Timed languages
for discrete-event systems

Ernesto Posse

Outline

- Introduction & Motivation
- Real-time
- Time in modelling formalisms
- Time in general purpose programming languages
- Properties of time
- A mini-timed language
- An example
- Conclusion

Introduction & Motivation

- Why time?
- Dynamic systems: change of state over time
- Implicit vs. explicit time
  - To describe time-dependent behaviour (modelling):
    Do a task with given time-constraints
  - To answer questions (analysis):
    When? How long?
    Will it happen before/after/between ...?
  - Need for observing the time of events or changes

Introduction & Motivation

- Existing modelling formalisms:
  - Timed Automata (Alur & Dill ’90, Lynch & Vaandrager ’91)
  - Timed Petri Nets (Merlin ’74)
  - Statecharts (Harel ’84)
  - DEVS (Zeigler ’76 ’2000)

Introduction & Motivation

- Existing languages:
  - LOTOS, E-LOTOS and G-LOTOS
  - Esterel
  - Lustre
  - Signal
  - Argos
  - ...
- Process algebras with timing:
  - Timed CSP
  - Timed CCS
  - Timed ACP

Introduction & Motivation

- Modal logics
  - Real-time CTL
  - Real-time LTL

Introduction & Motivation

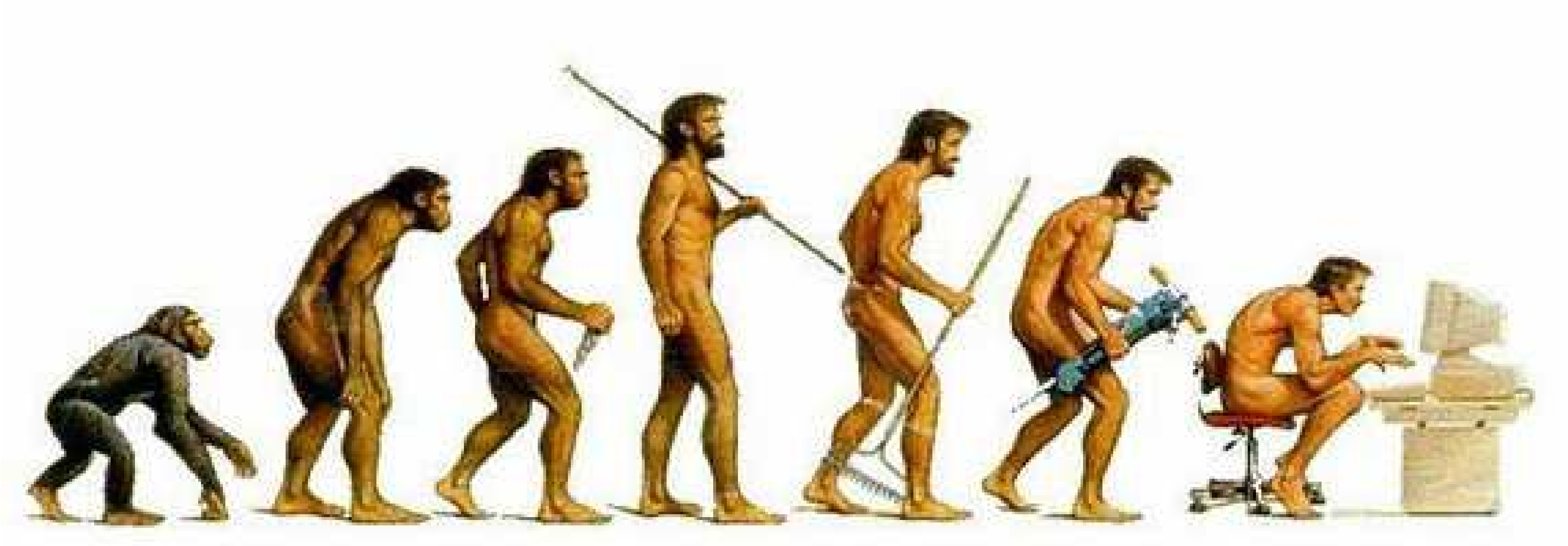
- Who uses these?
- Companies
  - European Space Agency, NASA, Airbus, Lockheed Martin, Texas Instruments, Philips, ...
- Areas
  - Avionics & Aerospace
  - Defence & Military
  - Transportation (railways & automotive)
  - Semiconductors & hardware
  - Telecom
  - Human-computer interaction

Real-time

- Real-time: reactive systems
- Real numbers (continuous-time) vs. natural numbers (discrete-time)
- Discrete Event Systems: continuous-time but only discrete changes of state

