On Timed Components and their Abstraction

Oded Maler

CNRS-VERIMAG
Grenoble, France

September 2007

Plan

- Introduction: the importance of the timed level of abstraction
- A crash course in timed automata
- Scheduling with Timed Automata
- Abstraction
- Ongoing Work

Levels of Abstraction in Dynamic Description

It is well known that the same phenomenon can be described at different levels of abstraction

The more detailed level should give better predictions but would be computationally harder to analyze (and will require more detailed observations).

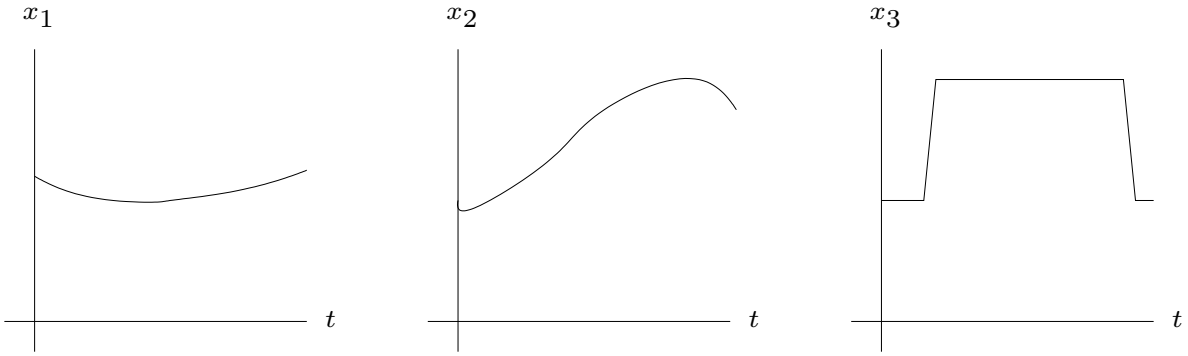
The trick of science/math has always been to find the level which is sufficiently refined to give meaningful results and sufficiently abstract to be tractable computationally

Physics, chemistry, biology, physiology, psychology, sociology, economy, ...

From Grenoble to Nancy: Continuous View

Let $x = (x_1, x_2, x_3)$ be a real-valued vector representing the location of my center of mass in a coordinate system adapted to the surface of the earth

The trip is specified as a 3-dimensional signal $x(t)$



Such behaviors (signals, trajectories) are generated by differential equations (or hybrid automata)

From Grenoble to Nancy: Discrete View

The trip is described as a sequence of states and transitions:

$$\text{Grenoble} \xrightarrow{\text{bus}} \text{Lyon} \xrightarrow{\text{plane}} \text{Metz} \xrightarrow{\text{bus}} \text{Nancy}$$

Transitions are considered as atomic, instantaneous events

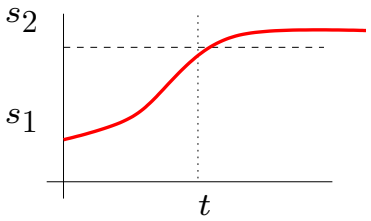
Such behaviors are generated by automata, transition systems, discrete-event systems, petri nets, process algebra, and worse

Sometimes we want to keep some of the continuous information, to express the fact that things take time

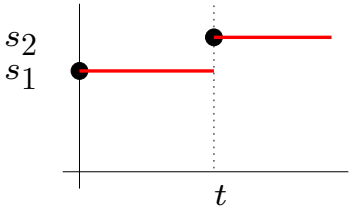
From Grenoble to Nancy: Timed View

The process of moving from one place to another is abstracted from it numerical details, but the **time** from initiation and termination is maintained

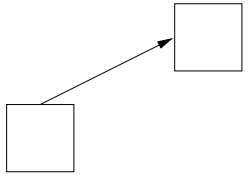
Grenoble $\xrightarrow{\text{bus}}$ on.bus $\xrightarrow{50}$ Lyon $\xrightarrow{\text{plane}}$ on.plane $\xrightarrow{70}$ Metz $\xrightarrow{\text{bus}}$ on.bus $\xrightarrow{25}$ Nancy



Continuous



Timed



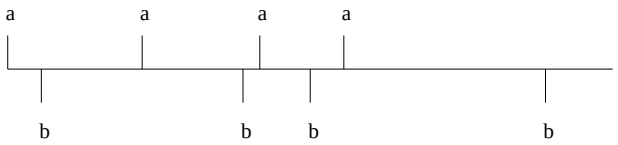
Dirscrete

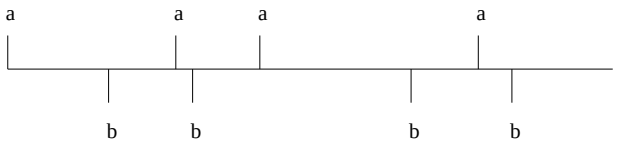
Mathematically Speaking

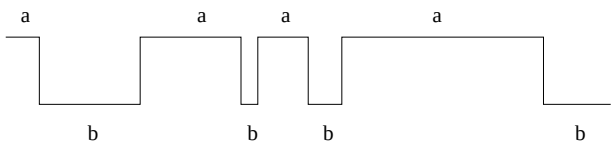
Discrete behaviors are viewed as **sequences** of events without metric timing information, only order or partial-order between the events.

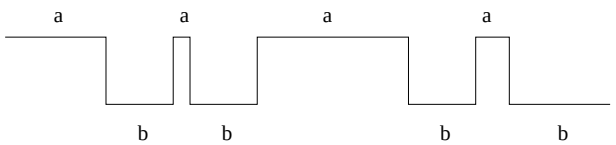
A timed behavior involves the embedding of the sequence into the real time axis.

a, b, a, b, a, b, a, b









Timed Dynamical Systems

What is the appropriate dynamical system model for the intermediate timed level?

We do not need arbitrary continuous variables

We need discrete states that tell us where we are (in the abstract state space)

We need additional information that tell us how long we have been in this or that state

This additional information is encoded using “clock” variables

Timed Automata are n-Tuples...

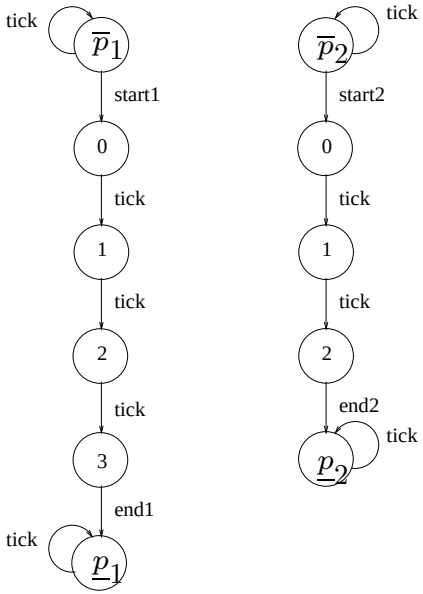
A timed automaton is $\mathcal{A} = (Q, C, I, \Delta)$ where...

The above is a sad fact that dooms timed automata into the formal verification circles and prevents it from being comprehensible to those who really need it

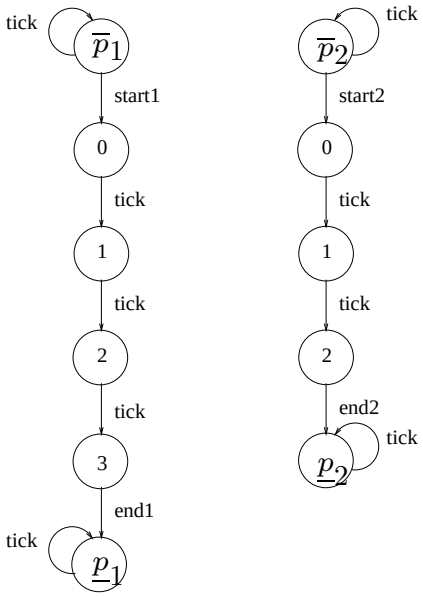
I'll try to avoid this as much as possible by giving intuitive explanations (hope you will not be offended)

Adding Time to Automata

Consider two processes that take 3 and 2 times units, respectively, after they start. We model the passage of 1 unit of time by a special $tick$ transition.



