Timed Systems: The Unmet Challenge

Oded Maler

CNRS - VERIMAG
Grenoble, France

QAPL, April 2014

A Concrete Motivation for an Abstract Talk

- You want to run an application represented as a task graph on a new multi-core platform
- Tasks are annotated by execution times and communication rates
- In addition to compilation you need to:
- Map tasks to processors, schedule them, allocate buffers and channels, select data transfer mechanisms
- These **deployment** problems are difficult and can have serious consequences on performance
- We don't want application programmers to deal with them and want to provide automatic support
- Timed systems give, in principle, the conceptual and mathematical framework to handle such problems

What is the Message of this Talk?

- Models of **timed systems** are extremely important
- They represent a level of abstraction which underlies almost any domain of engineering and daily life
- In particular, they are useful for performance analysis and optimization of embedded systems (and systems in general)
- Unfortunately, sociological factors, both in academia and industry, as well as complexity problems, prevent this potential from being fully realized
- We are doing our best to change this situation
- Paper appears in **From Programs to Systems**, LNCS 8415

Outline

1. **The timed level of abstraction in modeling**
2. Timed automata and the heavy burden of the formal methodist
3. Strange encounters with reality

Levels of Abstraction: Low

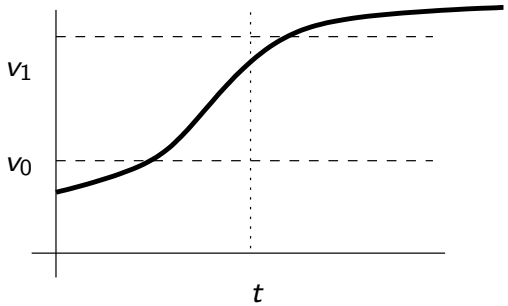
- Phenomena can be modeled at different levels of abstraction
- Lower levels are more detailed, zoomed at more "elementary" entities and are supposed to be closer to God's reality
- This is, at least what reductionists (physicists, molecular biologists) want us to believe
- The price of more detailed models is :
- It is hard to build them and to measure initial/boundary conditions
- It takes much more **computation time** to analyze (simulate, verify) them

Levels of Abstraction: High

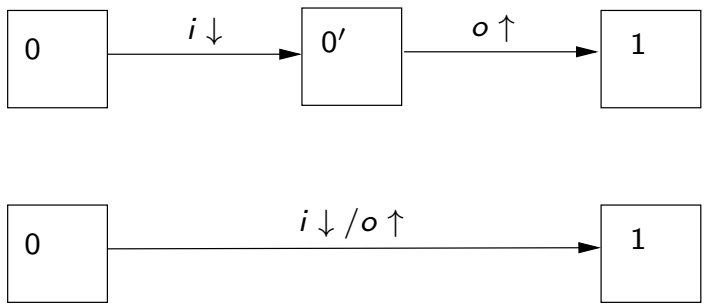
- High level models are more coarse
- They use concepts that, in principle, could be derived as aggregation/abstraction of lower-levels entities
- But more often than not, only in principle
- Example: in Civil Engineering, the resistance of a beam to different loads (module of elasticity) is **not** derived from a detailed models of zillions of interacting molecules
- Remark: software is exceptional in the sense of having a formal equivalence between several levels, (eg compilation)

Concrete Example: Transistors and Gates

- At a lower-level a logical gate, say inverter, is an electrical circuit whose voltage at the output port depends on the voltage in its input
- Its behavior is a signal, a trajectory of a continuous dynamical system
- At an abstract Boolean level we say that when the input goes down the output takes a transition from 0 to 1