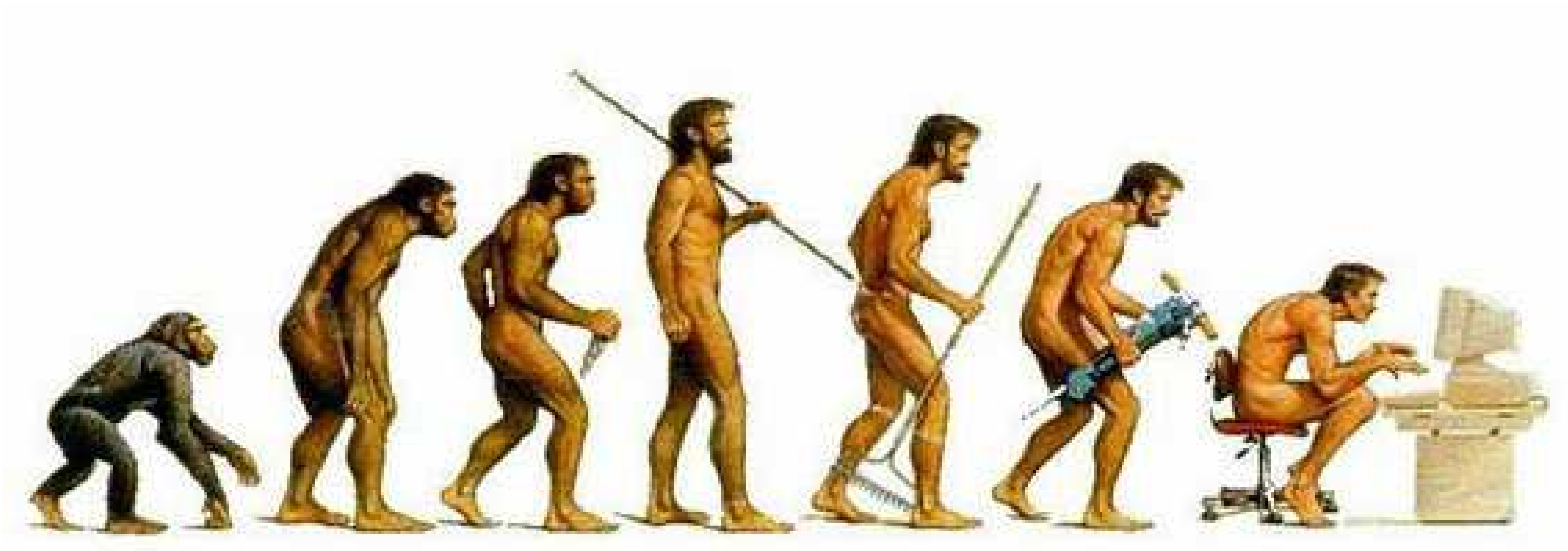
Real-time

- Dinosaurs and circuits



Real-time vs. simulated time

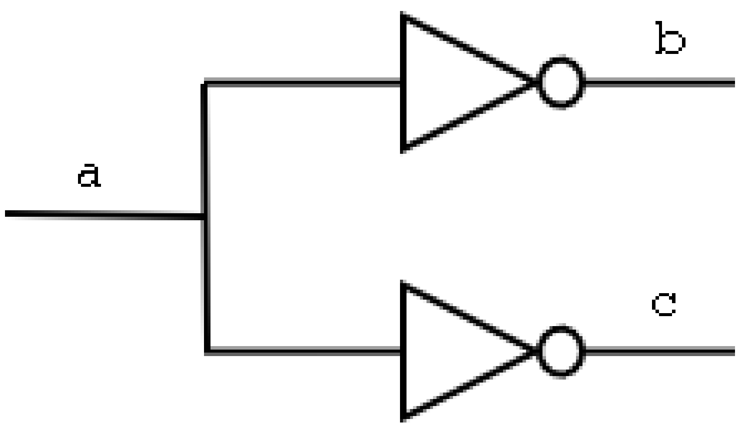
- Dinosaurs and circuits



- Physical clock vs. "logical" clock

Real-time vs. simulated time

- Gates have time delays before firing



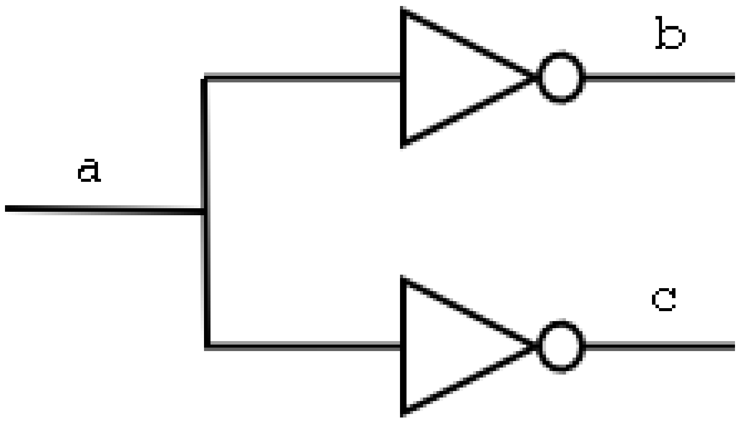
Correct history

(true concurrency)

time	0	2	...
a	0	0	...
b	0	1	...
c	0	1	...

Real-time vs. simulated time

- Gates have time delays before firing



Incorrect history

(naive simulation)

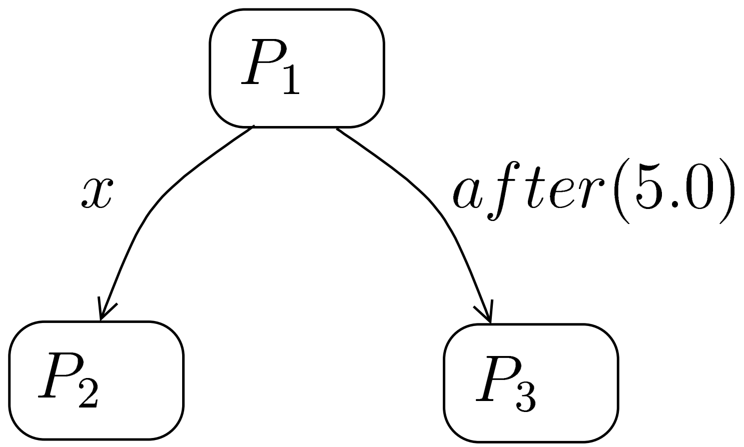
time	0	2	4	...
a	0	0	0	...
b	0	1	1	...
c	0	0	1	...

Real-time vs. simulated time

- Solution:
  - Modelling & Analysis:
    - Abstract physical time as logical time
    - Timed-traces (sequences of events tagged with time-stamps)
    - Abstract simulation algorithms
  - Simulation: event-scheduling
- In timed languages and formalisms, time could be considered either physical or logical.
- Logical trace $\longleftrightarrow$ physical trace

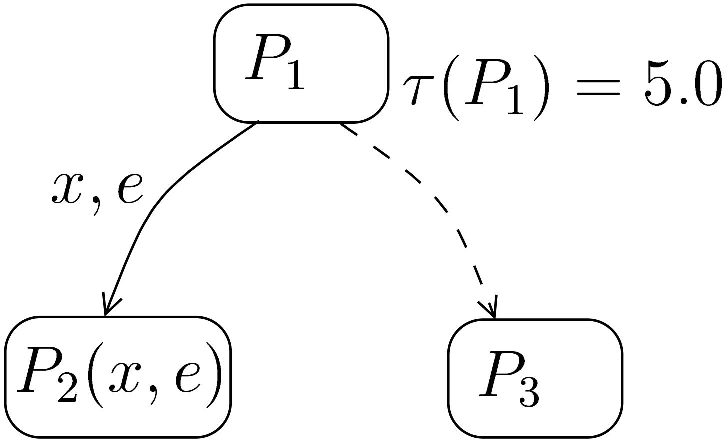
Time in modelling formalisms

- Statecharts: after(*delay*)



Time in modelling formalisms

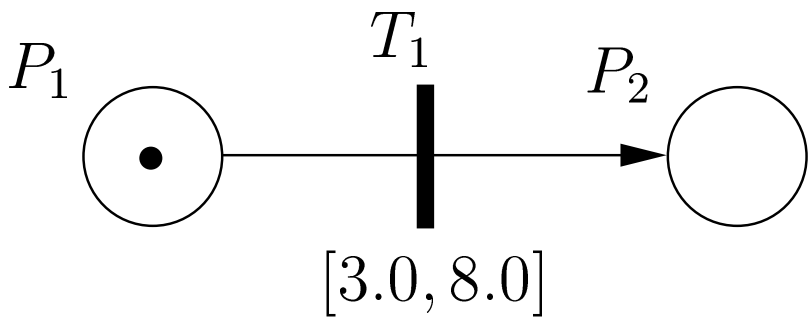
- DEVS: time-advance and elapsed-time



$$
\begin{aligned}
\delta^{ext}((P_1, e), x) &= P_2(x, e) \\
\delta^{int}(P_1) &= P_3
\end{aligned}
$$

Time in modelling formalisms

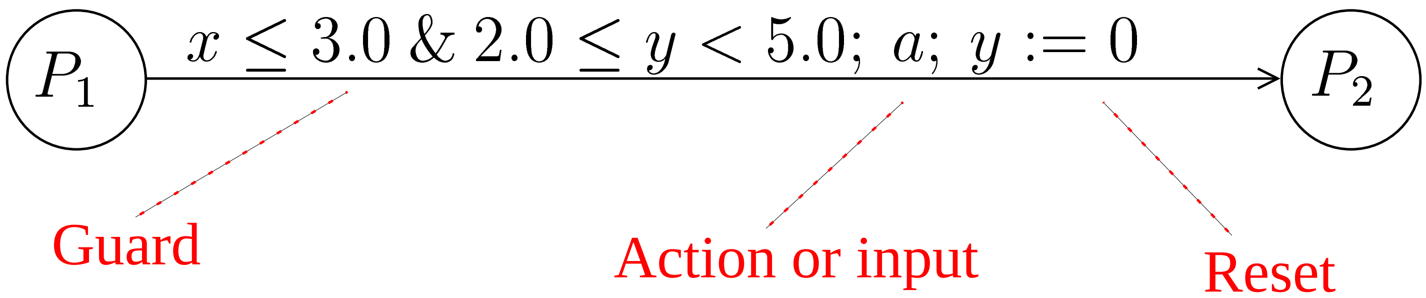
- Timed Petri Nets: (interval) timed-transitions



