

Systematically exploring control programs (Lecture I)

Ratul Mahajan
Microsoft Research

Joint work with Jason Croft,
Matt Caesar, and Madan Musuvathi

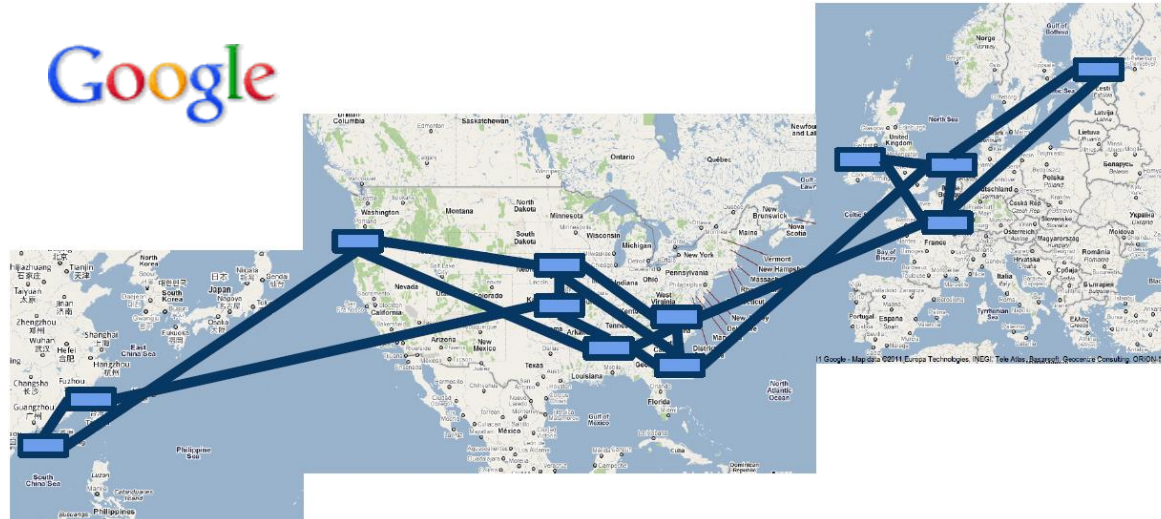
Control programs are everywhere

From the smallest of networks to the largest



Control programs are everywhere

From the smallest of networks to the largest



The nature of control programs

Collection of rules with triggers and actions

motionPorch.Detected:

```
if (Now - tLastMotion < 1s
    && lightLevel < 20)
    porchLight.Set(On)
tLastMotion = Now
```

@6:00:00 PM:

```
porchLight.Set(On)
```

@6:00:00 AM:

```
porchLight.Set(Off)
```

packetIn:

```
entry = new Entry(inPkt.src,
                  inPkt.dst)
if (!cache.Contains(entry)
    cache.Insert(entry, Now)
```

CleanupTimer:

```
foreach entry in cache
    if (Now - cache[entry] < 5s)
        cache.Remove(entry)
```

Buggy control programs wreak havoc



One nice morning in
the summer



Buggy control programs wreak havoc

“I had a rule that would turn on the heat, disarm the alarm, turn on some lights, etc. at 8am

I came home from vacation to find a warm, inviting, insecure, well lit house that had been that way for a week.....

That’s just one example, but the point is that it has taken me literally YEARS of these types of mistakes to iron out all the kinks.”

Control programs are hard to reason about

motionPorch.Detected:

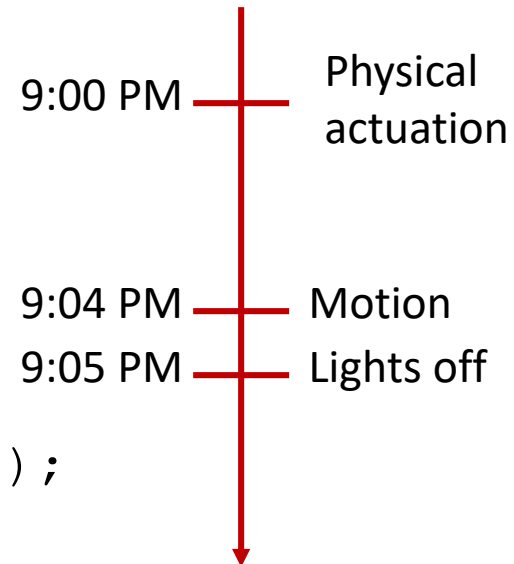
```
if (Now - timeLastMotion < 1 secs
    && lightMeter.Level < 20)
    porchLight.Set(On);
timeLastMotion = Now;
```

porchLight.StateChange:

```
if (porchLight.State == On)
    timerPorchLight.Reset(5 mins);
```

timerPorchLight.Fired:

```
if (Now.Hour > 6AM && Now.Hour < 6PM)
    porchLight.Set(Off);
```



Large input
space

Dependence on
time

Rule
interaction

Desirable properties for bug finders



Sound



Complete



Fast

Two bug finding methods



Testing



Model checking

Two threads in model checking



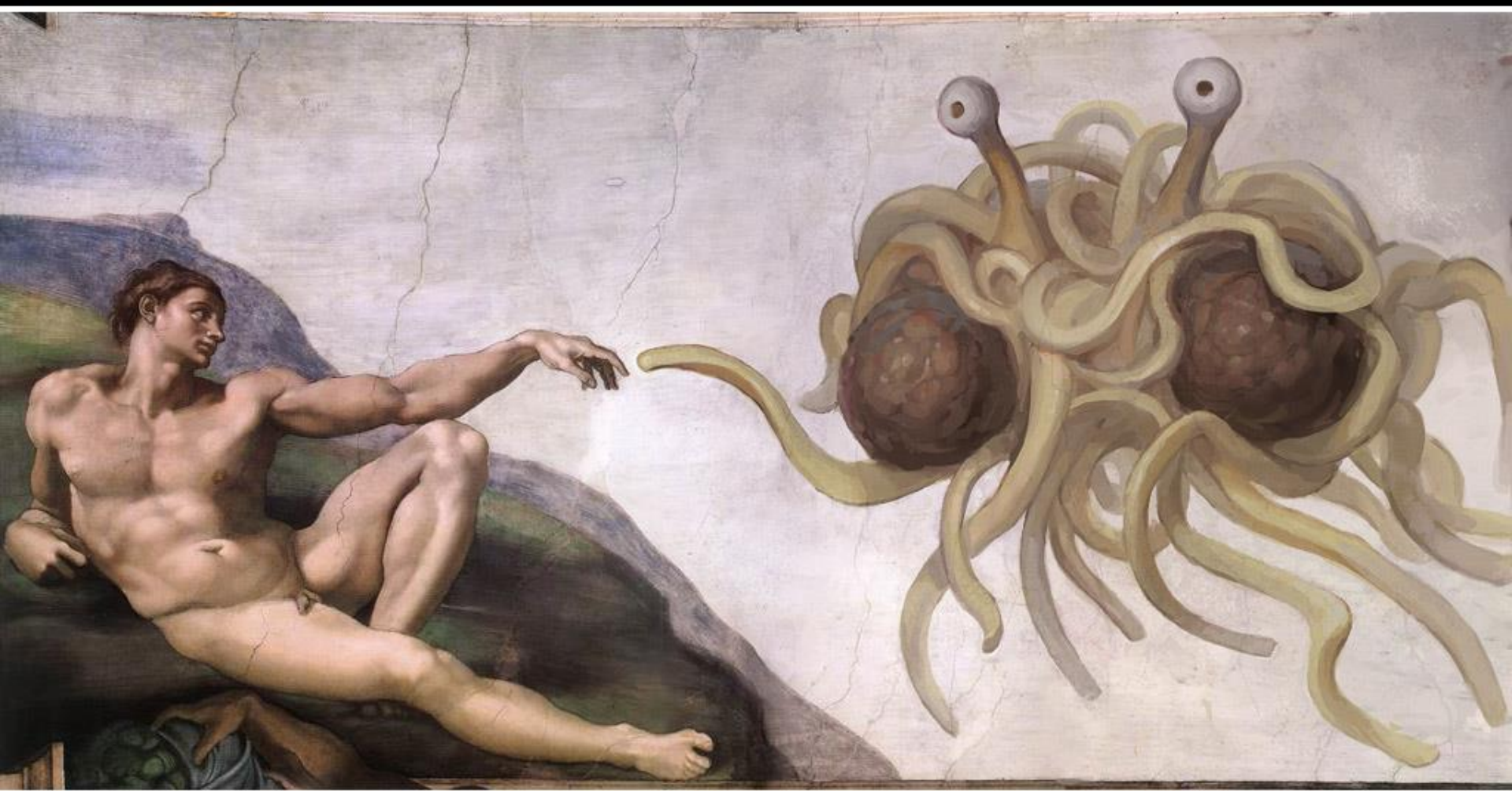
Check models



Check code

Model checking code

FSM is the most popular abstraction

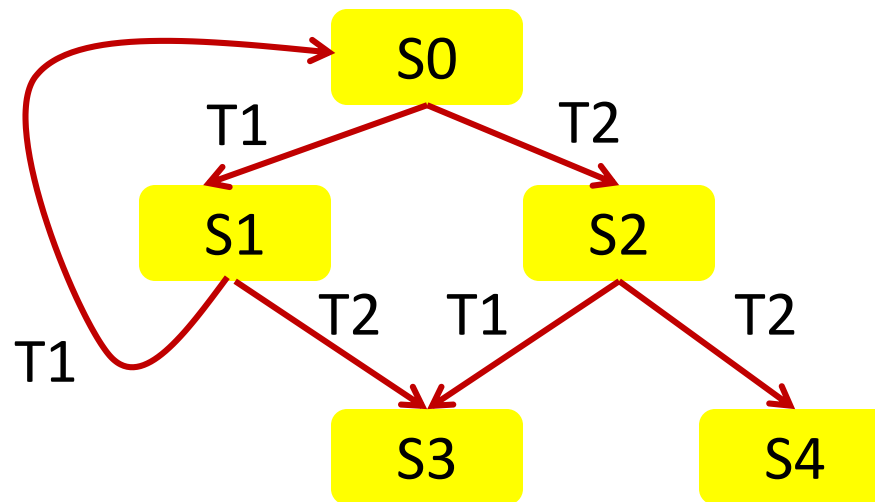


TOUCHED BY HIS NOODLY APPENDAGE

Model checking code

FSM is the most popular abstraction

- Decide what are “states” and “transitions”



Example

motionPorch:

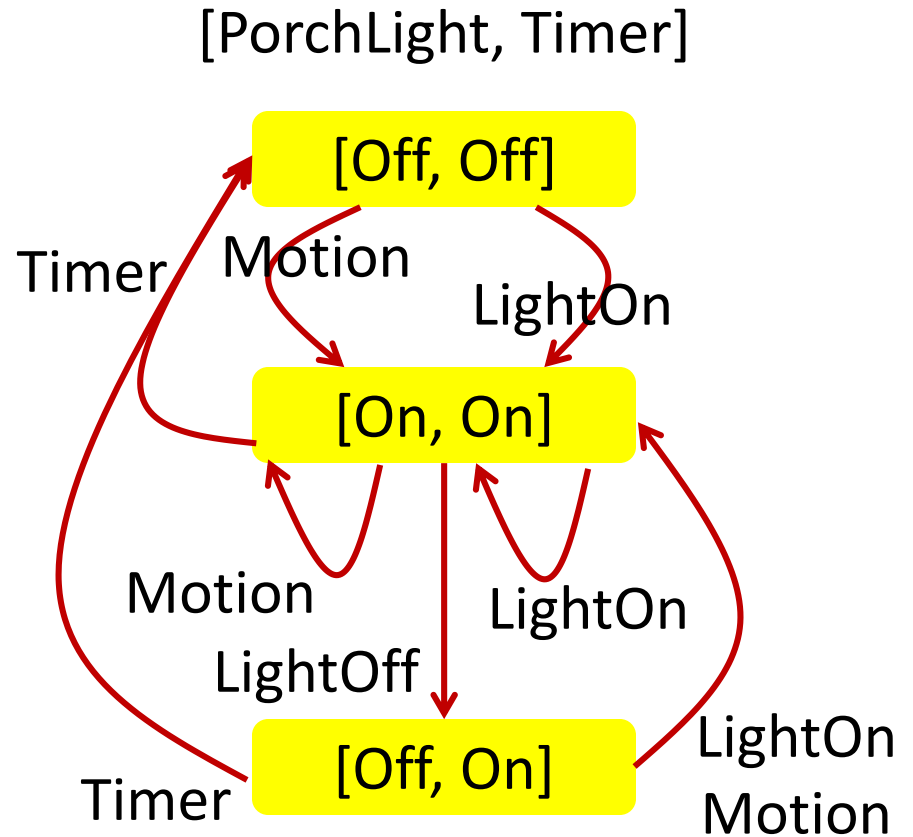
```
porchLight.Set(On)  
timer.Start(5 mins)
```

porchLight.On:

```
timer.Start(5 mins)
```

timer.Fired:

```
porchLight.Set(Off)
```



Exploring input space

motionPorch:

```
if (lightLevel < 20)
  porchLight.Set(On)
  timer.Start(10 mins)
```

