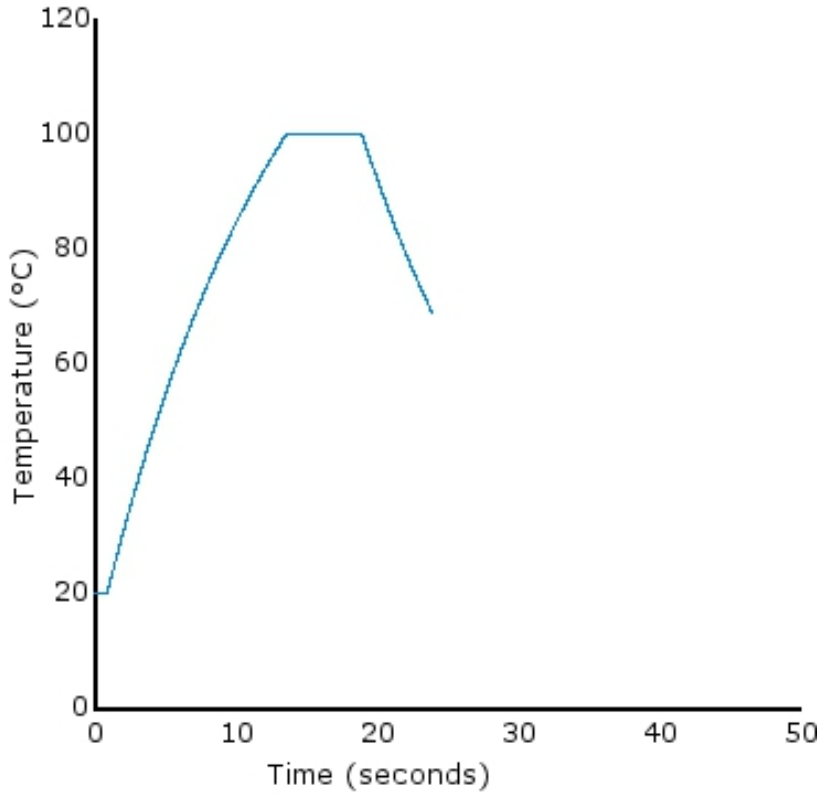
A hybrid automata example



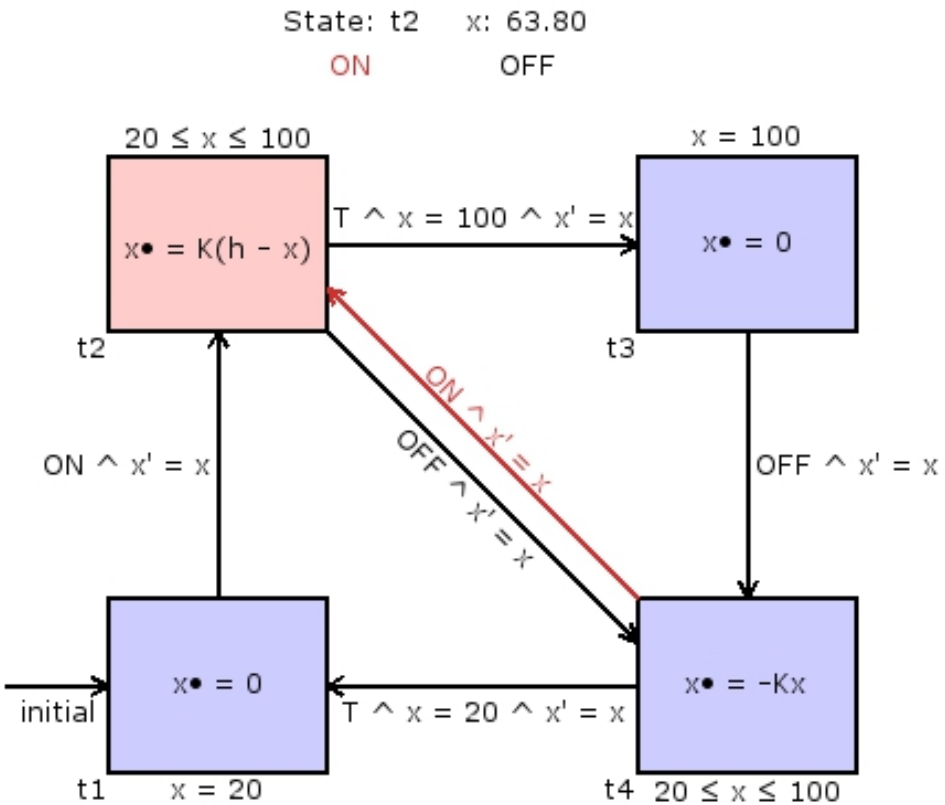


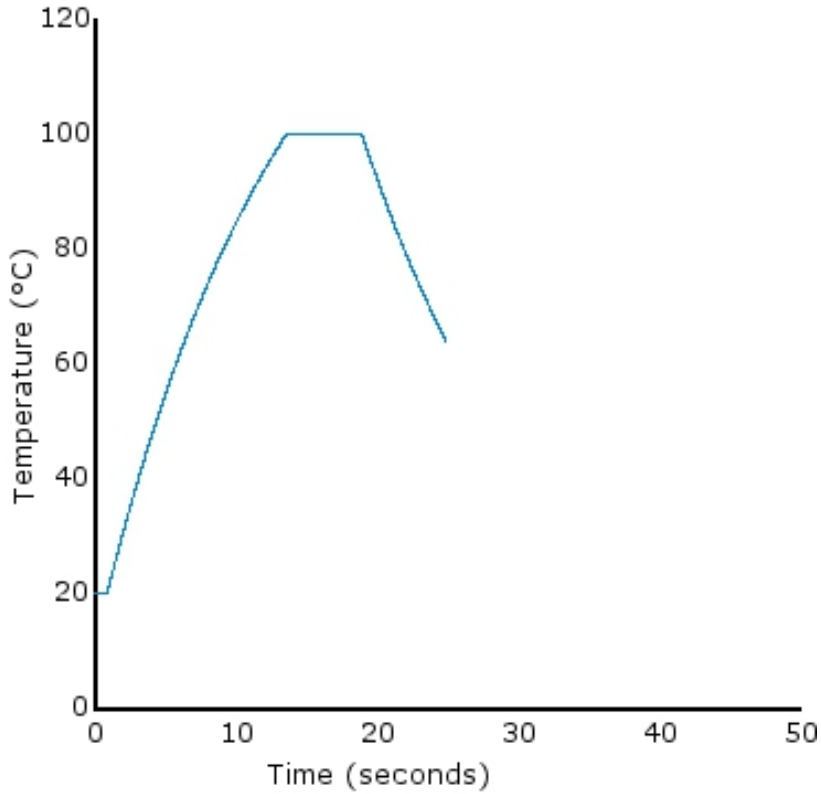
A hybrid automata example



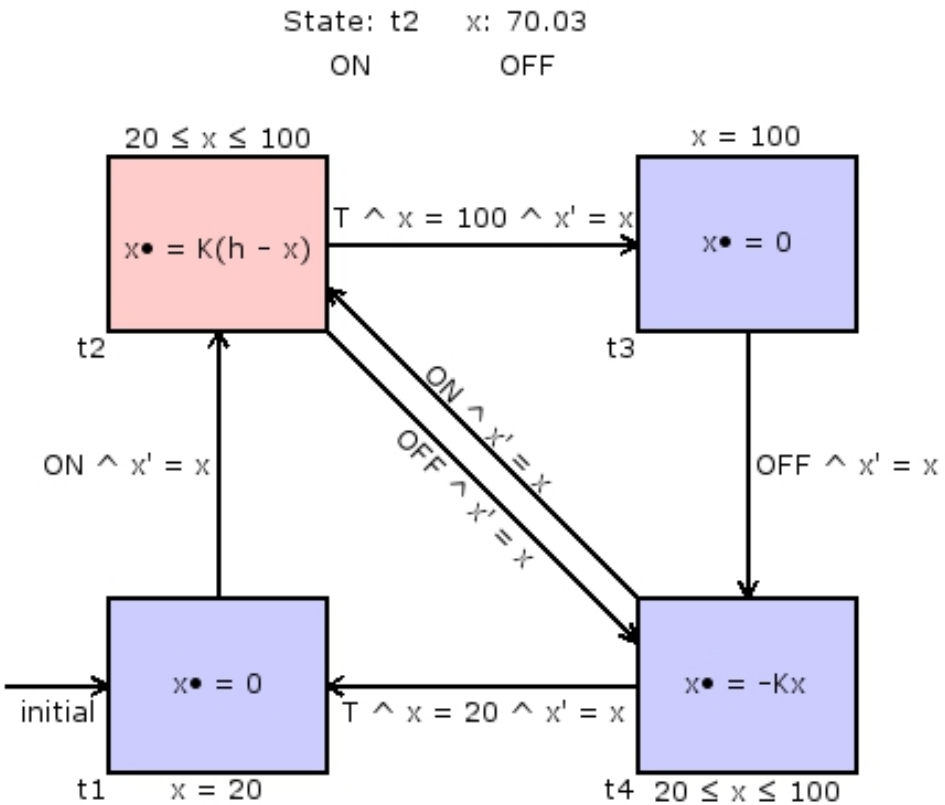


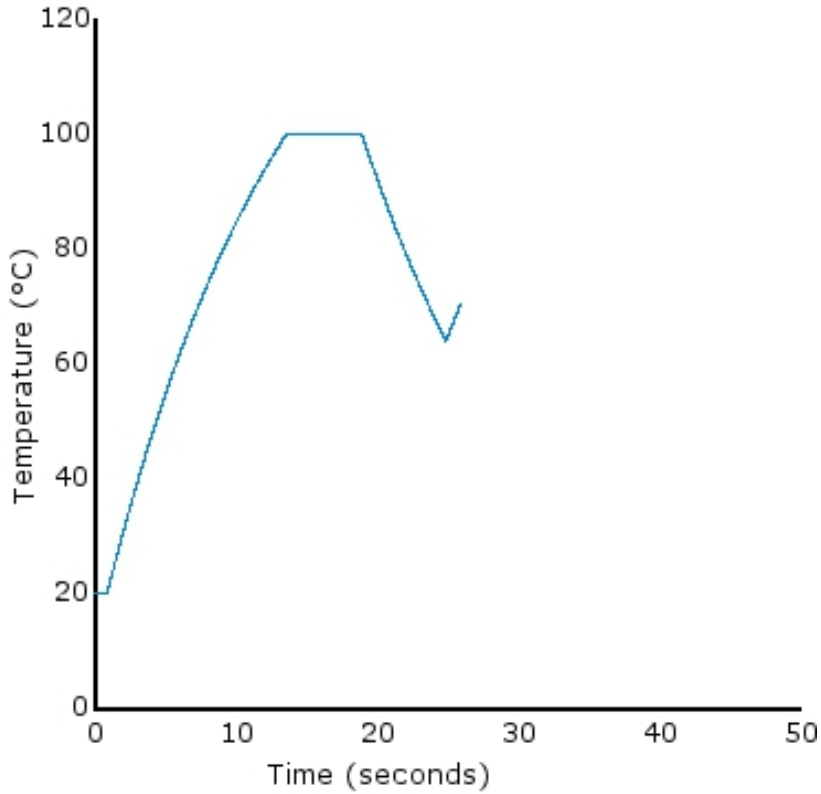
A hybrid automata example



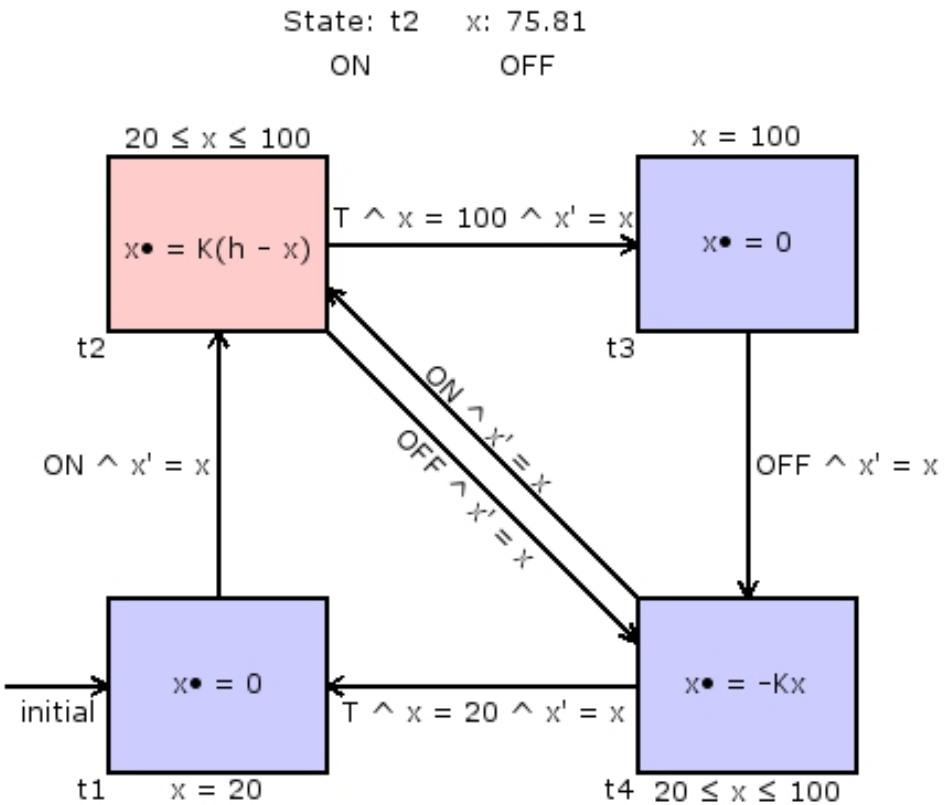


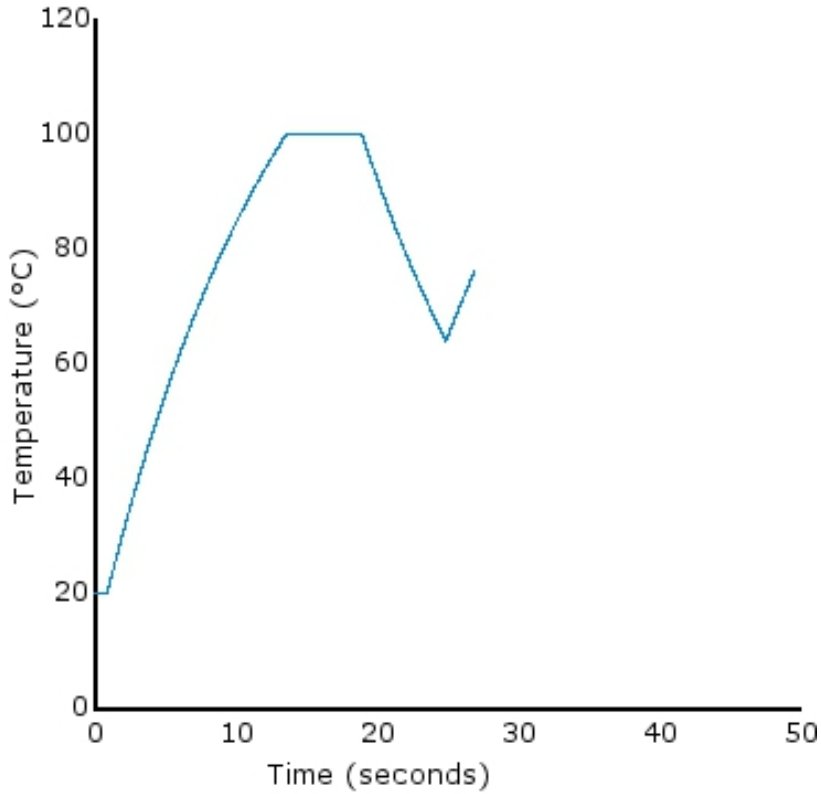
A hybrid automata example



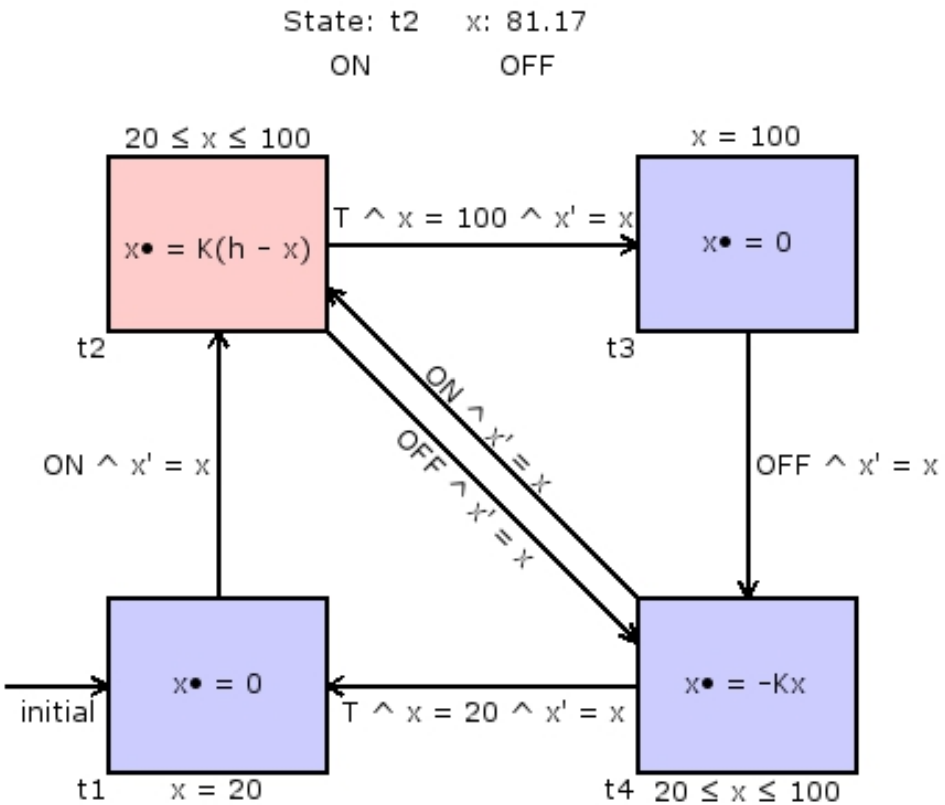


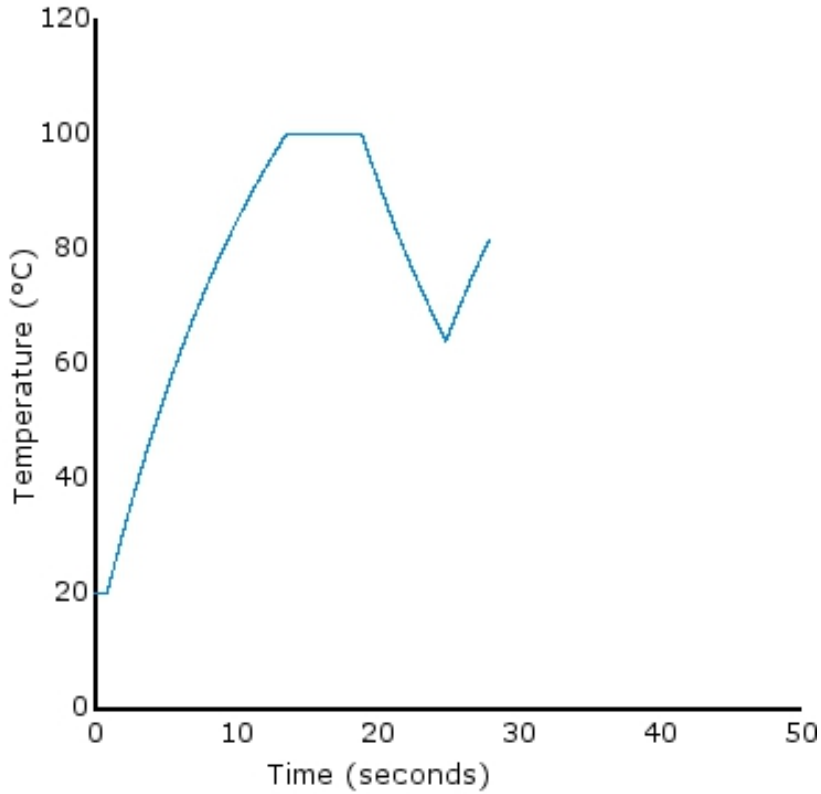
A hybrid automata example



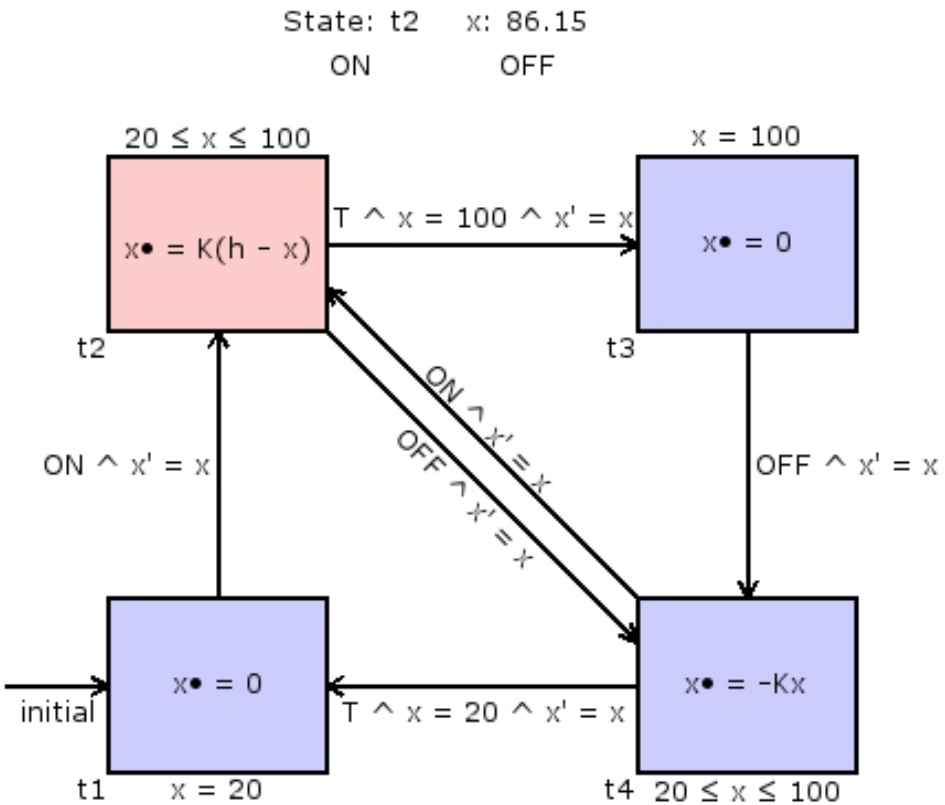


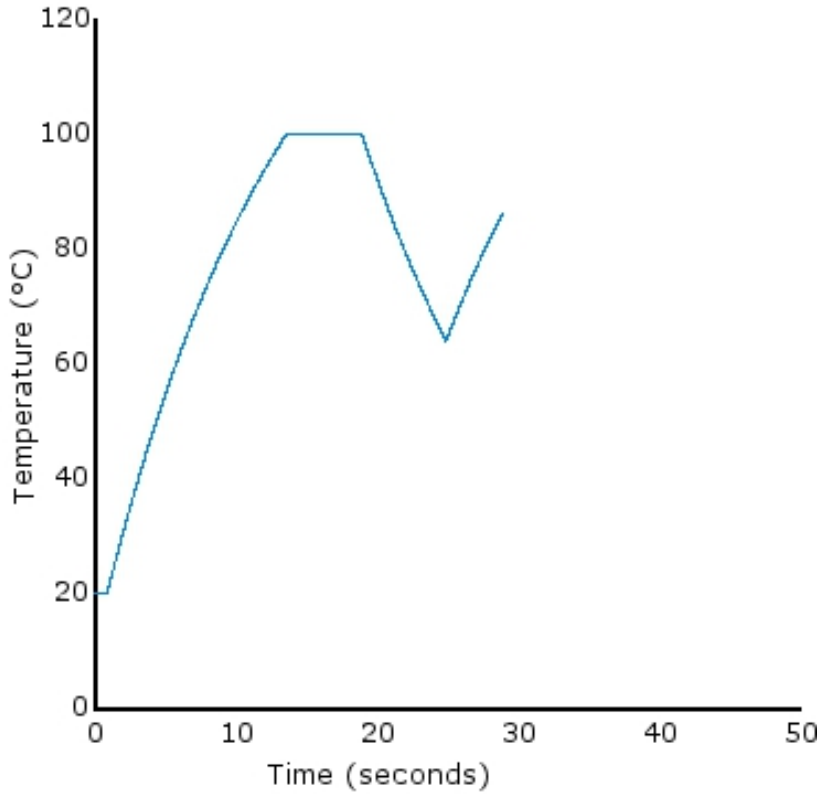
A hybrid automata example



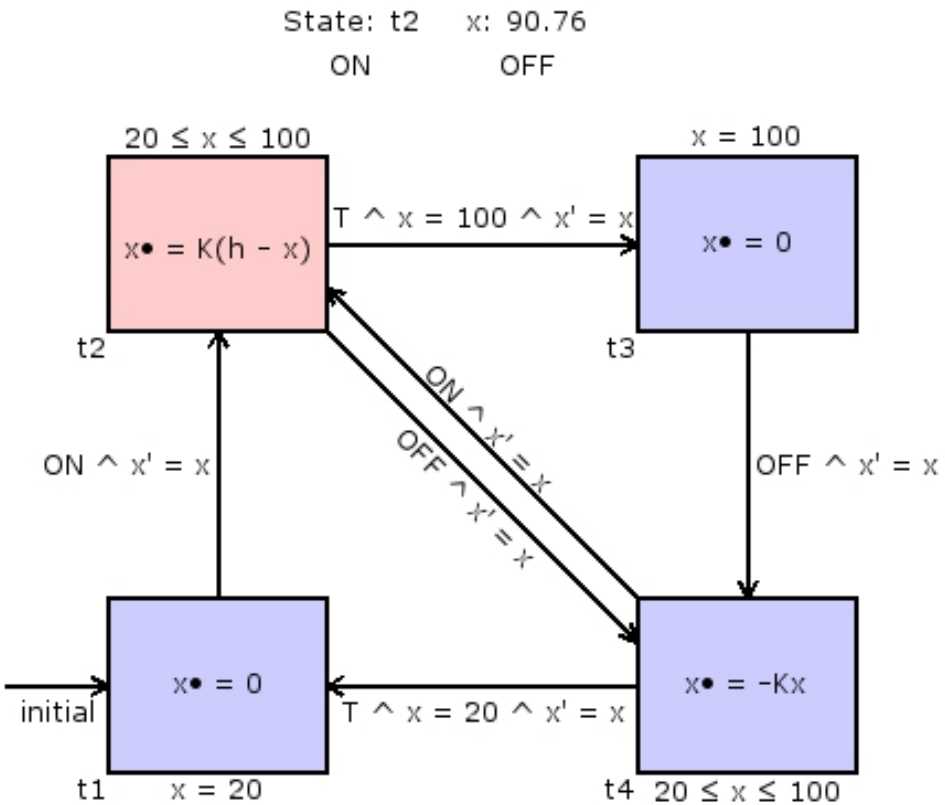


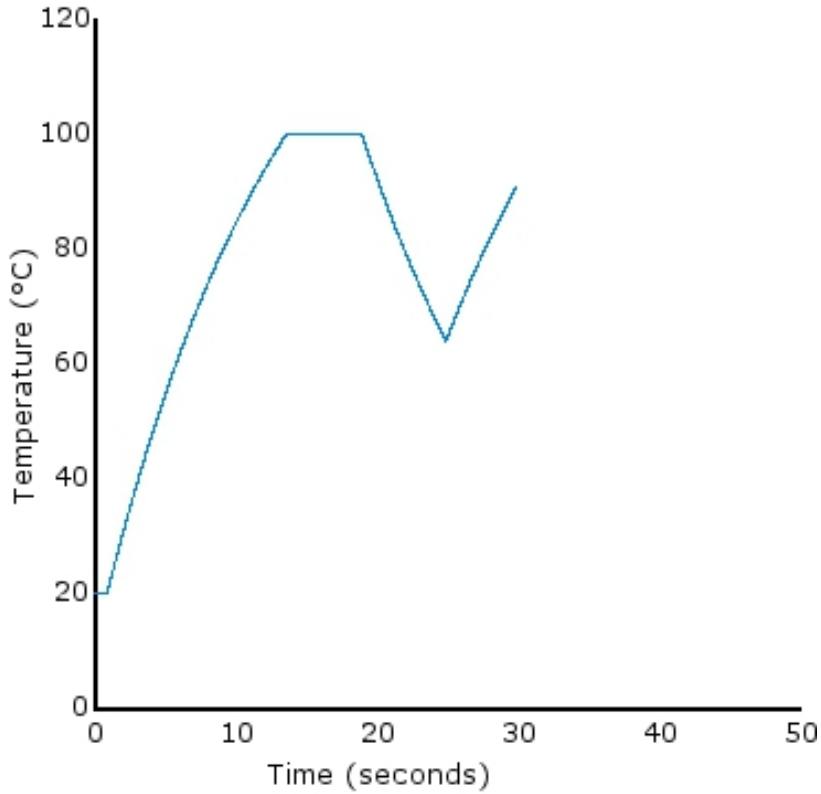
A hybrid automata example



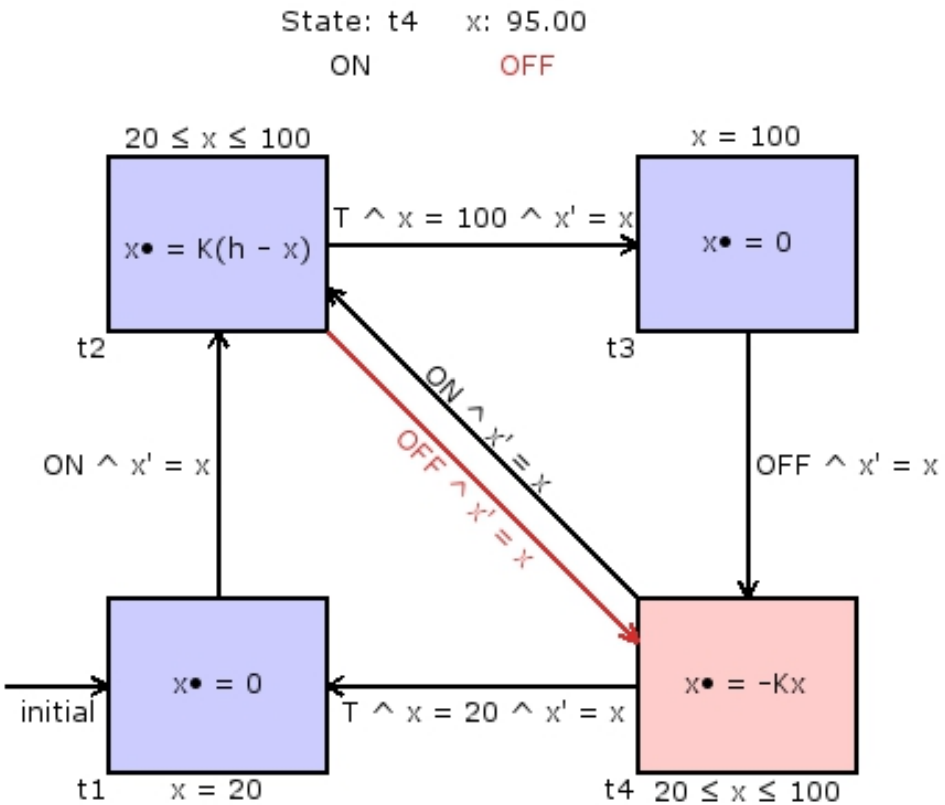


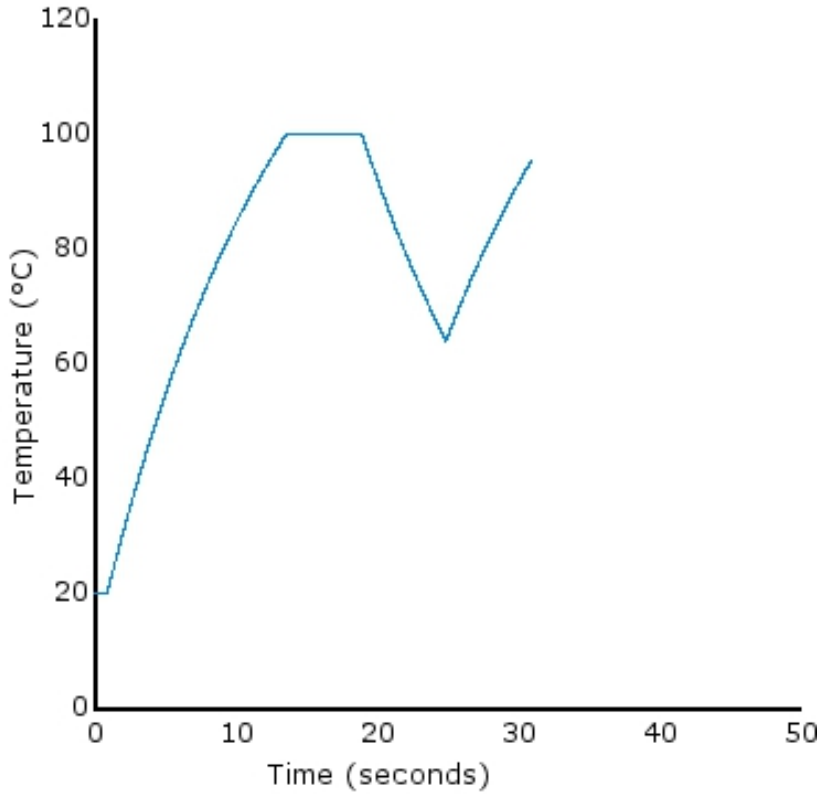
A hybrid automata example



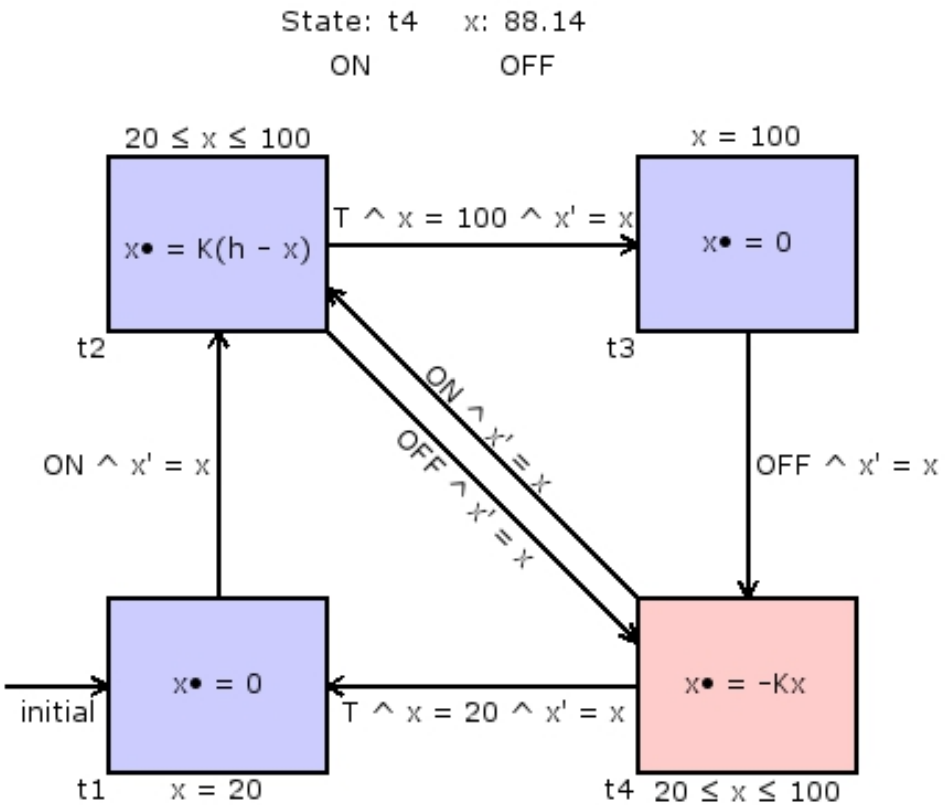


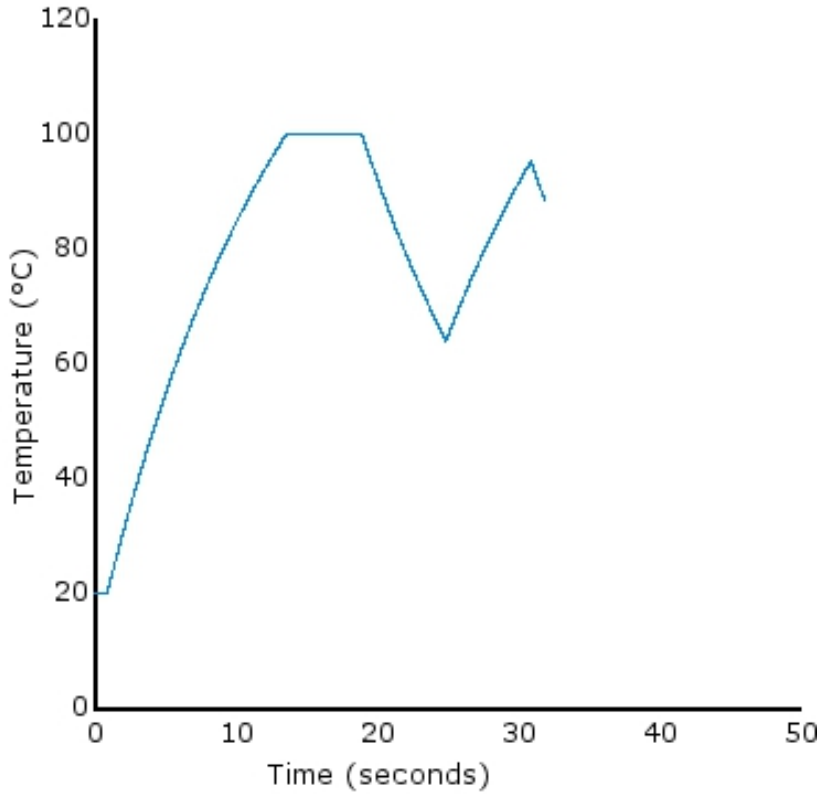
A hybrid automata example



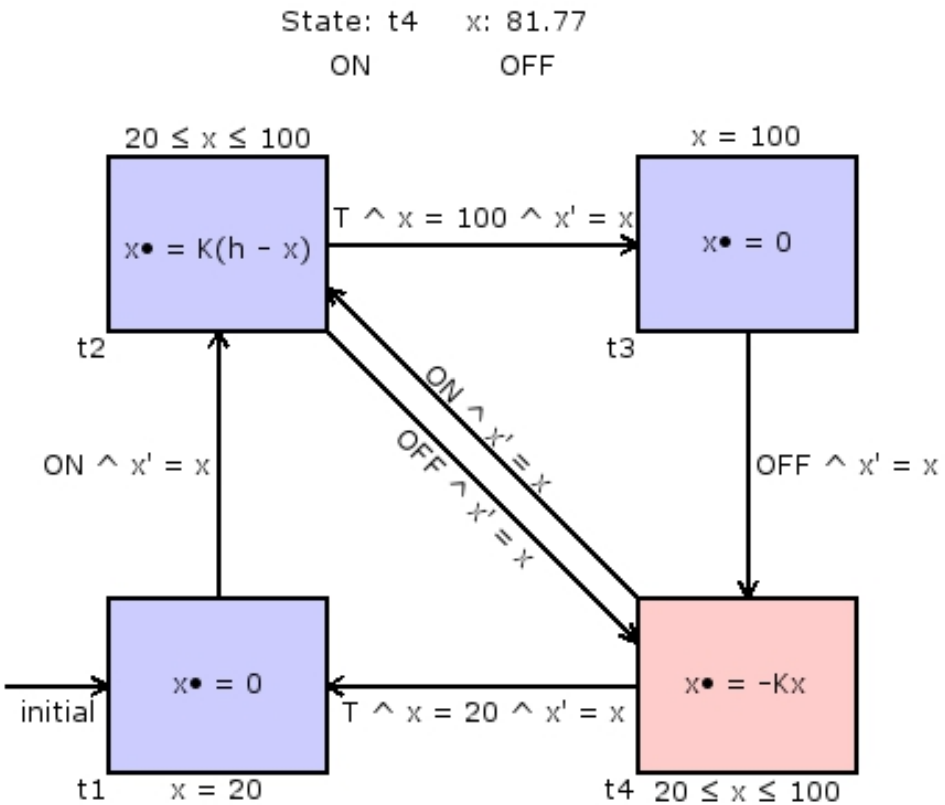


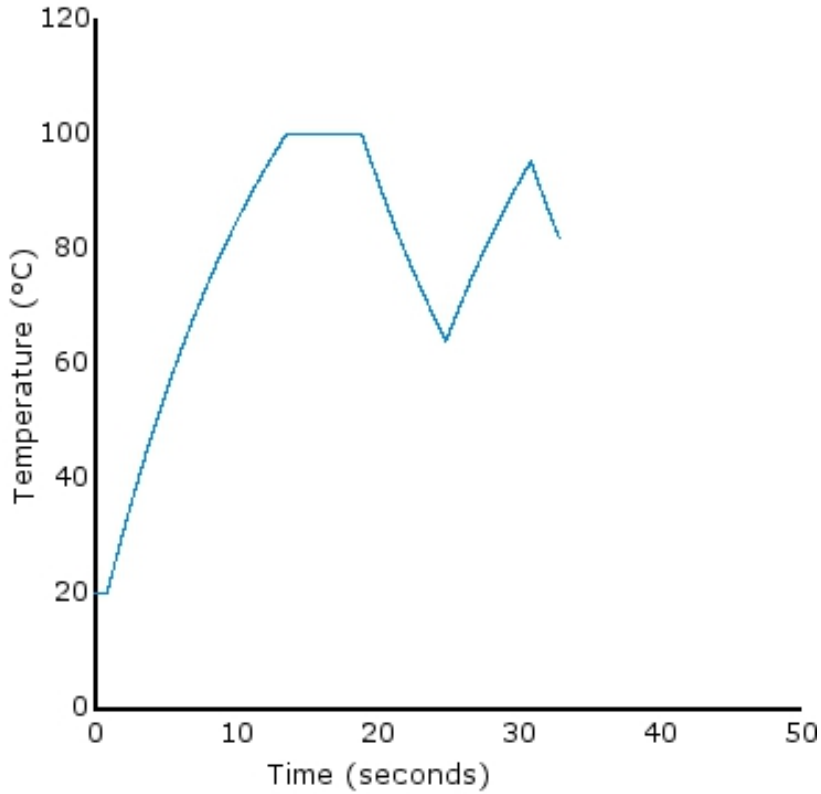
A hybrid automata example



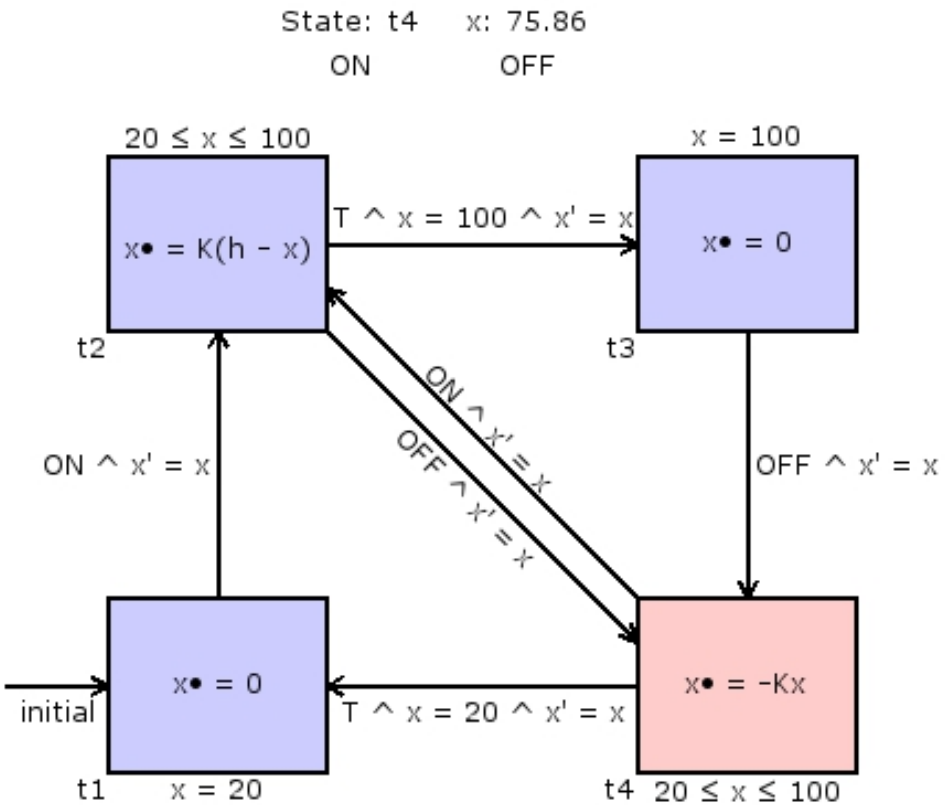


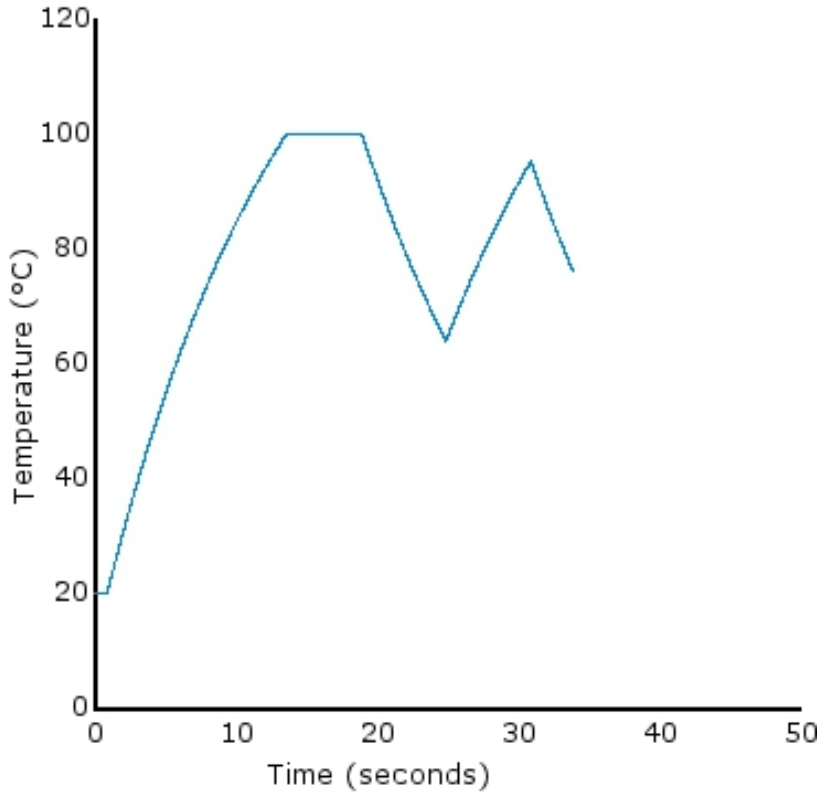
A hybrid automata example



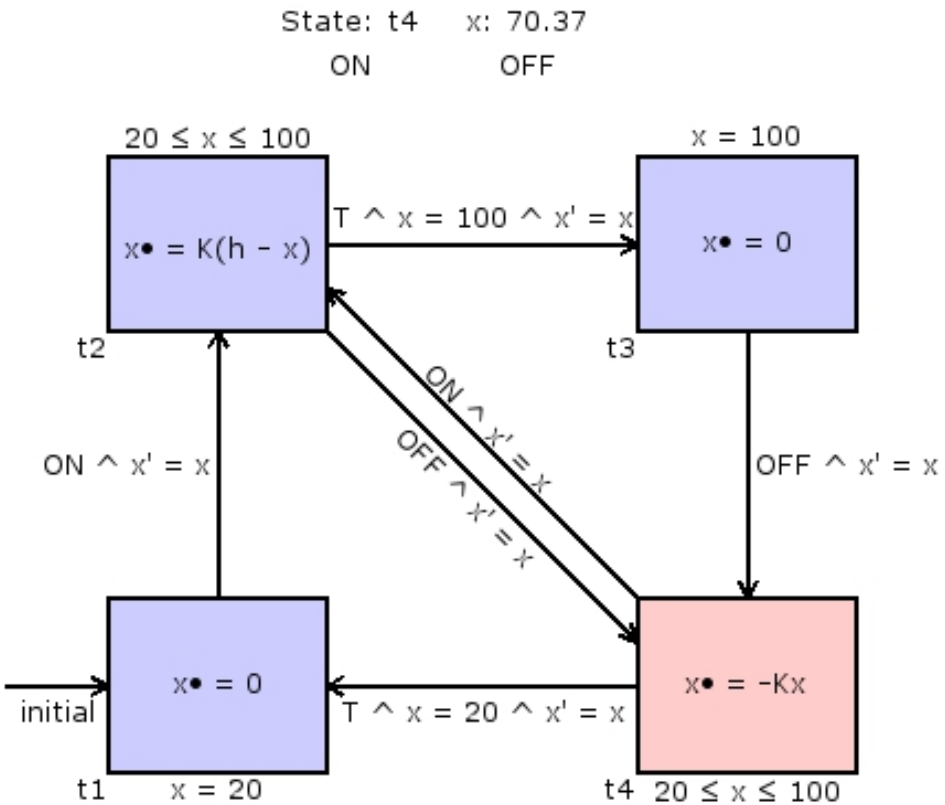


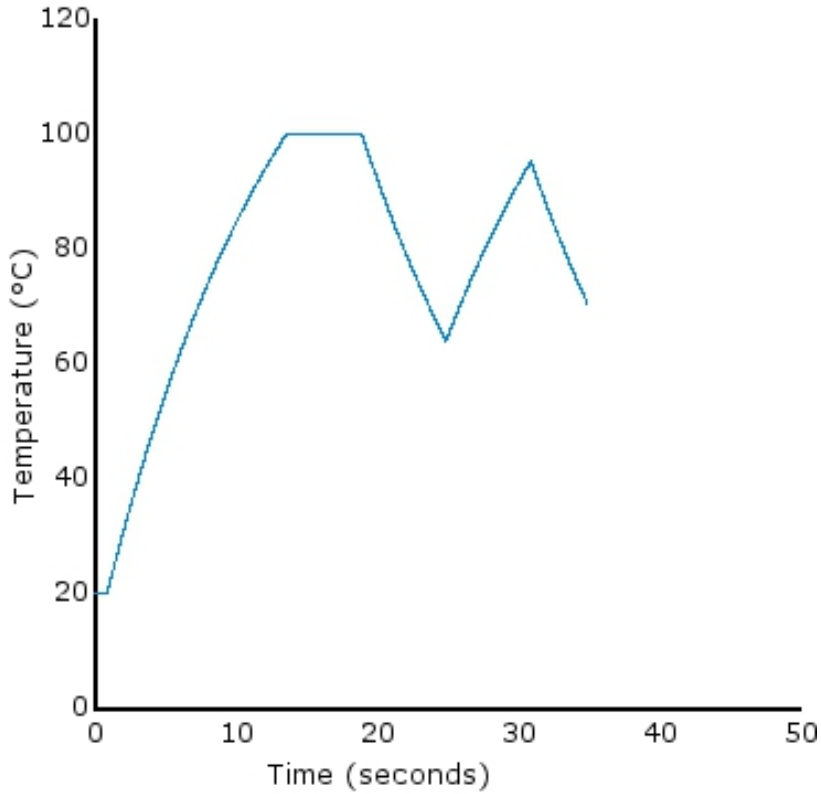