Time in modelling formalisms

- Timed Automata: multiple clocks, clock guards, clock reset

Clocks: x, y



Time in programming languages

- Library functions/procedures:
  - Sleep
  - Timeout
  - Interrupt
- Implemented based on the underlying OS
- Dependent of the system's clock
- Not primitive language constructs

Time in programming languages

```
def task1():
  do_something()

def task2():
  do_some_other_thing()

t1 = Timer(30.0, task1)
t2 = Timer(25.0, task2)

t1.start()
t2.start()

sleep(20.0)
t1.cancel()
```

Time in programming languages

```
class A(Thread):
  def run(self):
    sleep(5.0)
class B(Thread):
  def __init__(self, other):
    Thread.__init__(self)
    self.other = other
  def run(self):
    self.other.join(3.0)

a = A()
b = B(a)
a.start()
b.start()
```

Time in programming languages

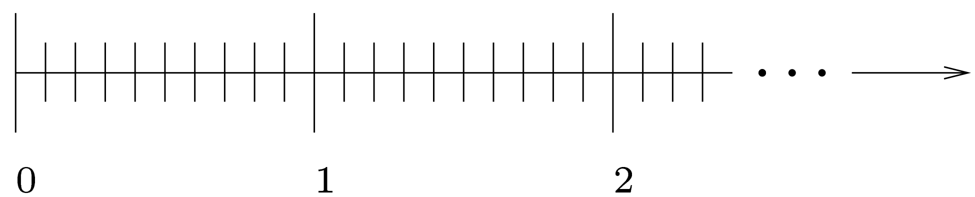
```
class A(Thread):
  def run(self):
    sleep(5.0)
class B(Thread):
  def __init__(self, other):
    Thread.__init__(self)
    self.other = other
  def run(self):
    self.other.join(3.0)
    if self.other.isAlive():
      course_of_action_1()
    else:
      course_of_action_2()
```

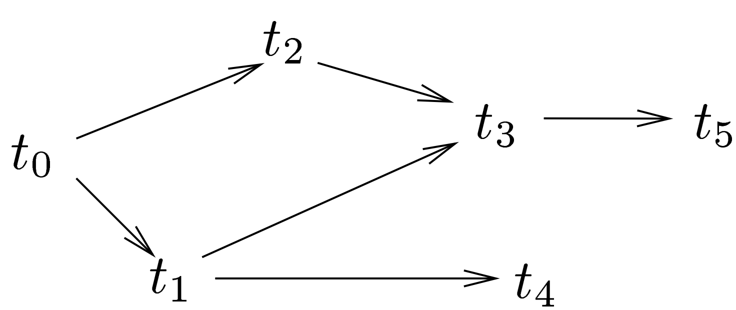
Properties of time

- Time models: set of assumptions and properties of time and systems w.r.t. time.
- Assumptions
  - Events are instantaneous
  - Newtonian time: single global logical clock
  - Real numbers as time-base
  - Maximal parallelism
  - Maximal progress

Properties of time

- Time base:
  - Real numbers vs. natural numbers
  - Total linear order vs. partial order





Properties of time

- Distinguish between “event transitions" and “evolution"
- Event (or action) transitions

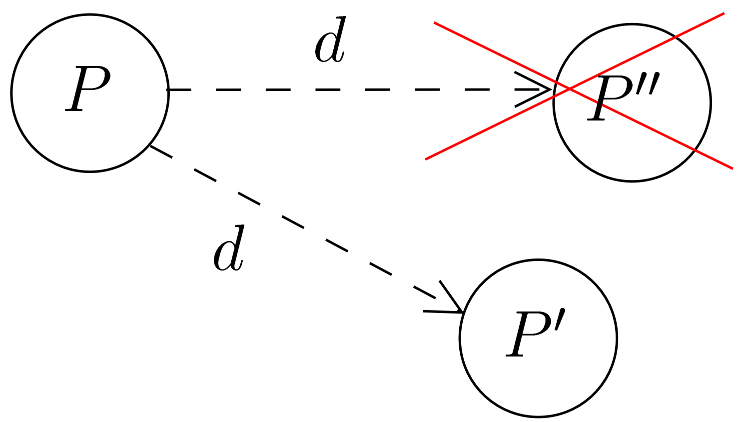
$$P \xrightarrow{\alpha} P'$$

- Time evolution

$$P \overset{d}{\rightsquigarrow} P'$$

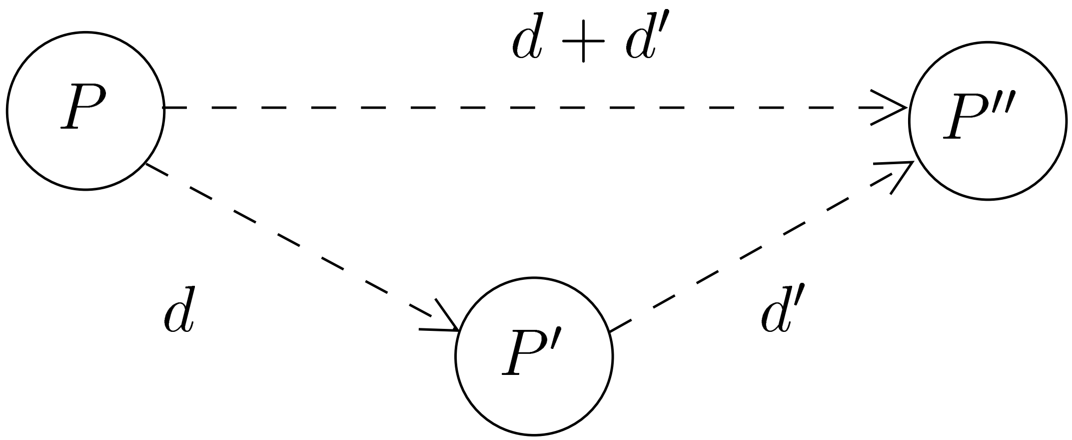
Properties of time

- Evolution is deterministic



Properties of time

- Time additivity and time interpolation



Properties of time

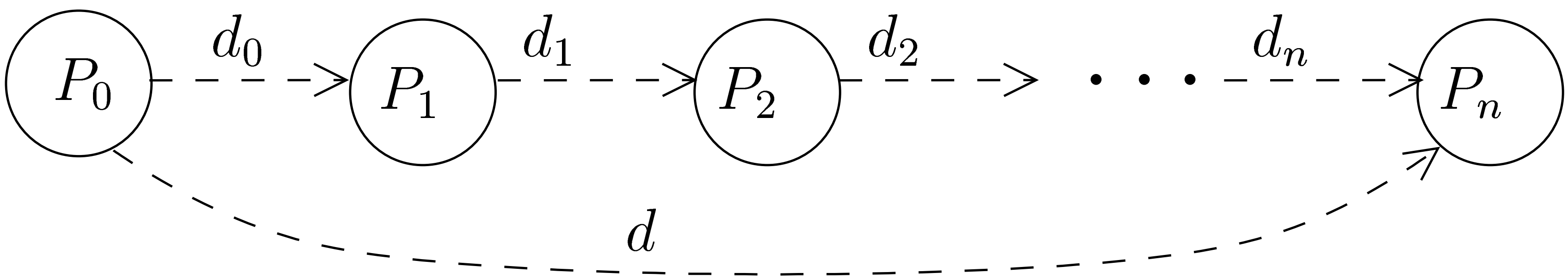
- Time closure
- Zeno sequence (infinite sequence of evolution with finite duration)

