An Introduction to HyTech

Purandar Bhaduri
Dept. of CSE
IIT Guwahati
Email: pbhaduri@iitg.ernet.in

GLOBAL INITIATIVE OF ACADEMIC NETWORKS

Outline

- Hybrid Systems and Hybrid Automata
- Safety Requirements
- Linear Hybrid Automata
- HyTech
- Examples
- References

Hybrid Systems

Continuous Dynamical Systems

Differential Equations

(Physics, Engineering, Control)

+

Discrete Systems

State Machines, Automata

(Computer Science)

Hybrid Systems

Continuous + Discrete Dynamics

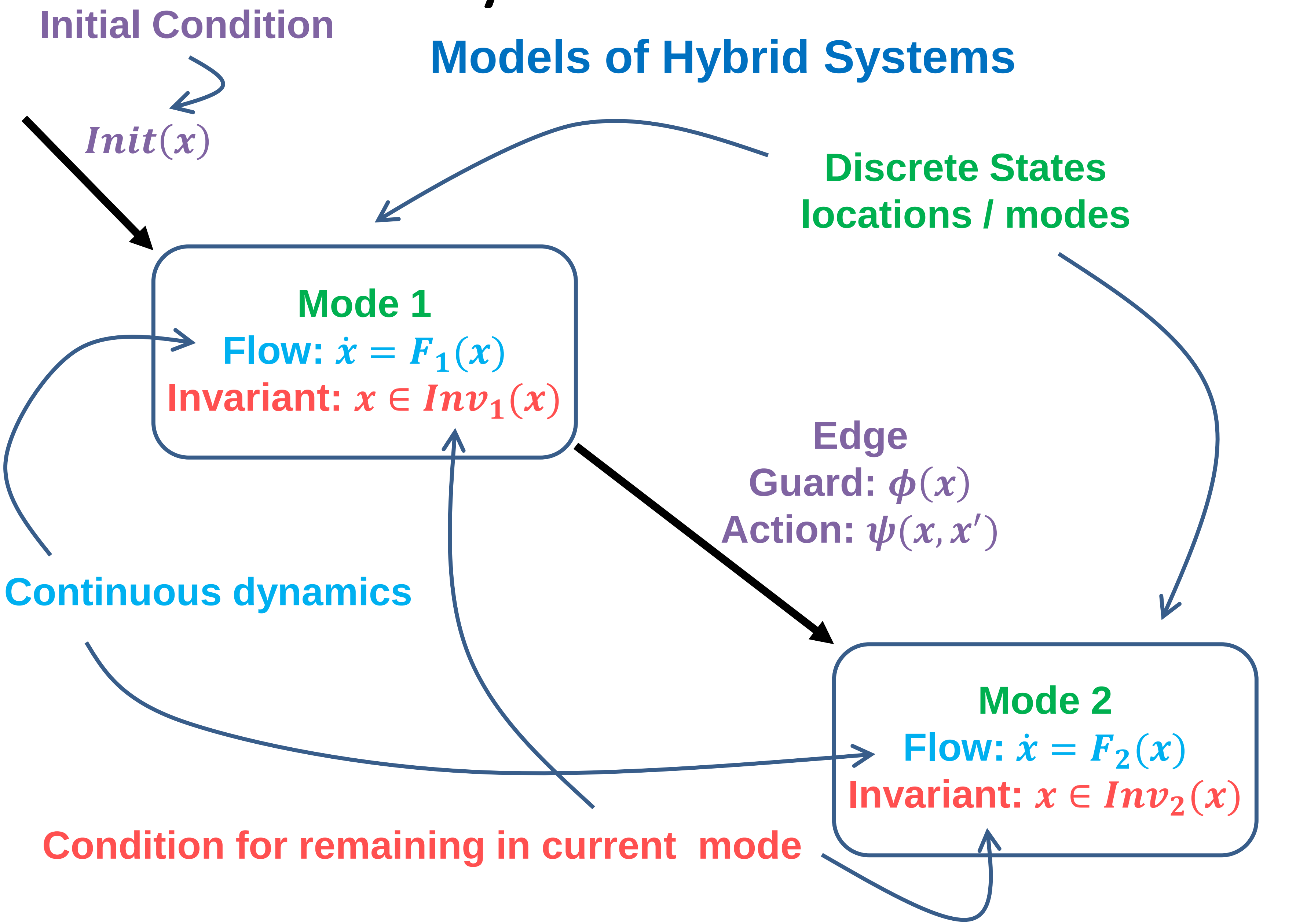
(CPS, Software controlled systems)