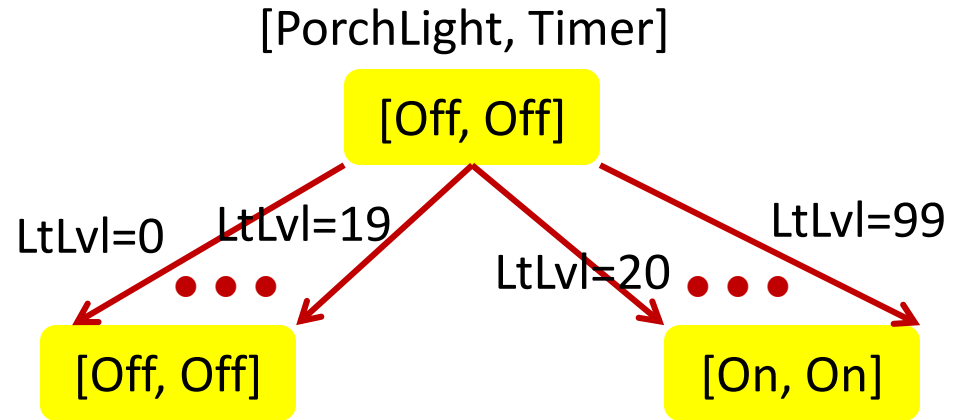
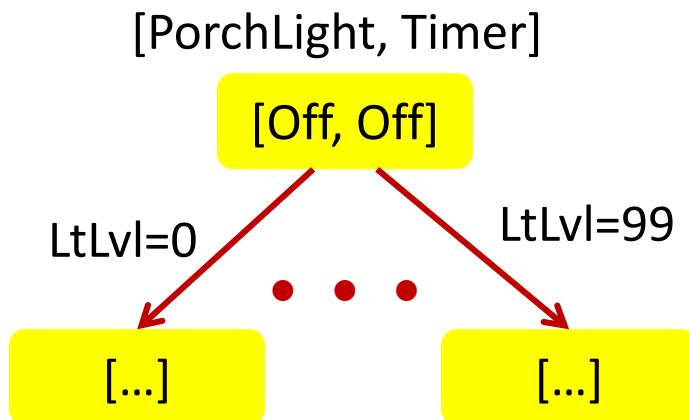
porchLight.On:

```
timer.Start(5 mins)
```

timer.Fired:

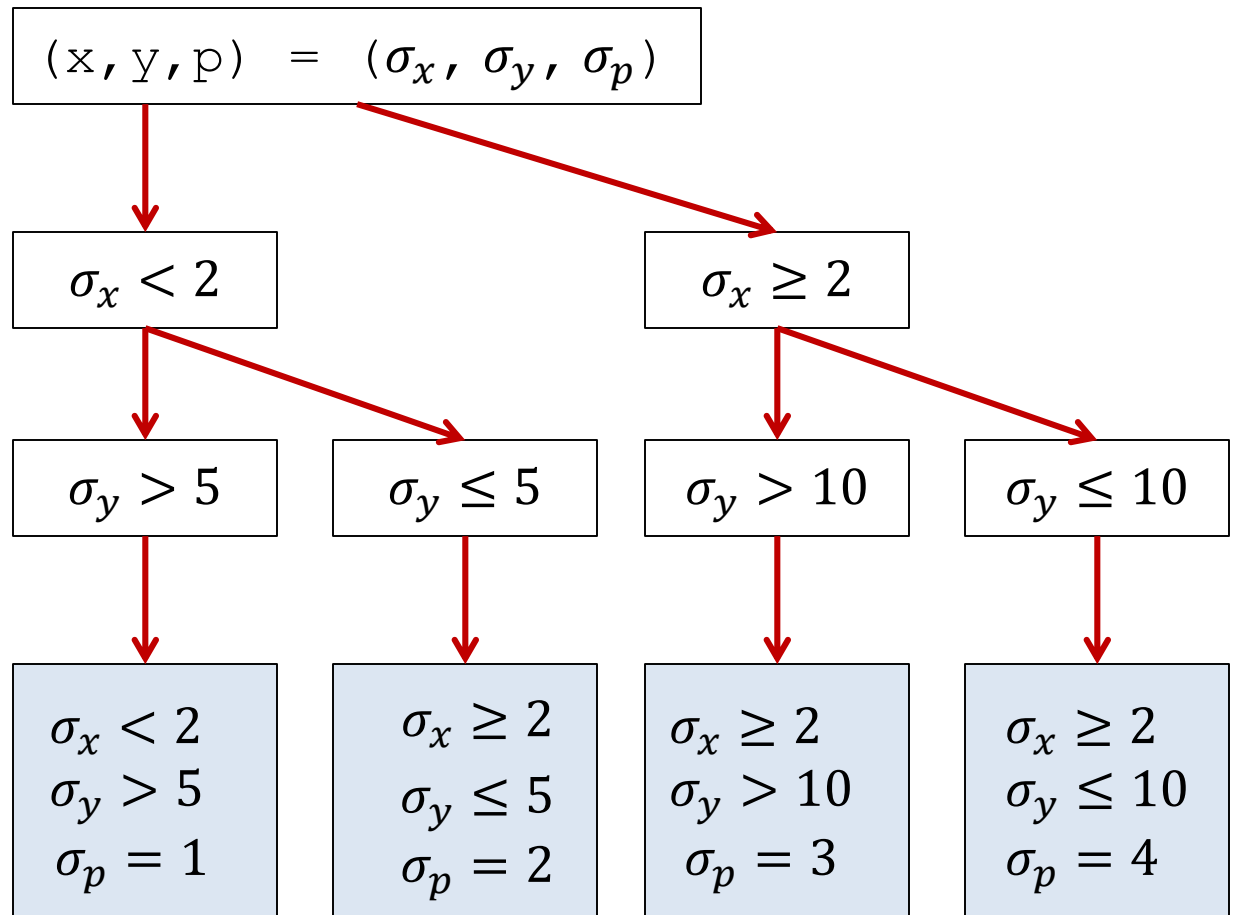
```
porchLight.Set(Off)
```

To explore comprehensively,
must consider all possible
values of input parameters



Symbolic execution

```
if (x < 2)
  if (y > 5)
    p = 1;
  else
    p = 2;
else
  if (y > 10)
    p = 3;
  else
    p = 4;
```



Finding equivalent inputs using symbolic execution

1. Symbolically execute each trigger
2. Find input ranges that lead to same *state*

motionPorch:

```
if (lightMeter.level < 20)
  porchLight.Set(On)
  timer.Start(5 mins)
```

$LtLvl < 20$

$LtLvl \geq 20$

porchLight.On:

```
timer.Start(5 mins)
```

$LtLvl = *$

timer.Fired:

```
porchLight.Set(Off)
```

$LtLvl = *$

Finding equivalent inputs using symbolic execution

1. Symbolically execute each trigger
2. Find input ranges that lead to same *state*

motionPorch:

`x = lightMeter.Level`

`LtLvl= 0`

`• • • •`

`LtLvl= 99`

porchLight.On:

`timer.Start(5 mins)`

timer.Fired:

`porchLight.Set(Off)`

Efficiently exploring the input space

Pick random values in equivalent classes

motionPorch:

```
if (lightMeter.level < 20)
  porchLight.Set(On)
  timer.Start(5 mins)
```

LtLvl < 20

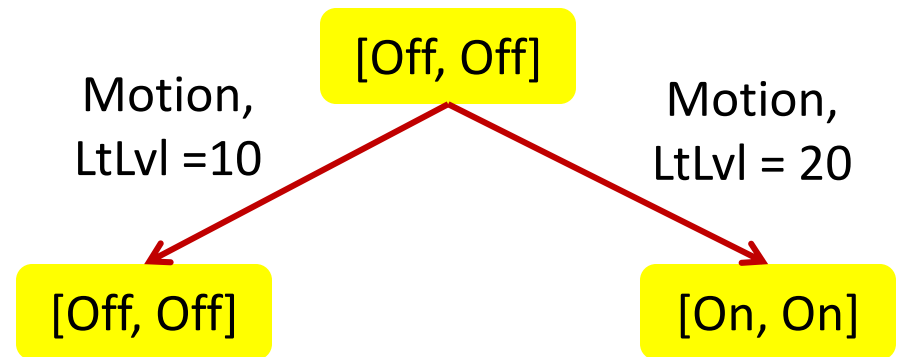
LtLvl ≥ 20

porchLight.On:

```
timer.Start(5 mins)
```

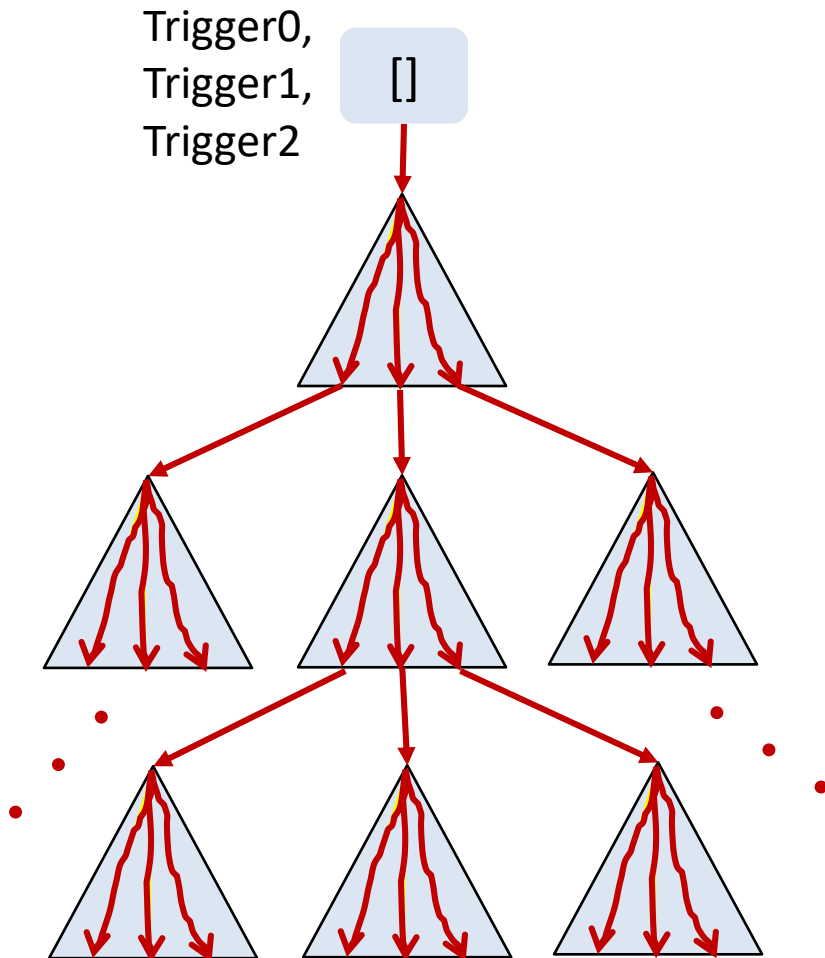
timer.Fired:

```
porchLight.Set(Off)
```

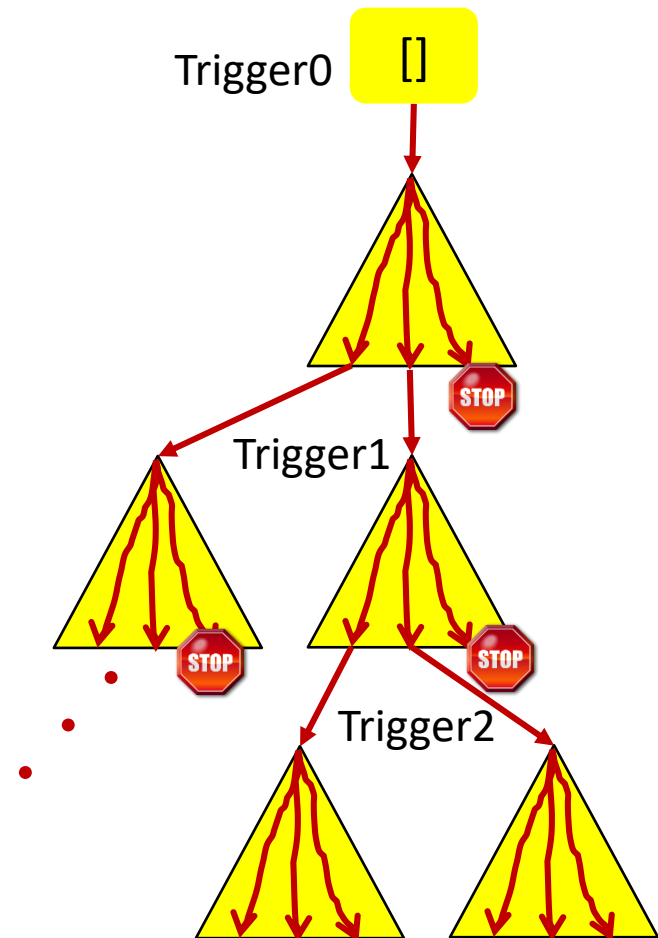


Use symbolic execution alone?