A hybrid automata example



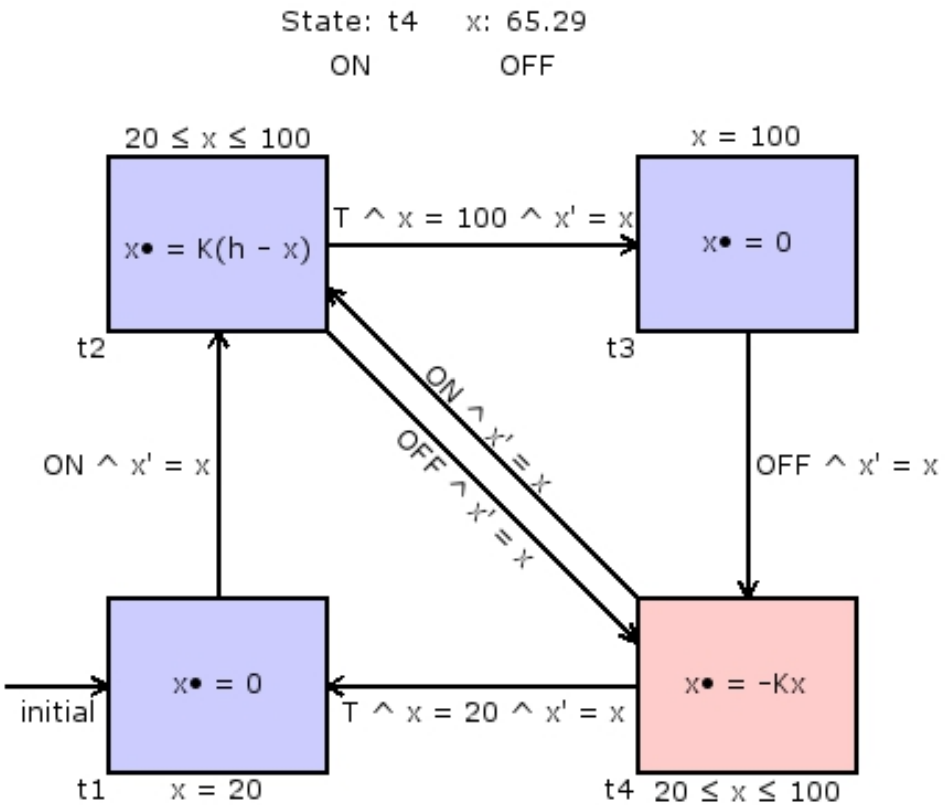


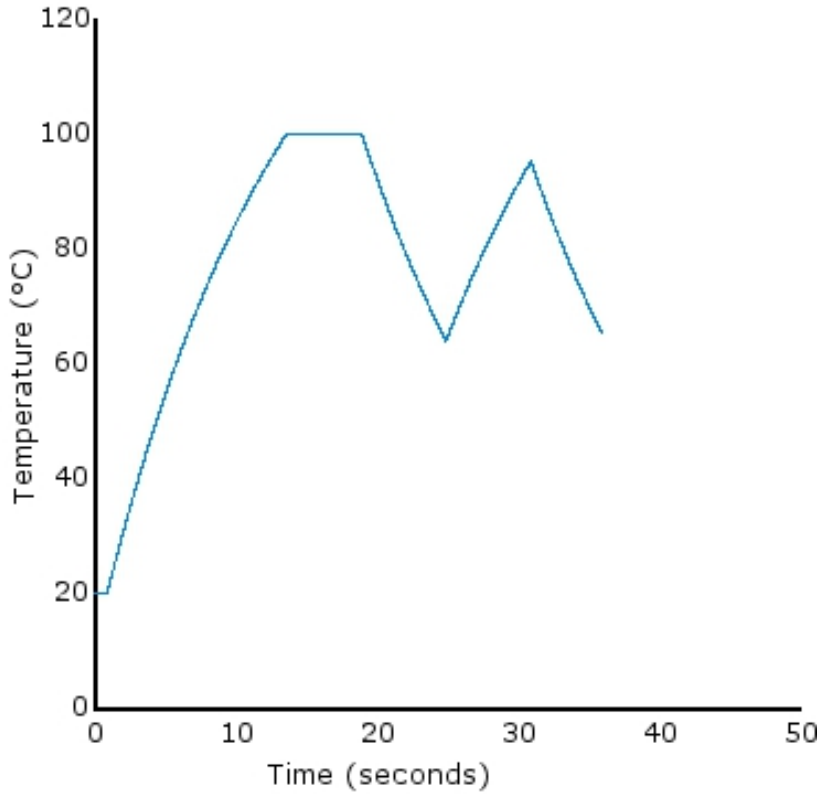
A hybrid automata example



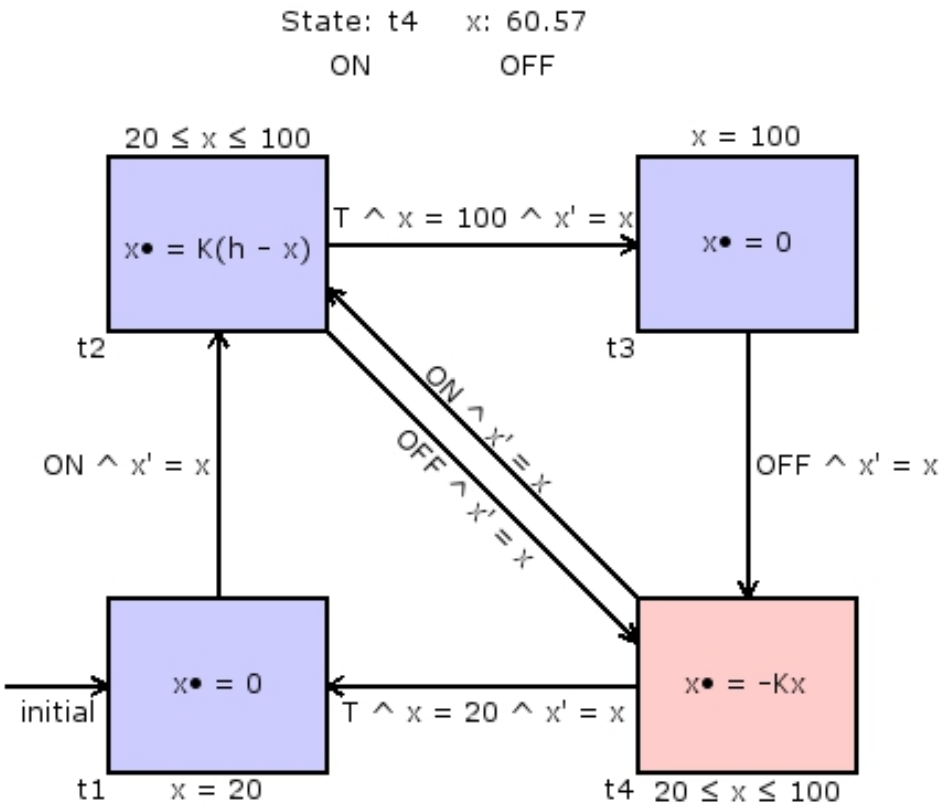


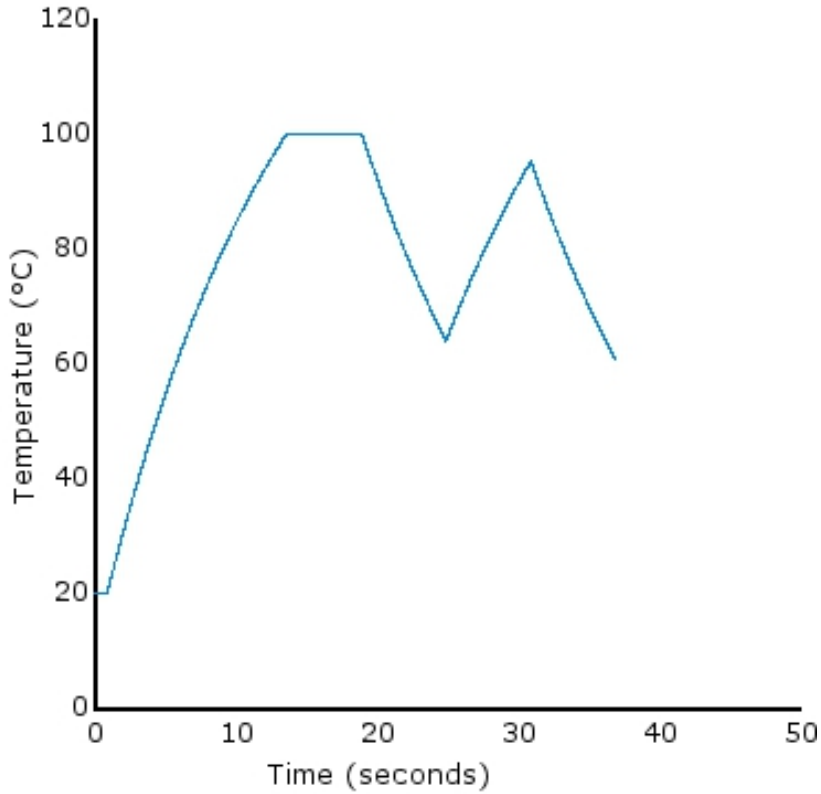
A hybrid automata example



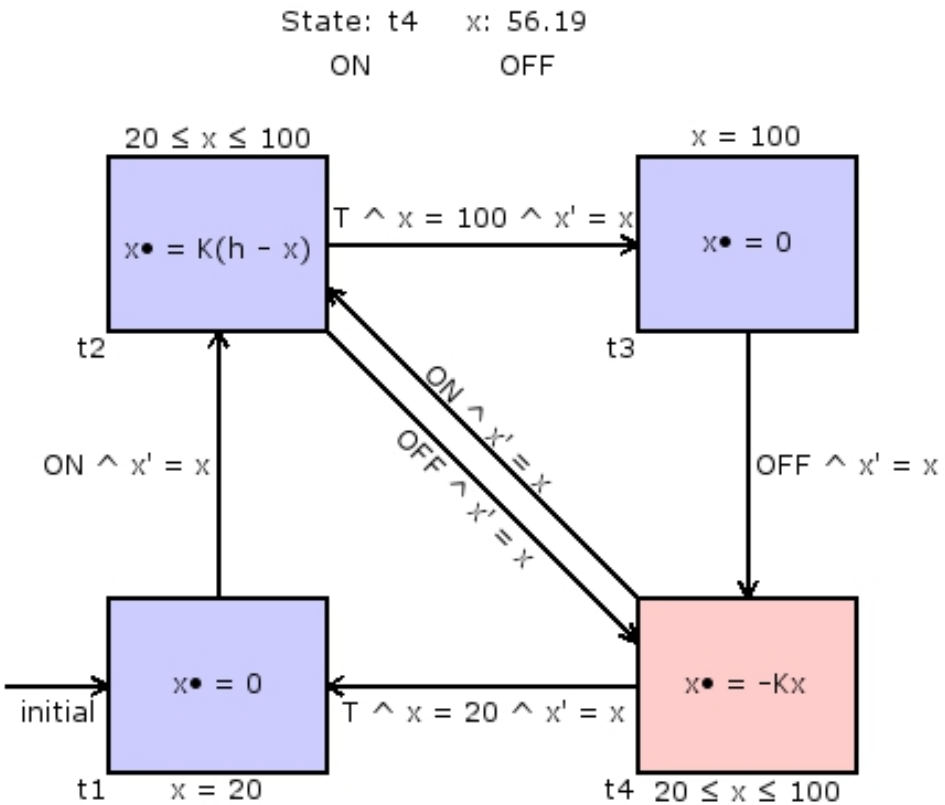


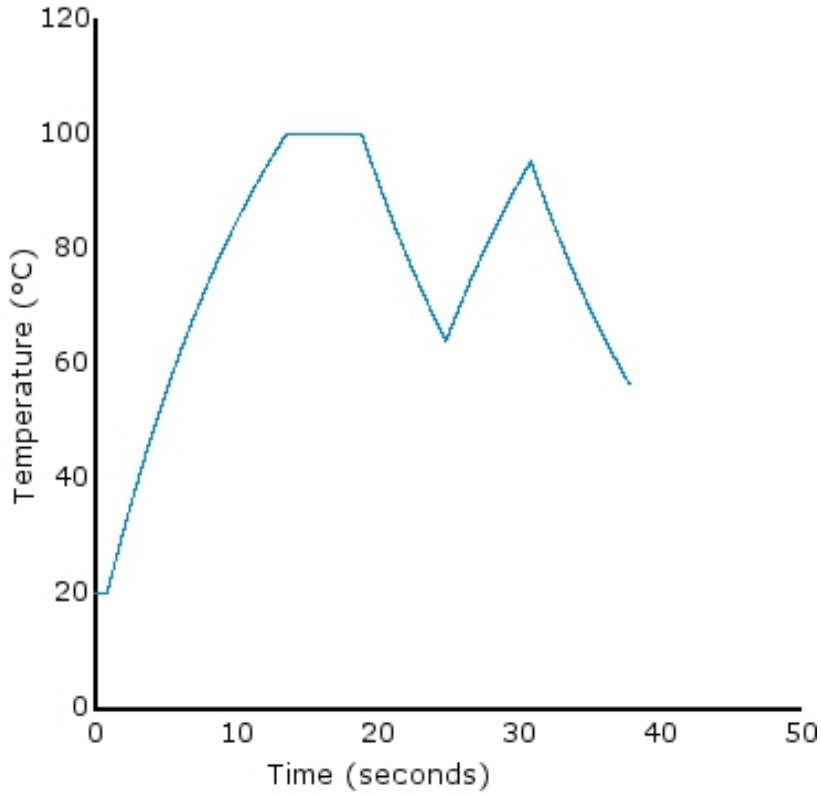
A hybrid automata example



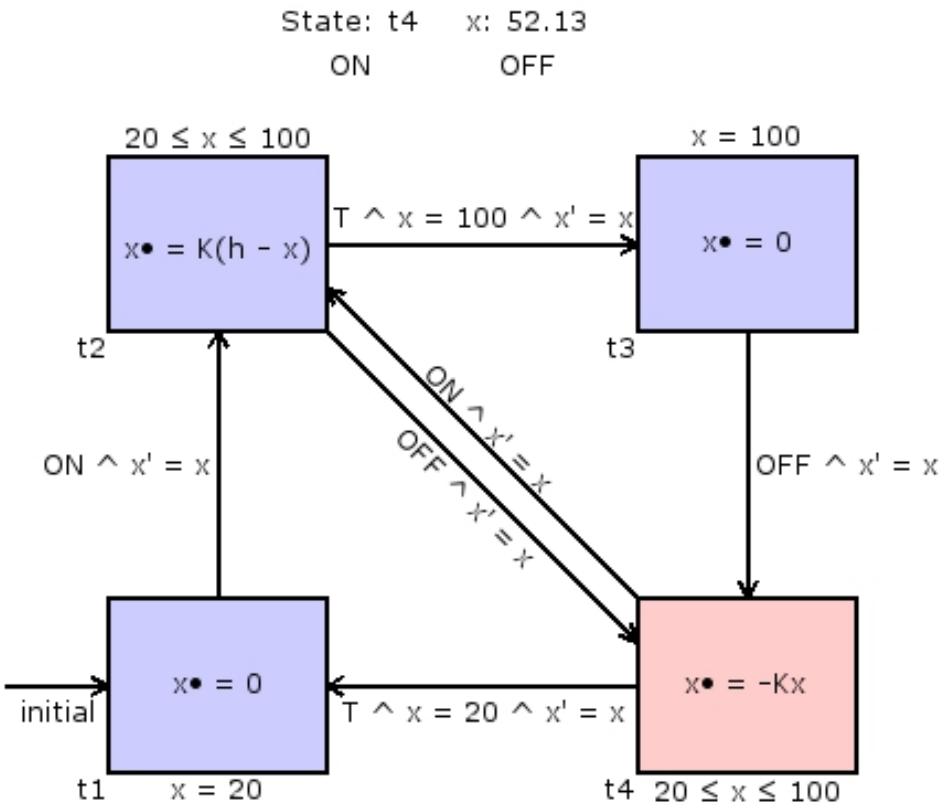


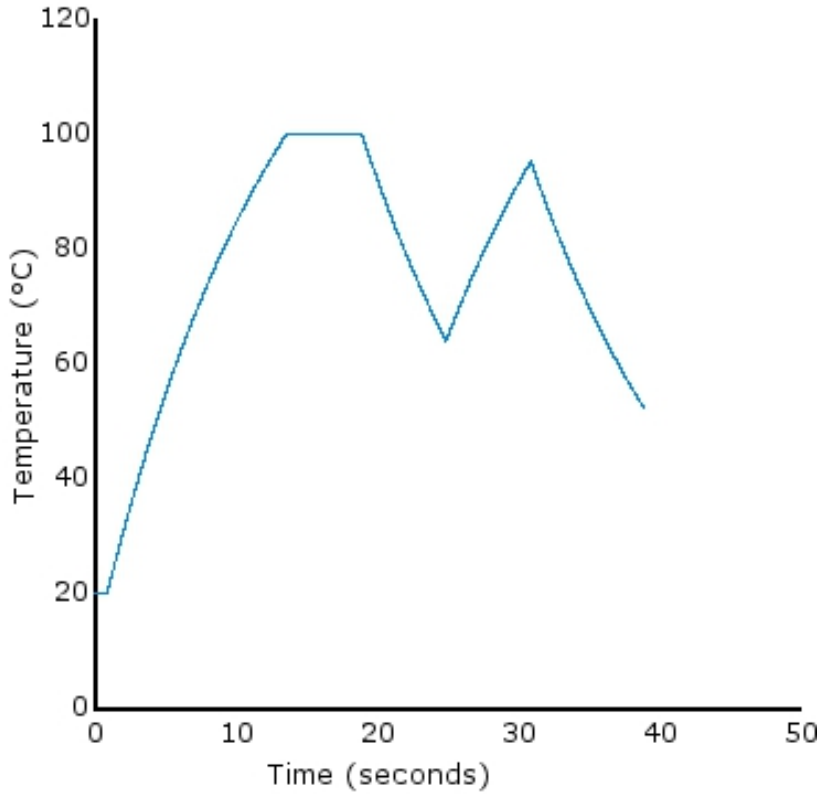
A hybrid automata example



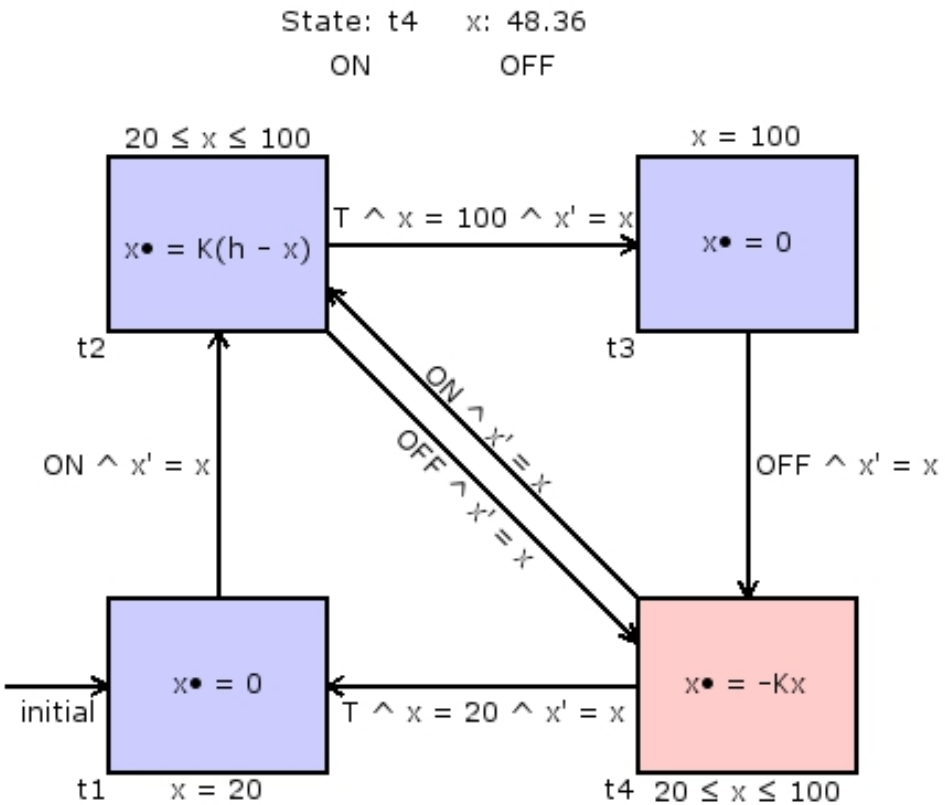


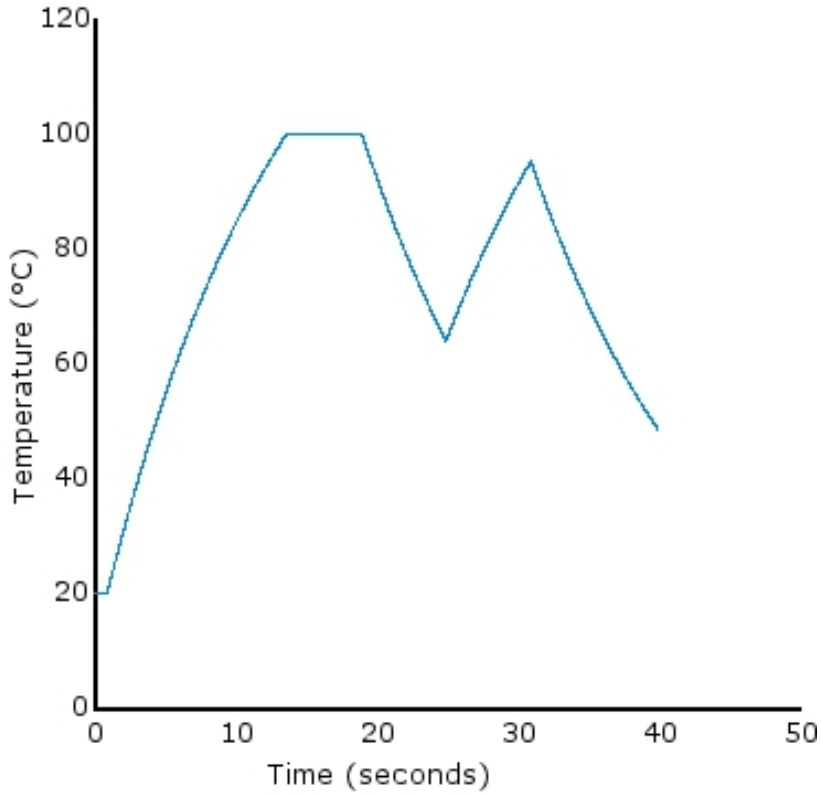
A hybrid automata example



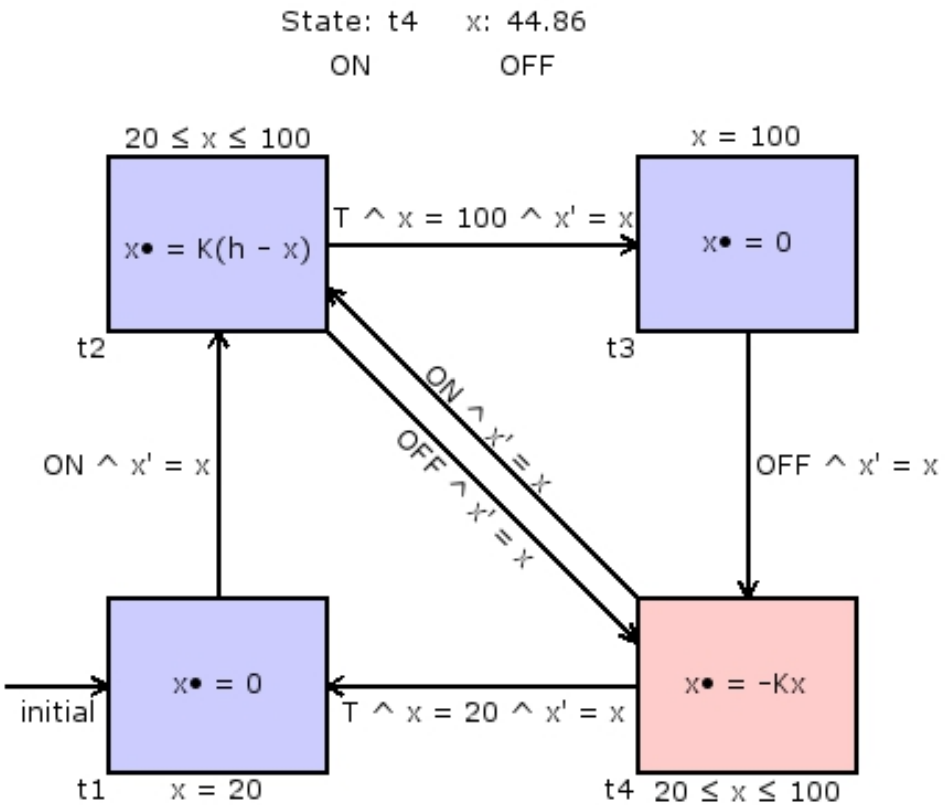


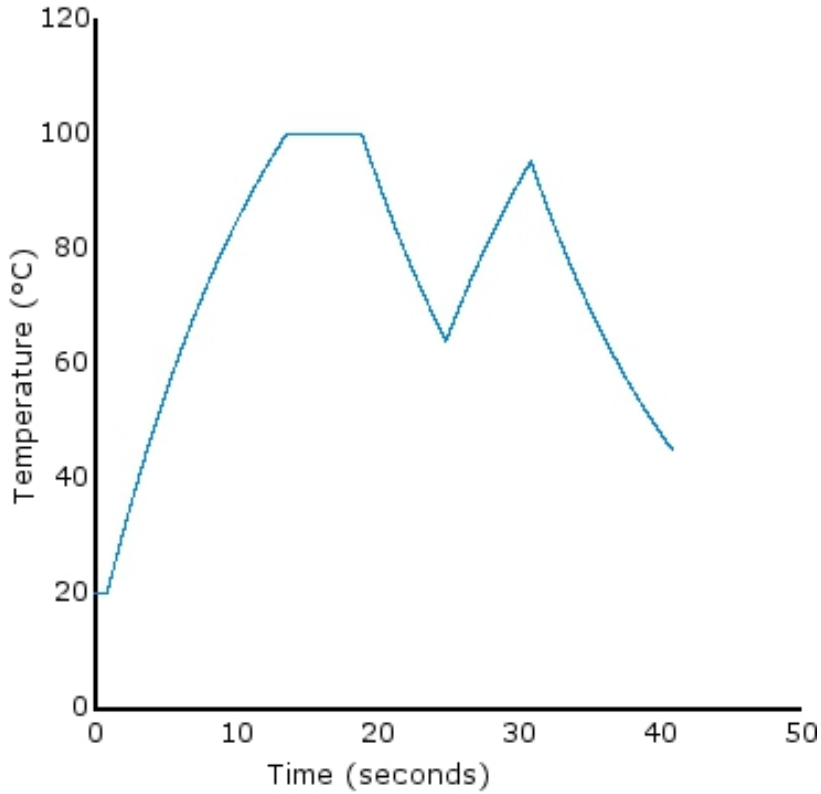
A hybrid automata example



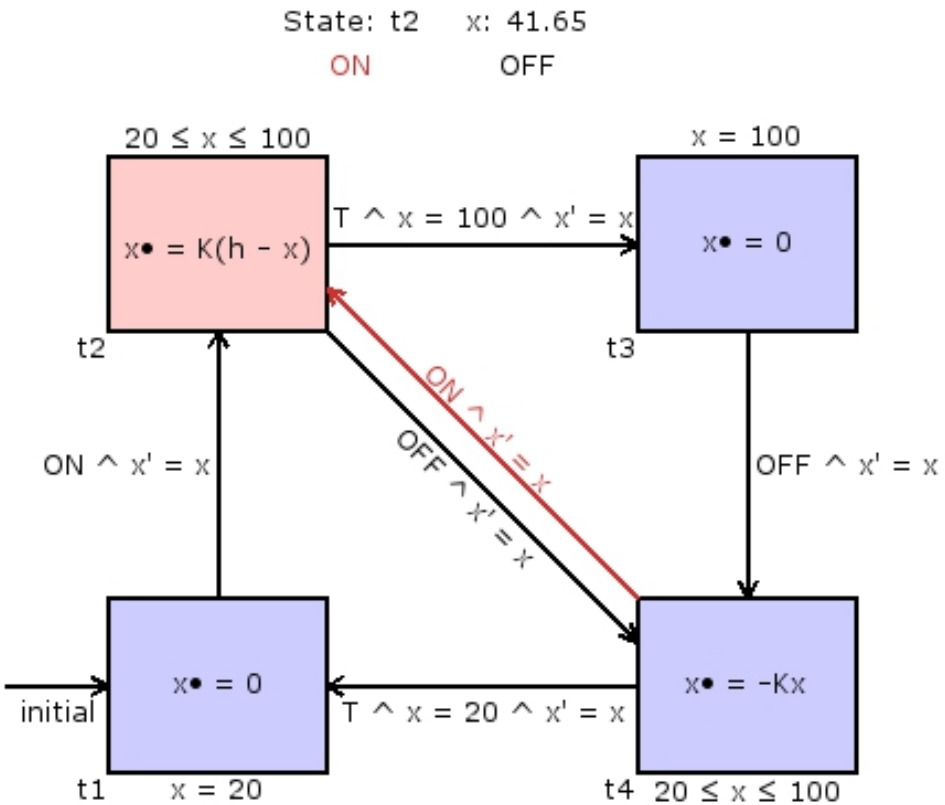


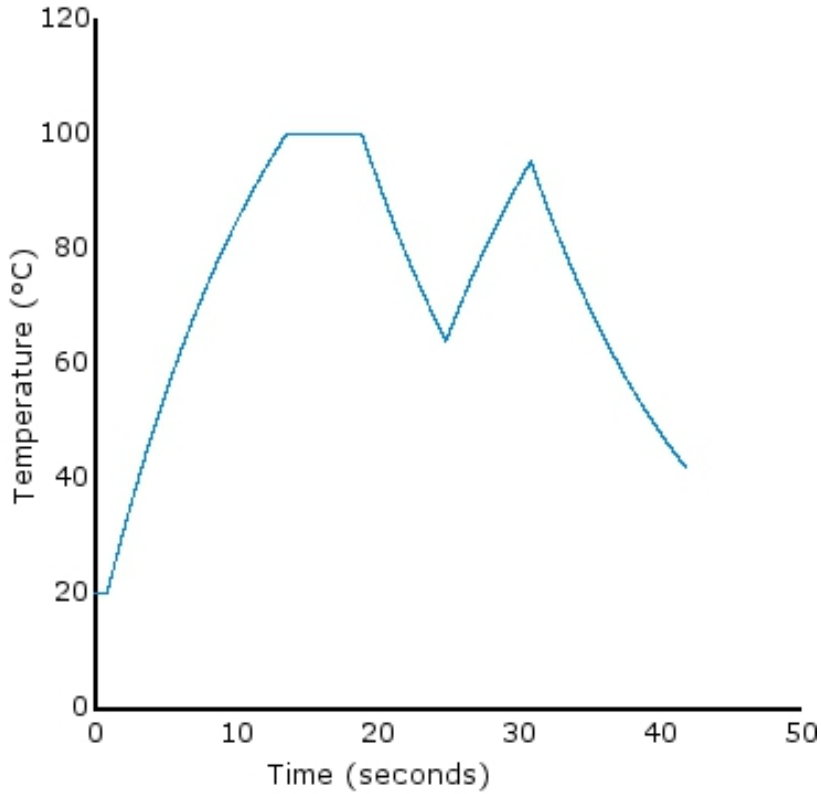
A hybrid automata example



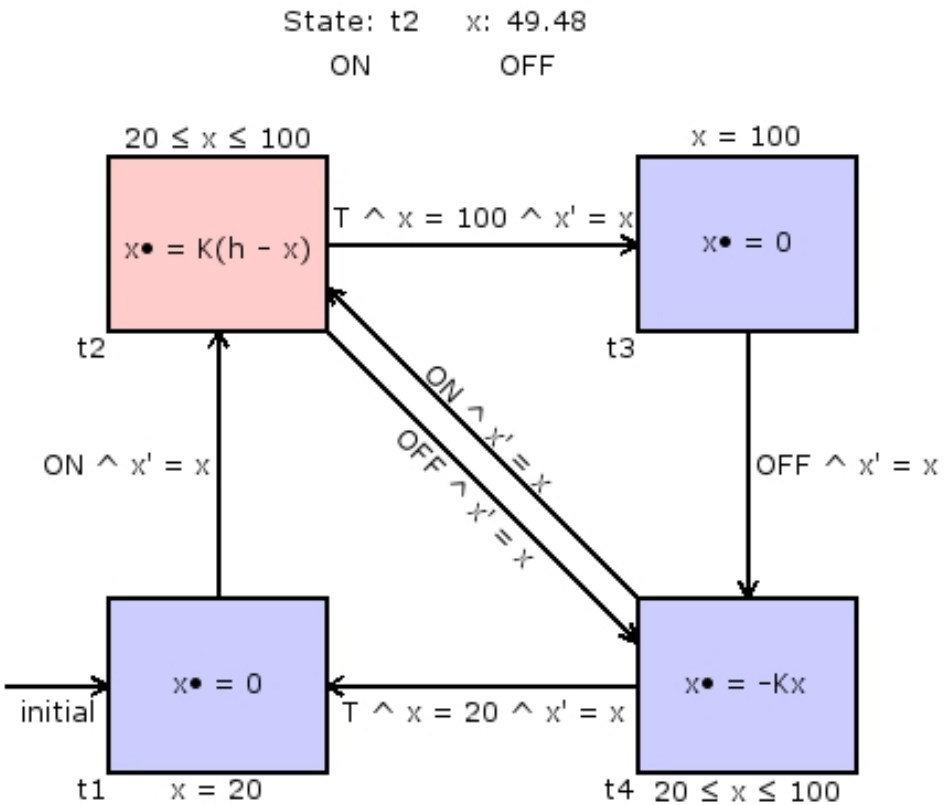


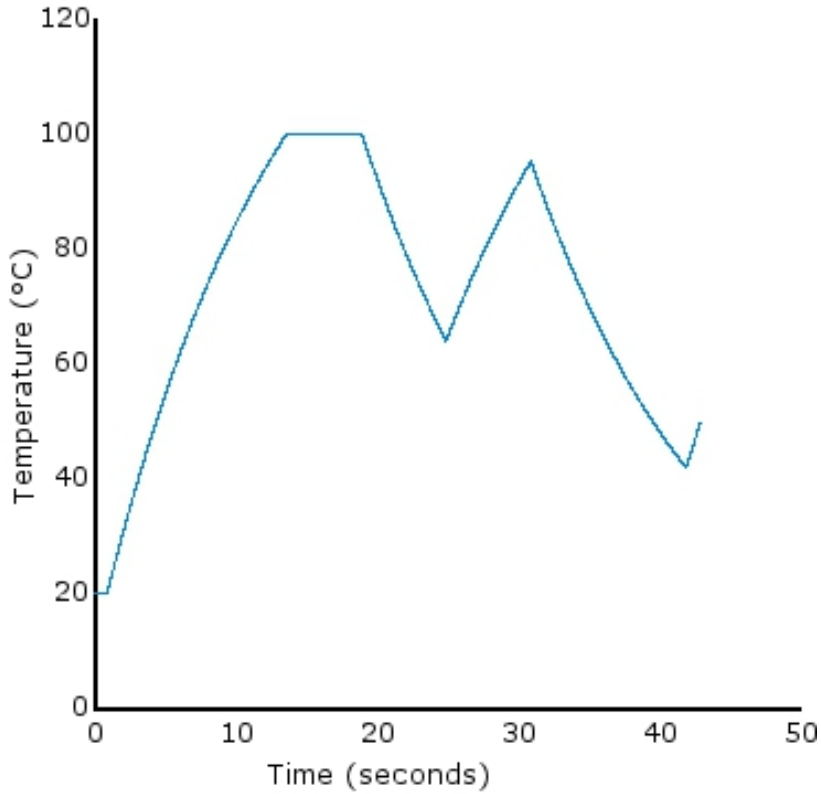
A hybrid automata example



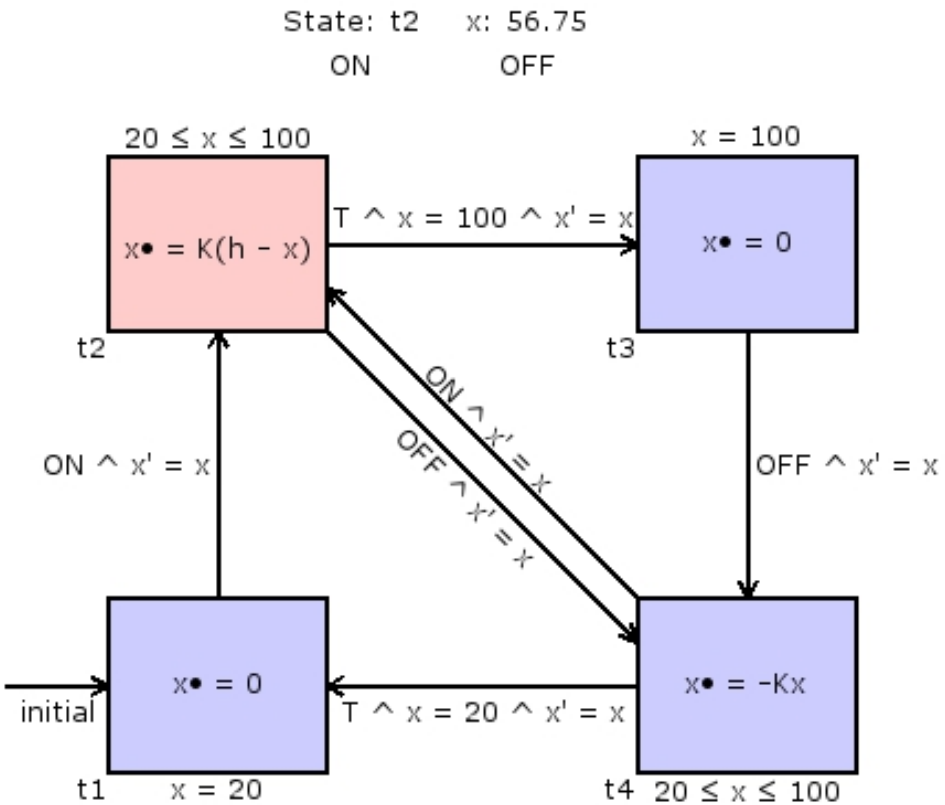


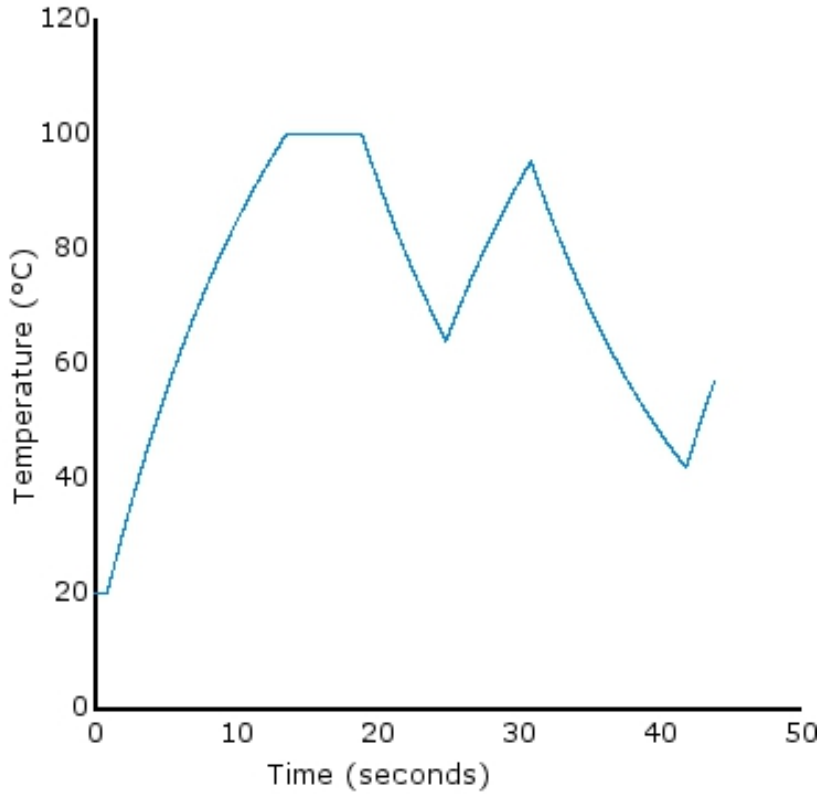
A hybrid automata example



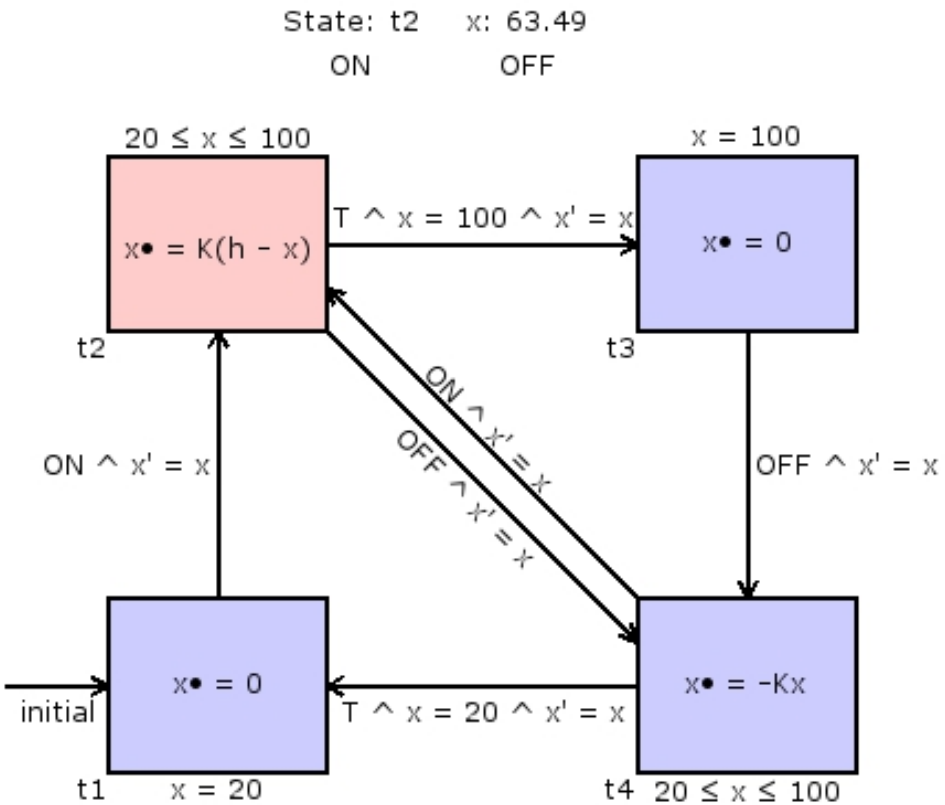


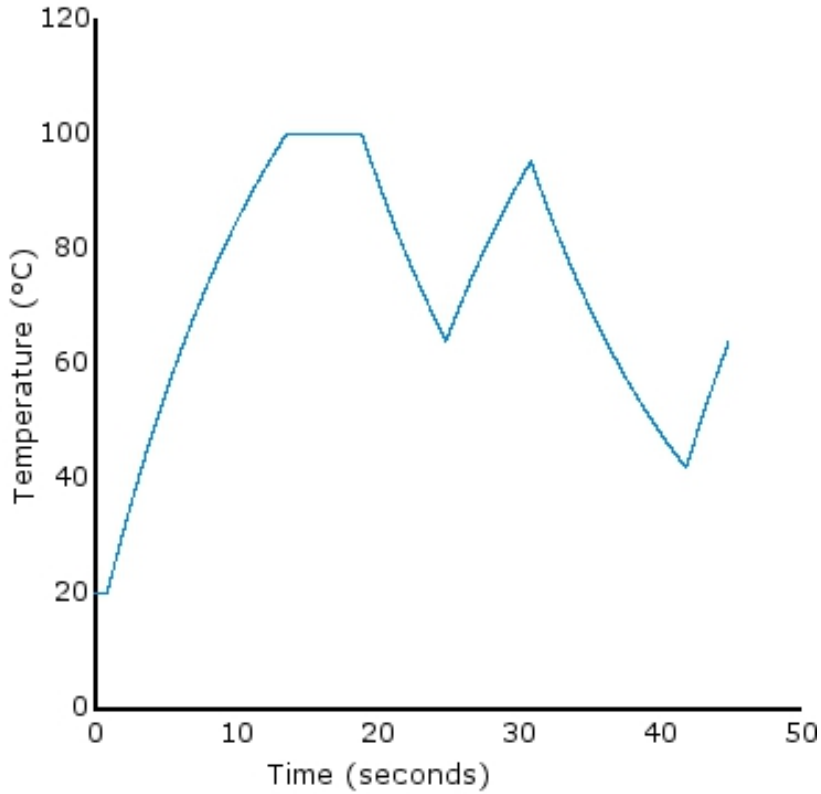
A hybrid automata example



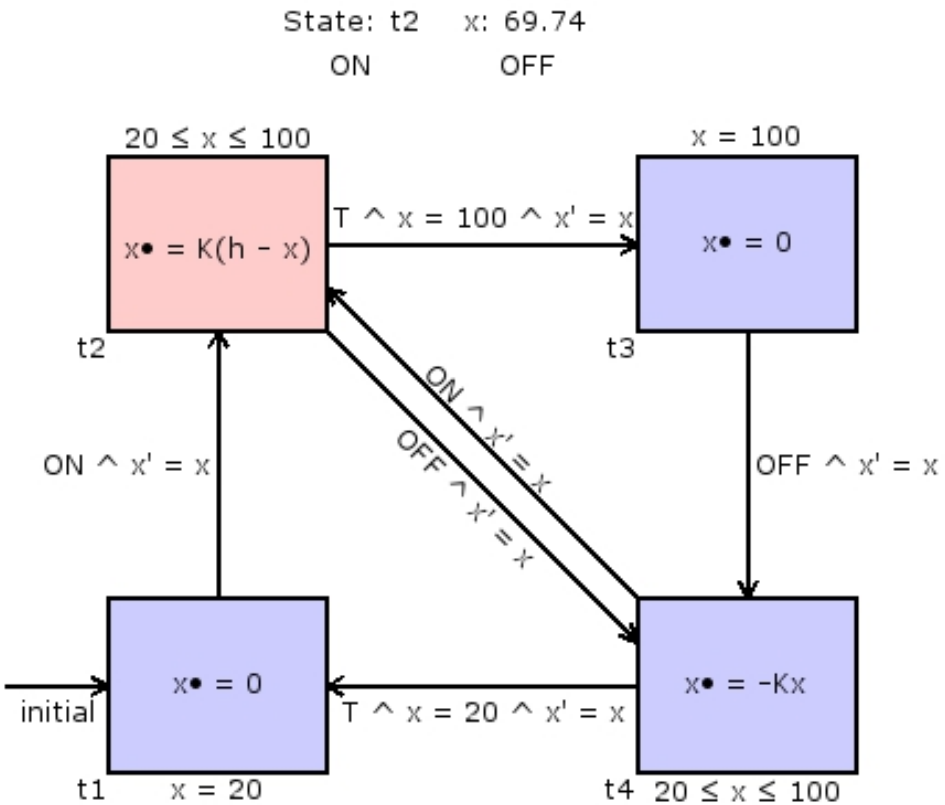


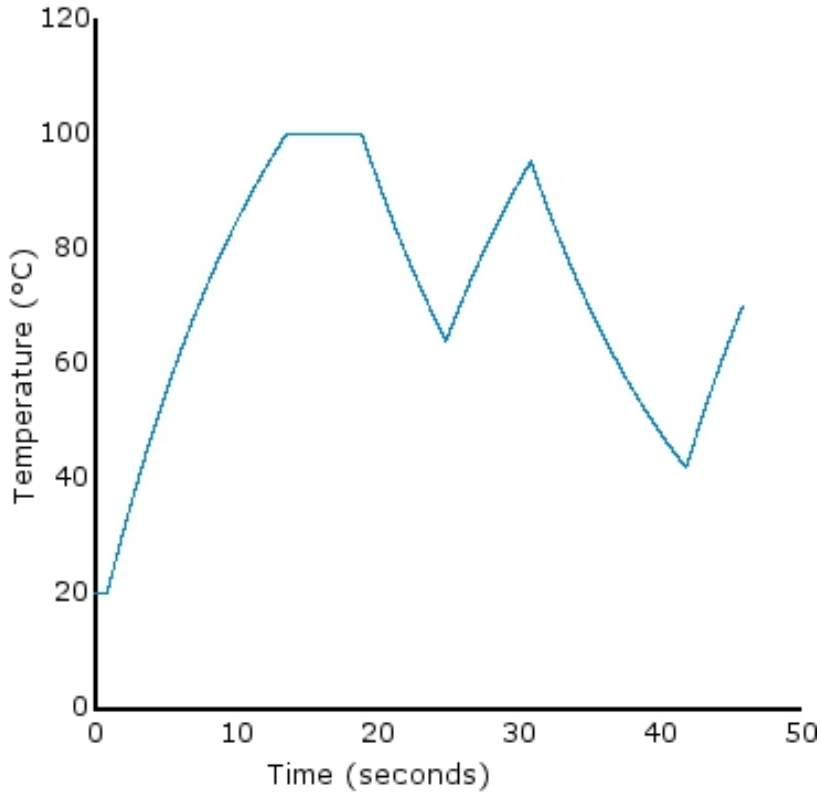
A hybrid automata example