Hybrid Systems

- Continuous dynamics
  - Real-valued state variables $X = \{x_1, \dots, x_n\}$
  - State: $\sigma \in R^n$
  - Flow: a curve $\boldsymbol{\chi}: T \rightarrow R^n$
    - T : set of time points (usually non-negative reals)
    - Often described as a differential equation $\dot{\boldsymbol{\chi}} = \boldsymbol{F}(\boldsymbol{\chi}, \boldsymbol{t})$ or differential inclusion $\dot{\boldsymbol{\chi}} \in \boldsymbol{F}(\boldsymbol{\chi}, \boldsymbol{t})$
- Discrete dynamics
  - Control modes Q
  - Transitions (jumps)
- Hybrid (dynamical) system
  - Both continuous and discrete state variables
    - State space: $Q \times R^n$
  - A trajectory is a sequence of flows and jumps
-

Hybrid Automata

Models of Hybrid Systems



Safety Requirement

- Safety Property
  - nothing bad will ever happen
  - Often specified by describing “unsafe” states
  - Satisfied iff all reachable states are safe
  - Safety Verification = Computing Reachable States
- Safety for Hybrid Automata
  - Specified using state assertion: $\phi(v)$ for control mode v is a predicate over X; e.g., $\phi(v)(x_1, x_2) \triangleq x_2 \geq x_1$
  - the states for which ϕ is true are called ϕ-states
  - Let *unsafe*: state assertion for HA *A.*
  - Then *A* satisfies the safety requirement specified by *unsafe* if *unsafe* is false for all reachable states of *A*.
  - Sometimes additional variables and control modes may be necessary to specify safety requirement.

Computing Reachable States

- Compute a state assertion *reach* which is true for the reachable states of HA A.
- If there is a state for which *reach* and *unsafe* are both true then the safety requirement is violated; if not the safety requirement is satisfied
- Computing state assertion *reach*
  - For a state assertion ϕ let $Post(\phi)$ be a state assertion that is true precisely for the jump and flow successors of ϕ-states
  - Compute $\phi_1 = Post(init)$: all states that are reachable by trajectories of length one (single jump or flow)
  - Compute $\phi_2 = Post(\phi_1), \phi_3 = Post(\phi_2), \ldots$
  - If $\phi_{k+1} = \phi_k$ for some number k then *reach* $= \phi_k$

Can we compute *reach* this way?

- For state assertion ϕ need to be able to compute $Post(\phi)$
  - Can be done efficiently for a restricted class of HA: *Linear Hybrid Automata*
- Iterative computation of reach must converge within a finite number of applications of *Post*
  - Can be guaranteed for an even more restricted class of HA: Timed Automata
  - Practical solution: iterate till available time or space resources are exhausted
    - Semidecision procedure: no guarantee of termination

Linear Hybrid Automata

- Hybrid Automaton model
  - very expressive but prohibits automatic analysis
- Linear Hybrid Automata: restricted class of HA
  1. *Linearity*: flow, invariant, initial, jump conditions are convex linear predicates
     - finite conjunction of linear inequalities with rational coefficients and constants) over variables in $X \cup \dot{X}$,
     - e.g., $(2x_1 - 3\,\dot{x_2} \leq \frac{3}{4}) \wedge (3\,\dot{x_1} - x_2 \geq 5)$
  2. *Flow independence*: flow conditions are predicates over the variables in $\dot{X}$ i.e., do not contain variables from X
- **Theorem**: If A is an LHA and ϕ is a linear state assertion for A then $Post(\phi)$ can be computed and it is a linear state assertion for A.
- Intuition: In an LHA every flow curve can be replaced by a straight line between the two endpoints.

