Systematically exploring control programs

Ratul Mahajan

Microsoft Research

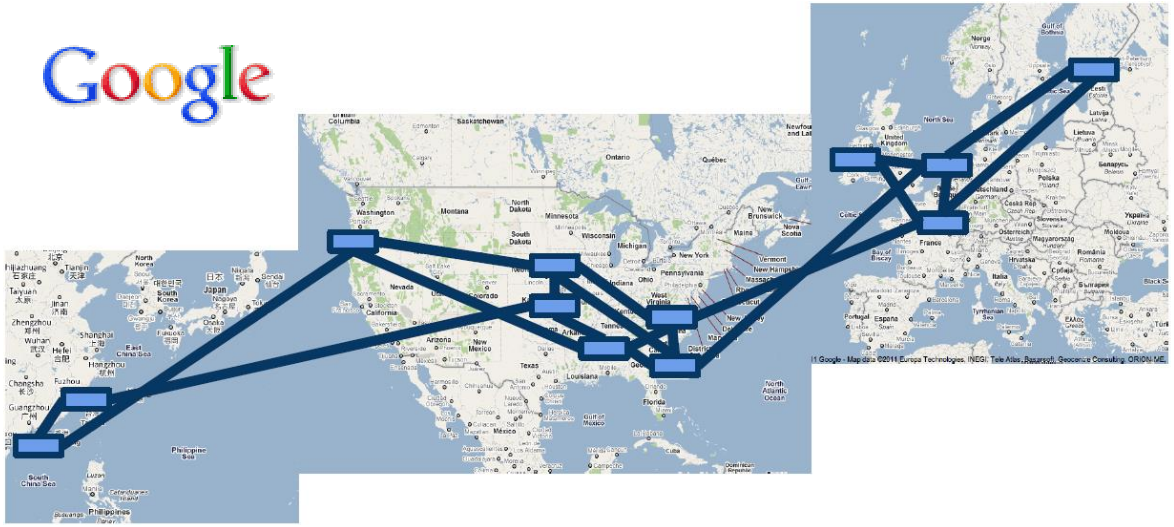
Joint work with Jason Croft,
Matt Caesar, and Madan Musuvathi

Control programs run networks

From the smallest to the largest

Control programs run networks

From the smallest 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

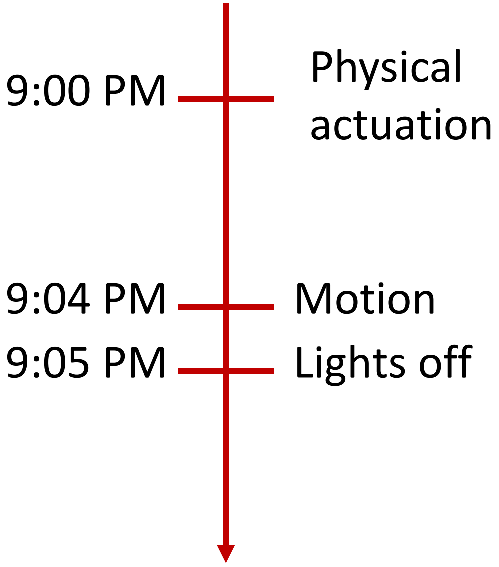
One nice morning in the summer

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 sec
          && 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);
```



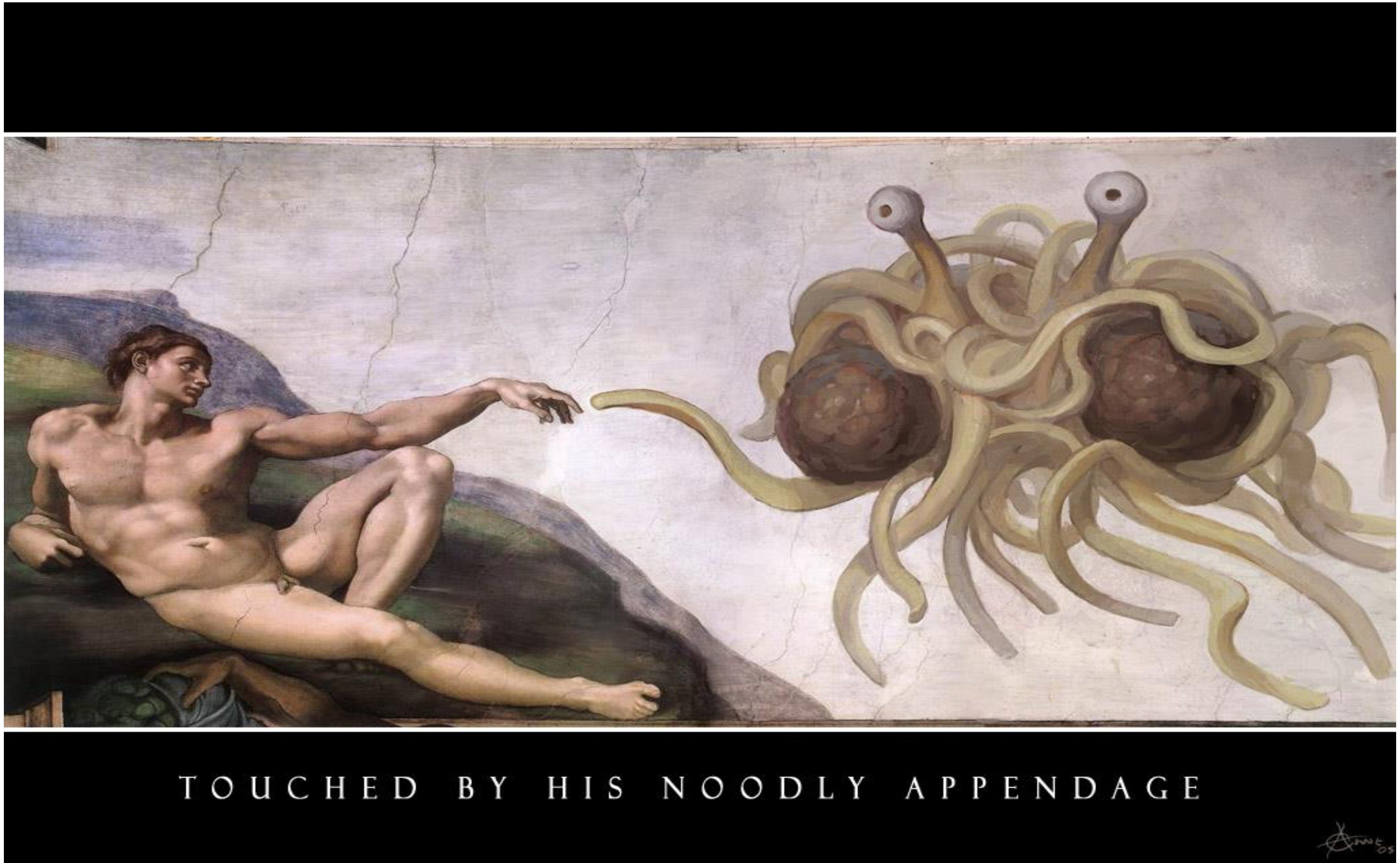
Cross-rule interactions

Intimate dependence on time

Many possible environments

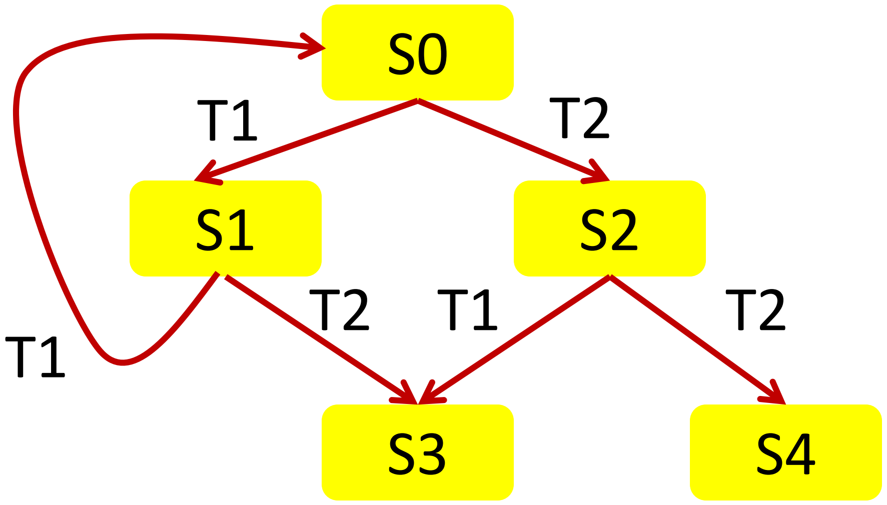
Systematically exploring programs

Exploring programs using FSMs



Exploring programs using FSMs

1. Decide what are states and transitions
2. Explore all transitions from all states

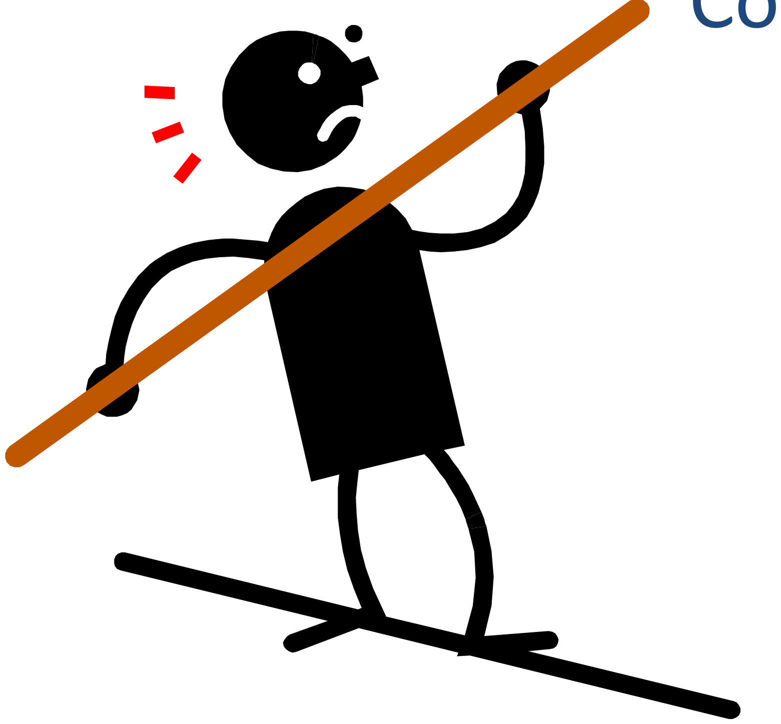


Challenge: Dependence on time

