

Switching Logic Synthesis Using Integrated Learning and Verification

Susmit Jha
PhD Student
UC Berkeley

Join work with my advisor Professor Sanjit Seshia
Dr. Ashish Tiwari (SRI) and Dr. Sumit Gulwani (MSR)

From Verification to Synthesis

- Automatic verification and testing of systems has been a widely studied field.
- Growing interest in adapting verification techniques for automatic synthesis. (Solar-Lezama et al APLOS06, Srivastava et al POPL10)
- Verification-guided synthesis tools depend on proofs or counter-examples obtained during verification for synthesis.
- Issues with verification-guided synthesis
 - No proof or counter-example
 - Computationally very expensive
 - Verification problem may be undecidable.

SILVER: Synthesis Using Integrated Learning and Verification

Can we automate synthesis beyond the limitations of verification techniques ?

SILVER: Synthesis Using Integrated Learning and Verification

Can we automate synthesis beyond the limitations of verification techniques ?

YES

Assumption on
structure of the
system

+

Lightweight
Verification /
Testing

+

Algorithmic
Learning
from examples

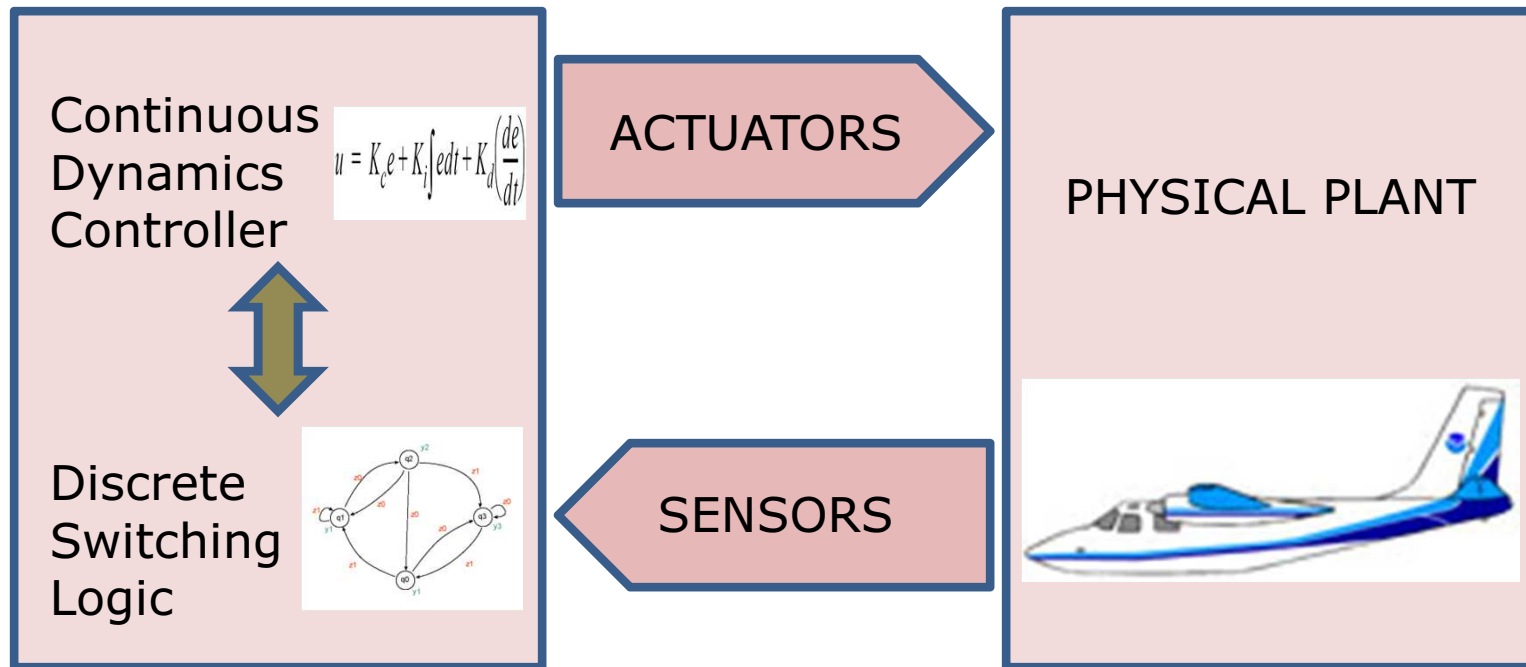
We illustrate this approach by applying our approach to synthesis of hybrid systems (ICCPS2010) and synthesis of programs (ICSE2010).

Outline

- Problem Definition
- Related Work
- Our approach for
 - Synthesis for Safety Objectives
 - Synthesis for Performance Objectives
- Results
- Conclusion and Future Work

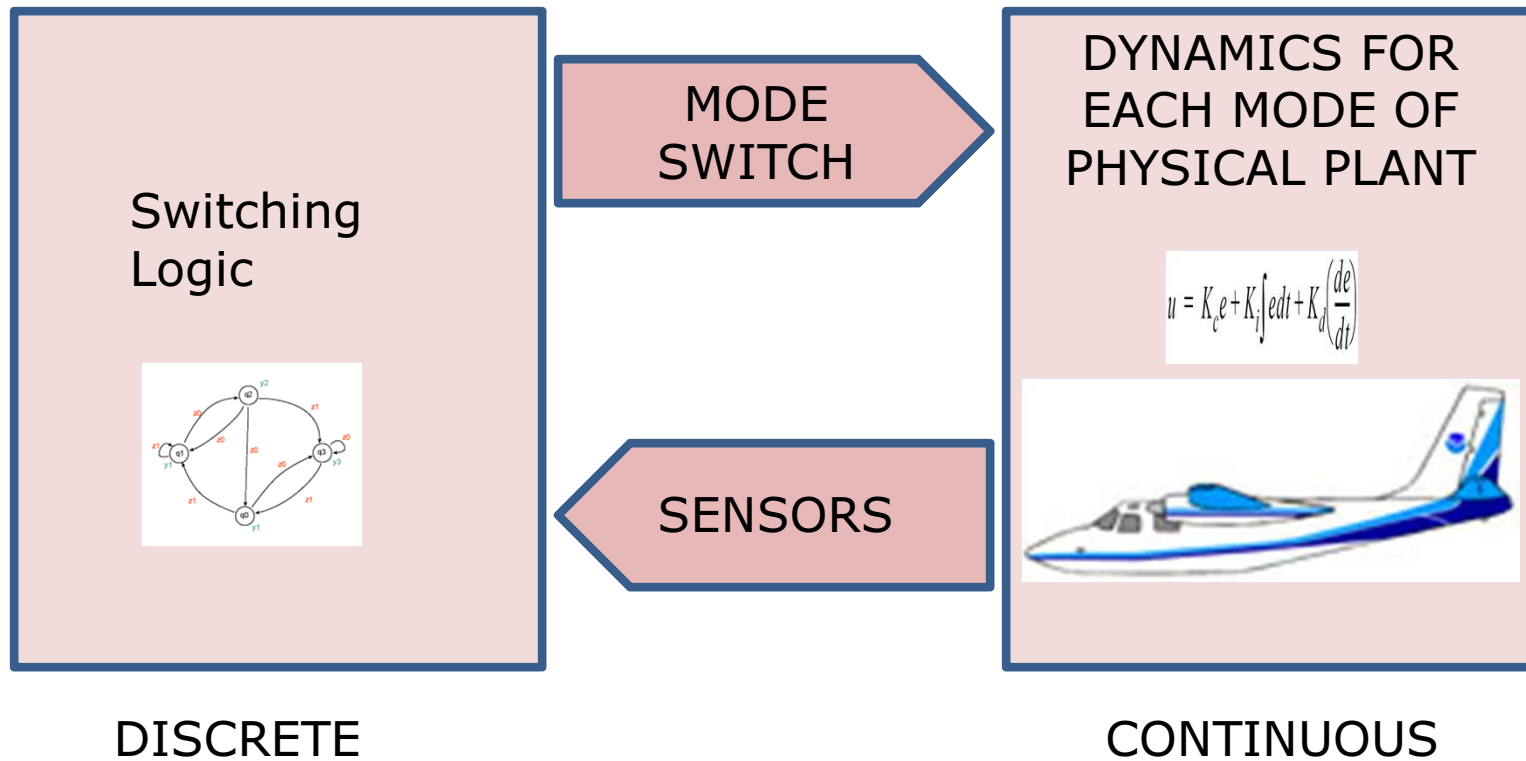
Cyber-physical System

Schematic View



Cyber-physical System

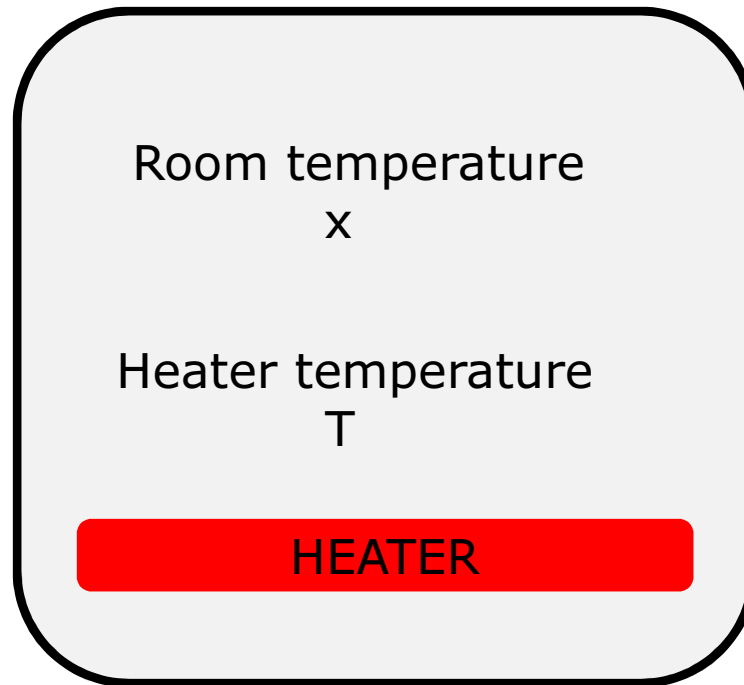
Multi-mode dynamics control



Toy Example: Room temperature controller

Thermostat Controller

Outside
Temperature
16 Celsius

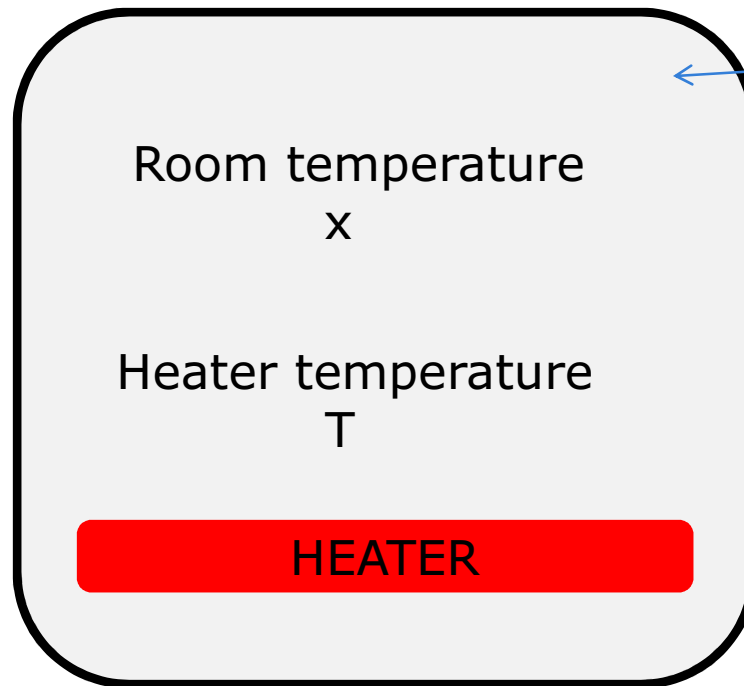


Room

Toy Example: Room temperature controller

Thermostat Controller

Outside
Temperature
16 Celsius



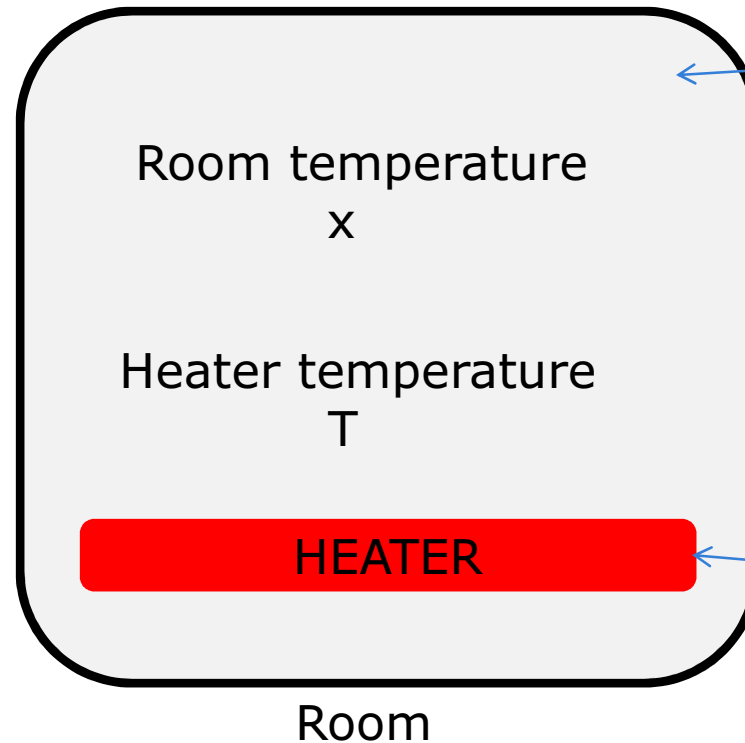
Room

Heats or cools
proportional to
difference between
room temperature
and the heater or
outside world

Toy Example: Room temperature controller

Thermostat Controller

Outside
Temperature
16 Celsius



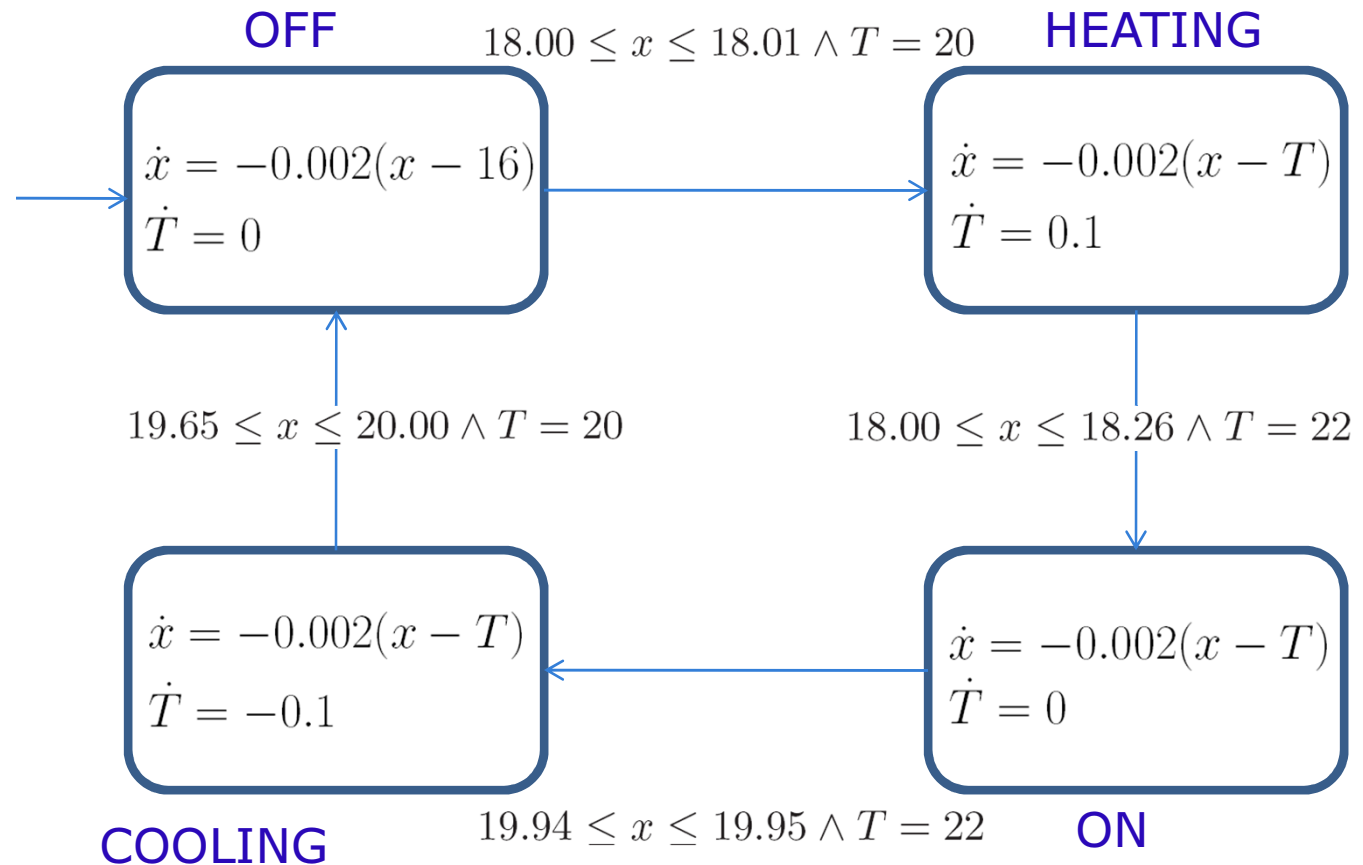
Heats or cools proportional to difference between room temperature and the heater or outside world

Heats or cools at constant rate of 0.1 Celsius/second.