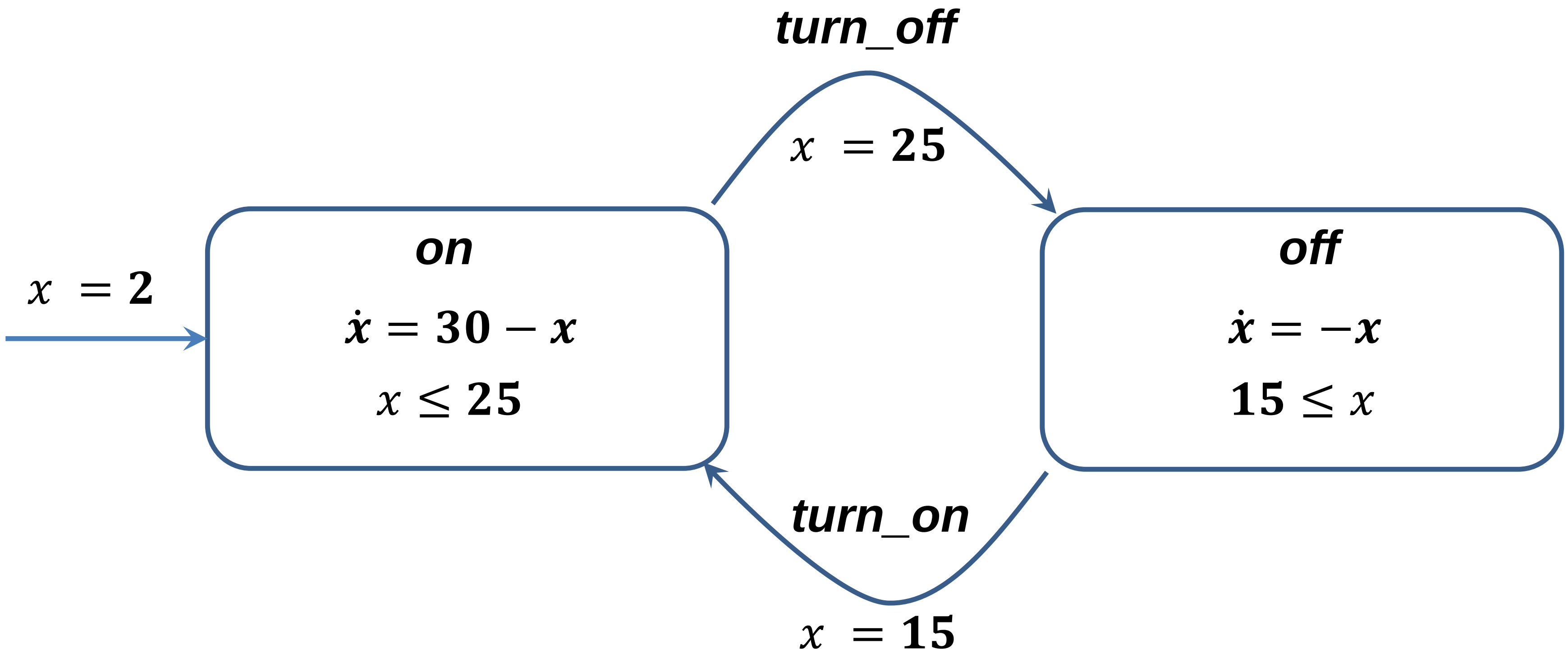
What if your HA is not an LHA?

- Nonlinear HA cannot be verified directly
- Have to replace a nonlinear HA by an LHA
  1. *Clock Translation*: sometimes the value of a variable can be determined from a past value and the time that has elapsed
  2. *Linear Phase Portrait Approximation*: Relax nonlinear flow, invariant, initial and jump conditions using weaker linear conditions.

Example 1: A Thermostat

- Two operating modes: *on* and *off*
- Initially the heater is *on* and the temperature x is 15 degrees.
- When the heater is *on* the temperature rises at the rate $-x + 30$ degrees per hour.
- When the heater is *off* the temperature falls at the rate x degrees per hour.
- Heater *can be* turned *off* when $x = 25$
- Heater *can be* turned *on* when $x = 15$
- *Inavriants* in modes are used to force mode switches
  - E.g., the invariant $x \leq 25$ in mode *on* says that a mode switch must occur before the temperature rises above 25 degrees

