Applied Concurrency Theory
Lecture 6: real-time models
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 systems must always 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 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)
The ‘ping’ command: answers takes some time

$ ping vasy.inria.fr

nearly 0.5 second  nearly 20 seconds

No answer from vasy.inria.fr

vasy.inria.fr is alive
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)

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

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

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

As time passes, \( p(t) \) decreases and the transition to the next state(s) becomes increasingly more certain.
Influence of $\lambda$

The higher the value of $\lambda$, the faster the transition.

The mean waiting time in the current state is $1 / \lambda$.
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)
  \[
  (\forall t_1, \ldots, t_n, t_{n+1} \mid 0 \leq t_1 \leq \ldots \leq t_n \leq t_{n+1} \quad (\forall s_1, \ldots, s_n, s_{n+1} \in S)
  \]
  \[
  P \{ X(t_{n+1}) = s_{n+1} \mid X(t_1) = s_1, \ldots, X(t_n) = s_n \} =
  P \{ X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n \}
  \]

- and is time-homogeneous
  \[
  (\forall t, t' \mid 0 \leq t \leq t') \quad (\forall s, s' \in S)
  \]
  \[
  P \{ X(t') = s' \mid X(t) = s \} = P \{ X(t' - t) = s' \mid X(0) = s \}
  \]

then it must follow an exponential distribution
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.
Reason #4 (intuitive):

An exponential distribution with parameter $\lambda$ models the time elapsed between successive events that:

- are independent  \textit{(this condition is essential)}
- occur randomly with a constant mean rate $\lambda$

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’
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
CTMCs can be represented as (finite- or infinite-state) transition systems, in which the transitions are labelled with $\lambda$, $\mu$, etc. (parameters of exponential laws)
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 \( N \times N \) 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 \( V_\infty \) obtained by solving the equation \( tV_\infty . Q = 0 \)
Interactive Markov chains
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:
    \[ \lambda;B \parallel \mu;B = (\lambda+\mu);B \]
  - bisimulation relations ‘compatible’ with the parallel composition (compositionality, congruence)
H. Hermanns PhD thesis (see References below)

**The IMC model**
- an LTS with additional rate transitions ‘rate $\lambda$’
- 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 $\lambda | |rate \mu = rate \lambda;rate \mu [[]] rate \mu;rate \lambda$

**Stochastic (strong or branching) bisimulation**
- $\tau;B1 [[]] rate \lambda;B2 = \tau;B1$ ($\tau$-transitions have priority)
- $\lambda;B [[]] \mu;B = (\lambda+\mu);B$ (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

The 'Hubble Space Telescope'

and its stabilising unit
The Huble telescope has 6 gyroscopes
As time passes, the gyros may fail
The average lifetime of gyros is 10 years (120 months)
\[ \lambda = \frac{12 \text{ months}}{120} = 0.1 \]

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)
\[ \mu = \frac{12 \text{ months}}{0.12} = 100 \]

When in sleep mode, a shuttle mission must be sent to repair/reset Hubble, which takes about 2 months
\[ \nu = \frac{12 \text{ months}}{2} = 6 \]
Without operational gyro, Hubble crashes
process HUBBLE [LAMBDA, MU, NU] : noexit :=
    hide FAIL in
    (
    (
    )
    | [FAIL] |
    CONTROLLER [FAIL, MU, NU] (6, false)
    >> (* system reset *)
    HUBBLE [LAMBDA, MU, NU]
)
The GYRO process

The GYRO process is defined as follows:

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)

LOTOS specification with Markov gates LAMBDA, MU, NU

CAESAR and CAESAR.ADT

BCG graph (LTS) with LAMBDA, MU, NU

rename
"LAMBDA" -> "fail; rate 0.1"
"MU"->"suspend; rate 100"
"NU"->"repair; rate 6"

BCG_LABELS (generalized renaming)

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

...
Analysis trajectory for the Hubble (2/2)

BCG_MIN (stochastic strong minimization) →
BCG graph (IMC) with rate and "i" transitions
BCG_MIN (stochastic branching minimization) →
BCG graph (CTMC with labels)

38 states 67 trans. =
9 states 12 trans. =

BCG_TRANSIENT (transient analysis) →
numerical data (probabilities) → Excel, gnuplot

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
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

