

# Compositional Timing Analysis

Ramzi Ben Salah    Marius Bozga    Oded Maler

CNRS - VERIMAG  
Grenoble, France

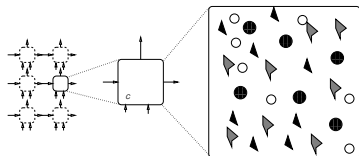
2009

# Apology

- ▶ The message of this paper is not easy to communicate
- ▶ It represents many years of work (theory and implementation)
- ▶ It is based on a very intuitive ideas concerning a fundamental problem in hierarchical system design
- ▶ But technically it consists of a series of transformations on timed automata which are hard to follow
- ▶ Even for the authors
- ▶ I will do my best

# A Motivating and Challenging Example

- ▶ Consider a living cell which at some level of abstraction can be viewed as a soup where zillions of complex molecules move and interact
- ▶ At this levels one can analyze, for example, the effect of injecting some new molecule on the dynamics of concentrations of the other molecules
- ▶ When we analyze a tissue consisting of many such cells, it is impractical to compose many detailed cell models
- ▶ At the higher level we want a simpler model where a cell is a module exchanging some signals with its neighbors



## Less Fascinating but Still Interesting Motivations

- ▶ Low-level physical transistor model is abstracted into a model with a relatively-small number of state variables used in a higher level model of..
- ▶ A transistor level circuit which is, in turn, abstracted into a gate or standard cell model which is used in a higher level model of..
- ▶ A digital circuit which realizes some micro-architecture element ...
- ▶ ...
- ▶ Hardware, software, internet, world, universe...

# Principles of Hierarchical Design/Analysis Methodology

- ▶ Complex systems are made of subsystems (components, modules)
- ▶ These subsystems, in turn, are made of subsystems
- ▶ ...
- ▶ At a given level of abstraction a component admits a model  $M$  with some level of granularity
- ▶ Moving to the next higher level where the component is composed with other components, we would like to replace  $M$  by a more abstract model  $M'$

## A Wish List for the Reduced Model $M'$

- ▶ It should be much less complex than  $M$  (less state variables, simplified dynamics)
  - ▶ Otherwise the analysis of the higher-level system will explode
  - ▶ To achieve that,  $M'$  should abstract away from many internal details
  - ▶ Consequently  $M'$  is less precise than  $M$
- ▶ On the other hand  $M'$  should be sufficiently faithful to the interface behavior of  $M$ 
  - ▶ So that if we substitute  $M'$  for  $M$  in the high-level model the reduced model will not deviate much from the detailed one
- ▶ It is desirable to derive  $M'$  from  $M$  automatically

# On the Semantics of being Less Precise

- ▶ Two different approaches to relate the concrete and abstract models:
- ▶ Metric based (physics, traditional engineering):
  - ▶ Model reduction: a system of differential equations with  $n$  variables is replaced by a system with  $m < n$  variables
  - ▶ Underlying this approach is the notion that the observable trajectories of  $M'$  are close to those of  $M$
- ▶ Set theoretic non determinism (CS, verification):
  - ▶ Since in  $M'$  some variables are projected away, the system becomes more non deterministic and admits more behaviors than  $M$
  - ▶ This is expressed by the inclusion of (observed) behaviors  $L(M) \subseteq L(M')$
  - ▶ This means that whatever you prove (correctness or worst-case performance) using  $M'$  holds as well for  $M$
- ▶ This is the approach that we use

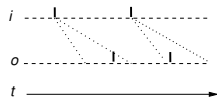
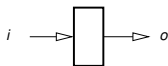
## More Technically Speaking

- ▶ We propose a fully-automated and tool-supported methodology for deriving  $M'$  from  $M$  for the case of networks of timed components
- ▶  $M$  is a product of timed automata representing an acyclic network of timed components; It has one clock per component
- ▶  $M$  realizes a (non-deterministic) timed transducer, it maps timed input behaviors to sets of timed output behaviors
- ▶  $M'$  is a timed automaton with less states and less clocks
- ▶  $M'$  over-approximates  $M$  as a timed transducer: for any input, the set of outputs  $M'$  produces includes all the outputs produced by  $M$