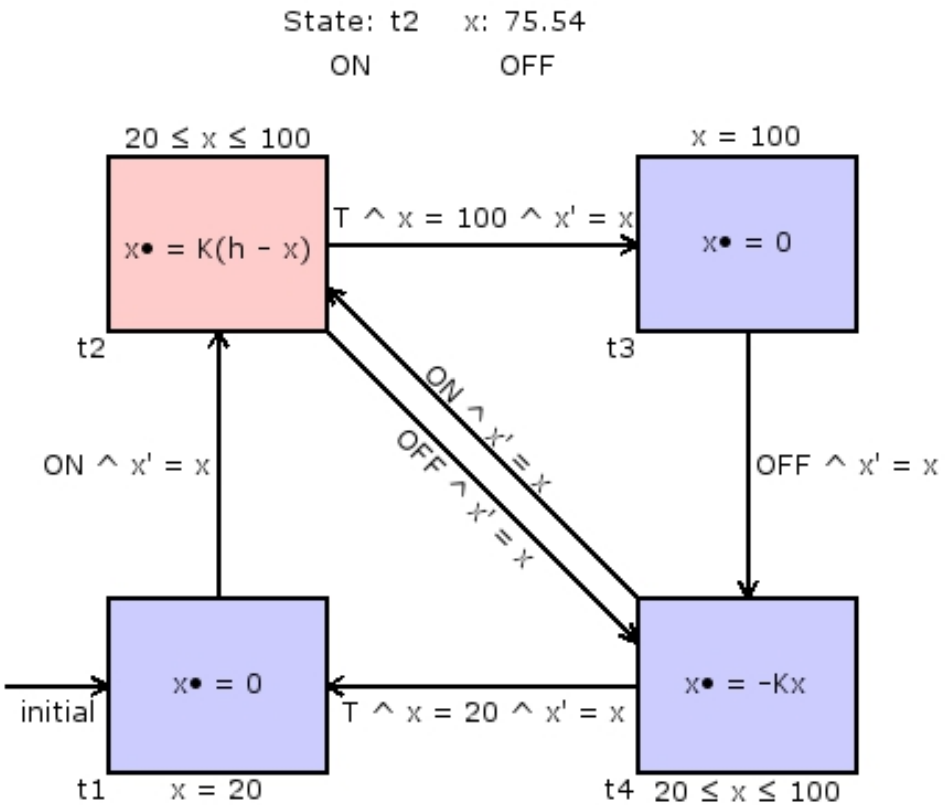


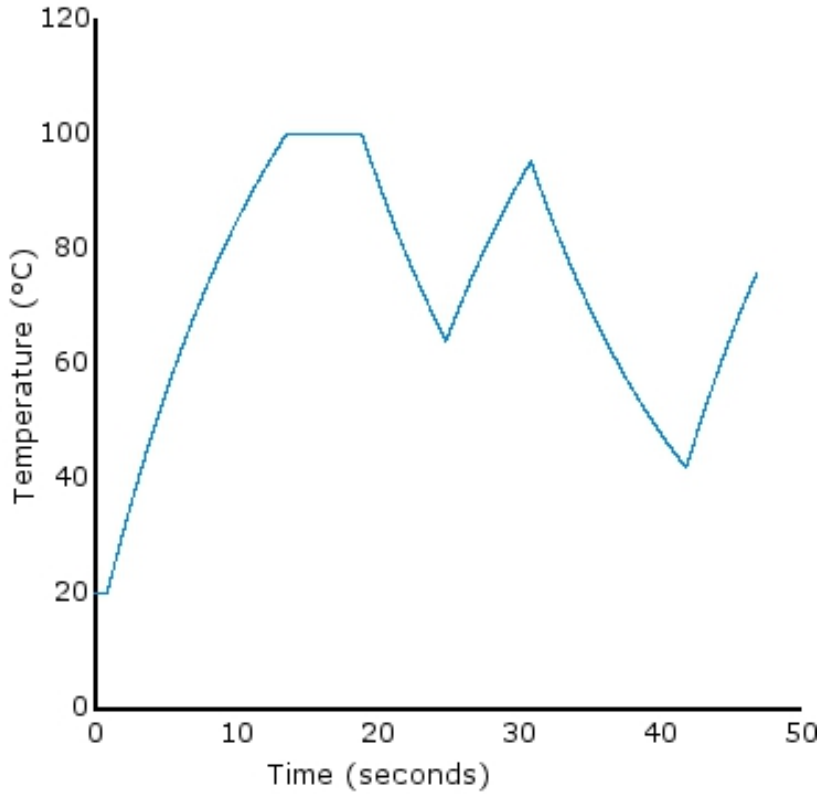
A hybrid automata example



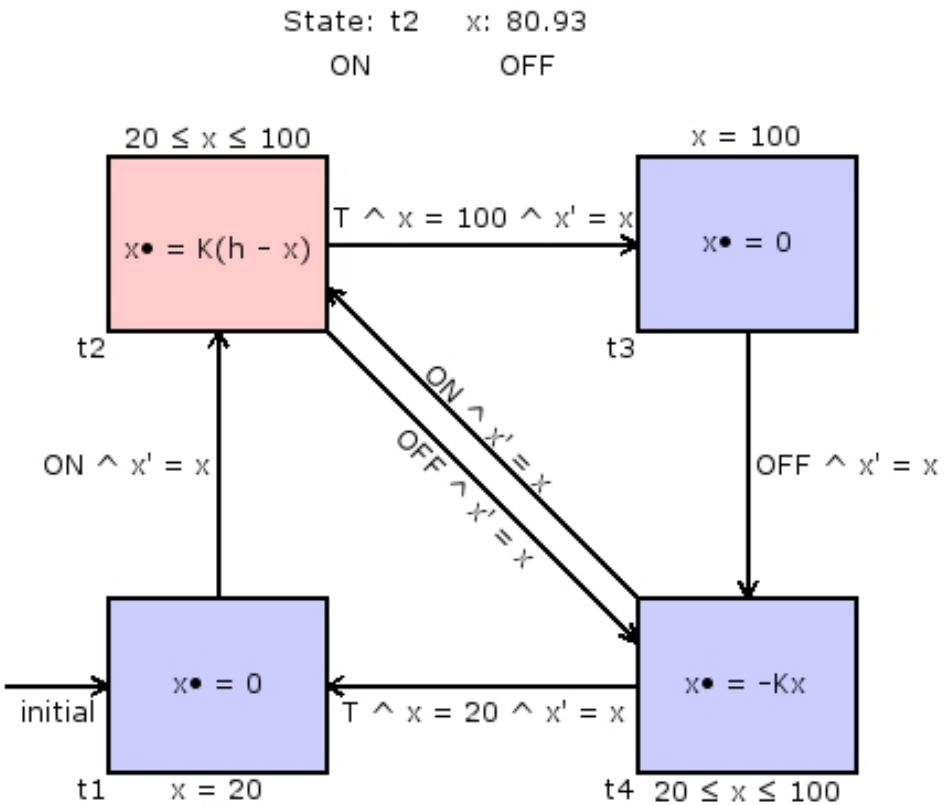


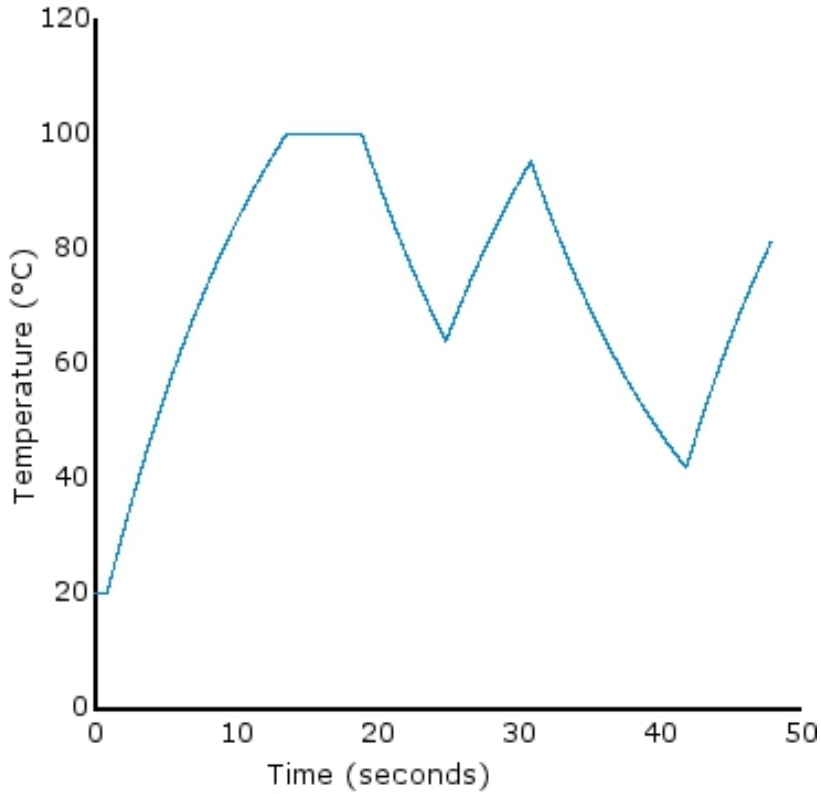
A hybrid automata example



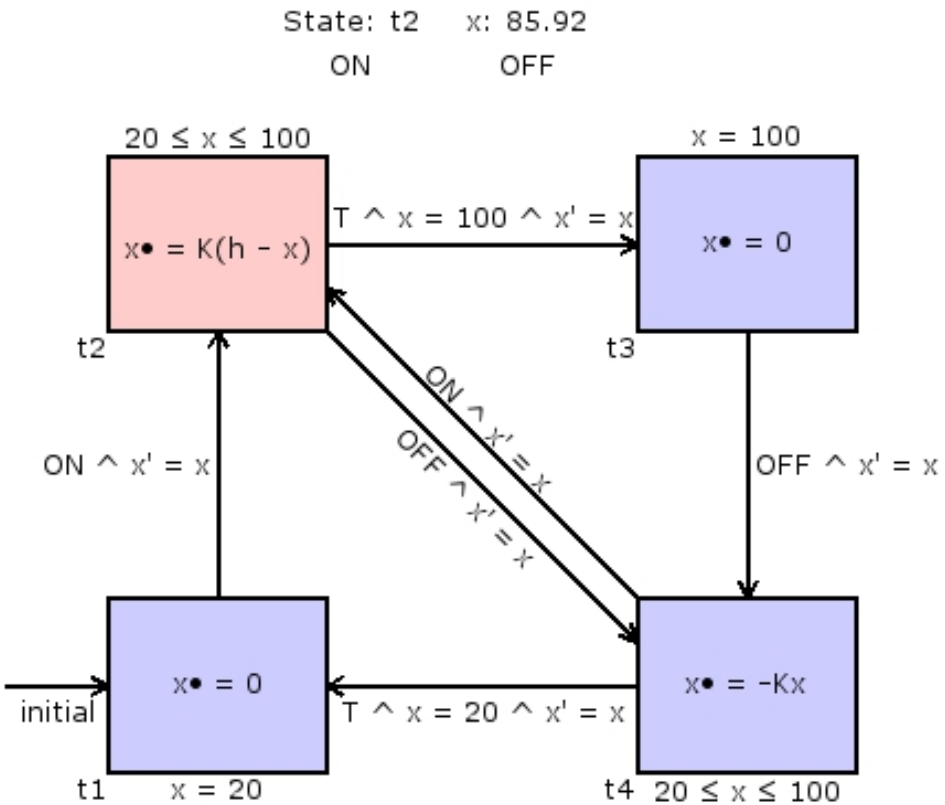


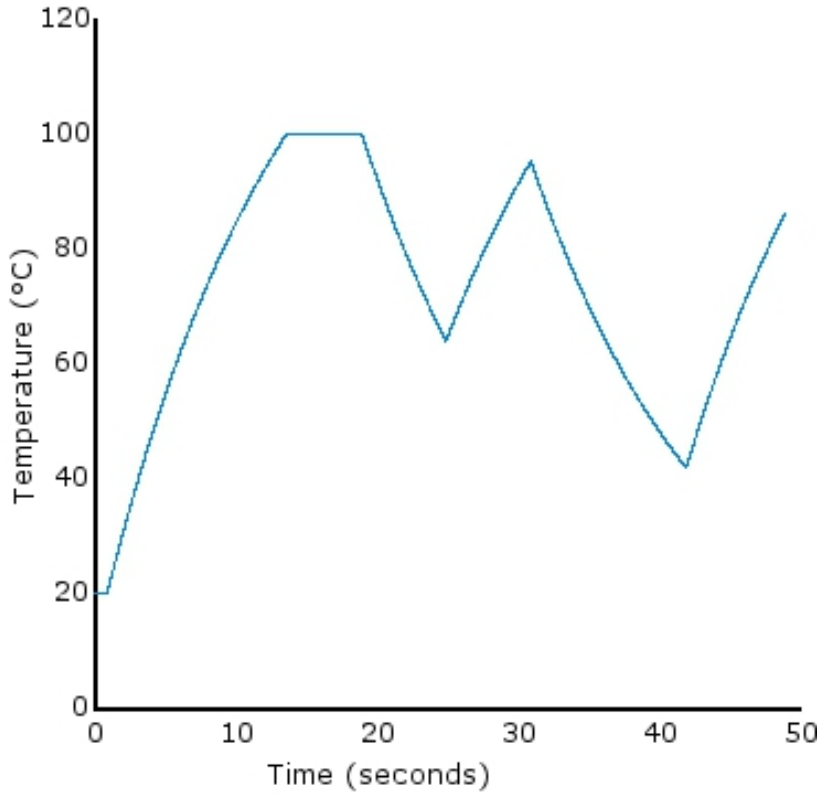
A hybrid automata example



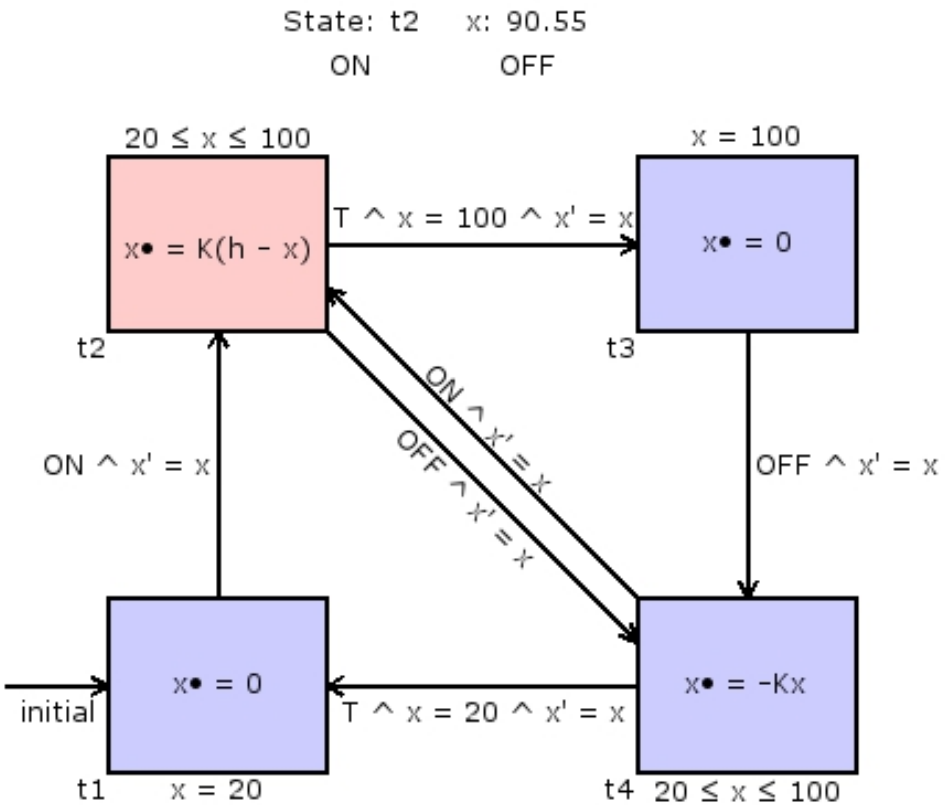


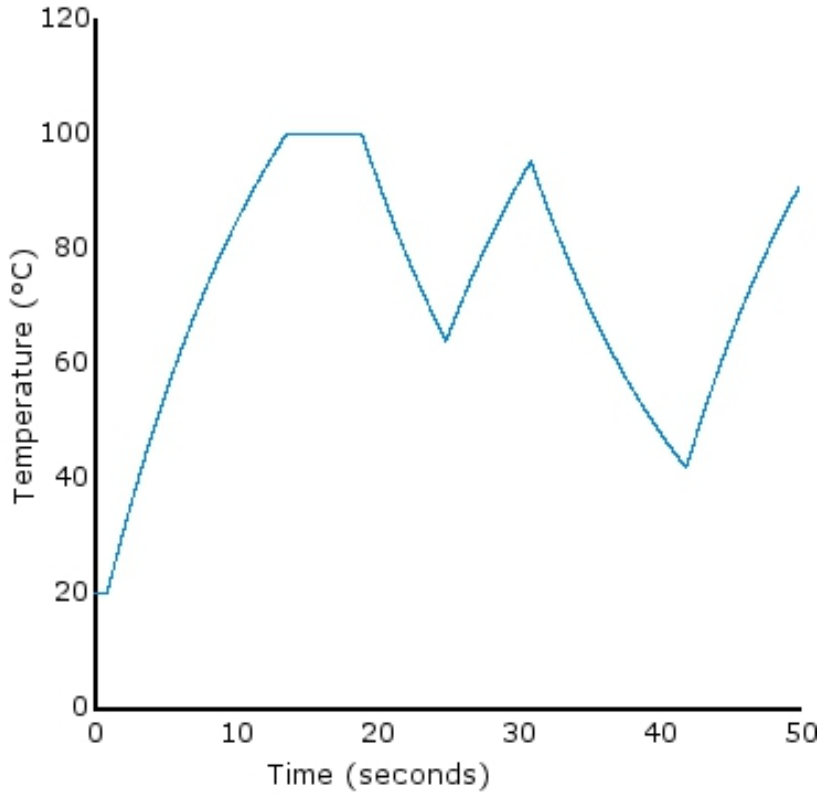
A hybrid automata example





A hybrid automata example





Definition

A hybrid automata H $= <$ *Loc*, *Edge*, Σ, *Inv*, *Flow*, *Jump* $>$ where

- $Loc = \{l_1, .., l_n\}$ representing n control modes or locations.
- Σ is the input alphabet comprising of event names.
- $Edge \subseteq Loc \times \Sigma \times Loc$ are the set of edges between locations.
- Three sets for the set of continuous variables, their rate of change and their updated values represented as follows: $X = \{x_1, .., x_m\}$ $\dot{X} = \{\dot{x_1}, .., \dot{x_m}\}$ $X' = \{x'_1, .., x'_m\}$.
- $Init(l)$: Is a predicate whose free variables are from X. It specifies the possible valuations of these when the HA starts in l.
- $Inv(l)$: Is a predicate whose free variables are from X and it constrains these when the HA resides in l.