<table>
<thead>
<tr>
<th>time</th>
<th>&quot;repair&quot;</th>
<th>&quot;fail&quot;</th>
<th>&quot;suspend&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1.52E-11</td>
<td>0.5994</td>
<td>1.24E-09</td>
</tr>
<tr>
<td>0.1</td>
<td>5.45E-07</td>
<td>0.59403</td>
<td>4.34E-06</td>
</tr>
<tr>
<td>1</td>
<td>0.00248872</td>
<td>0.543138</td>
<td>0.00373419</td>
</tr>
<tr>
<td>10</td>
<td>0.105761</td>
<td>0.414947</td>
<td>0.105725</td>
</tr>
<tr>
<td>100</td>
<td>0.102729</td>
<td>0.414615</td>
<td>0.102786</td>
</tr>
<tr>
<td>1.00E+03</td>
<td>0.0974923</td>
<td>0.393478</td>
<td>0.097546</td>
</tr>
<tr>
<td>1.00E+04</td>
<td>0.0577739</td>
<td>0.233175</td>
<td>0.0578058</td>
</tr>
<tr>
<td>1.00E+05</td>
<td>0.00031195</td>
<td>0.00125902</td>
<td>0.00031212</td>
</tr>
<tr>
<td>1.00E+06</td>
<td>6.03E-27</td>
<td>2.43E-26</td>
<td>6.04E-27</td>
</tr>
</tbody>
</table>
Timed automata
Timed automata

- A theoretical model for specifying hard real time
  R. Alur and D. Dill.
  *A theory of timed automata.*

- 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.)
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
Guards and invariants

- ‘when’ guards attached to transitions
  a transition cannot fire when its guard is false

  \[\text{when } (c \geq 10) \text{ means:}\]
  \[\text{this transition may only be fired after 10 time units}\]

- ‘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

  \[\text{invariant } (c \leq 10) \text{ means:}\]
  \[\text{must quit this state before 10 time units}\]
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

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

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
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 ∪ {⊥}`
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
Nondeterministic choice

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

Loops and breaks

```
do {  
:: snd_data ;  
altn {  
:: rcv_ack ; break  
:: timeout  
}  
}
```
Exception throwing

\[
\begin{align*}
do \{ \\
:: \text{snd\_data}; \\
\text{alt} \{ \\
:: \text{rcv\_ack}; \textbf{break} \\
:: \text{timeout}; \textbf{throw} (\text{err}) \\
\} \\
\}
\end{align*}
\]

There is a related ‘\texttt{try ... catch}’ operator
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

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

\[ \text{par \{ } \right. \\
\text{:: Sender ()} \\
\text{:: \textbf{relabel} \{snd, rcv\} by \{snd\_data, rcv\_data\} Channel ()} \\
\text{:: \textbf{relabel} \{snd, rcv\} by \{snd\_ack, rcv\_ack\} Channel ()} \\
\text{:: Receiver ()} \\
\text{\}} \]

\textbf{where:}

\textbf{process} Channel() \{ snd \ldots rcv \}
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 {
  :p1: ...
  :p2: ...
  :1-p1-p2: ...
  :1-p1-p2: ...
}
```
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
  \[\text{when } (c \geq 2) \{ = c = 0 = \}\]

- Clock c must be reset no later than time 2
  \[\text{invariant } (c \leq 2) \{ = c = 0 = \}\]

- Clock c must be reset at time 2
  \[\text{invariant } (c \leq 3) \text{ when } (c \geq 2) \{ = c = 0 = \}\]

- Caution: in a choice, a ‘must’ might become a ‘may’
A given action should take place after \text{TD\_MIN} and before \text{TD\_MAX}:

```plaintext
clock c;
{= c = 0 =};
invariant (c <= \text{TD\_MAX})
when (c >= \text{TD\_MIN})
... // continue
```

\textbf{TD\_MIN} \hspace{1cm} \textbf{TD\_MAX}

constraint of ‘invariant’

constraint of ‘when’

intersection of both constraints
Example 3: (CTMC-like) ‘rate’ transitions

■ A ‘rate $\lambda$’ 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:

```plaintext
clock c;
real x;
{= x = Exp ($\lambda$) , c = 0 =} ;
when (c >= x) ... // continue
```

(other distributions are supported: uniform, normal)

(this involves probabilities, so the model is not, strictly speaking, a CTMC, but a STA)
Example 4: probabilistic timed lossy channel

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

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

Lecture 6

- SHA
- STA
- PTA
- MA
- IMC
- CTMC
- LTS
- DTMC
- PA/MDP
- TA

- continuous dynamics
- arbitrary distributions
- discrete probabilities
- nondeterminism
- exp. distr. delays
- time/clocks

- exp. distr.
delays
- nondeterminism
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 mcptaprint
- Send your files and results to Alexander
Conclusion
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
    [http://www-i2.informatik.rwth-aachen.de/imca/fmco09.pdf](http://www-i2.informatik.rwth-aachen.de/imca/fmco09.pdf)
MODEST language and tool

- **MODEST Web site**
  - [http://www.modestchecker.net/](http://www.modestchecker.net/)

- **MODEST syntax reference (2012)**
  - [http://www.modestchecker.net/Documentation/Modest%20Syntax%20Reference.pdf](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

- **Overview of the MODEST language (2004)**
    - [http://doc.utwente.nl/48984/1/0000011b.pdf](http://doc.utwente.nl/48984/1/0000011b.pdf)