Symbolic, path-based



Concrete, state-based



Exploring temporal behavior: soundness

motionPorch:

```
porchLight.Set(On)  
timerDim.Start(5 mins)  
timerOff.Start(10 mins)
```

porchLight.On:

```
timerDim.Start(5 mins)  
timerOff.Start(10 mins)
```

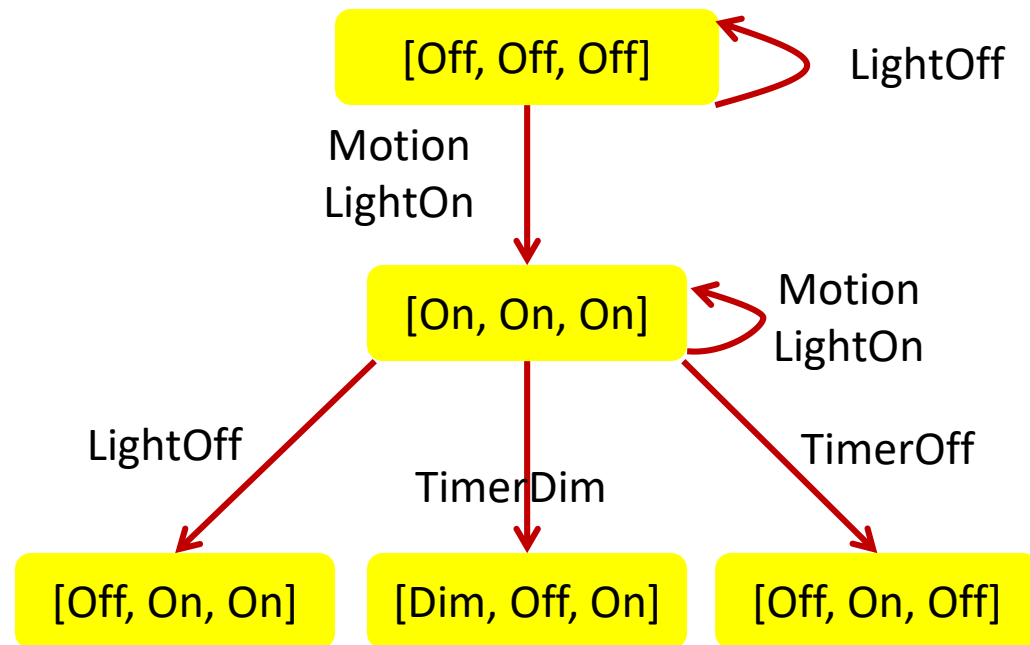
timerDim.Fired:

```
porchLight.Set(Dim)
```

timerOff.Fired:

```
porchLight.Set(Off)  
if timerDim.On()  
    Abort();
```

[PorchLight, TimerDim, TimerOff]



Exploring temporal behavior: completeness

motionPorch:

```
if (Now - tLastMotion < 60)
    porchLight.Set(On)
    timer.Start(600)
tLastMotion = Now
```

porchLight.On:

```
timer.Start(600)
```

timer.Fired:

```
porchLight.Set(Off)
```

To explore comprehensively,
must fire all possible events
at all possible times

Trigger0:

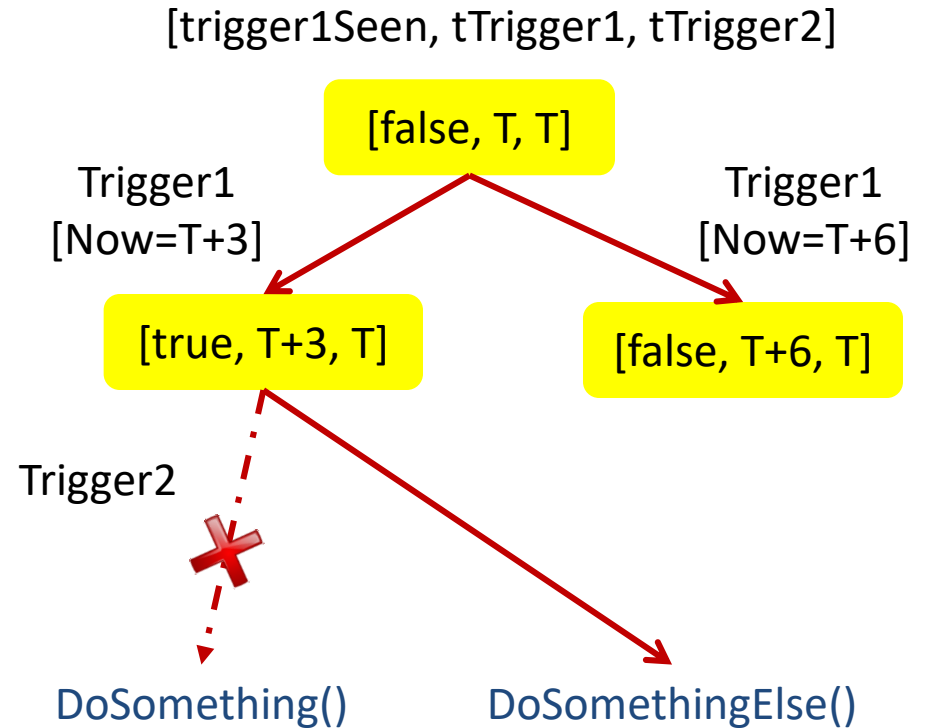
```
tTrigger1 = Now  
tTrigger2 = Now  
trigger1Seen = false
```

Trigger1:

```
if (Now - tTrigger1 < 5)  
    trigger1Seen = true  
tTrigger1 = Now
```

Trigger2:

```
if (trigger1Seen)  
    if (Now - tTrigger2 < 2)  
        DoSomething()  
    else  
        DoSomethingElse()
```



Trigger0:

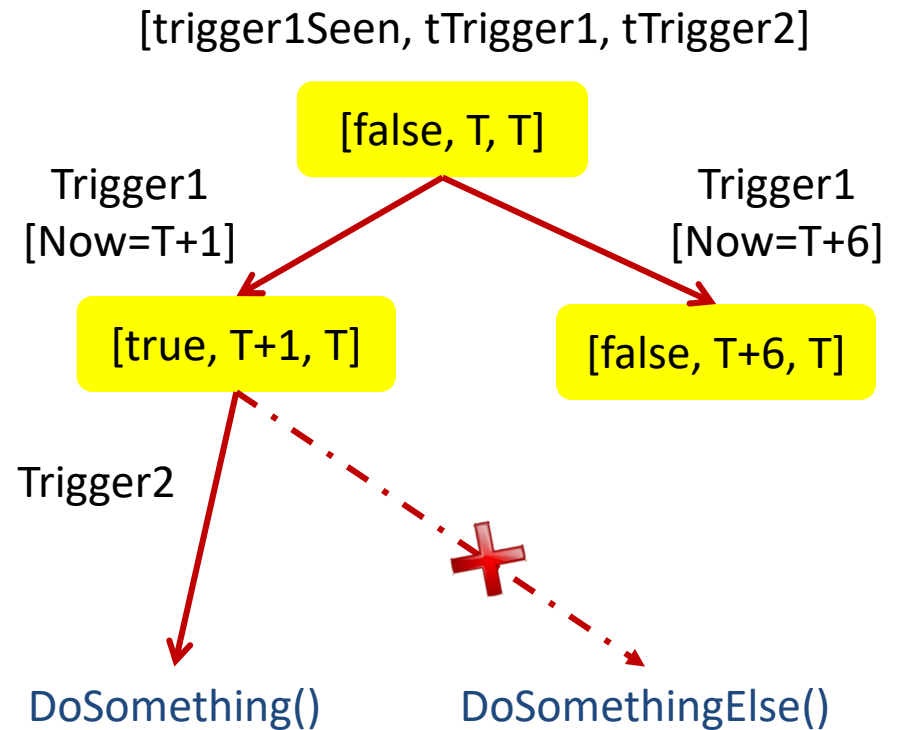
```
tTrigger1 = Now
tTrigger2 = Now
trigger1Seen = false
```

Trigger1:

```
if (Now - tTrigger1 < 5)
    trigger1Seen = true
    tTrigger1 = Now
```

Trigger2:

```
if (trigger1Seen)
    if (Now - tTrigger2 < 2)
        DoSomething()
    else
        DoSomethingElse()
```



The tyranny of “all possible times”



Timed automata

FSM (states, transitions) + the following:

- Finite number of real-values clocks (VCs)
- All VCs progress at the same rate, except that one or more VCs may reset on a transition
- VC constraints gate transitions

Trigger0:

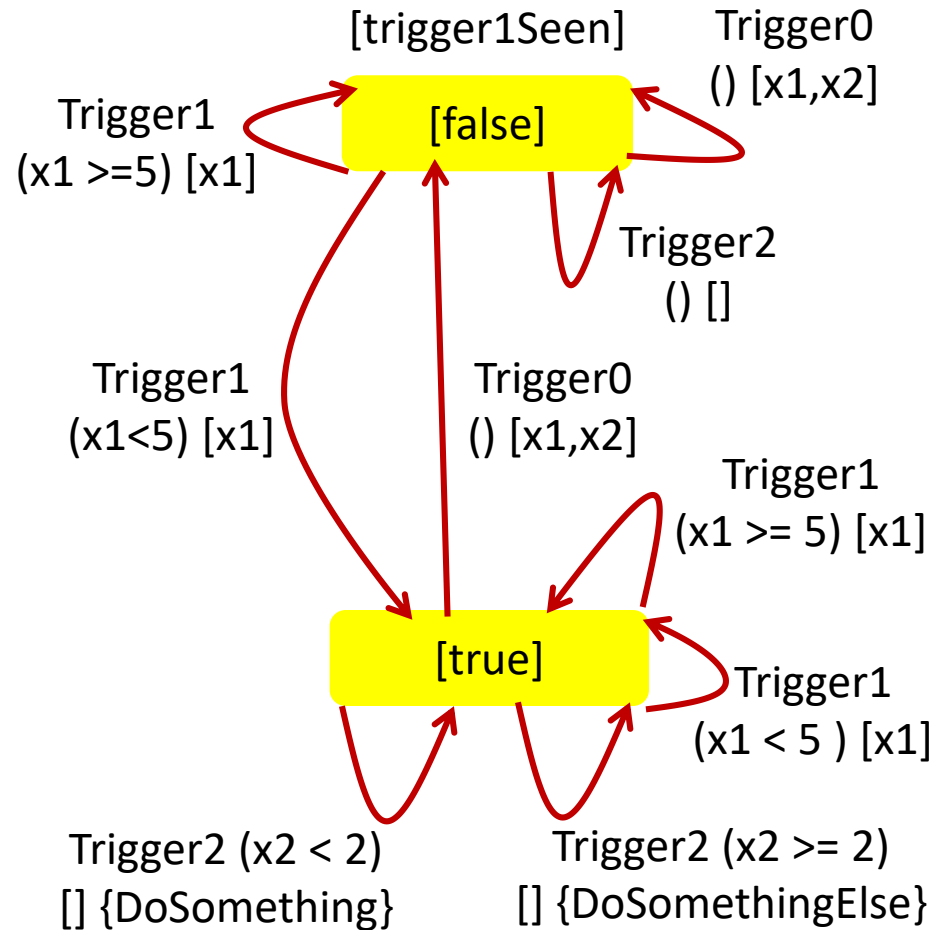
```
tTrigger1 = Now  
tTrigger2 = Now  
trigger1Seen = false
```

Trigger1:

```
if (Now - tTrigger1 < 5)  
    trigger1Seen = true  
    tTrigger1 = Now
```

Trigger2:

```
if (trigger1Seen)  
    if (Now - tTrigger2 < 2)  
        DoSomething()  
    else  
        DoSomethingElse()
```