Cools to 20 and heats to 22

Hybrid Automata

Dynamics: Location \rightarrow Ordinary Differential Equations (ODEs)
Switching Logic \subseteq Location \times Location \rightarrow Predicates

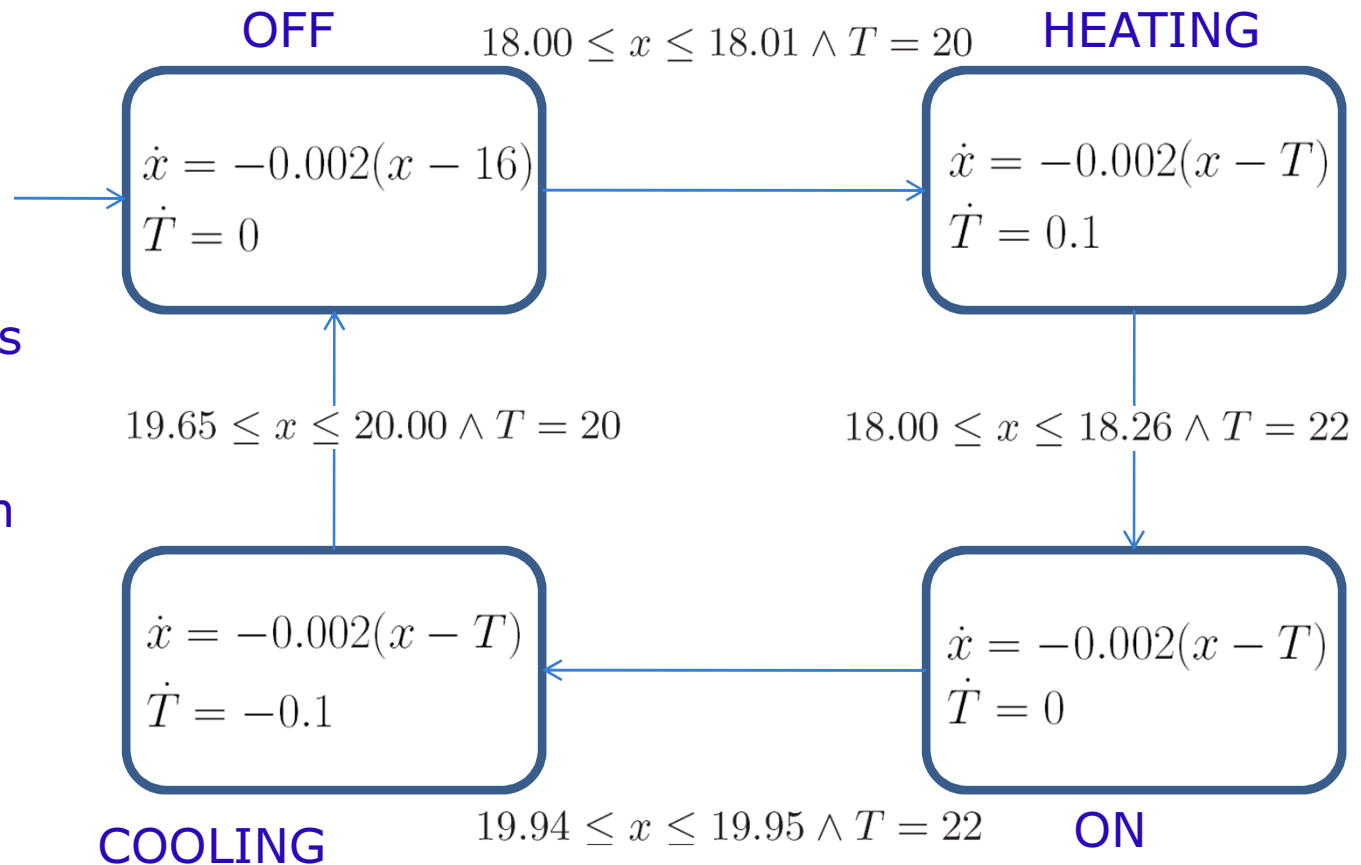


Hybrid Automata

Dynamics: Location \rightarrow Ordinary Differential Equations (ODEs)
 Switching Logic \subseteq Location \times Location \rightarrow Predicates

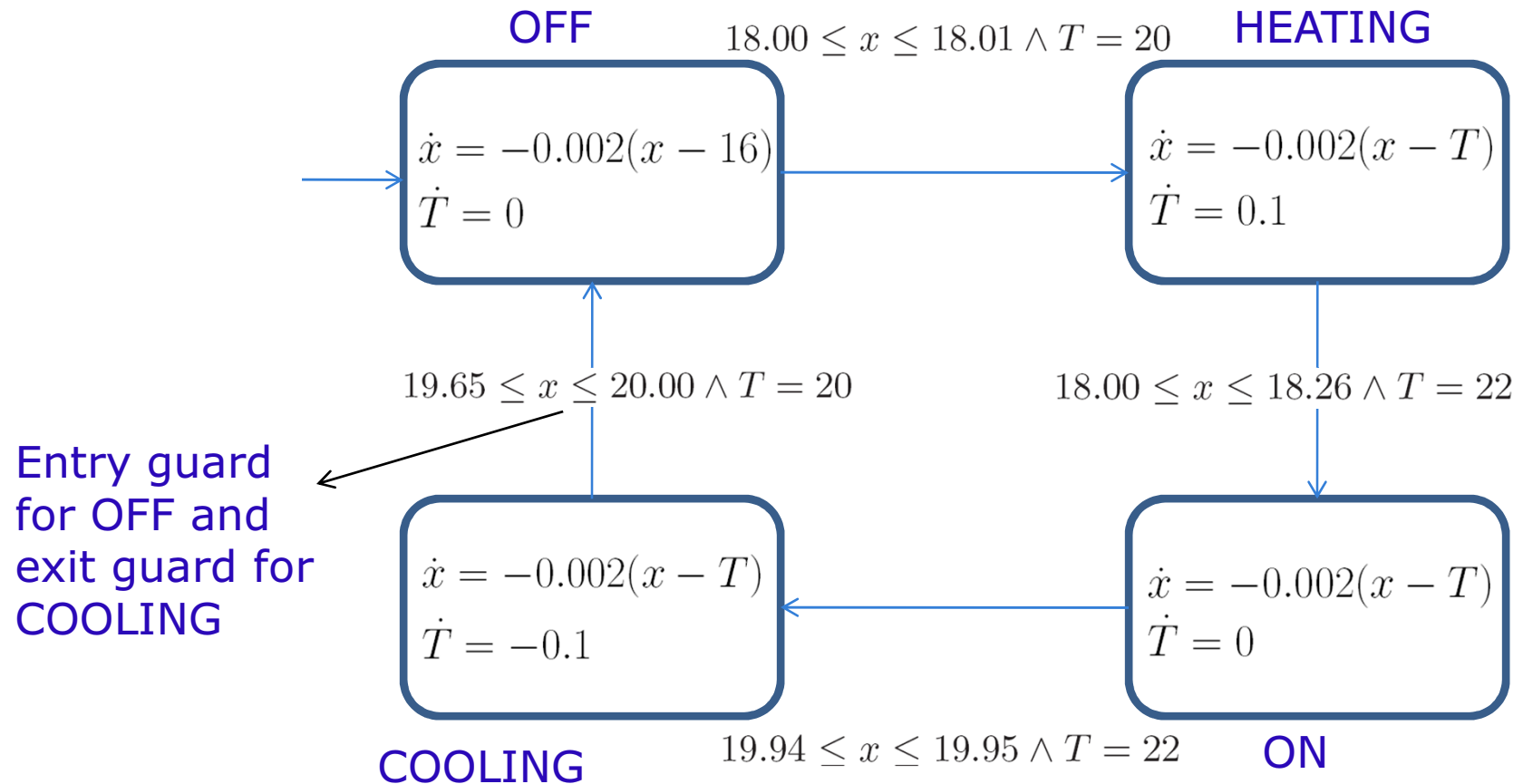
Guards are predicates on the continuous variables.

A predicate on continuous variable represents a set of states.



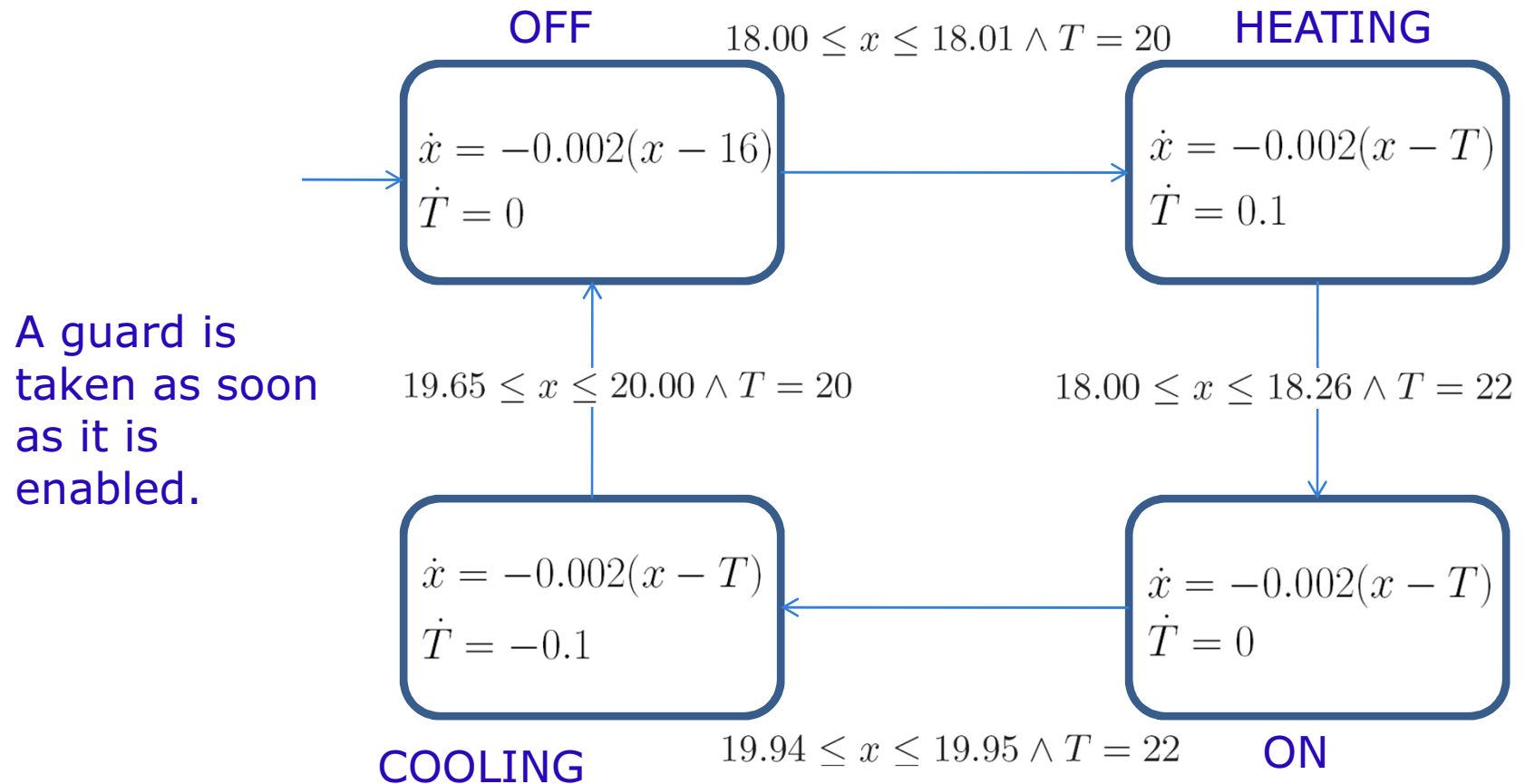
Hybrid Automata

Dynamics: Location \rightarrow Ordinary Differential Equations (ODEs)
 Switching Logic \subseteq Location \times Location \rightarrow Predicates



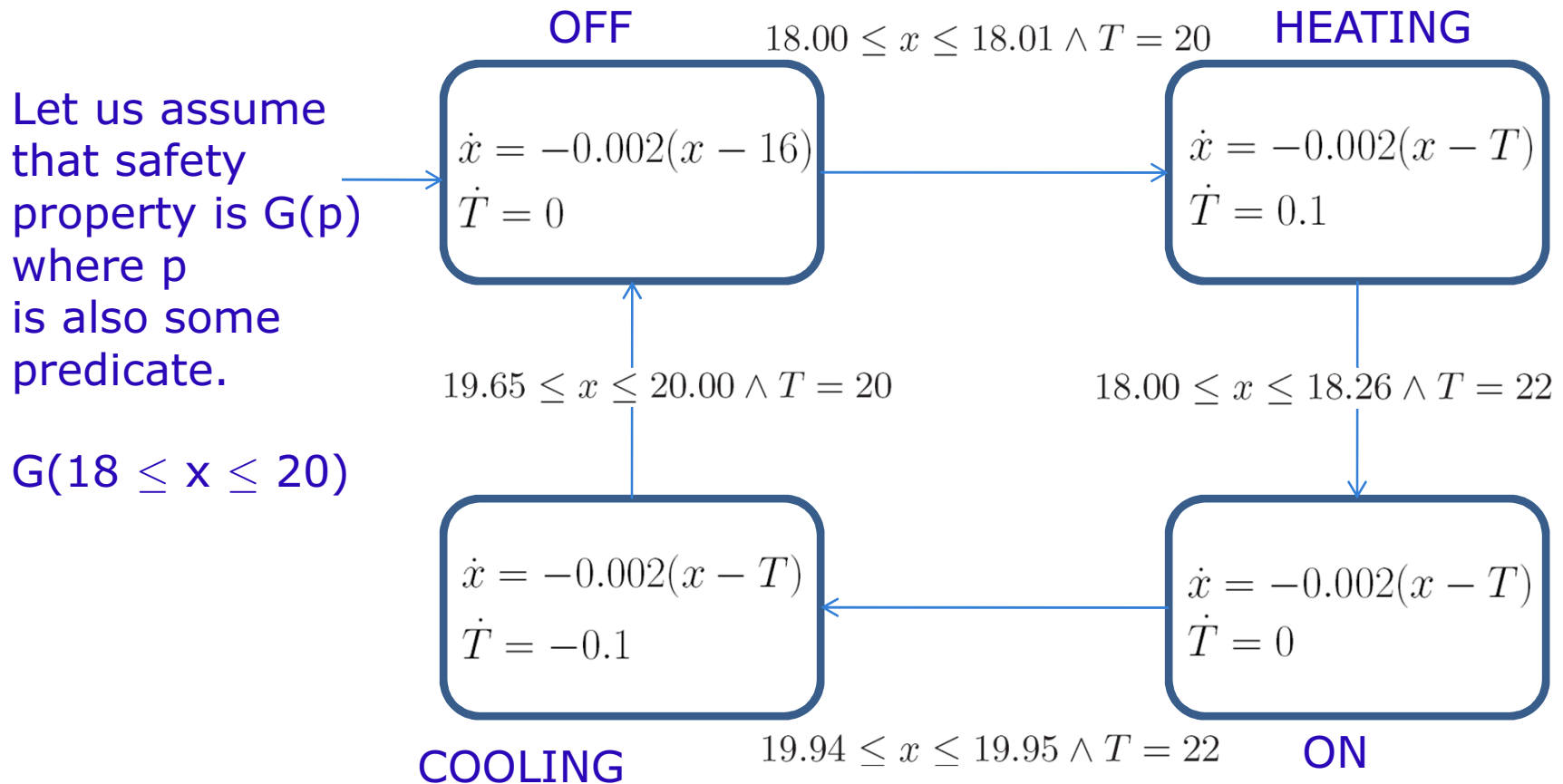
Hybrid Automata

Dynamics: Location \rightarrow Ordinary Differential Equations (ODEs)
 Switching Logic \subseteq Location \times Location \rightarrow Predicates

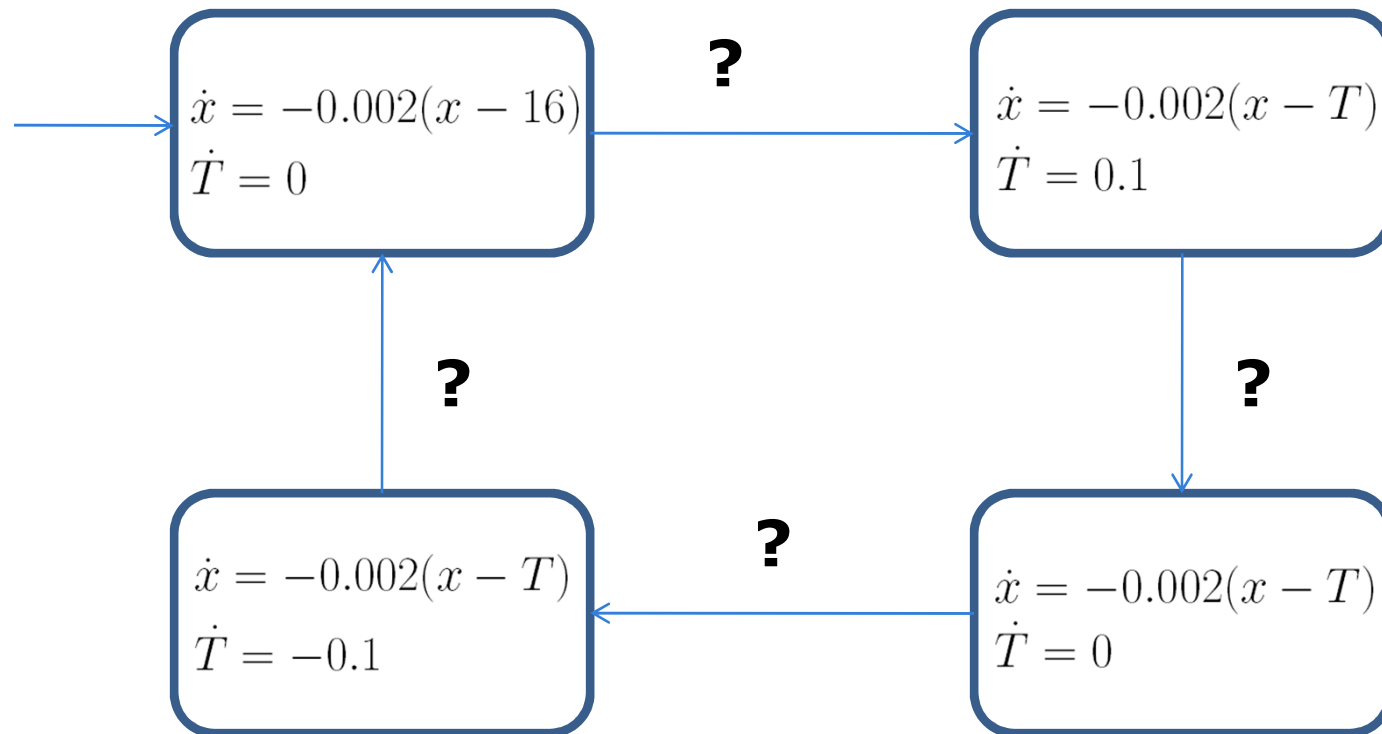


Hybrid Automata

Dynamics: Location \rightarrow Ordinary Differential Equations (ODEs)
 Switching Logic \subseteq Location \times Location \rightarrow Predicates



Switching Logic Synthesis Problem



- Room temperature must lie between 20 and 22 Celsius.
- Minimize switching between the modes

Switching Logic Synthesis Problem

Given ODEs describing dynamics of each mode of a CPS, synthesize the switching logic such that the system

- Meets **safety** objectives
- Meets some **performance** objectives

Related Work

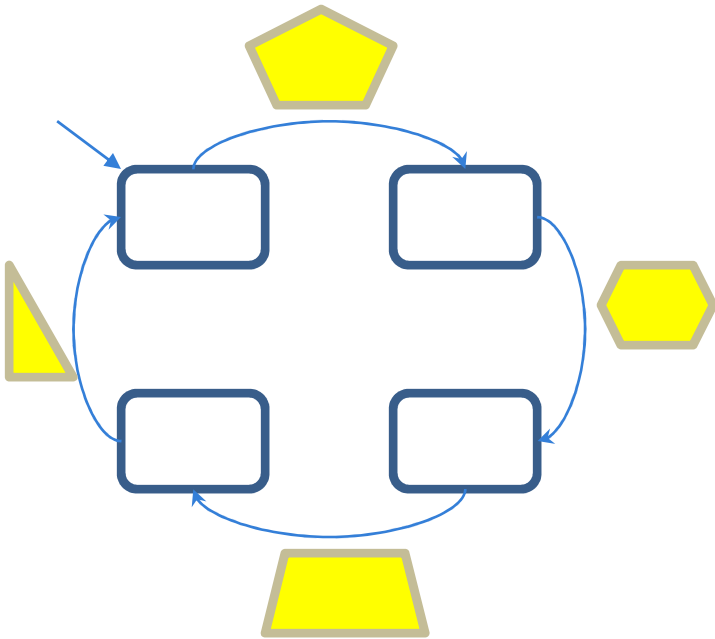
- Fixpoint computation for synthesis – compute reachable states
 - Game-theoretic approaches (Tomlin et al, Proceedings of IEEE 2000)
 - Abstraction-based reasoning (Tabuada, SCL 08; Cury et al, Automated Control 98)
 - Solving $\exists \forall$ formula (Taly et al, VMCI09)
- Numerical simulation for testing
 - Donze and Maler, HSCC 07
 - Kapinski et al, HSCC 03

SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation

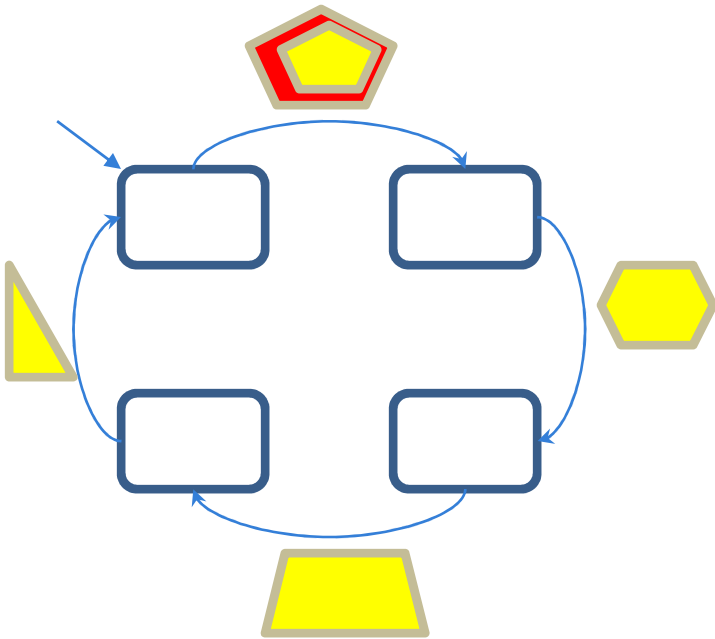
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



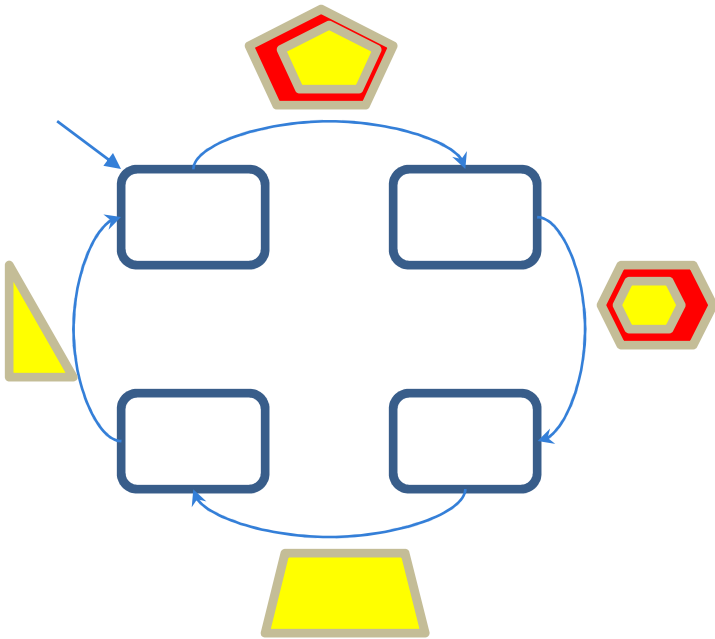
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



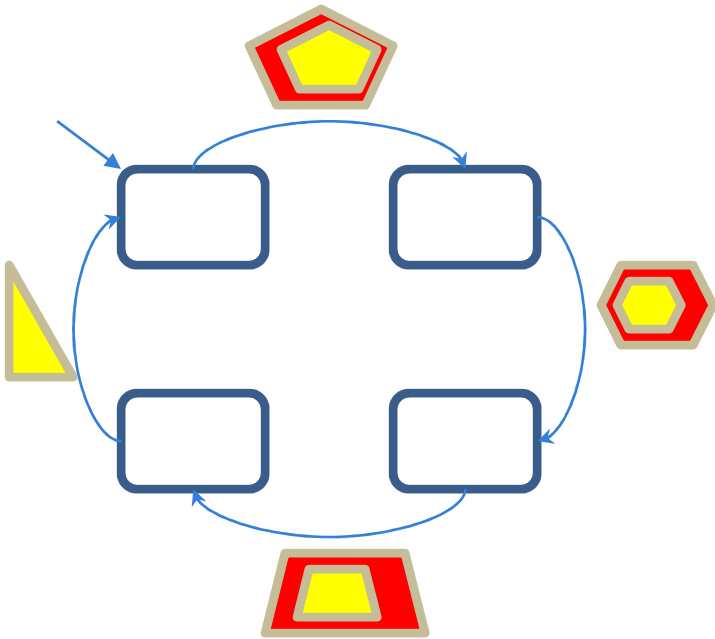
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



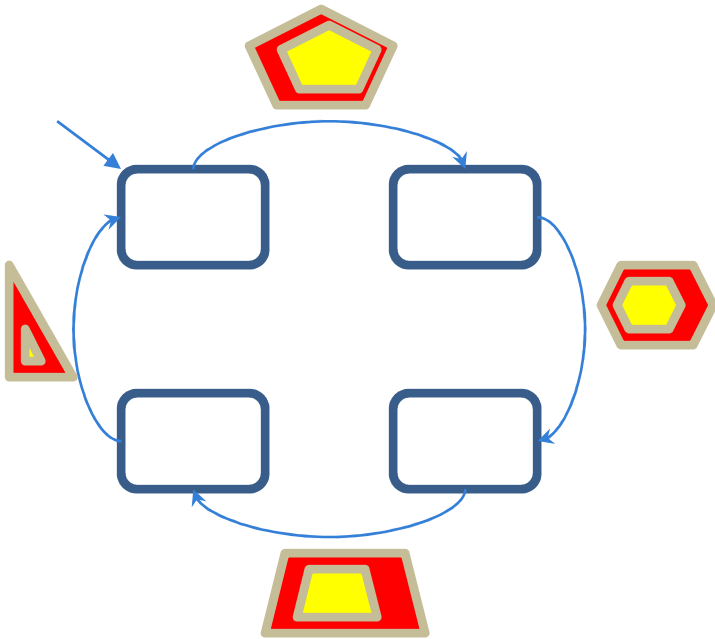
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



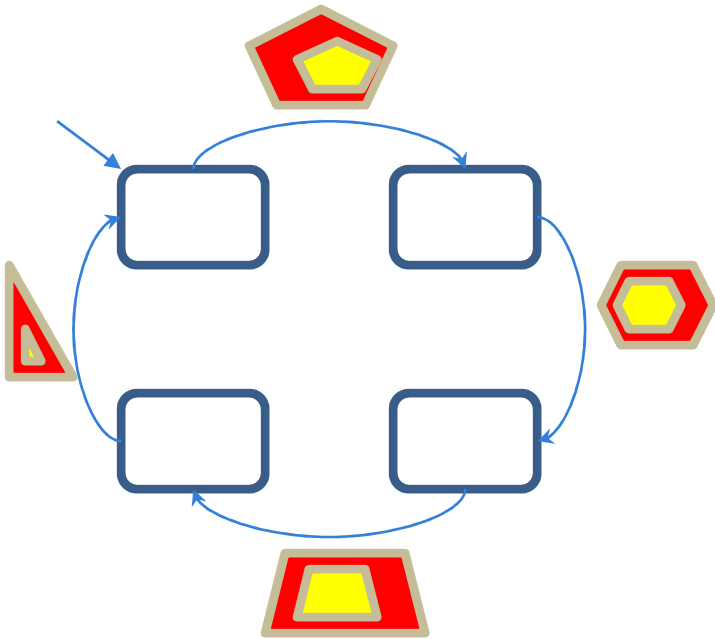
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



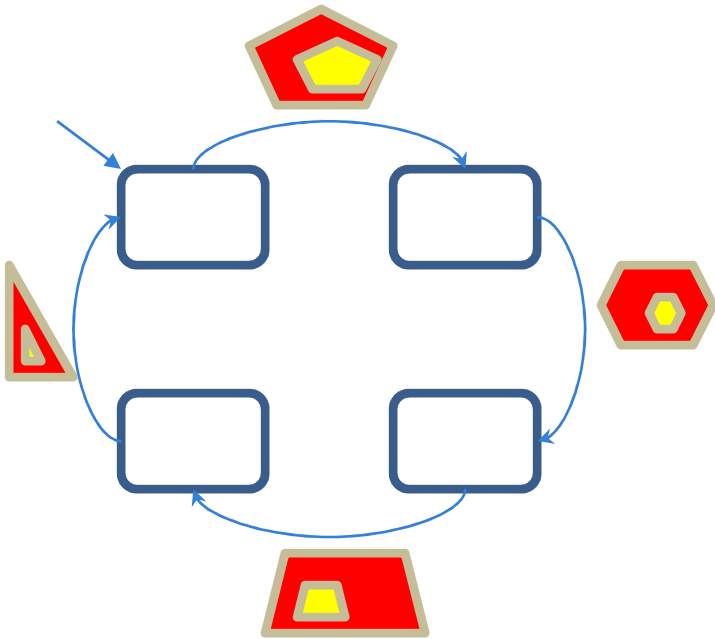
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



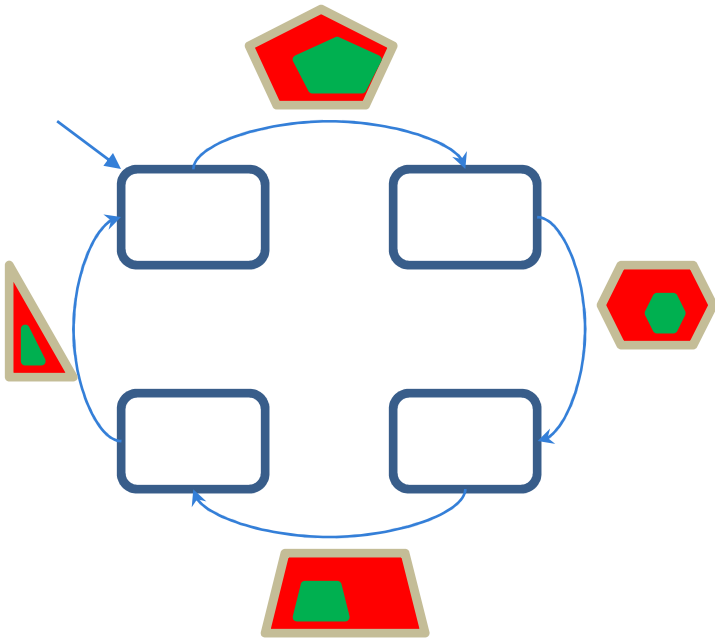
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



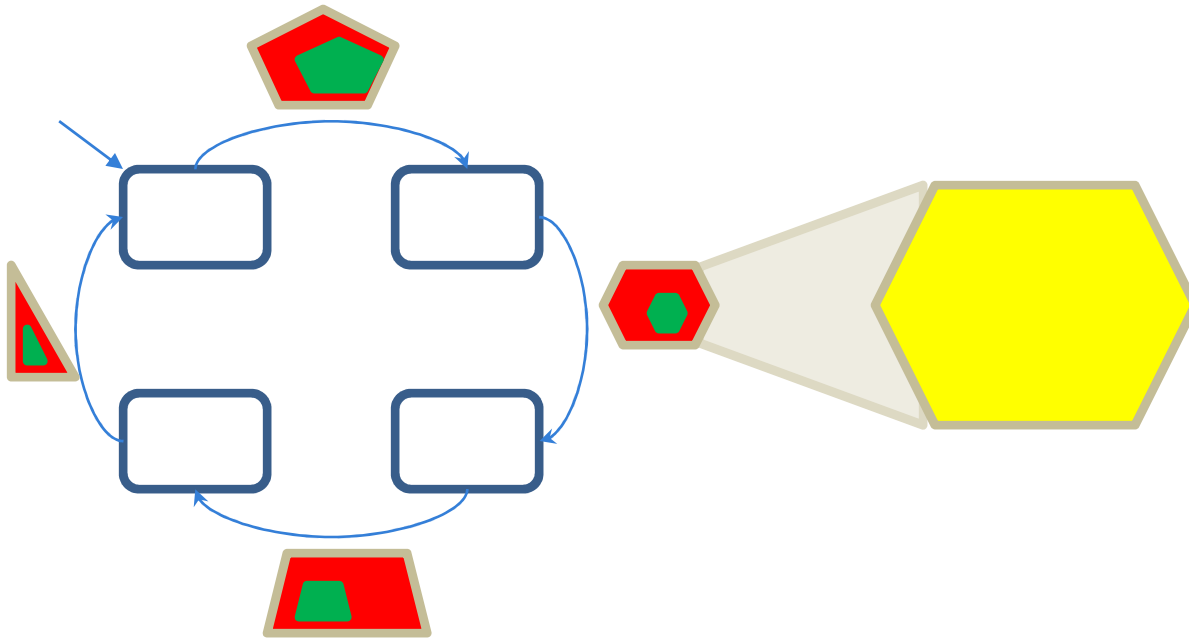
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



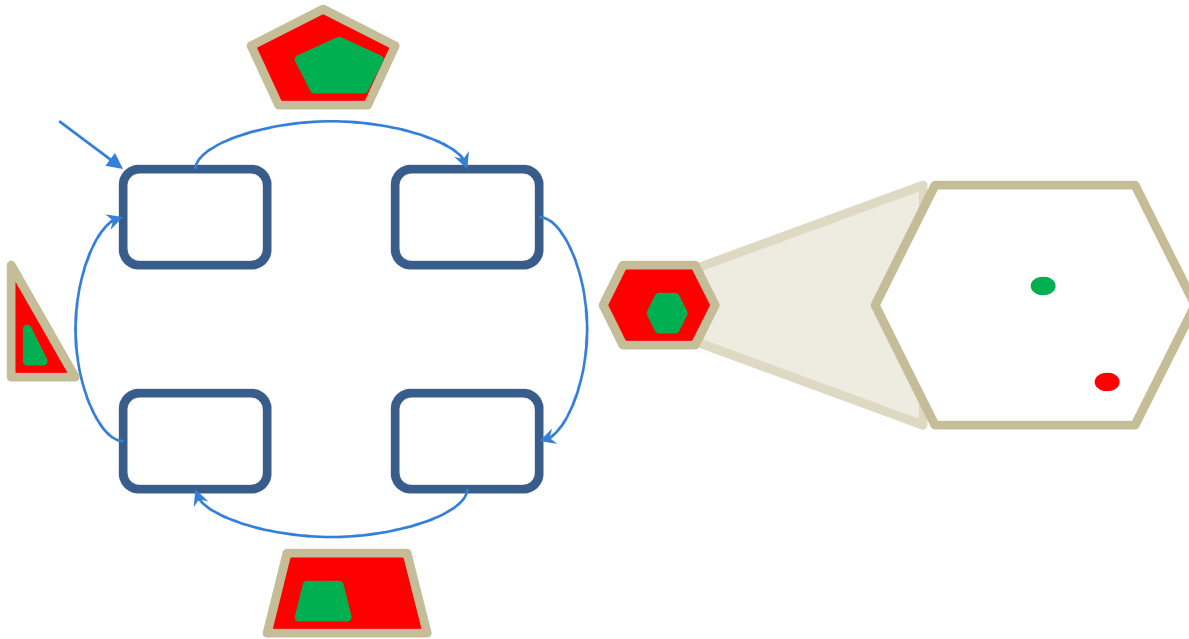
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



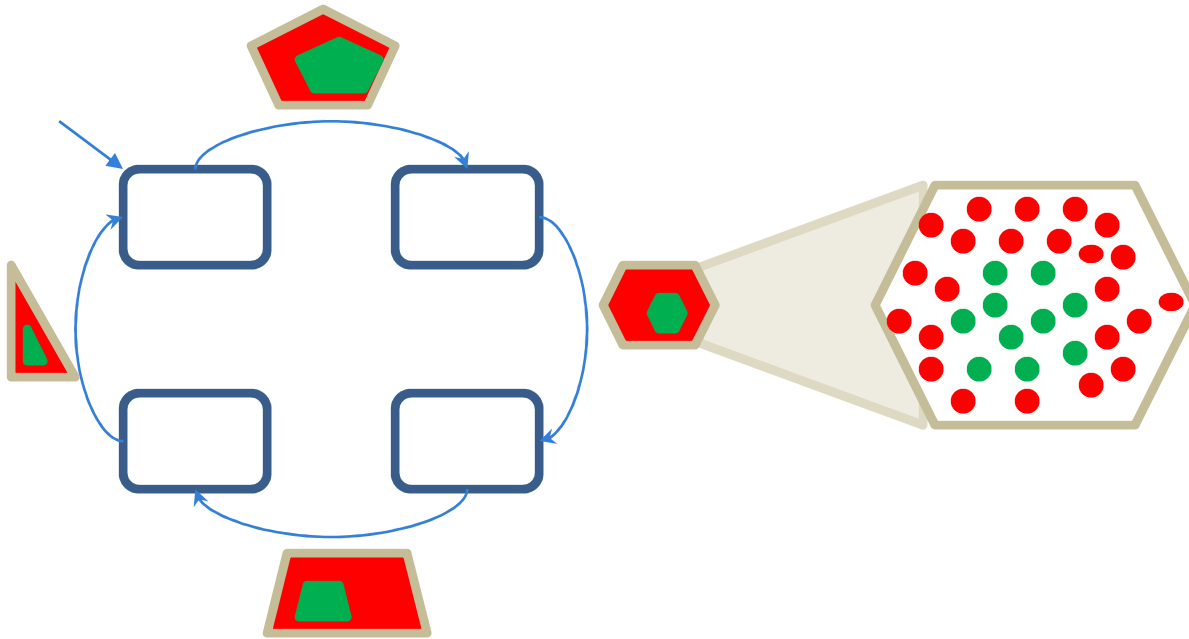
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