```
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()
```

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

The tyranny of “all possible times”



Completeness

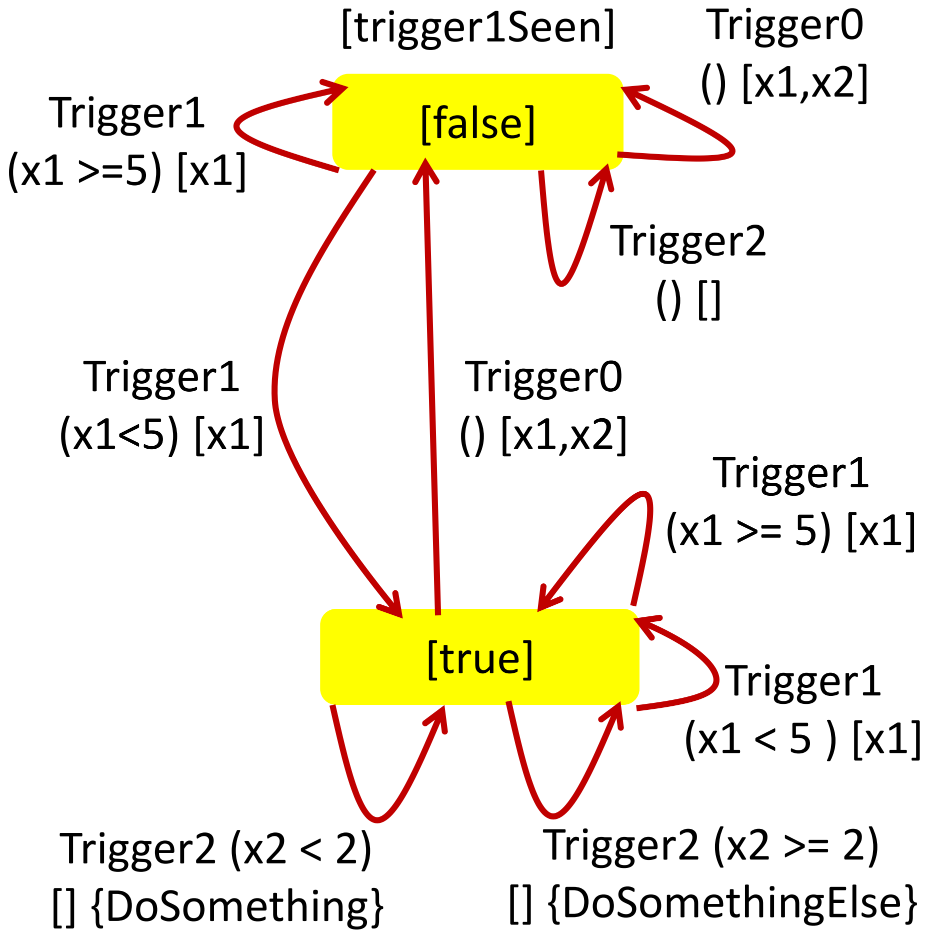
Tractability

Timed automata

FSM (states, transitions) plus:

- 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: $\boxed{x < 2}$ ✓ $\boxed{x < y + 2}$ ✓

- No arithmetic operation involving two VCs $\boxed{\xcancel{x + y < z}}$
- No multiplication operation involving a VC $\boxed{\xcancel{2x < 3}}$
- No irrational constants in constraints $\boxed{\xcancel{x < \sqrt{2}}}$

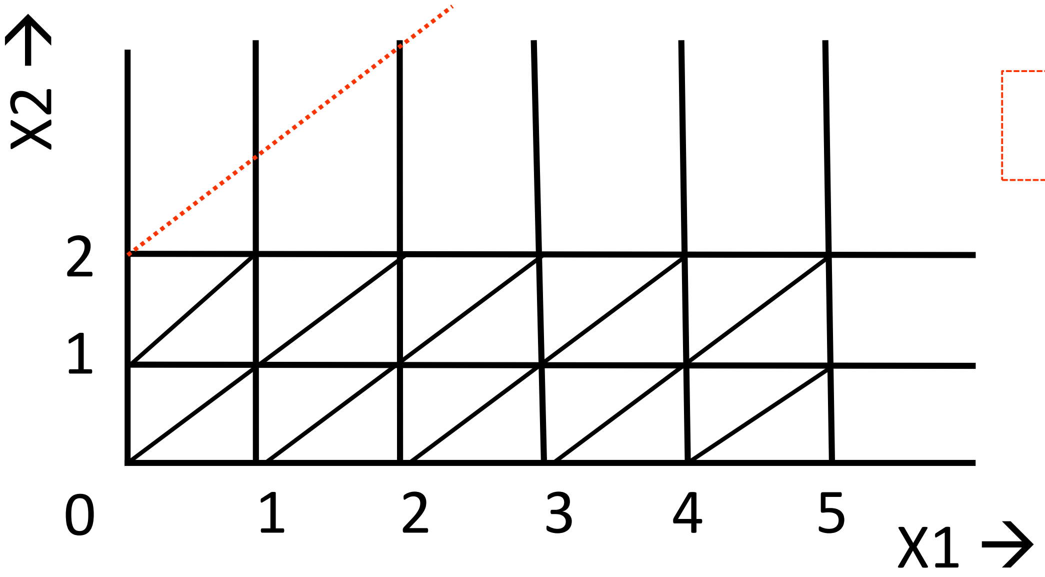
Time can be partitioned into equivalence regions

Region construction

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

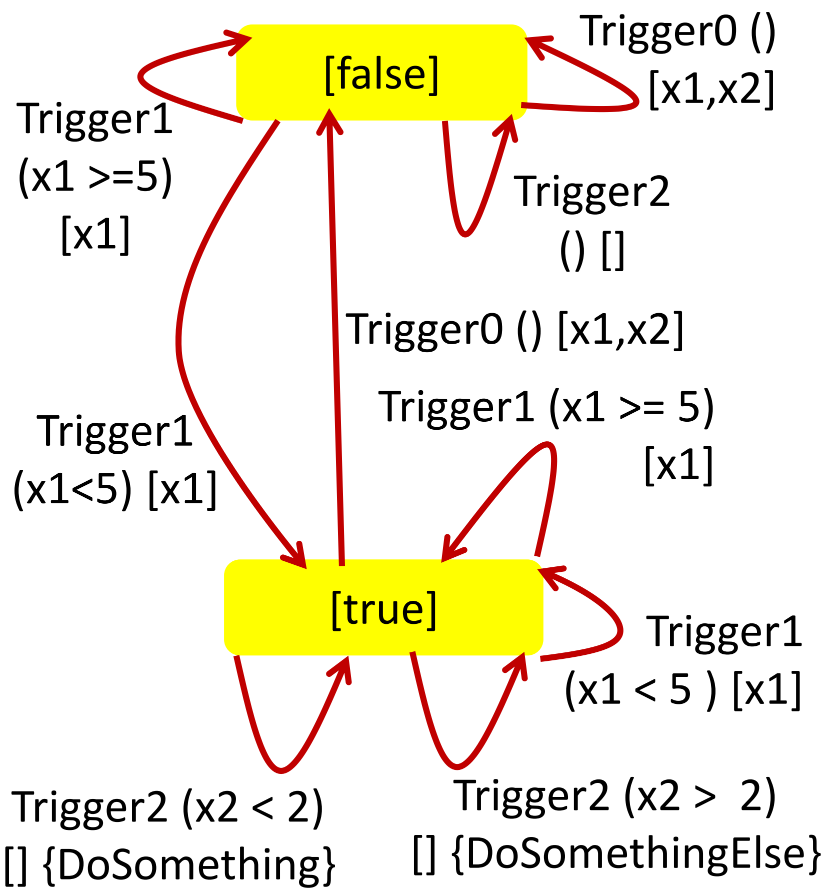
Straight lines $\forall x\colon \{x = c \mid c = 0, 1, \ldots c_x\}$

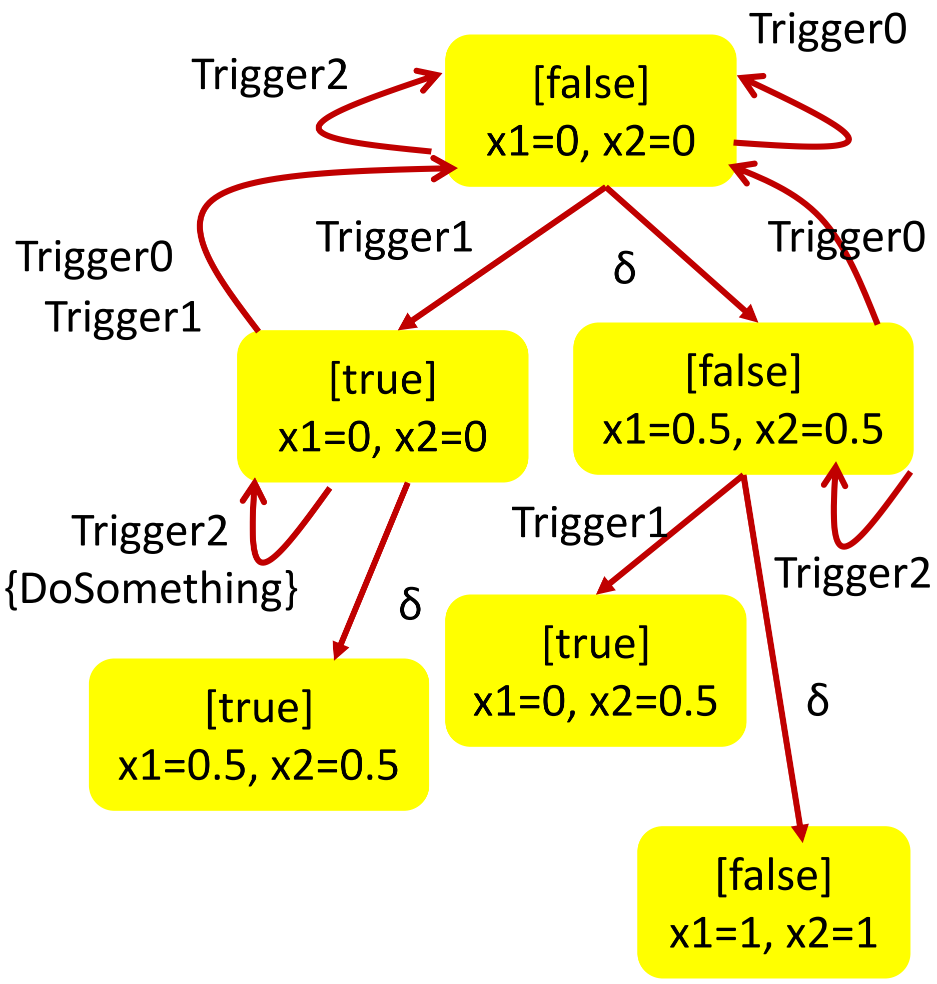
Diagonals lines $\forall x, \mathrm{y}\colon \{\mathrm{fract}(x) = \mathrm{fract}(\mathrm{y}) | x < \mathrm{c}_x, \mathrm{y} < \mathrm{c}_\mathrm{y}\}$



x2 < x1 + 2

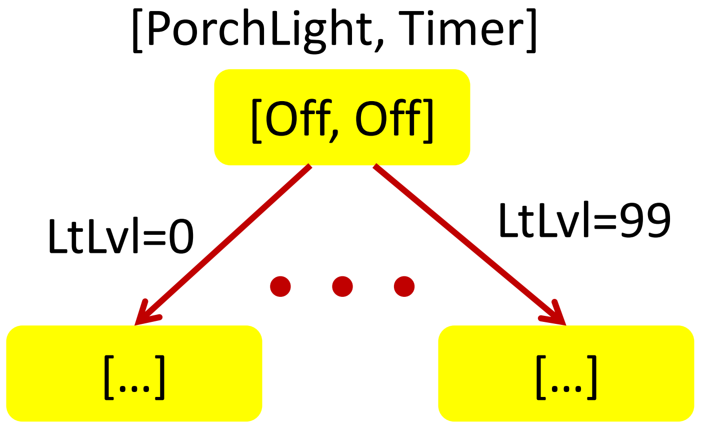
Exploring a TA: Region automata

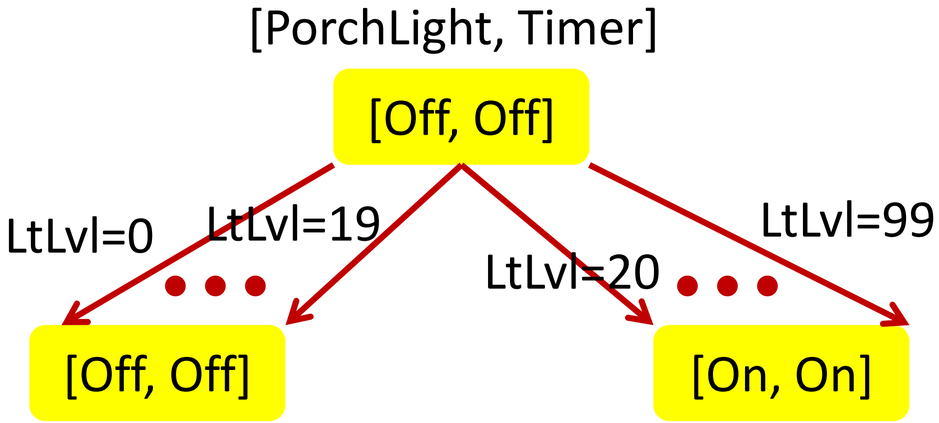




Challenge: Many possible environments