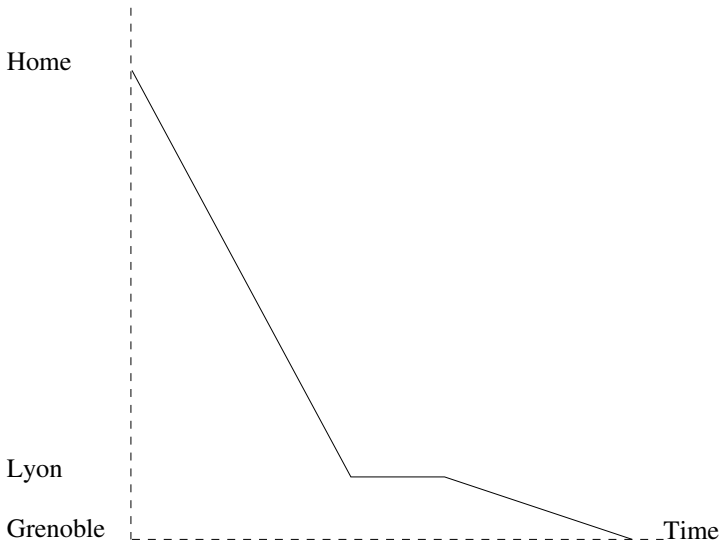
Continuous

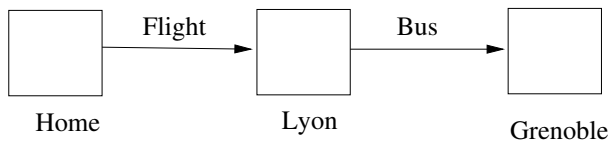


Discrete

Concrete Example: Coming to Grenoble

- Coming to Grenoble from your hometown via Lyon airport
- Low level description: a trajectory of the center of mass of the person on the spatial earth coordinates
- High level: fly to Lyon than take bus to Grenoble (sequence of transitions)





Concrete Example: Software

- Low level: a piece of code that transforms some input to some output using instructions that run on some hardware platform
- High level description: decode or filter an image

```
for i=1 to 1024 do
  ...
  ...
  something
  ...
  ...
end
```

process an image

Which Information is Sufficient?

- From the more abstract discrete point of view you have:
- Some kind of a (physical) process that you do not care so much about its **intermediate** details
- Unless you are an airplane pilot or an electron or a programmer
- What is important is that at the end of the day?:
  - You will be in Grenoble
  - The gate will switch from 0 to 1 following Boole-Shannon rules
  - The image will be decoded
- This is "functional" reasoning

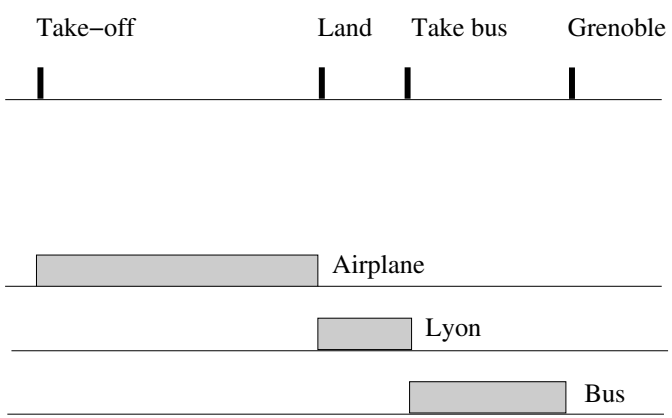
You Cannot Get Rid Completely of the Physics

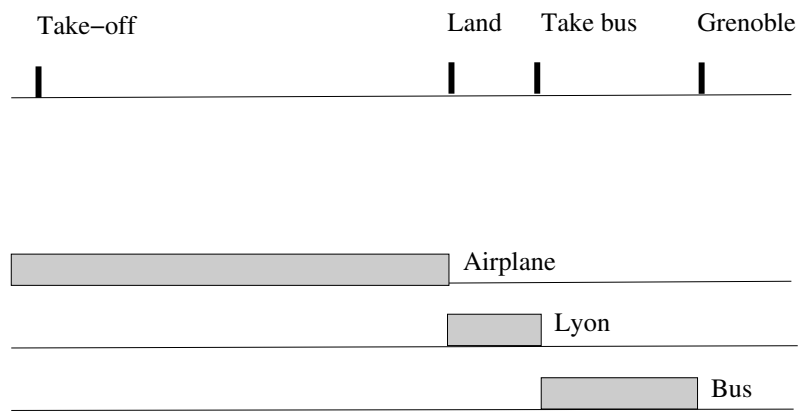
- To determine the clock rate your computer can use, you need to know **how long** it takes to switch from 0 to 1
- To see UTube on your smart phone you care about the **execution time** of your decoding algorithm
- To come on time to the conference you need to know the **duration** of the flight
- The purely discrete automaton model does not distinguish between flying from Paris and flying from San Francisco
- It is an abstract sequence of transitions:

  take-off $\rightarrow$ fly $\rightarrow$ land

Timed Behaviors

- Hide intermediate values of the process but represent:
- The distance between **events** (threshold crossing, starting, stopping) or
- The **durations** of sojourn in states