Properties of timed automata

If VC constraints are such that:

$$x < 2 \quad \checkmark$$

$$x < y + 2 \quad \checkmark$$

No arithmetic operation involving two VCs

$$\cancel{x + y < z}$$

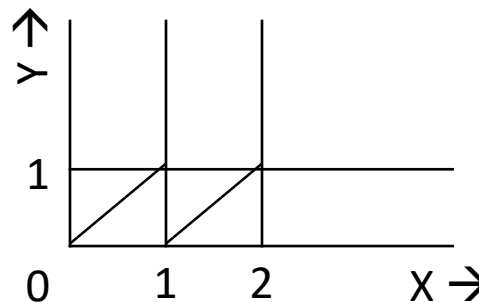
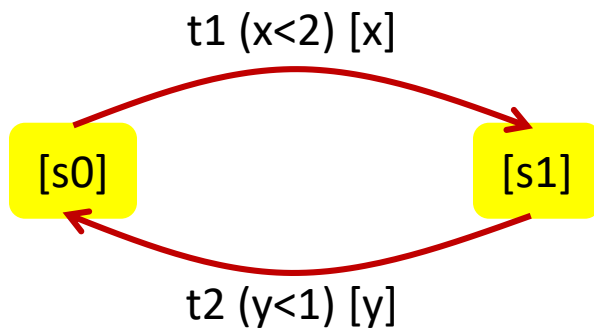
No multiplication operation involving a VC

$$\cancel{2x < 3}$$

No irrational constants in constraints

$$\cancel{x < \sqrt{2}}$$

Time can be partitioned into equivalence regions



28 regions

- Corner points (6)
- Line segments (14)
- Spaces (8)

Trigger0:

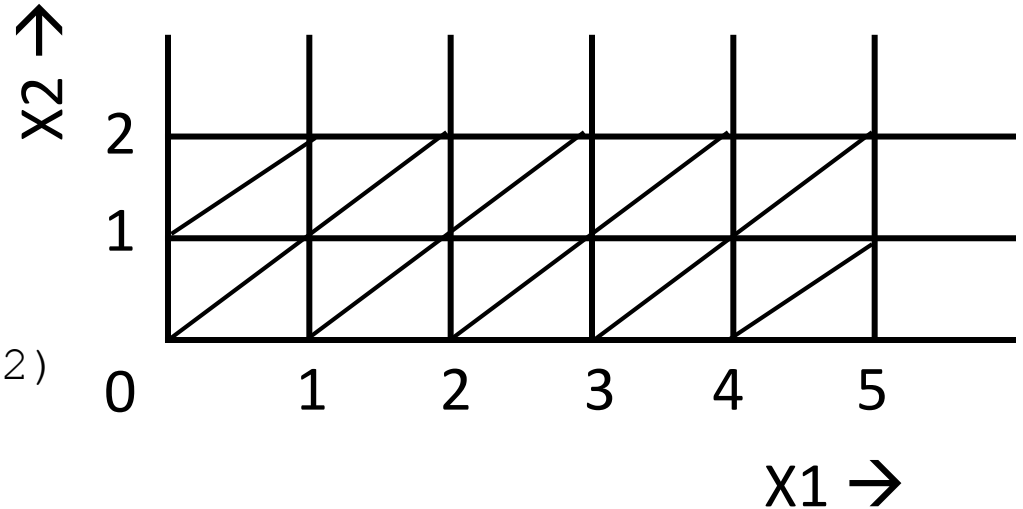
```
tTrigger1 = Now  
tTrigger2 = Now  
trigger1Seen = false
```

Trigger1:

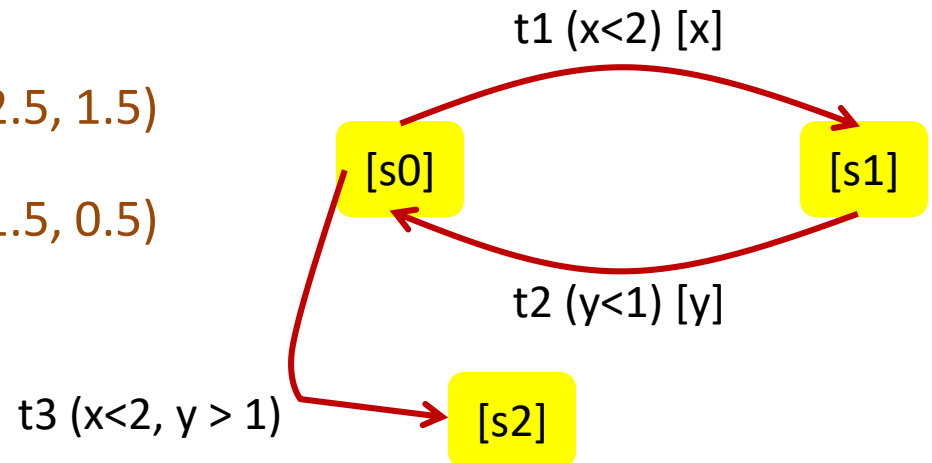
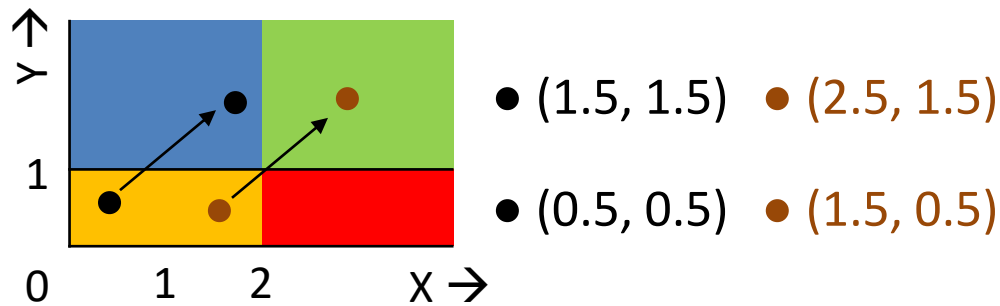
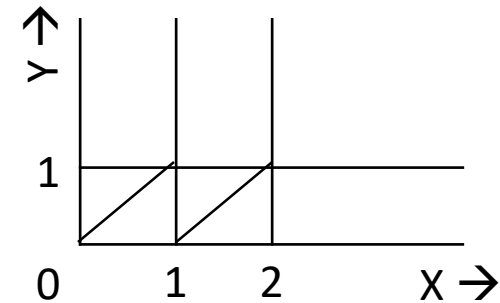
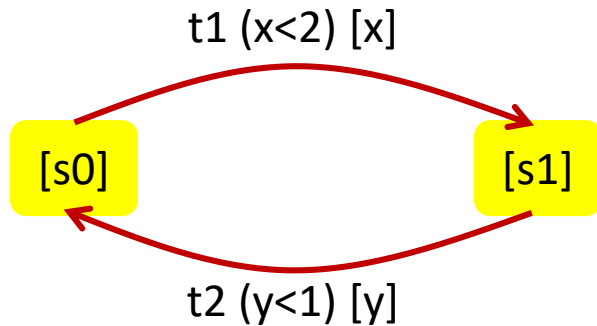
```
if (Now - tTrigger1 < 5)  
    trigger1Seen = true  
tTrigger1 = Now
```

Trigger2:

```
if (trigger1Seen)  
    if (Now - tTrigger2 < 2)  
        DoSomething()  
else  
    DoSomethingElse()
```



Why regions are fine-grained



Region construction

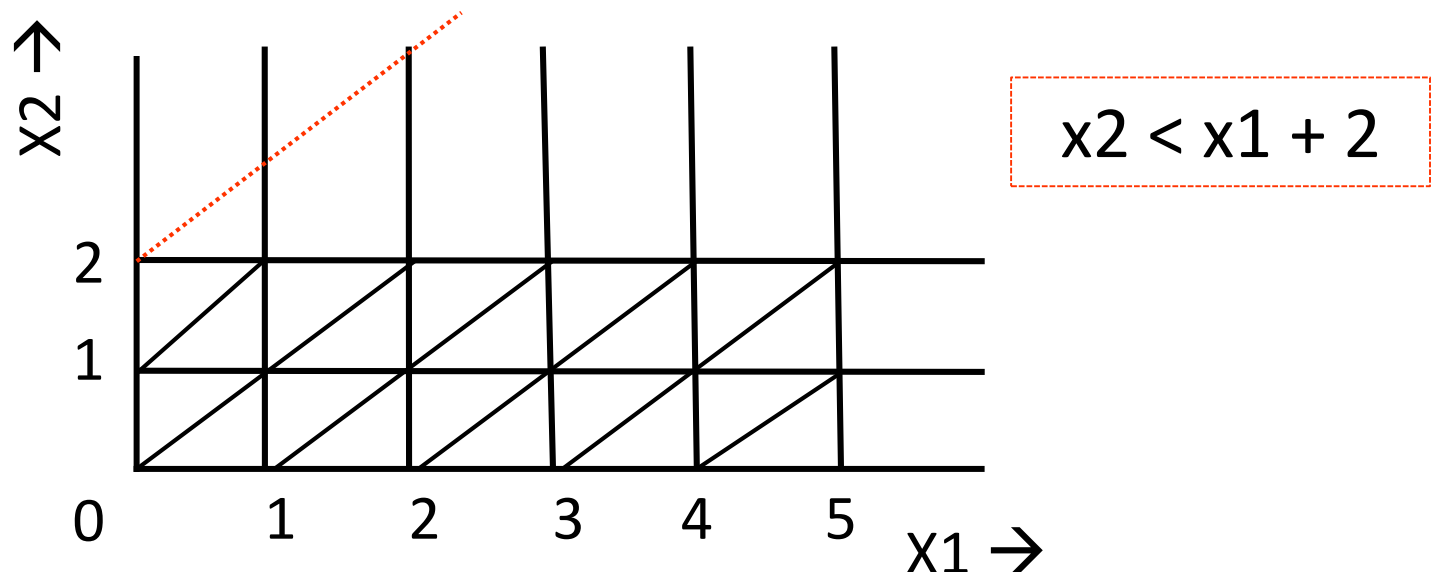
If integer constants and simple constraints (e.g., $x < c$)