- $Flow(l)$: Is a predicate whose free variables are from $X \cup \dot{X}$ and it specifies the rate of change of these variables when the HA resides in l.
- $Jump(e)$: Is a function that assigns to the edge e a predicate whose free variables are from $X \cup X'$. This predicate specifies when the mode switch using e is possible. It also specifies the updated values of the variables when this mode switch happens.

Definition

The semantics of a HA H$= < Loc, Edge, \Sigma, Inv, Flow, Jump >$ is provided using a timed transition system TTA $= < Q, Q_0, \Sigma, \rightarrow >$

- Q is for the form (l, v) where l is a location and $v \in [X \rightarrow R]$ such that v satisfies $Inv(l)$. Q is called the state space of H.
- $Q_0 \subseteq Q$ of the form (l, v) such that v satisfies $Init(l)$.
- $\rightarrow$ is the set of transitions consisting of either:
  - Discrete transitions: For each edge $e = (l, \sigma, l')$, we have $(l, v) \xrightarrow{\sigma} (l', v')$ if $(l, v) \in Q$, $(l', v') \in Q$ and (v, v') satisfy $Jump(e)$. These take zero time.
  - Continuous transitions: When control remains in a location and time progresses. Here the continuous variables evolve according to the ODEs as long as the invariant holds.

A revolutionary approach for drug validation