```
motionPorch:
  if (lightLevel < 20)
    porchLight.Set(On)
    timer.Start(10 mins)
porchLight.On:
  timer.Start(5 mins)
timer.Fired:
  porchLight.Set(Off)
```

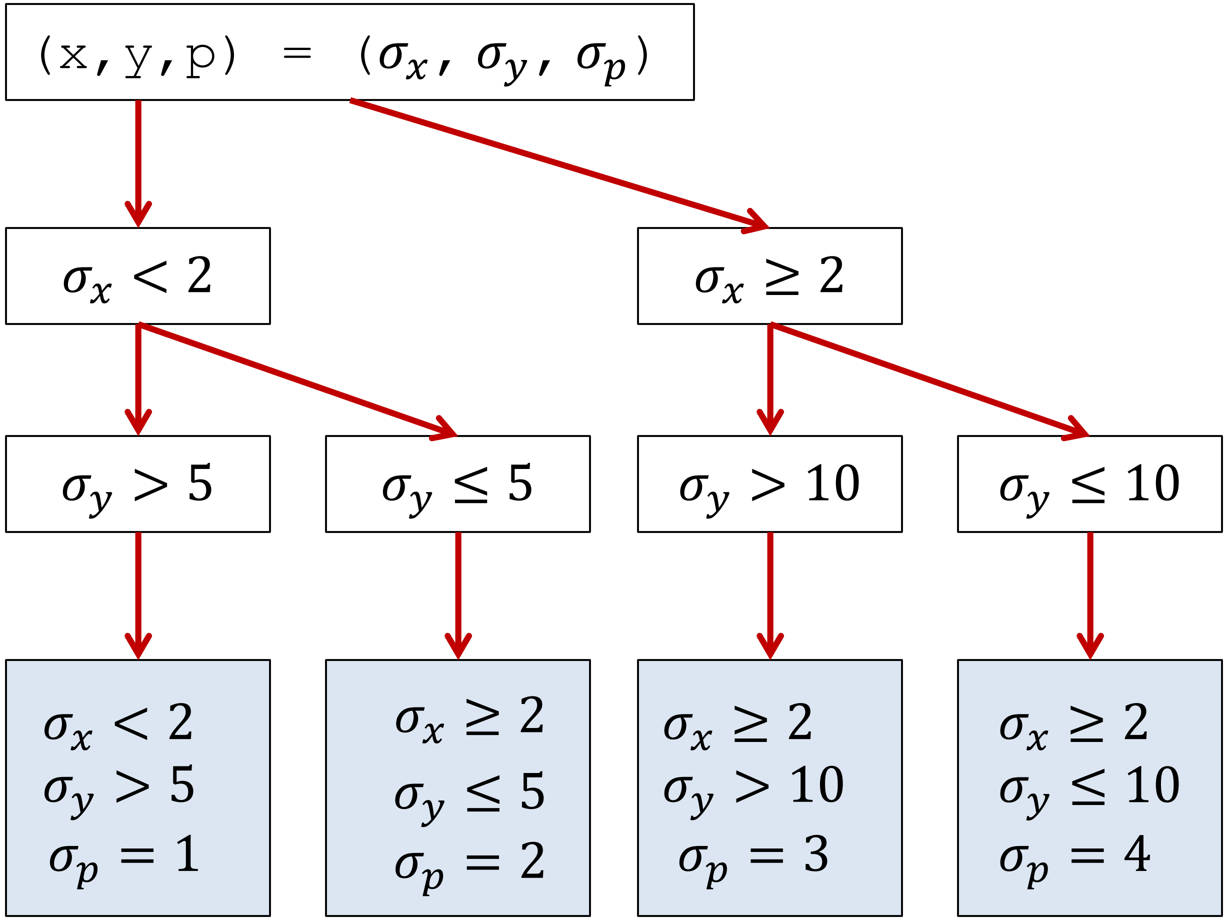




To explore comprehensively, must we consider all possible environments?

Symbolic execution

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



(x,y,p) = $(\sigma_x, \sigma_y, \sigma_p)$

$\sigma_x < 2$

$\sigma_x \geq 2$

$\sigma_y > 5$

$\sigma_y \leq 5$

$\sigma_y > 10$

$\sigma_y \leq 10$

$\sigma_x < 2$
$\sigma_y > 5$
$\sigma_p = 1$

$\sigma_x \geq 2$
$\sigma_y \leq 5$
$\sigma_p = 2$

$\sigma_x \geq 2$
$\sigma_y > 10$
$\sigma_p = 3$

$\sigma_x \geq 2$
$\sigma_y \leq 10$
$\sigma_p = 4$

Finding equivalent environments

1. Symbolically execute each trigger
2. Find environmental conditions that lead to same *state*

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

$\text{LtLvl} < 20$ $\text{LtLvl} \geq 20$

$\text{LtLvl} = *$

$\text{LtLvl} = *$

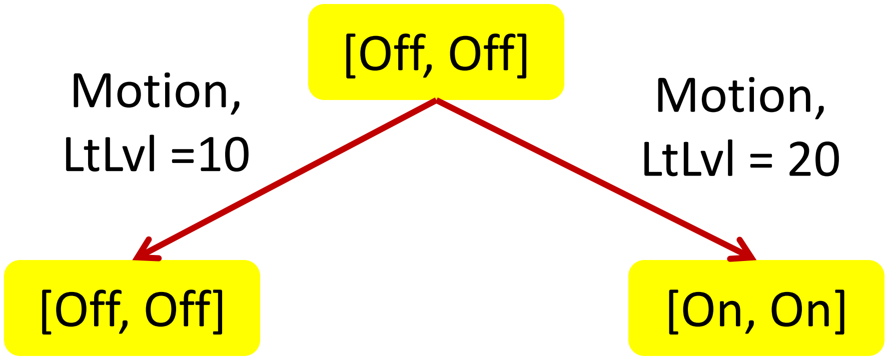
Efficiently exploring environments

Pick random values in equivalent classes

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

porchLight.On:
  timer.Start(5 mins)

timer.Fired:
  porchLight.Set(Off)
```

LtLvl $<$ 20

LtLvl $\geq$ 20



[Off, Off]

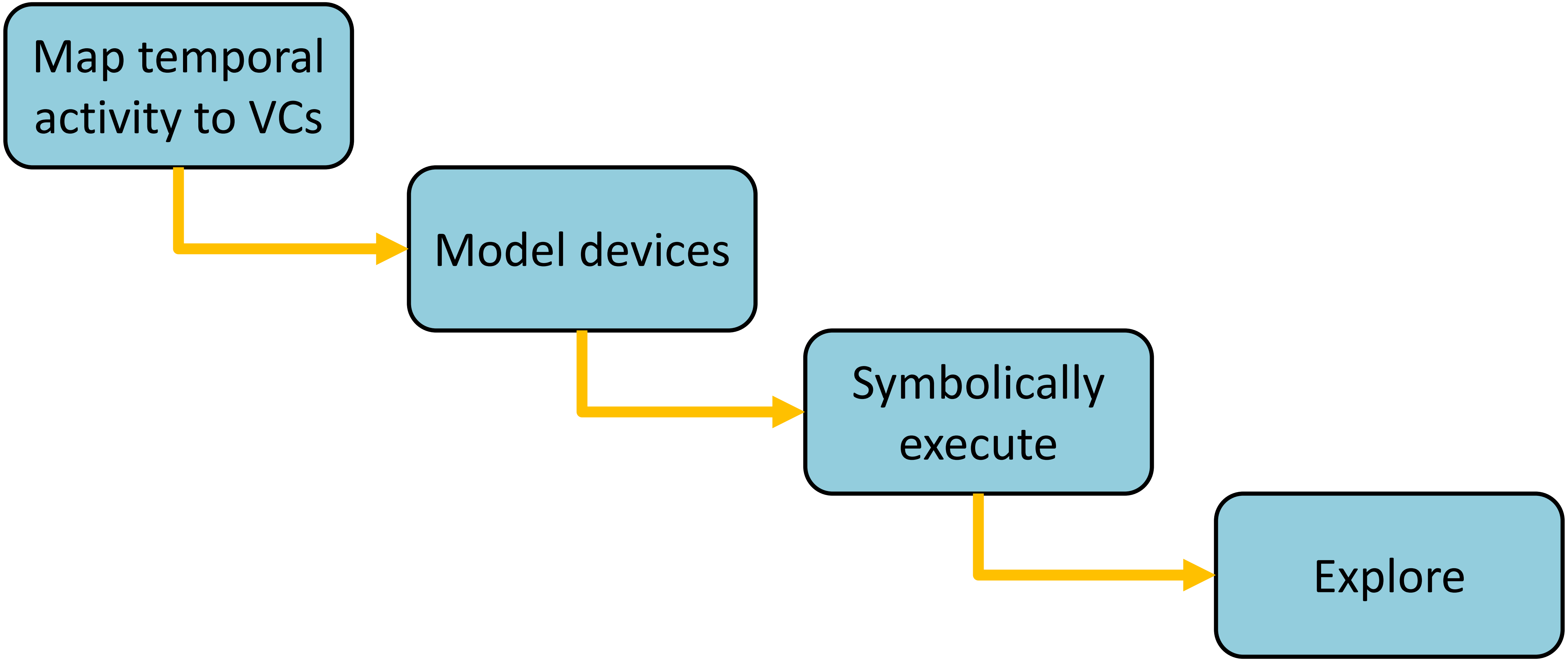
Motion, LtLvl =10

Motion, LtLvl = 20

[Off, Off]

[On, On]

DeLorean: A tool to explore control programs



Map temporal activity to VCs

Model devices

Symbolically execute

Explore

Mapping to VCs (1/2): Delay measurers

```
Trigger1:
   ...
   tLast = Now
   ...

Trigger2:
   ...
   if (Now - tLast < 60)
   ...
```

```
Trigger1:
   ...
   VC_tLast = 0
   ...

Trigger2:
   ...
   if (VC_tLast < 60)
   ...
```

Mapping to VCs (2/2): Timers

```
Trigger1:
   ...
   timer1.Start(600)
   ...

timer1.Fired:
   ...
```

```
Trigger1:
   ...
   VC_timer1 = 0
   ...

VC_timer1 == 600:
   ...
```