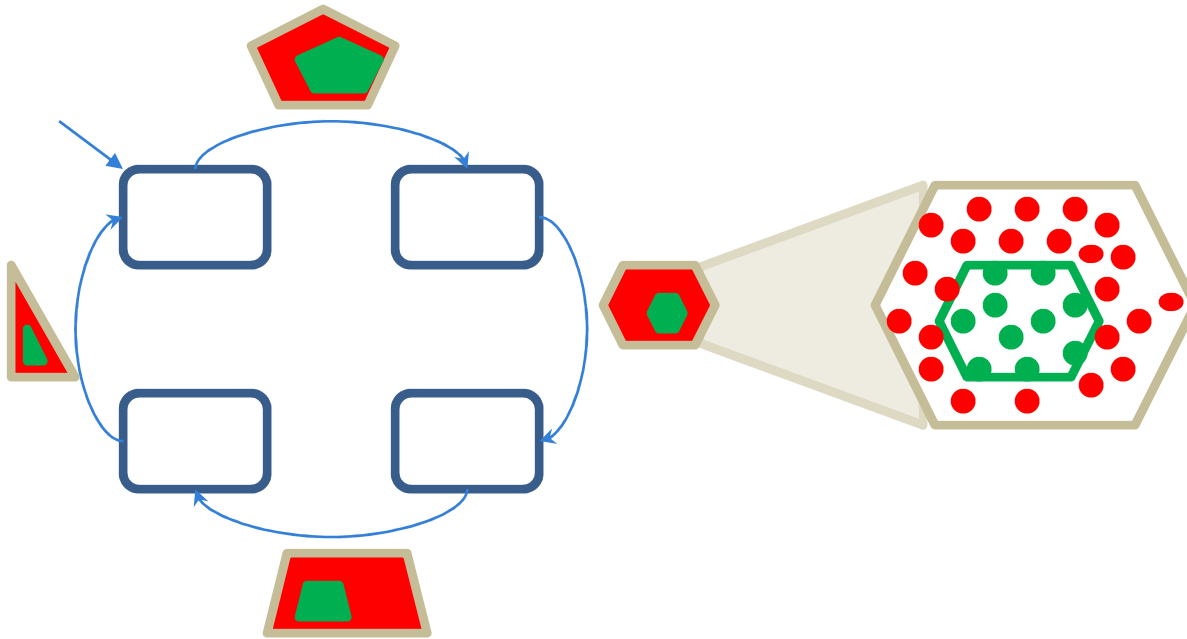
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



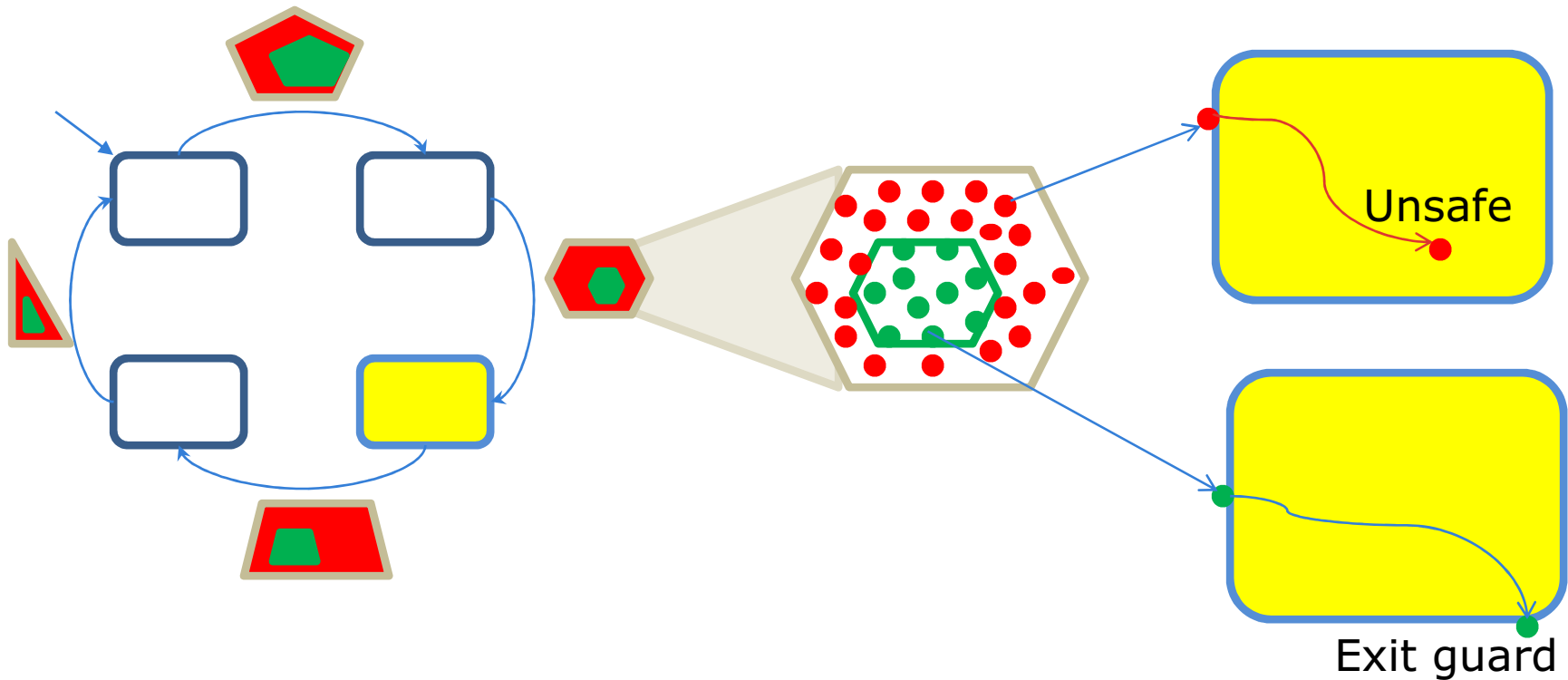
SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



SILVER for switching logic synthesis

Fixpoint computation + Sample and Learn + Numerical Simulation



Fixpoint Computation

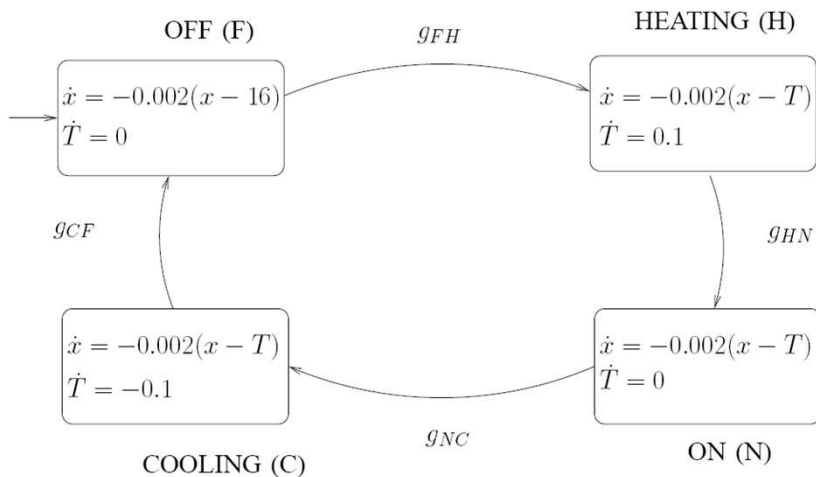
$$Mode_1, I \models \phi_S \mathbf{W} \left(\bigvee_{k \in M} g'_{1k} \right)$$

$$Mode_i, \bigvee_{j \in M} g'_{ji} \models \phi_S \mathbf{W} \left(\bigvee_{k \in M} g'_{ik} \right) \text{ for } i = 1..k$$

Fixpoint Computation

$$Mode_1, I \models \phi_S \mathbf{W} \left(\bigvee_{k \in M} g'_{1k} \right)$$

$$Mode_i, \bigvee_{j \in M} g'_{ji} \models \phi_S \mathbf{W} \left(\bigvee_{k \in M} g'_{ik} \right) \text{ for } i = 1..k$$



$$F, I \models \phi_S \mathbf{W} g_{FH}$$

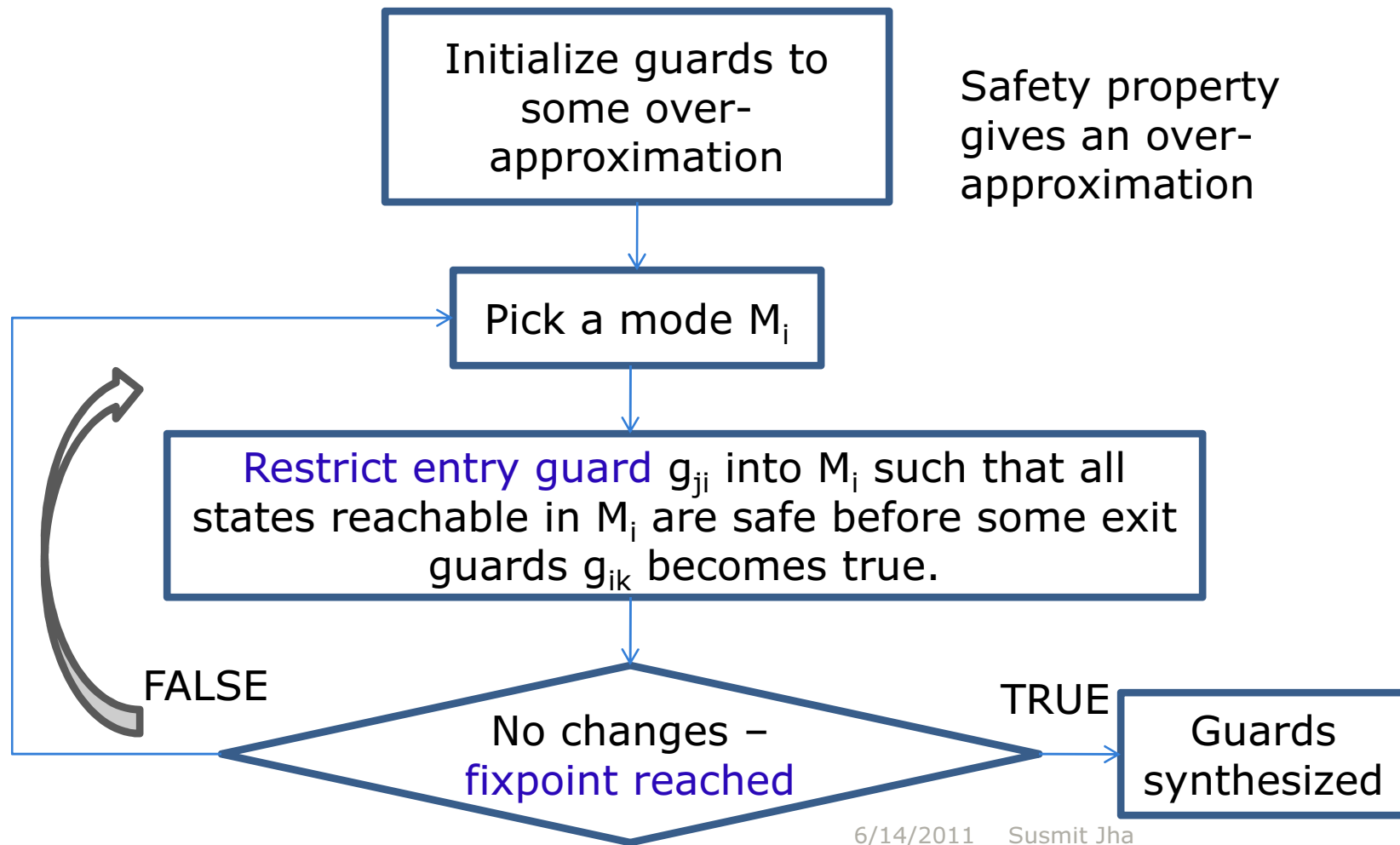
$$F, g_{CF} \models \phi_S \mathbf{W} g_{FH}$$

$$H, g_{FH} \models \phi_S \mathbf{W} g_{HN}$$

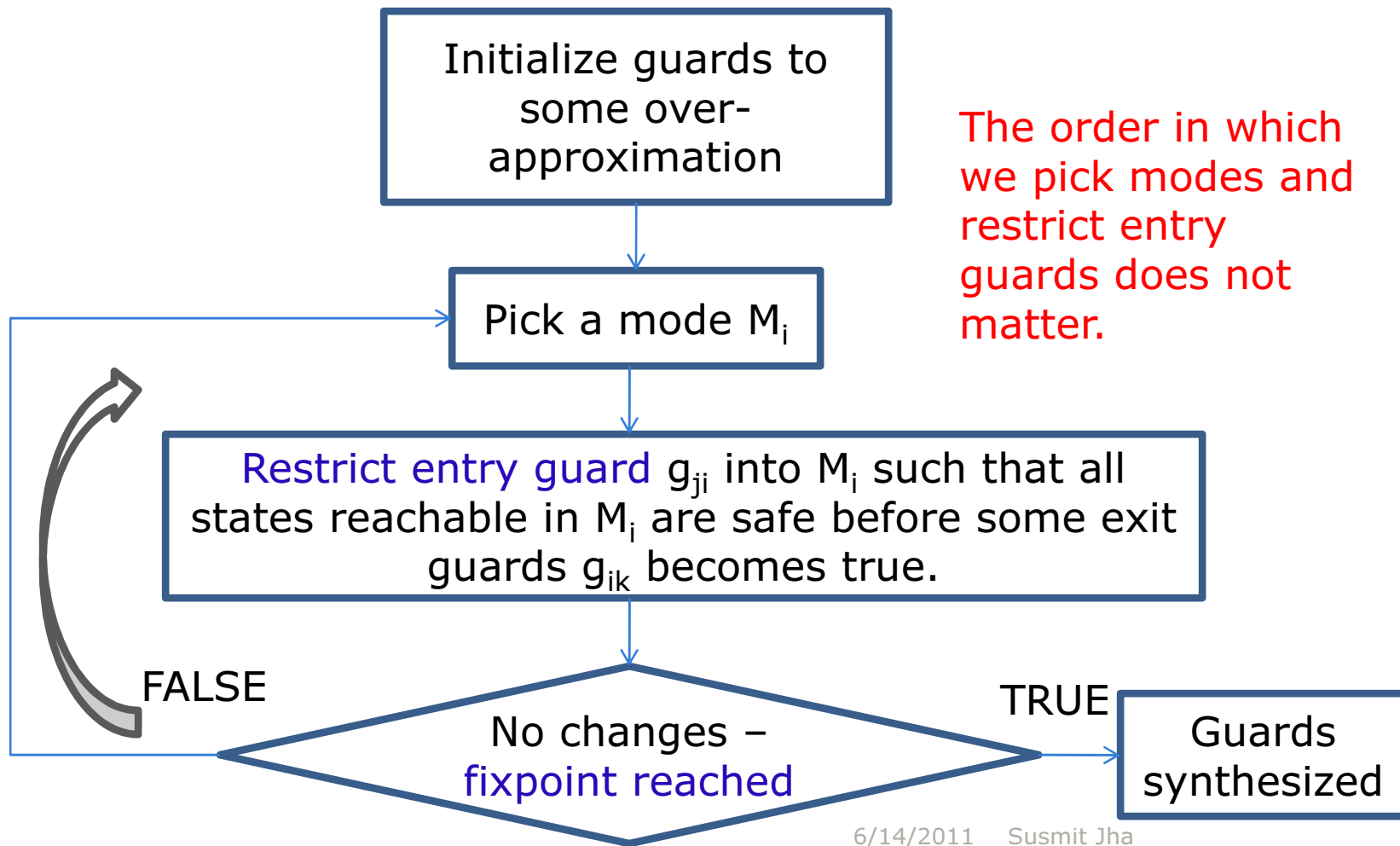
$$N, g_{HN} \models \phi_S \mathbf{W} g_{NC}$$

$$C, g_{NC} \models \phi_S \mathbf{W} g_{CF}$$

Fixpoint Computation



Fixpoint Computation



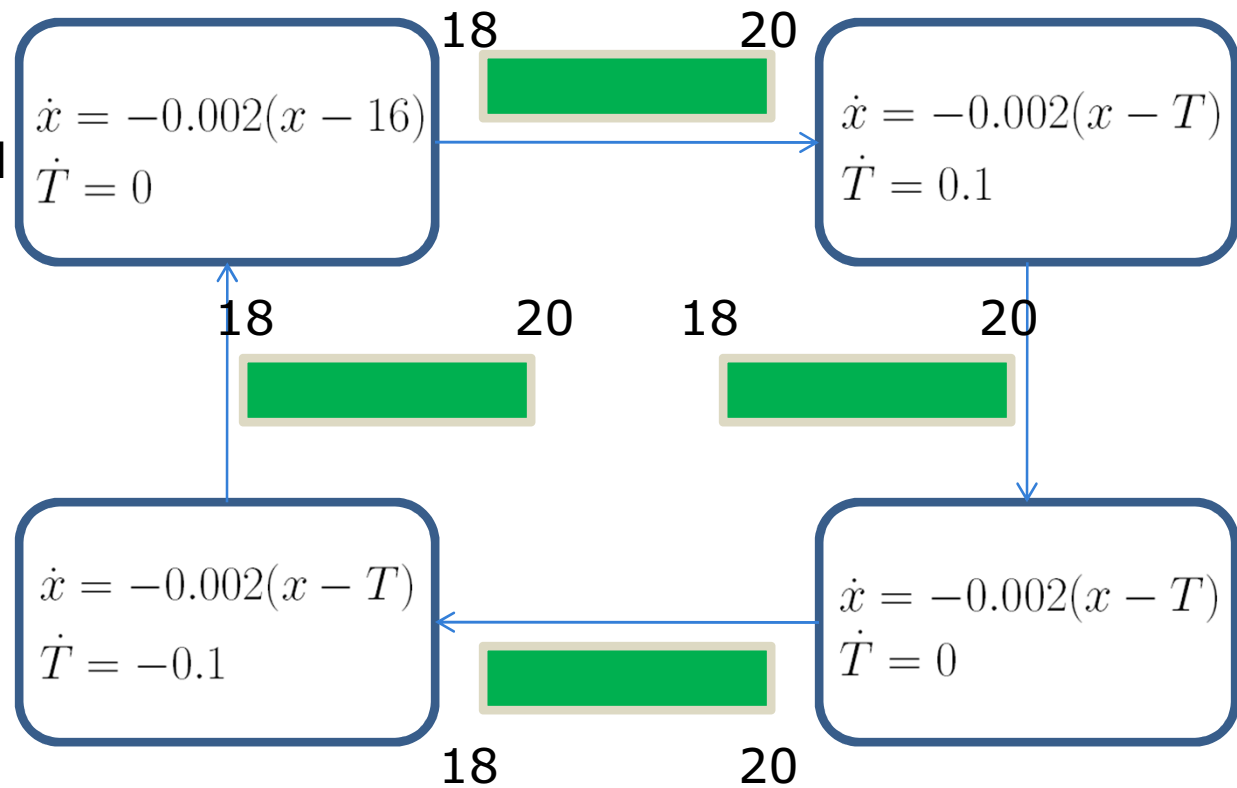
Toy Example: Thermostat

Assumptions

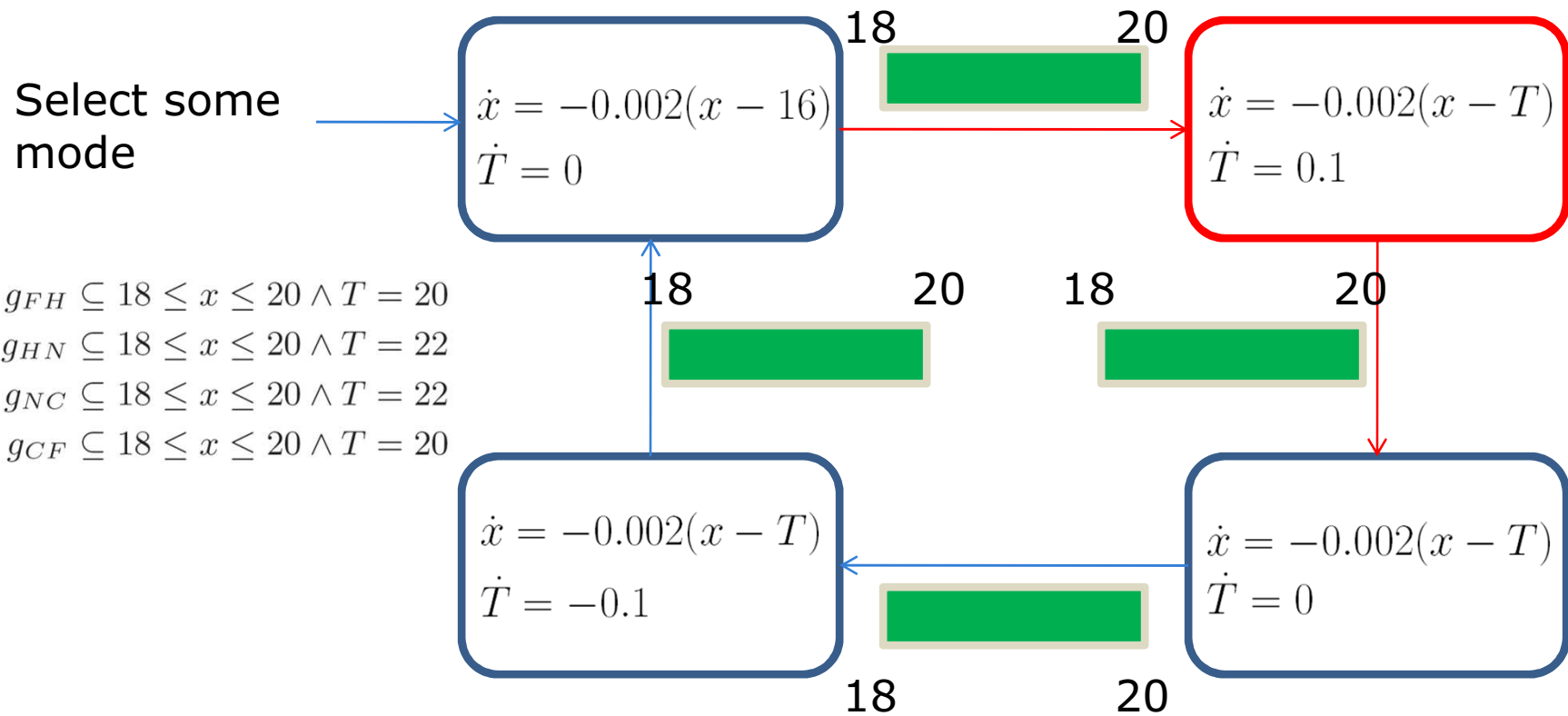
Guards are assumed to be intervals;