Hybrid Automaton for Thermostat

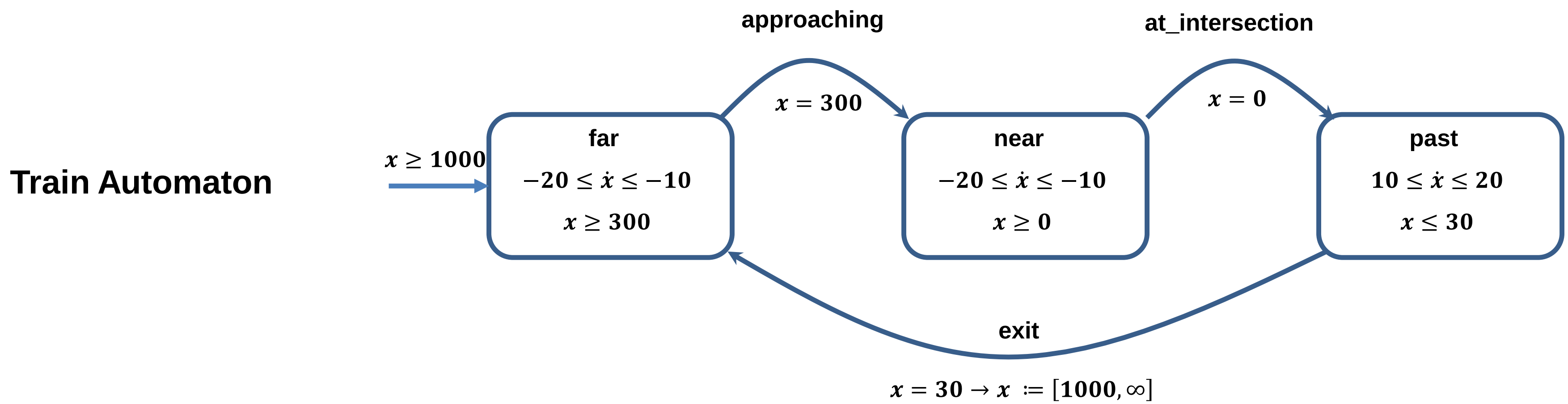


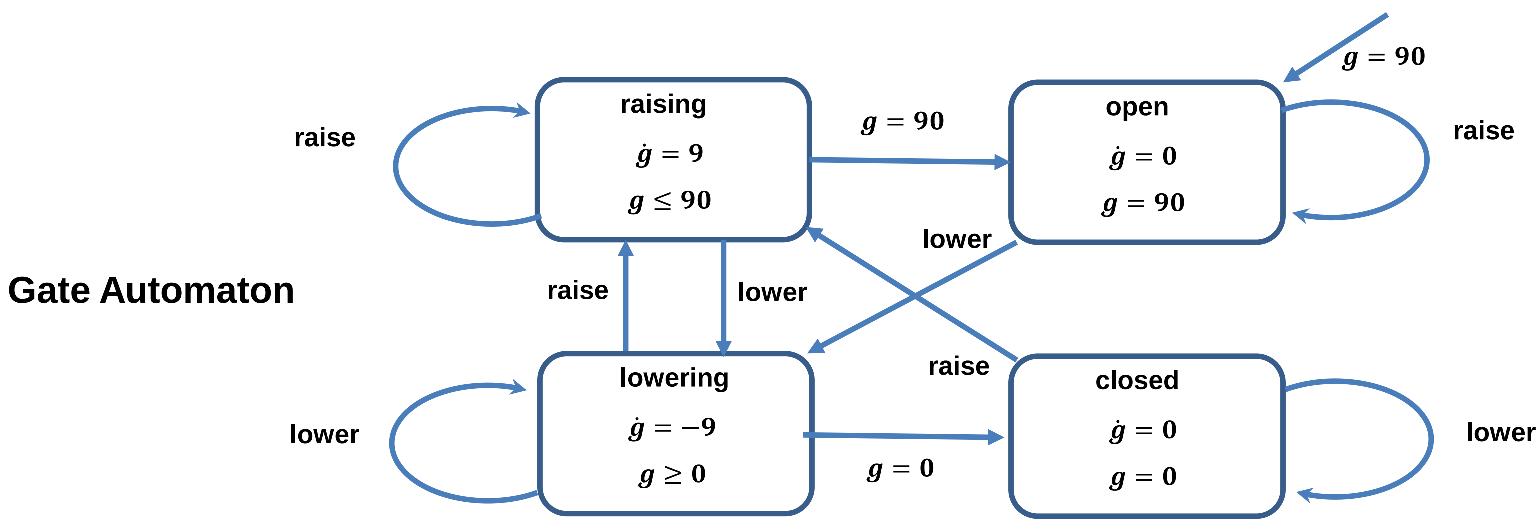
Not a Linear Hybrid Automaton!
Can use clock translation to convert to LHA.