# Timed Components

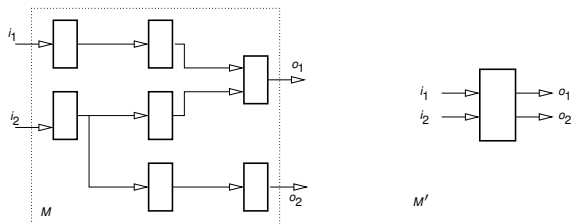
- ▶ A timed component is a device that reacts to a timed stream of input events by emitting a timed stream of output events
- ▶ Each output event is emitted some time after the input event that triggered it



- ▶ Timed components can model:
  - ▶ Execution time of a software module
  - ▶ Propagation delay in a digital circuit
  - ▶ Time to transmit a packet in a network
  - ▶ Time to respond to web query

# (Acyclic) Networks of Timed Components

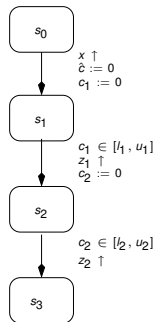
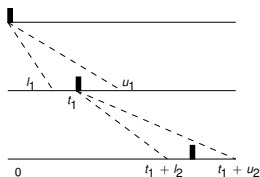
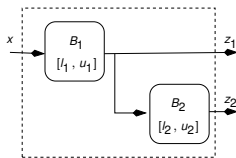
- ▶ Output of one component is an input for others
- ▶ Digital circuits, precedence between tasks, etc.



- ▶ We want to build an abstract model of the network as a component that over-approximates its timed I/O behavior

# Intuition on the Nature of the Abstraction I

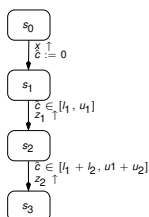
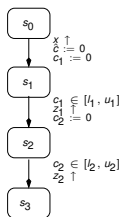
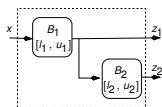
- ▶ Consider two timed components  $B_1$  and  $B_2$  each reacting to an input event within some  $t \in [l_i, u_i]$  time, connected in a network



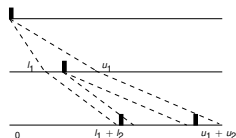
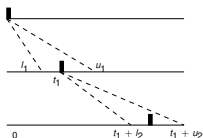
- ▶ In the detailed model  $z_1$  will take place within  $[l_1, u_1]$  time after  $x$  while  $z_2$  within  $[l_2, u_2]$  after  $z_1$
- ▶ Clock  $\hat{c}$  is an auxiliary clock that measures the time since input event  $x$

# Intuition on the Nature of the Abstraction II

- ▶ We discard internal clocks  $c_1$  and  $c_2$  and project timing constraints on input clock  $\hat{c}$



- ▶ In the abstract model  $z_2$  may happen at any  $t \in [l_1 + l_2, u_1 + u_2]$  regardless of the time of  $z_1$



# The Steps of the Abstraction Technique

$$\mathcal{A}_X \Rightarrow \mathcal{A}_X^{+\hat{C}} \Rightarrow \mathcal{A}_X^r \Rightarrow \mathcal{A}_X^{\hat{C}} \Rightarrow \mathcal{A}_{X_{io}} \Rightarrow \mathcal{A}_{X_{io}}^m$$

- ▶ Augment a network modeled by timed automaton  $\mathcal{A}_X$  with auxiliary clocks triggered by input events to obtain  $\mathcal{A}_X^{+\hat{C}}$
- ▶ Perform reachability computation on  $\mathcal{A}_X^{+\hat{C}}$  to obtain the equivalent interpreted timed automaton  $\mathcal{A}_X^r$
- ▶ Project the timing constraints in  $\mathcal{A}_X^r$  on the input clocks  $\hat{C}$  to obtain  $\mathcal{A}_X^{\hat{C}}$  whose qualitative semantics is exact but its timed semantics is an over approximation
- ▶ Project the transition labels in  $\mathcal{A}_X^{\hat{C}}$  on the interface variable to obtain  $\mathcal{A}_{X_{io}}$
- ▶ Minimize  $\mathcal{A}_{X_{io}}$  wrt to observable actions to obtain  $\mathcal{A}_{X_{io}}^m$

## Adding Input Clocks

- ▶ This is the hardest and most original part of our work
- ▶ Every input event generates a new clock upon its arrival
- ▶ The input event triggers a wave of reactions in the network
- ▶ Since the network is acyclic and every component has a finite upper bound on its reaction time, each event goes out of the system within finite time
- ▶ When the event leaves the system, its clock can be reused for other events
- ▶ Hence we can do with a finite number of clocks
- ▶ All the machinery of TA analysis is adapted to handle these dynamic clocks, monitor the life and death of events...

