Modeling and verification certain classes of systems

extensions of finite state-transition systems

- Model Checking Real-Time Systems
  TCTL formulae, a natural extension of CTL
- Model Checking Security Protocols
- Model Checking Probabilistic Systems

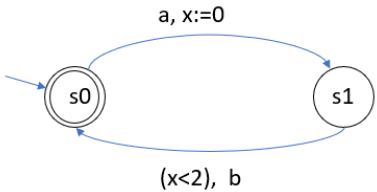
Timed Automata

- A timed automaton consists of an automaton (states and transitions) and real-valued variables for measuring time between transitions, named Clocks
- Clocks progress synchronously with time (time domain R^+), and can be reset by transitions
- a transition consists of a guard, an action and a reset of clocks
- invariants can be added to states
- The operational semantics of a timed automaton is an (infinite-state) timed transition system.

Rajeev Alur and David L. Dill.
A theory of timed automata, Theoretical Computer Science, 126, 2, 1994

An example

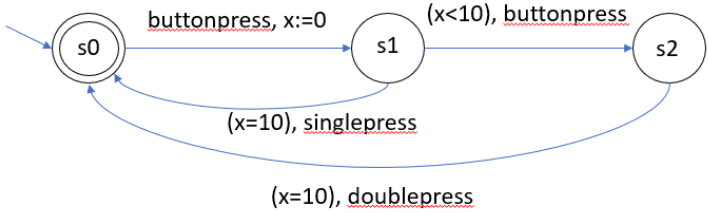
This timed automaton has one clock: x



This automaton moves from initial state $s0$ to $s1$ by executing action a. The clock x is set to 0 along this transition.
While in state $s1$, the clock x shows the time elapsed since the occurrence of the last a.
The transition from $s1$ to $s0$ is enabled only if this value is less than 2.
The the automaton moves back to state $s0$ and the cycle is repeated.

Another simple example

This timed automaton has one clock: x

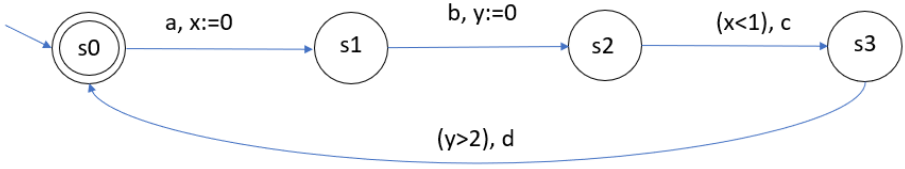


This automaton leaves the initial state *s*0 by executing action *buttonpress*, and reset the clock *x* to 0. If action *buttonpress* is executed again before 10, the automaton moves to state *s*2, and, when the clock reaches 10, recognises a double click of the button. Otherwise, being in state *s*1, when the clock reaches 10, the automaton recognises a single click of the button.

Another example

The timed automaton has two clocks: x and y.
The automaton cycles among the states $s0$, $s1$, $s2$, $s3$

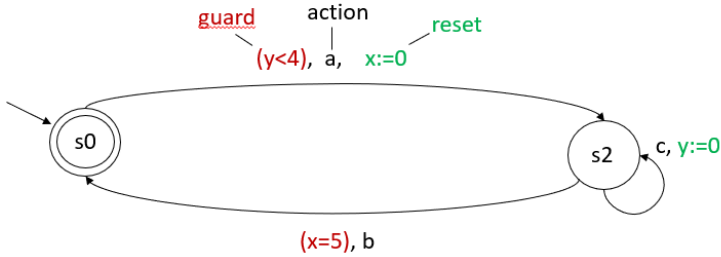
Clock x is set to 0 each time it moves from $s0$ to $s1$ executing a.
c happens within time 1 from the preceeding a
clock y is set while executing b
the delay between b and the following d is greater than 2



Timed Automata

x,y: clock

transition relation: $s \xrightarrow{action,time} s'$



$$s_0 \xrightarrow[3.2]{a} s_1 \xrightarrow[5.1]{c} s_1 \xrightarrow[8.2]{b} s_0 \cdots$$

value of x:	0	0	1.9	5	$\cdots$
value of y:	0	3.2	0	3.1	$\cdots$

Timed Automata

A transition is labelled with one action, a constraint (guard) and the reset of zero or more clocks (a, g, r). A reset is a clock to be set to zero.