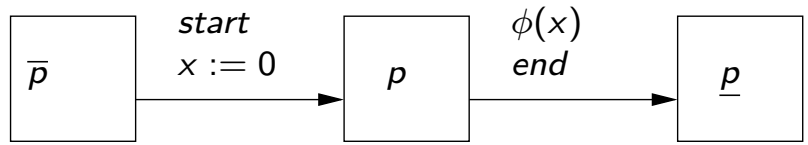


Timed Dynamical Systems

- A new intermediate class of dynamical systems, between:
- Models based on **differential equations**: continuous behaviors (trajectories, signals)
- Models based on **automata**: discrete behaviors, sequences of state/events
- Timed automata are the **dynamical systems** for timed behaviors
- They generate discrete-valued signals or time-event sequences (= sequences of time-stamped events)

Basic Elements: Processes that Take Time

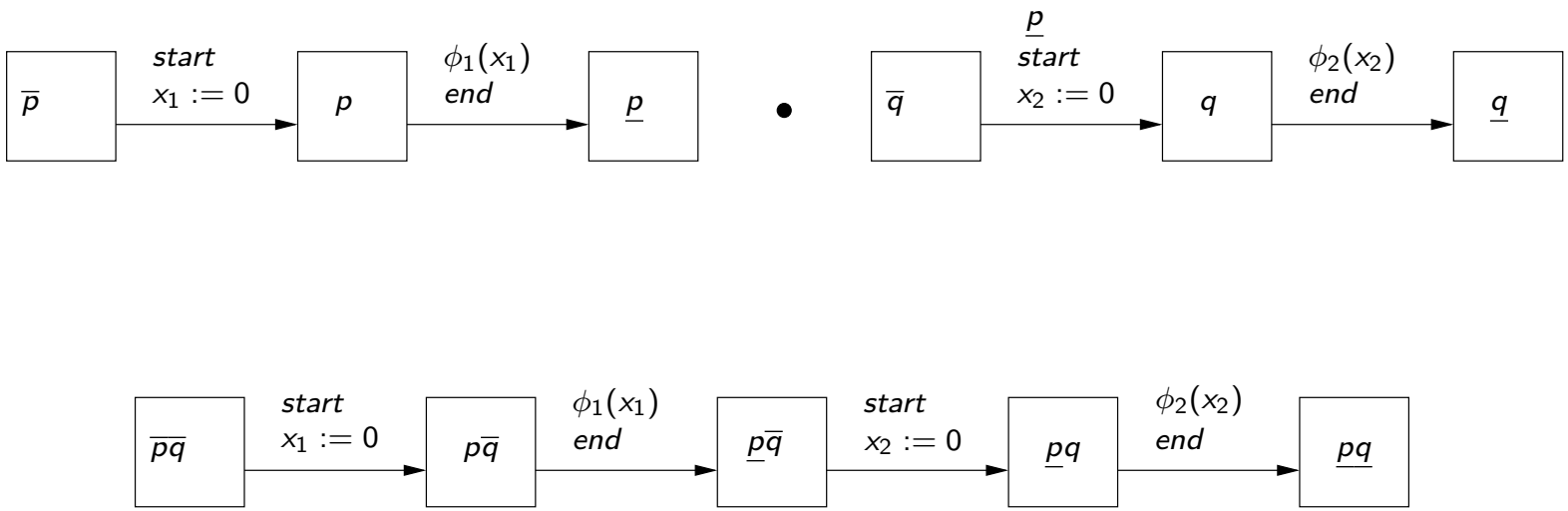
- Processes that take some time to conclude after having started
- Mathematically they are simple timed automata:



- An **idle** state $\overline{p}$; a **start** transition which resets a clock x to measure time elapsed in **active** state p
- An **end** transition guarded by a temporal condition $\phi(x)$
- Condition ϕ can be **true** (no constraint), $x = d$ (deterministic), $x \in [a, b]$ (non-deterministic) or probabilistic

Sequential Composition

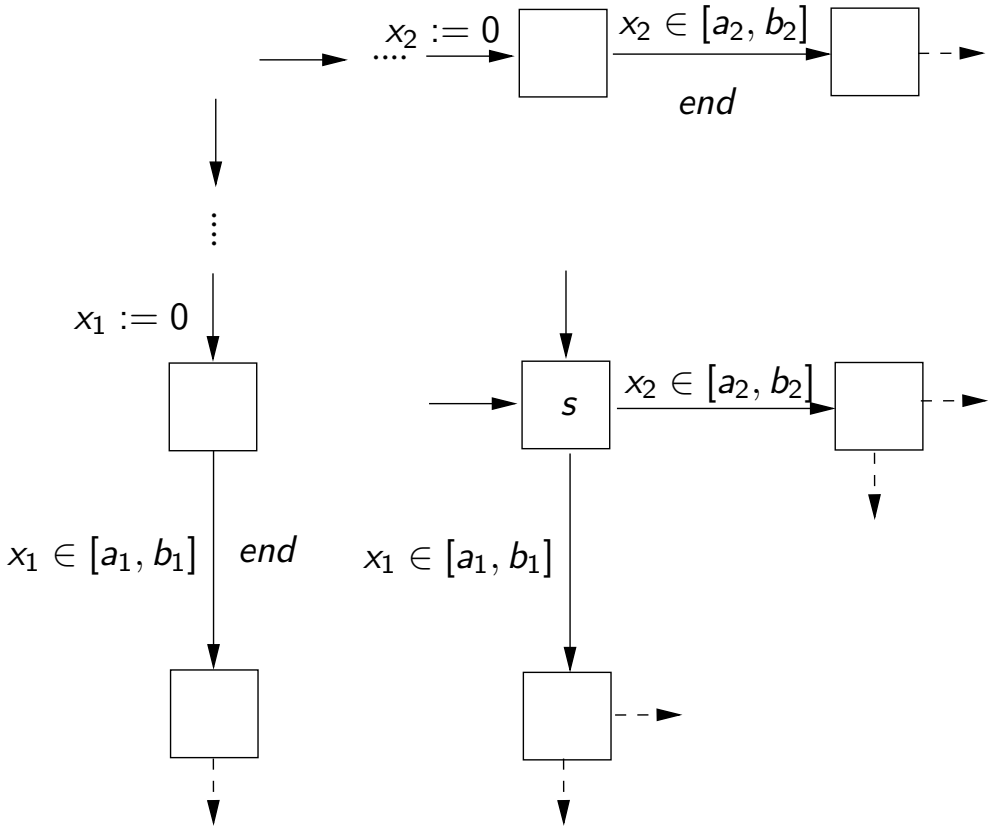
- Sequential composition captures precedence relations between tasks, for example p precedes q:



- You take the bus after you land
- A gate switching triggers a change in the next gate
- You can start processing the image after having decoded it

Parallel Composition

- Parallel composition models partially-independent processes, sometimes competing with each other (race)



- Analyzing the possible behaviors of such concurrent timed processes is at the heart of almost anything we do

Questions

- Will there be a glitch in the circuit?
- Will he finish his boring talk by the coffee break?
- Will the meal be ready exactly when the guests arrive?
- Will my student finish the thesis before I run out of money?
- Will the image be processed before the arrival of the next one?
- Will the server answer the query before the attention span of the client expires?
-
- All these are questions about possible paths in timed automata

Intermediate Summary

- I hope by now you are convinced that timed systems are important for **modeling**
- You can formulate with them all kinds of interesting questions in an important level of abstraction
- It is the level of abstraction that people use implicitly in scheduling, timing analysis, planning - you name it
- Now remains the question, how can you **use** these models to provide **answers** to these questions
- To answer this question, let us have a retrospective look at "formal verification" our home discipline

Outline

1. The timed level of abstraction in modeling
2. **Timed automata and the heavy burden of formal methodist**
3. Strange encounters with reality

What is Verification?

