Applied Concurrency Theory
Lecture 6 : real-time models

Hubert Garavel

Alexander Graf-Brill

Real-time problems

Real-time models

- In classical or probabilistic models, there is a notion of chronology between events, but no precise timing

- Examples:
  - one does not specify how long time will be spent in a state before firing a transition
  - one does not specify how long time it takes to fire a transition: is it instantaneous? does it take time?

- Real-time models address this issue
  - they carry more information than classical (untimed) models

Hard real-time

- Hard real-time systems must <u>always</u> react timely
  - ‘a correct output produced too late is a wrong output’
- No deviation from deadlines allowed
- Safety-critical systems often have hard real-time parts

Soft real-time

- Soft real-time systems must usually react timely

- There is some tolerance wrt deadlines
  - the system can be late from time to time

- Being late should remain exceptional:
  - otherwise the mission of the system is compromised
  - for instance, human users stop using it

- This leads to probabilistic analyses:
  - availability
  - reliability

Continuous-time Markov chains

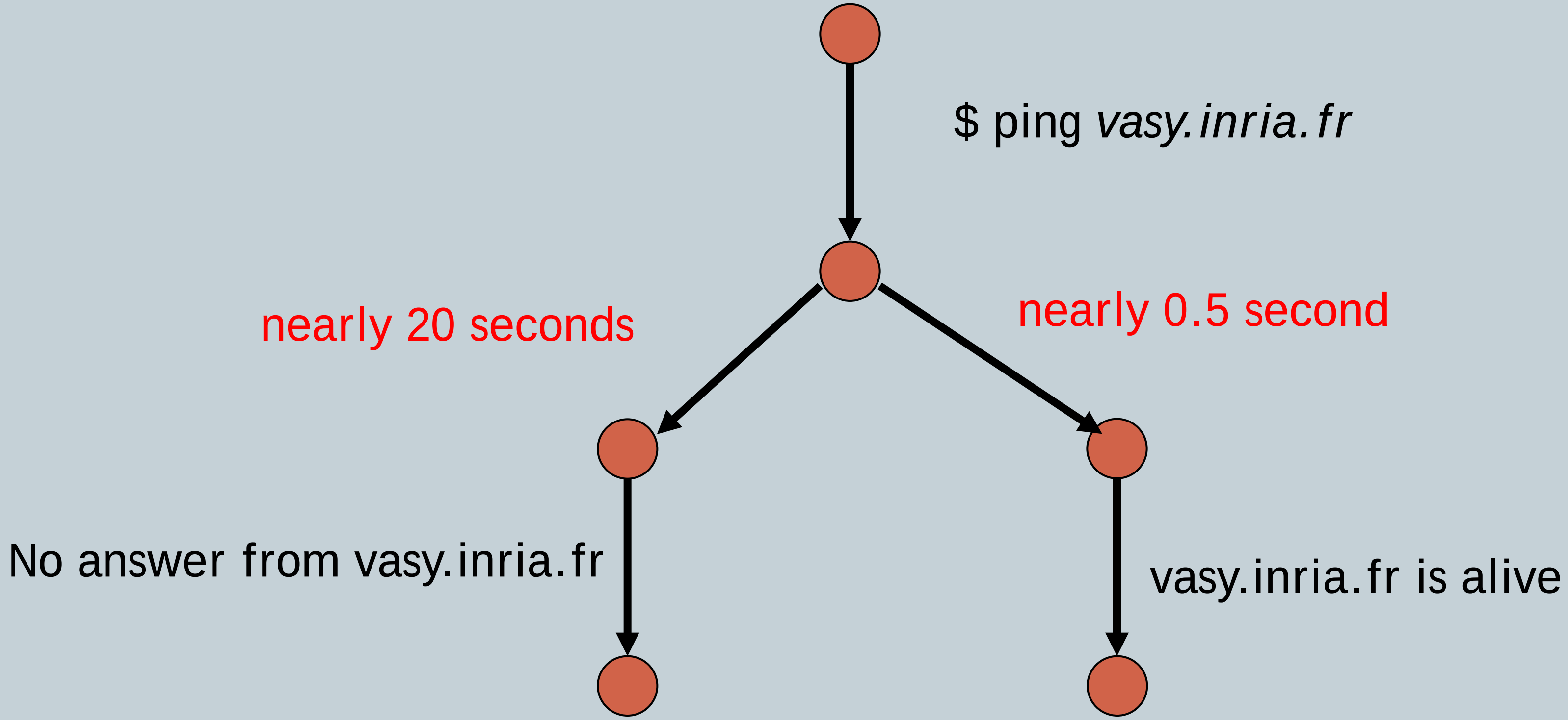
Examples of quantitative properties

- What is the probability of shutdown occurring within 4 hours?
- What is the long-run probability that 4 or more sensors are operational?
- What is the worst-case error probability over all possible initial configurations?
- What is the expected size of the message queue after 30 minutes?
- What is the worst-case expected time taken for the protocol to terminate?

(source: University of Birmingham)

Soft-real time example

The 'ping' command: answers takes some time



Continuous-Time Markov chains (CTMC)

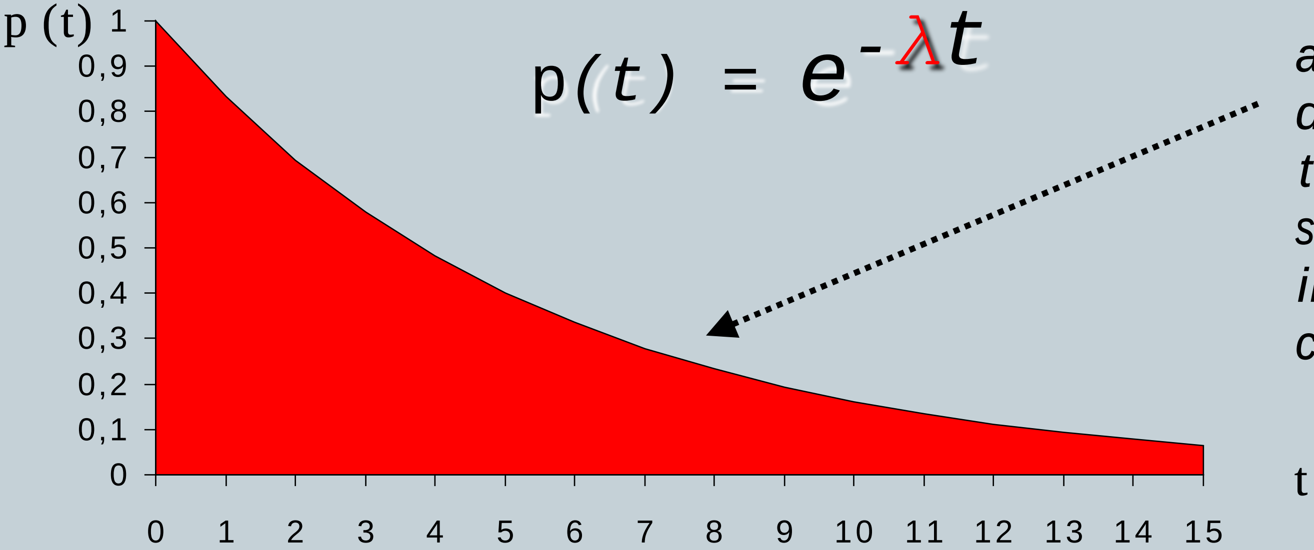
- In an automaton, transitions are discrete: at each time instant (e.g., clock 'tick'), the current state changes to another state

- In a DTMC, transitions are discrete too: at each time instant, the current probability distribution evolves from one set of states to another set of states

- In a CTMC, transitions are continuous: as time elapses, the probability distributions evolves progressively (no discrete clock ticks, but continuous passing of time)

Exponential distributions (time-homogeneous CTMCs)

p (t) : probability to be still in the same state(s) at time t



$$p(t) = e^{-\lambda t}$$

as time passes, p(t) decreases and the transition to the next state(s) becomes increasingly more certain

λ is a constant that expresses the mean rate of the exponential law (in terms of physical units, λ is a frequency, i.e., the inverse of a duration)