$$P_0 \overset{d_0}{\dashrightarrow} P_1 \overset{d_1}{\dashrightarrow} P_2 \overset{d_2}{\dashrightarrow} \cdots$$

$$\sum_{i=0}^{\infty} d_i = d < \infty$$

Properties of time

- Time closure: every Zeno sequence has a limit



$$\sum_{i=0}^{\infty} d_i = d < \infty$$

Properties of time

- No progress
- Zeno-divergence: Never reaching a limit

$$P_0 \overset{1/2}{\dashrightarrow} P_1 \xrightarrow{\alpha_1} P_2 \overset{1/4}{\dashrightarrow} P_3 \xrightarrow{\alpha_2} P_4 \overset{1/8}{\dashrightarrow} \cdots$$

$$\sum_{i=1}^{\infty} \frac{1}{2^i} = 1 < \infty$$

Properties of time

- No progress
- Spin-divergence: getting stuck in an instant

$$P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_1} P_2 \xrightarrow{\alpha_2} P_3 \xrightarrow{\alpha_2} P_4 \xrightarrow{\alpha_2} \cdots$$

Timed languages

- Common primitives
  - Sleep
  - Delay (uninterruptable sleep)
  - Timeout
  - Interrupt
  - Event-time dependence

A mini-timed language

- Describing simple reactive and interactive processes
- Based on Timed CSP and Timed CCS
- Processes exist and execute in parallel
- Communication by message-passing over channels
- Abstraction mechanisms

A mini-timed language

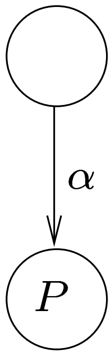
- The dead process

$$0$$

A mini-timed language

- Single action

$$\alpha \to P$$



- Example

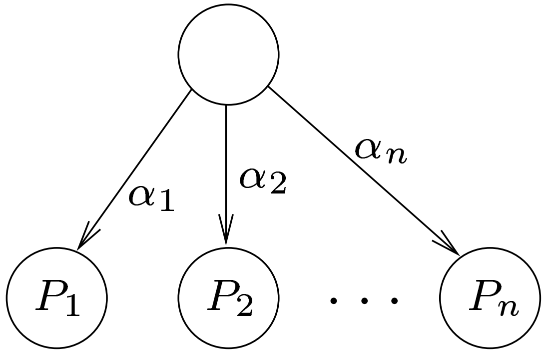
$$Printer = accept.job \to print.job \to 0$$

$$DoInternalStuff = \tau \to \tau \to \tau \to 0$$

A mini-timed language

- Alternative actions

$$\alpha_1 \to P_1 \,|\, \alpha_2 \to P_2 \,|\, \ldots \,|\, \alpha_n \to P_n$$



- Example

$$Printer = accept.job \to print.job \to 0 \,|\, shutdown \to 0$$

A mini-timed language

- Output action (sending a message over a channel)

$$c!v \rightarrow P$$

- Input action (receiving a message over a channel)

$$c?x \rightarrow P(x)$$

- Example

$$Printer = accept?job \rightarrow print!job \rightarrow 0$$

A mini-timed language

- Recursion: loops

$$N = P(N)$$

- Example

$$Printer = accept.job \rightarrow print.job \rightarrow Printer$$

$$OneCellBuffer = in?x \rightarrow out!x \rightarrow OneCellBuffer$$

A mini-timed language

- Parallel composition

$$P_1 \,||\, P_2$$

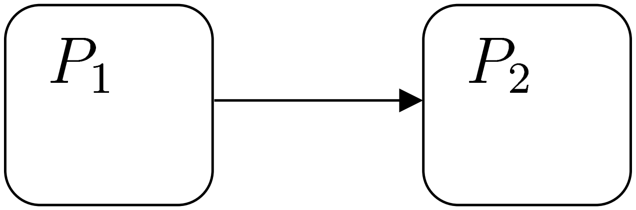
- Example

$$\begin{aligned} Leg_1 &= up.1 \rightarrow down.1 \rightarrow Leg_1 \\ Leg_2 &= down.2 \rightarrow up.2 \rightarrow Leg_2 \\ Robot &= Leg_1 \,||\, Leg_2 \end{aligned}$$

A mini-timed language

- Sequential composition

$$P_1 \,;\, P_2$$



- Example

$$\begin{array}{rcl} Runner & = & run \rightarrow 0 \\ Walker & = & walk \rightarrow 0 \\ Jumper & = & jump \rightarrow 0 \\ Group & = & (Runner \parallel Walker); Jumper \end{array}$$

A mini-timed language

- (Limited) support for dynamic structure:

$$Virus = reproduce \rightarrow (Virus \,||\, Virus)$$

$$\begin{aligned} Virus &\xrightarrow{reproduce} Virus \,||\, Virus \\ &\xrightarrow{reproduce} Virus \,||\, Virus \,||\, Virus \\ &\xrightarrow{reproduce} Virus \,||\, Virus \,||\, Virus \,||\, Virus \\ &\xrightarrow{reproduce} \ldots \end{aligned}$$

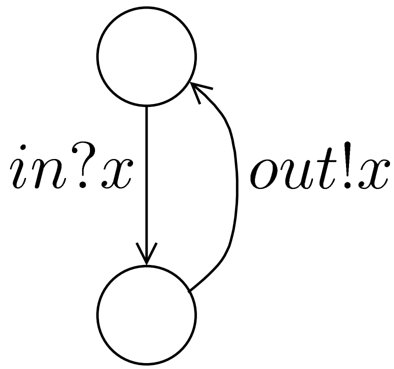
A mini-timed language

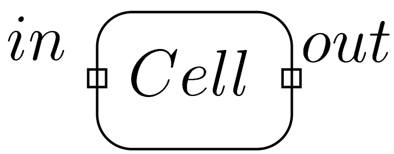
- Communication:
  - Message-passing over channels
  - Unicasting vs. Multicasting
  - Synchronous vs. asynchronous

A mini-timed language

- Communication

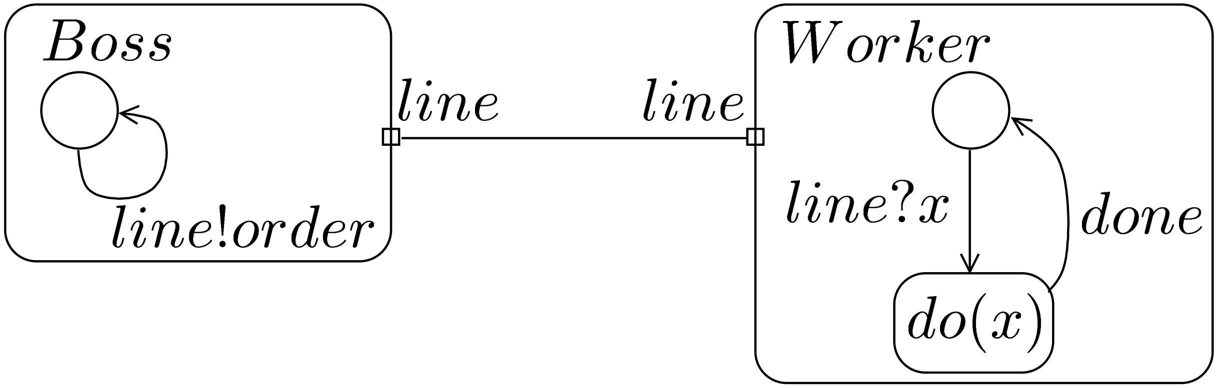
$$Cell = in?x \rightarrow out!x \rightarrow Cell$$