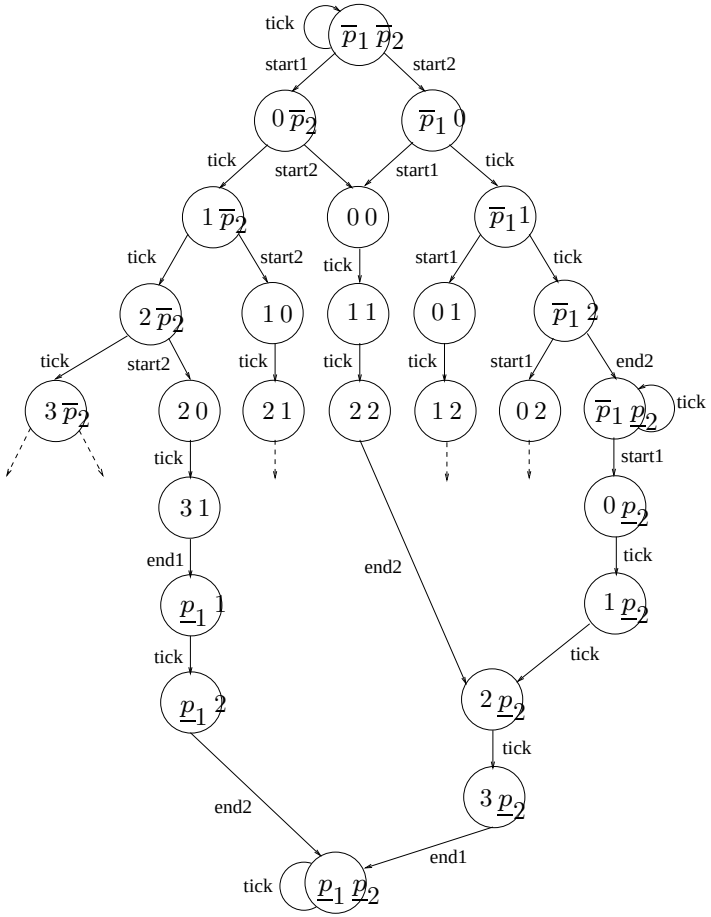
Possible Behaviors of the Processes



P_1 waits one time unit and then starts:

$$\overline{p}_1 \xrightarrow{tick} \overline{p}_1 \xrightarrow{start1} 0 \xrightarrow{tick} 1 \xrightarrow{tick} 2 \xrightarrow{tick} 3 \xrightarrow{end1} \underline{p}_1$$

The Two Processes in Parallel



Possible Joint Behaviors

Both processes start at time 2:

$(\overline{p}_1, \overline{p}_2) \stackrel{tick}{\longrightarrow} (\overline{p}_1, \overline{p}_2) \stackrel{tick}{\longrightarrow} (\overline{p}_1, \overline{p}_2) \stackrel{start1}{\longrightarrow} (0, \overline{p}_2) \stackrel{start2}{\longrightarrow} (0,0) \stackrel{tick}{\longrightarrow} (1,1) \stackrel{tick}{\longrightarrow} (2,2) \stackrel{end2}{\longrightarrow} (2, \underline{p}_2) \stackrel{tick}{\longrightarrow} (3, \underline{p}_2) \stackrel{end1}{\longrightarrow} (\underline{p}_1, \underline{p}_2)$

P_1 starts at 0 and P_2 starts at 2:

$(\overline{p}_1, \overline{p}_2) \stackrel{start1}{\longrightarrow} (0, \overline{p}_2) \stackrel{tick}{\longrightarrow} (1, \overline{p}_2) \stackrel{tick}{\longrightarrow} (2, \overline{p}_2) \stackrel{start2}{\longrightarrow} (2,0) \stackrel{tick}{\longrightarrow} (3,1) \stackrel{end1}{\longrightarrow} (\underline{p}_1, 1) \stackrel{tick}{\longrightarrow} (\underline{p}_1, 2) \stackrel{end2}{\longrightarrow} (\underline{p}_1, \underline{p}_2)$

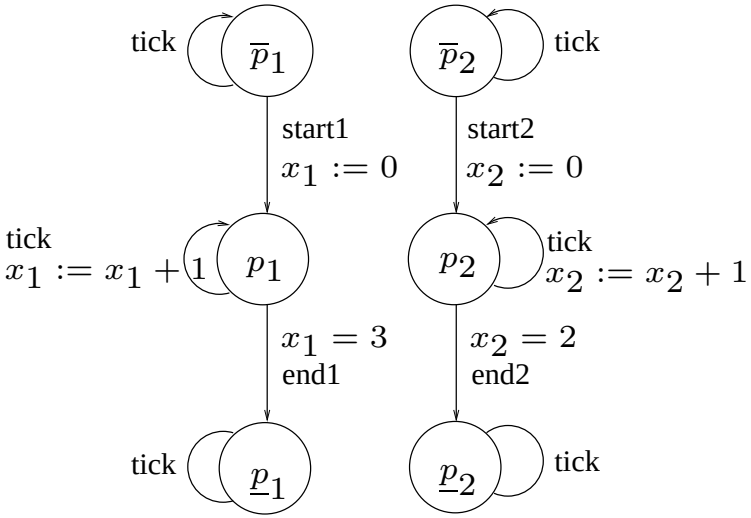
P_2 starts at 0 and P_1 starts after P_2 ends:

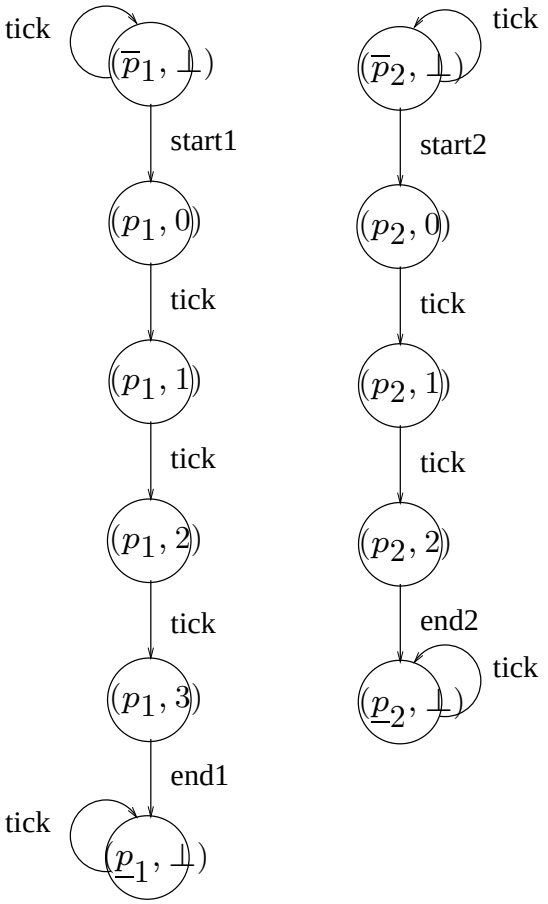
$(\overline{p}_1, \overline{p}_2) \stackrel{start2}{\longrightarrow} (\overline{p}_1, 0) \stackrel{tick}{\longrightarrow} (\overline{p}_1, 1) \stackrel{tick}{\longrightarrow} (\overline{p}_1, 2) \stackrel{end2}{\longrightarrow} (\overline{p}_1, \underline{p}_2) \stackrel{start1}{\longrightarrow} (0, \underline{p}_2) \stackrel{tick}{\longrightarrow} (1, \underline{p}_2) \stackrel{tick}{\longrightarrow} (2, \underline{p}_2) \stackrel{tick}{\longrightarrow} (3, \underline{p}_2) \stackrel{end1}{\longrightarrow} (\underline{p}_1, \underline{p}_2)$

Interleaving:

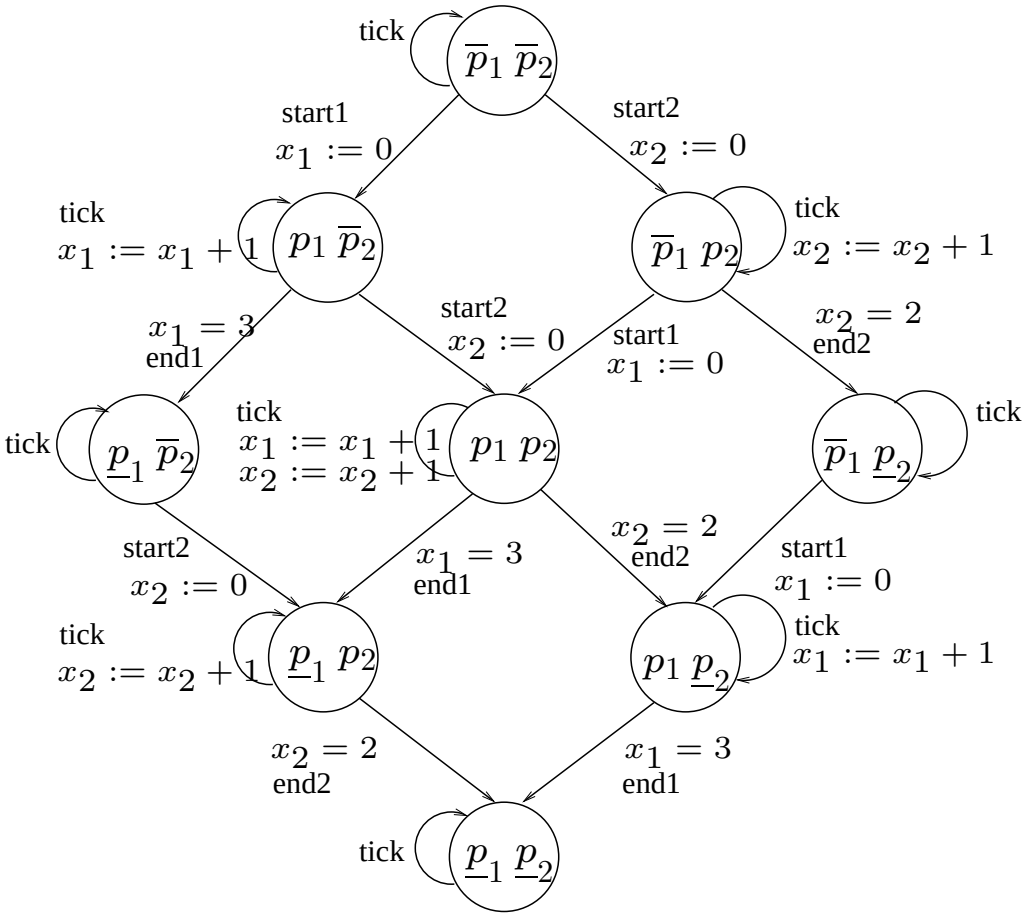
$(\overline{p}_1, \overline{p}_2) \stackrel{start1}{\longrightarrow} (0, \overline{p}_2) \stackrel{start2}{\longrightarrow} (0,0) = (\overline{p}_1, \overline{p}_2) \stackrel{start2}{\longrightarrow} (\overline{p}_2, 0) \stackrel{start1}{\longrightarrow} (0,0)$

Using Clock Variables





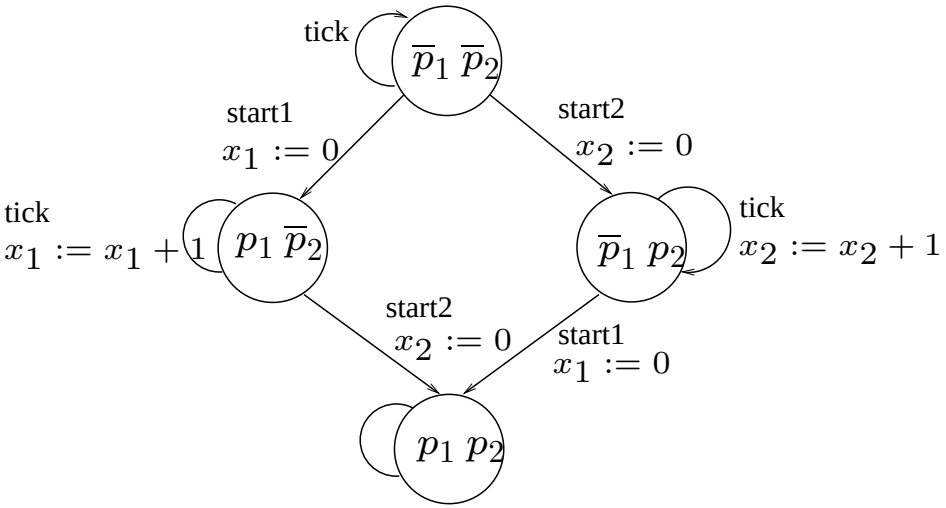
Clock Variables: the Composition



The Notion of a State

Warning: in automata augmented with variables, the **state** is encoded in both the discrete state (location) and the values of the variables.

The merging into (p_1, p_2) is misleading: via different paths you reach different clock valuations.



The Joy of Clock Variables

They allow succinct and natural representation of the system.

Transitions are labeled by **guards** and **resets**.

Different clocks represent the time elapsed since certain events.

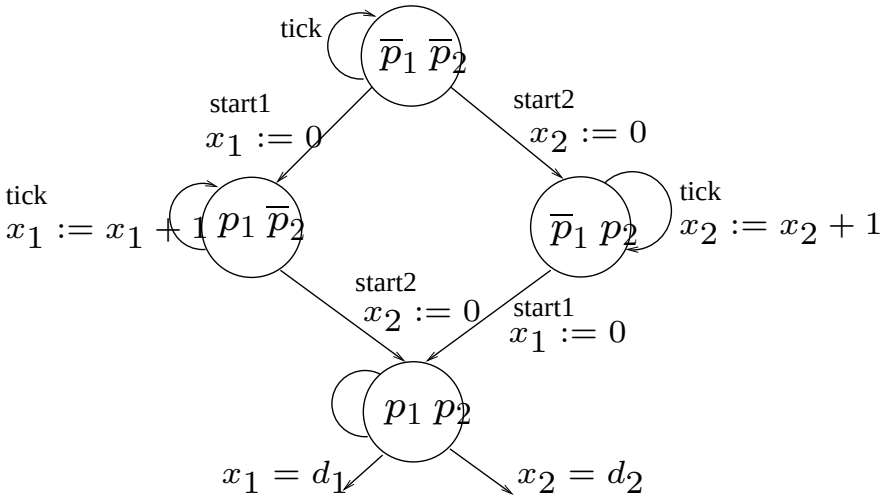
In the worst-case, however, one needs to expand the automaton by adding clock values to states.

You can use **symbolic** rather than enumerative encoding of the set of reachable states.

You can work in **dense** time without committing a-priori to time granularity.

Symbolic Representation

Assume the two processes with durations d_1 and d_2 such that $d_1 < d_2$ and that p_2 starts 2 time units after p_1.



The set of clock values that can be reached at state (p_1, p_2) is $\{(2,0),(3,1),(4,2),\ldots(d_1, d_1-2)\}$ and its size depends on d_1.

It can be, however, represented by a fixed size formula $X_1 - X_2 = 2 \wedge X_1 \leq d_1$

From Discrete to Dense Time

So far we have assumed a fixed time granularity Δ associated with a $tick$.

Discrete time flows in Δ quanta by the $tick$ transitions. These transitions induce self-loops on the states of all automata.

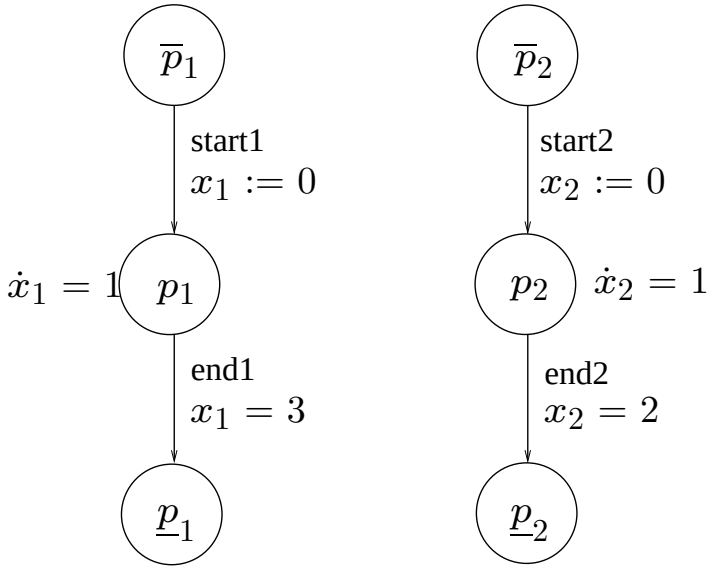
Other transitions can be taken only at time points $n\Delta$, $n \in \mathbb{N}$.