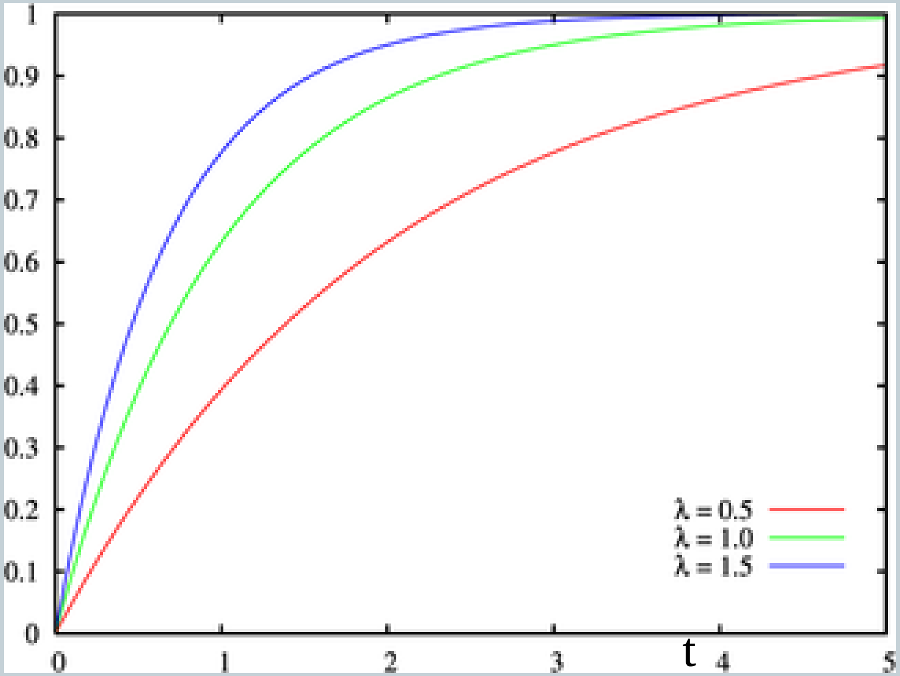
Influence of λ

The higher the value of **λ**, the faster the transition

The mean waiting time in the current state is 1 / **λ**

1 - p (t)

probability of having moved to the next state(s)



Why using exponential distributions? (1/4)

- Reason #1 (mathematical)

If a stochastic process $\{ X(t), t \geq 0 \}$ of state space S

- has the 'memoryless' Markov property (i.e., is a CTMC)

$$(\forall t_1, \ldots, t_n, t_{n+1} \mid 0 \leq t_1 \leq \ldots \leq t_n \leq t_{n+1})\ (\forall s_1, \ldots, s_n, s_{n+1} \in S)$$
$$P\{ X(t_{n+1}) = s_{n+1} \mid X(t_1) = s_1, \ldots, X(t_n) = s_n \} = P\{ X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n \}$$

- and is time-homogeneous

$$(\forall t, t' \mid 0 \leq t \leq t')\ (\forall s, s' \in S)$$
$$P\{ X(t') = s' \mid X(t) = s \} = P\{ X(t' - t) = s' \mid X(0) = s \}$$

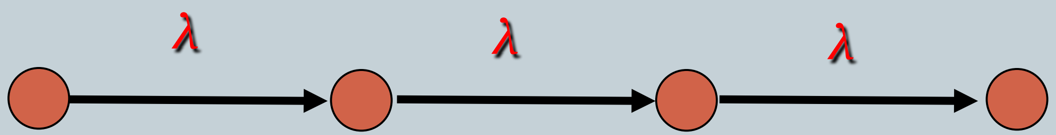
then it must follow an exponential distribution

Why using exponential distributions? (2/4)

- Reason #2 (mathematical)

Other 'useful' distributions can be expressed (exactly or arbitrarily closely) as a composition of exponential laws

Example: Erlang distributions are sequences of exponential law



- Reason #3 (pragmatic)

Exponential laws are convenient mathematical approximations enabling to do numerical computations efficiently and providing 'reasonable' results

Why using exponential distributions? (3/4)

- Reason #4 (intuitive):

  An exponential distribution with parameter ***λ*** models the time elapsed between successive events that:

  - are independent *(this condition is essential)*
  - occur randomly with a constant mean rate ***λ***

  Examples:

  - The duration between two successive clients entering a shop
  - The number of times a dice must be thrown to obtain a sequence of 10 consecutive ‘6’

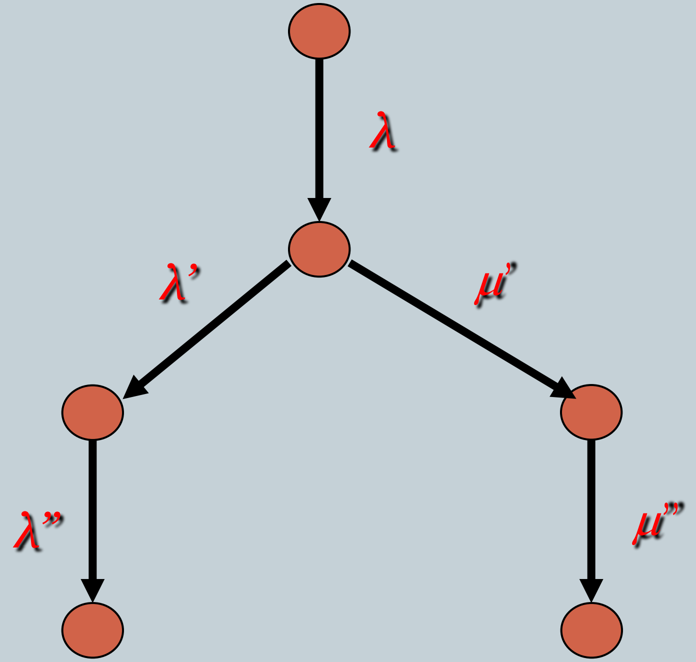
Why using exponential distributions? (4/4)

They describe the external behavior of systems whose internal structure is not entirely known

- natural phenomena
  - physics
  - chemistry
  - biology
- information theory (hidden Markov models)
  - data compression [Shannon] - entropy encoding
  - correction of transmission errors [Viterbi]
- computer science
  - pattern recognition
  - machine learning
  - Google's Pagerank algorithm

Graphical representation of CTMCs

CTMCs can be represented as (finite- or infinite-state) transition systems, in which the transitions are labelled with λ, μ, etc. (parameters of exponential laws)



Matrix representation (1/2)

- As for DTMCs, the current state of a CTMC can be represented by a probability vector V(t)
  - i-th element of V(t) : probability of being in state i at time t
  - contrary to DTMCs, t is continuous here, not discrete
- As for DTMCs, a CTMC with N states is represented by an NxN matrix Q (‘*generator matrix*’)
  - $i \neq j \Rightarrow Q[i, j]$ = rate $\lambda > 0$ of the transition from state i to state j, or zero if there is no such transition
  - $Q[i, i] = - \sum_{j \neq i} Q[i, j]$ // therefore $Q[i, i] \leq 0$
- Steady-state (i.e., long-run) probability vector V_∞ obtained by solving the equation ${}^{t}V_\infty \, . \, Q = 0$

Interactive Markov chains

18

Beyond CTMCs

- CTMCs are limited in the same way as DTMCs
  - mathematicians apply CTMCs to physical, chemical, etc. issues
  - they don't see the need for parallel composition
- We (computer scientists) want more:
  - we want to build systems with components
  - these components often run in parallel
  - we need action labels to synchronize components
  - we want message passing communication, not only shared variables
  - we want nondeterminism and tau-transitions

What would be a good extension of CTMCs?

