Figure 1: Limiting delay times.

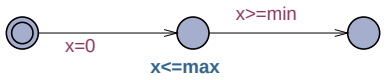
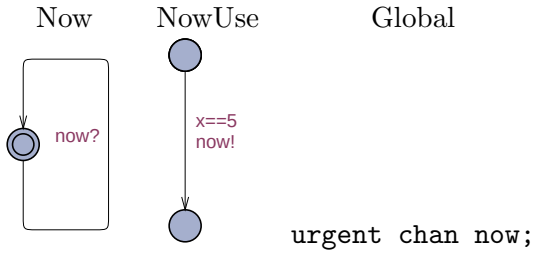


Figure 2: Design pattern for urgent reaction.



1 Limiting delays

One of the most basic modeling patterns in timed automatas is restricting delays. To start measuring a delay, a clock x must be set to 0. Any transition that must wait at least min time units, needs a guard $x \geq min$. If we must react before the time exceeds max then the state needs an invariant $x \leq max$. (See Fig. 1.)

2 Reacting to a changing variable

Sometimes, it is necessary to react to a changing variable without delay. In this section, we describe a trick using and urgent channels that is capable of forcing a reaction to a change in variables without delay.

Suppose, we need to perform a transition as soon as x reaches the value 5. Then obviously, we need to have the guard $x == 5$ on the edge, but this does not force the transition. To force the transition, we add a synchronization $now!$ with the global urgent channel now and we add a process Now that can only do $now?$ synchronizations (see Fig. 2).

This solution works because by itself the Now process cannot perform any steps, so time delay is possible. As soon as the guard $x == 5$ becomes true, the synchronization is enabled and time cannot proceed until either some action makes the guard false or the transition with the $now!$ is taken.

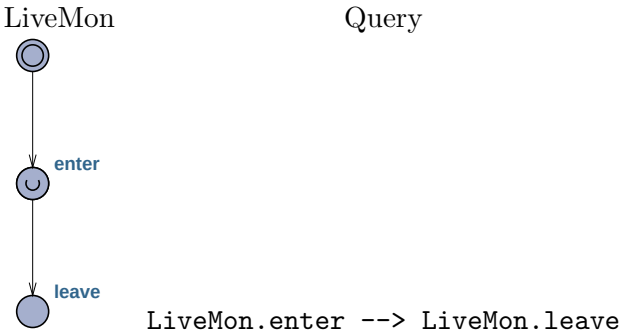
If we need more than one urgent reaction than, we can simply attach the $now!$ synchronization to all urgent reaction edges. A single copy of the Now process suffices.

3 Testing for livelocks

A livelock is an infinite sequence of steps, in which time does not proceed. To test if a model has live locks, one can simply put the LiveMon process in parallel with the model and verify the query `LiveMon.enter --> LiveMon.leave` (see Fig. 3).

Proof. If the model M contains a livelock then there is a sequence $M \rightarrow^{+} M' \rightarrow^{\omega}$, where in the sequence $M' \rightarrow^{\omega}$ time remains the same. Thus, we have the sequence $M \parallel LiveMon \rightarrow^{+}$

Figure 3: Testing for livelocks



$M' \parallel LiveMon \rightarrow M' \parallel LiveMon.enter \rightarrow^{\omega}$, where in the tail $M' \parallel LiveMon.enter \rightarrow^{\omega}$ LiveMon doesn't perform a step. Hence the query fails for $M \parallel LiveMon$. If the model M is livelock free and we put LiveMon in parallel then whenever LiveMon takes the transition to $LiveMon.enter$, there can be at most finitely many steps in which time does not proceed. So once the step to enter is taken, we can always take it voluntarily but if we try to wait as long as possible then eventually delay is the only possibility for the model and the step from enter to leave must be taken due to the urgent marking of enter.

4 Avoiding Livelocks a.k.a. Inserting Delays

The environment is often an automaton that can perform steps that are not restricted by time constraints. This may lead to a livelock. The easy way to avoid livelocks is to insert delays. Delays can be inserted into both edges and into states. In both cases, we need a clock x. To insert a delay into an edge with a condition c, a label a and updates u

$$\xrightarrow{C;a;u}$$

we add $x = 0$ and insert a node with constaint $x <= 1$ and add $x == 1$ to the outgoing edge:

$$\xrightarrow{C;a;u,x=0} \boxed{x \leq 1} \xrightarrow{x==1}$$

Adding a delay in a state is easier: just add $x = 0$ to the updates of all incoming edges and add $x \geq 2$ to the conditions of all outgoing edges:

$$\xrightarrow{C_1;a_1;u_1} (I) \xrightarrow{C_2;a_2;u_2} \Rightarrow \xrightarrow{C_1;a_1;u_1,x=0} (I) \xrightarrow{C_2\&\&x\geq 1;a_2;u_2}$$