Reducing the number of VCs: Combining 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
```

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 environmental condition

Limitations of device modeling

Values can change arbitrarily

Key value pairs of a device are independent

Different devices are independent

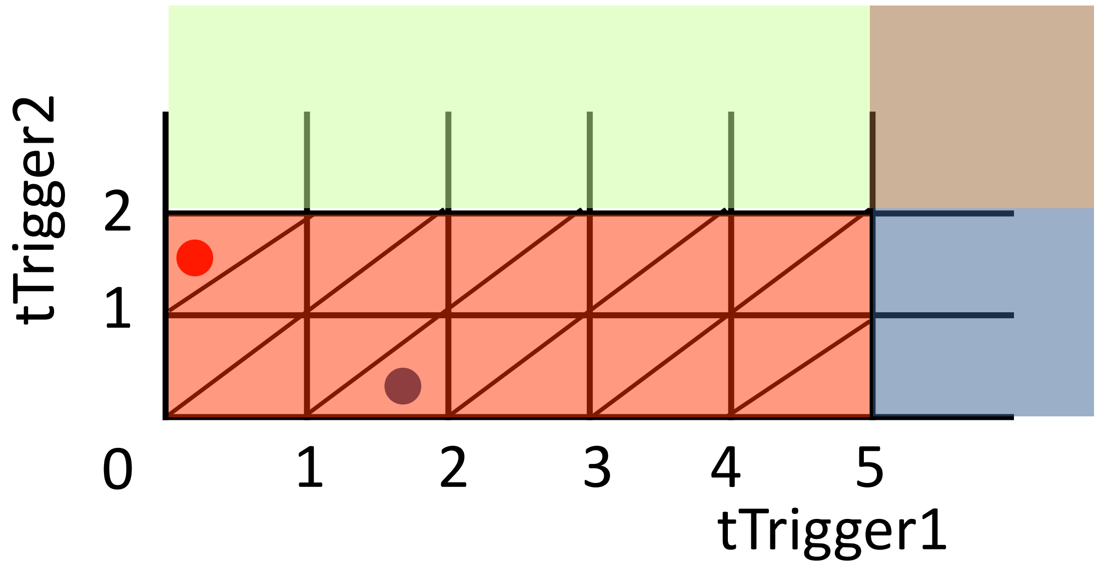
Exploration using TA