- Many approaches proposed, but unsatisfactory
- What is a good solution?
  - 2 kinds of transitions: normal + rates, or mixed (normal, rate)
  - a parallel composition operator that matches the intuition
  - a parallel operator that is conservative
  - bisimulation relations to compare and minimize models
  - bisimulation relations that subsume lumpability:
    $\lambda;B \ [] \ \mu;B = (\lambda+\mu);B$
  - bisimulation relations 'compatible' with the parallel composition (compositionality, congruence)

IMC (Interactive Markov Chains)

- H. Hermanns PhD thesis (see References below)
- The IMC model
  - an LTS with additional rate transitions 'rate λ'
  - nondeterminism and taus are allowed
  - choice between ordinary and rate transitions is ok
- Parallel composition
  - same as in LOTOS
  - only constraint: no synchronization allowed on rate transitions
  - rates interleave: rate λ||rate μ = rate λ;rate μ [] rate μ;rate λ
- Stochastic (strong or branching) bisimulation
  - τ;B1 [] rate λ;B2 = τ;B1 (τ-transitions have priority)
  - λ;B [] μ;B = (λ+μ);B (lumpability)

Advantages of IMCs

- A very simple and elegant model
  - nice parallel composition
  - nice bisimulation relations
  - enables compositional state space generation

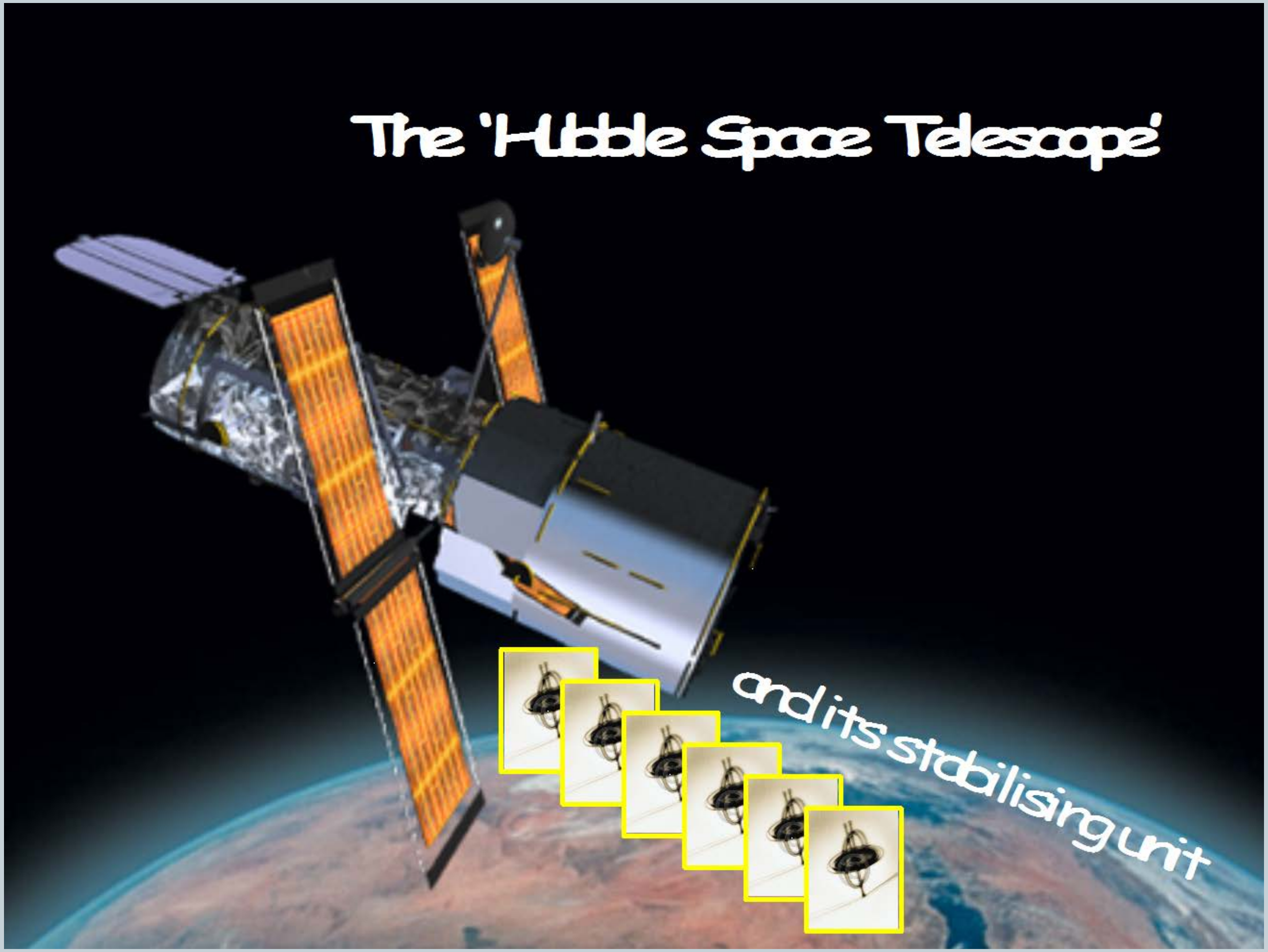
- Upward-compatible with standard process calculi
  - a superset of process calculi
  - a superset of the LTS model
  - existing tools do not have to be deeply modified

Available tools for IMCs

- CADP : the reference implementation
  - LOTOS state space generators unchanged
  - dedicated minimization tool (BCG_MIN with –rate option)
  - dedicated relabelling tools (BCG_LABELS)
  - parallel composition (EXP.OPEN with ‘-rate’ option)
- *IMCA – IMC Analyzer* (Univ. RWTH Aachen)
  - a new recent toolset
- PRISM
  - supports a parallel extension of CTMCs, but not IMCs
  - each transition seems to combine an action label and a rate

Application of IMCs: The Hubble space telescope

Example from H. Hermanns publications



A simple Markov model for Hubble

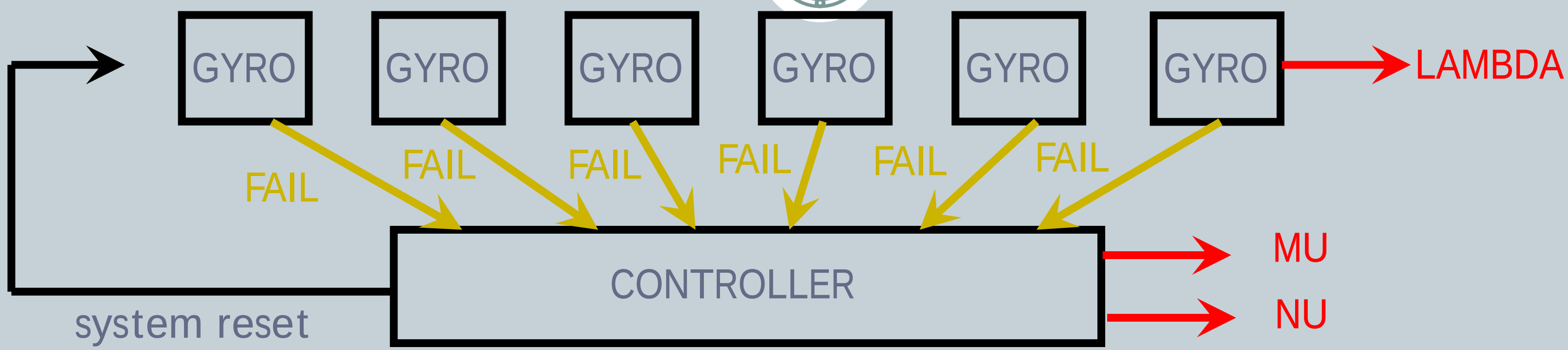
- The Huble telescope has 6 gyroscopes
- As time passes, the gyros may fail
- The average lifetime of gyros is 10 years (= 120 months)

  λ = *12 months / 120 = 0.1*
- Hubble falls into sleep if only two gyros are left
- Turning on sleep mode requires to halt all equipments, which takes about 3.6 days (= 0.12 month)

  μ = *12 months / 0.12 = 100*
- When in sleep mode, a shuttle mission must be sent to repair/reset Hubble, which takes about 2 months