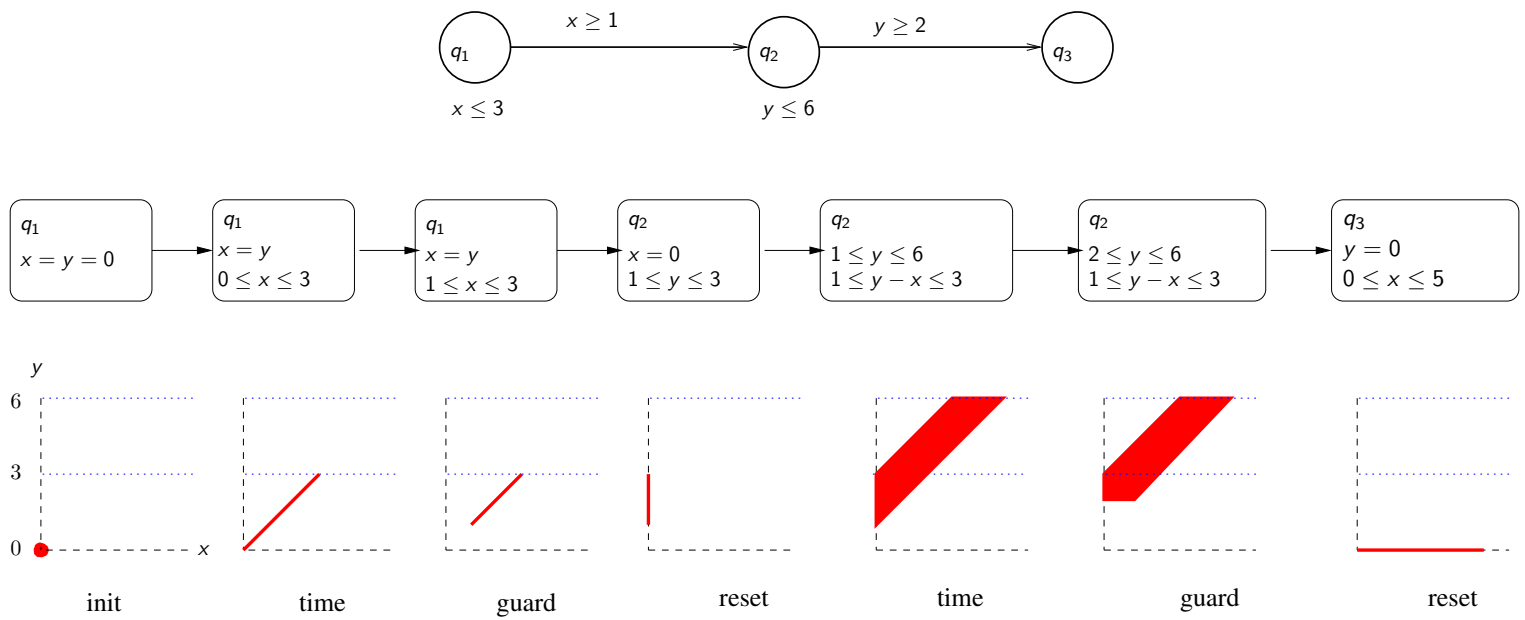
- Most of verification is about showing that components in a network of automata interact properly with each other
- Some sequences of events are considered ok while others violate the requirements, bad things happen (safety) or some good things do not (liveness)
- Models are discrete and often abstract away from the data and focus on control/synchronization
- The systems in question are open and under-determined
- This means that a model may have **many** executions, some correct and some incorrect
- Verification is a kind of exhaustive simulation that explores all paths in a huge automaton

Extending Verification to Timed Systems

- In addition to the non-determinism associated with external discrete actions
- There is also **dense** temporal non-determinism
- We do not know execution times, propagation delays, inter-arrival times and process durations with precision
- We model them typically using bounded intervals
- Following the safety-critical spirit of verification, we attempt to reason **universally** about this uncertainty space
- Compute **all** possible behaviors under all choices of duration values

Timed Verification Tools I

- Construct the reachability graph in an extended state-space which includes the clocks values
- Due to the dense non-determinism one has to treat **sets** of clock valuations (similar to hybrid systems)