```
unexploredStates = {$S_{initial}$}          //state = Variables values + VC region + ready timers
exploredStates = {}
While (unexploredStates $\neq \phi$)
    $S_i$ = $PickNext$(unexploredStates)
    foreach trigger in Events, $S_i.ReadyTimers$
        foreach environment in Environments
            $S_o$ = $Compute$($S_i$, trigger, environment)
            if ($S_o \notin$ exploredStates) unexploredStates.Add($S_o$)
    if ($S_i.ReadyTimers = \phi$)
        $S_o$ = $AdvanceRegion$($S_i$)
        if ($S_o \notin$ exploredStates) unexploredStates.Add($S_o$)
    exploredStates.Add($S_i$)
```

Optimization: Predicting successor states

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

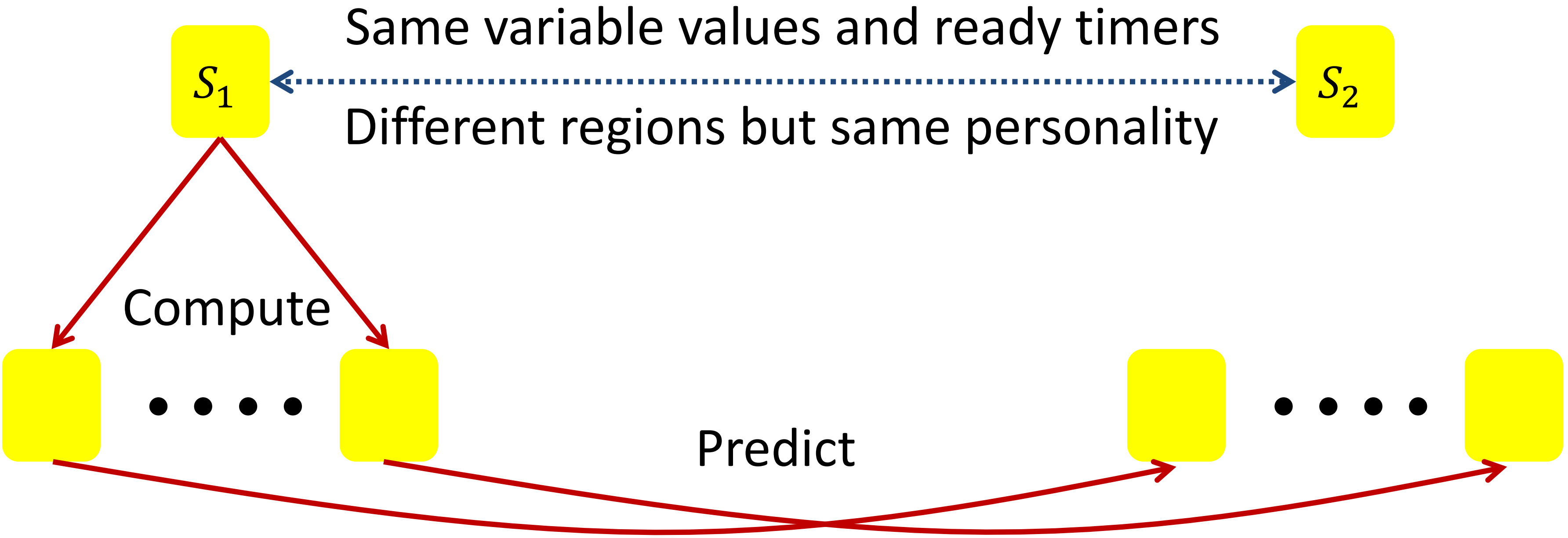
```
Trigger1:
  if (x1 < 5)
    trigger1Seen = true
  x1= 0
Trigger2:
  if (trigger1Seen)
     if (x2 < 2)
        DoSomething()
     else
        DoSomethingElse()
```



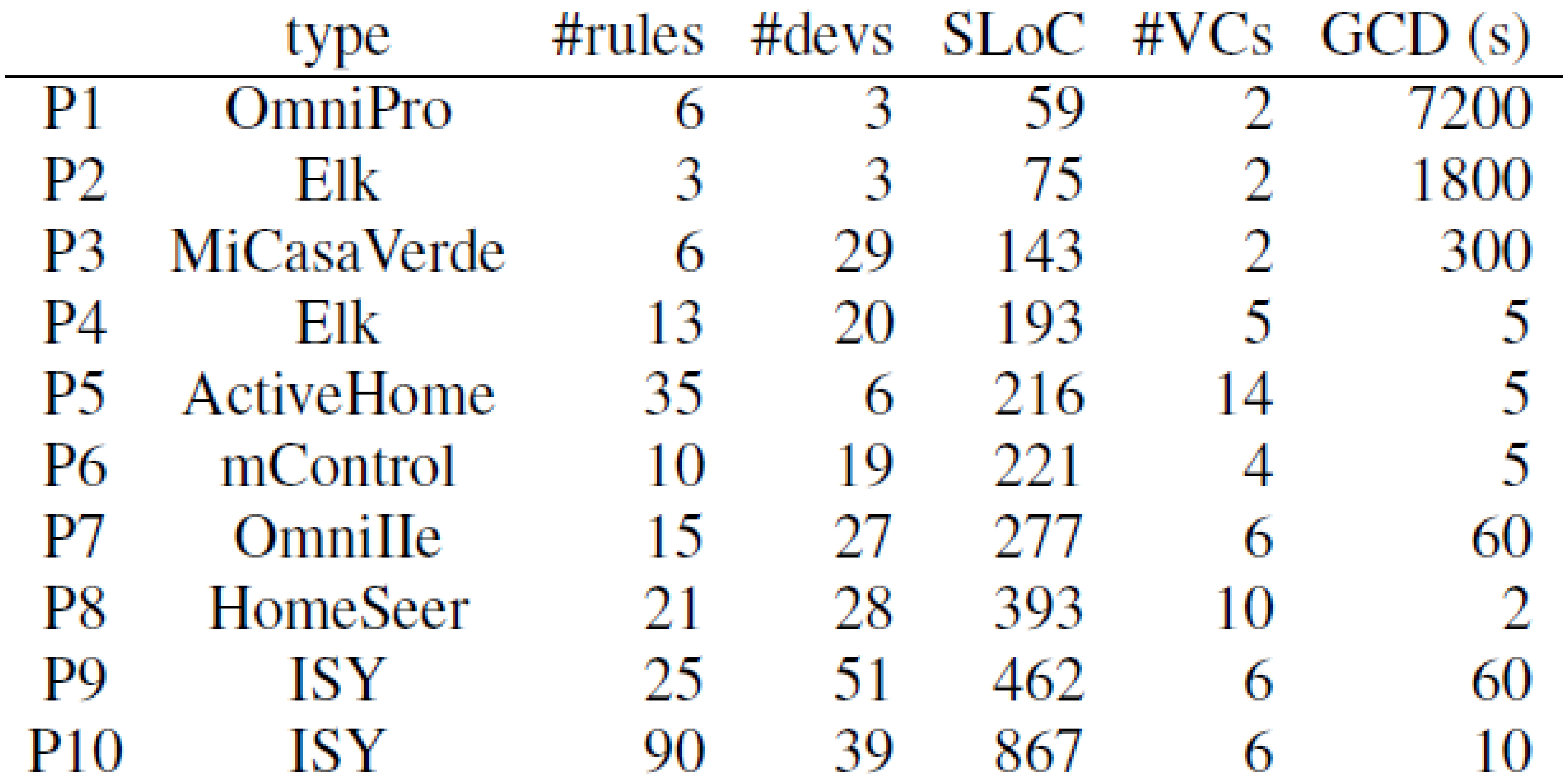
Optimization: Predicting successor states

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

Clock personality: region's evaluation of clock constraints


S_1
Same variable values and ready timers
Different regions but same personality
S_2
Compute
Predict

Evaluation on ten real home automation rograms



	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	OmniIIe	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

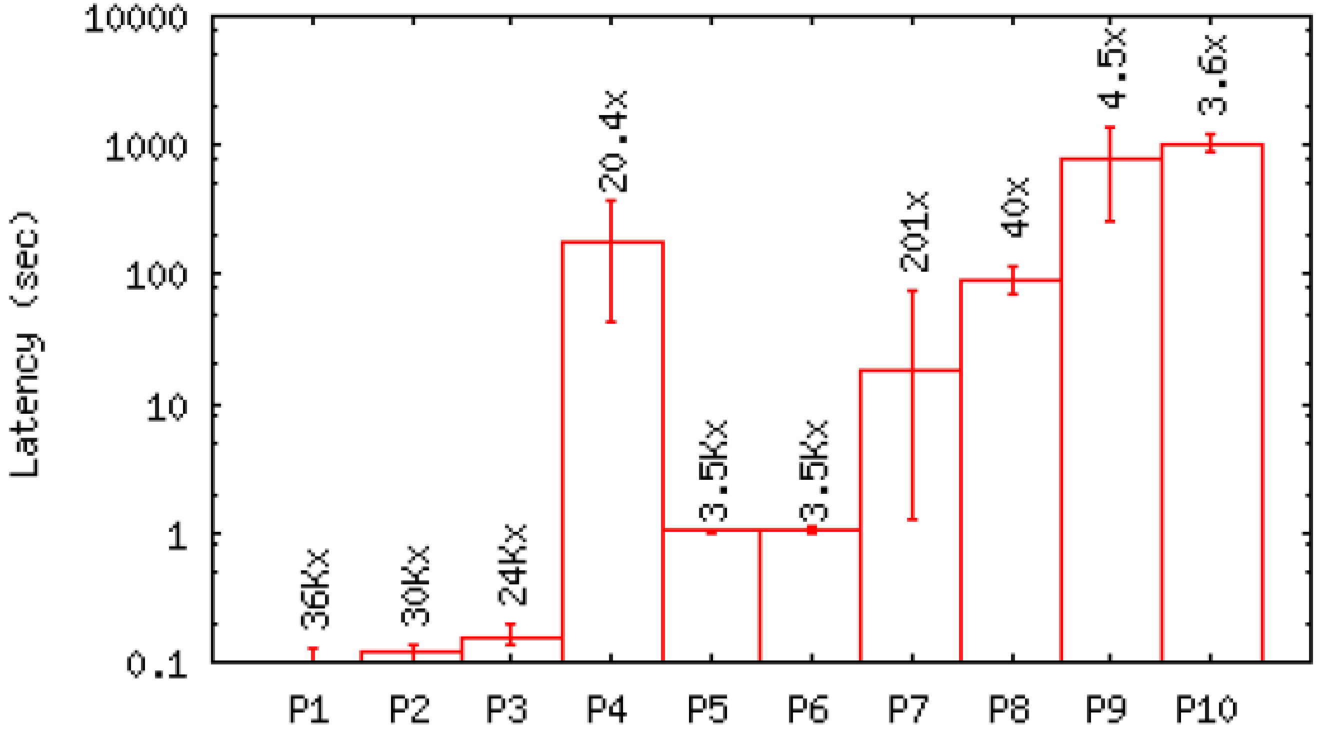
P10-1: Dimmer wouldn’t turn on despite motion

- No rule to cover a small time window

P10-2: One device in a group behaved differently

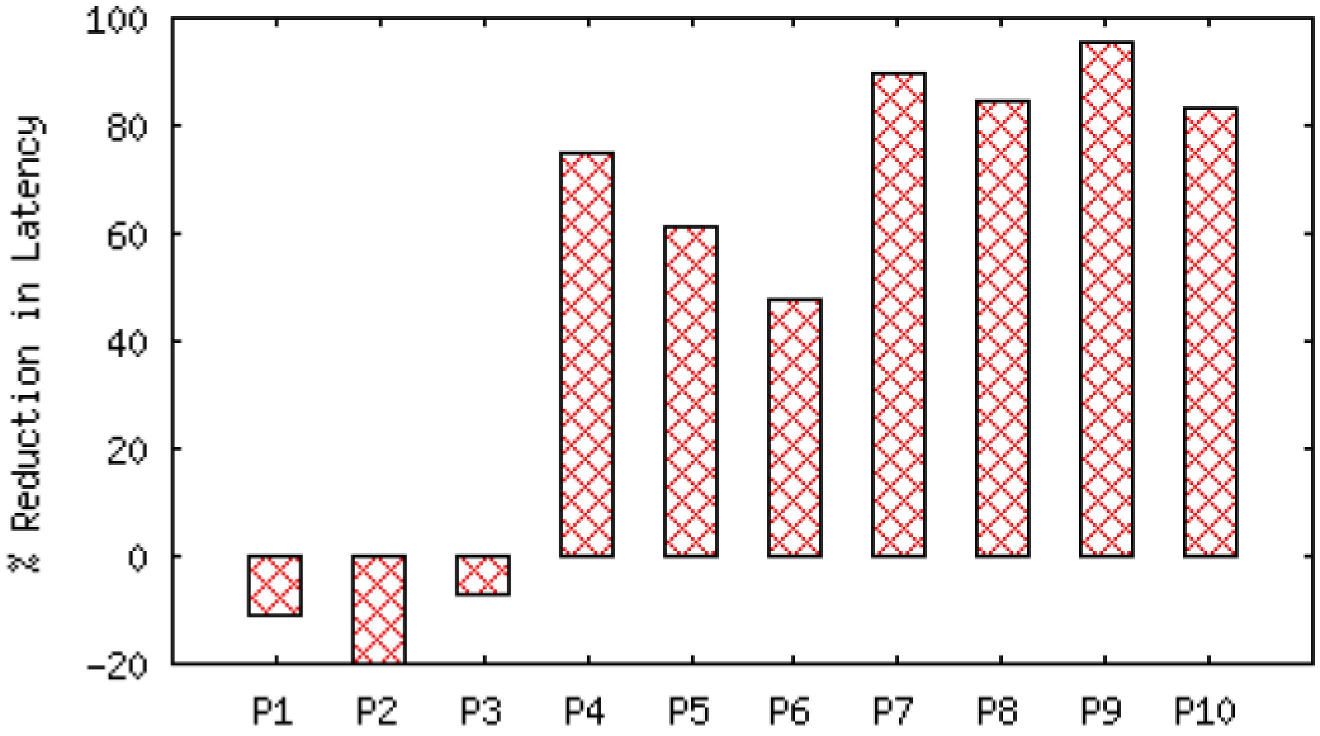
- Missing reference to the device in one of the rules

Performance of exploration



Time to “fast forward” the home by one hour

Benefit of successor prediction



Successor prediction yields significant advantage

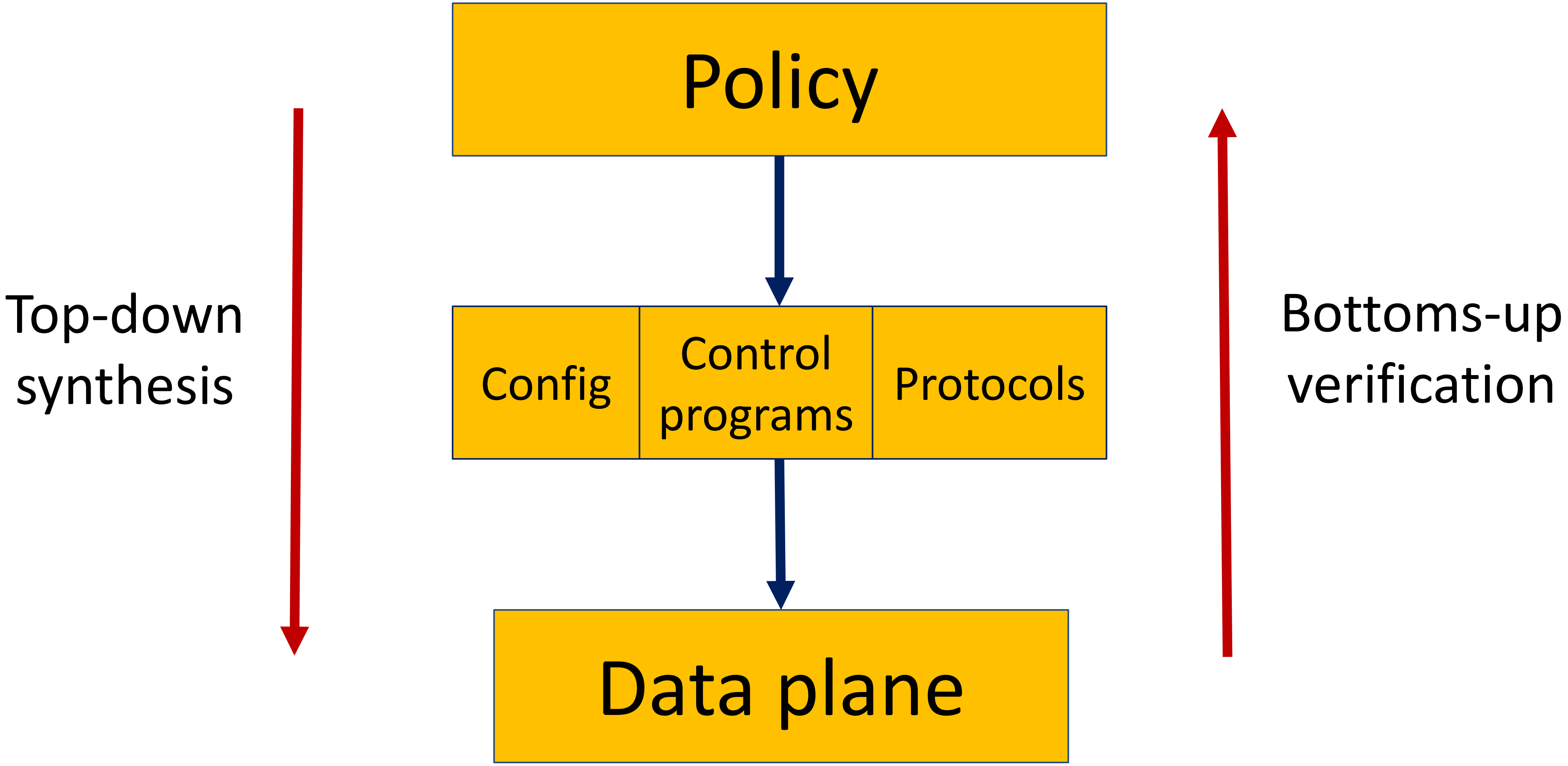
Ongoing work: Exploring OpenFlow programs

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

Scale is similar but additional challenges:

- Dynamically created VCs
- Variable number of VCs along different paths

Control program verification in context



Top-down synthesis

Policy

Config

Control programs

Protocols

Data plane

Bottoms-up verification

Summary

Control programs are tricky to debug

- Interaction between rules
- Intimate dependence on time
- Many possible environments

DeLorean addresses these challenges using

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

Backup

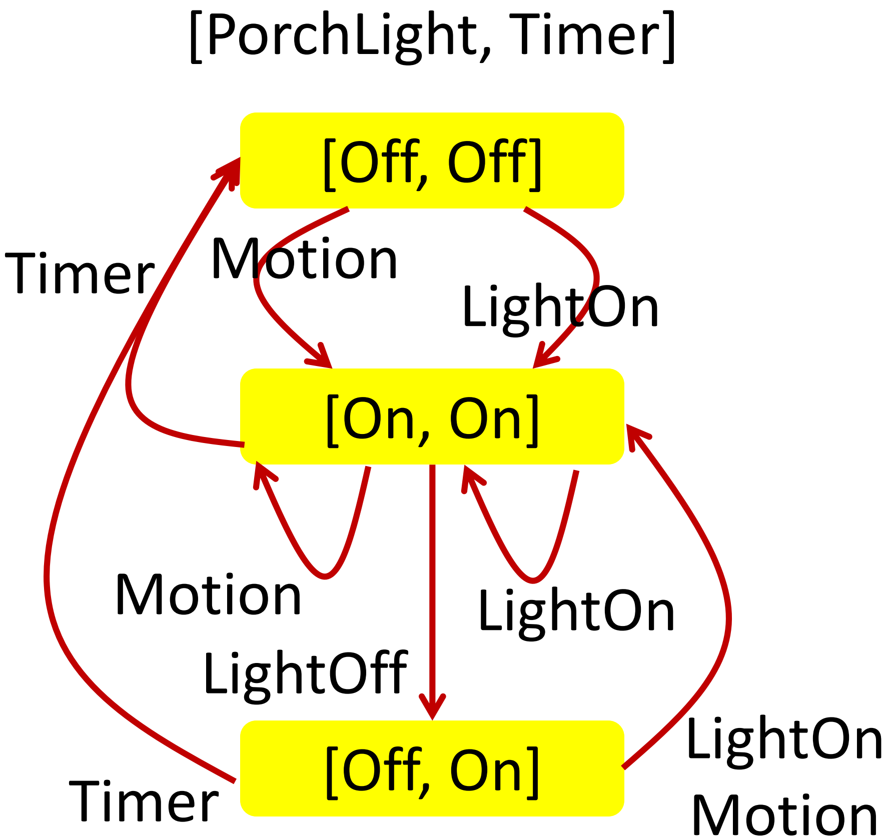
Two bug finding methods

Testing

Model checking

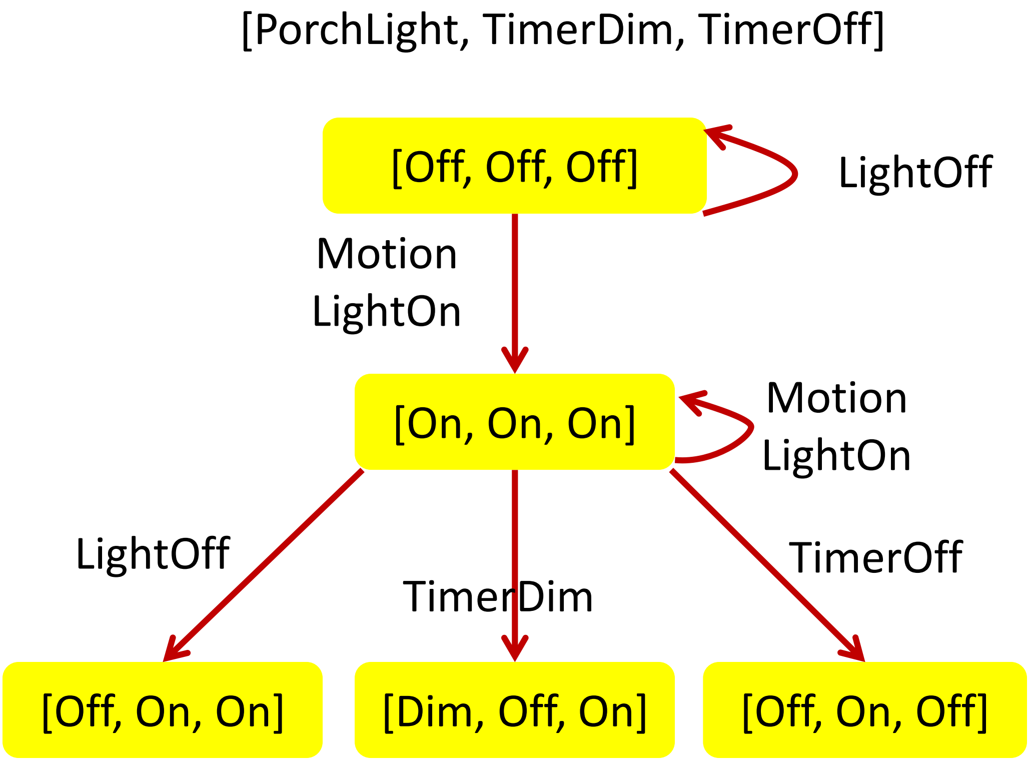
Example

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

porchLight.On:
  timer.Start(5 mins)

timer.Fired:
  porchLight.Set(Off)
```