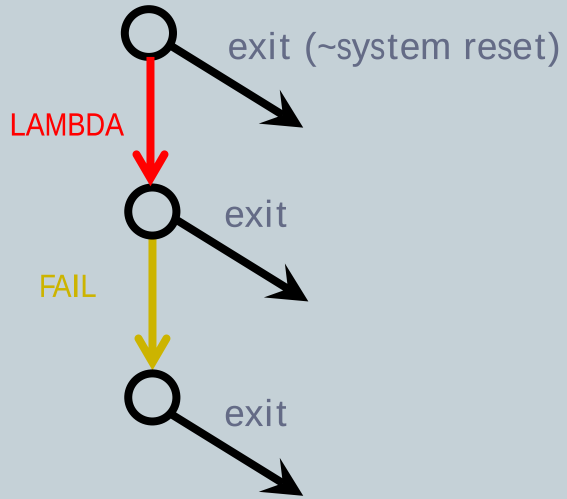
  ν =*12 months / 2 = 6*
- Without operational gyro, Hubble crashes

Compositional modelling of Hubble




```
process HUBBLE [LAMBDA, MU, NU] : noexit :=
    hide FAIL in
      (
          (
          GYRO [LAMBDA, FAIL] ||| GYRO [LAMBDA, FAIL] ||| GYRO [LAMBDA, FAIL] |||
          GYRO [LAMBDA, FAIL] ||| GYRO [LAMBDA, FAIL] ||| GYRO [LAMBDA, FAIL]
          )
        |[FAIL]|
        CONTROLLER [FAIL, MU, NU] (6, false)
        >> (* system reset *)
        HUBBLE [LAMBDA, MU, NU]
      )
endproc
```

The GYRO process

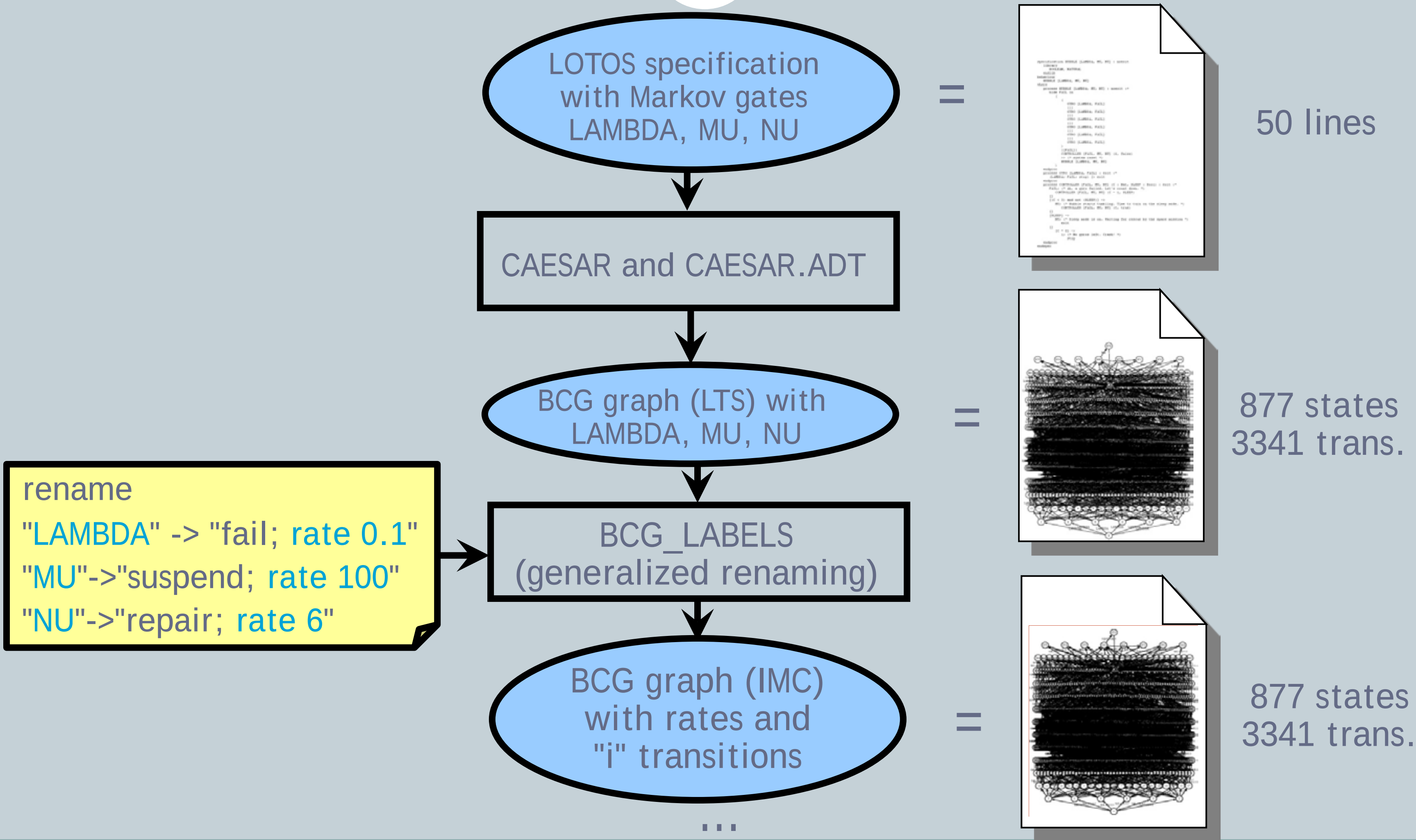


process GYRO [LAMBDA, FAIL] : exit :=
(LAMBDA; FAIL; stop) [> exit
endproc

The CONTROLLER process


```
process CONTROLLER [FAIL, MU, NU] (C : Nat, SLEEP : Bool) : exit :=
     FAIL; (* Ah, a gyro failed. Let's count down. *)
           CONTROLLER [FAIL, MU, NU] (C - 1, SLEEP)
     []
     [(C < 3) and not (SLEEP)] ->
          MU; (* Hubble starts tumbling. Time to turn on the sleep mode. *)
               CONTROLLER [FAIL, MU, NU] (C, true)
     []
     [SLEEP] ->
          NU; (* Sleep mode is on. Waiting for the space mission to reset Hubble. *)
               exit
     []
     [C = 0] ->
           i; (* No gyros left. Crash! *)
               stop
endproc
```

Analysis trajectory for the Hubble (1/2)



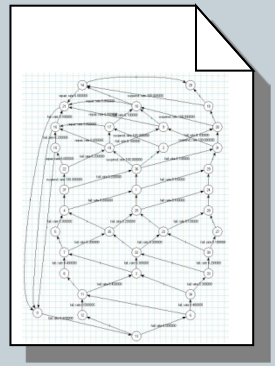
Analysis trajectory for the Hubble (2/2)

...

BCG_MIN (stochastic strong minimization)

BCG graph (IMC) with rate and "i" transitions

=

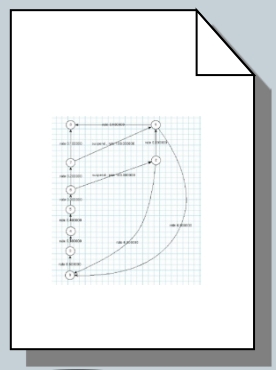


38 states
67 trans.

BCG_MIN (stochastic branching minimization)

BCG graph (CTMC with labels)

=



9 states
12 trans.

