

Alexander Schulz Rosengarten, *Reinhard von Hanxleden*
Kiel University

Frédéric Mallet, Robert de Simone, Julien DeAntoni
INRIA Sophia Antipolis

Lecture 15: Time in SCCcharts

Alexander Schulz Rosengarten, *Reinhard von Hanxleden*
Kiel University

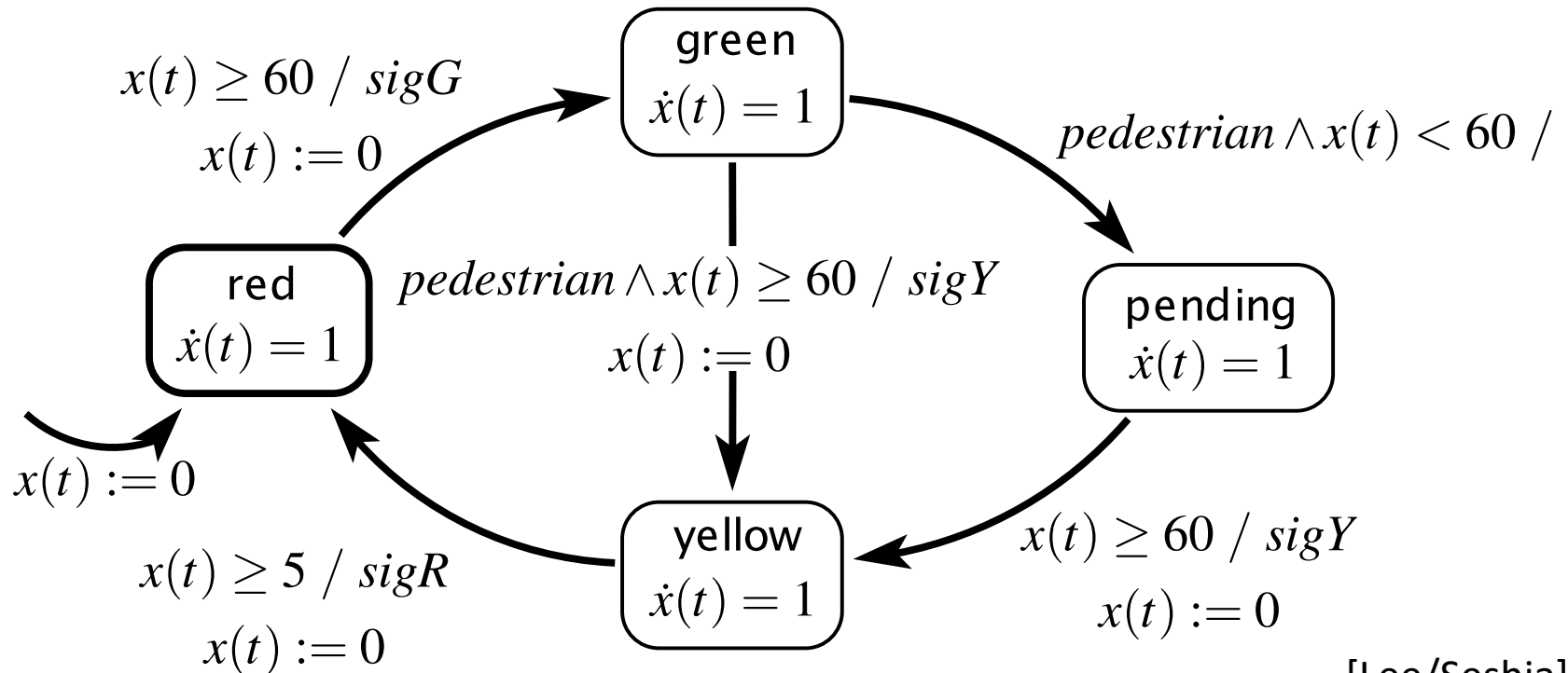
Frédéric Mallet, Robert de Simone, Julien DeAntoni
INRIA Sophia Antipolis

Traffic Light as Timed Automaton

continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

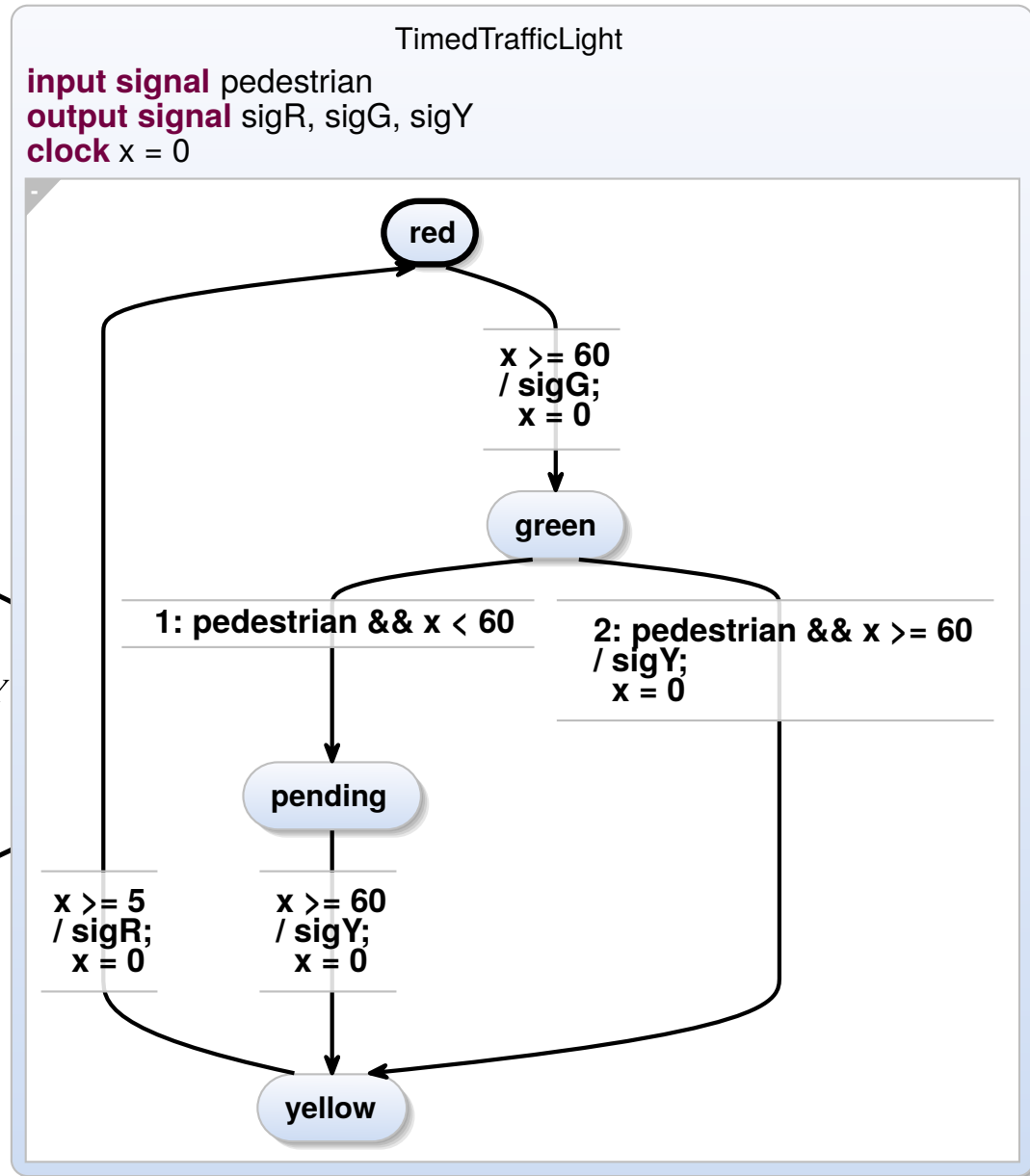
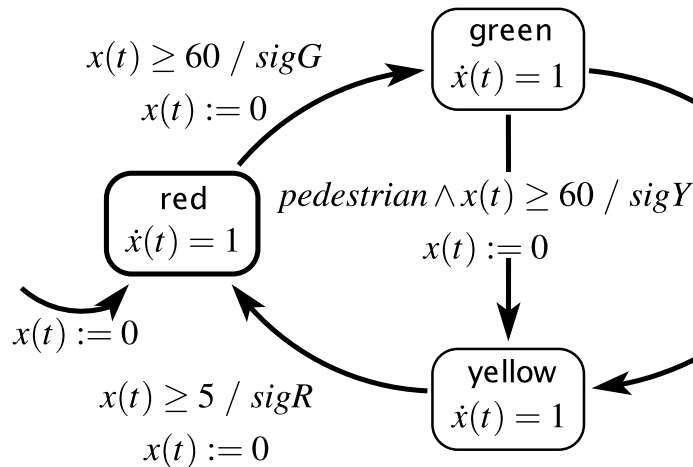
outputs: *sigR*, *sigG*, *sigY*: pure



[Lee/Seshia]

Traffic Light in SCCharts

continuous variable: $x(t) : \mathbb{R}$
inputs: *pedestrian*: pure
outputs: *sigR, sigG, sigY*: pure



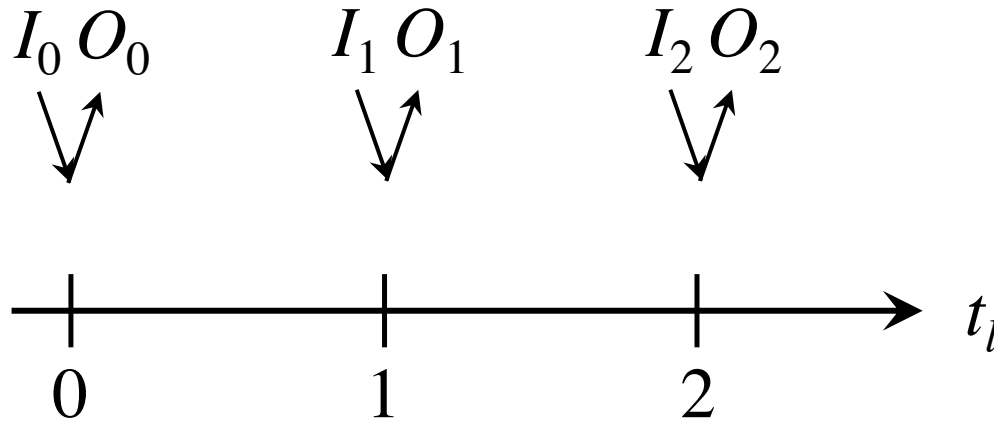
Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

Roadmap

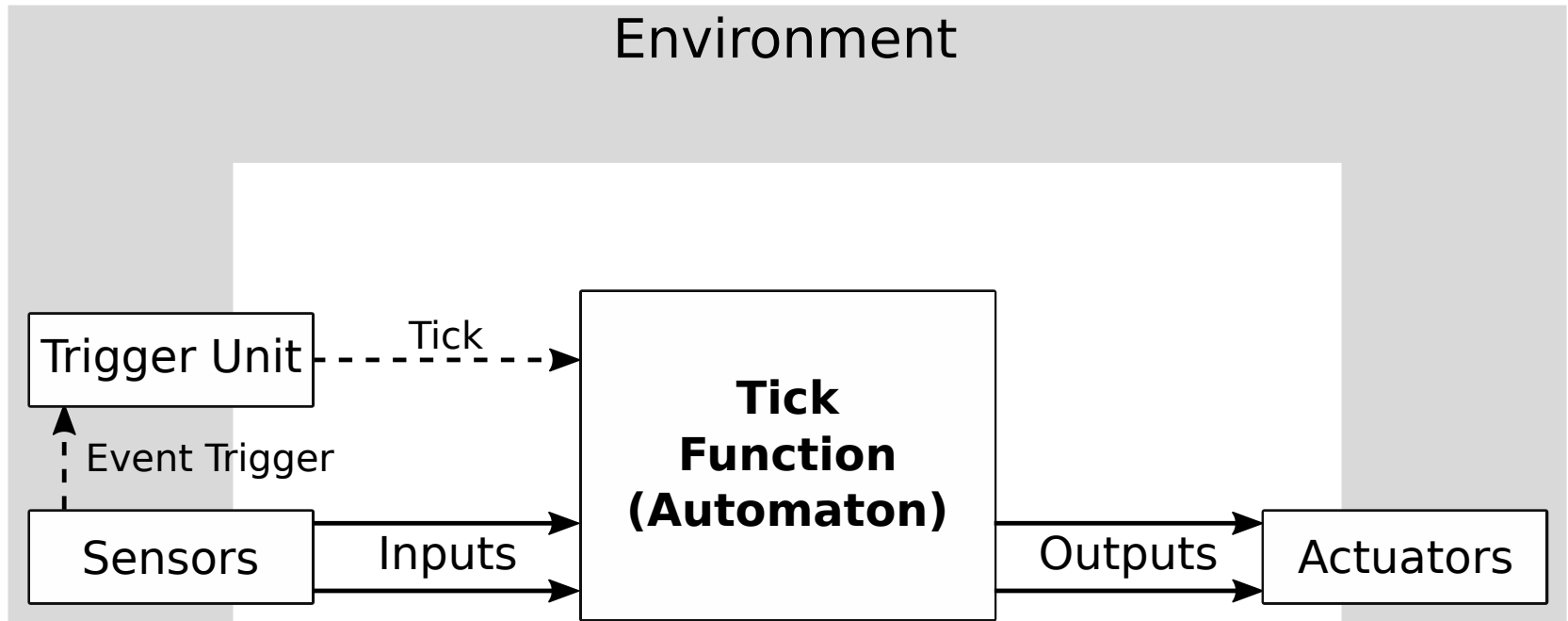
1. Traffic Light Example
- 2. Execution Models**
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo

Discrete (Logical) Time in Synchronous Programming



- Synchrony Hypothesis:
Outputs are synchronous with inputs
- Computation "does not take time"
- Actual computation time does not influence result
- Sequence of outputs **determined** by inputs

Event-Triggered Execution

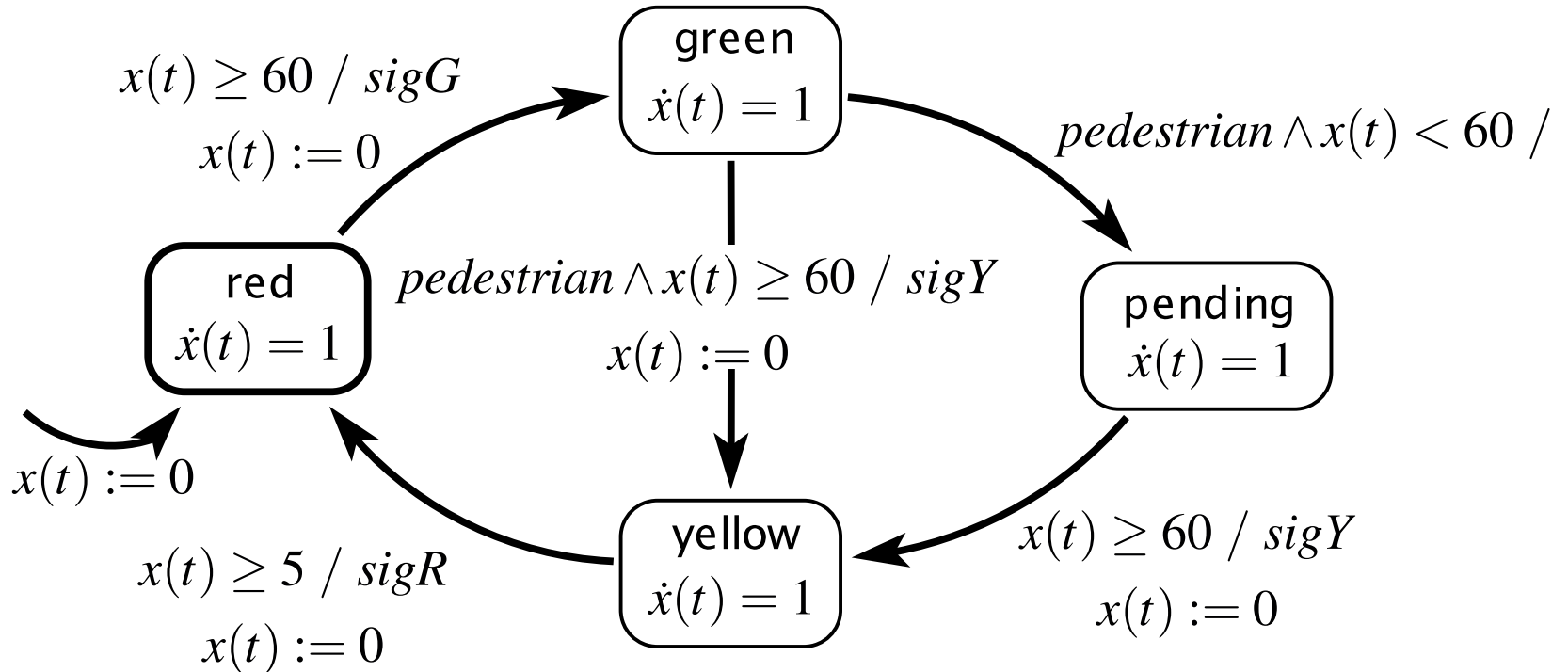


continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

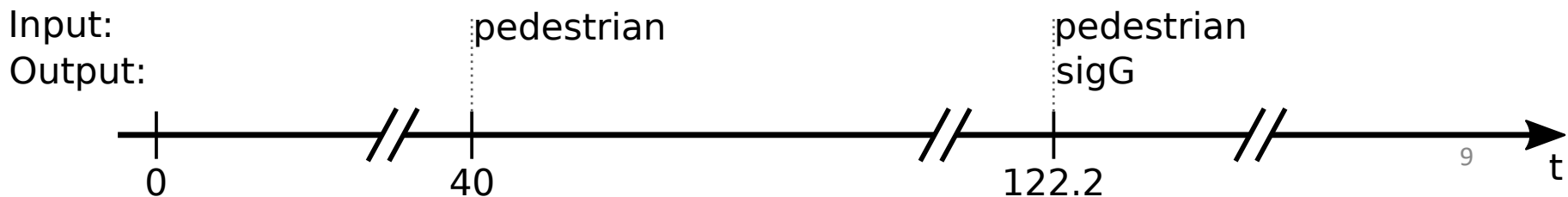
outputs: *sigR*, *sigG*, *sigY*: pure

Assume pedestrian button
pressed at $t = 40$ and $t = 122.2$



[Lee/Seshia]

Event-Triggered Execution, with initial tick at $t = 0$:



Synchronous Execution

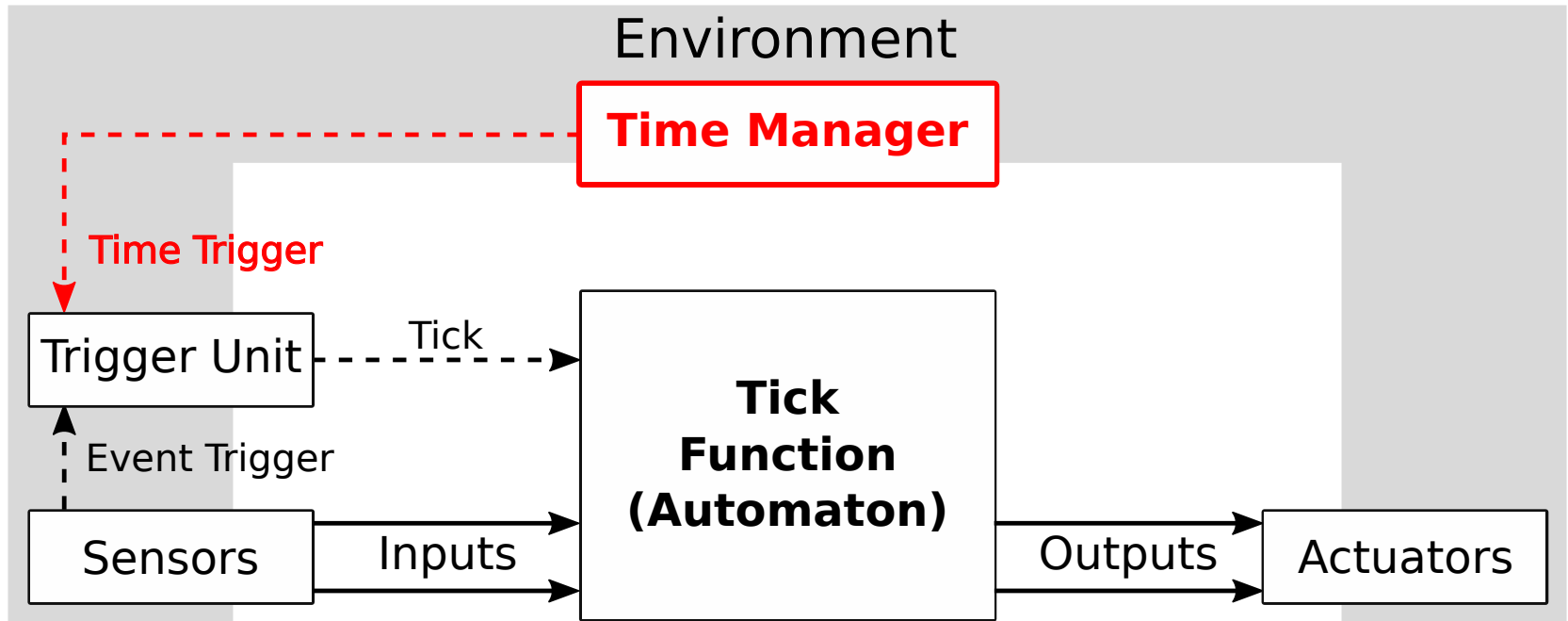
```
Initialize Memory
for each input event do
    Compute Outputs
    Update Memory
end
```

```
Initialize Memory
for each clock tick do
    Read Inputs
    Compute Outputs
    Update Memory
end
```

Fig. 1 Two common synchronous execution schemes: event driven (left) and sample driven (right).

[Benveniste et al., *The Synchronous Languages Twelve Years Later*, Proc. IEEE, 2003]

Time-Triggered Execution

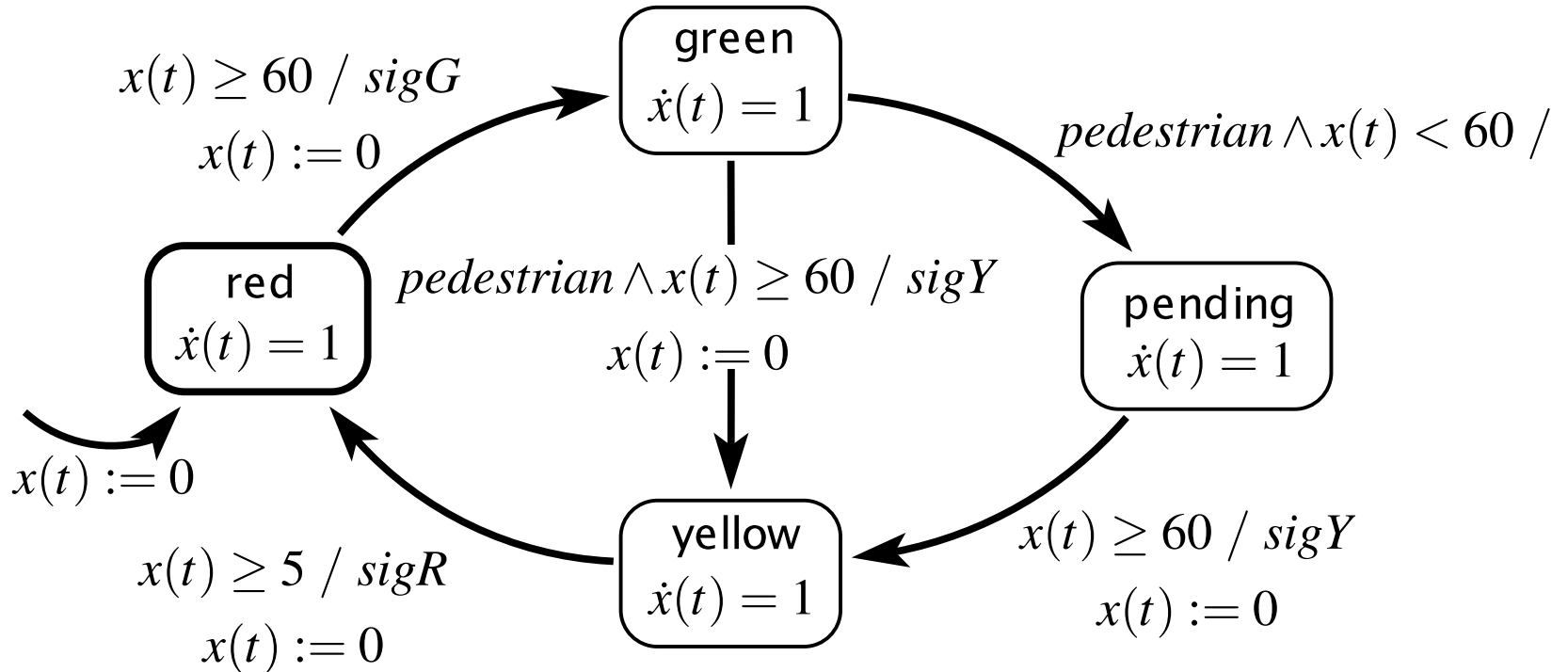


continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

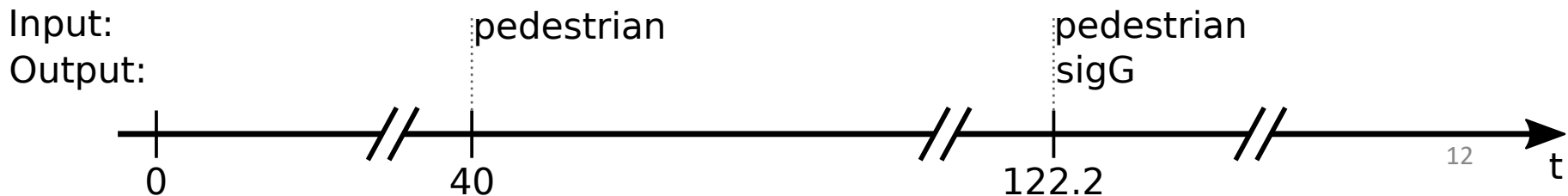
outputs: *sigR*, *sigG*, *sigY*: pure

Assume pedestrian button
pressed at $t = 40$ and $t = 122.2$



[Lee/Seshia]