Straight lines

$$\forall x: \{x = c \mid c = 0, 1, \dots, c_x\}$$

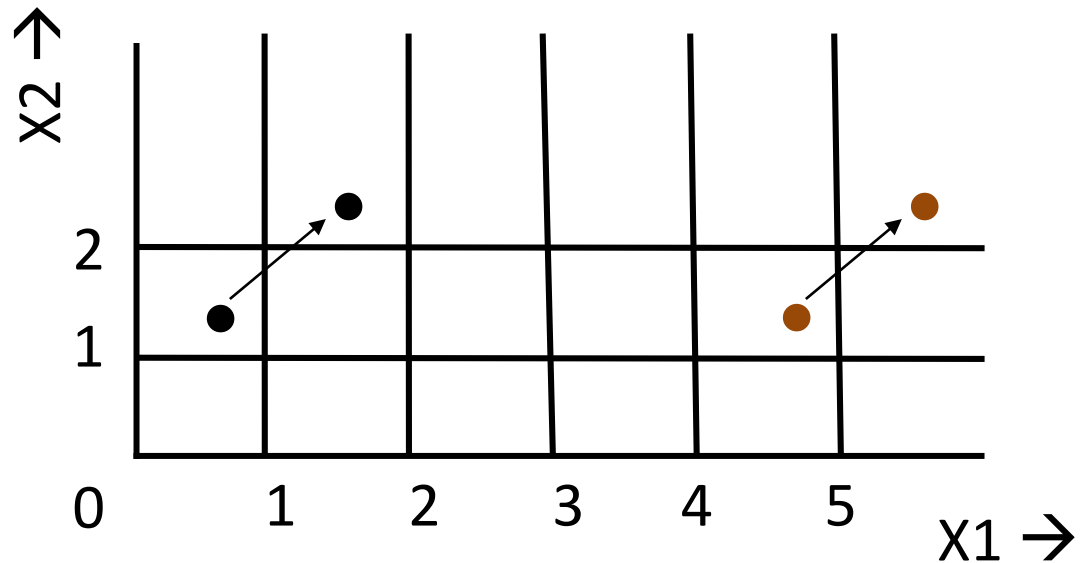
Diagonals lines

$$\forall x, y: \{\text{fract}(x) = \text{fract}(y) \mid x < c_x, y < c_y\}$$



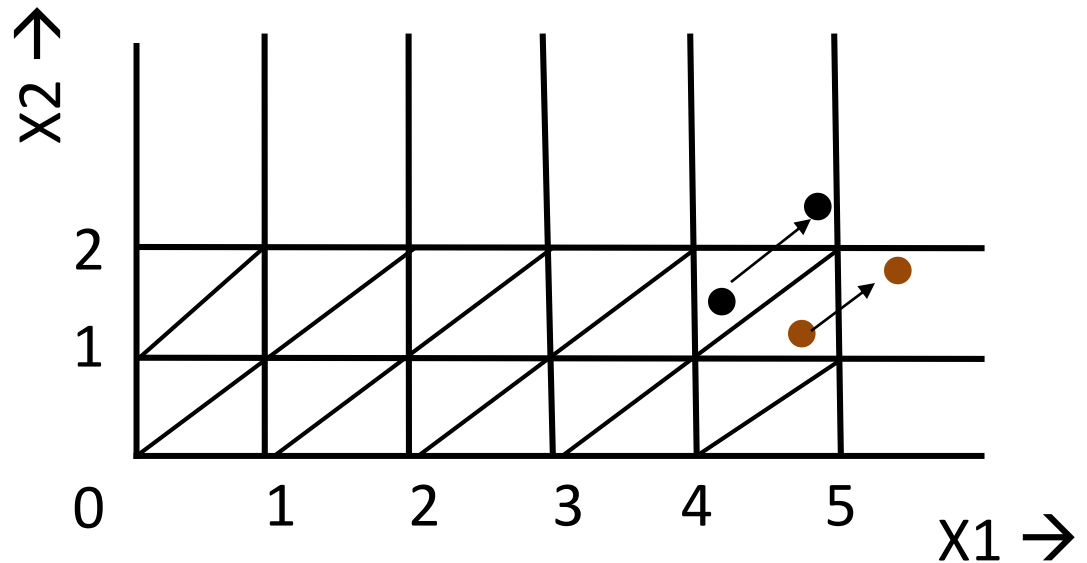
Why this construction works

1. $X_1 < 5$
2. $X_2 < 2$
3. $X_1 < 5 \ \&\& \ X_2 > 2$

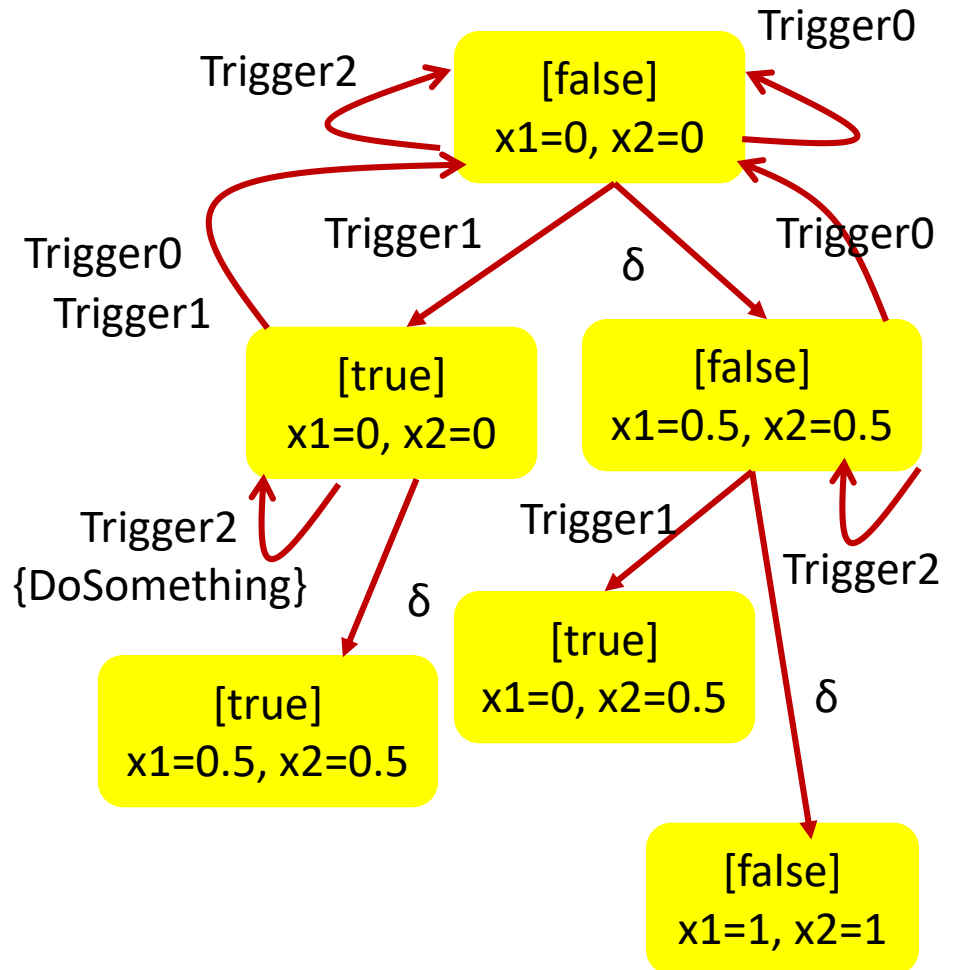
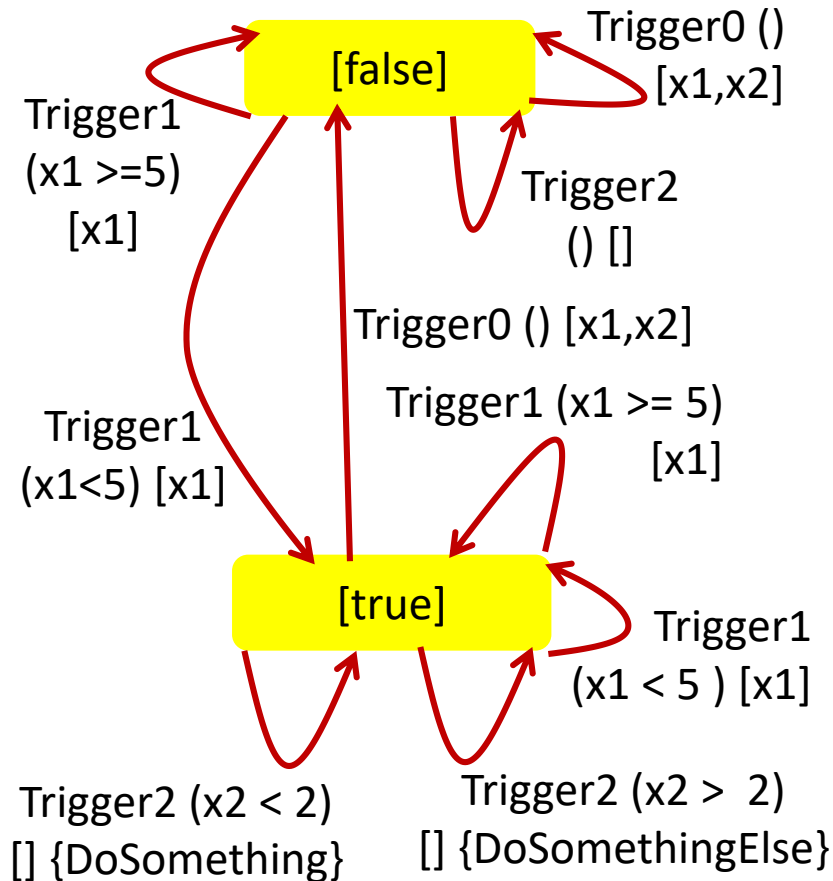


Why this construction works

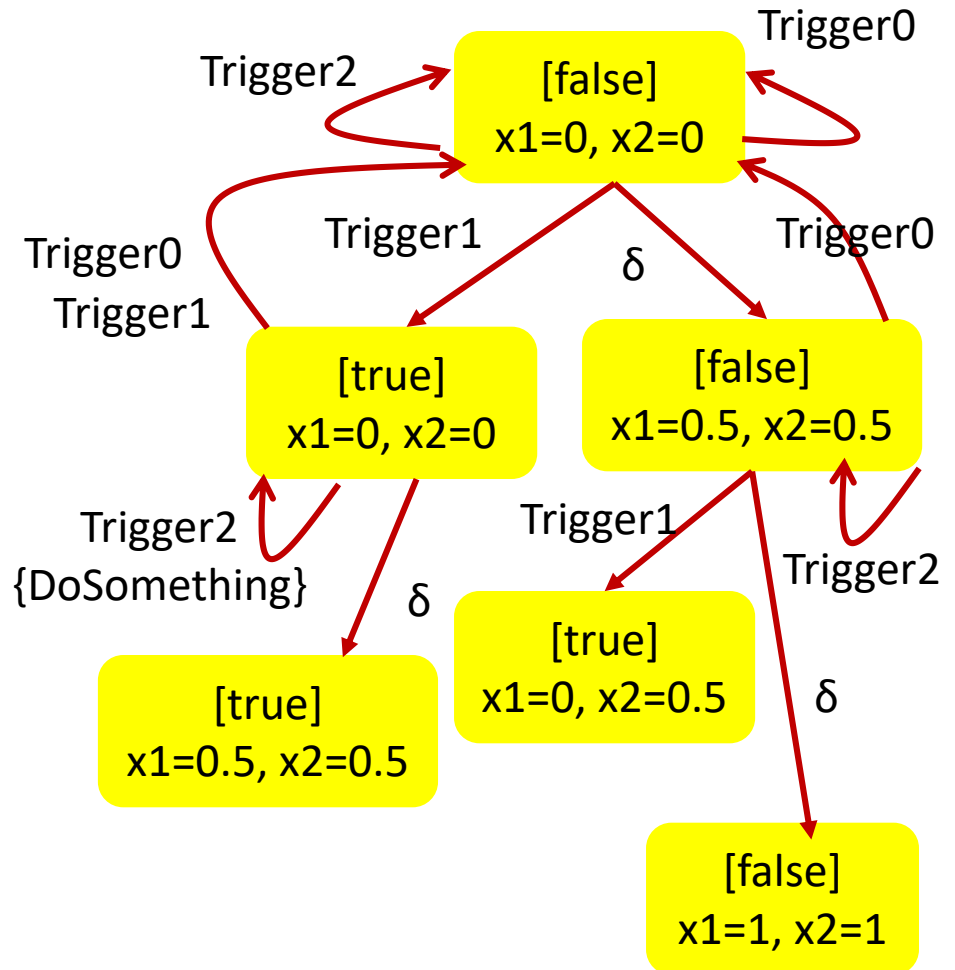
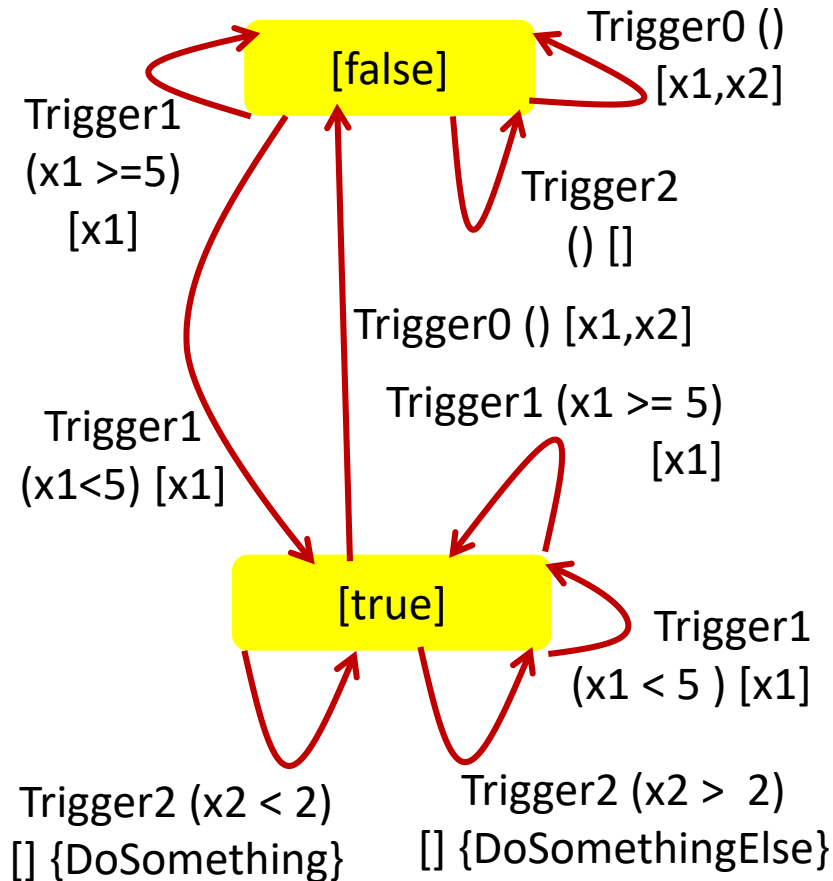
1. $X_1 < 5$
2. $X_2 < 2$
3. $X_1 < 5 \ \&\& \ X_2 > 2$



Exploring a TA



Exploring a TA



Systematically exploring control programs (Lecture II)

Ratul Mahajan
Microsoft Research

Joint work with Jason Croft,
Matt Caesar, and Madan Musuvathi

Recap: The nature of control programs

Collection of rules with triggers and actions

motionPorch.Detected:

```
if (Now - tLastMotion < 1s
    && lightLevel < 20)
    porchLight.Set(On)
tLastMotion = Now
```

@6:00:00 PM:

```
porchLight.Set(On)
```

@6:00:00 AM:

```
porchLight.Set(Off)
```

packetIn:

```
entry = new Entry(inPkt.src,
                  inPkt.dst)
if (!cache.Contains(entry))
    cache.Insert(entry, Now)
```

CleanupTimer:

```
foreach entry in cache
    if (Now - cache[entry] < 5s)
        cache.Remove(entry)
```

Recap: Timed automata

FSM (states, transitions) + the following:

- Finite number of real-values clocks (VCs)
- All VCs progress at the same rate, except that one or more VCs may reset on a transition
- VC constraints gate transitions

Recap: Properties of timed automata

If VC constraints are such that:

$x < 2$



$x < y + 2$



No arithmetic operation involving two VCs

~~$x + y < z$~~

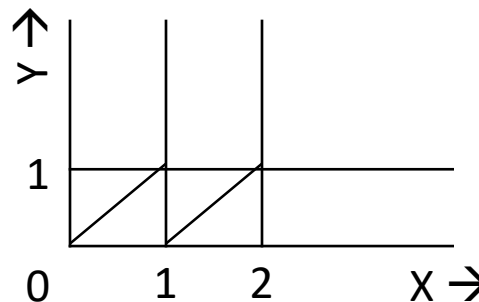
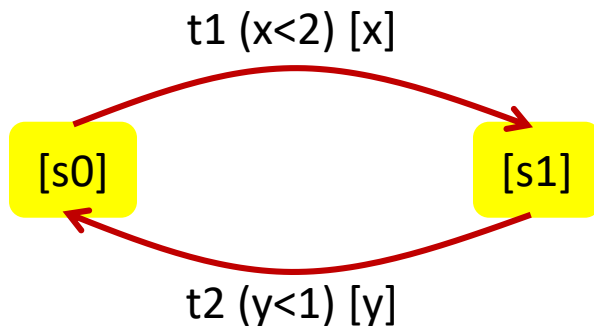
No multiplication operation involving a VC

~~$2x < 3$~~

No irrational constants in constraints

~~$x < \sqrt{2}$~~

Time can be partitioned into equivalence regions



28 regions

- Corner points (6)
- Line segments (14)
- Spaces (8)

Recap: Region construction

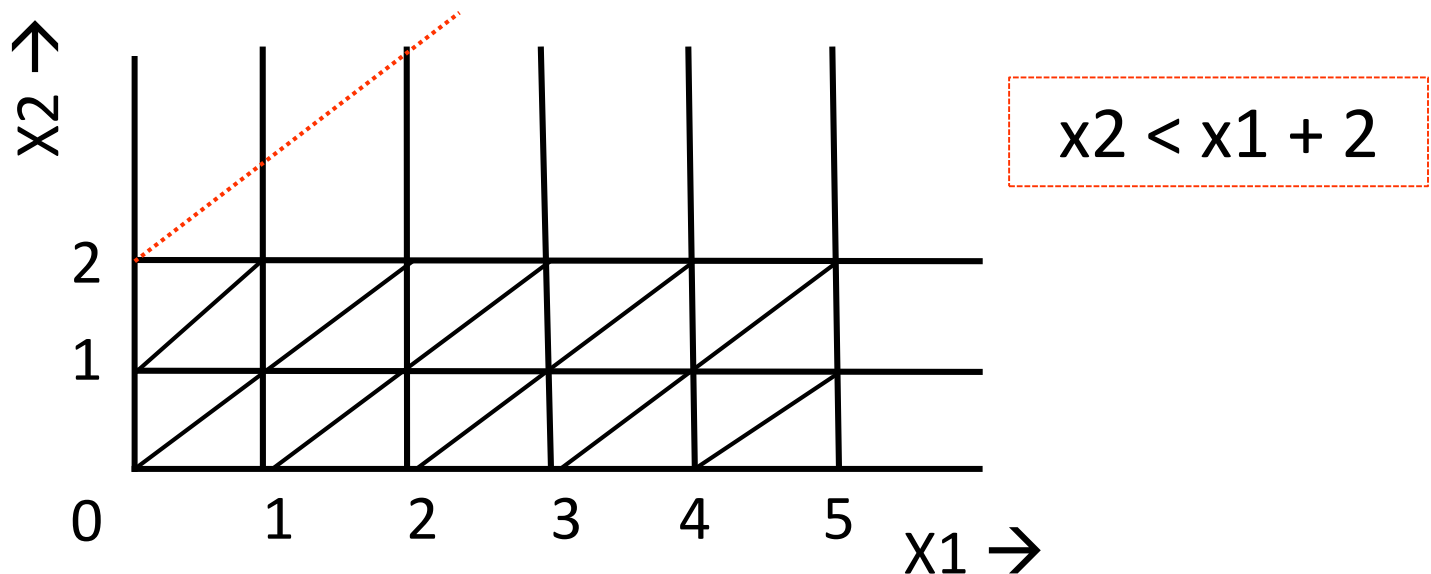
If integer constants and simple constraints (e.g., $x < c$)

Straight lines

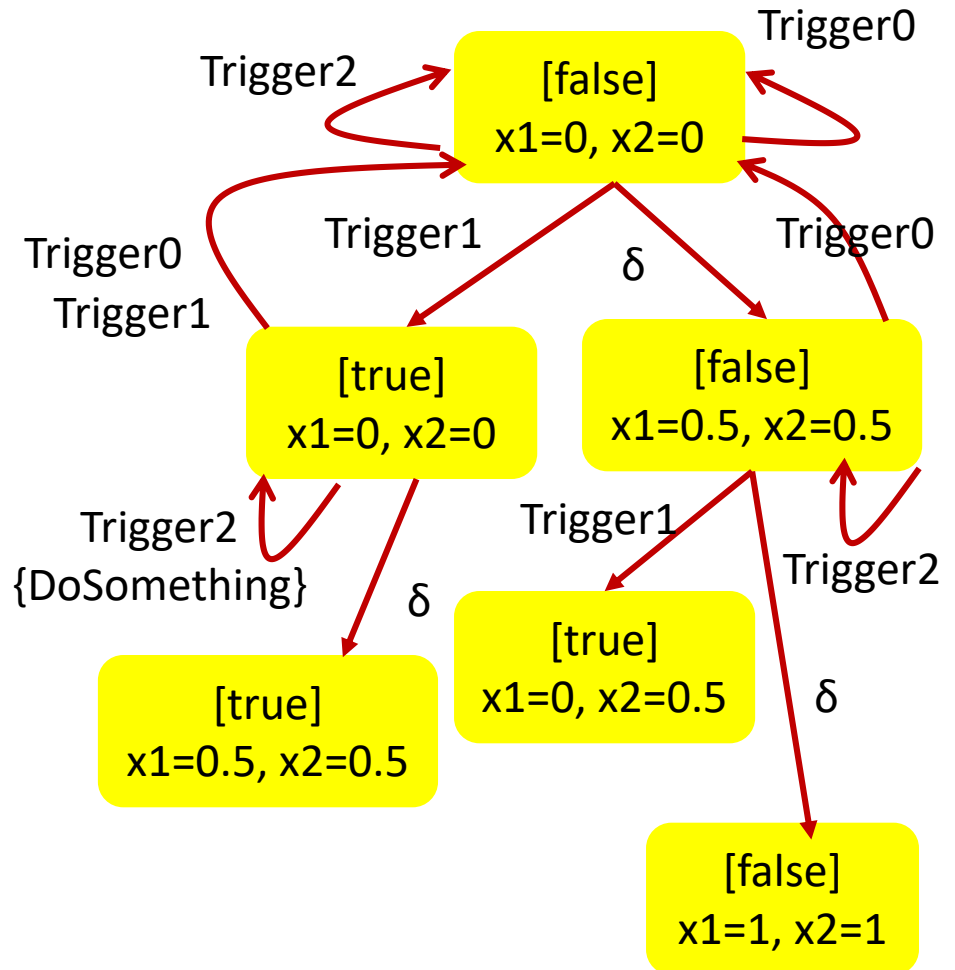
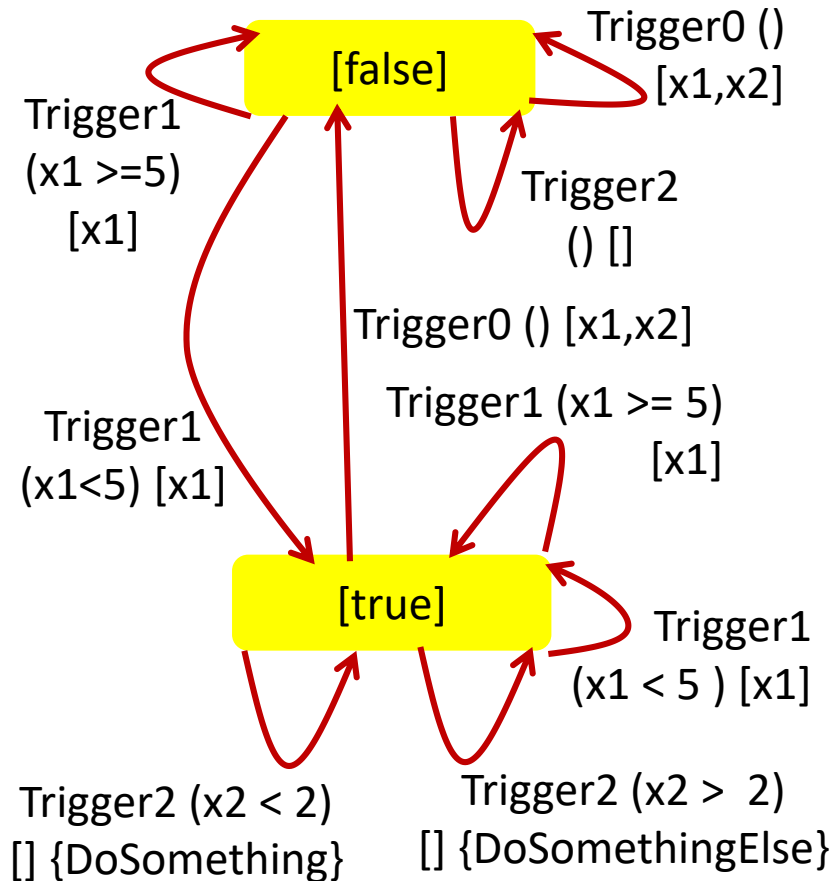
$$\forall x: \{x = c \mid c = 0, 1, \dots, c_x\}$$

Diagonals lines

$$\forall x, y: \{\text{fract}(x) = \text{fract}(y) \mid x < c_x, y < c_y\}$$



Recap: Exploring a TA



Exploring control programs with TAs

1. Mapping time-related activity to VCs
2. Model devices
3. Construct time regions
4. Compute equivalent classes for inputs
5. Explore states

Mapping to VCs (1/4): Delay measurers

Trigger1:

...

tLast = Now

...

Trigger2:

...

if (Now - tLast < 60)

...

Trigger1:

...

VC_tLast = 0

...

Trigger2:

...

if (VC_tLast < 60)

...

Mapping to VCs (2/4): Periodic timers

```
timer1.Period = 600  
timer1.Event += Timer1Fired  
...
```

Timer1Fired:

```
...
```

```
VC_timer1 = 0
```

```
...
```

VC_timer1 == 600:

```
...
```

```
VC_timer1 = 0
```

Mapping to VCs (2/4): Delayed actions

Trigger1:

...

timer1.Start(600)

...

timer1.Fired:

...

Trigger1:

...

VC_timer1 = 0

...

VC_timer1 == 600:

...

Mapping to VCs (4/4): Sleep calls

Trigger:

...

Sleep(10)

...

Trigger:

... // pre-sleep actions

VC_sleeper = 0

VC_sleeper == 10:

... // post-sleep actions

Reducing the number of VCs: Combining periodic timers

```
timer1.Period = 600  
timer1.Event += Timer1Fired  
timer2.Period = 800  
timer2.Event += Timer2Fired  
...
```

Timer1Fired:

...