Precision till second place of decimal

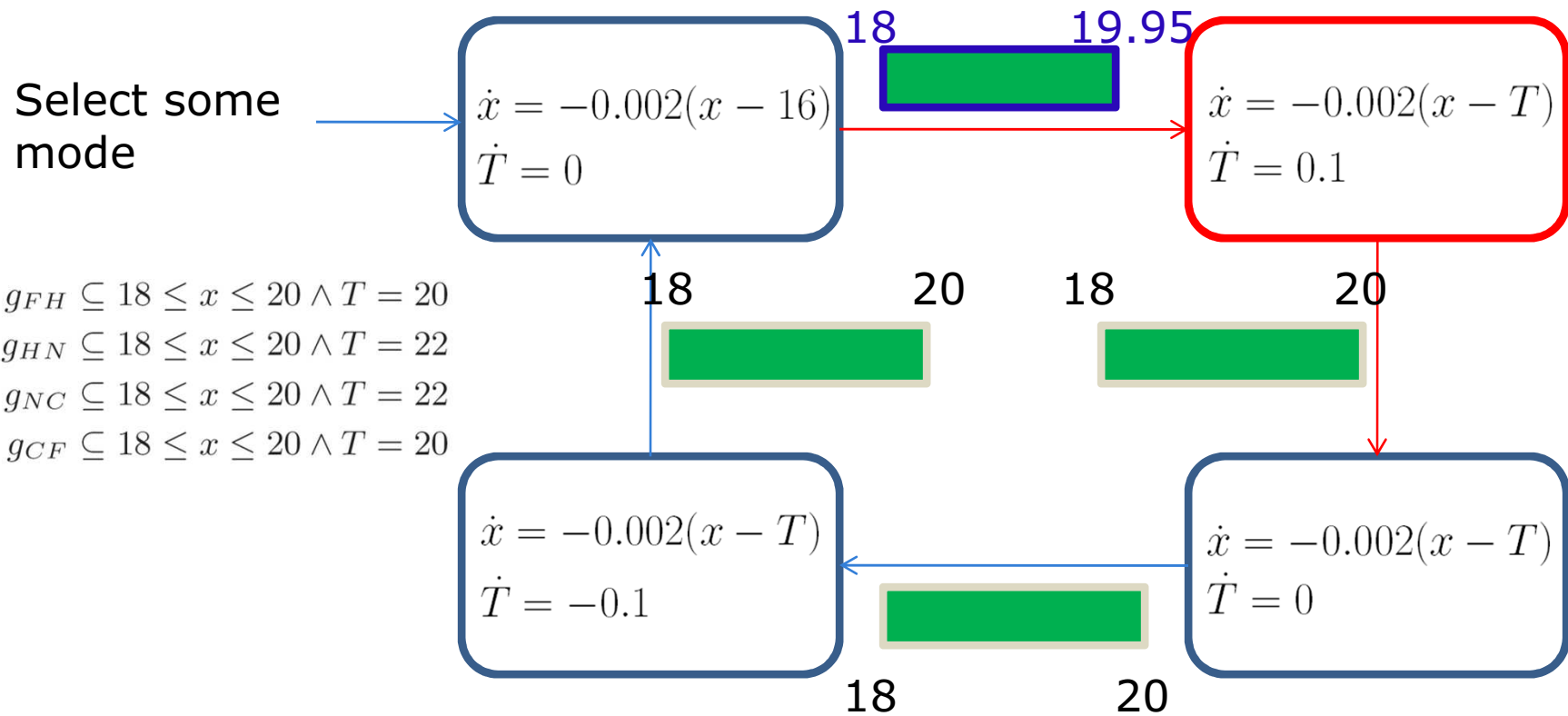
- $g_{FH} \subseteq 18 \leq x \leq 20 \wedge T = 20$
- $g_{HN} \subseteq 18 \leq x \leq 20 \wedge T = 22$
- $g_{NC} \subseteq 18 \leq x \leq 20 \wedge T = 22$
- $g_{CF} \subseteq 18 \leq x \leq 20 \wedge T = 20$