- These sets constitute a restricted type of polyhedra called **zones** represented as DBMs

Timed Verification Tools II

- Verification algorithms developed in a series of theses and tools at Verimag under the guidance of Joseph Sifakis
- Kronos, Open-Kronos, IF: Sergio Yovine, Alfredo Olivero, Conrado Daws, Stavros Tripakis, Marius Bozga
- The most celebrated and actively maintained tool these days is UPPAAL (Uppsala and Aalborg), started with Wang Yi, Paul Pettersen, and Kim Larsen
- Continued under Kim's guidance with major contributions by Gerd Behrman and Alexandre David
- The only problem is that this approach rarely scales up beyond toy problems
- It is also PSPACE-hard

Fighting the Clock Explosion

- I spent around ten years of my life in trying to scale up
- Using numerical decision diagrams that give a unified symbolic representation for discrete states and clocks (CAV 97)
- Using timed polyhedra that provide a canonical representation for non-convex sets (ICALP 00)
- Using heuristic search to solve scheduling problems (CAV 01)
- Using bounded model-checking via SMT solvers (FTRTFT 02)
- The last heroic attempt was **Compositional Timing Analysis** with Ramzi Ben Salah and Marius Bozga, EMSOFT 2009
- Was all this a waste of time?

On the Sociology of Science

- Before giving a hopefully negative answer let me reflect a bit on the state of science
- Ideally one would like to apply noble first class science and mathematics to solve real problems
- Formal Language Theory and Compilation, Information Theory and Telecommunication, Number Theory and Cryptography
- We accept **good** mathematics for its own sake and
- Technological gadgets produced by people who do not formulate themselves in a clean mathematical way
- However, we should be careful not to commit the double sin of doing **mediocre** mathematics over **marginal** questions under the excuse of **imagined** applications
- This seems to be unavoidable in the current state of affairs and the structure of scientific communities (and industry)

Outline

1. The timed level of abstraction in modeling
2. Timed automata and the heavy burden of the formal methodist
3. **Strange encounters with reality**

Getting Real

- Context: in the ATHOLE project we promised to help solving one of the most pressing questions in informatics:
- How to deploy efficiently application programs on parallel multi-core architectures
- The self-confidence was based partly on our knowledge of timed automata, scheduling, SMT solvers, etc.
- The lessons learned were of many sorts, I will mention some
- I am sure most of it is known to many people but each person discovers things in his own way and order

The Practical and Theoretical Difference between Theory and Practice

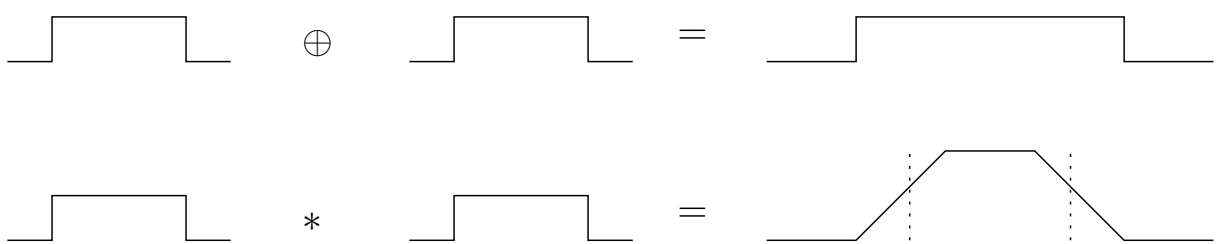
- The theoretician has the liberty to choose the problems and to ignore aspects that are outside the scope of his interest and capabilities
- The **real** practitioner does not have this choice, his deadlines are not self-imposed and his time is measured
- The theoretician solves **general** problems: verification applies, in principle, to all automata, all temporal logic formulae, etc.
- The partitioner solves one problem at a time
- Consequently the real-life scope of a theoretical solution is any number of problems in $[0, \infty)$: it is 0 if the compromises with reality were too violent, and infinity if they were clever
- A theoretician observation: $[0, \infty)$ and 1 are not comparable

From Correctness to Performance

- Correctness is a Boolean **performance measure**
- A performance measure is a way to associate cost/utility with individual system behaviors and with the system as a whole
- We can measure elapsed time, associate costs with states and transitions and accumulate them along runs
- We need not necessarily Booleanize them via inequalities such as deadline conditions - we can remain **quantitative**
- We should provide real numbers (and vectors of real numbers in multi-criteria) as answers
- Many people will agree on that and performance evaluation is a major issue in the embedded world

Who Needs Universal Quantification?