# Reachability/Simulation Graph and Interpreted Timed Automaton

- ▶ The standard technique to analyze timed automata using symbolic states of the form  $(q, Z)$  where  $q$  is a discrete state and  $Z$  is a subset of the clock space (zone)
- ▶ It leads to an equivalent automaton with an additional property: all paths are realizable under the timing constraints
- ▶ Relaxing the timing constraints of this automaton the qualitative untimed semantics is preserved
- ▶ Applying this analysis to the automaton augmented with input clocks, we add redundant constraints to the computed zones that do not affect the behavior
- ▶ But after projection on the auxiliary clocks these constraints are those that remain
- ▶ Output transitions now become conditioned upon the time elapsed since the events that triggered them

# The Other Steps

- ▶ Projection
- ▶ Hiding all the internal non-observable actions making them silent transitions
- ▶ Minimizing the obtained automaton by merging states that admit the same observable behaviors
- ▶ This is more involved than in untimed systems because we have also to merge zones (invariants and guards)
- ▶ A lot of work: 65K lines of C++ code inside the IF toolbox
- ▶ A front-end language: digital circuits made of gates with bi-bounded delay

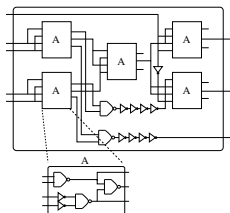


# Applications

- ▶ The reduced model can be an accompanying specification (contract) of the component, like specifying the response characteristic of electrical components
- ▶ Can be used to analyze large networks by divide and conquer
- ▶ Pick a subnetwork  $M$  which can still be analyzed using TA techniques, create the abstraction  $M'$  and compose it with the rest of the network
- ▶ Pick a subnetwork of that and so on
- ▶ We demonstrate how it can be applied to systems beyond the capabilities of current tools

## Example: Wave Pipelining

- ▶ A technique for improving the throughput of sequential circuits beyond a synchronous approach
- ▶ A second wave of inputs may arrive before the previous wave has been completely processed



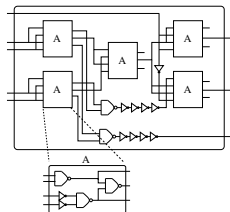
- ▶ Circuit has 36 logic gates with bi-bounded delays, each modeled by a timed automaton with one clock
- ▶ What is the highest frequency in which waves do not interfere? (assuming input is periodic with some jitter)

# Wave Pipelining: non Compositional Approach

- ▶ First we apply a non-compositional approach directly to a product of 36 timed automata
- ▶ The longest delay path is 1086
- ▶ We compose the circuit with an input generator of period 1200 and jitter of 10 which generates only one wave through the circuit
- ▶ Analysis takes 7 minutes to obtain a reachability graph with more than 26K symbolic states and 50K transitions
- ▶ Reducing the input period to 1000 to allow two simultaneous waves, the analysis gets stuck

## Wave Pipelining: Compositional Approach

- ▶ The circuit admits 5 instances of the sub-circuit *A*, two connected to the primary inputs
- ▶ Compose *A* with input generator of period 247 and jitter 10; May induce 2 simultaneous waves in the circuit
- ▶ Our abstraction reduces the reachability graph from 213/321 states/transitions to 34/58 with output jitter 32
- ▶ We compose the third copy of *A* with such an input generator and reduce the obtained graph from 270/411 to 36/62 and output jitter of 54...
- ▶ At the end we show no interference; reducing the input frequency from 247 to 246 we detect interference



# Conclusions

- ▶ We have developed a new technique for compositional timing analysis using timed automata
- ▶ The essence of the technique is to use the internal clocks of the components to eliminate infeasible behaviors and then discard these clocks
- ▶ The technique can be particularly useful when the number of simultaneous excitation waves in the network is much smaller than the number of components
- ▶ We used the technique to analyze system of a size much much beyond the capabilities of existing tools
- ▶ There is still a long way to go..

# A Personal Opinion on Timed Systems

- ▶ The level of abstraction provided by timed automata is extremely important
- ▶ It deserves to be one of the foundations of any theory of embedded systems
- ▶ Unfortunately one has to cross heavy definitions with large  $n$ -tuples to get to the point
- ▶ Consequently the domain attracts mostly pure theoreticians who care more about theorems than about the poor engineers (real or imaginary) that can benefit from the insights of the model
- ▶ I am not sure the situation will change after this talk, but all this work has been guided by optimism

Thank You