Example 2: Railway Crossing

- Three components: train, gate, controller
- Speed of train: always between 10 and 20 m per second
- Initially
  - Train at least 1000 m away from intersection
  - Gate fully raised
- As train approaches
  - It triggers a sensor 300 m away from the intersection with gate fully raised
  - Controller then sends a ‘lower’ command to the gate after a delay of up to α seconds
- On receiving the ‘lower’ command the gate is lowered at a rate of 9 degrees per second
- Once the train has exited the intersection and is 30 m away it sends an exit signal to the controller
- The controller then commands the gate to be raised
- Role of the controller
  1. Ensure that the gate is closed whenever the train is at the intersection.
  2. The gate is not closed unnecessarily long.

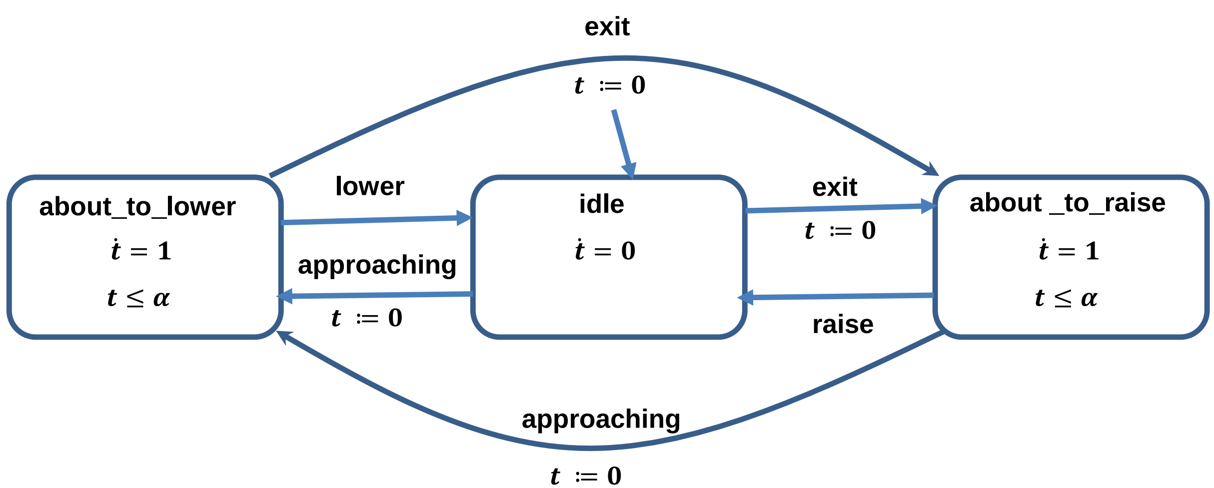
Hybrid Automata for Railway Crossing





Hybrid Automata for Railway Crossing (cont)

Controller Automaton



Railway Crossing System

- **Linear Hybrid Automaton: Modelled as the parallel composition of three LHA**
- **Communication through event synchronization and shared variables**