A mini-timed language

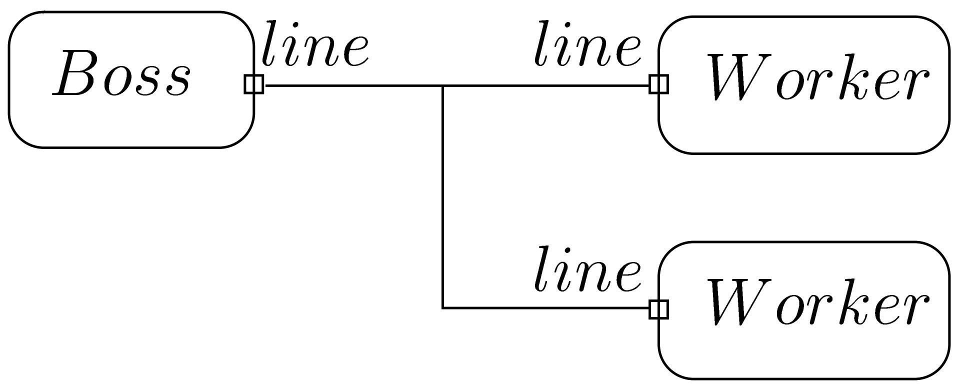
$$
\begin{aligned}
Boss &= line!order \rightarrow Boss \\
Worker &= line?x \rightarrow do(x) \rightarrow Worker \\
Factory &= Boss \,||\, Worker
\end{aligned}
$$



- Channels are common names

A mini-timed language

$$BigFactory \quad = \quad Boss \,||\, Worker \,||\, Worker$$



- Unicasting vs. Multicasting
- Unicasting leads to non-determinism

A mini-timed language

$$
\begin{aligned}
Boss &= line!order \to Boss \\
Worker &= line?x \to do(x) \to Worker \\
Factory &= Boss \,||\, Worker
\end{aligned}
$$

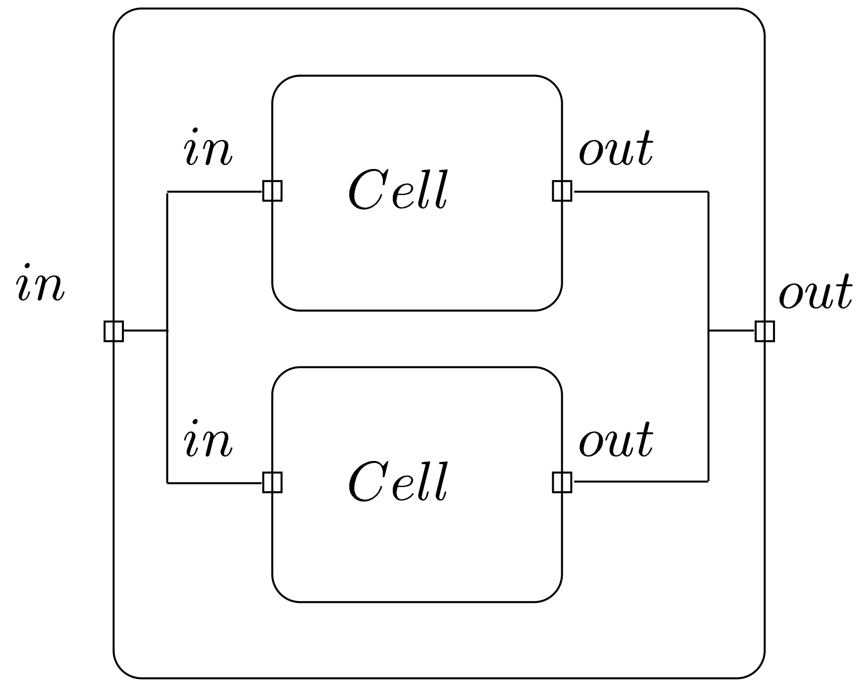
- Synchronous communication: (rendez-vous or handshake) send action is blocking
- Asynchronous communication: send action is non-blocking