The *state* of a timed automaton is given by the location and the values of the clocks at a given time.
State: $(location, x = v, y = w)$ with $v, w \in R^+$

Execution trace:

$$(s_0, x = 0, y = 0) \xrightarrow{a} (s_1, x = 0, y = 3.2) \xrightarrow{c}$$
$$(s_1, x = 1.9, y = 0) \xrightarrow{b} (s_0, x = 5, y = 3.1) \cdots$$

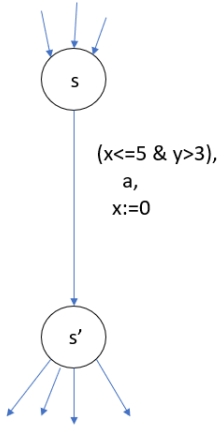
A timed automaton models a system operating in a number of distinct *modes*. Each mode is represented by a location. Mode changes occur as a consequence of the execution of actions.

While the system remains in a given location, progress of time is reflected by the values of the clocks, whose values increase all at the same rate.

Adding invariants

Let us consider the following transition of a timed automaton, where *a* is an action used for synchronisation with other automata.

What happens when *a* does not occur? Until *a* does not occur, time alone flows



Clocks: x, y

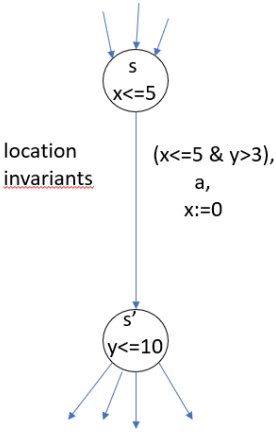
Transitions:

(s, x=2.4, y=3.1415) —a→ (s', x=0, y=3.1415)

(s, x=2.4, y=3.1415) —wait(1,1)→ (s, x=3.5, y=4.2415)

Adding invariants

Invariants ensure progress



Clocks: x, y

Transitions:

(s, x=2.4, y=3.1415) —wait(3.2)→ ✗

(s, x=2.4, y=3.1415) —wait(1,1)→ (s, x=3.5, y=4.2415)

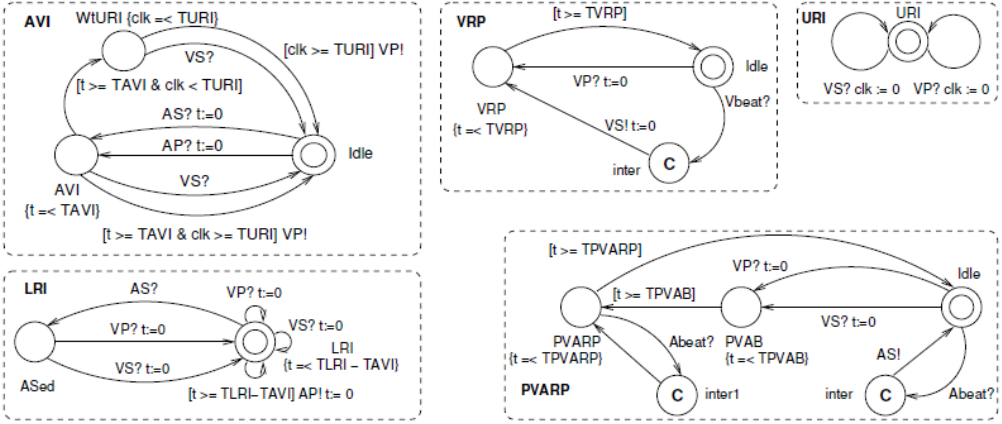
Networks of timed automata

A network of timed automata is a finite set $\{A1, \cdots, An\}$ of timed automata. Let L_i be the set of locations of automaton A_i. A location vector of the network is a set $l = \{l1, \cdots, ln\}$, where $li \in Li, i = 1, \cdots, n$. The initial location vector $l0$ of the network is the set $\{l0_1, \cdots, l0_n\}$ of initial locations.

Complementary actions: two actions of the same name $a?$ and $a!$ are said to be complementary, or matching.

In a network of timed automata, the complementary actions are synchronisation actions and become the network internal action, denoted by τ.

A TA network modeling the behavior of a pacemaker



A TA network modeling the behavior of a pacemaker

Local clocks. Each automaton enforces some constraint on the intervals between pairs of events, using clocks named t (this name is local to each automaton).

Global clocks. clk is a global clock and it is accessed both by the URI and the AVI automata.

Square brackets enclose guards and curly brackets enclose state invariants. Two concentric circles denote the initial location. Letter *C* denotes committed locations, meant to model an immediate transition.