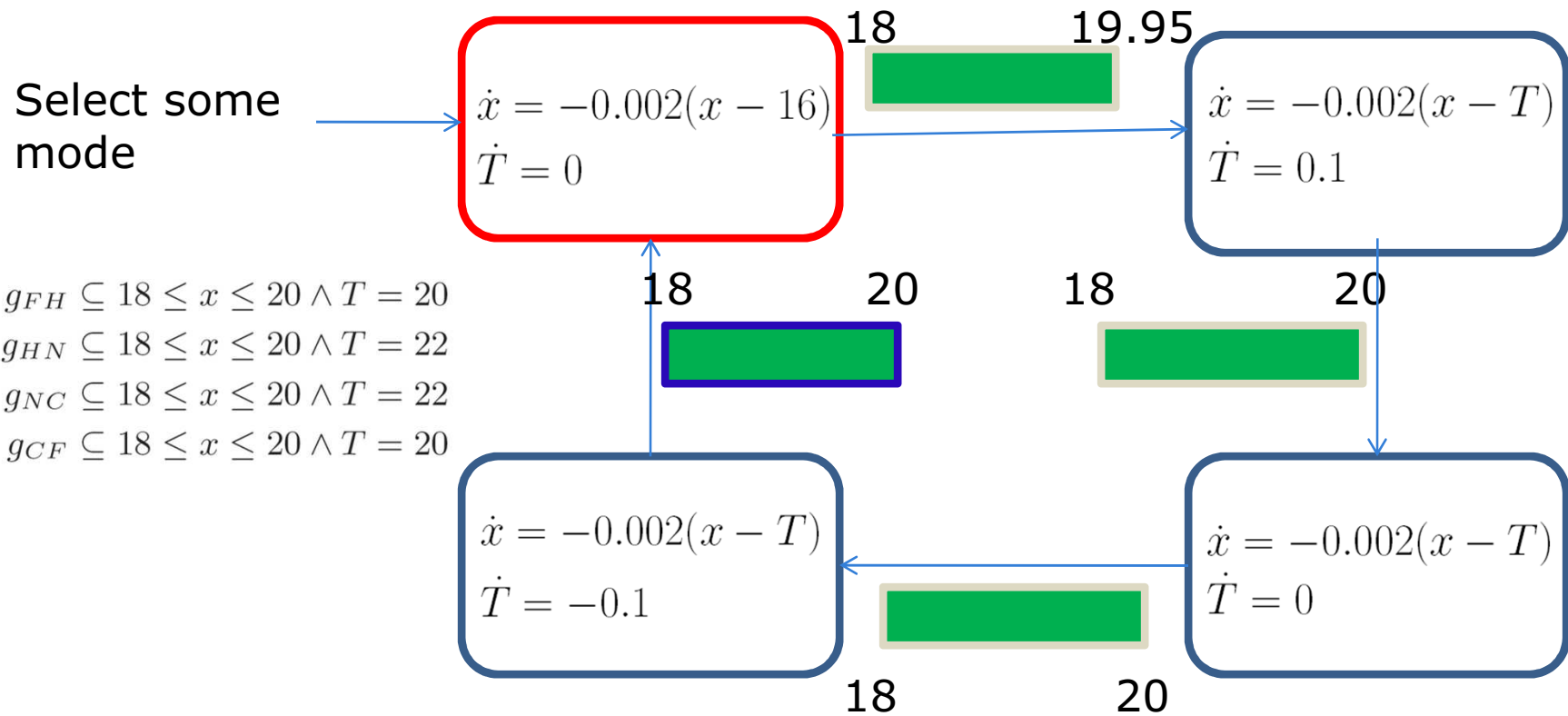
Toy Example: Thermostat



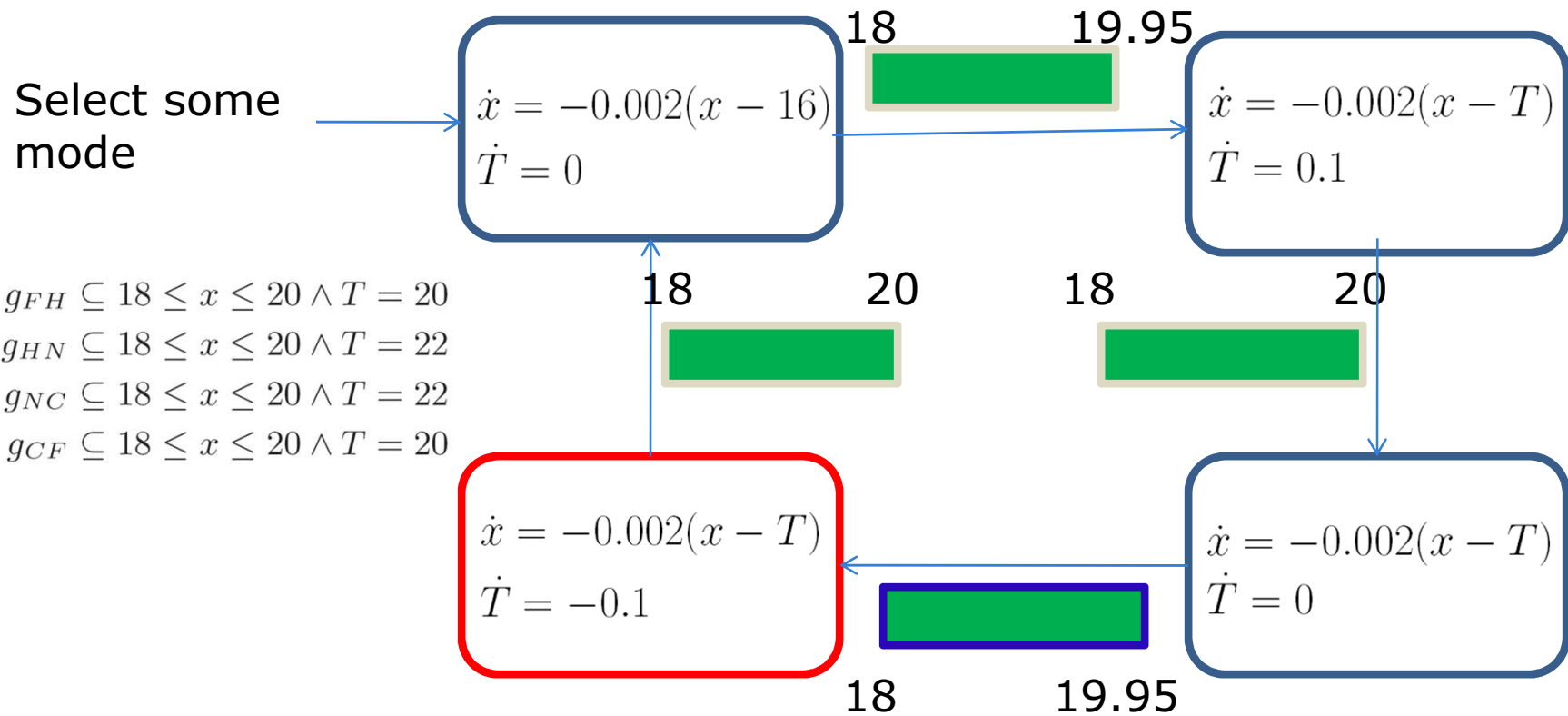
Toy Example: Thermostat



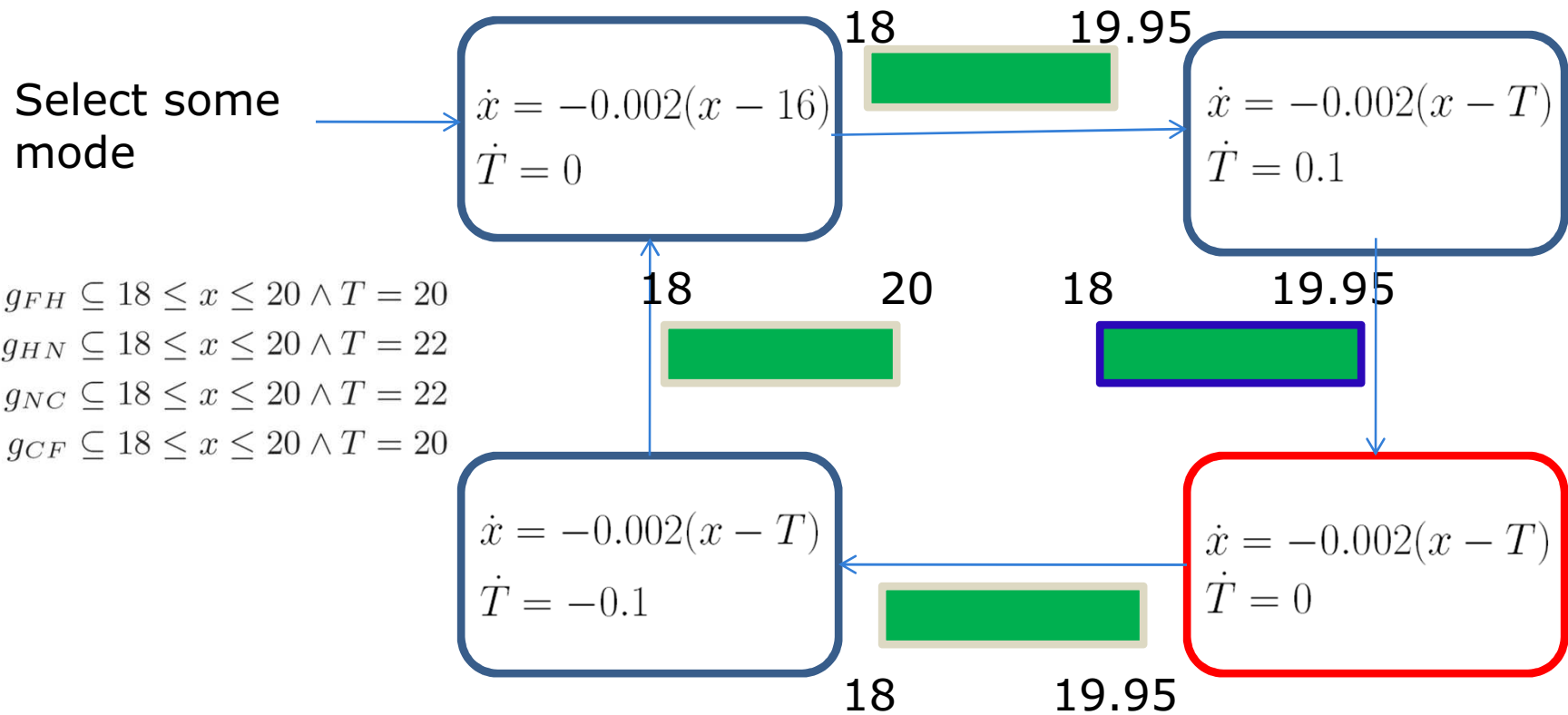
Toy Example: Thermostat



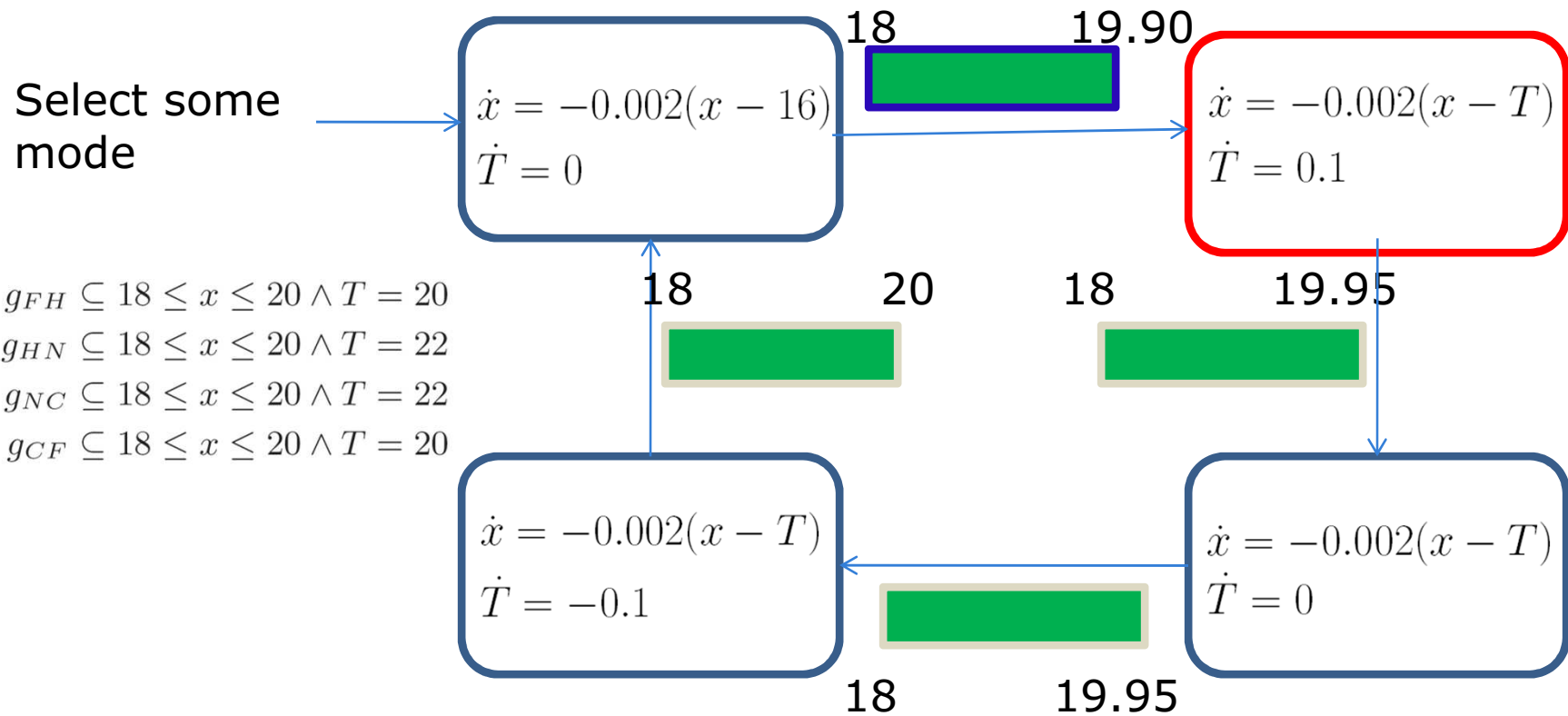
Toy Example: Thermostat



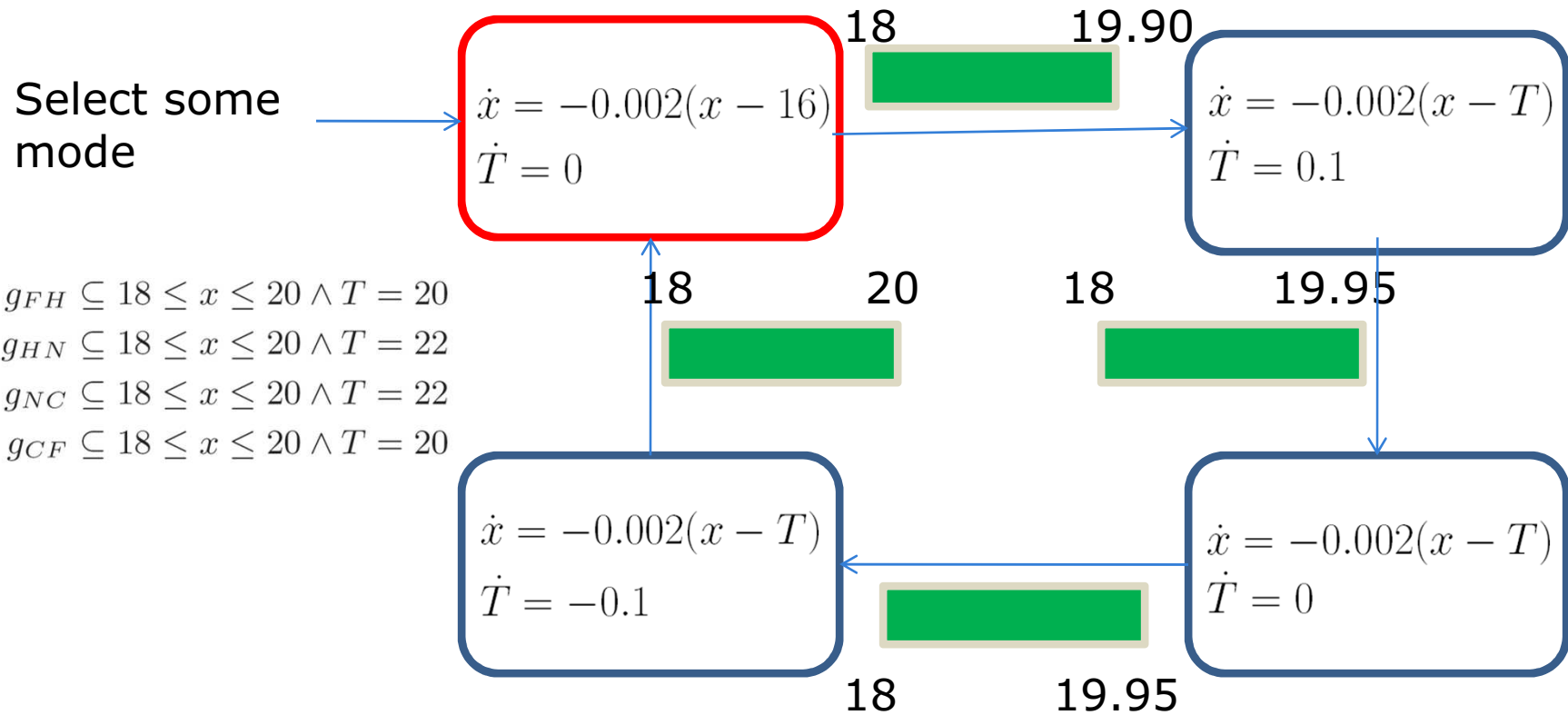
Toy Example: Thermostat



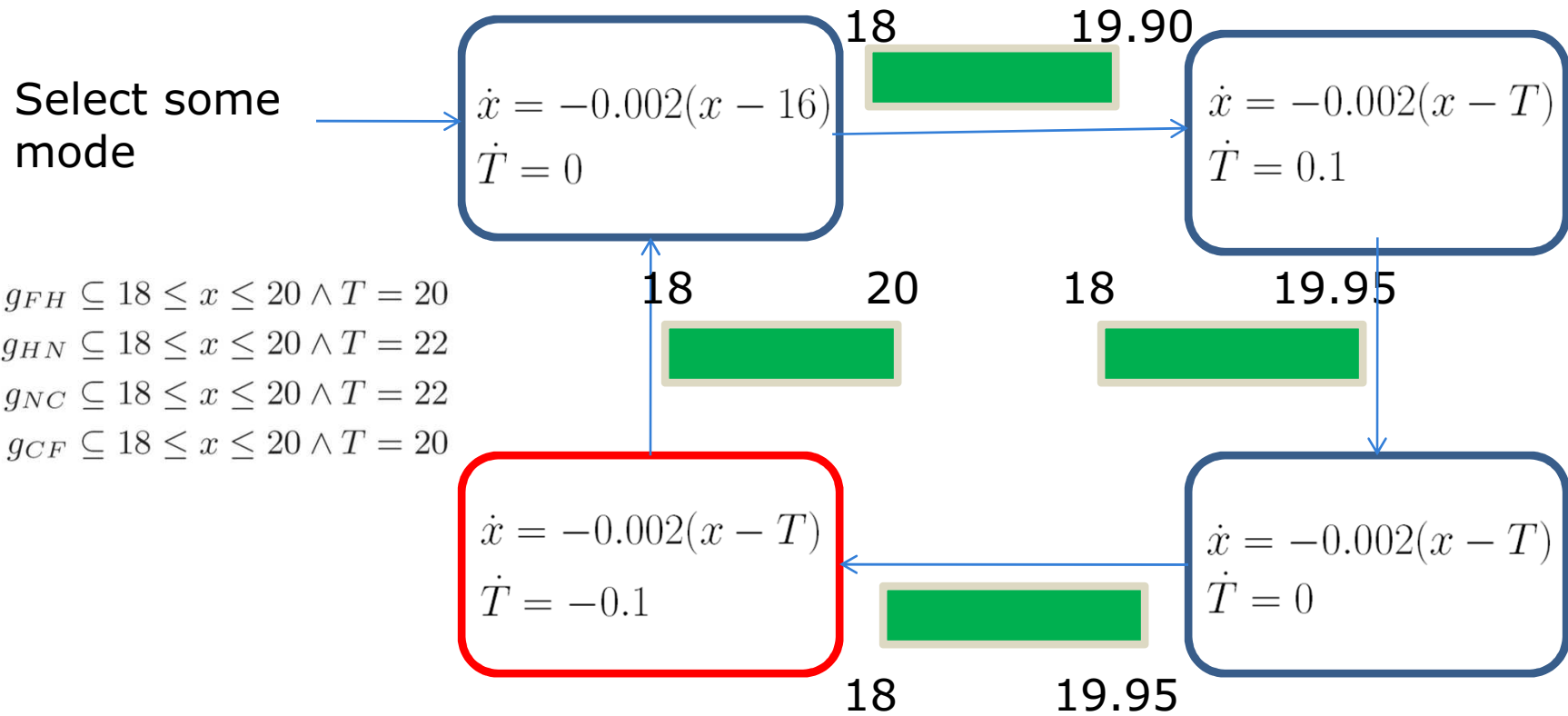
Toy Example: Thermostat



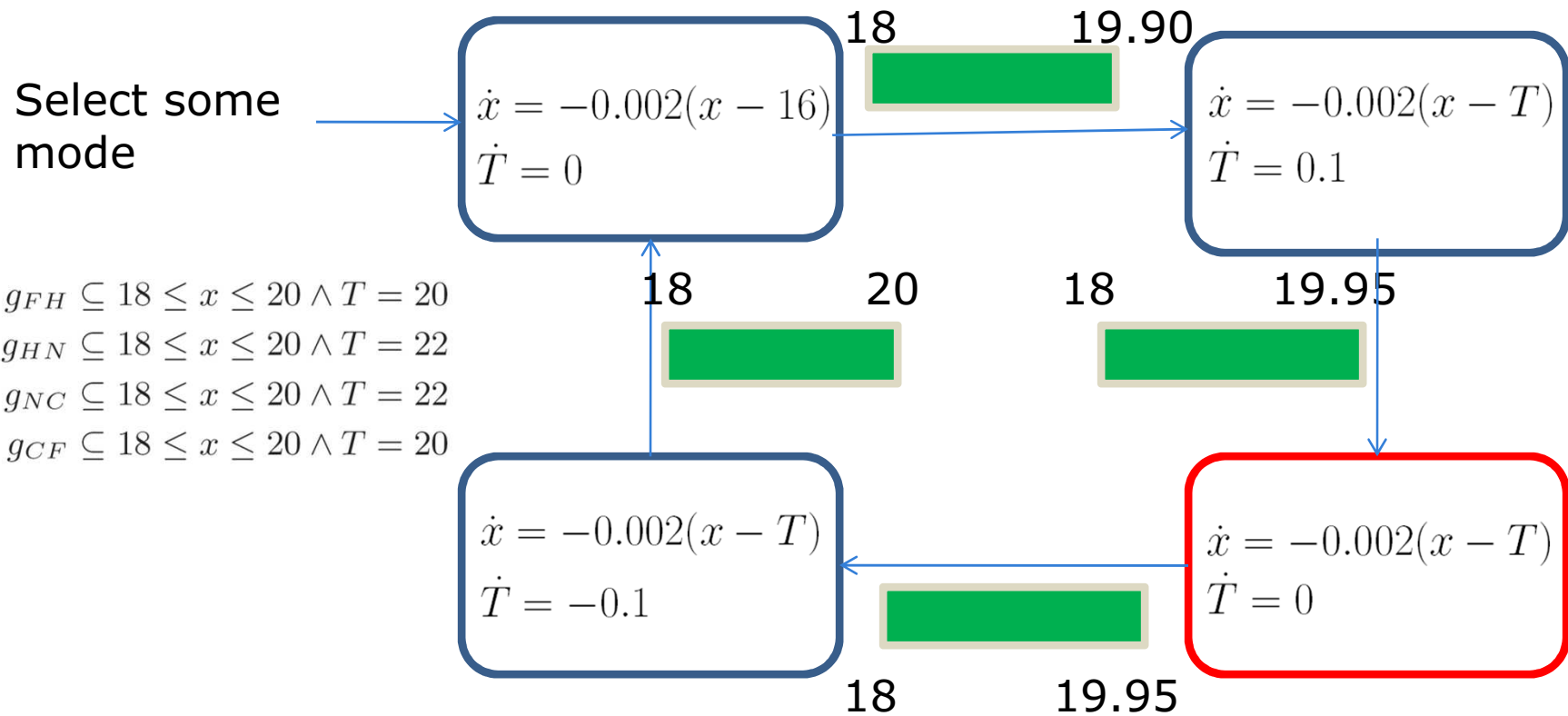
Toy Example: Thermostat



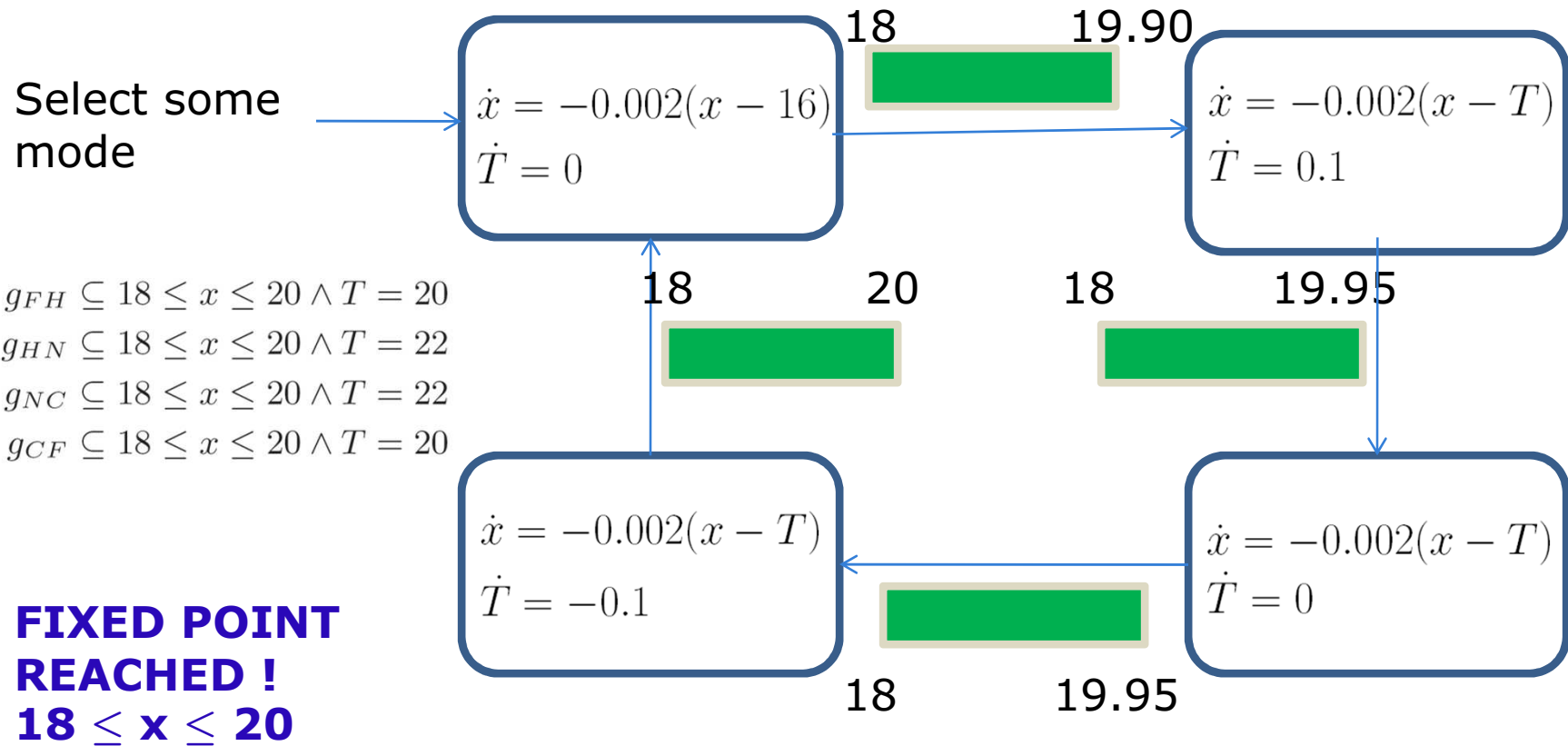
Toy Example: Thermostat



Toy Example: Thermostat



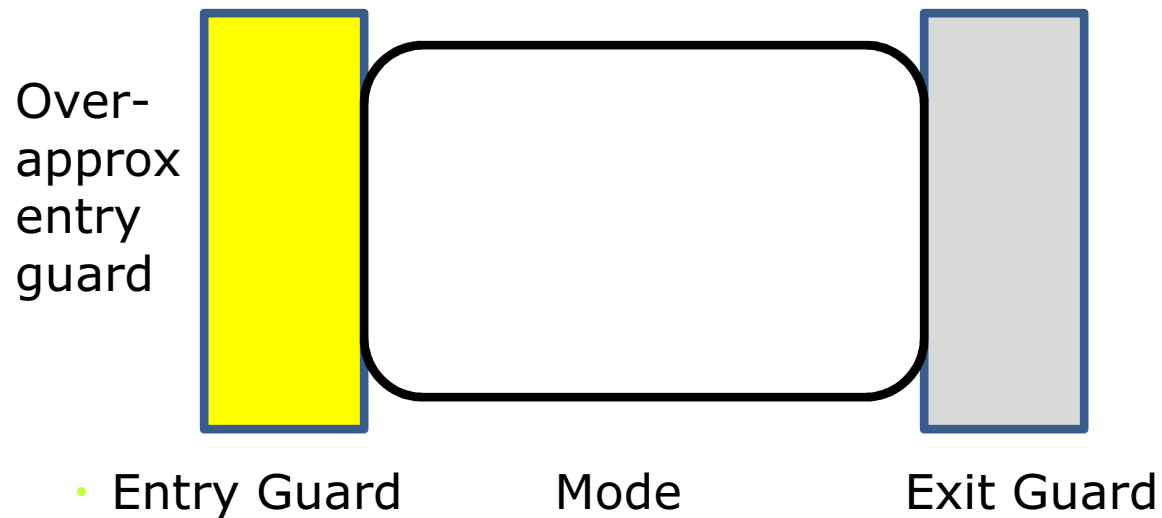
Toy Example: Thermostat



Learning guards

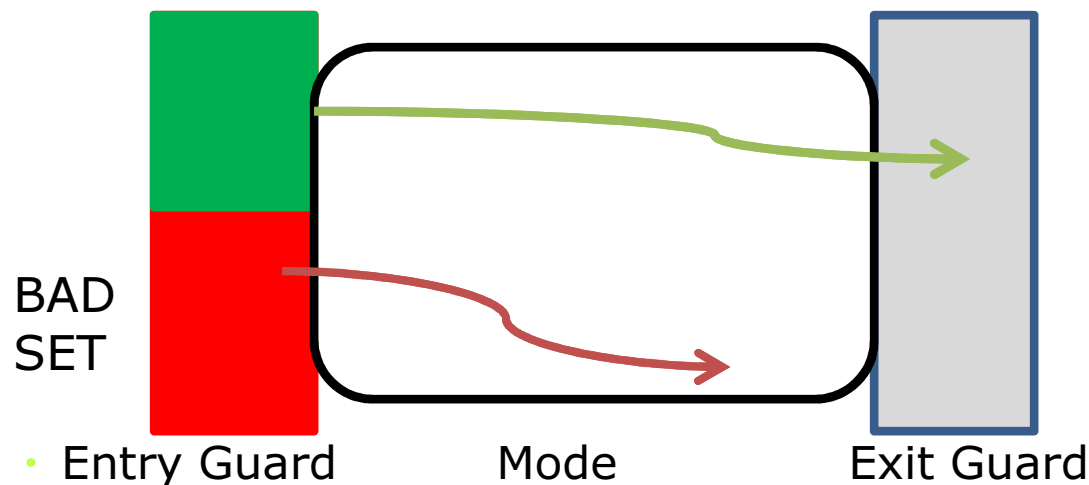
- **Definition:** Bad set is the subset of over-approximate guard for which simulation reaches unsafe state and hence,
restricted guard = over-approx guard – Bad set
- **Assumption:** **Guards** and **bad set** discovered in each iteration of the fixpoint algorithm belongs to some class of predicates.
 - Intervals: $c1 \leq x \leq c2 \wedge c3 \leq y \leq c4$
 - Difference logic: $c1 \leq x-y \leq c2 \wedge c3 \leq y \leq c4$
 - Linear: $c1 \leq 2x+6y \leq c2 \wedge c3 \leq y \leq c4$