A mini-timed language

- Channels are common names

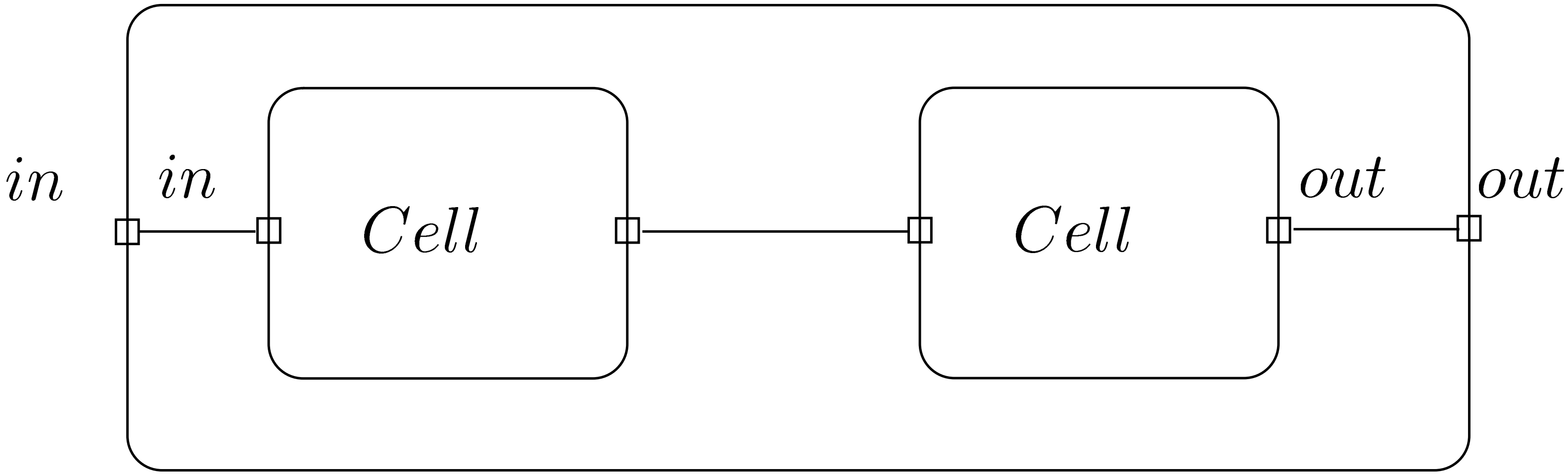
$$Cell = in?x \to out!x \to Cell$$

$$Cell \,||\, Cell$$



A mini-timed language

But what if we want



A mini-timed language

- Process interface: parameters in its definition

$$Cell(in, out) = in?x \to out!x \to Cell(in, out)$$

  in and out are now private

- Such definition can be thought of as a “class” of processes

A mini-timed language

- Process instantiation

$$Cell(a, b)$$

becomes

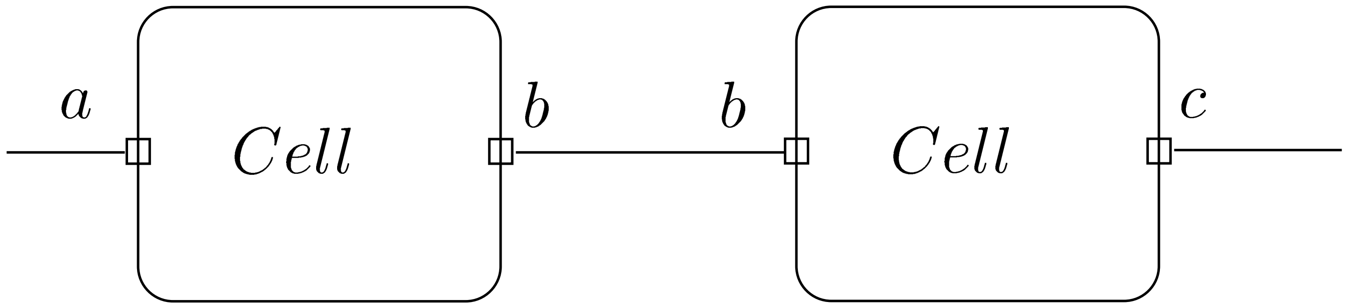
$$a?x \to b!x \to Cell(a, b)$$

A mini-timed language

$$Cell(a,b)\,||\,Cell(b,c)$$

$$=$$

$$a?x \rightarrow b!x \rightarrow Call(a,b) \quad || \quad b?x \rightarrow c!x \rightarrow Cell(b,c)$$

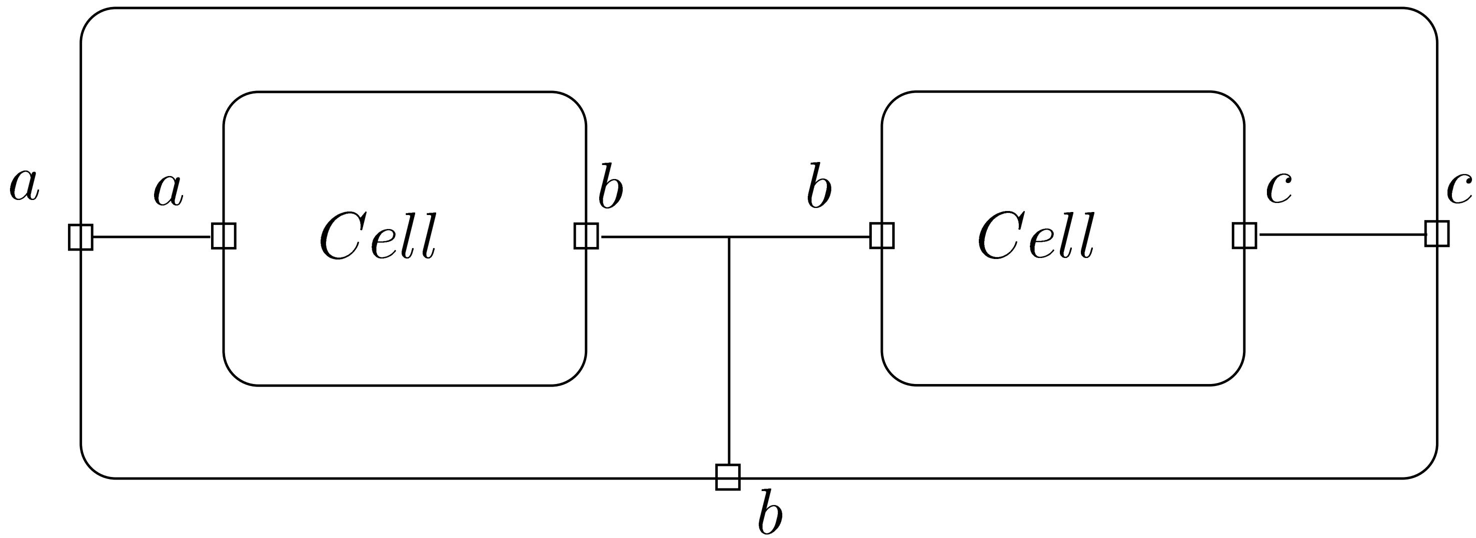


A mini-timed language

$$Cell(a, b) \,||\, Cell(b, c)$$

$$=$$

$$a?x \rightarrow b!x \rightarrow Call(a, b) \quad || \quad b?x \rightarrow c!x \rightarrow Cell(b, c)$$



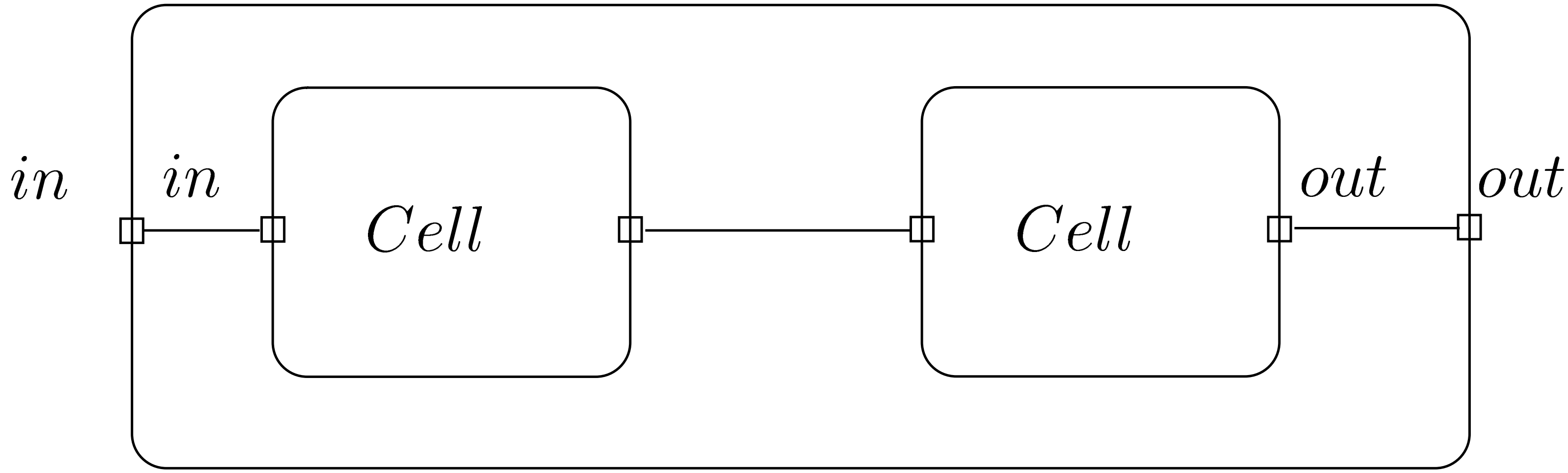
A mini-timed language

- Hiding (abstraction)

$$P \backslash \{x_1, x_2, ..., x_n\} \text{ or new } x_1, x_2, ..., x_n : P$$

- Example

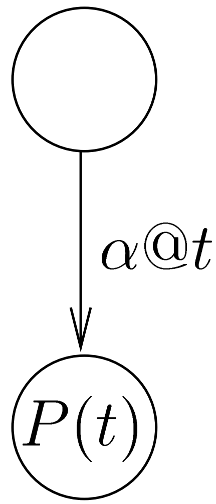
$$TwoCellBuffer(in, out) = (Cell(in, m) \,||\, Cell(m, out)) \backslash \{m\}$$