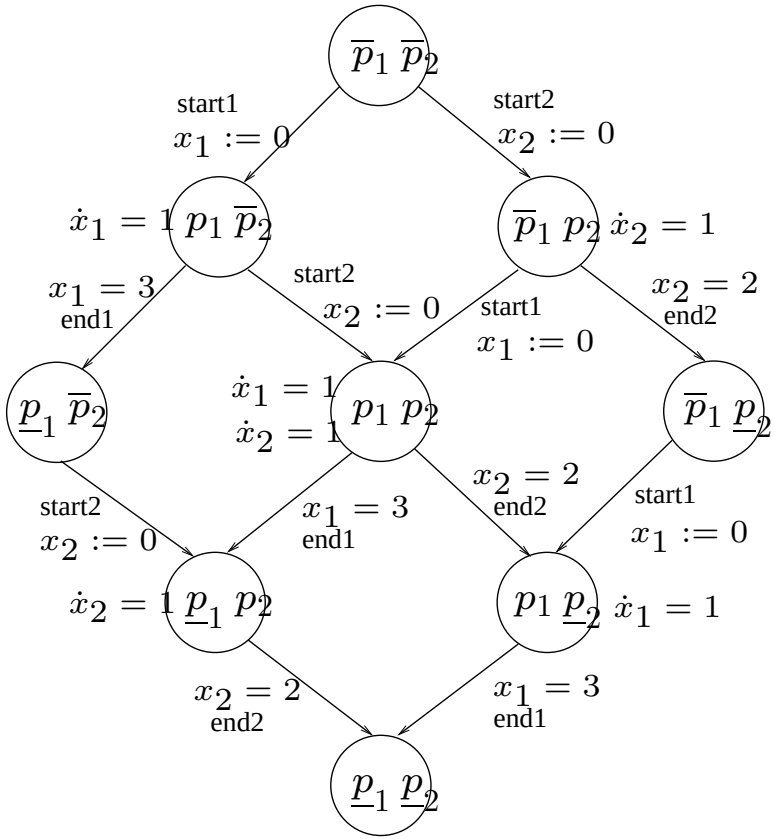
By considering clocks as continuous variables we can use time-passage of **arbitrary** length.

Time passage, instead of being represented by $tick$ transitions, can be modeled by all active clocks advancing with **derivative** 1 when the automaton stays in a state.

The timed automaton is viewed as a simple kind of a hybrid automaton whose evolution alternates between passage of time and discrete transitions.

The Two Processes as Two Timed Automata





Modeling Temporal Uncertainty

The major strength of timed automata is their ability to express **temporal uncertainty**.

"The duration of a task (or the distance between two events) is somewhere in the interval $[l, u]$"

Using dense time this means **anywhere** in $[l, u]$ not just l or u

Verification can be done with respect to **all** choices of values in the interval

This CS non-determinism is an alternative/complement to probabilistic modeling of uncertainty (for example exponential distribution of durations)

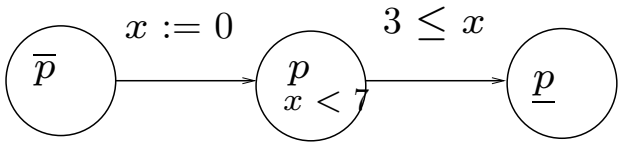
Modeling Temporal Uncertainty with TA

There are different ways to model urgency/non-urgency in TA:

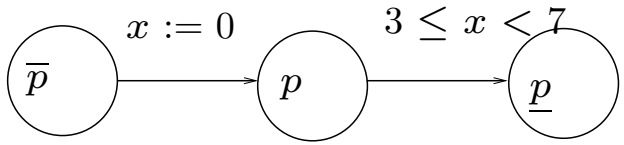
1) Invariants (staying conditions) that the clocks must satisfy in order to remain in a state and "let" time progress.

2) Deadlines on transitions.

Example: a task whose duration is between 3 and 7 time "units":



Invariants



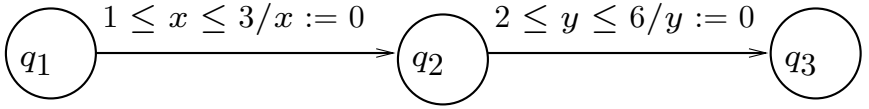
Deadlines

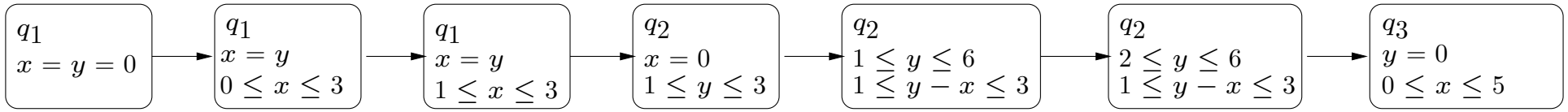
$$(\overline{p},\bot) \xrightarrow{2.5} (\overline{p},\bot) \xrightarrow{start} (p,0) \xrightarrow{3.8} (p,3.8) \xrightarrow{end} (\underline{p},\bot)$$

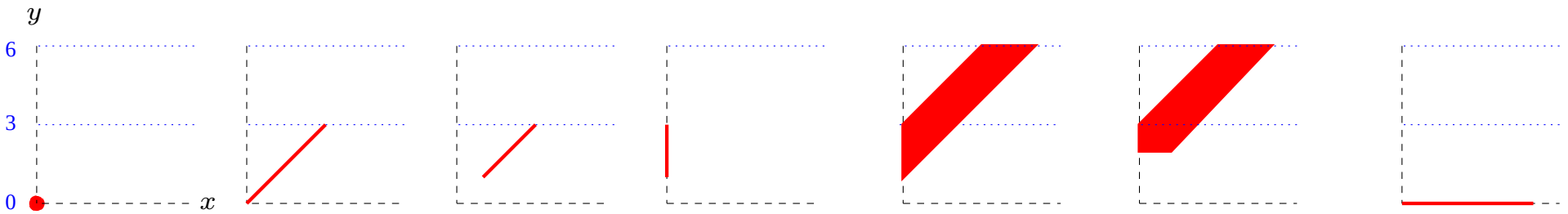
$$(\overline{p},\bot) \xrightarrow{t_1} (\overline{p},\bot) \xrightarrow{start} (p,0) \xrightarrow{t_2} (p,t_2) \xrightarrow{end} (\underline{p},\bot)$$

$$t_1 \in [0,\infty), t_2 \in [3,7].$$

Verification (Reachability) of Timed Automata







Timed Automata are n-Tuples...

A timed automaton is $\mathcal{A} = (Q, C, I, \Delta)$ Q: a set of states, C: a set of clocks,

I: staying condition (invariant), assigning to every q a conjunction I_q of inequalities of the form $c \leq u$, for some clock c and integer u

Δ: a transition relation consisting of tuples (q, ϕ, ρ, q') where q and q' are states,

$\rho \subseteq C$ is the set of clocks reset by the transition, and

ϕ (the transition guard) is a conjunction of formulae of the form $c \geq l$ for some clock c and integer l.

A clock valuation is a function $\mathbf{v} : C \rightarrow \mathbb{R}_+ \cup \{0\}$ and a configuration is a pair $(q, \mathbf{v})$ consisting of a discrete state (location) and a clock valuation.

Runs of Timed Automata

A *step* of the automaton is one of the following:

- A discrete step: $(q, \mathbf{v}) \xrightarrow{\delta} (q', \mathbf{v}')$, for some transition $\delta = (q, \phi, \rho, q') \in \Delta$, such that $\mathbf{v}$ satisfies ϕ and $\mathbf{v}' = R_\rho(\mathbf{v})$.