Recall: Event-Triggered Execution:

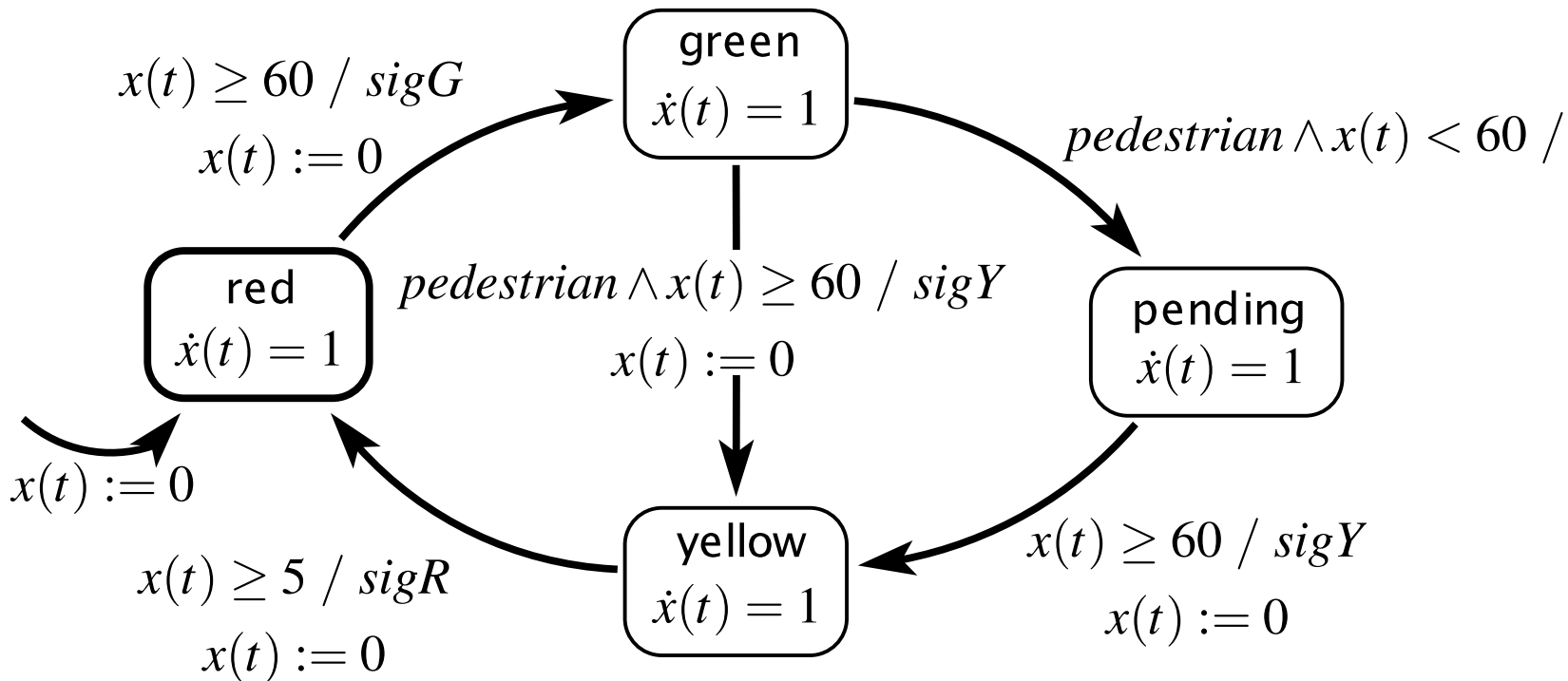


continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

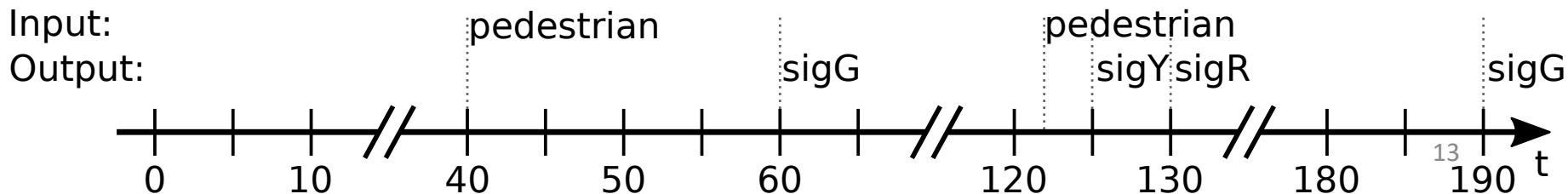
outputs: *sigR*, *sigG*, *sigY*: pure

Assume pedestrian button
pressed at $t = 40$ and $t = 122.2$



[Lee/Seshia]

Time-Triggered Execution (every 5 sec):



Multiform Notion of Time

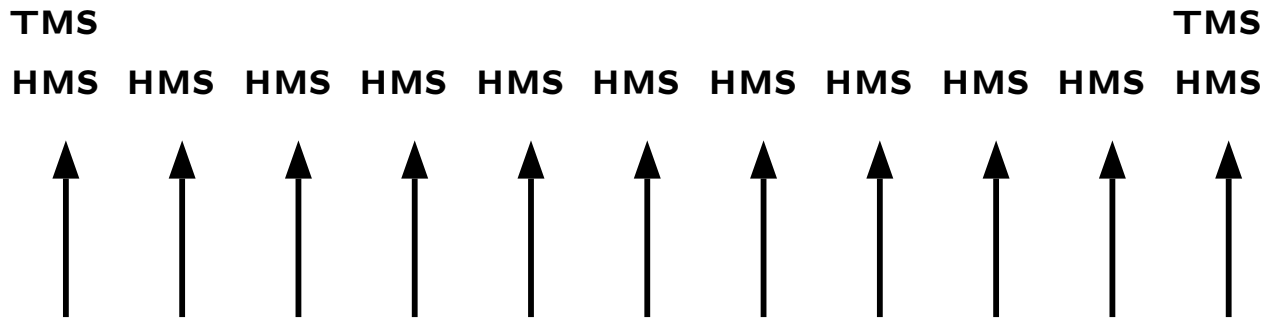
Only the simultaneity and precedence of events are considered.

This means that the physical time does not play any special role.

This is called multiform notion of time.

[<https://en.wikipedia.org/wiki/Esterel>]

Packaging Physical Time as Events



[Timothy Bourke, SYNCHRON 2009]

Event "HMS": 100 μ sec have passed since last HMS

Event "TMS": 1000 μ sec have passed since last TMS

A Problem With That ...

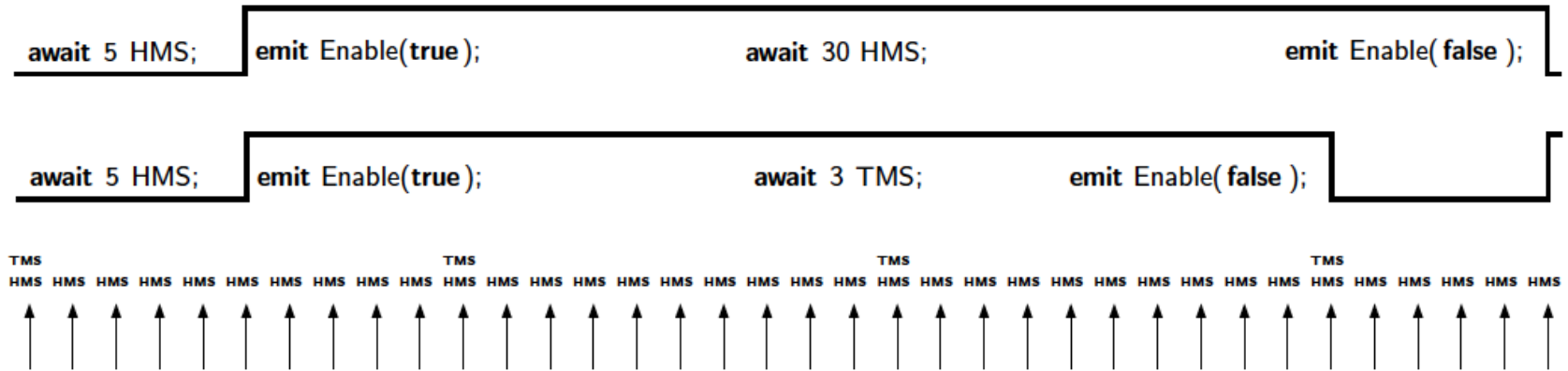


Fig. 4: Granularity of timing inputs

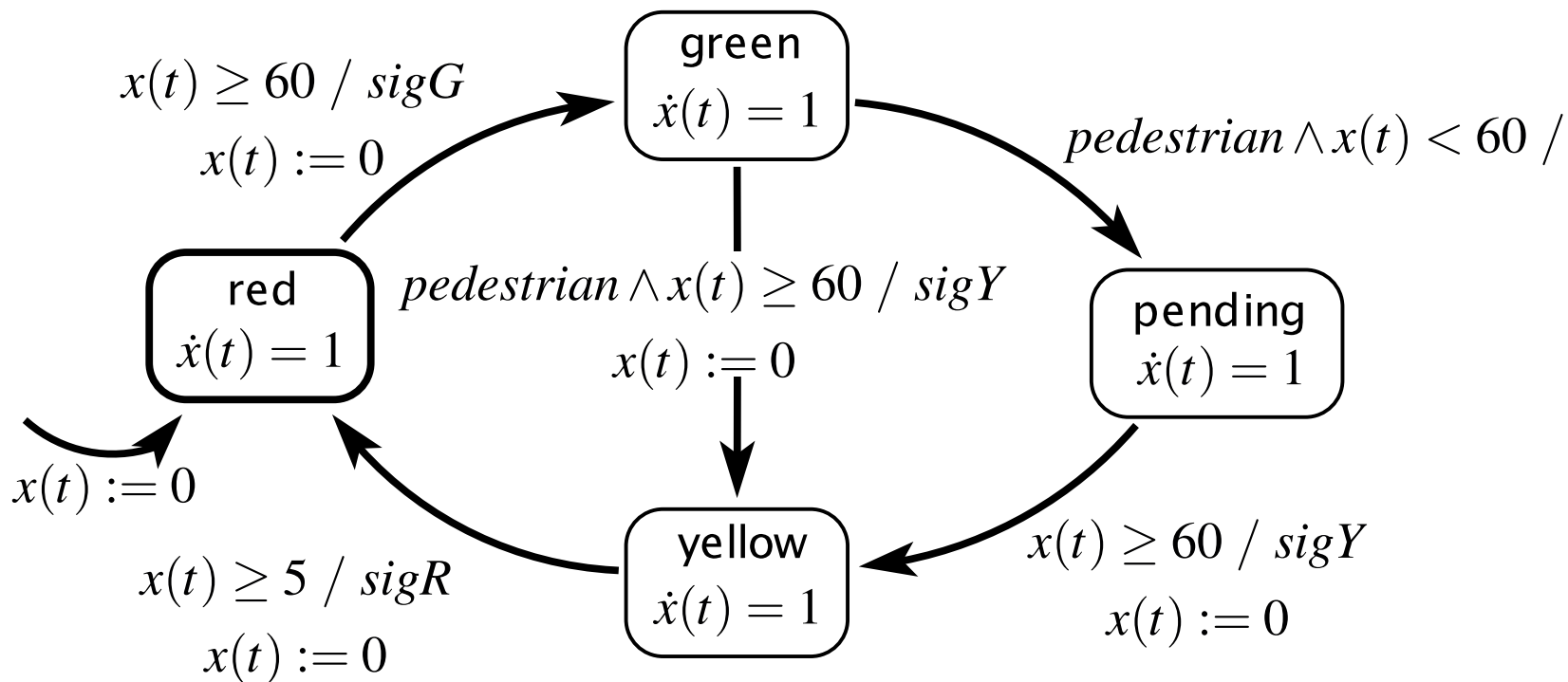
[Timothy Bourke, SYNCHRON 2009]

continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

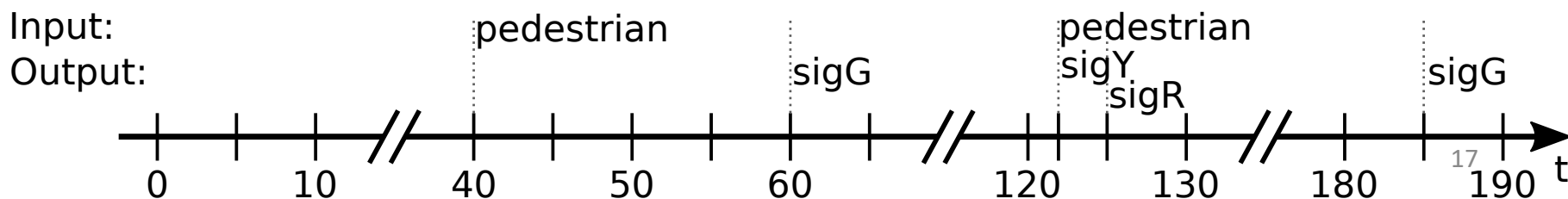
outputs: *sigR*, *sigG*, *sigY*: pure

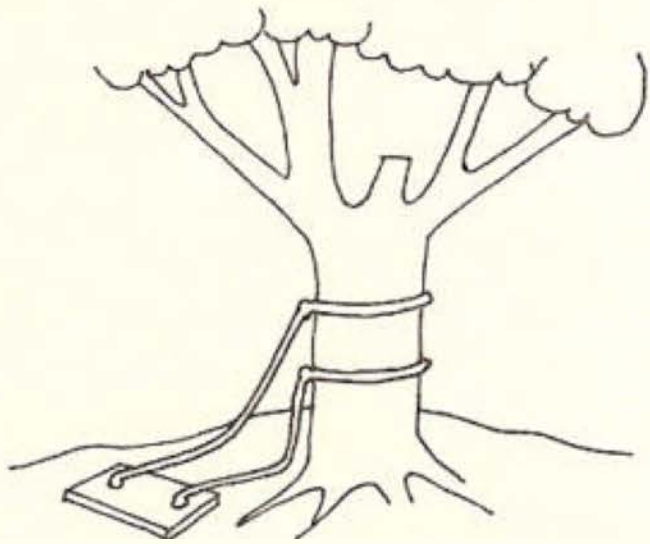
Assume pedestrian button
pressed at $t = 40$ and $t = 122.2$



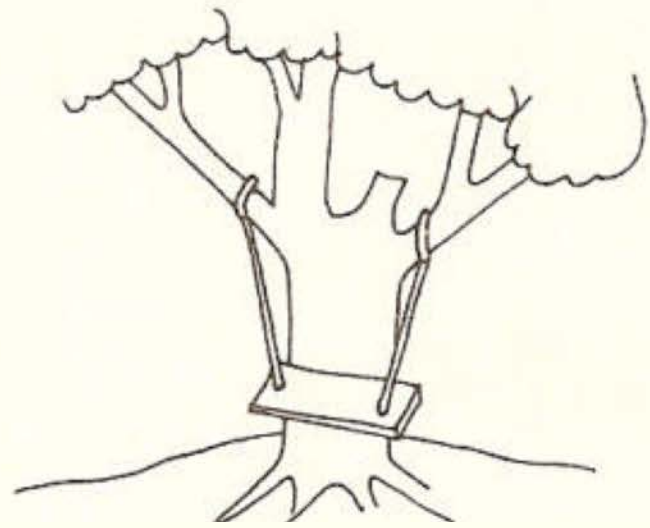
[Lee/Seshia]

Time-Event-Triggered Execution, Multiform Time:

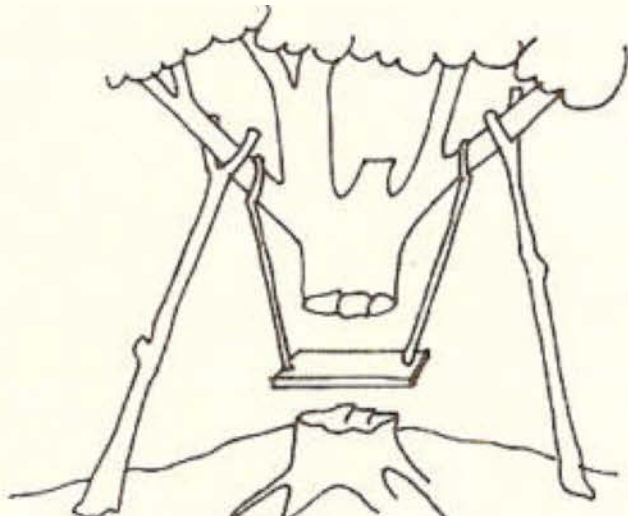




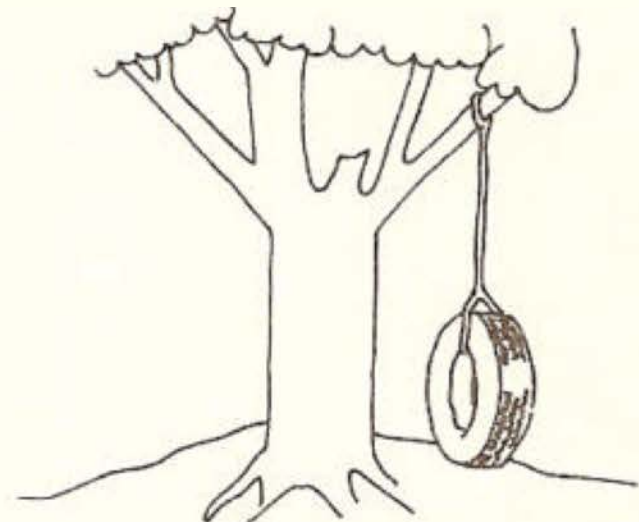
Event-Triggered



Time-Triggered



Time-Event-Triggered



Eager

ACKNOWLEDGEMENTS TO UNKNOWN AUTHOR

What the User (Probably) Wanted

„We assume here that a transition is taken as soon as it is enabled. Other transition semantics are possible.“

[Lee/Seshia 2017]

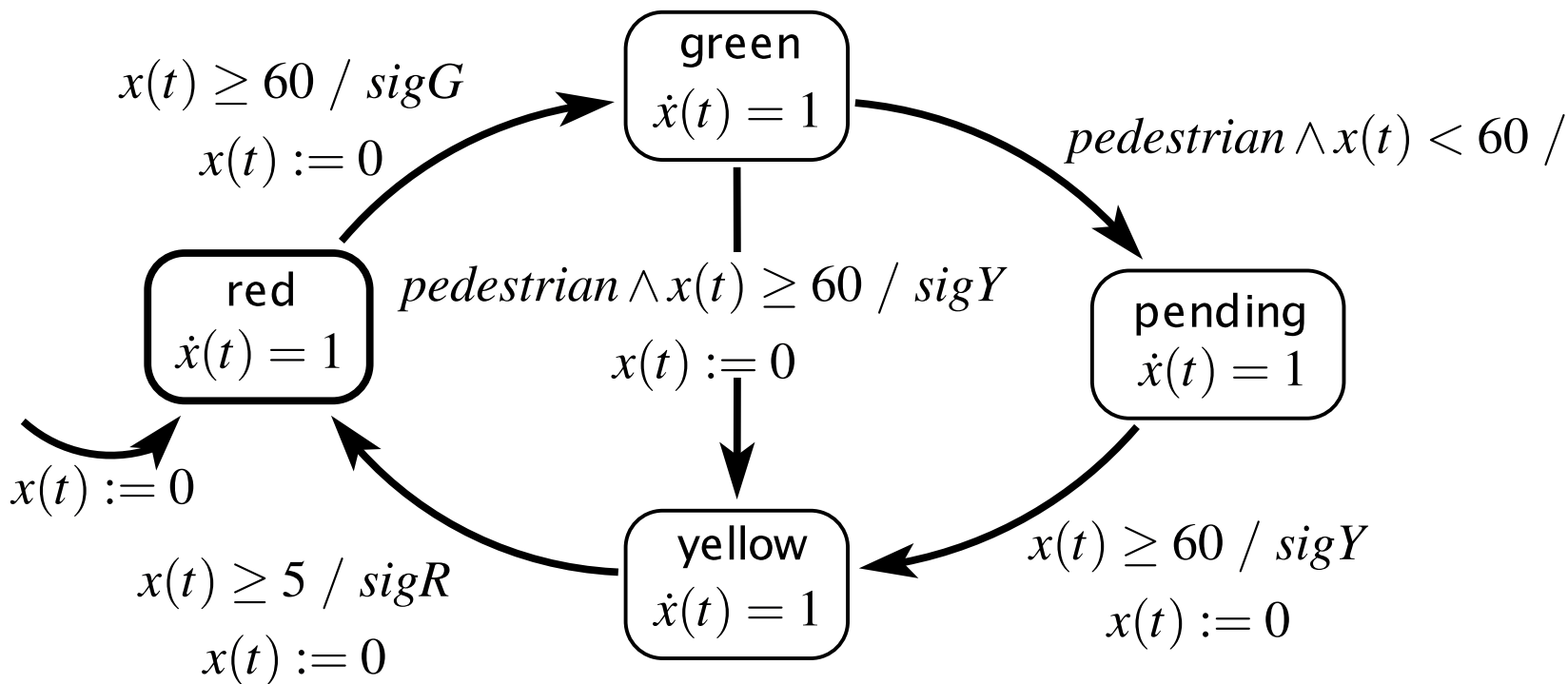
We call this **eager** semantics.

continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

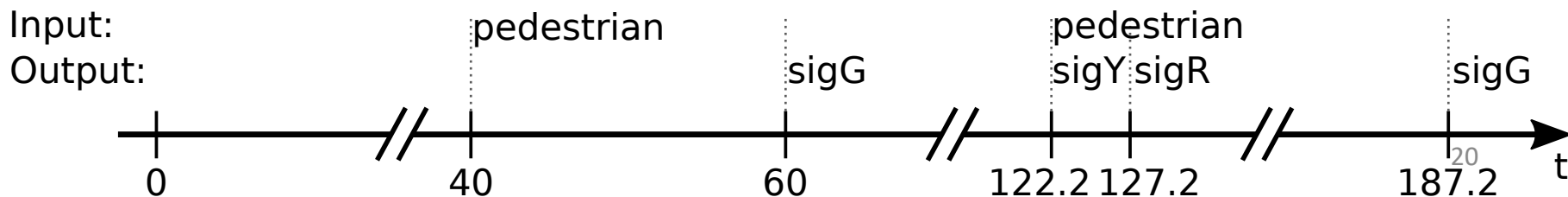
outputs: *sigR*, *sigG*, *sigY*: pure

Assume pedestrian button pressed at $t = 40$ and $t = 122.2$



[Lee/Seshia]

Eager Semantics:



Time in SCCharts – Requirements

1. **Seamless fit into synchronous paradigm**
 - Still deterministic behavior – outputs fully determined by inputs
 - No changes to underlying SC (Sequentially Constructive) MoC
2. **Approximate eager semantics**
 - Modulo run-time variations and imperfections of physical timers
3. **Scalability**
 - E.g., allow arbitrary number of (concurrent) timers
4. **Fine granularity**
 - Gcd may be arbitrarily small, w/o performance penalty
 - E.g., may have timeouts of 1 sec and 3.1415926 msec in same model
5. **Time composability**
 - E.g., waiting 1 sec. twice should mean the same as waiting 2 sec's once

Time in SCCharts – Requirements

6. Preserve temporal order and simultaneity
 - E.g., timers started in same tick and running same duration should expire in same tick
7. Minimize impact of physical timer variations
 - E.g., avoid accumulations of timer imperfections
8. Give application access to physical time and tick computation time
 - Facilitates e.g. load-dependent execution modes
9. Lean, application-independent interface to environment
 - E.g., interface should not change if number of timers changes
10. Fit into Single Language-Driven Incremental Compilation (SLIC) concept
 - New timing constructs are just syntactic sugar on top of existing SCCharts
 - Transforming away timing constructs requires only local changes
 - No changes needed to compilation back-end