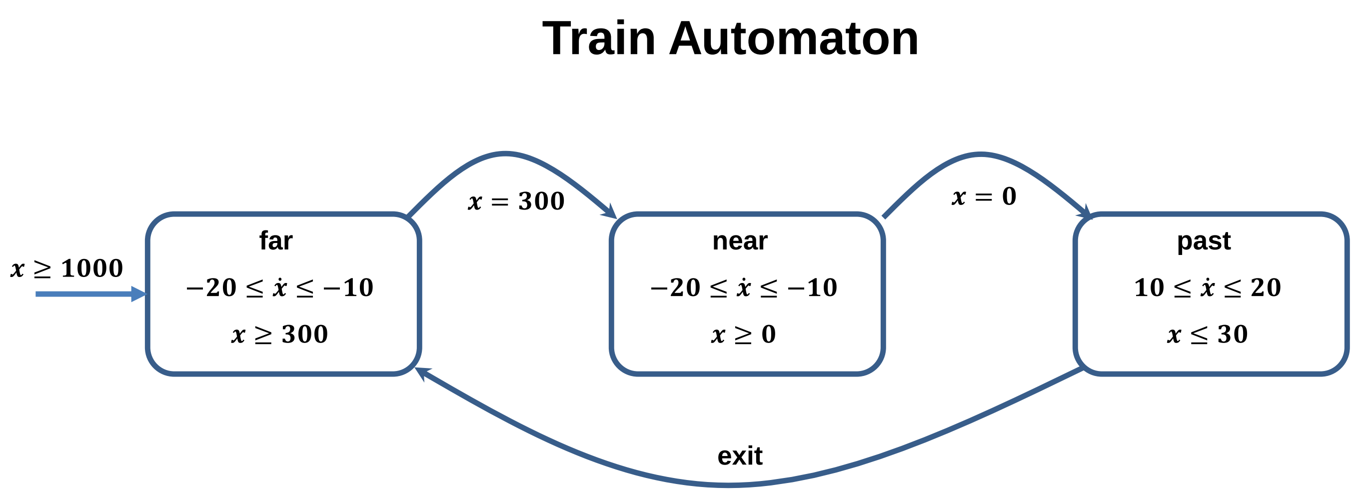
HyTech

- Symbolic model checker for LHA
  - Dynamics: linear differential inequalities of the form $\boldsymbol{A}\,\dot{\boldsymbol{x}} \sim \boldsymbol{b}$
- State sets represented by polyhedral constraints
- Termination is not guaranteed! (Unlike TA)
  - Many examples do terminate
  - Can explore behavior over a bounded interval of time
- Useful for Parametric Analysis
  - A system is described using parameters
    - E.g., the time at which the controller decides to issue the *lower* command in order for the gate to be closed by the time the train reaches the crossing
  - Designer interested in knowing which values of the parameter required for correctness
  - HyTech computes necessary and sufficient constraints on parameter values that guarantee correctness.

HyTech

- Input (text file)
  1. Collection of LHA
     - Automatically composed for analysis
  2. Sequence of analysis commands
     - Simple while programming language
       - Date type “state assertion”
       - Operations include Post
       - Boolean operators and existential quantification
       - Built-in macros: reachability, parametric analysis, conservative approximation of state assertions, generation of error trajectories

Example: LHA in HyTech




```
automaton train
synclabs : app,            -- (send) approach signal for train
           exit;           -- (send) signal that train is leaving
initially far & x>=1000;
loc far: while x>=300  wait {dx in [-20,-10]}
         when x=300 sync app goto near;
loc near: while x>=0   wait {dx in [-20,-10]}
         when x=0 goto past;
loc past: while x<=30 wait {dx in [ 10, 20]}
         when x=30 do {x' = 1000} sync exit goto far;
end -- train
```

Example: Analysis Commands

Region: a set of states

```
var init_reg, final_reg, reached: region;

init_reg := loc[train]=far & x>=1000 &
            loc[controller]=idle &
            loc[gate]=open & y=90;
final_reg := loc[gate] = raising & x<=10 | loc[gate]=open & x<=10 | loc[gate] =
lowering & x<=10;

reached := reach forward from init_reg endreach;

if empty(reached&final_reg)
        then prints "Train-gate controller maintains safety requirement";
        else prints "Train-gate controller violates safety requirement";
endif;
```

Example: Parametric Analysis

```
var init_reg, final_reg, reached: region;

init_reg := loc[train]=far & x>=1000 &
                loc[controller]=idle &
                loc[gate]=open & y=90;
final_reg := loc[gate] = raising & x<=10 | loc[gate]=open & x<=10 | loc[gate] =
lowering & x<=10;

reached := reach forward from init_reg endreach;

prints "Conditions under which system violates safety requirement";
print omit all locations
             hide non_parameters in reached & final_reg endhide;
```

Outputs the constraint on the parameter α under which the system is not correct.

References

1. Henzinger, Thomas A., Pei-Hsin Ho, and Howard Wong-Toi, “A user guide to HyTech" *TACAS, LNCS 1019,* Springer, 1995.
2. Henzinger, Thomas A., Pei-Hsin Ho, and Howard Wong-Toi. "HyTech: a model checker for hybrid systems." *International Journal on Software Tools for Technology Transfer*, vol 1, 1997.