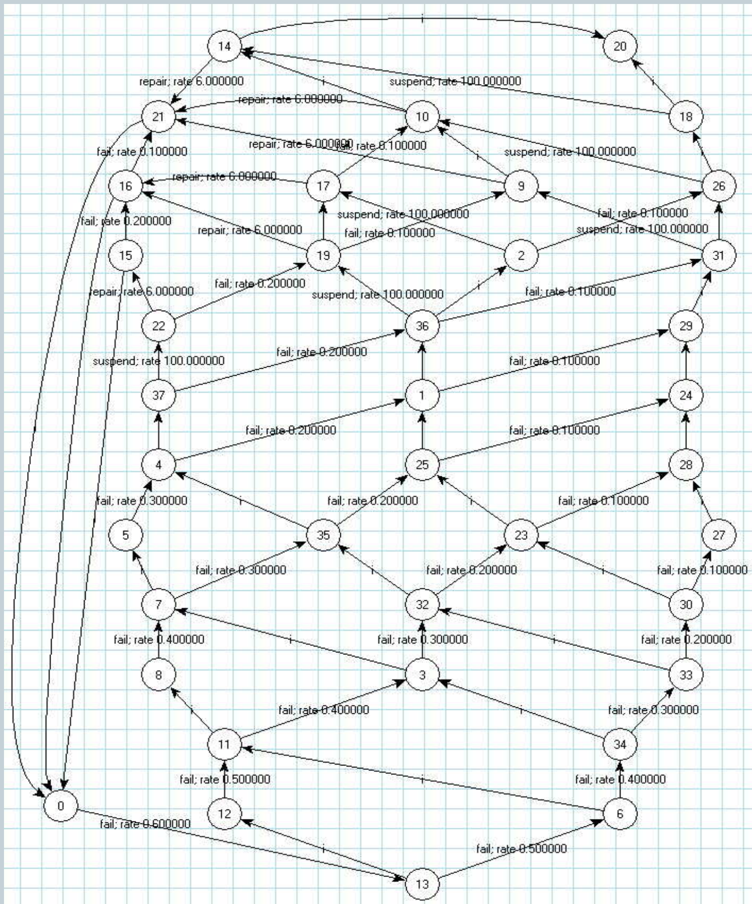
luckily, minimization, removed all "i" transitions (this is not always the case), so we get a CTMC (only rate-transitions), not an IMC, and we can do numerical CTMC analysis

BCG_TRANSIENT (transient analysis)

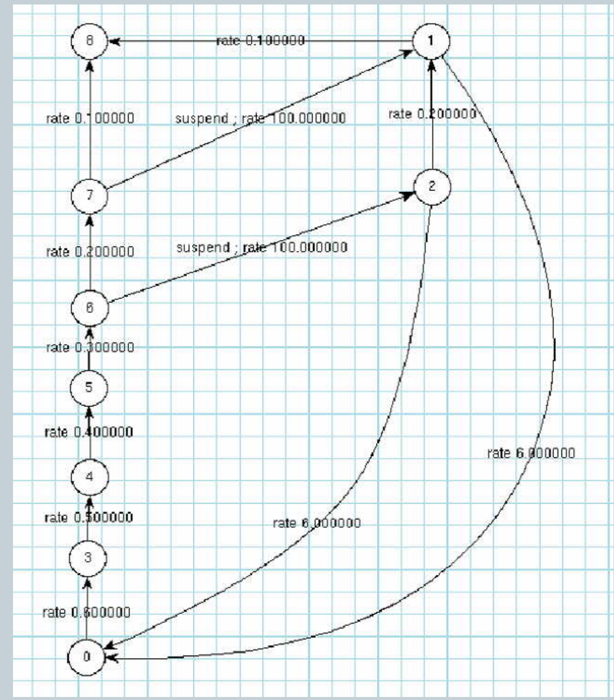
numerical data (probabilities)

Excel, gnuplot

Minimized IMCs for the Hubble

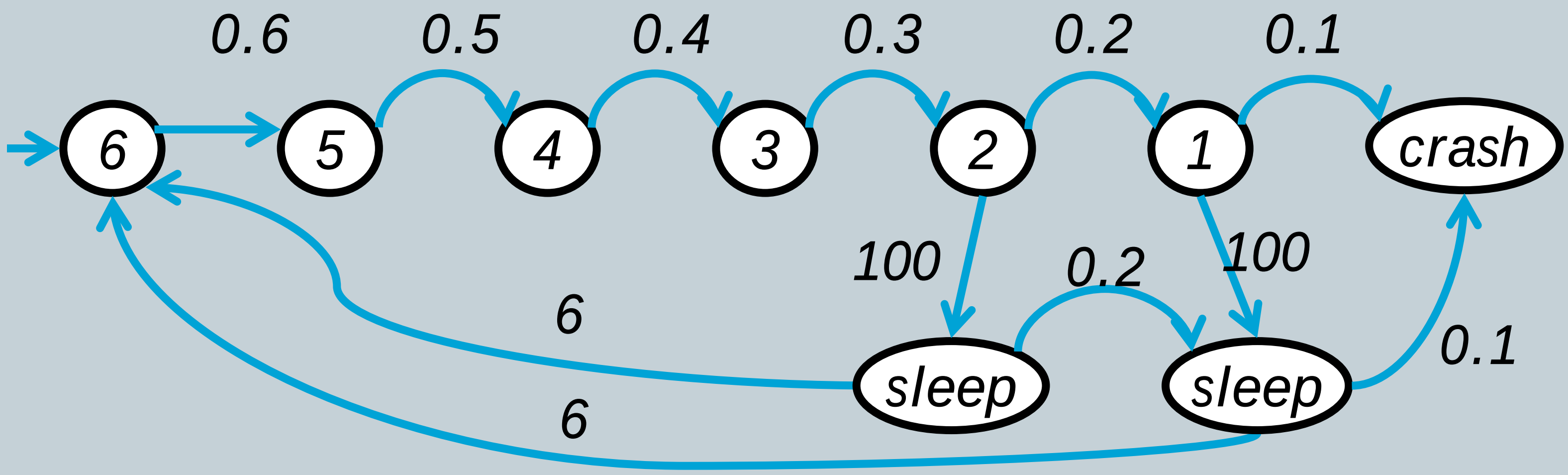


after stochastic strong minimization
(38 states, 67 transitions)



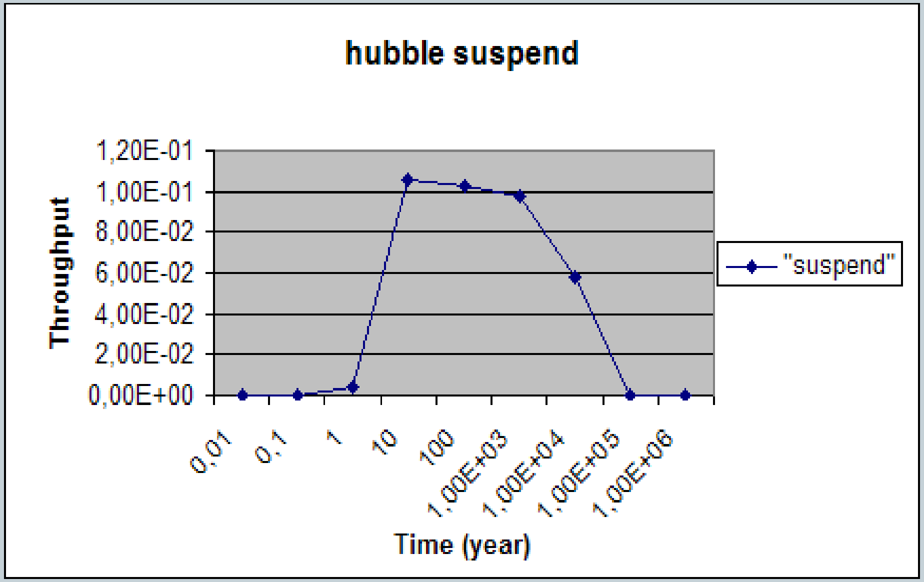
after stochastic branching minimization
(9 states, 12 transitions)

Visual verification of the final CTMC



Analysis of the Hubble using BCG_TRANSIENT

time	"repair"	"fail"	"suspend"
0.01	1.52E-11	0.5994	1.24E-09
0.1	5.45E-07	0.59403	4.34E-06
1	0.00248872	0.543138	0.00373419
10	0.105761	0.414947	0.105725
100	0.102729	0.414615	0.102786
1.00E+03	0.0974923	0.393478	0.097546
1.00E+04	0.0577739	0.233175	0.0578058
1.00E+05	0.00031195	0.00125902	0.00031212
1.00E+06	6.03E-27	2.43E-26	6.04E-27



Timed automata

Timed automata

- A theoretical model for specifying hard real time

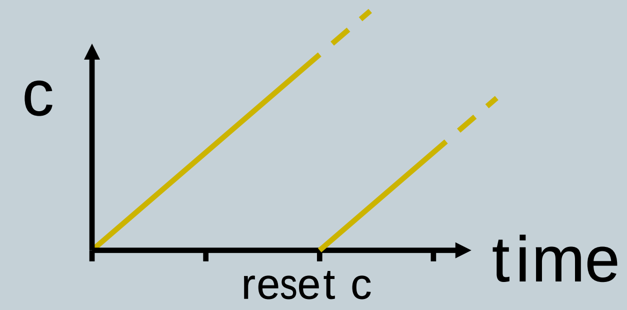
  R. Alur and D. Dill.
  A theory of timed automata.
  Theoretical Computer Science, 126:183-235, 1994.