- Human trials of drugs have seen several catastrophic consequences i.e. BIA 10-2474, a drug trialed in France to treat Parkinson's. There have been several adverse events including deaths. [a]
- An Organ on Chip (OoC) [b] is a biofluidic chip with living cells that is capable of recreating organ level physiology.
- Such OoCs are being developed to simplify the clinical trial phase of drugs.

---

[a] Butler, D. and Callaway, E. Scientists in the dark after fatal French clinical trial. Nature 529 (2016).

[b] Bhatia, S. N. and Ingber, D. E. Microfluidic organs-on-chips. Nature biotechnology 32, 760–772 (2014)

A Lung on Chip: Revolutionary approach for drug validation

a

[a] Source:www.ted.com/talks/francis_collins_we_need_better_drugs_now. Dated 21 March 2013. Esch, E. W., Bahinski, A. & Huh, D. Organs-on-chips at the frontiers of drug discovery. Nature reviews Drug discovery 14, (2015).

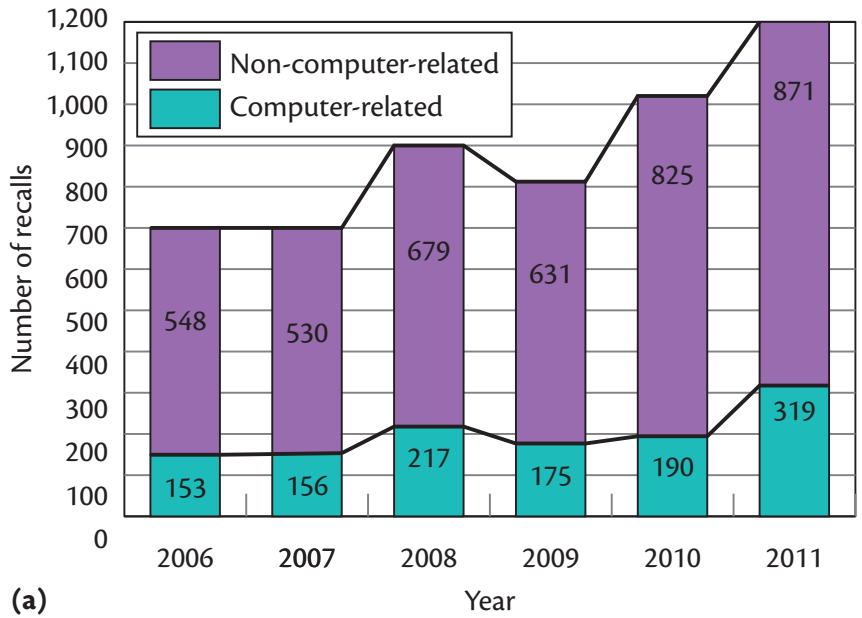
Problems

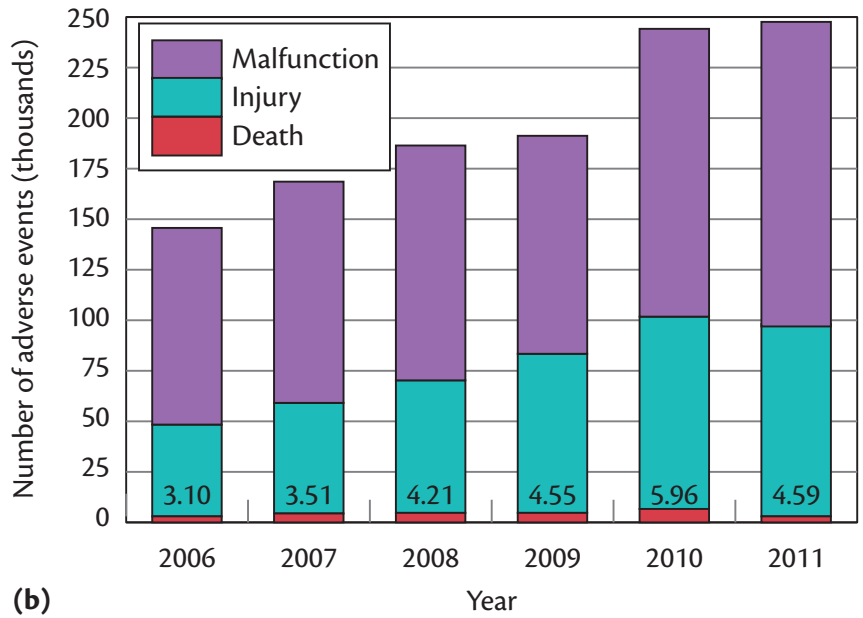
- Excess of 10 Million Pacemakers in use worldwide.
- Between 1990-2000, 41% (or 200,000) of the recalled pacemakers and implantable cardiovascular- defibrillators (ICDs) in USA were due to firmware issues [a].