Roadmap

1. Traffic Light Example
2. Execution Models
- 3. Dynamic Ticks**
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo



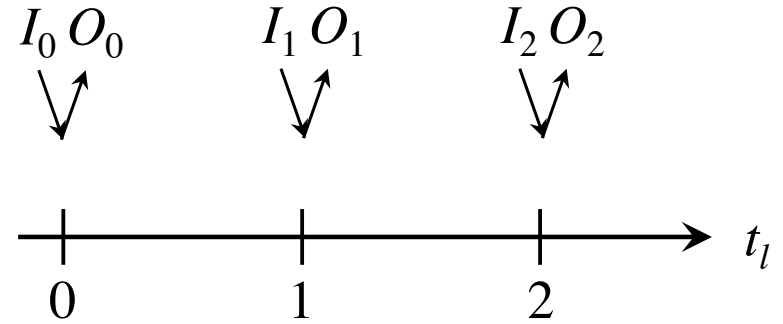
Real-Time Ticks for Synchronous Programming

Reinhard von Hanxleden (U Kiel)
Timothy Bourke (INRIA and ENS, Paris)
Alain Girault (INRIA and U Grenoble)

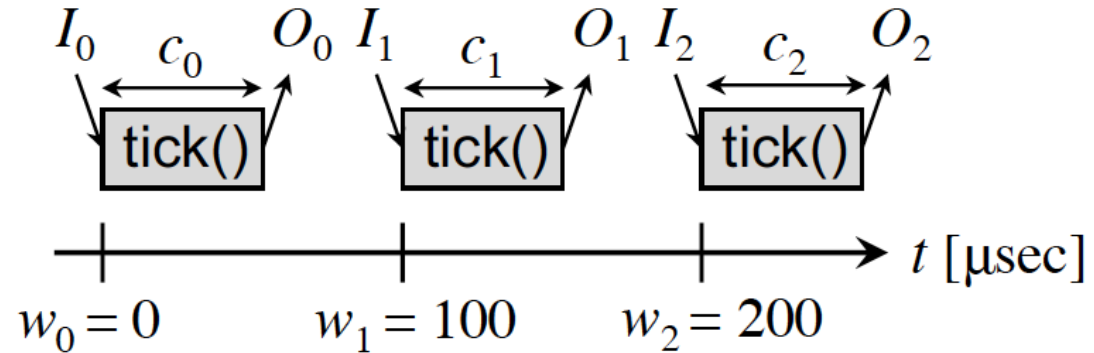
19 Sep 2017, FDL '17, Verona

Dynamic Ticks

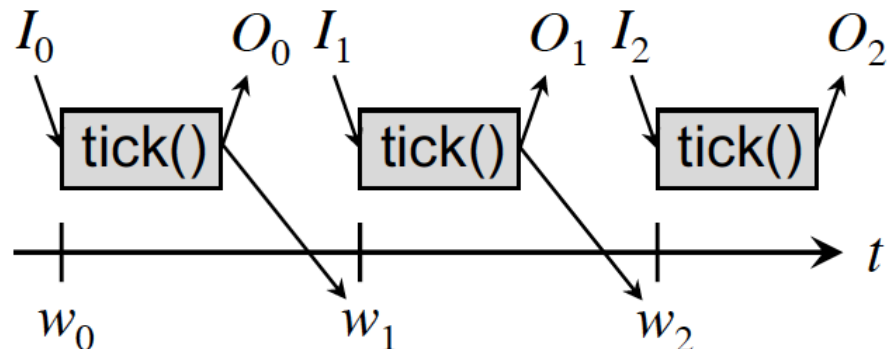
- Recall logical time:



- Physical time, time-triggered:



- Physical time, dynamic ticks:



```
module PAUSE_USEC:
```

```
input current_usec : integer;
```

```
% Simulated time
```

```
input wait_usec : integer;
```

```
% Time of delay
```

```
function min(integer, integer) : integer;
```

```
output wake_usec : combine integer with min;
```

```
% Time of next wake up
```

```
var my_wake_usec : integer in
```

```
% Local copy of wake_usec
```

```
% Compute physical time when PAUSE_USEC should terminate
```

```
my_wake_usec := ?current_usec + ?wait_usec;
```

```
% Loop until current_usec = my_wake_usec
```

```
trap done in
```

```
loop
```

```
  emit wake_usec(my_wake_usec);
```

```
  pause;
```

```
  if ?current_usec = my_wake_usec then
```

```
    exit done;
```

```
  end if;
```

```
end loop
```

```
end trap
```

```
end var
```

```
end module
```

```
int main()
{
    int notDone, prev_tick_end_usec = 0;

    RACE_reset(); // Reset automaton
    time_reset(); // Initialize time

    // Loop until tick function terminates
    do {
        // Set inputs
        RACE_I_current_wall_usec(get_current_wall_usec());
        RACE_I_prev_tick_end_usec(prev_tick_end_usec);

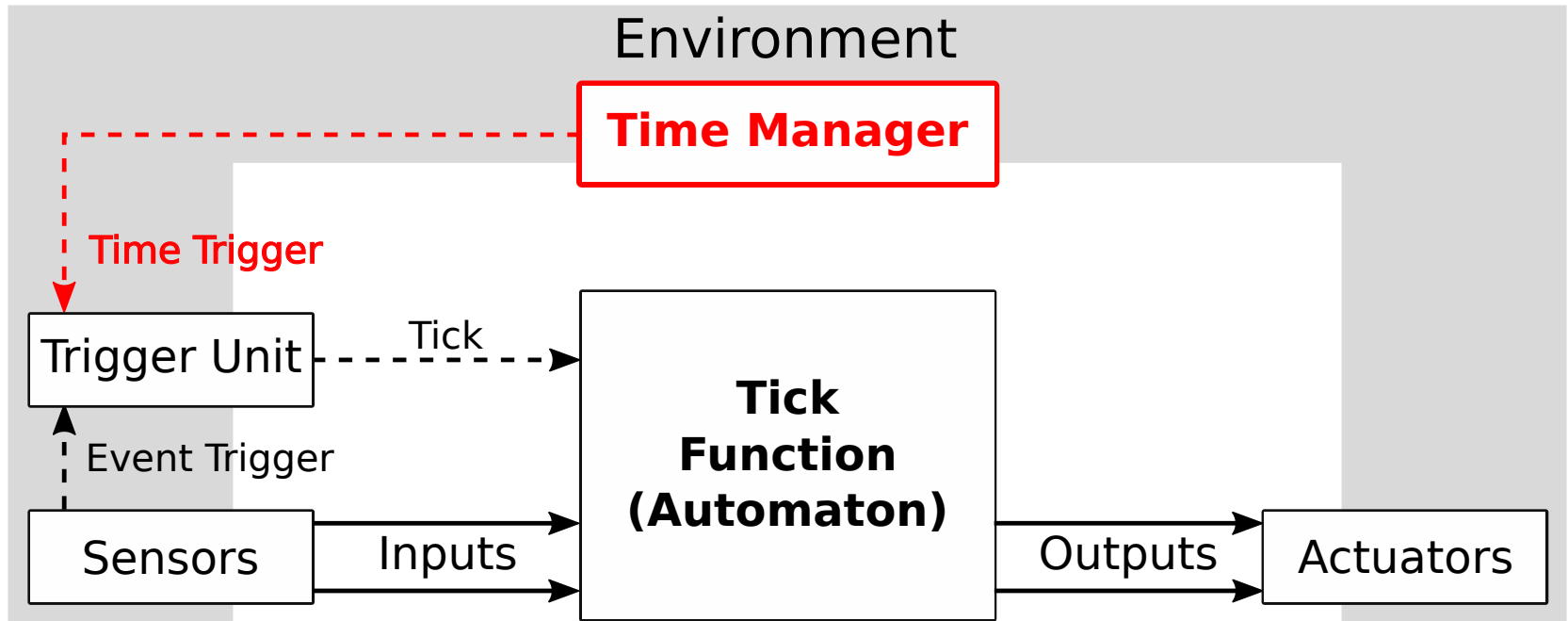
        notDone = RACE(); // Call tick function
        prev_tick_end_usec = get_current_wall_usec();

        // Wait until wake_usec
        microsleep(wake_usec - prev_tick_end_usec);
    } while (notDone);

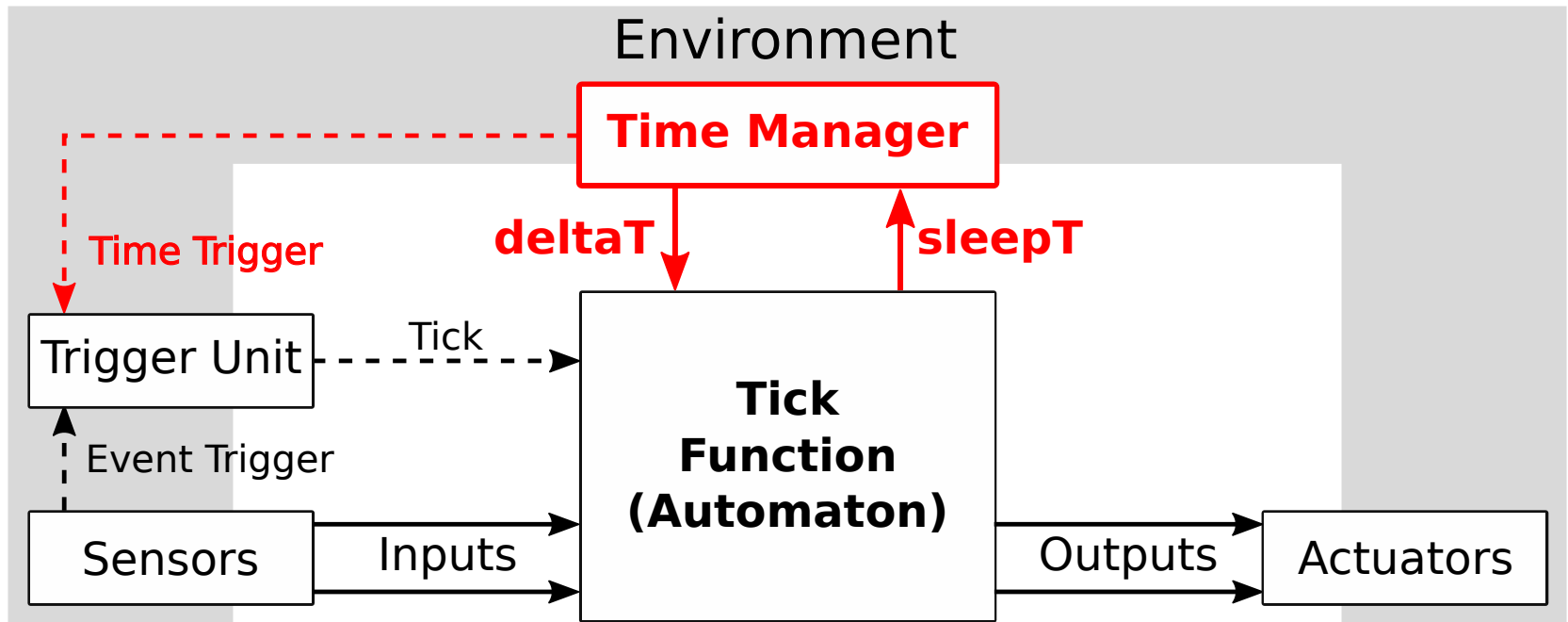
    return 0;
}
```