Timer2Fired:

...

```
VC_timer = 0
```

...

```
VC_timer == 600:
```

...

```
VC_timer == 800:
```

...

```
VC_timer = 0
```

Reducing the number of VCs: Combining sleep calls

Trigger:

```
Act1()  
Sleep(5)  
Act2()  
Sleep(10)  
Act3()
```

Trigger:

```
Act1()  
VC_sleeper = 0  
sleep_counter = 1;
```

VC_sleeper == 5:

```
Act2()
```

VC_sleeper == 15:

```
Act3()
```


Modeling devices

Model a device using one of more key value pairs

- Motion sensor: Single key with binary value
- Dimmer: Single key with values in range [0..99]
- Thermostat: Multiple keys

Keys can be notifying or non-notifying

- Triggers are used for notifying keys

Queries for values are treated as program inputs

Limitations of device modeling

Values can change arbitrarily

Key value pairs of a device are independent

Different devices are independent

Constructing time regions

1. Extract VC constraints using symbolic execution
2. Construct time regions using the constraints

Trigger0:

```
tTrigger1 = Now
tTrigger2 = Now
trigger1Seen = false
```

Trigger1:

```
if (Now - tTrigger1 < 5)
    trigger1Seen = true
tTrigger1 = Now
```

Trigger2:

```
if (trigger1Seen)
    if (Now - tTrigger2 < 2)
        DoSomething()
    else
        DoSomethingElse()
```

Exploration using TA

Region state = Variables values + VC region + ready timers

1. exploredStates = { }
2. unexploredStates = { $S_{initial}$ }
3. **While** (unexploredStates $\neq \phi$)
4. S_i = PickNext(UnexploredStates)
5. **foreach** event in Events, S_i .ReadyTimers
6. **foreach** input in Inputs
7. S_o = Compute(S_i , event, input)
8. if ($S_o \notin$ exploredStates) unexploredStates.Add(S_o)
9. **if** (S_i .ReadyTimers = ϕ)
10. S_o = AdvanceRegion(S_i) //also marks ReadyTimers
11. **if** ($S_o \notin$ exploredStates) unexploredStates.Add(S_o)
12. exploredStates.Add(S_i)

Optimization: Predicting successor states

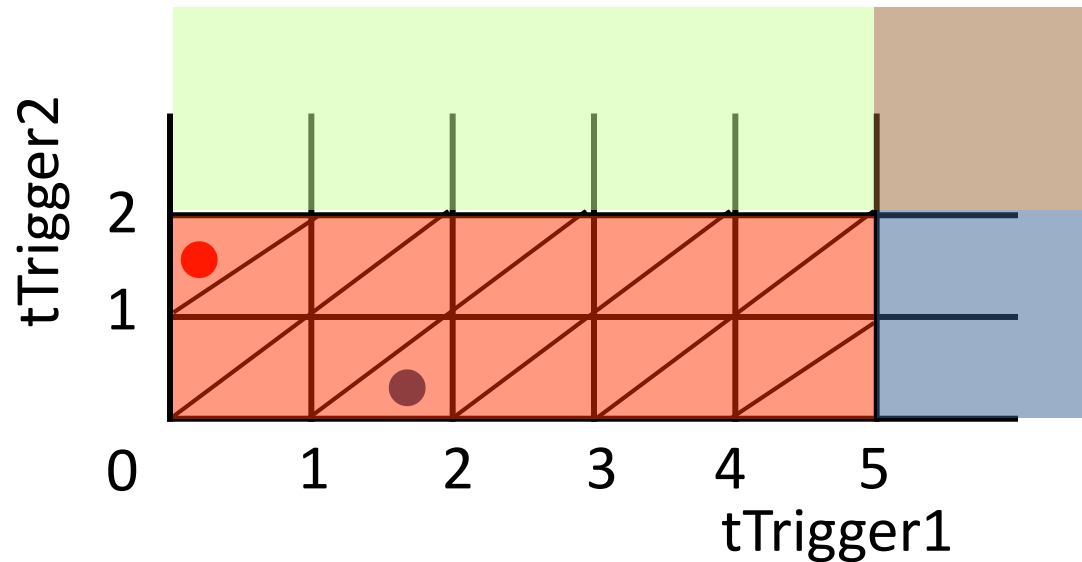
Observation: Multiple region states can have identical response to a trigger

Trigger1:

```
if (x1 < 5)
    trigger1Seen = true
x1 = 0
```

Trigger2:

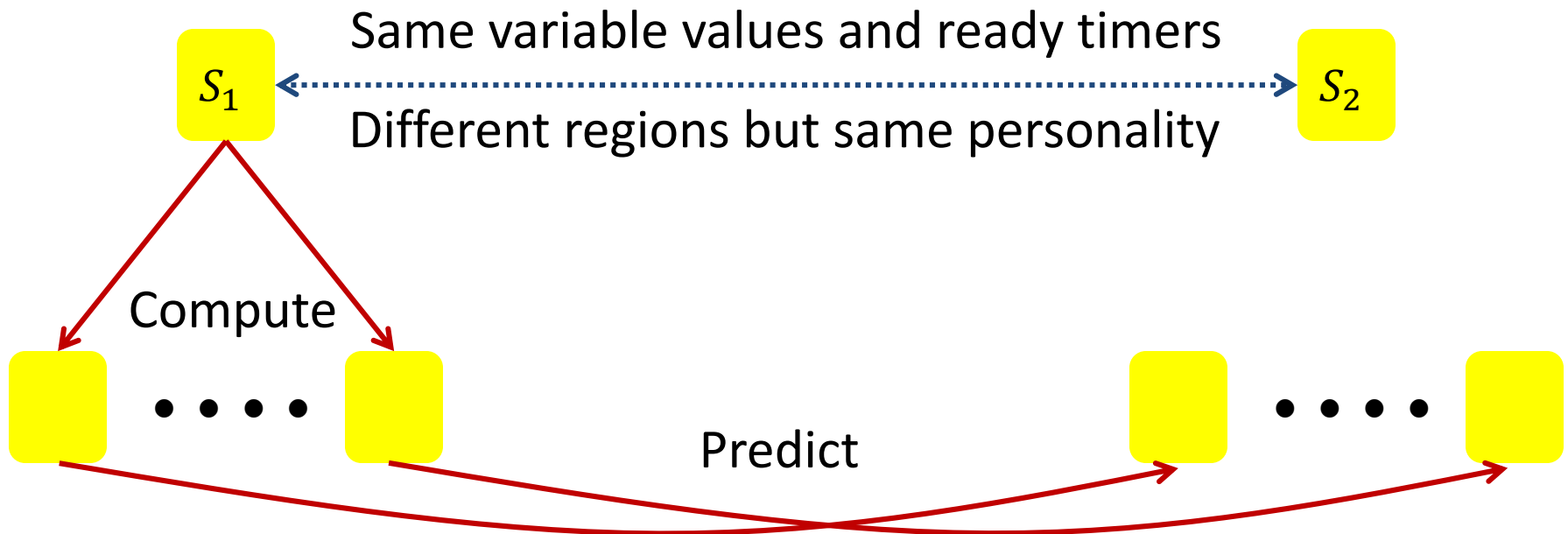
```
if (trigger1Seen)
    if (x2 < 2)
        DoSomething()
    else
        DoSomethingElse()
```



Optimization: Predicting successor states

Observation: Multiple region states can have identical response to a trigger

Clock personality: region's evaluation of clock constraints

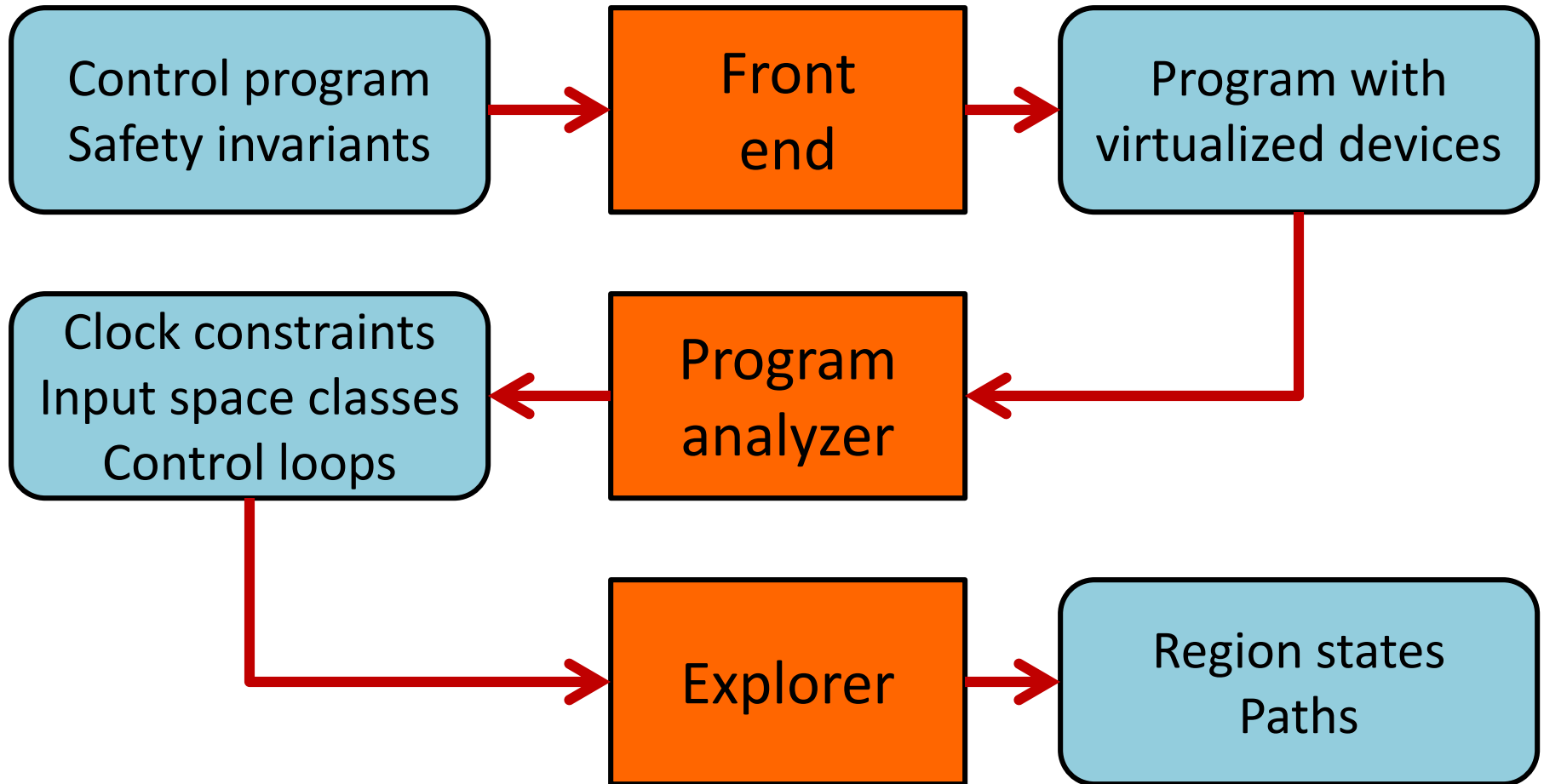


Optimization: Independent control loops

Observation: Control programs tend to have multiple, independent control loops

1. Determine independent sets of variables
2. Explore independent sets independently

DeLorean



Demo

Evaluation on ten real home automation programs

| | type | #rules | #devs | SLoC | #VCs | GCD (s) |
|-----|-------------|--------|-------|------|------|---------|
| P1 | OmniPro | 6 | 3 | 59 | 2 | 7200 |
| P2 | Elk | 3 | 3 | 75 | 2 | 1800 |
| P3 | MiCasaVerde | 6 | 29 | 143 | 2 | 300 |
| P4 | Elk | 13 | 20 | 193 | 5 | 5 |
| P5 | ActiveHome | 35 | 6 | 216 | 14 | 5 |
| P6 | mControl | 10 | 19 | 221 | 4 | 5 |
| P7 | OmniIle | 15 | 27 | 277 | 6 | 60 |
| P8 | HomeSeer | 21 | 28 | 393 | 10 | 2 |
| P9 | ISY | 25 | 51 | 462 | 6 | 60 |
| P10 | ISY | 90 | 39 | 867 | 6 | 10 |

Example bugs

P9-1: Lights turned on even in the absence of motion

- Bug in conditional clause: used OR instead of AND

P9-2: Lights turned off between sunset and 2AM

- Interaction between rules that turned lights on and off

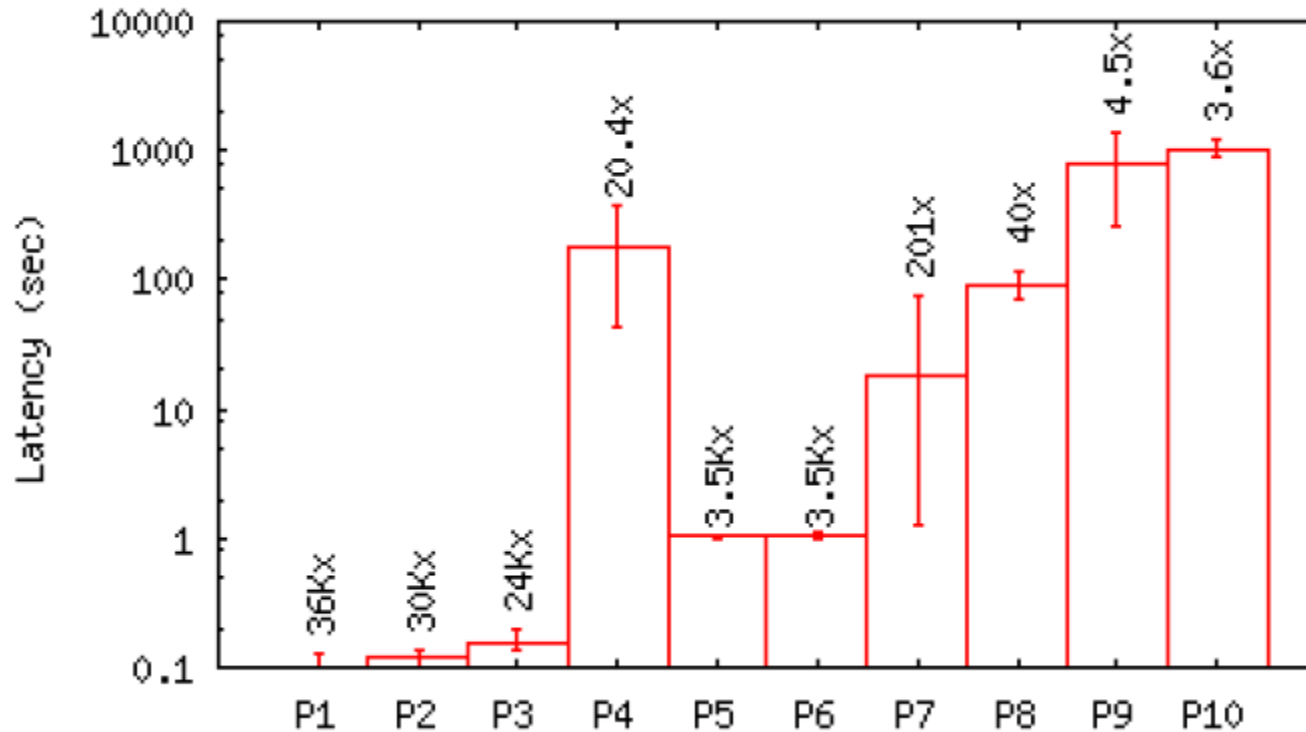
P10-1: Dimmer wouldn't turn on despite motion