[a] Jiang, Zhihao, Miroslav Pajic, and Rahul Mangharam. "Cyber physical modeling of implantable cardiac medical devices." Proceedings of the IEEE100.1 (2012): 122-137.

Device problems are continuing

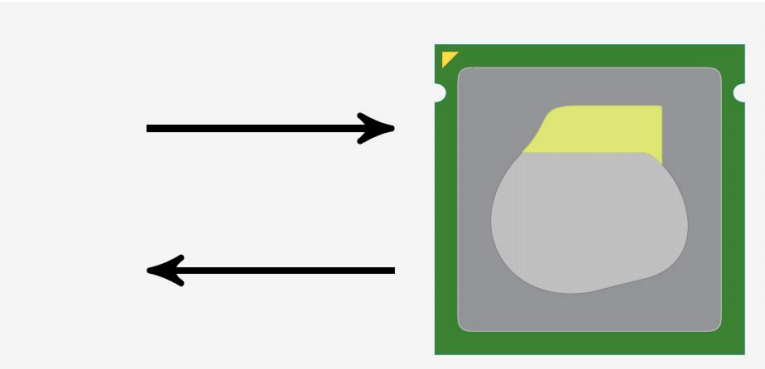
Problems





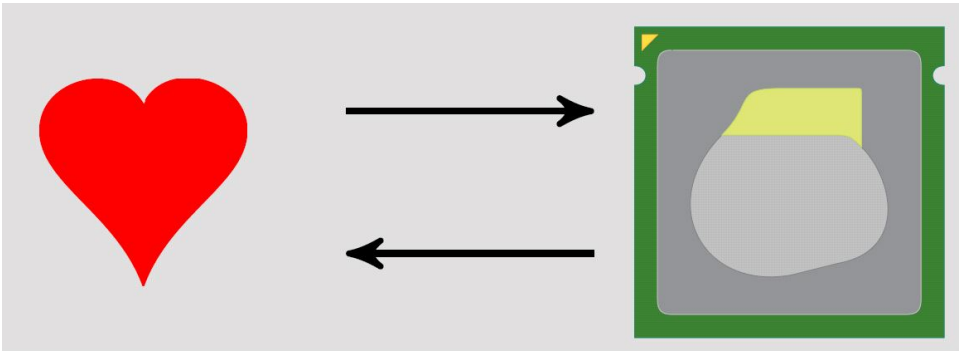
[a] Alemzadeh, H., Iyer, R. K., Kalbarczyk, Z. & Raman, J. Analysis of safety-critical computer failures in medical devices. Security & Privacy, IEEE 11, 14 26 (2013)

- This validation is usually done open-loop.
- System under validation is stimulated using an input trace to observe its response.
- Limitations: Coverage criteria dependent, exhaustive simulation infeasible.



Simulation for validating a pacemaker

- Operating a **controller** under test in closed-loop with the actual physical process (the **plant or the environment**) [6].
- The design of the controller follows the principles of real-time systems.
- The controller is digital in nature, while the plant usually exhibits continuous dynamics and is uncontrollable.



Emulation for validating a pacemaker
(Actual heart + pacemaker model)

Bio-emulation example

a

[a] source:ABI / FMHS Experiment Dated 18 May 2015

Limitations of emulation

- The plant and the controller may need to be designed in parallel i.e. a rehabilitation robot.