- Implemented in many tools:
  - KRONOS (Grenoble) and UPPAAL (Uppsala and Aalborg)
  - PRISM, MODEST, etc.
  - popular model
- There exist alternative models
  - timed Petri nets
  - timed process calculi (Timed CCS, Timed CSP, ET-LOTOS, etc.)

Principles of timed automata

- Clocks: special variables to measure time
  - different from a central clock (e.g., POSIX time(2) function)
  - clocks are declared explicitly by the specifier
  - there may be several clocks
  - beware (too many clocks => undecidability!)
- Clocks increase linearly with rate 1 as time elapses
- One can only ‘reset’ clocks
  - but not assign them a non-zero value

clock c



Guards and invariants

- ‘when’ guards attached to transitions

  a transition cannot fire when its guard is false

  when (c >= 10) means:

  this transition may only be fired after 10 time units

- ‘invariant’ conditions attached to states

  one can remain in the state while the invariant is true

  when the invariant becomes false, one must leave urgently

  invariant (c <= 10) means:

  must quit this state before 10 time units

The MODEST toolset

many thanks to Holger Hermanns and Arnd Hartmanns

MODEST: the language

- MODEST: A Unified Language for Quantitative Models

- Combines:
  - process algebra constructs (LTS, nondeterminism)
  - probabilities (Markov Decision Processes)
  - time (Timed Automata)

- Formal semantics defined by SOS (Structural Operational Semantics) rules
- Suitable compositionality properties

Modest is more concise than Prism

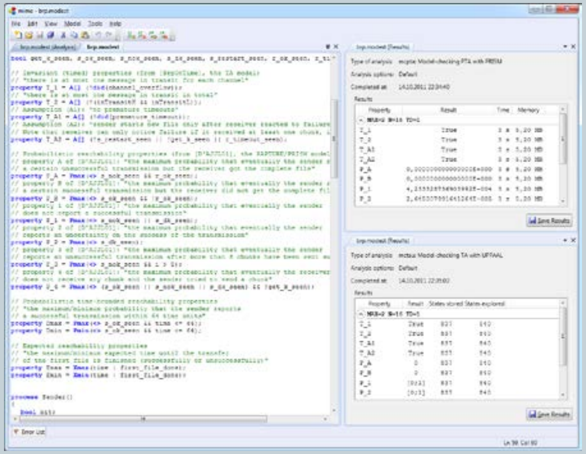
{= backoff = DiscreteUniform (0, pow (2, bc + 4) - 1) =}

MODEST code (1 line)

equivalent PRISM program

MODEST: the toolset

- A suite of tools developed at Saarland University
- Web site: http://www.modestchecker.net
- Currently, 5 tools:
  - mcpta: model checker for STA (uses Prism as a backend)
  - mctau: model checker for TA (uses Uppaal as a backend)
  - modes: discrete-event simulator for STA
  - mosta: visualisation using Graphviz
  - mime: graphical user-interface (Windows only)



The MODEST language

Data types in MODEST

- Basic data types
  - **bool**
  - **int**
  - **int** (min..max) // bounded integers (min and max are constants)
  - **real** // + 3 special types: **clock**, **reward**, **var**

- (Single-dimension) arrays
  - **int** [10]

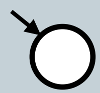
- Named records (a.k.a. 'structs')
  - **datatype** Point = {**real** X, **real** Y, **real** Z}

- Option types
  - **int option** // presumably : int $\cup$ {$\perp$}

Actions and sequential composition

- Inaction

  stop

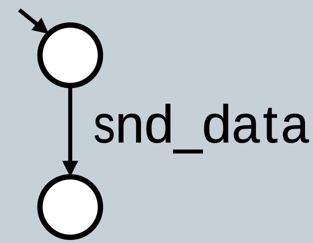
  or

  {==} **// null en LOTOS NT**

- Actions (visible or ‘tau’)

  action snd_data;

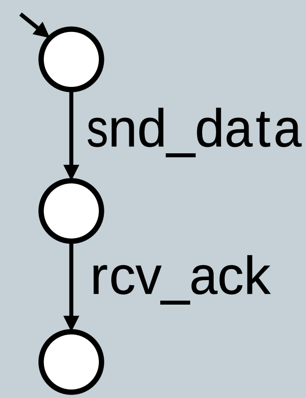
  snd_data

  - no data inputs/outputs (?/!)

- Sequential composition

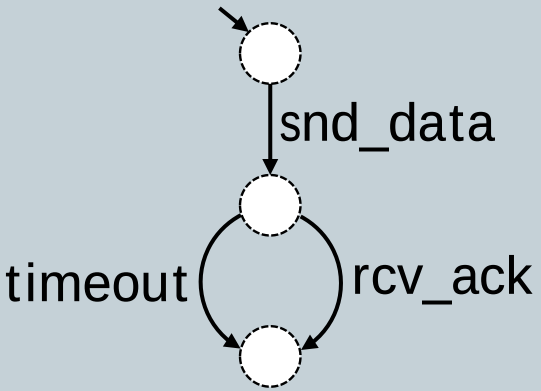
  snd_data ; rcv_ack



Choices and loops

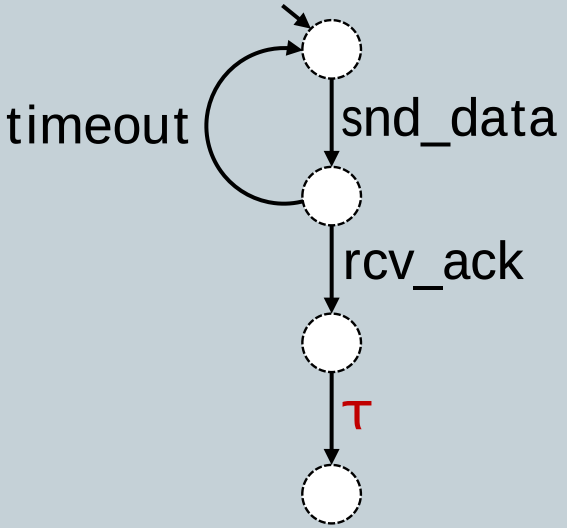
- Nondeterministic choice

```
snd_data ;
alt {
:: rcv_ack
:: timeout
}
```



- Loops and breaks

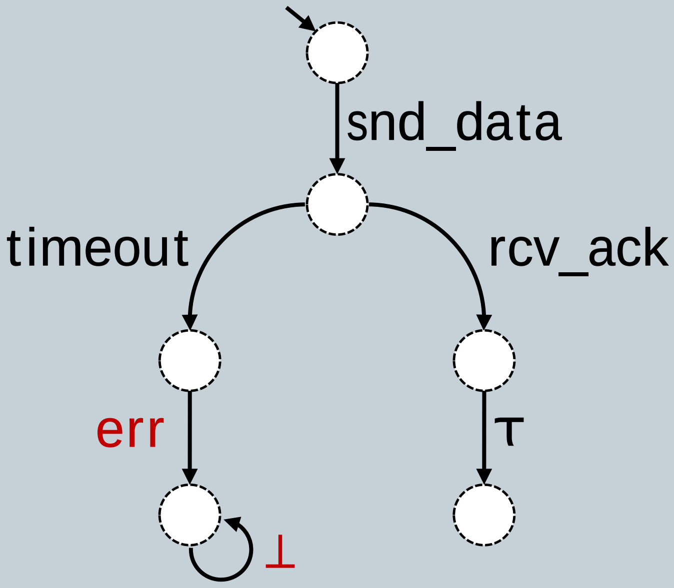
```
do {
:: snd_data ;
   alt {
   :: rcv_ack ; break
   :: timeout
   }
}
```



