Timed automata.

1 Introduction

We shall in the following assume a reader's first knowledge about the concept of state machines or finite automata. A finite automata (state machine/state automata) is generally a directed graph, where vertices are known as states and edges as transitions. Edge labels are referred to as events.

State automata are successfully applied to modeling a variety of systems appearing in real life, such as communication protocols, queueing systems and both SW and HW systems in general. With the following examples in hand we shall however illustrate how state automata may come short in modeling certain systems adequately.

1.1 Queueing example

Consider a queueing system where state value $q$ represents the number of elements queued up at some instant in time. The label set $L$ in this case includes 2 elements $\{a, d\}$ representing arrival to the queue and departure respectively. The STD of such a queueing system is shown in figure (1).

![STD of queueing system](figure1.png)

If we ask ourselves the question: how many items may simultaneously be queued? or equivalently: what is the reachable set of states?, then answer according to the model above is: infinitely many or all states.

Now suppose we know that service is fast, i.e. below some maximum value: $T_S$, and additionally that the number of arrivals are bounded within every time interval, lets say within every time interval of length $T_A$ at most one item may arrive. Then common sense tells us that if $T_A > T_S$, only a limited number of items may be queued up simultaneously and conversely if $T_A < T_S$ no limits could be given. However such properties are impossible to model in the basic state automata framework, since it allows no specific reference to time.
1.2 Example: Protocol SW

Consider the communicating state machines in figure (2). They represent a model of a call setup protocol acting in a telephone system. \( M_a \) acts as the caller and \( M_b \) as the callee. Every label postfixed "\_Succ" represents a successful communication between peers, whereas postfix "\_Fail" represents a communication fail. In other terms, labels with postfix "\_Succ" are common to the two machines, whereas postfix "\_Fail" indicate labels belonging to one machine only.

![Figure 2: Protocol modified with additional escape event.](image)

Labels

Conn_Lost_A

and

Conn_Lost_B
represents messages transmitted from lower layers indicating lost links. Lost link messages will immediately enforce the call control protocol to move to the idle state as shown in figure (2). This mechanism is introduced to the system to remove deadlocks in the system.

In the above automata lost link messages are implicitly assumed to be generated by some timeout mechanism, activated at message transmission and timing out after a fixed time interval, indicating that the link was probably lost. However nothing prevents the event

\texttt{Conn\_Lost\_A}

to take place immediately after

\texttt{Conn\_Req\_Succ}

forcing the caller to return to the idle state. Then after a while the callee replies with

\texttt{Conn\_Rep\_Succ}

but this event is not receivable for the caller in state idle. So the only path away is for the callee to issue a

\texttt{Conn\_Lost\_B}

event and return to its idle state. In this case no deadlock occurred but a call setup was failed due to premature timeout. Common sense engineering would set a timer value $T$ for generating timeout event

\texttt{Conn\_Lost\_A}

to be at least the worst case round trip delay between peers. In that case timeout would indeed mean that something failed in the underlying protocol layers. Call setup would also fail but in this case due to an acceptable reason.

2 Timed automata

To accommodate the need for direct reference to time the concept of timed automata has been developed. Related to each state transition various timers may be started, reset, kept running or stopped. Each timer counts up from zero and times out when reaching its timeout value defined at startup or reset.
In the queueing example a timer $\tau_S$ would be started with a timeout value $\leq T_S$, when the first item arrives to the queue. Timeout of $\tau_S$ generates a departure event $d$ to the automata resulting in a transition to a lower number of items in queue. If transition is made to some state $i > 0$, i.e. there are items awaiting service, it is immediately reset and some new timeout value $\leq T_S$ is set. After the first arrival an arrival timer $\tau_A$ is started with a timeout value $\geq T_A$. Timeout of $\tau_A$ generates an arrival event $a$ resulting in state transition to higher state value (yet another item in queue). After $\tau_A$ timeout it is immediately reset with a new timeout value $\geq T_A$. Now arrivals will be separated in time by at least $T_A$ and service of single items will last at most $T_S$, which captures the desired behaviour.

In the communication protocol example events

Conn\_Lost\_A

and

Conn\_Lost\_B

are generated by timers $\tau_A$ and $\tau_B$ started at transitions from idle to pending, each with a constant timeout value $T$ higher than the worst case round trip delay. Both $\tau_A$ and $\tau_B$ are stopped under transition from pending to connected. In this case however we have not modelled the effect of transmission delay, so our model flaw remains. An easy way to model a worst case round trip delay is to assign a timer $\tau_D$ to the event

Conn\_Rep\_Succ

$\tau_D$ is started when entering the pending state with a timeout value less than the worst case round trip delay.

2.1 Timeout values

In the above examples we introduced timers started at transition instants with various timeout values. Typically timeout values where specified to belong to some interval; either lower than some value, higher or some combination. Another way of assigning timeout values is through stochastic modelling. In that case timeout values are typically drawn randomly, independently according to some distribution.