Learning guards



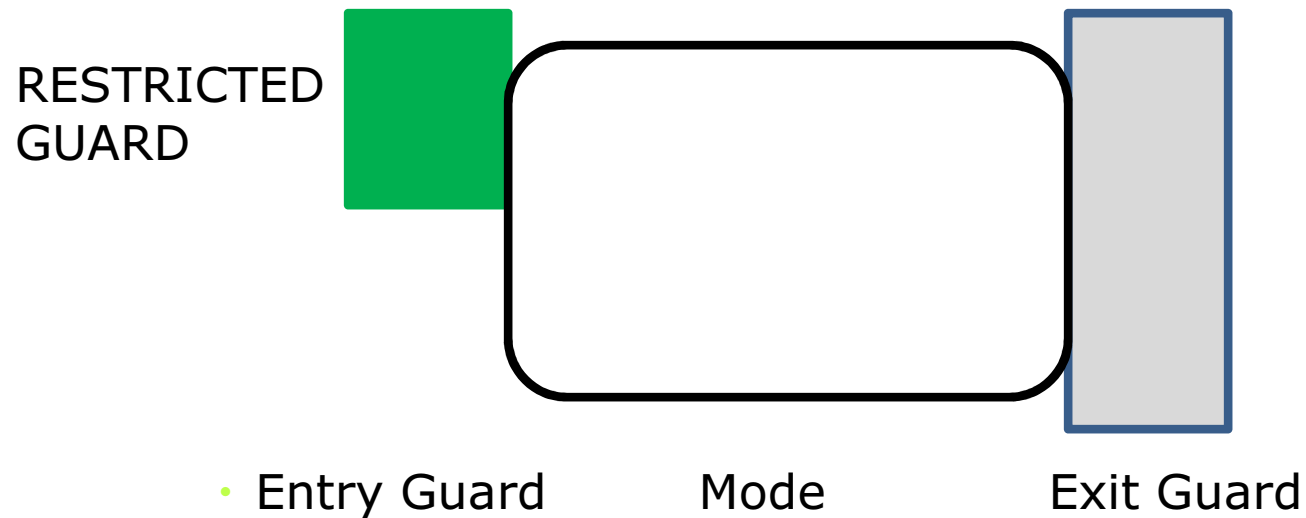
- Over-approx entry guard
 - Entry Guard
 - Exit Guard
- Restrict entry guard so that all states reachable in mode are safe

Learning guards



- Either unsafe state or exit guard reached first.
- Numerical simulation used as a black-box.
- Our technique naturally inherits all the limitations of numerical simulation.

Learning guards



Restricted Entry Guard = = over-approx guard - Bad set

Learning guards

Order of number of simulations needed to learn a predicate if predicate is:

- Conjunction of intervals: $\text{poly}(\log R, d)$
- Linearly independent linear inequalities: $\text{poly}(\log R, d, n)$
- Conjunction of arbitrary linear constraints: exponential in d
(learning convex polygons is NP-hard – COLT94,98)

R is range of variables, d is number of variables, n is the number of constraints.

Toy Example: Thermostat

Guard and bad-set is assumed to be an interval

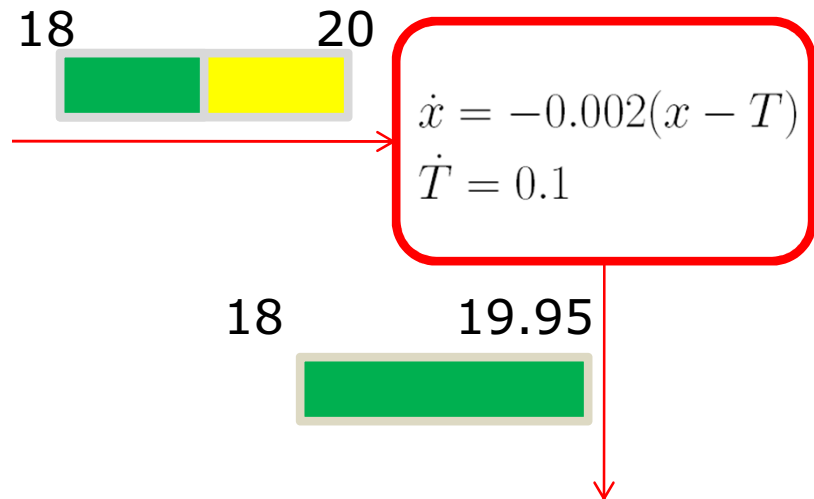
For intervals, we learn the true guards from the simulation oracle doing a binary search.

Simulate with initial condition

X=20 : Unsafe

X=18 : Safe

X=19 : Safe



Toy Example: Thermostat

Guard and bad-set is assumed to be an interval

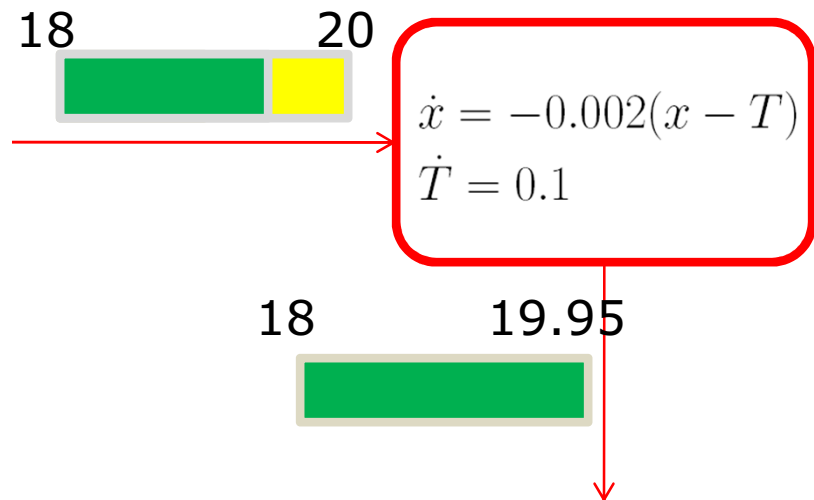
For intervals, we learn the true guards from the simulation oracle doing a binary search.

Simulate with initial condition

X=20 : Unsafe

X=19 : Safe

X=19.5 : Safe



Toy Example: Thermostat

Guard and bad-set is assumed to be an interval

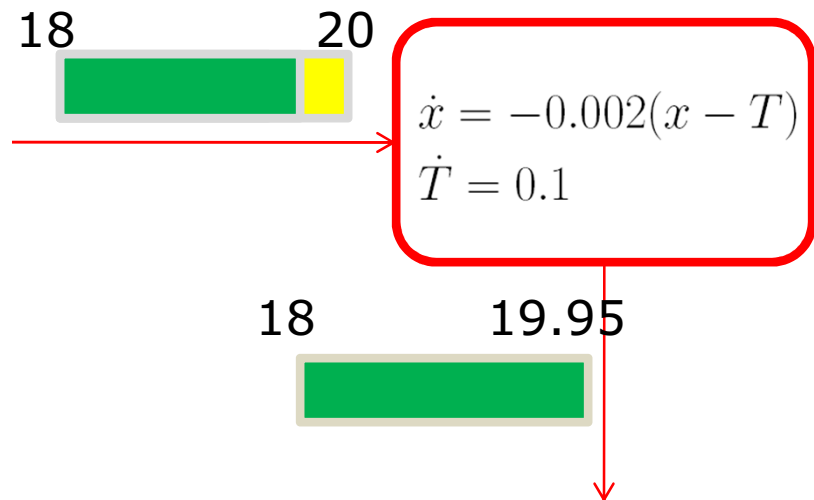
For intervals, we learn the true guards from the simulation oracle doing a binary search.

Simulate with initial condition

X=20 : Unsafe

X=19.5 : Safe

X=19.75 : Safe



Toy Example: Thermostat

Guard and bad-set is assumed to be an interval

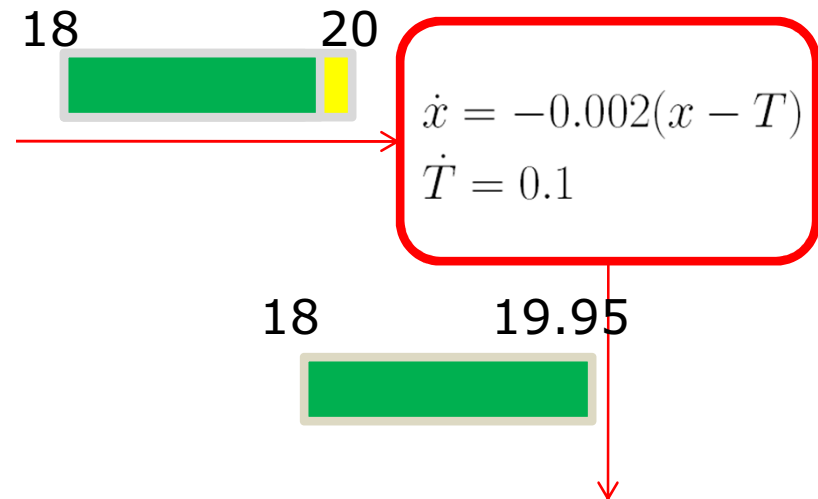
For intervals, we learn the true guards from the simulation oracle doing a binary search.

Simulate with initial condition

X=20 : Unsafe

X=19.75 : Safe

X=19.88 : Safe



Toy Example: Thermostat

Guard and bad-set is assumed to be an interval

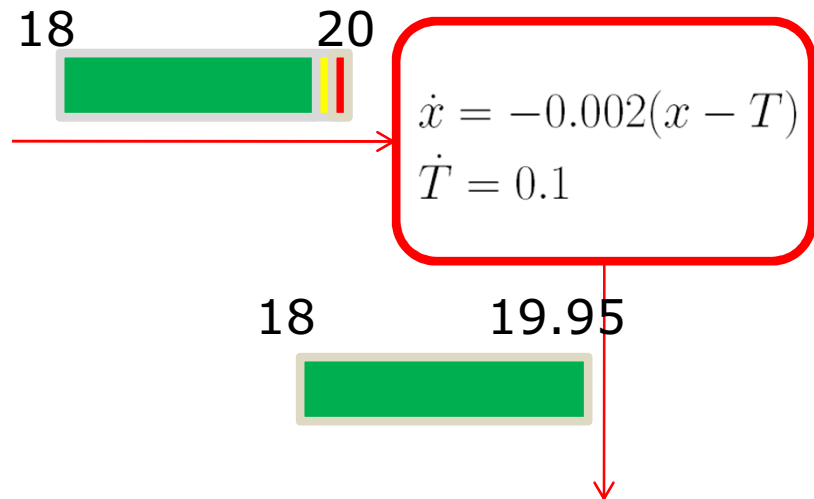
For intervals, we learn the true guards from the simulation oracle doing a binary search.

Simulate with initial condition

X=20 : Unsafe

X=19.88 : Safe

X=19.94 : UnSafe



Toy Example: Thermostat

Guard and bad-set is assumed to be an interval

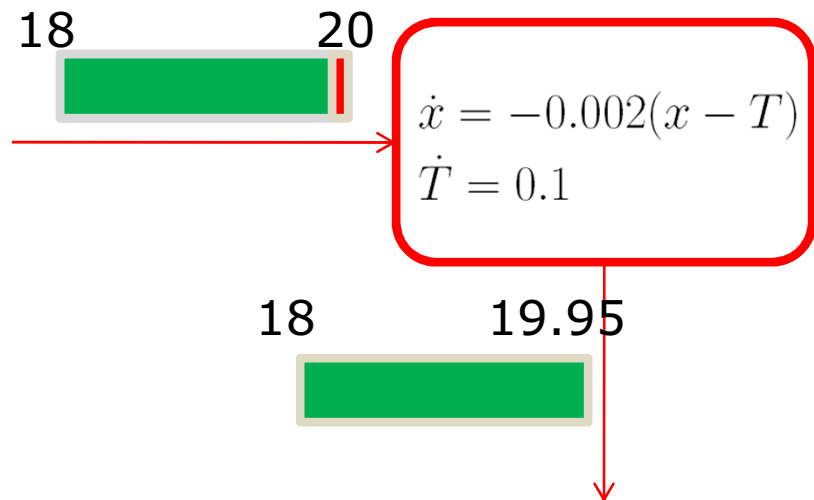
For intervals, we learn the true guards from the simulation oracle doing a binary search.