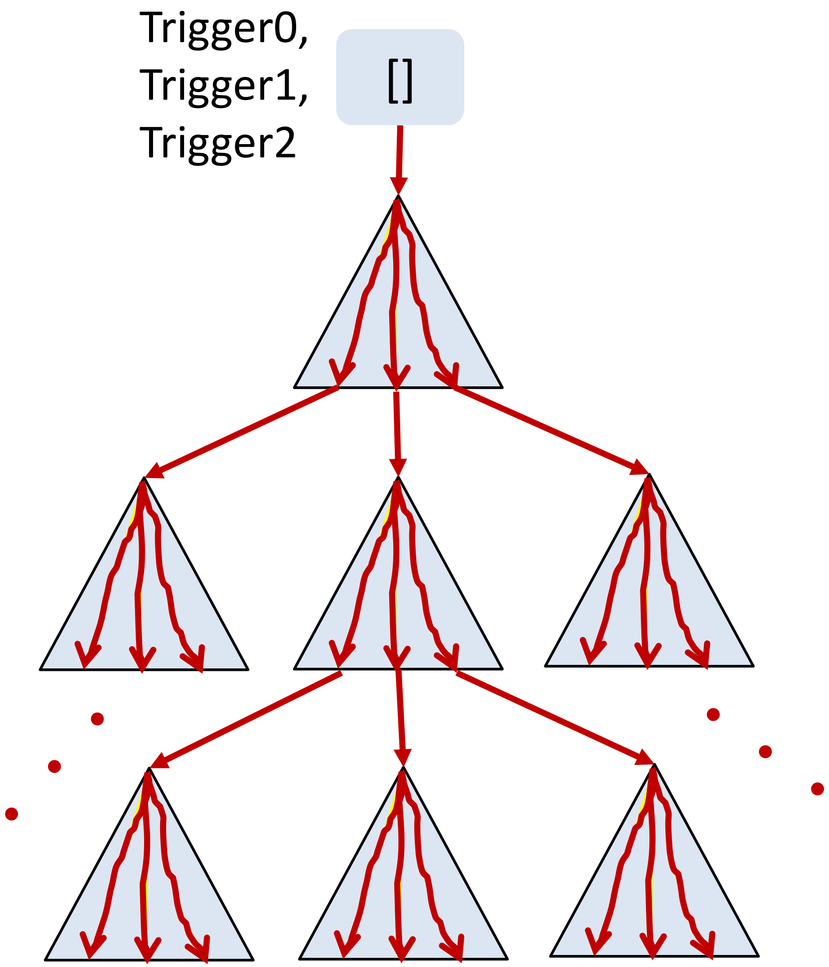
Exploring temporal behavior: soundness

```
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();
```

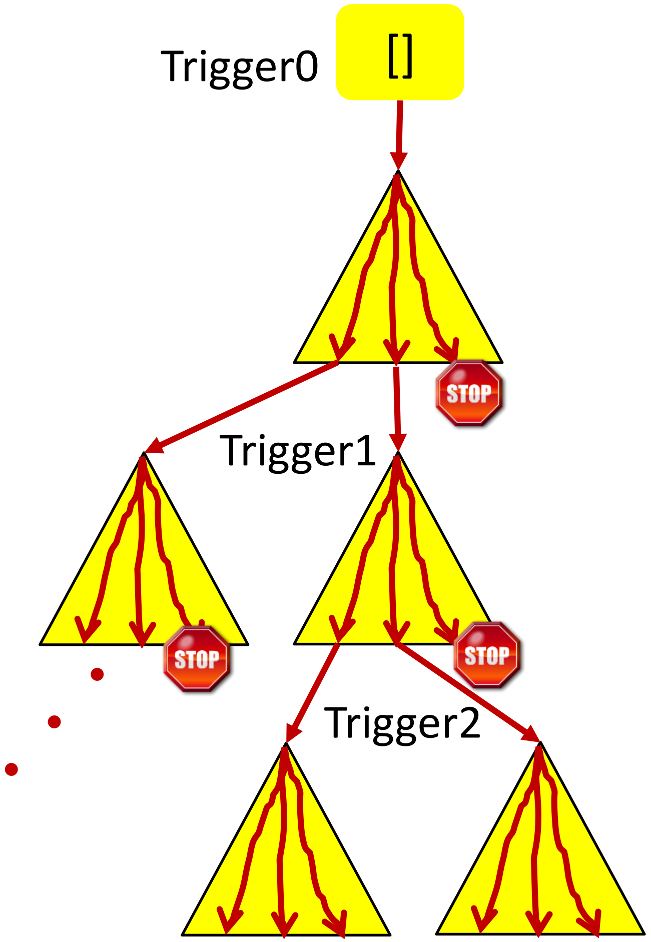


Use symbolic execution alone?

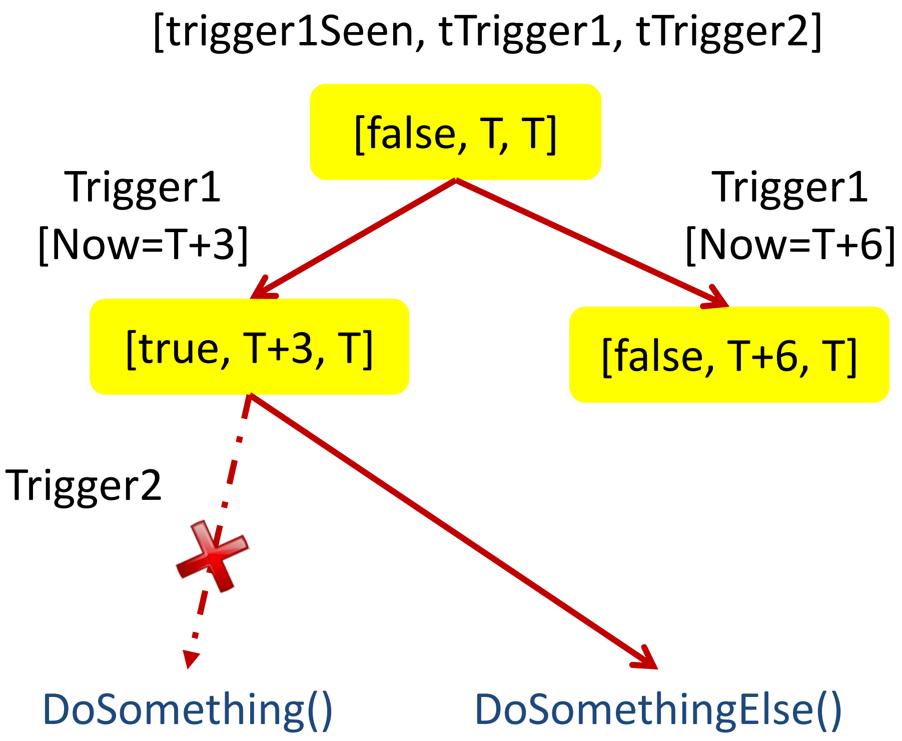
Symbolic, path-based



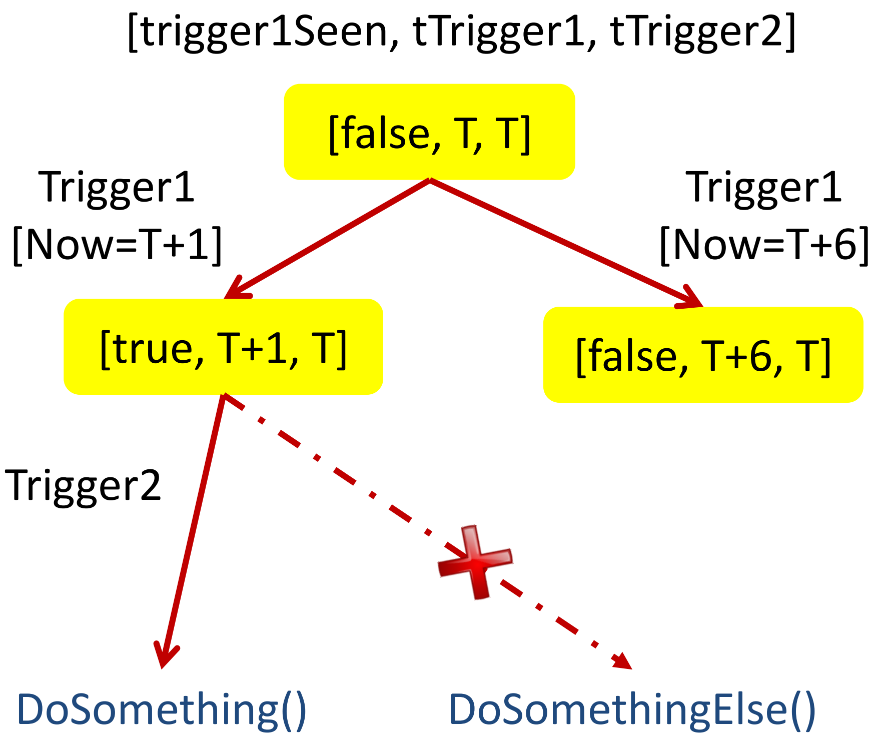
Concrete, state-based



```
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()
```