A mini-timed language

- Timed-prefix (when)

$$\alpha@t \to P(t)$$



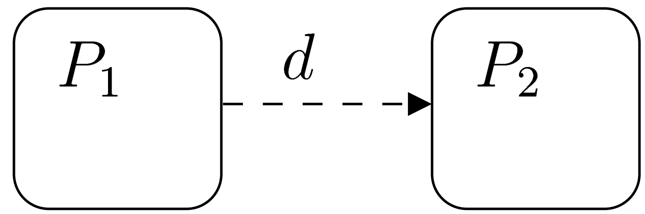
- Example

$$Timer(in, out) = begin \to in?x@e \to out!e \to Timer(in, out)$$

A mini-timed language

- Time-out (non-blocking wait)

$$P_1 \overset{d}{\triangleright} P_2$$



- Example

$$Printer = (accept?job \rightarrow print!job \rightarrow 0) \overset{100}{\triangleright} (shutdown \rightarrow 0)$$

$$AtomicDEVSstate = (in?x@e \rightarrow S_1(x, e)) \overset{ta}{\triangleright} S_2$$

A mini-timed language

- Simple delay (blocking wait)

$$\alpha \xrightarrow{d} P$$

$$=$$

$$\alpha \rightarrow (0 \overset{d}{\triangleright} P)$$

A mini-timed language

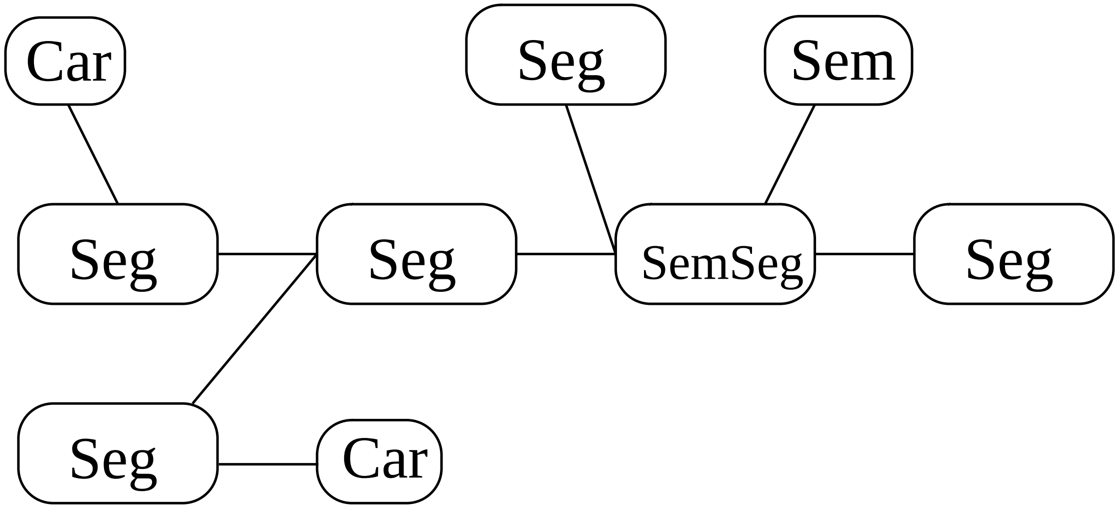
- Interval delay (non-deterministic delay)

$$\alpha \xrightarrow{D} P$$

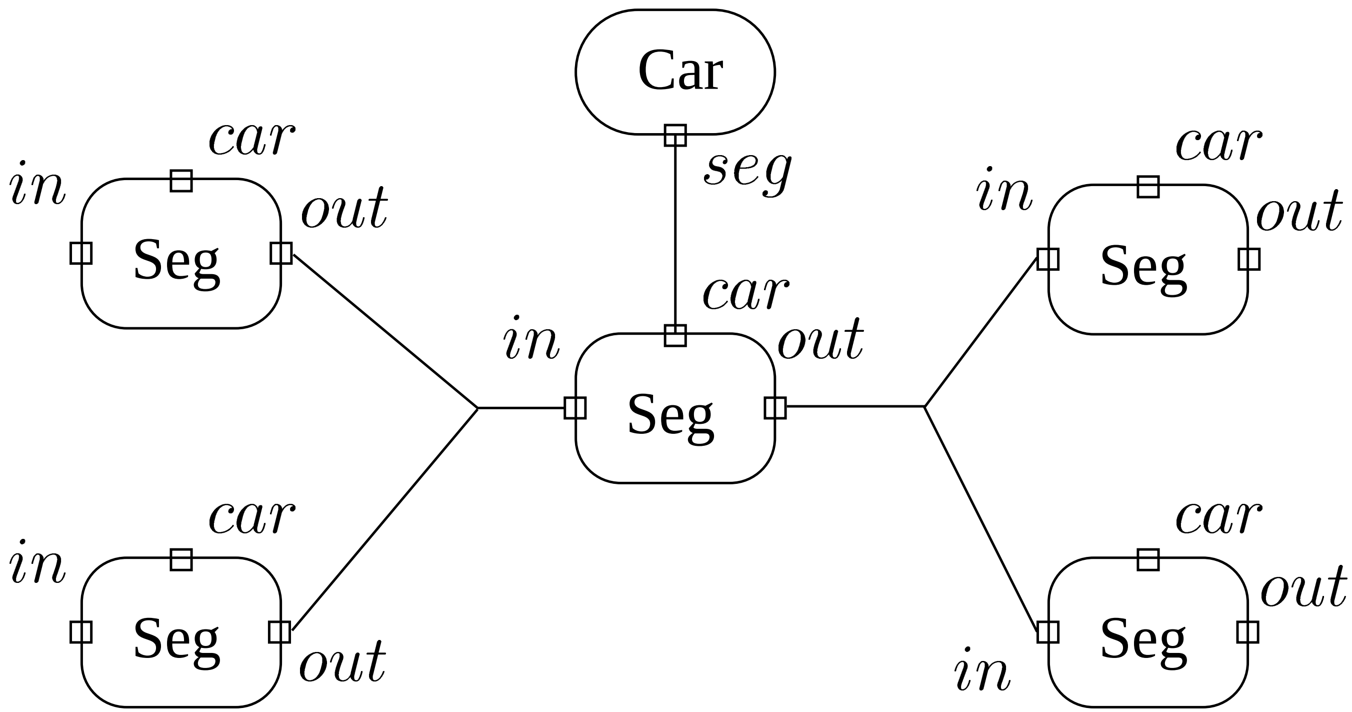
- Examples:

$$Runner = ready \rightarrow set \rightarrow go! \xrightarrow{[6.0,20.0]} finnish \rightarrow 0$$