- No rule to cover a small time window

P10-2: One device in a group behaved differently

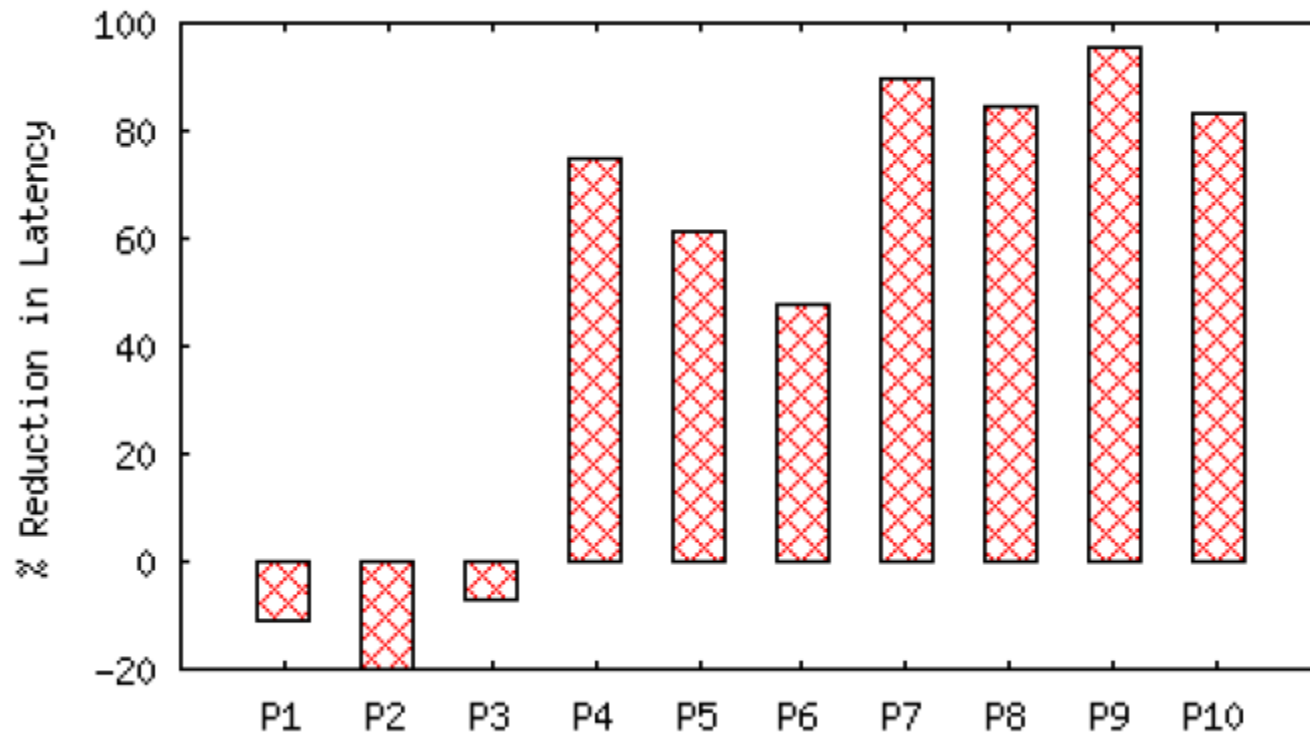
- Missing reference to the device in one of the rules

Performance of exploration



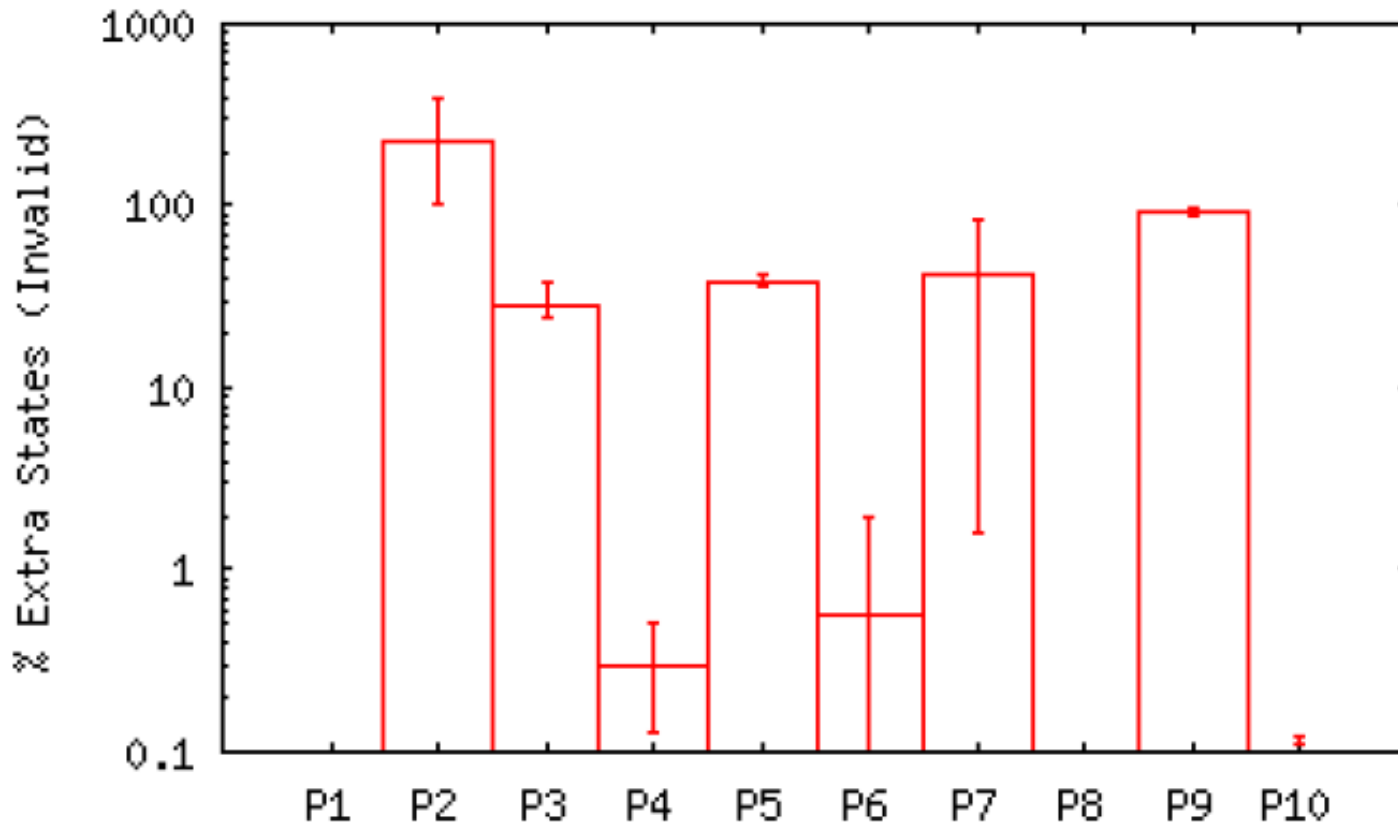
Time to “fast forward” the home by one hour

Benefit of successor prediction



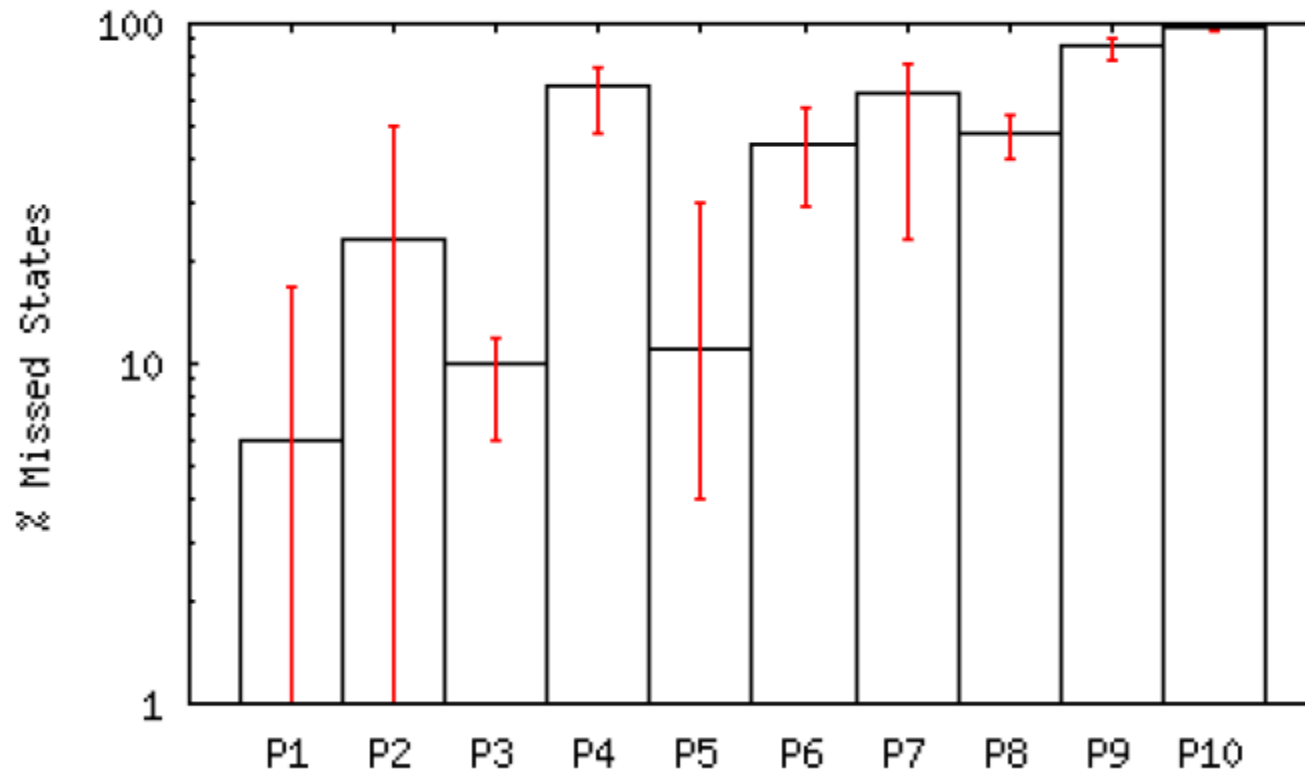
Successor prediction yields significant advantage

Comparison with untimed model checking



Untimed model checking reaches many invalid states

Comparison with randomized testing



Random testing misses many valid states

Exploring OpenFlow programs

| | #devs | SLoC | #VCs | GCD |
|--------------------------------------|-----------------------|------|----------|-----|
| MAC-Learning Switch (PySwitch) | 2 hosts, 2 sw, 1 ctrl | 128 | ≥ 6 | 1 |
| Web Server Load Balancer | 3 hosts, 1 sw, 1 ctrl | 1307 | ≥ 4 | 1 |
| Energy-Efficient Traffic Engineering | 3 hosts, 3 sw, 1 ctrl | 342 | ≥ 8 | 2 |

Additional challenges in OF programs

packetIn:

```
timer = new Timer(5s)
Insert(timer, inPkt.src, inPkt.dst)
```

Dynamically created VCs

Variable number of VCs along different paths

Open problems

Handling communicating control programs

Exploring all possible topologies

Summary

Control programs are tricky to debug

- Interaction between rules
- Large space of inputs
- Intimate dependence on time

These challenges can be tackled using

- Systematic exploration (model checking)
- Symbolic execution to find equivalent input classes
- Timed automata based exploration (equivalent times)