[trigger1Seen, tTrigger1, tTrigger2]

[false, T, T]

Trigger1
[Now=T+1]

Trigger1
[Now=T+6]

[true, T+1, T]

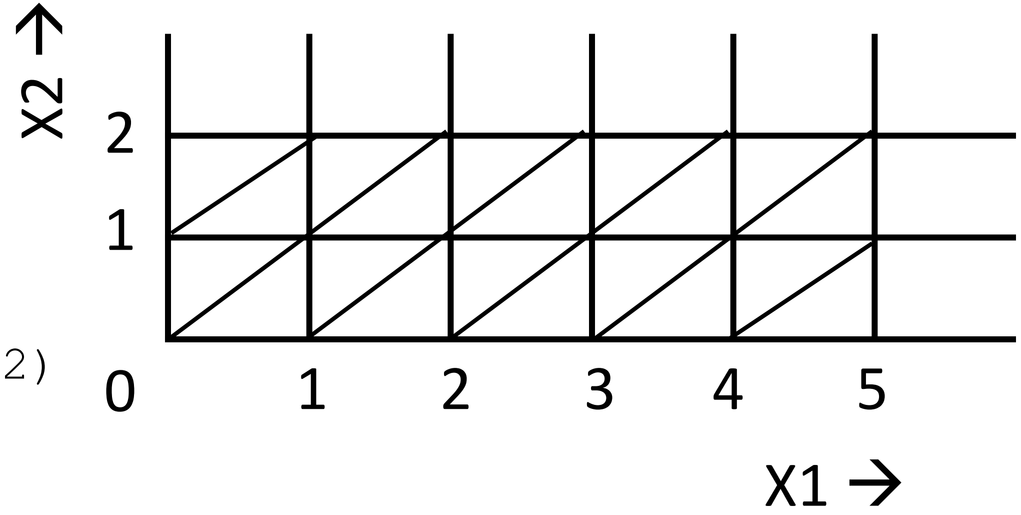
[false, T+6, T]

Trigger2

DoSomething()

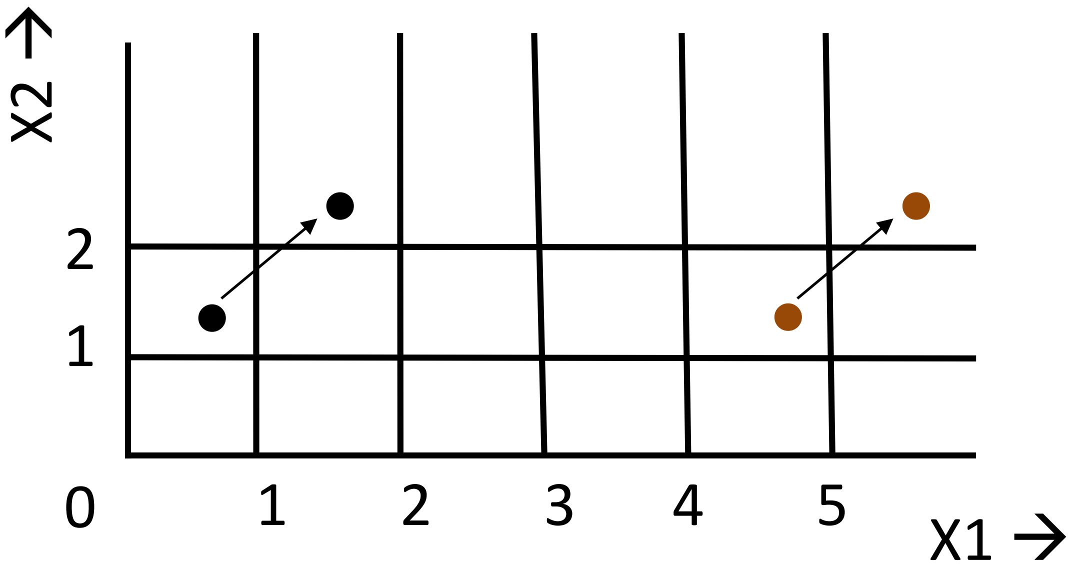
DoSomethingElse()

```
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()
```



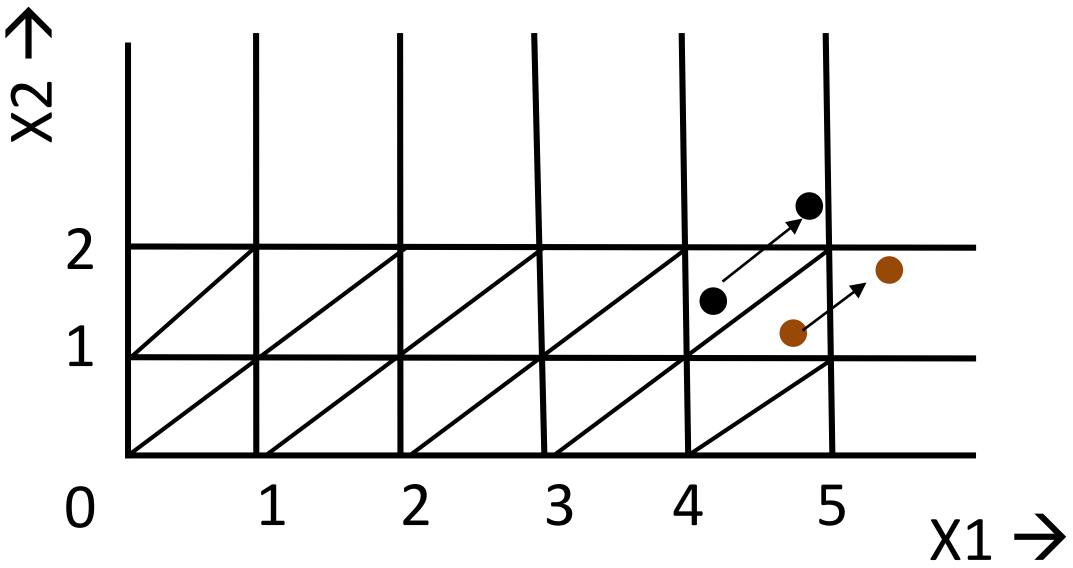
Why this construction works

1. X1 < 5
2. X2 < 2
3. X1 < 5 && X2 > 2

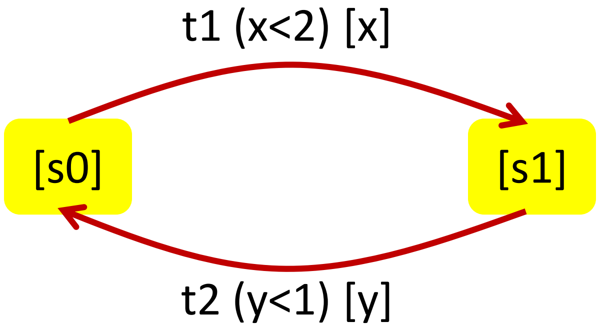


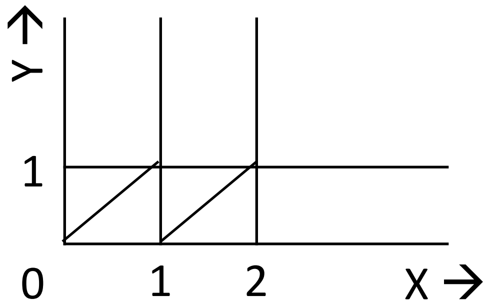
Why this construction works

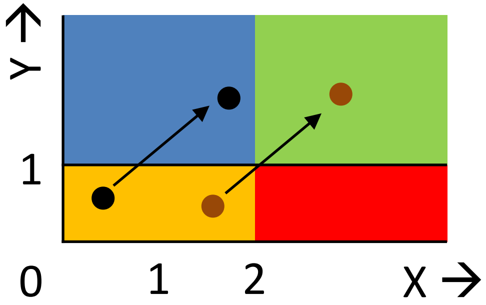
1. X1 < 5
2. X2 < 2
3. X1 < 5 && X2 > 2



Why regions are fine-grained

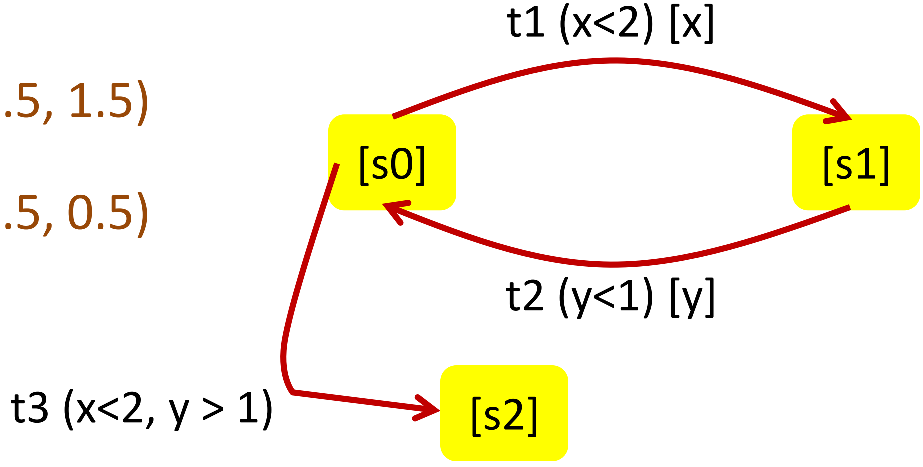






● (1.5, 1.5) ● (2.5, 1.5)

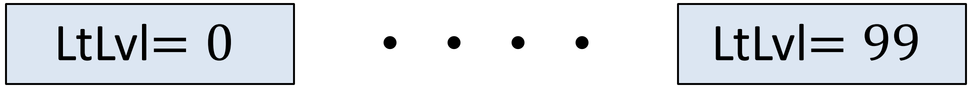
● (0.5, 0.5) ● (1.5, 0.5)



Finding equivalent environments

1. Symbolically execute each trigger
2. Find environmental conditions that lead to same *state*

```
motionPorch:
   x = lightMeter.Level
```



LtLvl= 0 . . . . LtLvl= 99

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

timer.Fired:
  porchLight.Set(Off)
```

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 (4/4): Sleep calls

```
Trigger:
    ... //pre-sleep actions
    Sleep(10)
    ... //post-sleep actions
```

```
Trigger:
    ...    //pre-sleep actions
    VC_sleeper = 0

VC_sleeper == 10:
    ...   //post-sleep actions
```

Reducing the number of VCs: Combining 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
```