Simulate with initial condition

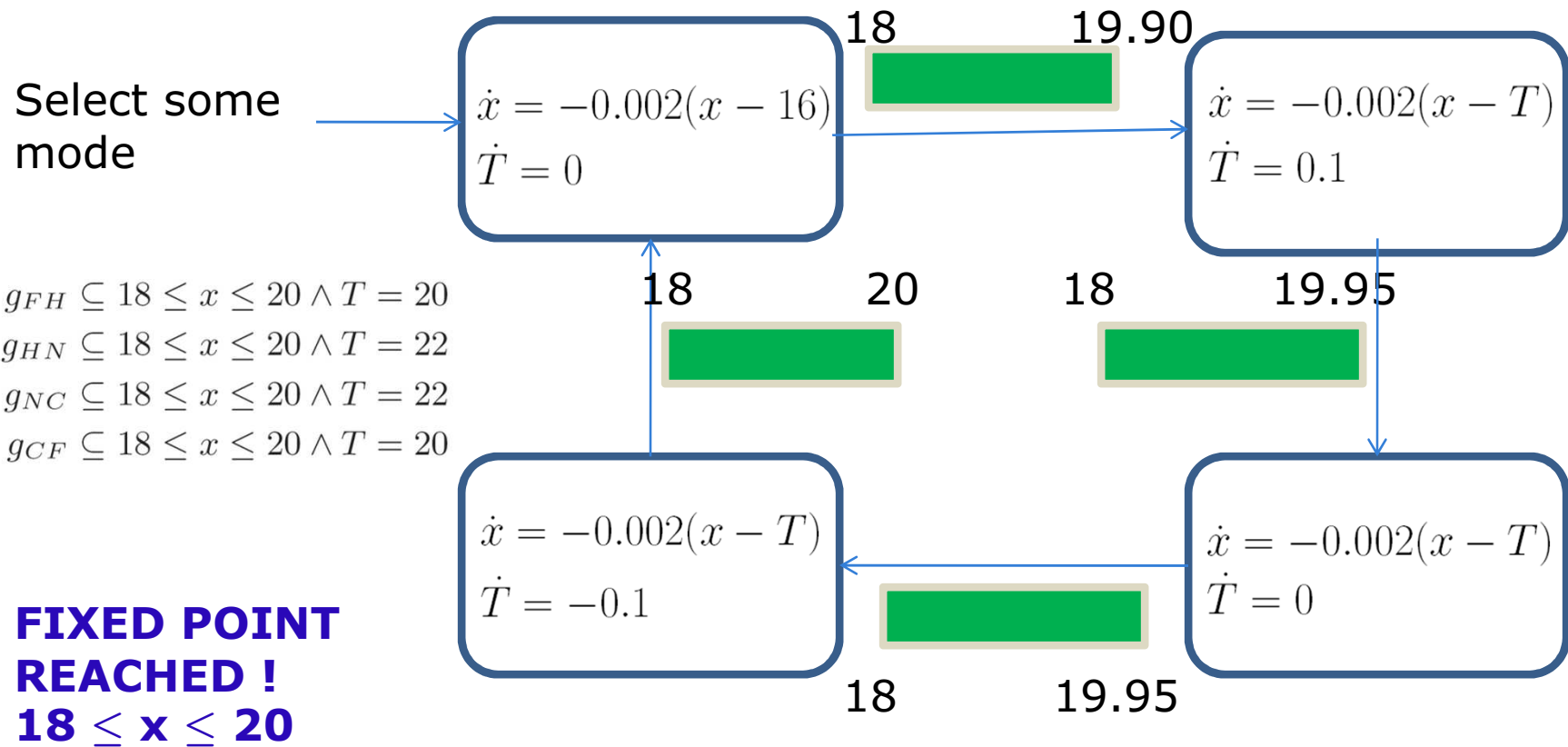
X=19.91 : Unsafe

X=19.89 : Safe

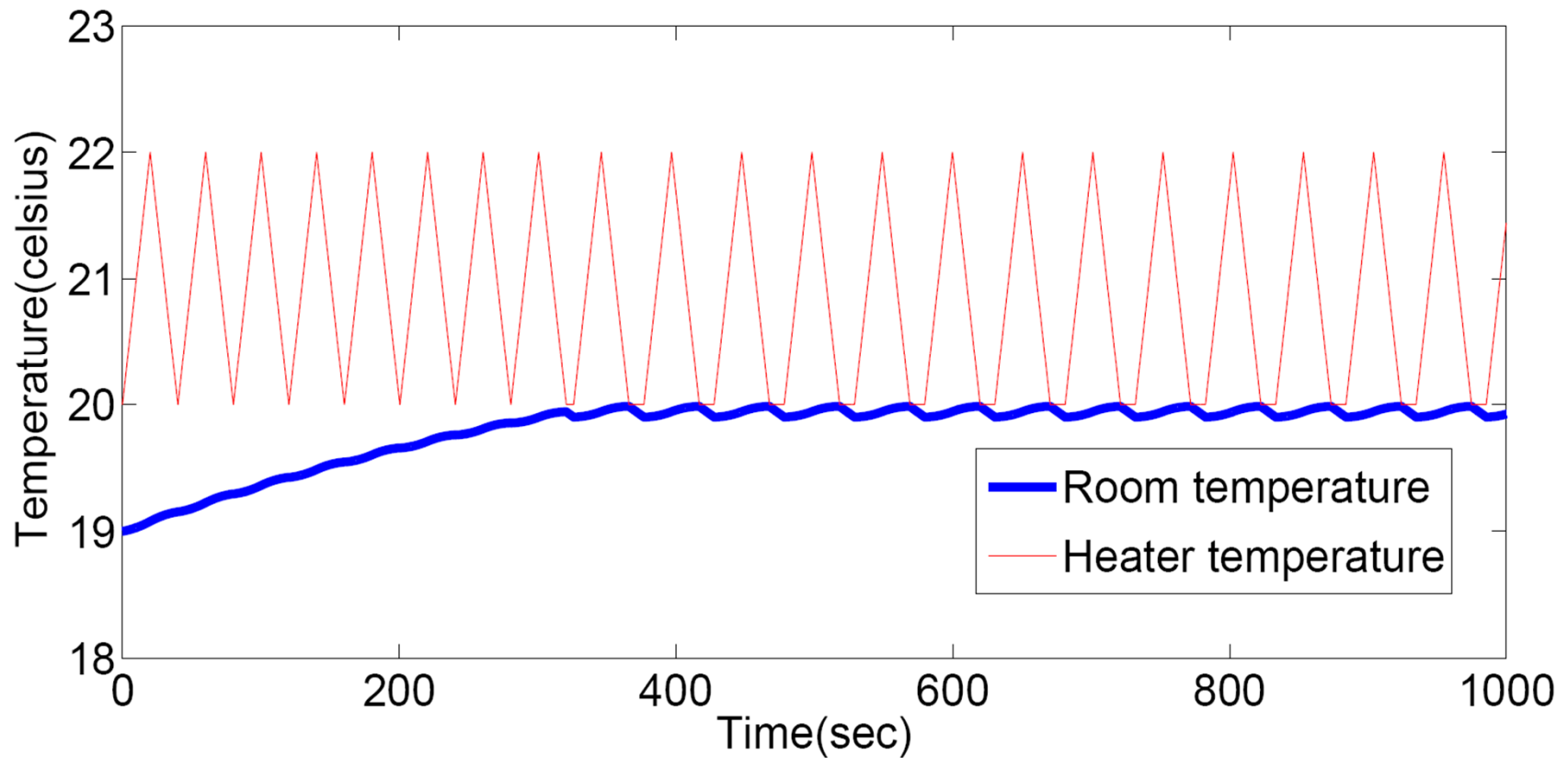
X=19.90 : Safe



Toy Example: Thermostat



Synthesized Thermostat for Safety Requirement

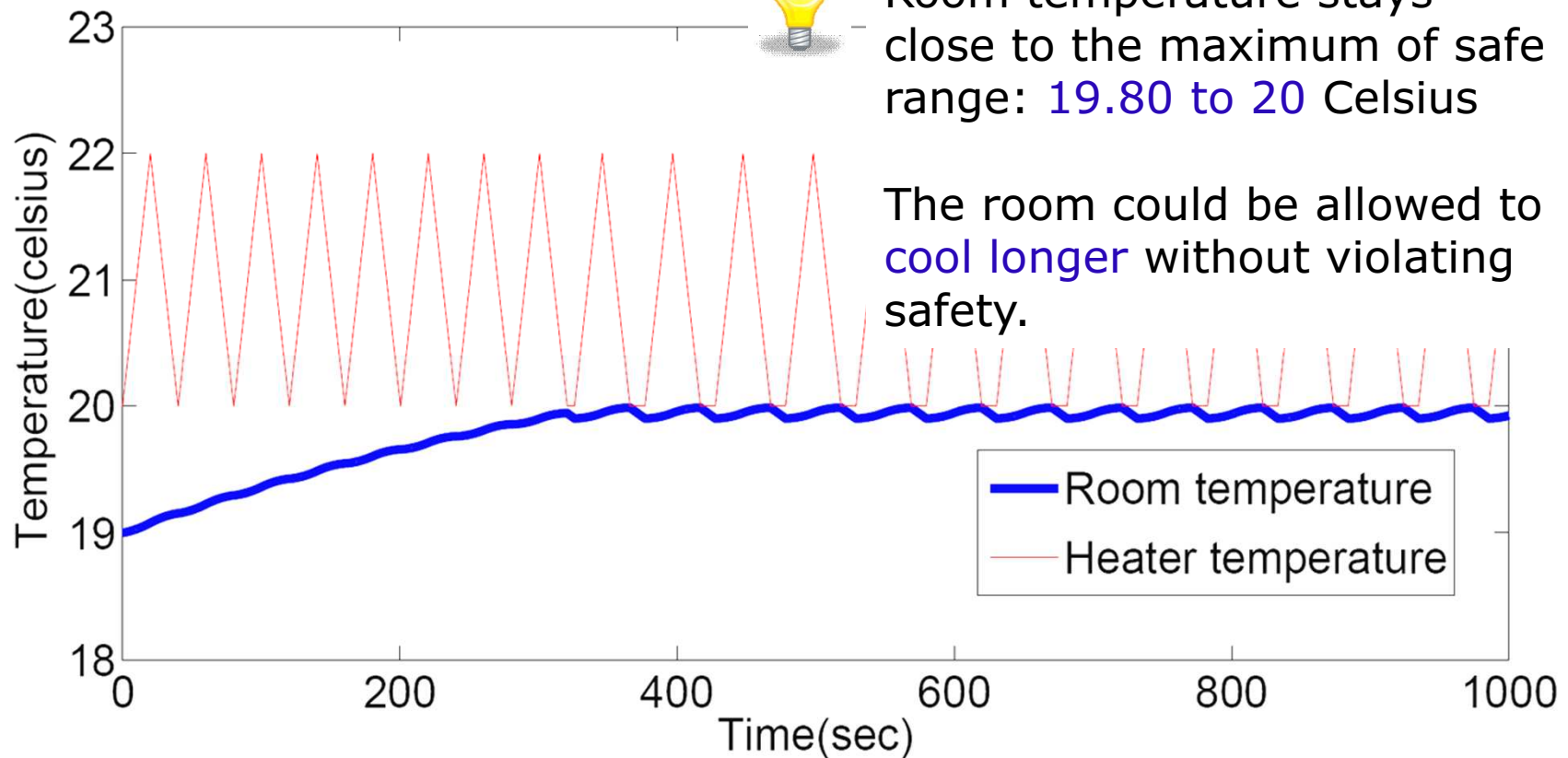


Synthesized Thermostat for Safety Requirement



Room temperature stays close to the maximum of safe range: 19.80 to 20 Celsius

The room could be allowed to cool longer without violating safety.

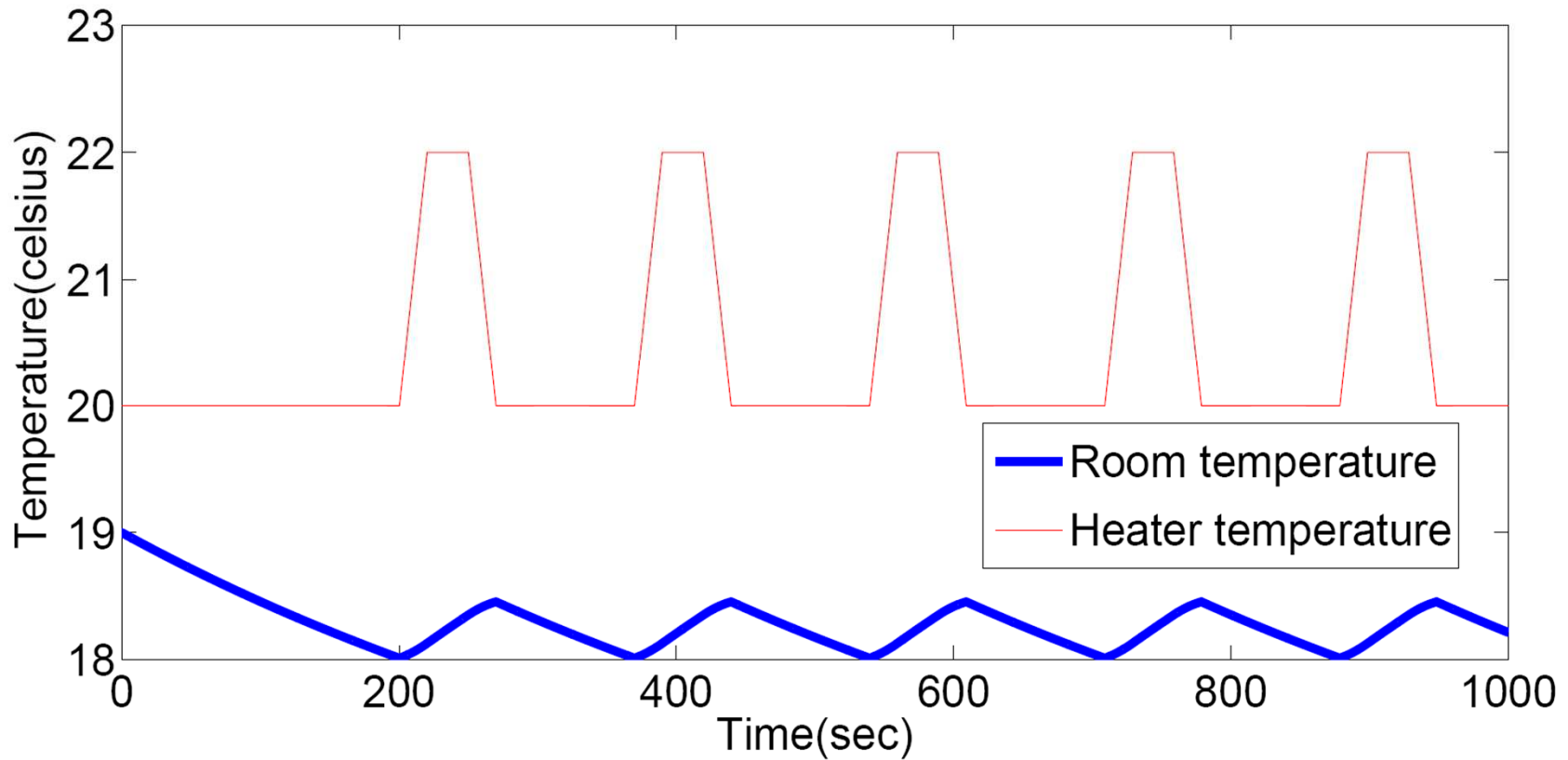


Dwell time

- Dwell time requirement specify lower and/or upper bound on time that can be spent in a mode.
- A way to model some performance properties: minimize switching, reduce stay in “costly” modes etc.
- Key idea:
 - Restrict entry guards to a mode to switch into the mode later (decreasing dwell-time)
 - Restrict exit guards to a mode to switch away from the mode later (increasing dwell-time)
 - Recompute fixed point to enforce safety properties

Synthesized Thermostat

100 s min dwell time in off mode



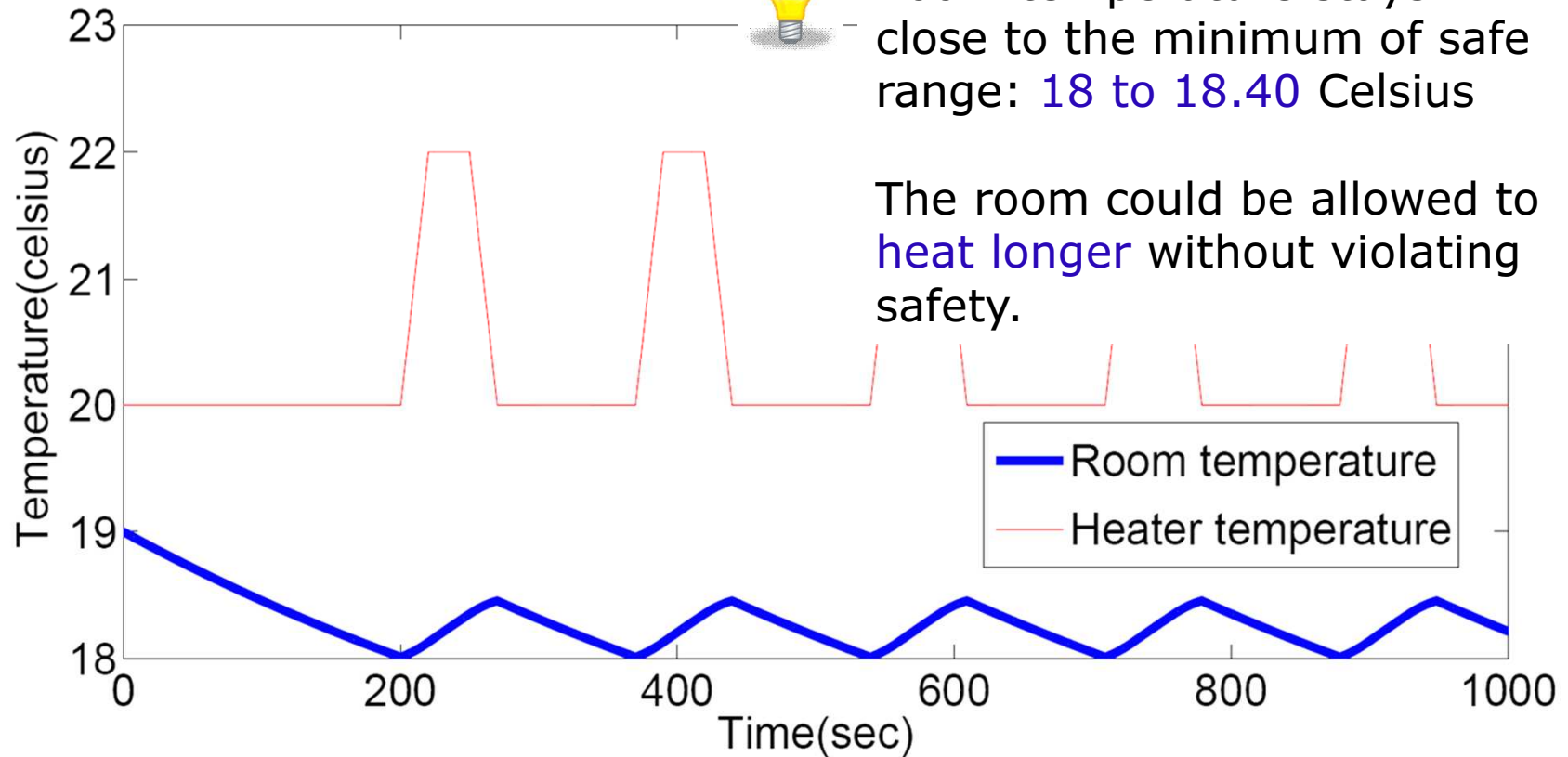
Synthesized Thermostat

100 s min dwell time in off mode



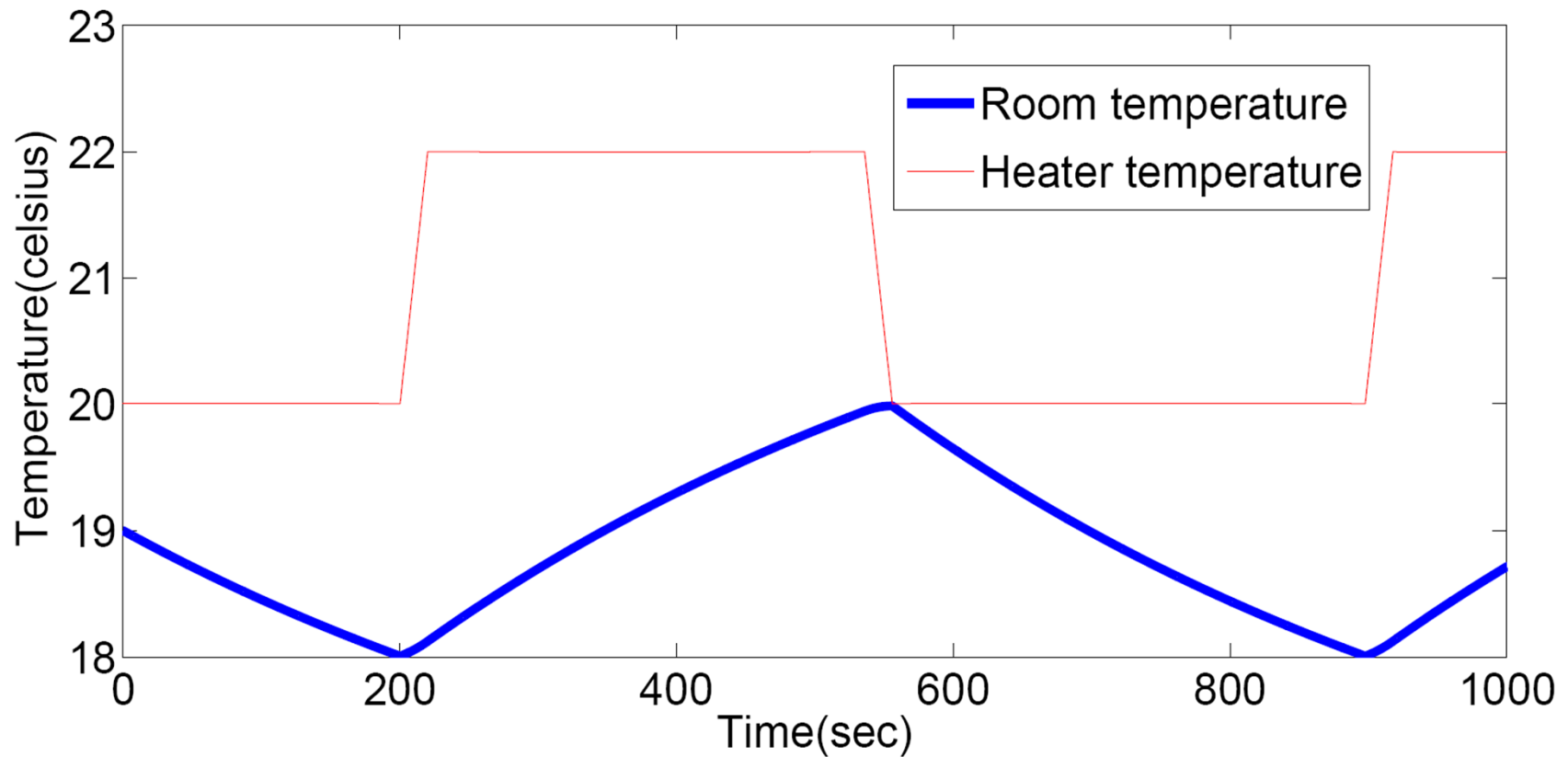
Room temperature stays close to the minimum of safe range: 18 to 18.40 Celsius

The room could be allowed to heat longer without violating safety.



Synthesized Thermostat

300 s min dwell time in off and on mode



Theorems

- **Soundness:**
Synthesized Design \models Given safety (and dwell-time) objectives.
- **Completeness:**
No Design Synthesized **if and only if** No Design Feasible with given safety (and dwell-time) objectives, and for given choice of class of predicates.

	Soundness	Completeness
Safety		
Safety + Dwell time		

Case Studies

<i>Example</i>	# of Iterations	Runtime (seconds)
Thermostat Controller		
v1	5	21.6
v2 Case A	6	26.2
v2 Case B	6	25.7
TCAS		
Case A	4	55.3
Case B	5	59.1
Automatic Transmission	6	83.6
Train Gate Controller		
Case A	3	22.5
Case B	4	28.3

Future Work

Generalizing dwell time to more general performance objectives for cyber-physical systems. Associate cost with each state,

- minimize cost averaged over time.
- minimize the maximum cost along any trajectory.
- Example: thermostat that consumes least energy

Thanks !

Conclusion

SILVER is a novel combination of algorithmic learning and numerical simulation based verification techniques was used to automatically synthesize switching logic for cyber-physical systems.

Goal

Provide an useful aid to designers and programmers.