- Model-in-the-loop simulation using Simulink and Stateflow: semantic issues [8, 1] and issues with model fidelity.
- Ptolemy [7] and Zélus [1] are tools with formal semantics. However, these are suitable for the modelling of closed systems using HA models. Also, like SL/SF they interact dynamically with ODE solvers. This is not good for emulation.
- Potemy has incorporated a QSS-based solver [2] to overcome the above. This, however, is unsuitable for open systems.

There is no approach for black-box validation of controllers (say Pacemakers) using real-time plant models. This requires:

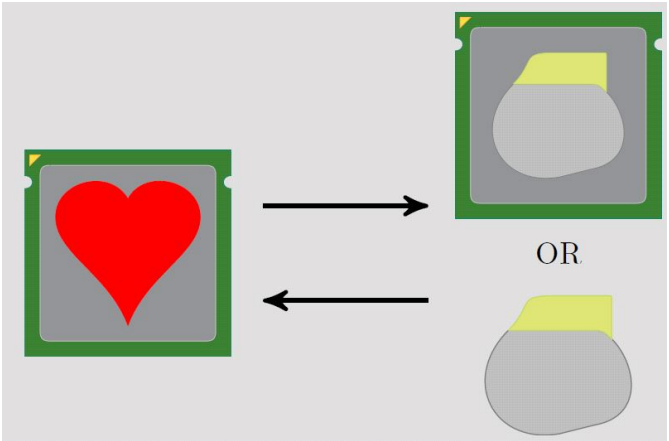
- Open models of the plant using a network of hybrid input output automata (HIOA [5]).
- Formal semantics of HIOA models and their compositions.
- Automatic techniques for modular code generation.
- Static timing analysis of the plant for plant-controller timing compatibility i.e. correct timing verification to ensure that the sampling time of the plant and controller match [3]

We propose the new technique of remulation for this.

What is remulation?

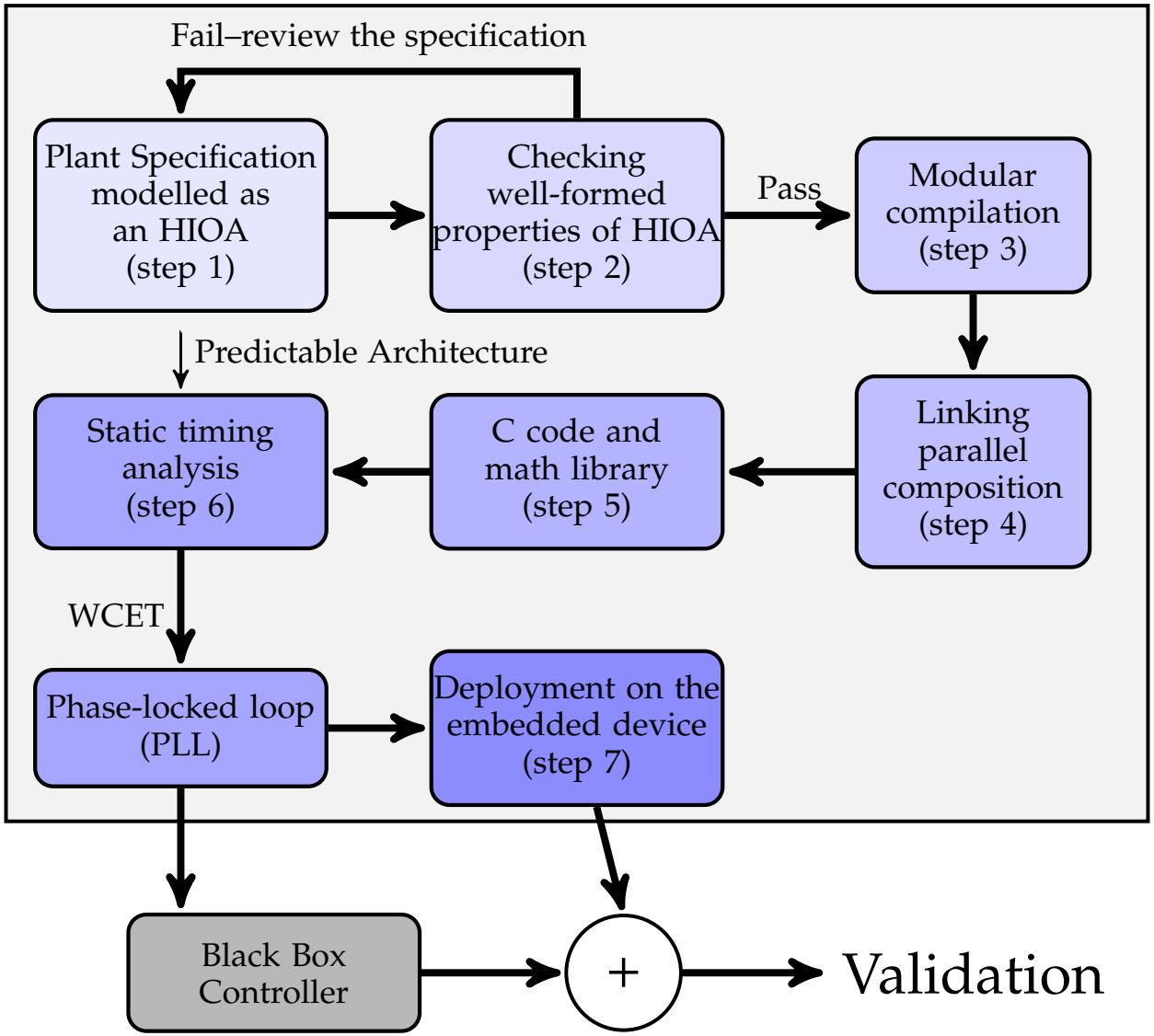
Remulation stands for reverse emulation using an executable model of the plant, we term a plant-on-a-chip (PoC). During remulation, the plant-controller relationship is reversed.