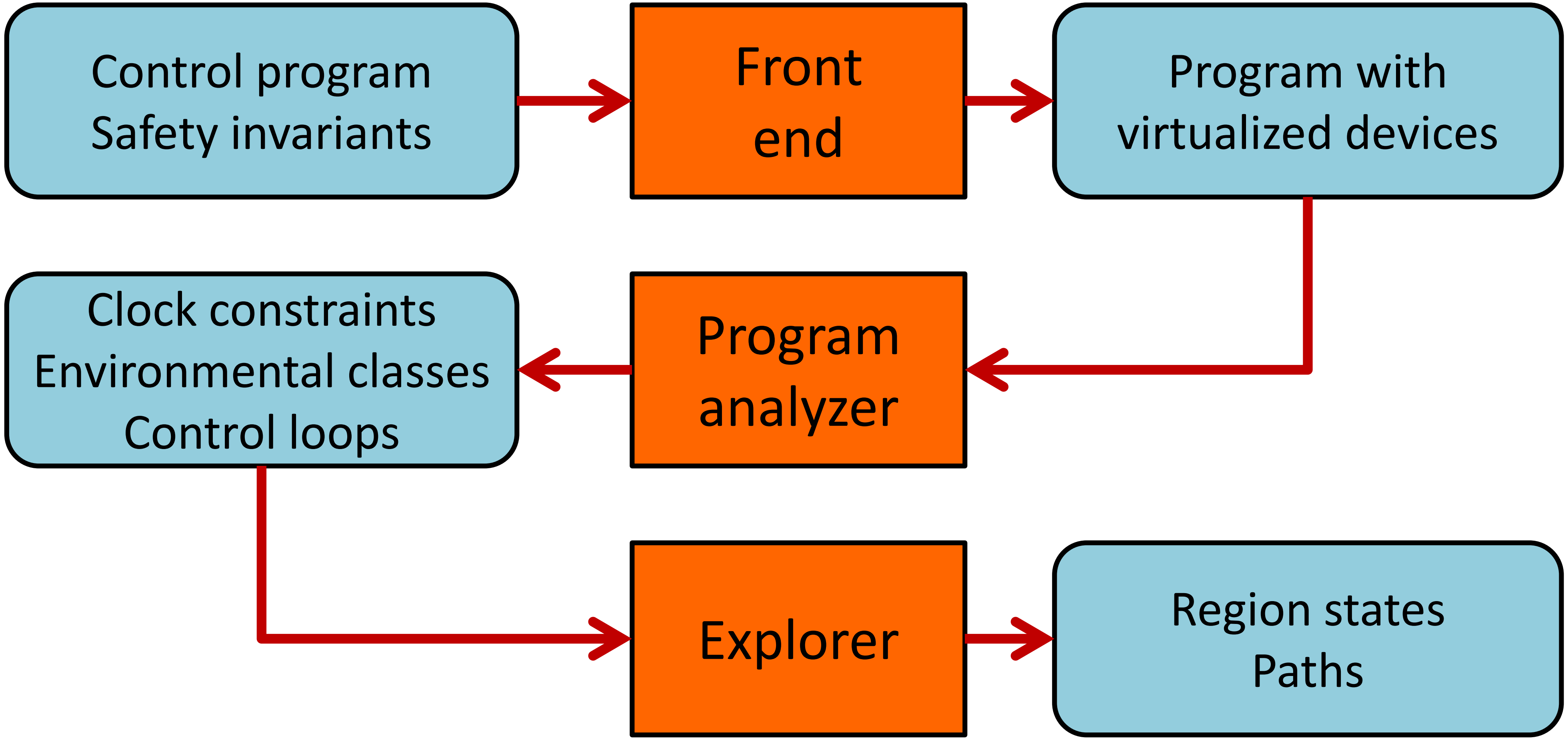
Constructing time regions

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()
```

DeLorean



Control program
Safety invariants

Front end

Program with virtualized devices

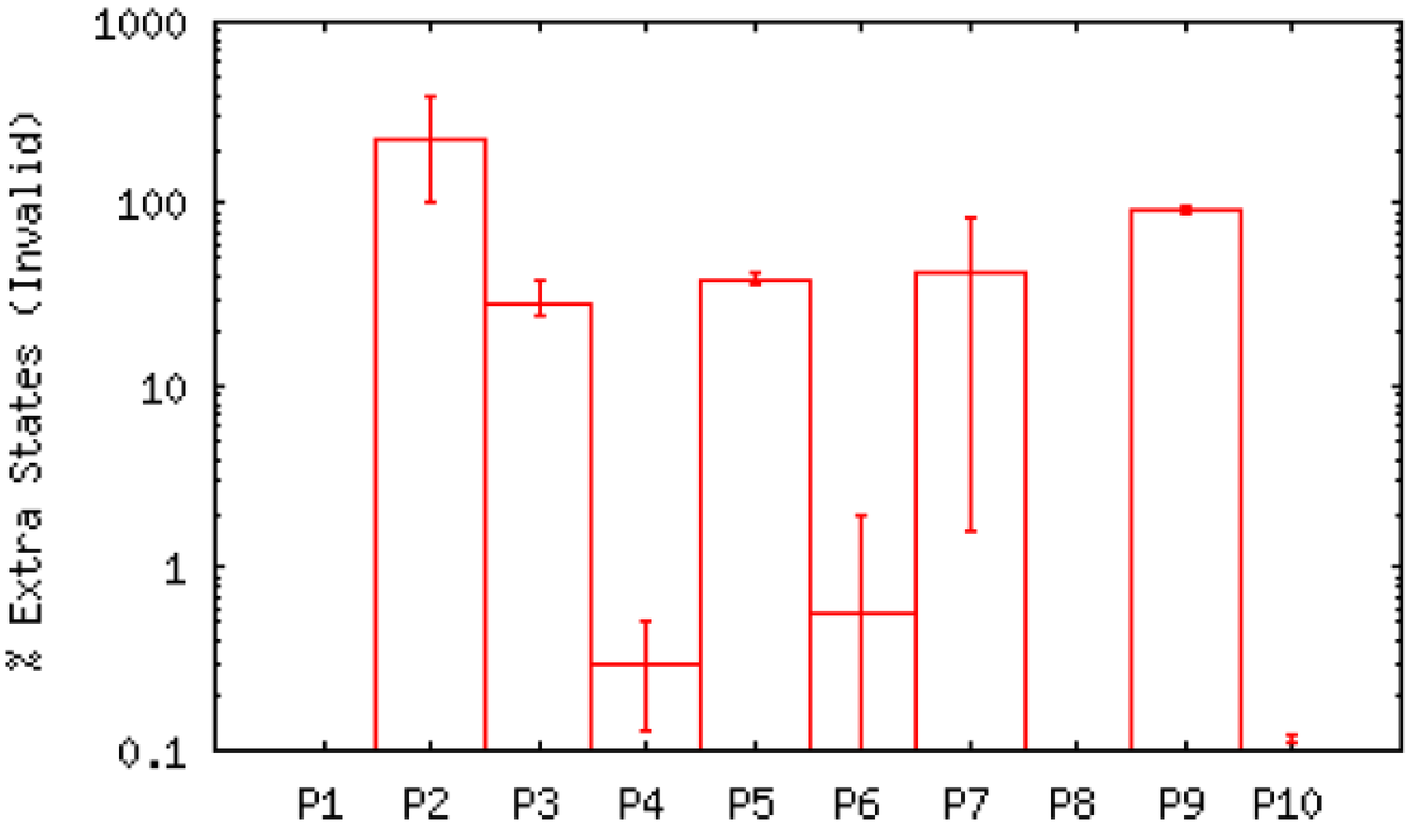
Program analyzer

Clock constraints
Environmental classes
Control loops

Explorer

Region states
Paths

Comparison with untimed model checking



Untimed model checking reaches many invalid states

Reducing the number of VCs: Combining sleep calls

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

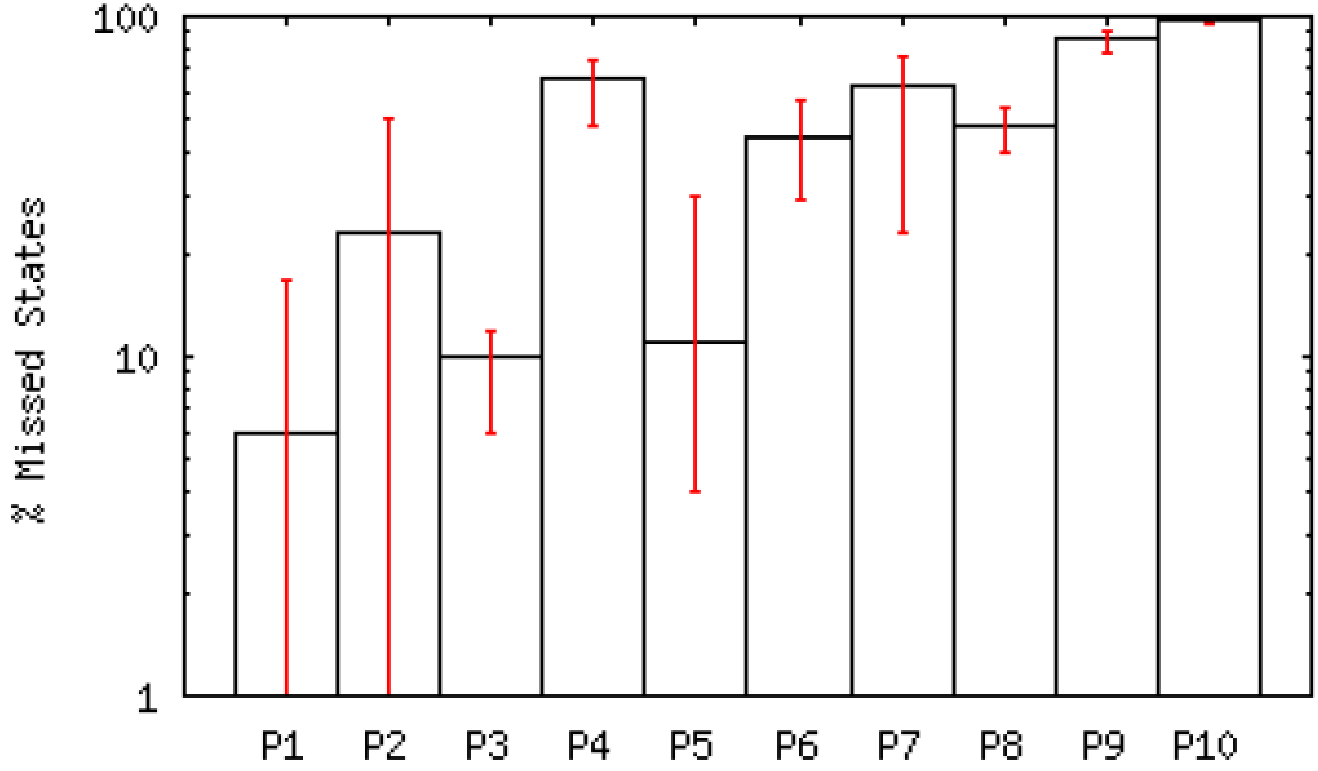
```
Trigger:
   Act1()
   VC_sleeper = 0
   sleep_counter = 1;

VC_sleeper == 5:
   Act2()

VC_sleeper == 15:
   Act3()
```

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

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