- A time step: $(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t\mathbf{1})$, $t \in \mathbb{R}_+$ such that $\mathbf{v} + t\mathbf{1}$ satisfies I_q.

A run of the automaton starting from a configuration $(q_0, \mathbf{v}_0)$ is a finite sequence of steps

$$\xi : \quad (q_0, \mathbf{v}_0) \xrightarrow{t_1} (q_1, \mathbf{v}_1) \xrightarrow{t_2} \cdots \xrightarrow{t_n} (q_n, \mathbf{v}_n).$$

Symbolic Reachability Computation

A symbolic state is (q, Z) where q is a discrete state and Z is a zone, a set of clock valuations satisfying a conjunction of inequalities $c_i - c_j \geq d$ or $c_i \geq d$.

Symbolic states are closed under the following operations:

- The time successor of (q, Z), the configurations reachable from (q, Z) by letting time progress without violating the staying condition of q:

$$Post^t(q, Z) = \{(q, \mathbf{z} + r\mathbf{1}) : \mathbf{z} \in Z, r \geq 0, \mathbf{z} + r\mathbf{1} \in I_q\}$$

- The δ-transition successor of (q, Z) is the configurations reachable from (q, Z) by taking the transition $\delta = (q, \phi, \rho, q') \in \Delta$:

$$Post^\delta(q, Z) = \{(q', R_\rho(\mathbf{z})) : \mathbf{z} \in Z \cap \phi\}$$

- The δ-successor of a time-closed symbolic state (q, Z) is the set of configurations reachable by a δ-transition followed by passage of time:

$$Succ^\delta(q, Z) = Post^t(Post^\delta(q, Z))$$

The Reachability Graph

The basic verification algorithm for TA consists of on-the-fly generation of the reachability (simulation) graph, $S = (N, \rightarrow)$

The nodes are symbolic states computed starting from $Post^t(s, \{\mathbf{0}\})$ and applying $Succ^\delta$ until termination (guaranteed due to finitely-many zones)

There is a path from (q, Z) to (q', Z') in S iff for every $\mathbf{v}' \in Z'$ there exists $\mathbf{v} \in Z$ and a run of $\mathcal{A}$ from $(q, \mathbf{v})$ to $(q', \mathbf{v}')$.

Hence the union of all symbolic states in S is exactly the set of reachable configurations.

This is the computation we want to do more efficiently

The Sources of Difficulty

Assume we have n interacting timed automata, each with m states and one clock ranging over $[0, d]$

The number of states can be up to m^n and the number of zones can be up to $d^n n!$, summing up to $m^n d^n n!$ symbolic states. Each zone takes $O(n^2)$ space

The representation of (convex) zones is fine but there is no nice representation for a union of zones and, even worse, the representation is not symbolic for the discrete states: symbolic states are of the form (q, Z) with q being an explicit n-vector.

Scheduling with Timed Automata (Y. Abdeddaim, 98-00)

As mentioned earlier, timed automata exhibit dense non-determinism: a transition can be taken at any point in an interval $[l, u]$

In verification, where the non-determinism is associated with the external uncontrolled world, we need to take all these choices into consideration

In synthesis/optimization where the choice of when to take a transition depends on us, sometimes we need not consider the whole interval but only some points in it that “dominate” the others

This turned out to be the case in optimal scheduling problems where it is sufficient to consider only a small subset of the runs

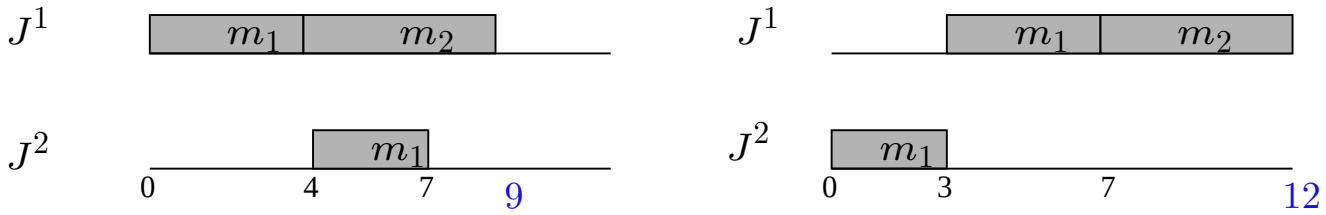
Deterministic Job-Shop Scheduling: the Problem

$$J^1: \quad (m_1, 4), (m_2, 5) \qquad\qquad J^2: \quad (m_1, 3)$$

Determine the execution times of the tasks such that:

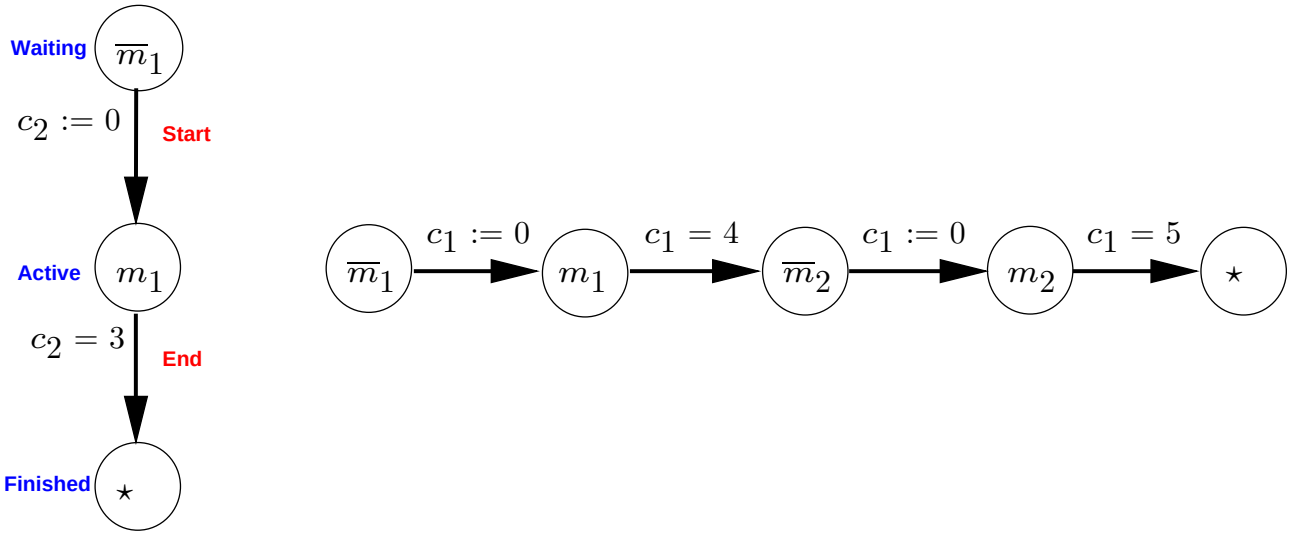
The termination time of the last task is minimal