- We have to synthesize a suitable model of the r-controller (the traditional plant).
- The r-plant (the usual controller) acts as an environment for the r-controller. The r-plant is black-box in nature.



Remulation for validating a pacemaker
(Heart model (real-time) + pacemaker actual/model)

Methodology overview



A recent view of biology

- Executable Biology – Executable computational algorithms that mimic biological phenomena.[a]
- A cell may be viewed as a reactive system: "The cell is a reactive system that expresses a dynamic narrative, in which the DNA code is one of many formative inputs." [b]

From reactive to Cyber-physical systems view of biology.

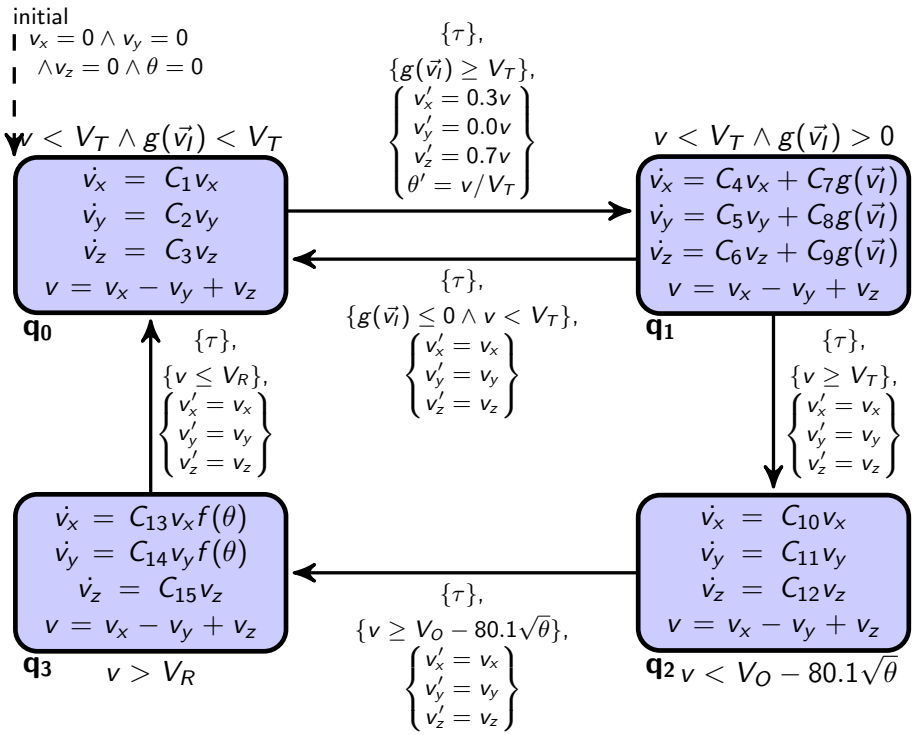
[a] Fisher and Henzinger, "Executable Cell Biology", Nature Biotechnology, 2007.

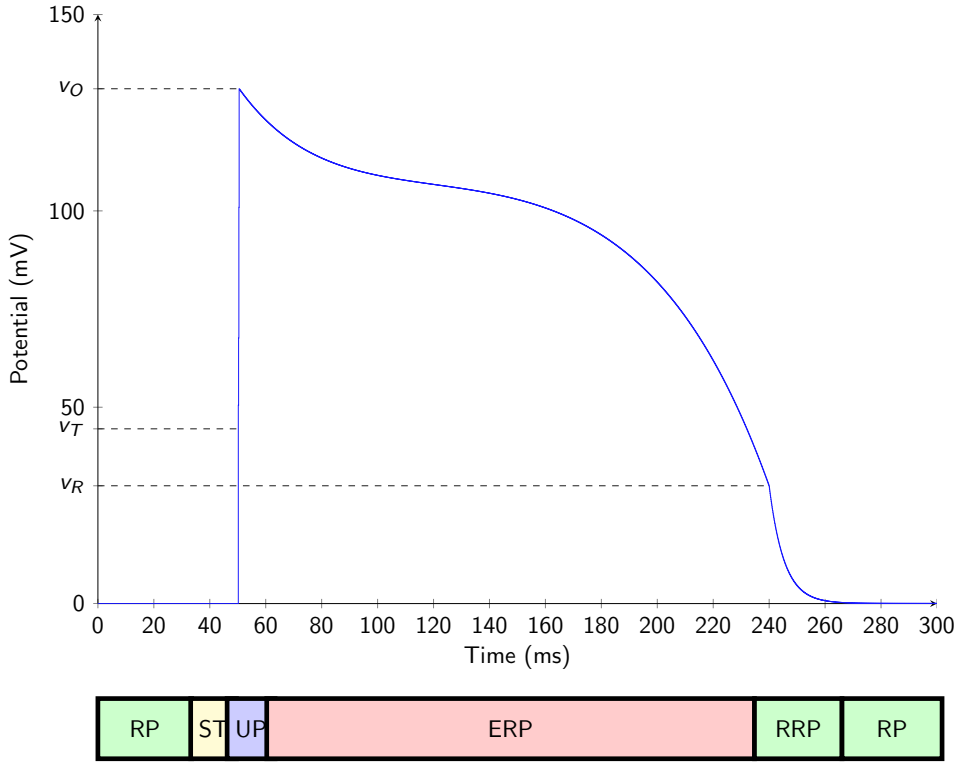
[b] Fisher, Harel and Henzinger, "Biology as Reactivity", CACM 2011.

BfOoC vs ExOoC

	BfOoC	ExOoC
Mechanism	Controls the fluid flow using various sensors to grow living cells and understand their behavior	Controls the data flow using switches to model cells and understand their behavior.
Goal	Synthesis of minimal functional units that are replicate tissue or organ level behavior.	Synthesis of organ level behavior from a device perspective.
Drug testing	Yes	No
Device testing	No	Yes
Time taken to re-configure	Days-Months	in Minutes
Cost	Expensive	Cheap
Fabrication technology	Produced in Labs since 2011 [1]	Mass produced by industry (since 1984) [2]

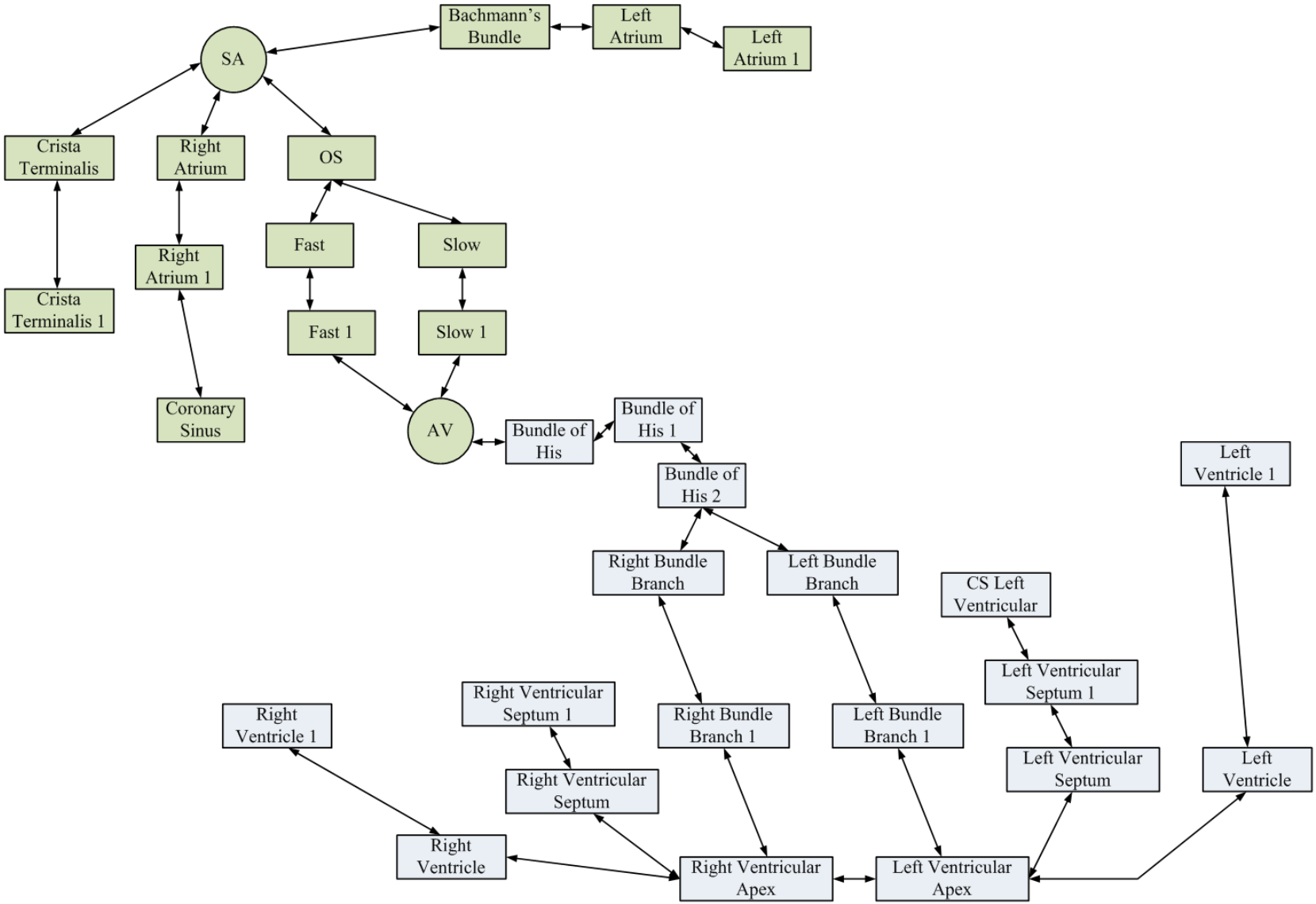
UoA Model - Cell





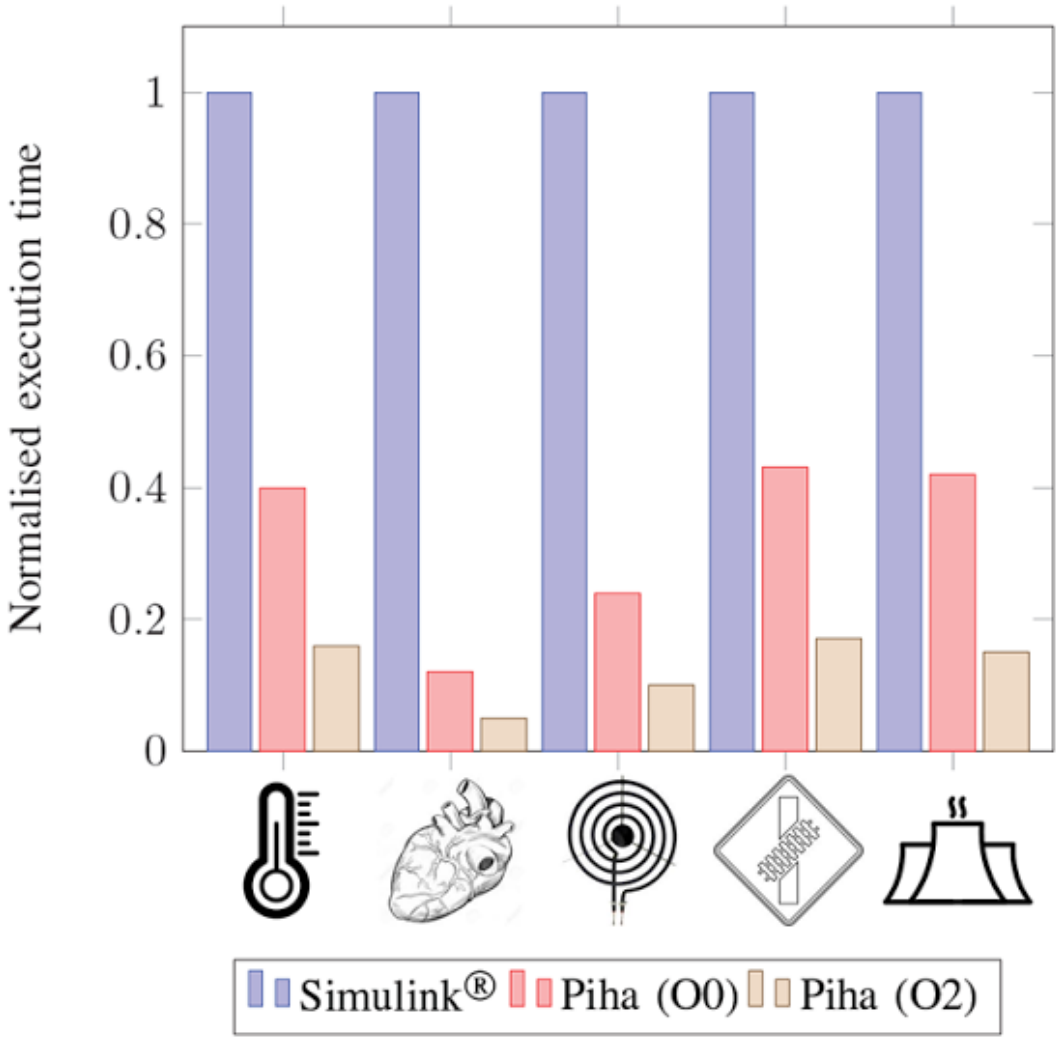
HA based on Ventricular AP.

Conduction Network

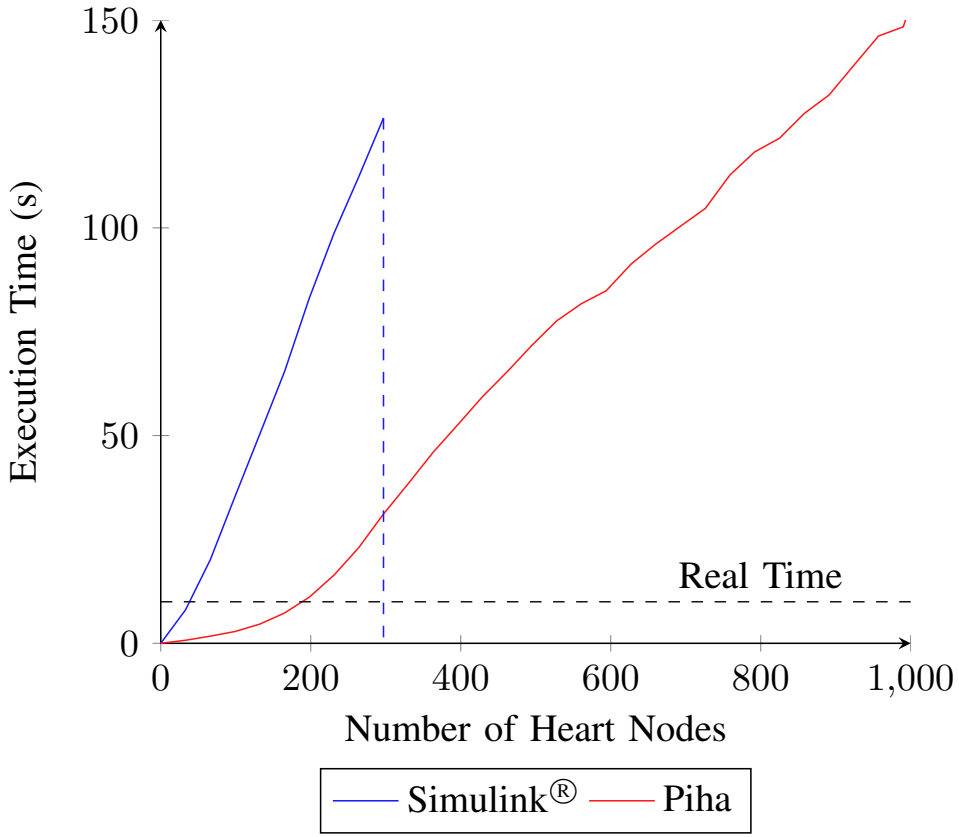


Simulink vs Piha – Execution time

On average 9.8 times faster than Simulink. For the heart conduction system it is two orders of magnitude faster.



Scalability Relative to Simulink



- 5 times more scalable
- 40 vs 200 cells for real-time emulation

Conclusions

- We considered the problem of controller (e.g. pacemaker) validation without using the actual plant (e.g. a live heart).
- Usually controllers are designed when plant exists. We propose reverse emulation to design the plant when controller exists.
- The design approach will need background on reactive / real-time systems.
- We will adapt the *synchronous approach*, which is widely used in the design of reactive / real-time systems.

Next lecture

- Introduction to the synchronous approach.
- The Esterel language.
- SyncCharts, a graphical synchronous Statecharts.
- The SCCharts tools and the pacemaker example.

Key references

- R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Hybrid Systems" , volume 736 of Lecture Notes in Computer Science. Springer-Verlag, 1993.
- E. A. Lee et al. "Modeling and Simulating Cyber-Physical Systems using CyPhySim", ACM Embedded Software (EMSOFT) conference, 2015.
- Kopetz, Hermann. "Real-time systems: design principles for distributed embedded applications", Springer, 2011.
- Albert M. K. Cheng, "Real-Time Systems: Scheduling, Analysis, and Verification ", Wieley, 2002.
- N Allen, S. Andalam, P. S. Roop, A. Malik, M. Trew and N. Patel, "Modular code generation for emulating the electrical conduction system of the human heart", Design Automation and Test in Europe (DATE), Dresden, Germany, 14-18 March 2016.
- A. Malik, P. S. Roop, S. Andalam, E. Yip and M. Trew, "A synchronous rendering of hybrid systems for designing Plant-on-a-Chip (PoC)", arXiv preprint arXiv:1510.04336.