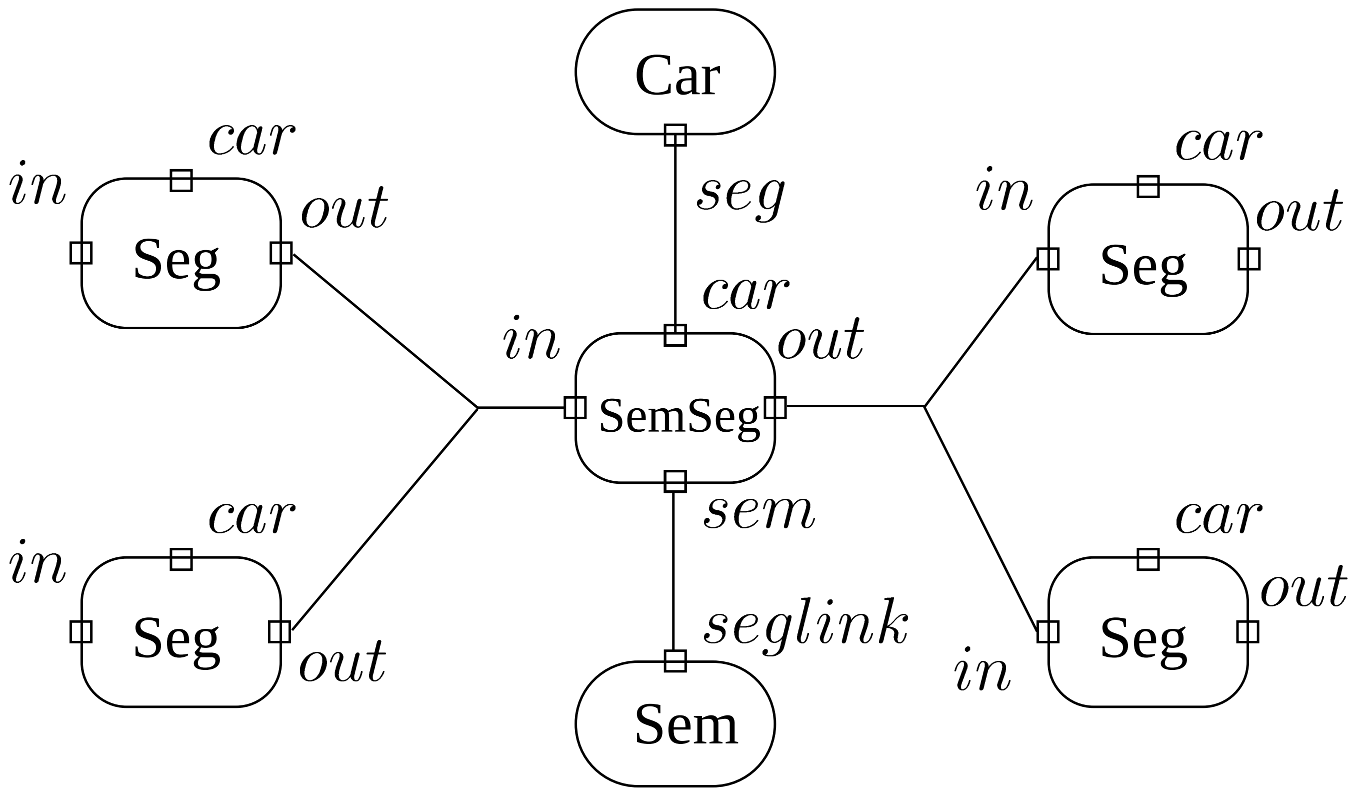
Example: a traffic network



Example: a traffic network



Example: a traffic network



Example: a traffic network

$$
\begin{aligned}
Sem(seg) &= Red(seg) \\
Red(seg) &= 0 \overset{10}{\triangleright} Green(seg)(10) \\
Green(seg)(n) &= (seg!@e \rightarrow Green(seg)(n-e)) \overset{n}{\triangleright} Red(seg)
\end{aligned}
$$

$$
\begin{aligned}
Car(seg)(speed) &= seg?length \xrightarrow{length/speed} seg! \\
&\rightarrow Car(seg)(speed)
\end{aligned}
$$

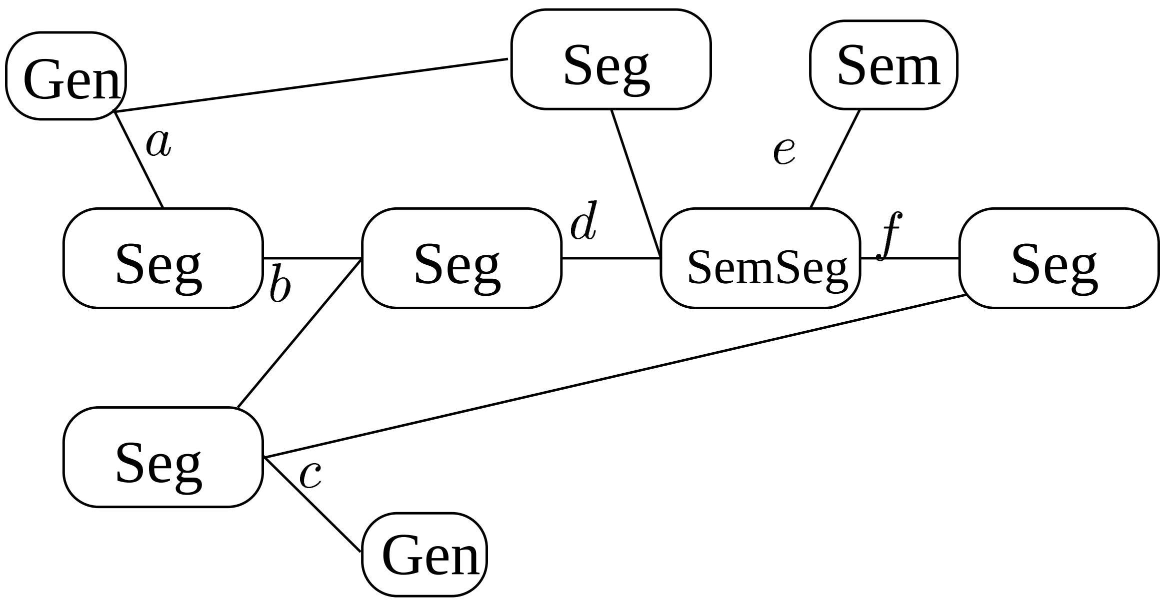
Example: a traffic network

$$
\begin{aligned}
Seg(in, out)(length) &= in?car \rightarrow car!length \\
&\rightarrow car? \rightarrow out!car \\
&\rightarrow Seg(in, out)(length)
\end{aligned}
$$

$$
\begin{aligned}
SemSeg(in, out, sem)(l) &= in?car \rightarrow car!l \rightarrow car? \\
&\rightarrow sem? \rightarrow out!car \\
&\rightarrow SemSeg(in, out, sem)(l)
\end{aligned}
$$

$$
\begin{aligned}
Gen(seg)(p) &= \textsf{new}\; car : (Car(car)(20) \\
&\qquad\qquad || \; seg!car \xrightarrow{p} Gen(seg)(p))
\end{aligned}
$$

Example: a traffic network



Example: a traffic network

$$
\begin{aligned}
Network \;=\; \textsf{new}\, a,b,c,d,e,f :\; & (Gen(a)(5) \\
& \,||\, Seg(a,b)(10) \\
& \,||\, Gen(c)(15) \\
& \,||\, Seg(c,b)(30) \\
& \,||\, Seg(b,d)(20) \\
& \,||\, Seg(a,d)(20) \\
& \,||\, SemSeg(d,f,e)(10) \\
& \,||\, Seg(f,c)(30) \\
& \,||\, Sem(e))
\end{aligned}
$$

Comparison of languages and formalisms

- DEVS and Statecharts: easy to model with timeouts & timed-prefix
- Timed Petri Nets: ?
- Timed automata: no multiple clocks
- LOTOS = CSP + ACT ONE, E-LOTOS = Timed CSP + ACT ONE
- Esterel: natural numbers as time-base; “counting" signals; deterministic
- CSP vs. CCS: multiway synchronization
- CSP, CCS vs ACP: non-blocking delay