Precedence and resource constraints are satisfied



Sometimes it is better not to start a task although the machine is idle

Modeling with Timed Automata

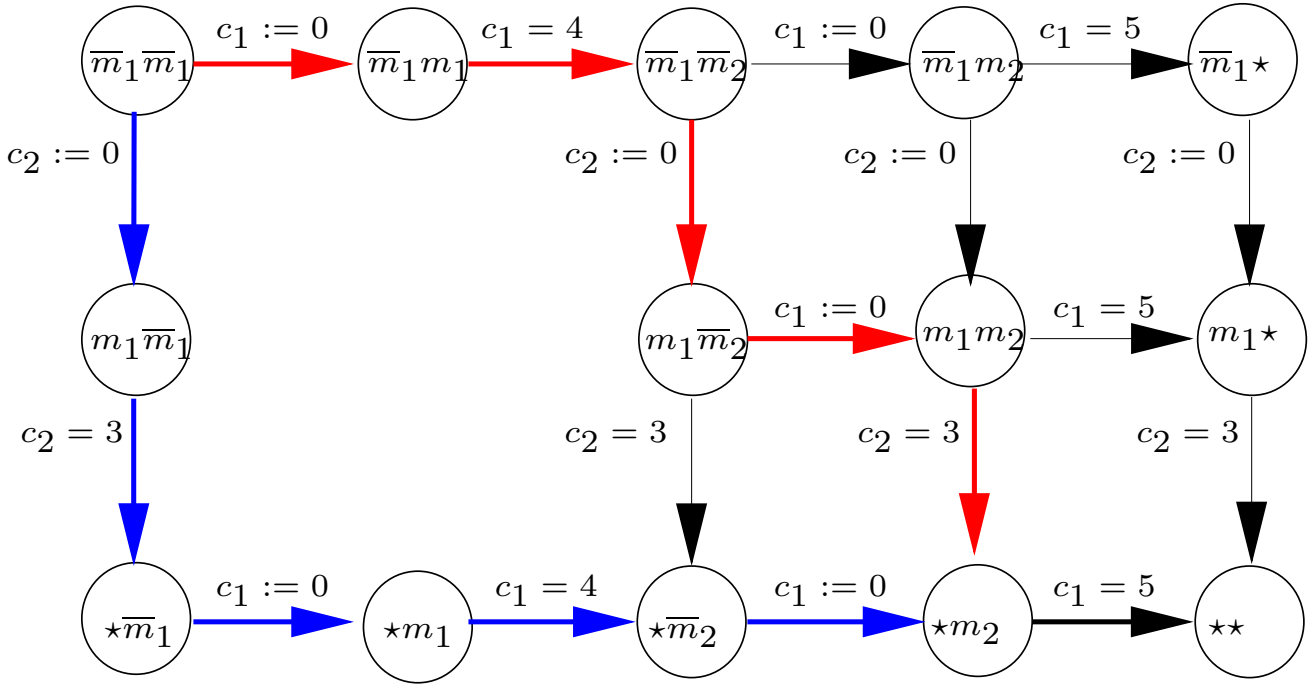


Each automaton represents the set of all possible behaviors of each task/job in isolation (respecting the precedence constraints)

The Start transitions are issued by the controller/scheduler and the End transitions by the environment

The Global Automaton

Resource constraints expressed via forbidden states in the product automaton



Optimal scheduling = shortest path problem for timed automata

Finding the Shortest Path

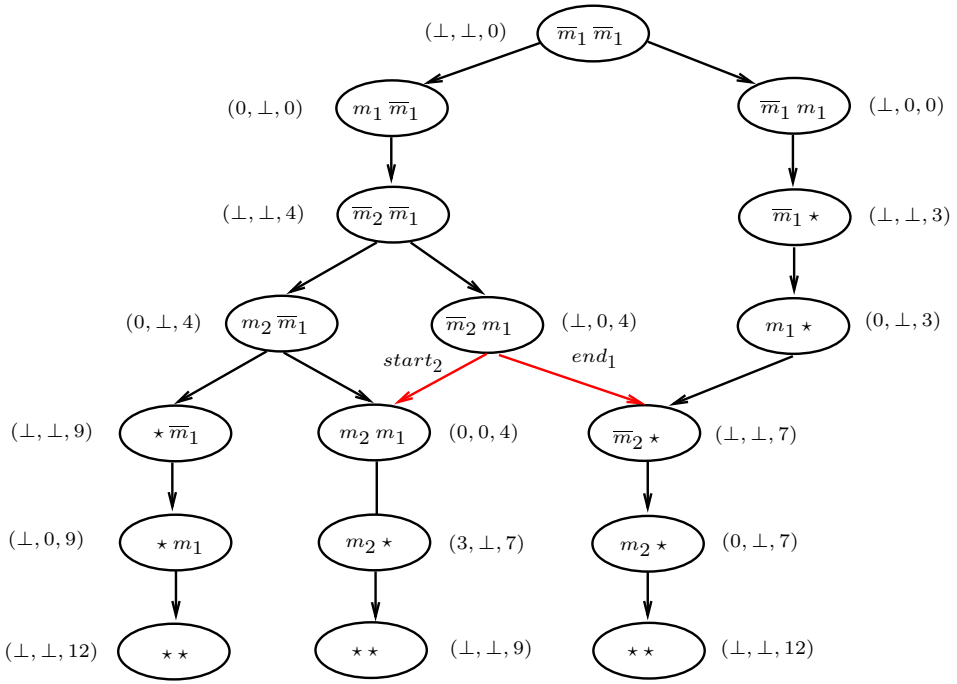
Add an additional clock T which is never reset to zero, hence it measures the absolute time since the beginning

Naive approach: perform zone-based reachability computation on the extended clock space (the graph is acyclic and all paths lead to the final state); Find the minimal value of T over all symbolic states associated with the final state

However, it can be shown that postponing a $start$ transition from t to t' is useless if the machine is not used by anyone else during $[t, t']$

Hence the optimum can be found among a finite number of schedules/runs where a transition not taken in a state at the first moment it was enabled will not be taken at that state at all

Scehduling with Timed Automata



Lessons: there is life after operations research

Abstraction (R. Ben Salah, M. Bozga, 02-07)

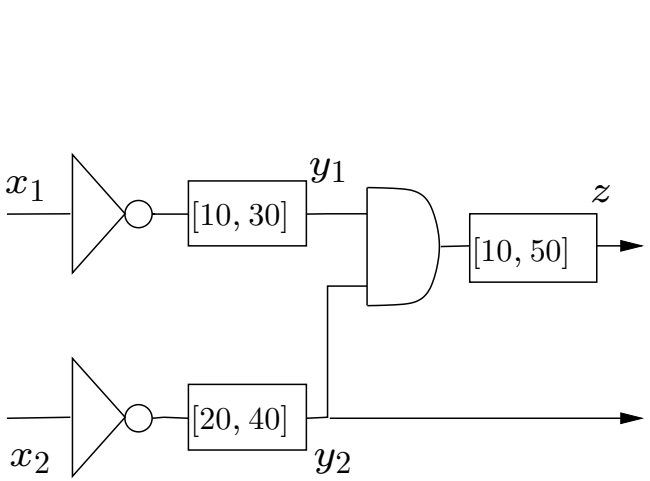
Principle is simple: the system $S = S_1||S_2||\cdots||S_n$ is made of components whose product explodes

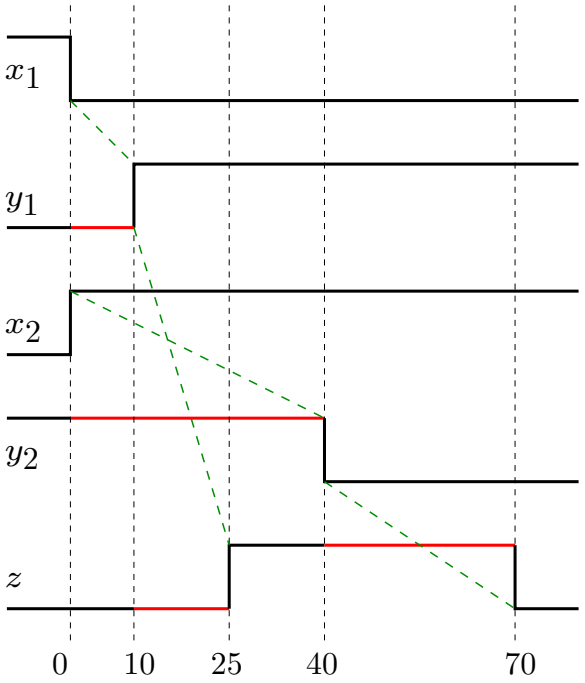
Replace each (or some) S_i by and S'_i such that $S'_i < S_i$ in syntax and $S'_i > S_i$ in semantics

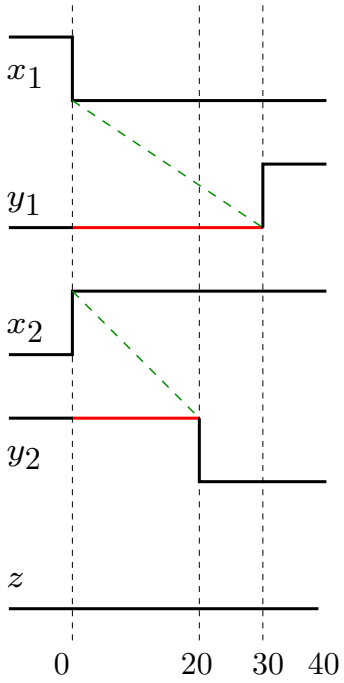
Correctness of $S' = S'_1||S'_2||\cdots||S'_n$ implies correctness of S and may be computationally easier