Exceptions

- Exception throwing

```
do {
:: snd_data;
   alt {
   :: rcv_ack; break
   :: timeout; throw (err)
   }
}
```

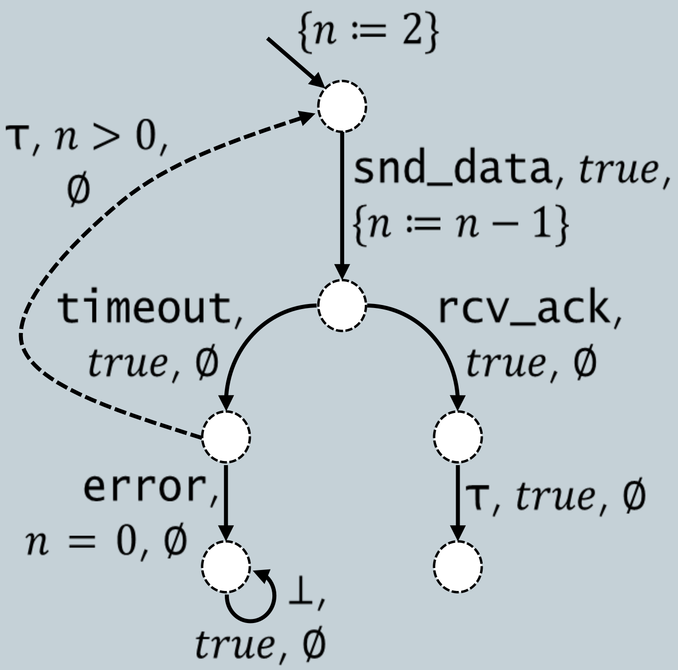


- There is a related ‘try … catch’ operator

Variables, assignments, and guards


```
int n = 2;

do {
:: snd_data {=  n=n-1  =} ;
   alt {
   :: rcv_ack ; break
   :: timeout ; alt {
      :: when (n > 0) tau
      :: when (n==0) throw (err)
      }
   }
}
```



Process and calls

- Process definitions
  - no gate parameters
  - value parameters are permitted

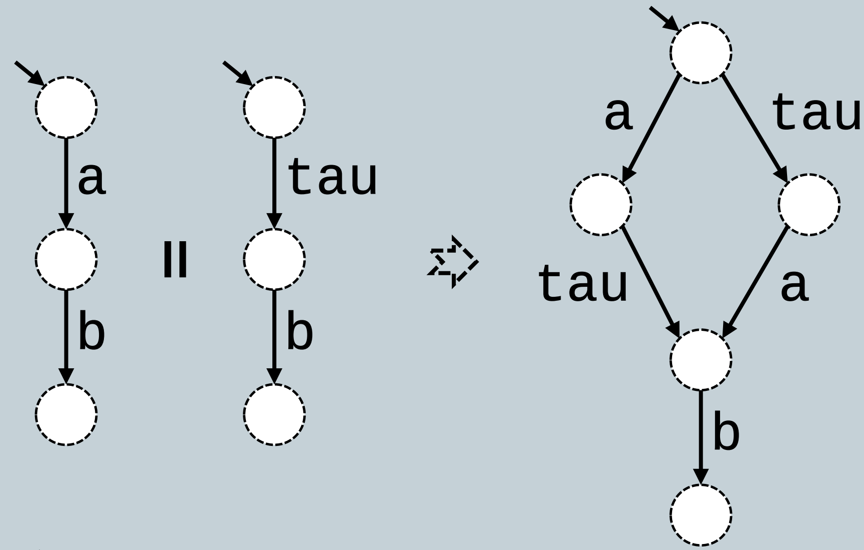
```
process Channel()
{
  snd ;
  alt {
  :: rcv
  :: timeout
  } ;
  Channel()
}
```

Parallel composition

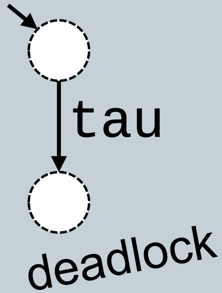
process P () { a ; b }

process Q () { tau ; b }

par {
:: P ()
:: Q ()
}

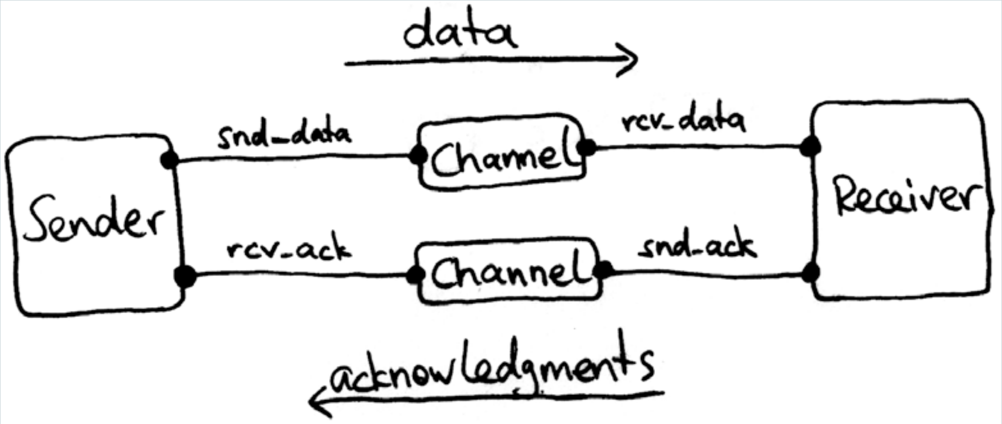


par {
:: P ()
:: Q ()
:: b ; a
}



Each process synchronizes only on its visible gates, which must be inferred by looking at the process body

Gate relabelling



par {
:: Sender ()
:: **relabel** {snd, rcv} **by** {snd_data, rcv_data} Channel ()
:: **relabel** {snd, rcv} **by** { snd_ack, rcv_ack } Channel ()
:: Receiver ()
}

where:
process Channel() {snd … rcv }

(MDP-like) probabilistic choice

- There is a 'palt' operator
- It follows the MDP strict alternation philosophy (first 'states', then 'nails')

- It is preceded by an action label (if absent: tau)

send palt {
  :p_1: ...
  :p_2: ...
  :$1-p_1-p_2$: ...
}

Timed automata primitives

- All the primitives of timed automata are there
- Declaration of clocks

  clock c ; // the time domain is dense (reals)

- Clock reset

  {= c = 0 **=}** // only value 0 allowed for clocks

- Clock constraint checking

  when (c >= 10) // restricted forms of conditions

- Invariants

  invariant (c <= 20) // restricted forms of conditions

Examples 1: simple time constraints

- Clock c must be reset at time 2 or later

```
when (c>=2) {= c=0 =}
```

- Clock c must be reset no later than time 2

```
invariant (c<=2) {= c=0 =}
```

- Clock c must be reset at time 2

```
invariant (c<=3) when (c>=2) {= c=0 =}
```

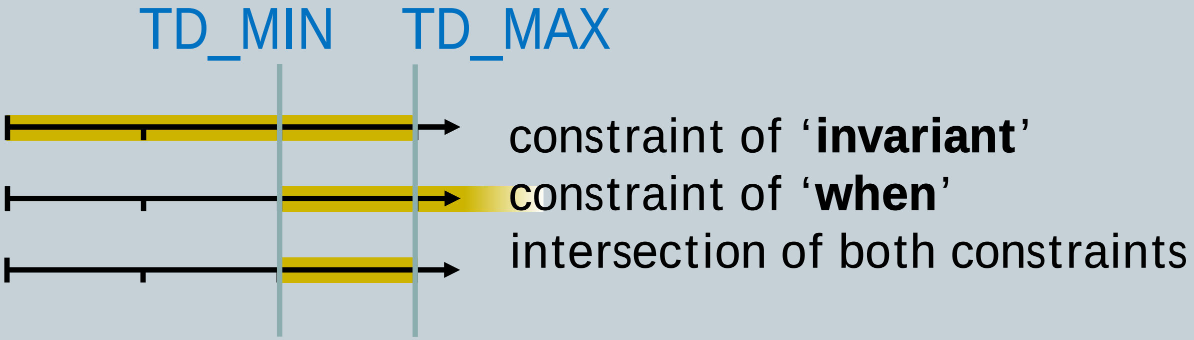
- Caution: in a choice, a ‘must’ might become a ‘may’