Host Code

Recall: Time-Triggered Execution



Eager Execution with Dynamic Ticks



deltaT: Time since last tick

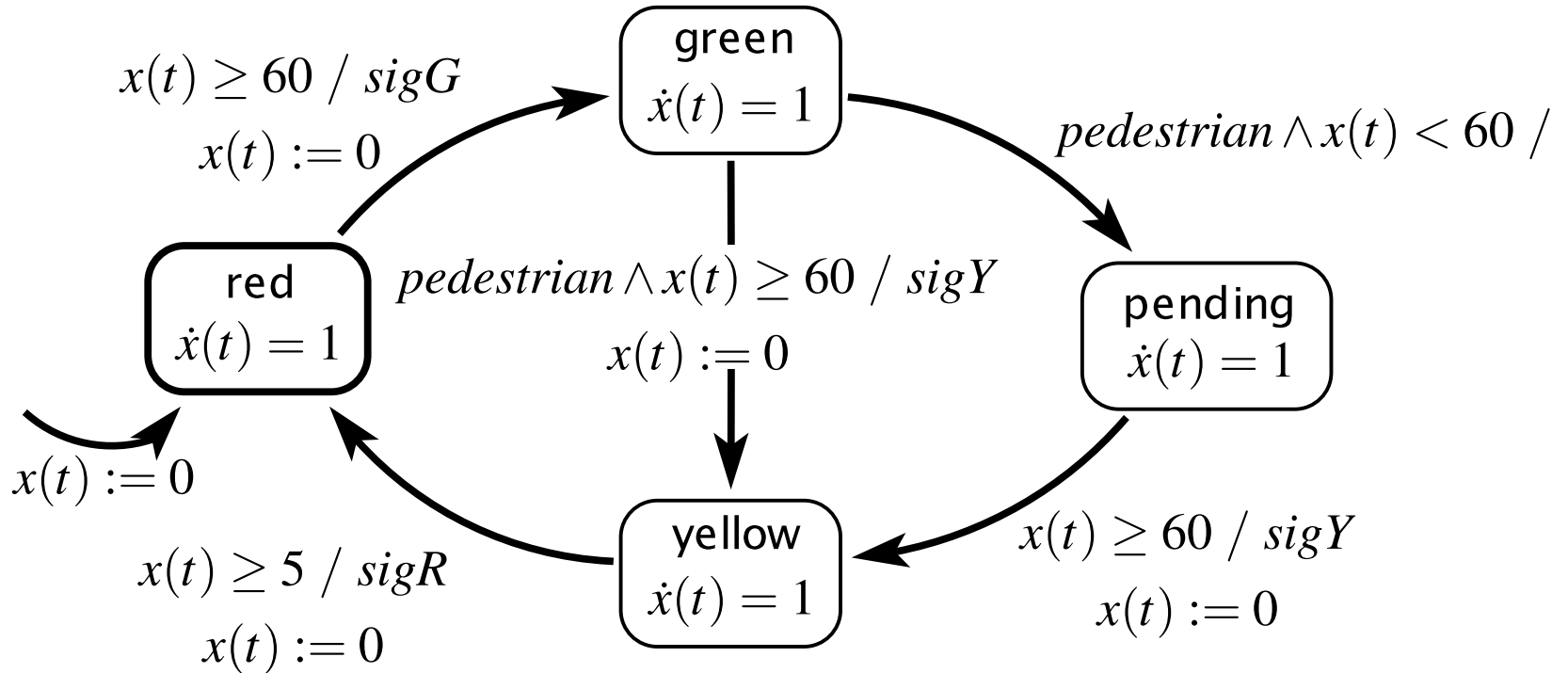
sleepT: Requested delay until next tick

continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

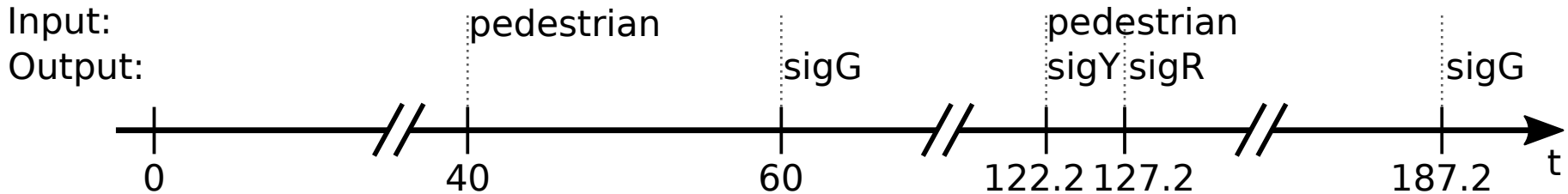
outputs: *sigR*, *sigG*, *sigY*: pure

Assume pedestrian button pressed at $t = 40$ and $t = 122.2$



[Lee/Seshia]

Recall: Eager Semantics

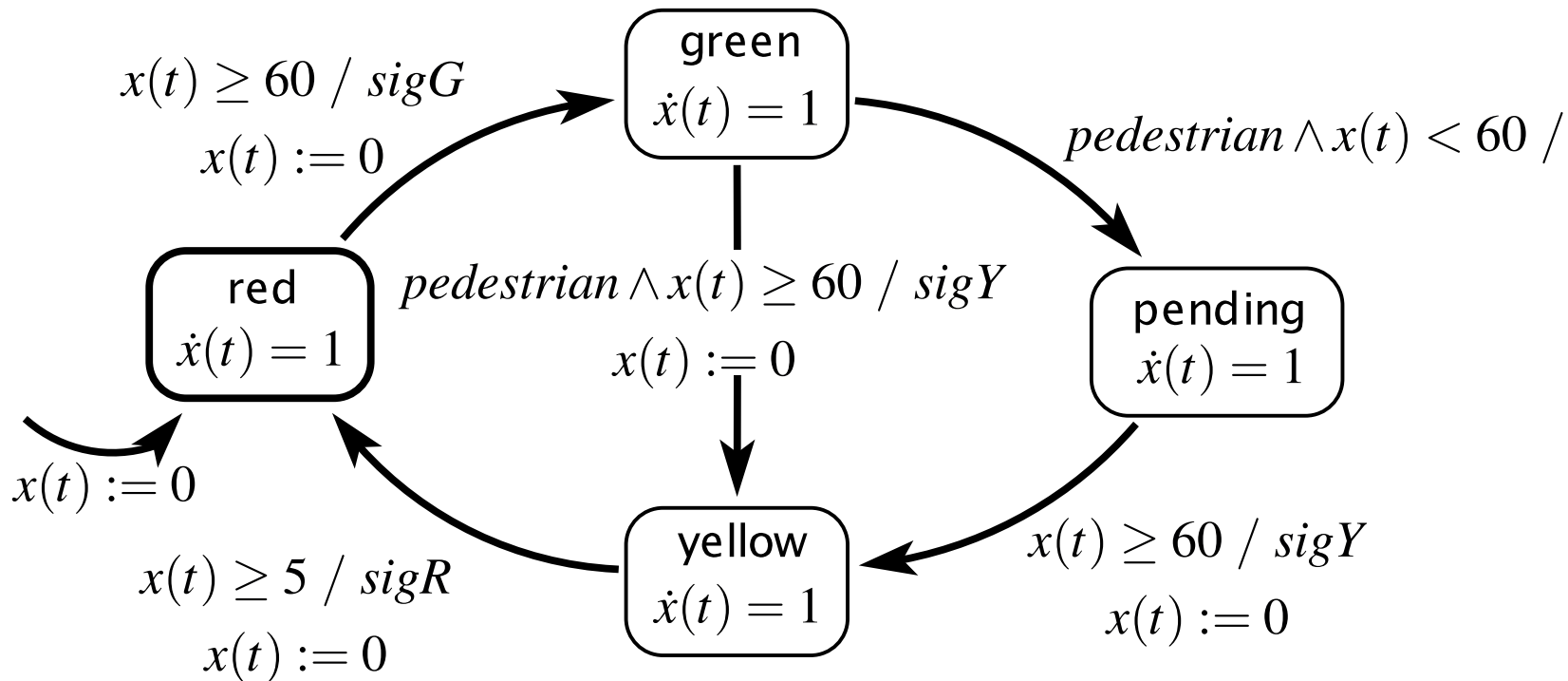


continuous variable: $x(t) : \mathbb{R}$

inputs: *pedestrian*: pure

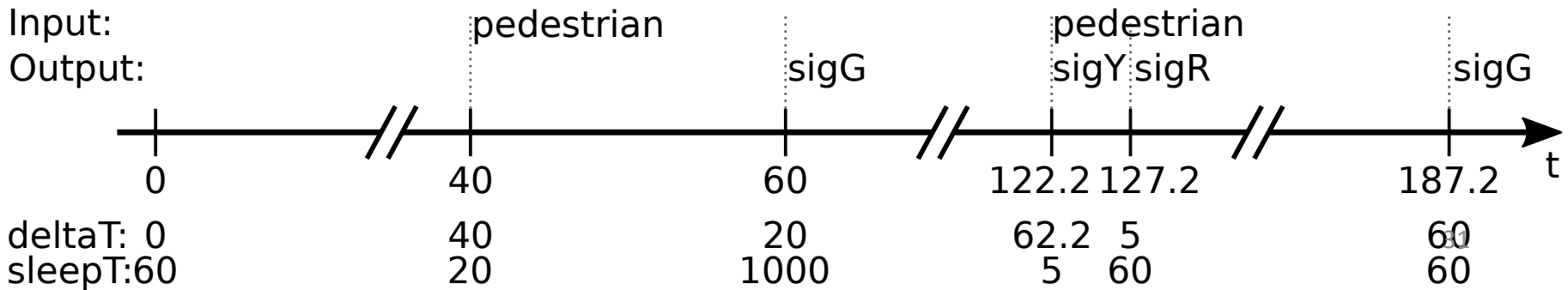
outputs: *sigR*, *sigG*, *sigY*: pure

Assume pedestrian button pressed at $t = 40$ and $t = 122.2$



[Lee/Seshia]

Eager Execution with Dynamic Ticks:



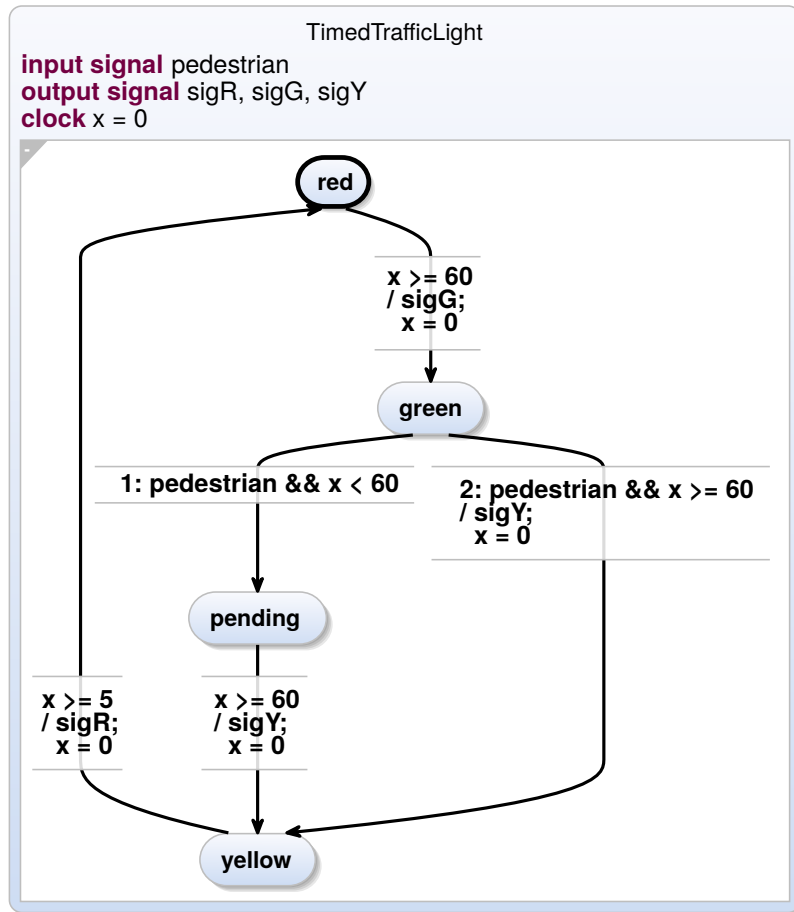
Multiform Notion of Time – Again!