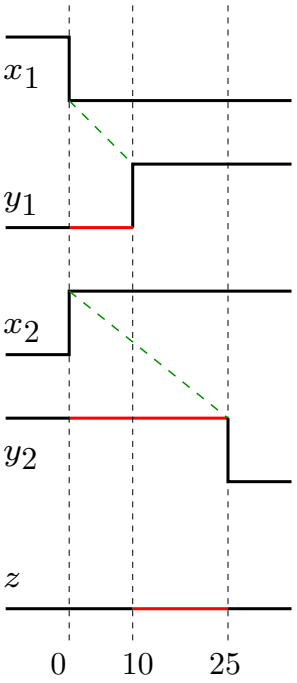
We developed an automatic methodology to create such abstractions, specialized (bot not restricted to) Boolean circuits with delays

Circuits with Bi-bounded Inertial Delays





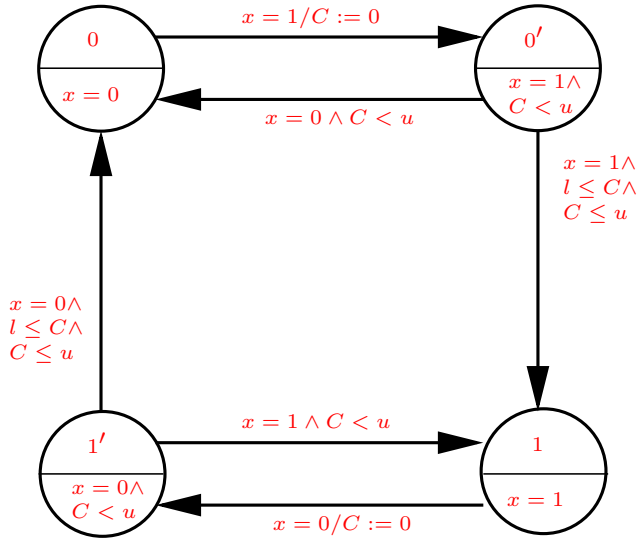




Modeling Circuits with Timed Automata

Our modeling approach, based on [Maler and Pnueli 95]: Decompose any gate into an instantaneous Boolean function and a bi-bounded (non-deterministic) inertial delay element

Model every delay element as a timed automaton with 4 states and 1 clock



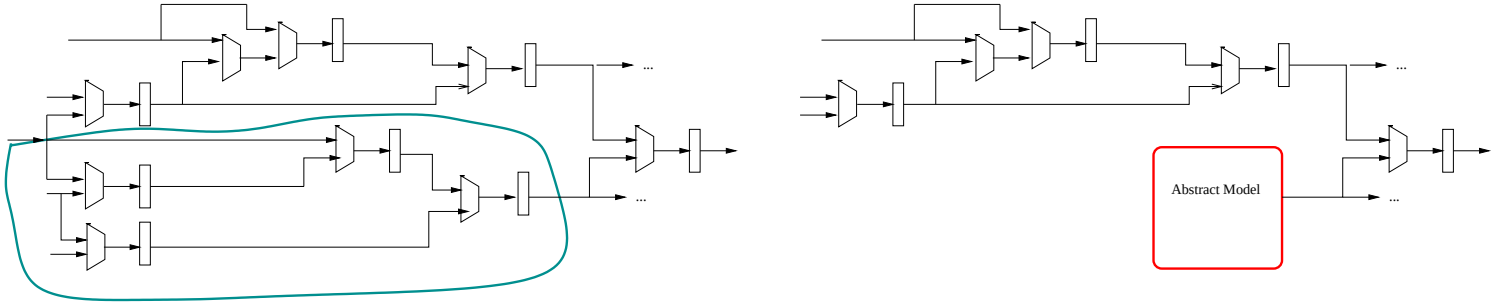
Composing all these automata we obtain a timed automaton with $O(2^n)$ states and n clocks

Abstraction of Acyclic Circuits

Start with a stable states, primary inputs change only once at start. This induces a non-countable number of possible behaviors

Each behavior admits a finite number of changes and stabilizes in a bounded amount of time. We want to compute the maximal stabilization time, that of the worst behavior

The basic idea: take a sub-circuit on the left, use TA technology to generate an approximate timed model of its output. It is then plugged as an input model to the rest of the circuit.



The Reachability Graph

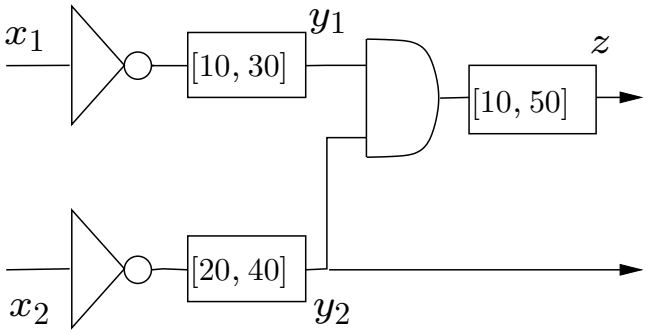
The reachability graph of a timed automaton can be viewed as an "interpretation" of the automaton:

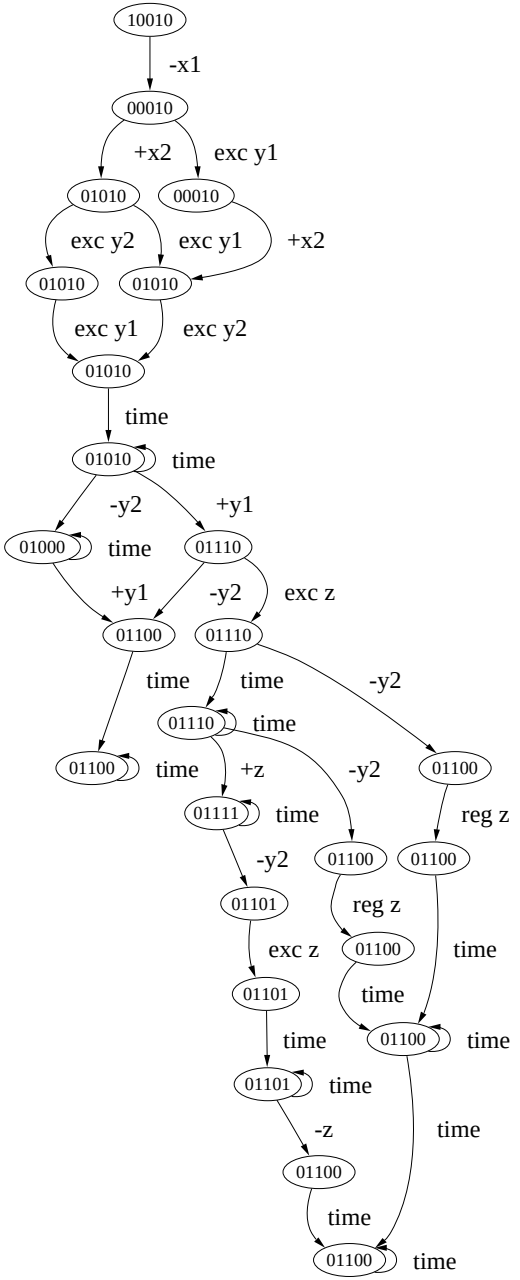
On on one hand we split some discrete states according to clock values

On the other, we remove transitions that are infeasible due to timing constraints.

By associating with each symoblic state (q, Z) the staying condition Z and with each outgoing transition the intersection of Z with the guard we obtain a TA equivalent to the original one where all states are reachable from the initial state.