Example 2: time interval constraints

- A given action should take place after TD_MIN and before TD_MAX:

```
clock c;

{= c=0 =} ;
invariant (c<=TD_MAX)
when (c>=TD_MIN)
… // continue
```



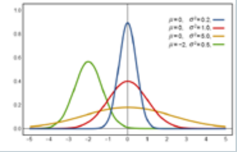
Example 3: (CTMC-like) ‘rate’ transitions

- A ‘rate λ’ transition is modelled using a clock
- Three steps:
  - select a random value x with a exponential distribution and reset the clock
  - time elapses - wait …
  - when the clock reaches x, resume the execution
- Specification in Modest:

(this involves probabilities, so the model is not, strictly speaking, a CTMC, but a STA)

```
clock c ;
real x ;
{=  x = Exp (λ) , c = 0  =} ;
when (c >= x) …      // continue
```

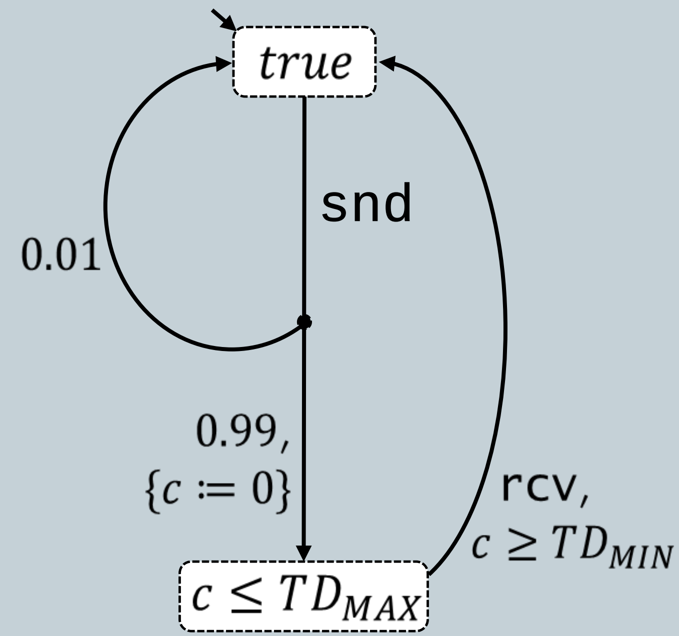
  - other distributions are supported: uniform, normal



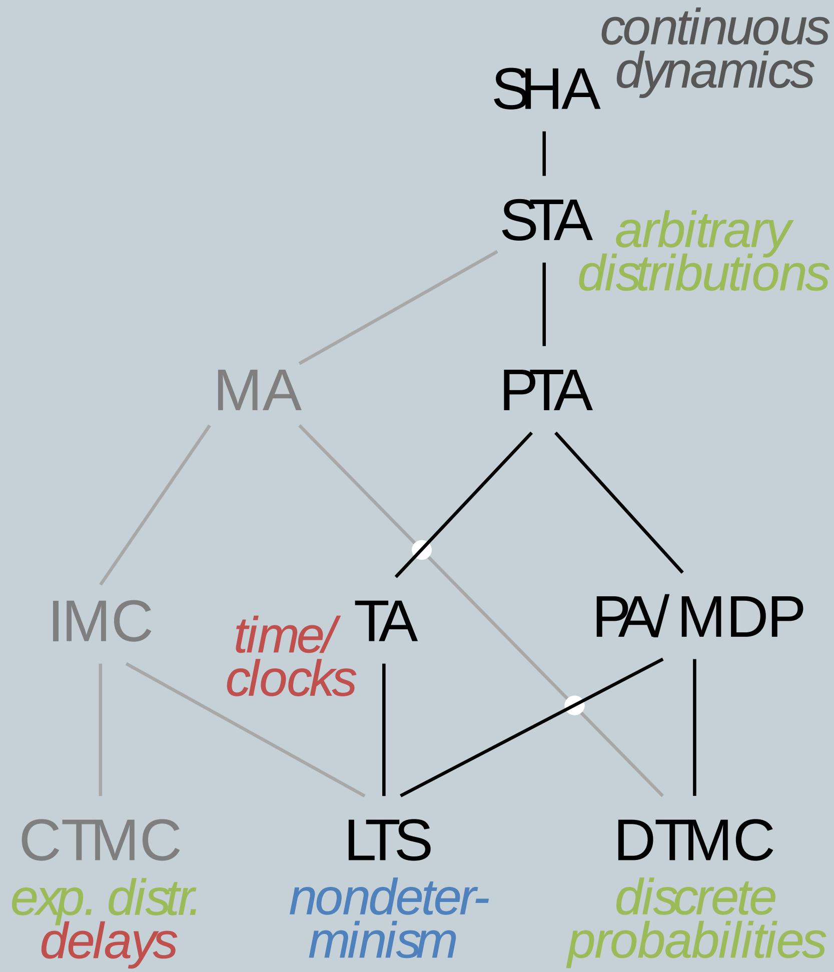
Example 4: probabilistic timed lossy channel

A lossy channel with 1% message loss probability and correct transmission delay in [TD_MIN, TD_MAX]

```
process Channel () {
  clock c;
  snd palt {
   :99: {= c=0 =};
        invariant (c<=TD_MAX)
        when (c>=TD_MIN) rcv
   : 1: {==} // do nothing
  } ;
  Channel ()
}
```



Modest expressiveness



Last challenge

From PRISM to MODEST

- Go back to the PRISM Manual v. 4.0.3
- Find the example of Probabilistic Timed Automaton given in this manual
- Translate this PTA in the Modest language
- Learn about the property specification language of Modest by visiting the case-studies page of the Modest web site
- Think of two properties that you would like to check on this PTA, express them, and check them using mcpta
- Send your files and results to Alexander

Conclusion

Final words...

- You should now be more familiar with those strange languages (CCS, LOTOS, LOTOS NT, pi-calculus, PRISM, MODEST) for concurrent systems

- None of these languages is perfect:
  - CCS, pi-calculus: mathematical notations rather than computer languages
  - LOTOS, LOTOS NT: no time – prob. and rates only with tricks
  - PRISM: very limited types only – too verbose
  - MODEST: limited types – thin documentation

  => this is still ongoing research

- But anyway, they are far better than programming languages (C++, Java) to study complex concurrent systems:
  - precise semantics
  - verification tools available
  - errors can be detected that could not be found by testing

References

CTMCs and IMCs

- CTMCs
  - Google: 'continuous time markov chains'
    gives dozens of mathematical tutorials on CTMCs

- IMCs
  - Holger Hermanns and Joost-Pieter Katoen.
    The How and Why of Interactive Markov Chains (2009)
    http://www-i2.informatik.rwth-aachen.de/imca/fmco09.pdf
  - Holger Hermanns.
    Interactive Markov Chains (book, 2002)
    http://www.springer.com/computer/swe/book/978-3-540-44261-5

MODEST language and tool

- MODEST Web site
  - http://www.modestchecker.net/
- MODEST syntax reference (2012)
  - http://www.modestchecker.net/Documentation/Modest%20Syntax%20Reference.pdf
- Presentation of the MODEST language (Sept. 2012)
  - Arnd Hartmann. MODEST – A Unified Language for quantitative models http://www.modestchecker.net/Link.aspx?id=pub:H12
- Overview of the MODEST language (2004)
  - H. Bohnenkamp, P. d'Argenio, H. Hermanns, J.-P. Katoen. MoDeST: A compositional modeling formalism for hard and softly timed systems. http://doc.utwente.nl/48984/1/0000011b.pdf