- Semantically, treat clocks (time) as a unit-less number
- As in timed automata, clocks must satisfy *monotonicity* (modulo resets) and *progress*
- Current implementation maps time (clock variables) to an approximation of real numbers (float), interpreted as seconds
- However, could also map clocks to integers, interpreted as Euros spent, fathoms travelled, or beers consumed

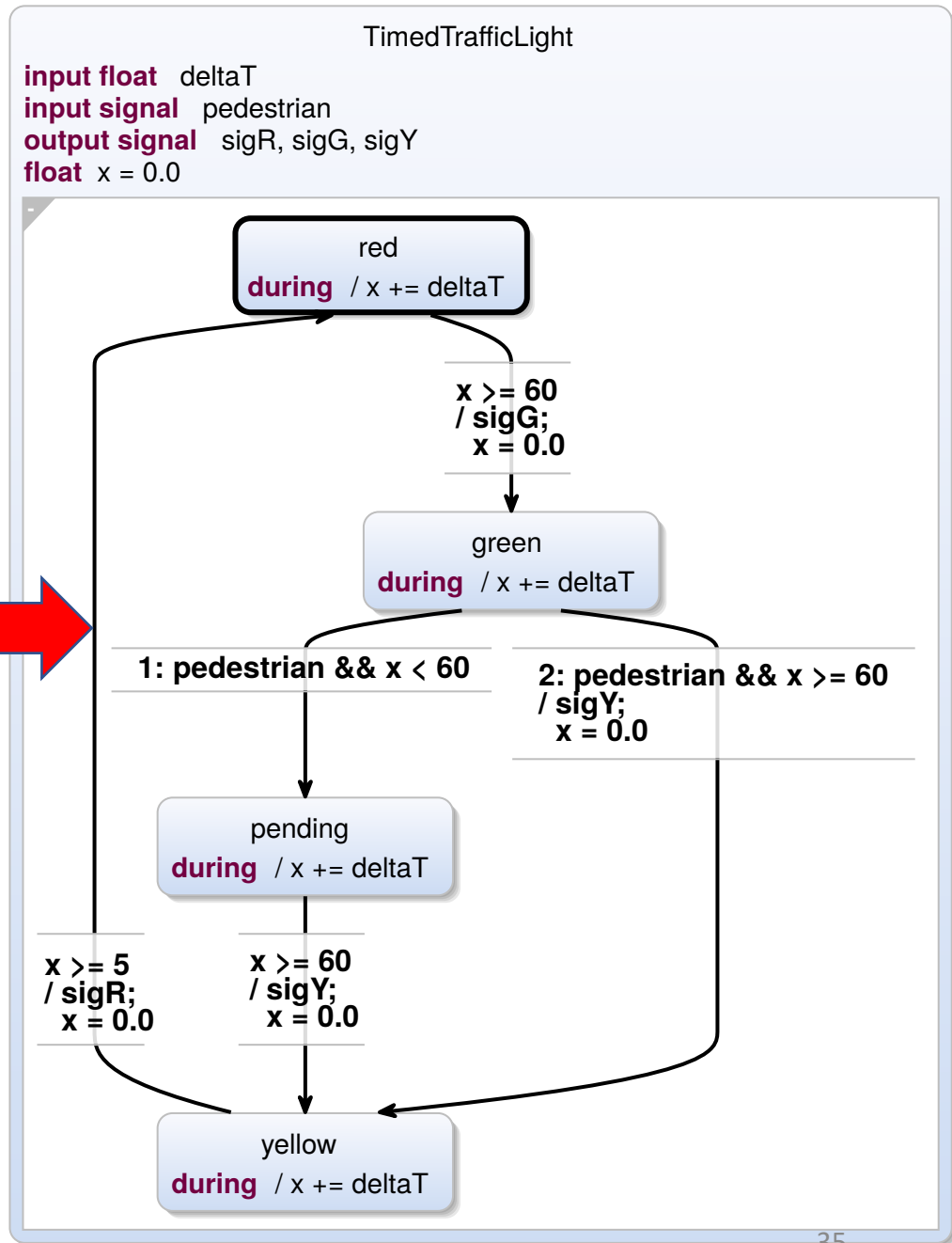
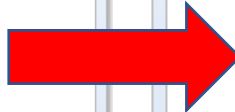
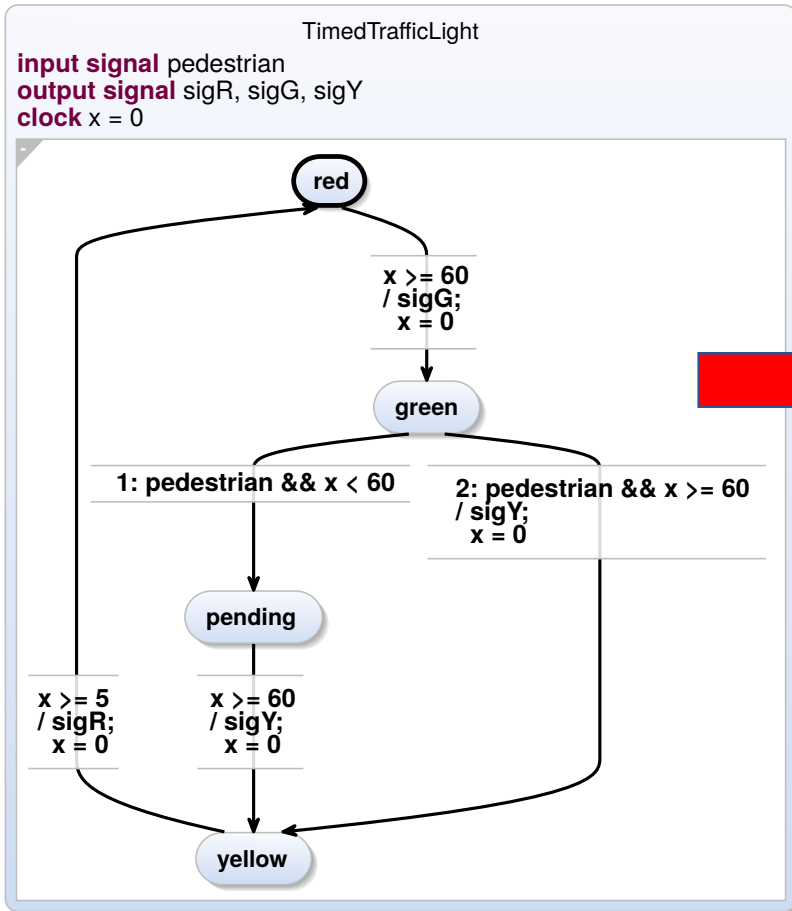
Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
- 4. Time in SCCharts: “clock”**
5. Multiclocks in SCCharts: “period”
6. Demo

Recall: Traffic Light in SCCharts



1st: Expand Clock



TimedTrafficLight

input signal pedestrian
output signal sigR, sigG, sigY
clock x = 0

red

input float deltaT
input signal pedestrian
output signal sigR, sigG, sigY
float x = 0.0