The abstraction is done by applying certain transformation to this timed automaton


x_1
y_1
z
[10, 30]
[10, 50]
[20, 40]
x_2
y_2


10010
-x1
00010
+x2
exc y1
01010
00010
exc y2
exc y1
+x2
01010
01010
exc y1
exc y2
01010
time
01010
time
-y2
+y1
01000
time
01110
+y1
-y2
exc z
01100
01110
time
time
-y2
01110
time
01100
time
+z
-y2
01100
01111
time
reg z
-y2
01100
01100
01101
reg z
exc z
01100
time
01101
time
time
01100
time
01101
time
-z
time
01100
time
01100
time

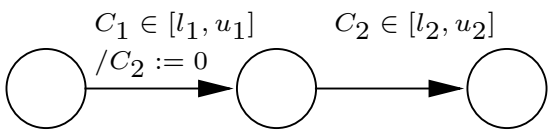
The Nature of the Abstraction

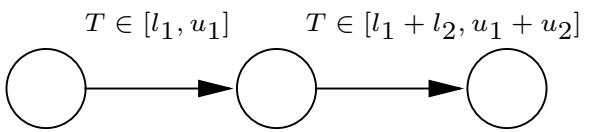
First, the obvious thing: hiding internal actions such as excitation and “regrets” of the outputs and all transitions of internal wires.

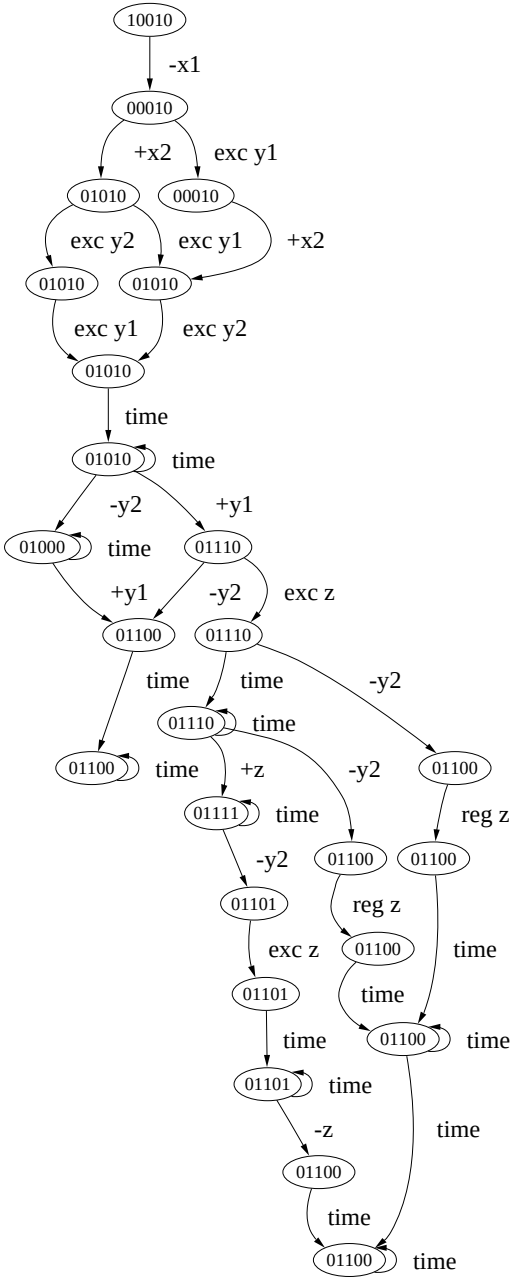
Relaxation of timing constraints by allowing things to happen at impossible times (but not in impossible orders!)

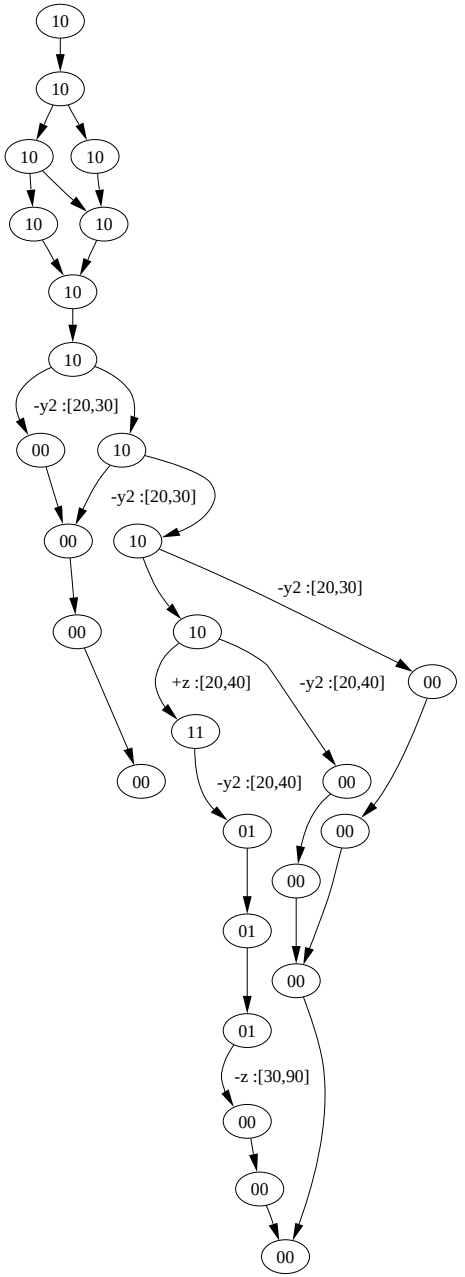
We project the TA obtained from the reachability graph on a subset of the clocks. The constraints related to the other clocks are removed.

For acyclic circuits it is natural to project only on the auxiliary clock T that measures absolute time. This way we keep the information about the time each transition can be taken (but lose some inter-dependence information).



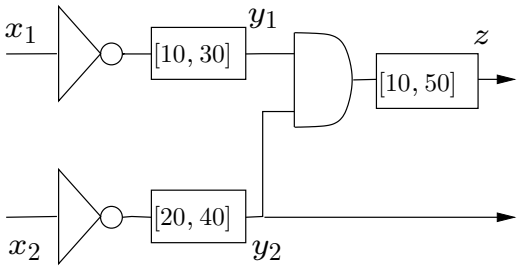


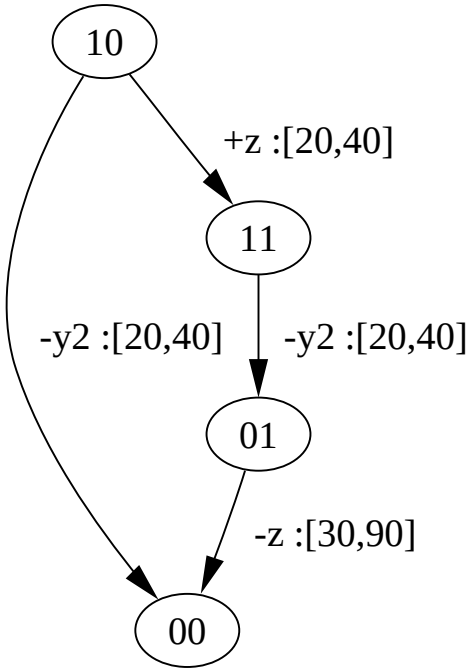

10010
-x1
00010
+x2
exc y1
01010
00010
exc y2
exc y1
+x2
01010
01010
exc y1
exc y2
01010
time
01010
time
-y2
+y1
01000
time
01110
+y1
-y2
exc z
01100
01110
time
time
-y2
01110
time
01100
time
+z
-y2
01100
01111
time
reg z
-y2
01100
01100
01101
reg z
exc z
01100
time
01101
time
time
01100
time
01101
time
-z
time
01100
time
01100
time


10
10
10
10
10
10
10
10
-y2 :[20,30]
00
10
-y2 :[20,30]
00
10
-y2 :[20,30]
00
10
+z :[20,40]
-y2 :[20,40]
00
11
00
-y2 :[20,40]
00
01
00
00
01
00
01
00
-z :[30,90]
00
00
00

Minimization

After minimization we obtain the following small-description abstraction for the observed behavior of the circuit:





Abstraction - Current Status

For acyclic circuits we could treat circuits with up to 100 gates. Still a far cry from static methods used in industry

We have developed a very interesting novel method for abstracting open timed components (the inputs may arrive anytime, not only in time zero)

Each event generates its own clock (which is killed after the event propagates through the system)

After projecting on those input clocks we obtain a reduced model where output events are constrained by the time elapsed since the events that triggered them

This way we obtain an approximation of the timed input-output behavior of the system (work in progress)

Thank You