Actions *VS*?, *VP*!, *AS*? and *AP*! are input/output actions towards the model of the heart (not shown here).

LRI automaton

The LRI automaton keeps the heart rate above a minimum value, defined by the Lower Rate Interval (TLRI).

The automaton starts in the LRI location. Upon a ventricular event (VP? or VS?), clock t is reset. Upon an atrial sense event (AS?), the automaton waits in location ASed until a ventricular event occurs, which causes the automaton to return to the LRI location, resetting t.

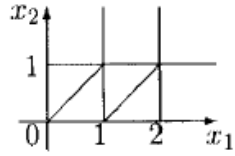
Since an atrial event must occur within $TLRI - TAVI$ seconds after the last ventricular event, an atrial pulse (AP!) must occur if the automaton remains in LRI for a longer interval.

This is specified by the invariant $\{t \in TLRI - TAVI\}$ on location LRI and by the edge labeled with the guard $[t > TLRI - TAVI]$, the action AP!, and the reset of t. This models the issue of an atrial pacing pulse by the pacemaker.

Reachability

Given an automaton A, reachability answers the question whether a distinguished set of locations of A is reachable or not.

For each clock x in A, let M_x be the maximal constant clock x is compared to in A.
Consider, clocks x_1, x_2 with Consider with $M_1 = 2$ and $M_2 = 1$.
Clock region



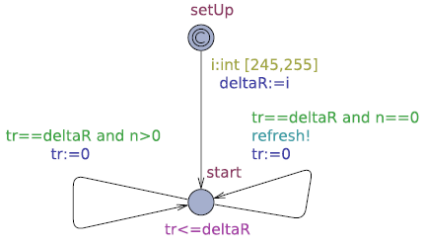
A simple reachability specification may require that the timed automaton never enters the region. Clock regions are finite. Region automaton.

Software tools

- Timed automata are a formalism for model checking real-time systems
- The theory of timed automata are implemented in the UPPAAL model checker
- simulation
- model checking

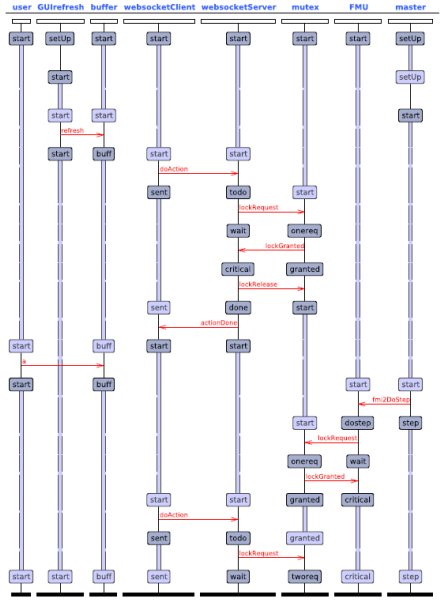
Example: a timed automaton in UPPAAL

The timed automaton operates on one clock `tr` and a variable `n`. When the clock is equal to the constant `deltaR`, an out edge is executed (`tr <= deltaR` is the state invariant). If variable `n` equals to 0, the TA sends a `refresh` message and resets the clock. Otherwise (`n > 0`), only resets the clock.



The TA models the automatic refresh actions performed by a GUI every `deltaR` time units, in absence of user actions (n==0). Initially, a random value in the interval $[245, 255]$ms is assigned to `deltaR` (4 Hertz)

Simulation


user
GUIrefresh
buffer
websocketClient
websocketServer
mutex
FMU
master
start
setUp
refresh
doAction
sent
todo
lockRequest
wait
onereq
lockGranted
critical
granted
lockRelease
done
actionDone
buff
a
fmi2DoStep
dostep
step
tworeq

Model checking

TCTL quantifiers

E - exists a path (E in UPPAAL).
A - for all paths (A in UPPAAL).
G - all states in a path ([] in UPPAAL).
F - some state in a path (<> in UPPAAL).

A[]p, A<>p, E<>p, E[]p and p –> q

where p is a local property

p::= a.l | gd | gc | p and p | p or p | not p | p imply p

Model checking

The types of properties that can be directly checked using the UPPAAL queries are quite simple (liveness, safety formulae).

A[] not deadlock
is a special case in UPPAAL for proving that for each reachable state, there exists at least one outgoing edge.

A[] (deltaR < deltaM and FMU.critical) imply c!=2
proves a property under specific timing constraints