1: pedestrian && x < 60

pending

x >= 5
/ sigR;
x = 0

x >= 60
/ sigY;
x = 0

yellow

TimedTrafficLight

red

during / x += deltaT

x >= 60
/ sigG;
x = 0.0

green

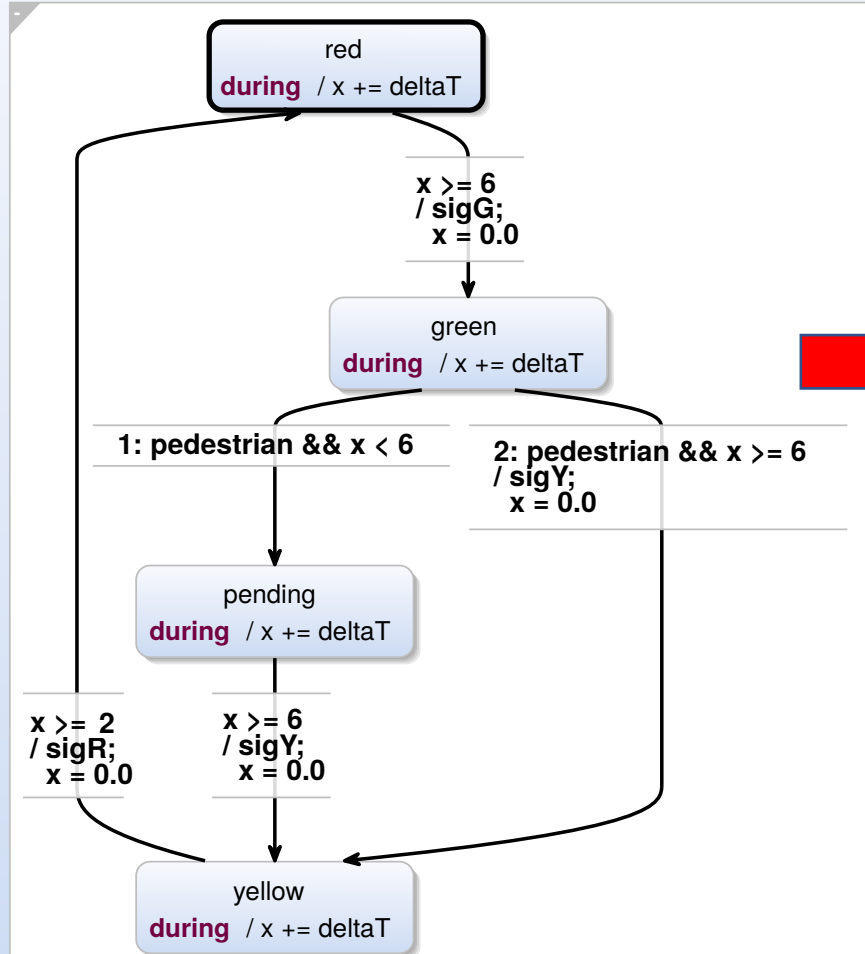
during / x += deltaT



2nd: Add Dynamic Ticks

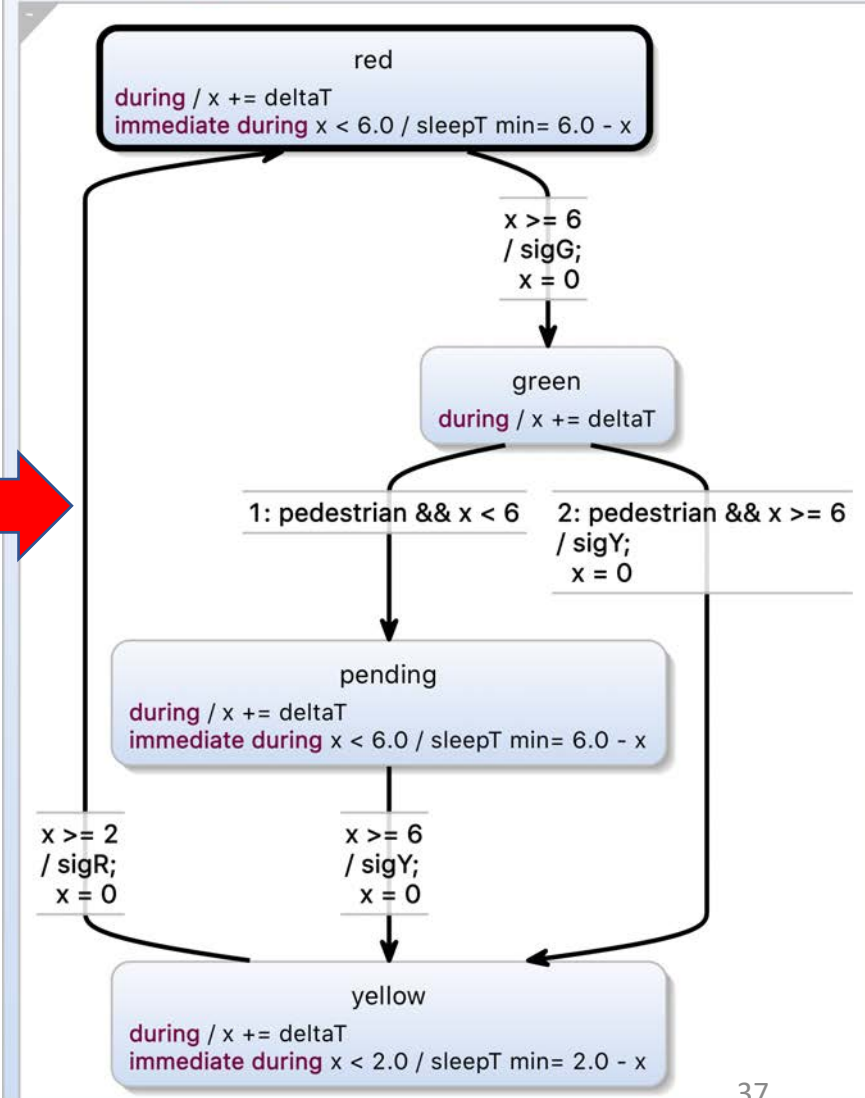
TimedTrafficLight

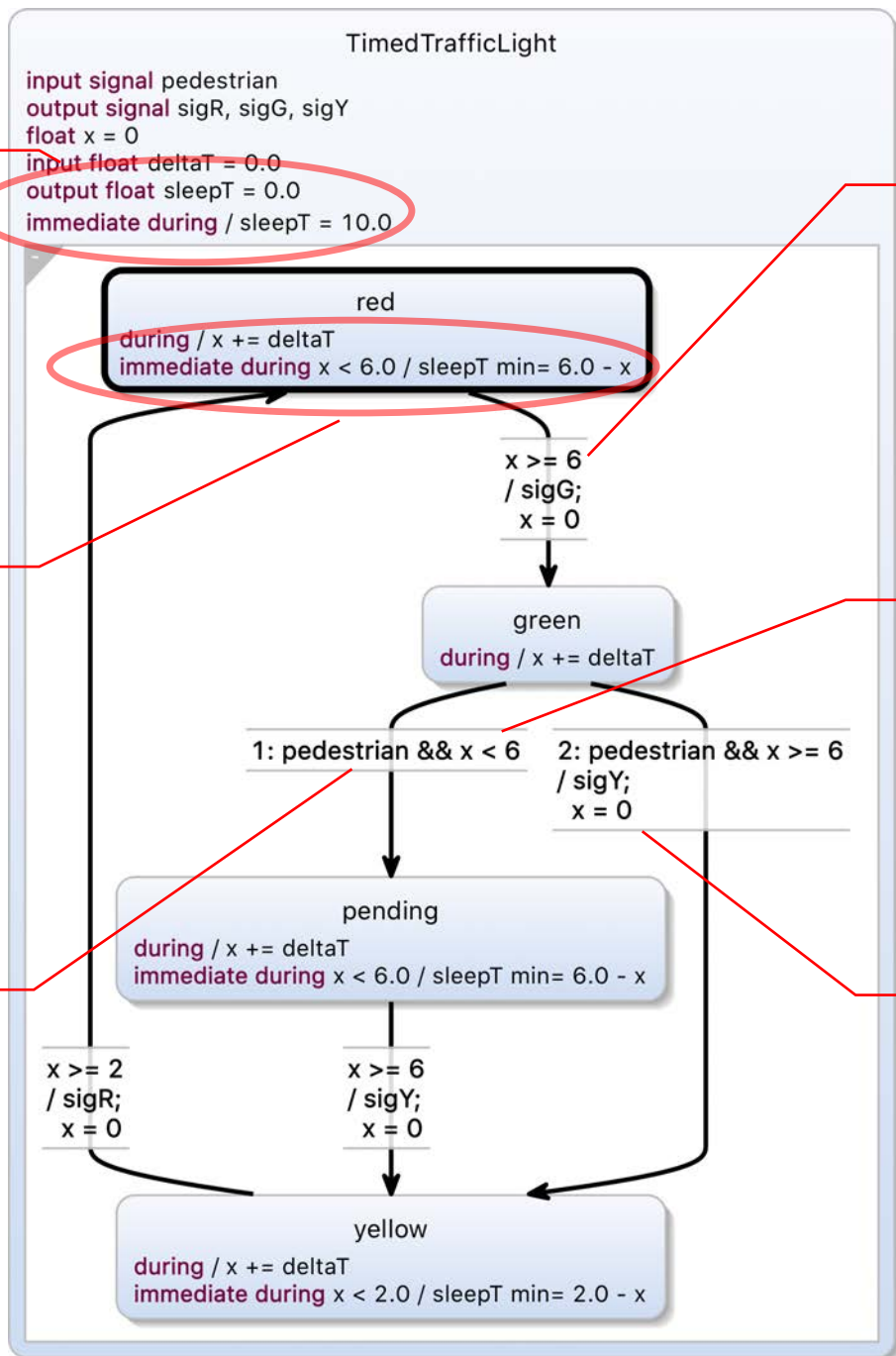
input float deltaT
input signal pedestrian
output signal sigR, sigG, sigY
float x = 0.0



TimedTrafficLight

input signal pedestrian
output signal sigR, sigG, sigY
float x = 0
input float deltaT = 0.0
output float sleepT = 0.0
immediate during / sleepT = 10.0





For sleepT,
SC MoC orders
reset and
update(s)

"x ≥ 6" induces
*lower timing
bound (ltb) of 6*

Must guard
timeout tick

"x < 6" cancels
ltb of 6

No reset here

For x,
SC MoC orders
reset, update,
read

Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
- 5. Multiclocks in SCCharts: “period”**
6. Demo

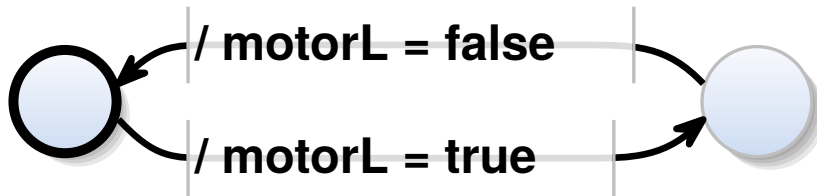
Multiclocks

Motor

output bool motorL = false, motorR = false

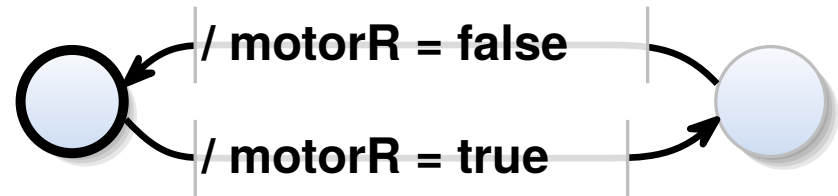
- Left

period 4.2



- Right

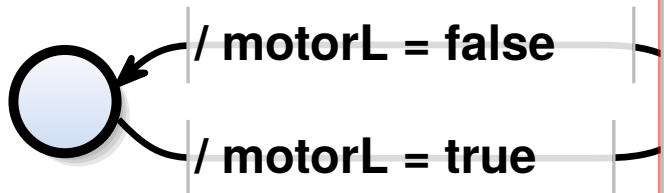
period 1.0



Multiclocks

output bool motorL = false, motorR = false

Left
period 4.2

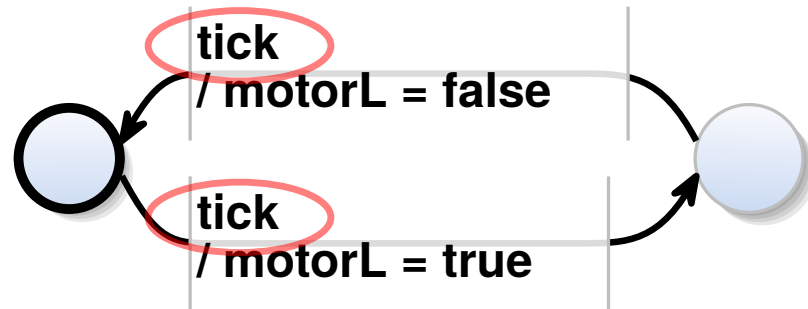


Motor

output bool motorL = false, motorR = false

clock x = 0
bool tick = false

Left



Period

1: x >= 4.2
/ x = 0;
tick = true

2: / tick = false



clock
bool

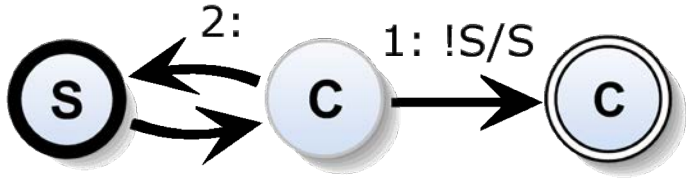
Right



Period

Roadmap

1. Traffic Light Example
2. Execution Models
3. Dynamic Ticks
4. Time in SCCharts: “clock”
5. Multiclocks in SCCharts: “period”
6. Demo



SCCharts

<http://www.sccharts.com/>



KIELER

The Key to Efficient Modeling

<http://www.rtsys.informatik.uni-kiel.de/en/research/kieler>



Eclipse Layout Kernel

<https://www.eclipse.org/elk/>

All available as open source under EPL

"avoid accumulations of timer imperfections"

Grab deltaT from environment, derive from it physical time t

The screenshot shows the KIELER IDE with a state machine diagram for a motor. The code on the left defines a state machine with two regions, 'Left' and 'Right'. The 'Left' region has a period of 4.2 and the 'Right' region has a period of 1.0. Both regions use a counter 'c' and a tick variable to manage state transitions. The state machine diagram shows the 'Left' and 'Right' regions with their respective state transitions and guards. The console at the bottom shows the values of variables over time, with 't' (physical time) increasing from 0 to 13.4530. The status bar at the bottom right shows 'deltaT: 0,2410 sec; sleepT: 1,0000 sec'.

```
1=@DynamicTicks
2 @DefaultSleep 1000
3 scchart Motor {
4= output bool motorL = false,
5   motorR = false
6 input float deltaT
7= @PrintFormat "%.4f"
8   output float t
9
10 during do t += deltaT
11
12 region Left {
13= @HardReset // In paper: @Har
14   period 4.2
15
16= initial state Off ""
17   do motorL = true go to On
18
19= state On ""
20   do motorL = false go to Off
21 }
22
23 region Right {
24= @HardReset // In paper: @Har
25   period 1.0
```

Variable	Value	User	History
deltaT	0,7720		0,7720, 0,2540, 1,0360, 1,0400, 1,0320, 0,8680, 0,1500, 1,0320, 1,0280, 1,0340, 0,9940, 0,0470, 1,0480, 1,0420, 1,0300, 1,0460, 0,0000
motorL	true		true, true, false, false, false, false, true, true, true, true, true, false, false, false, false, false
motorR	true		true, false, true, false, true, false, false, true, false, true, false, true, false, true, false
sleepT	1,0000		1,0000, 0,7460, 0,2240, 1,0000, 1,0000, 1,0000, 0,8500, 0,1120, 1,0000, 1,0000, 1,0000, 0,9530, 0,0340, 1,0000, 1,0000, 1,0000, 1,0000
t	13,4530		13,4530, 12,6810, 12,4270, 11,3910, 10,3510, 9,3190, 8,4510, 8,3010, 7,2690, 6,2410, 5,2070, 4,2130, 4,1660, 3,1180, 2,0760, 1,0460, 0,0000

PROBLEM: After 13 sec, accumulated 0.453 sec delay!

The culprit: always reset clocks to 0!

“avoid accumulations of timer imperfections”

SOLUTION: Change @HardReset (in paper) to @SoftReset (now default)

The screenshot displays the KIPLER IDE interface for a motor control system. The code on the left defines two regions, Left and Right, each with a state transition diagram. The Left region uses a counter 'c' to track time, with a state transition from Off to On when 'c' reaches 4.2. The Right region uses a counter 'c' to track time, with a state transition from Off to On when 'c' reaches 1.0. The console window at the bottom shows the simulation results, with a red circle around the value 13,0270 for variable 't'.

```
1=@DynamicTicks
2 @DefaultSleep 1000
3 scchart Motor {
4   output bool motorL = false,
5     motorR = false
6   input float deltaT
7   @PrintFormat "%.4f"
8   output float t
9
10  during do t += deltaT
11
12  region Left {
13    @SoftReset // In paper: @HardReset
14    period 4.2
15
16    initial state Off ""
17    do motorL = true go to On
18
19    state On ""
20    do motorL = false go to Off
21  }
22
23  region Right {
24    @SoftReset // In paper: @HardReset
25    period 1.0
```

Left

Right

1: c >= 4.2 / c -= 4.2; tick = true

2: / tick = false

during / c += deltaT / immediate during c < 4.2 / sleepT min= 4.2 - c

1: c >= 1.0 / c -= 1.0; tick = true

2: / tick = false

during / c += deltaT / immediate during c < 1.0 / sleepT min= 1.0 - c

Variable	Value	User	History
deltaT	0,4110		0,4110, 0,5750, 1,0330, 0,9880, 0,9880, 0,6240, 0,3620, 1,0340, 0,9870, 1,0000, 0,7920, 0,2060, 1,0060, 0,9950, 0,9800, 1,0460, 0,0000
motorL	true		true, true, false, false, false, false, true, true, true, true, true, false, false, false, false
motorR	true		true, false, true, false, true, false, false, true, false, true, false, true, false, true, false
sleepT	0,9730		0,9730, 0,3840, 0,5590, 0,9920, 0,9800, 0,9680, 0,5920, 0,3540, 0,9880, 0,9750, 0,9750, 0,7670, 0,1730, 0,9790, 0,9740, 0,9540, 1,0000
t	13,0270		13,0270, 12,6160, 12,0410, 11,0080, 10,0200, 9,0320, 8,4080, 8,0460, 7,0120, 6,0250, 5,0250, 4,2330, 4,0270, 3,0210, 2,0260, 1,0460, 0,0000

After 13 sec, accumulated only 0.027 sec delay!

Instead of reset to 0, subtract previously requested time-out

Summary

- Timed automata used not just for verification, but also for synthesis
- Synchronous execution model cleanly contains non-determinism, at timing I/O-interface
- Can extend easily to multi-clock design
- Multiform notion of time retained – but package “time” not as events, but clocks (represented as, e.g., integers)
- Added two keywords (clock, period) as extended SCChart features
- Further annotations (@HardReset, @DefaultSleep) to control external interface
- Same concepts can be applied to other synchronous languages

To Go Further

- Reinhard von Hanxleden, Timothy Bourke, Alain Girault.
Real-Time Ticks for Synchronous Programming.
Proc. Forum on Specification and Design Languages (FDL '17), Verona, Italy, September 2017
<https://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/fdl17.pdf>
- Alexander Schulz-Rosengarten, Reinhard von Hanxleden, Frédéric Mallet, Robert de Simone, Julien Deantoni.
Time in SCCharts.
In *Proc. Forum on Specification and Design Languages (FDL '18)*, Munich, Germany, September 2018
<https://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/fdl18.pdf>