- Because of safety-criticality or cost-criticality (hardware errors) verification always aspired to cover **all** possible points in the under-determination space
- In other words, a pessimistic worst-case approach
- This is, at the same time, too much and too little for most systems (soft real-time, best effort, mixed criticality)
- Too much because if the worst-case is rare we can live with it (as in daily life)
- Too little because we really want to know what will typically happen, not only what is possible in principle
- Solution: replace measureless duration bounds by distributions
- From Minkowski sum to convolution:



What to do with these Duration Probabilistic Automata?

- Probabilizing the timing uncertainty does not alleviate the scalability problem
- Computing probabilities over sets is harder than computing the sets themselves
- One direction to think about: **fat first search**, exploring only reachable sets of high probability
- The other solution: do random Monte-Carlo simulation, sample the uncertainty space and collect statistics
- Then call it **statistical model checking** to hide the fact that after 20 years we resort finally to what practitioners have always been doing...

One of my Favorite Quotations

- Kurt Vonnegut in Cat's Cradle says:
- *Beware of the man who works hard to learn something, learns it, and finds himself no wiser than before... He is full of murderous resentment of people who are ignorant without having come by their ignorance the hard way*
- If we replace exhaustive verification by Monte Carlo simulation what was the worth of the exhaustive verification episode?
- Well, there are still systems which are critical and require exhaustivity, especially wrt discrete under-determination
- Newcomers to any domain can bring fresh insights
- The other answer: **abstract modeling** with clear behavioral **semantics** and distinction between different types of non-determinism has some potential contribution

Between System Builders and Model Builders I

- System builders use formalisms such as C or Verilog to build their systems
- This coding is unavoidable if you want the system to be built; You **must** write this code
- Abstract models used in verification or performance analysis are considered by them as an extra burden
- By the way, I can understand them: I don't want anyone to tell me how to annotate my LaTeX code or use UML to structure my research
- Building **abstract** models, not corresponding to something **concretely executable** requires capabilities that many system builders do not have

Between System Builders and Model Builders II

- Consequently they use their designs as the models for simulation: the program (or circuit) models itself
- When the hardware and software already exists it is the most efficient way to evaluate performance and correctness
- But in stages of design-space exploration when the hardware architecture or configuration is not realized
- The software is run on a hardware simulator at some low granularity (cycle accurate...) and this is expensive

Between System Builders and Model Builders III

- If you want to explore different configurations, task mappings, scheduling policies, buffer sizing
- Simulation at this level might be too slow
- It is much more efficient to use a discrete event simulator in timed automaton style
- Computations and communications are modeled as timed processes that immobilize some resources for some duration
- Of course, you need to fill in the numbers (profiling, estimation, past experience) but you need **not** be precise and deterministic

The Design-Space Explorer I

- A prototype tool developed in the thesis of JF Kempf (2012) with the help of Marius Bozga and Olivier Lebeltel
- It has four components:
- **Application description**: task-graphs annotated with execution times and data transfer quantities
- **Input generators**: model of task arrivals (periodic, jitter, delayed periodic, bounded variability)
- **Architecture description**: processors and their speeds, memories, busses
- **Deployment**: mapping and scheduling

The Design-Space Explorer II

- We convert these descriptions into timed automata (in IF) representing all possible behaviors under all choices from the timing uncertainty space
- We analyze them using formal verification (when possible and useful) and mostly via statistical simulation
- Time will tell whether such techniques will find their way to the design flow of MPSoCs and/or the compilation and deployment chain
- More details will be given tomorrow by JF Kempf

Conclusions

- Timed automata are the best invention since cut-and-paste
- Although they are n-tuples they can be useful
- Not only for the paper industry or the tool-paper industry
- This requires more blood sweat and tears, less theorem hunting and less incremental bibliometric activities
- It is Time for a break