An alternative representation of timers is through the concept of clocks. Clocks are positive real valued state variables, confined to certain parts of the state space. We say that the automate state space is augmented with a vector of clocks $c = [c_1, c_2, .., c_N]$ where $N$ is the number of clocks and
where each vector element represents a timer.

To capture the inherent property of time; *different clocks move equal distances in equal time*, clock vectors are confined to curves $c = [(t - t_1) (t - t_2) .. (t - t_N)]$ for $t \in [t_A, t_B]$, where $N$ is the number of clocks, $t$ is physical time, $t_i$ is the latest instant in time prior to $t$ where $c_i$ was reset/started. $t_A$ is the latest instant in time prior to $t$ where some clock was reset/started and $t_B$ is the earliest instant later than $t$ some clock was reset. The behaviour of clocks is illustrated in figure 3.

![Clock moves under time advance and discrete transitions.](image)

Figure 3: Clock moves under time advance and discrete transitions.

Thus at all times a number of clock each assume a positive real value, and at state transitions one or more clocks may be reset, i.e. $c_i = 0$.

To capture the idea of timeout we introduce the concept of invariants and guards:

- An invariant defines the acceptable values of the clock vector, where an automata is allowed to reside in some state.
- A guard defines the acceptable values of the clock vector, where some state transition is allowed.

In other words the automata may only reside in some state during a period of time where the clock vector assumes a value acceptable to that state and a transition may only take place during a period of time where the clock vector assumes a value acceptable to that transition. As a result invariants and guards are relational/logical expressions assigned to states and transitions respectively.
For the queueing example a departure clock $c_d$ and an arrival clock $c_a$ should be defined. The departure clock is reset at the first arrival and during transitions between states $i + 1$ and $i$, where $i > 0$. Timing in the queueing system is feasibly modelled in an expanded queueing automata, where the arrival process appears a one automate and the queueing system as another. Communication between arrival process and queueing system is thought the common event $a$, indicating an arriving item.

![Timed automata model of a queueing system.](image)

A minimum interarrival time $T_a$ is appropriately modelled by assigning a guard to the self loop transition of the arrival process $c_a \geq T_a$. A maximum service time $T_s$ is modelled by assigning an invariant $c_d \leq T_d$ to every state $i > 0$. $c_a$ is then reset during arrival and $c_d$ during service completion (departure).
For the call control protocol example we model link failure timeout by assigning a clock \( c_p \) to the pending state of the caller. Constant timeout values \( T \) are modelled by a combination of an invariant \( c_p \leq T \) assigned to the pending state and a guard \( c_p = T \) assigned to the timeout transition to the idle state. The media delay is modelled by assigning a clock \( c_d \) to the pending state of the callee. A varying delay in the interval \([T_A, T_B]\) is modelled by assigning an invariant \( c_p \leq T_B \) to the pending state and a guard \( c_p \geq T_B \) to the transition to state connected. The resulting timed automata is shown in figure 5.

![Timed automata model of call control protocol.](image)

Figure 5: Timed automata model of call control protocol.

### 3 Analyzing timed automata

For timed automata the state space is augmented with a vector of real positive clock values. The value of each clock may be decisive for the transitions possible in a given moment in time. Thus analysis of the reachable states in a timed automata should answer the question if some augmented state \([s, c]\) is reachable from some initial state. Since the state space is so called uncountable due to the real clock values, graph based algorithms some inadequate to solve the problem.

However if we assume timeout limits to be integer, a solution to the problem exists. In this case we define the following equivalence relation \( \rho \) on the space of clock values. (When two clock vector values \( c \) and \( c' \) are related we write \( c \rho c' \)):

\[
\begin{align*}
  c & \rho \ c' \\
  \downarrow \\
  7
\end{align*}
\]
\[
\begin{align*}
  n < c_i < m &\iff n < c'_i < m \\
n = c_i < m &\iff n = c'_i < m \\
n < c_i = m &\iff n < c'_i = m \\
n = c_i = m &\iff n = c'_i = m \\
\frac{c_i}{c_j} < \frac{c'_i}{c'_j} &\iff \frac{c'_i}{c'_j} < \frac{c'_i}{c'_j} \\
\frac{c_i}{c_j} = \frac{c'_i}{c'_j} &\iff \frac{c'_i}{c'_j} = \frac{c'_i}{c'_j}
\end{align*}
\]

where \(\text{frac}\) indicates fractional value.

At is easily verified that \(\rho\) is an equivalence relation on the space of positive real clock vectors \(R^N_+\) of some finite dimension \(N\). From algebra we know that an equivalence relation \(\rho\) induces a partition, in this case of \(R^N_+\). A partition is a set of disjoint subsets (equivalence classes) \(\{C_1, C_2, \ldots\}\) covering the set. If \(c \in C_i\) then \(C_i = \{c' \mid c \rho c'\}\). For a clock vector of dimension two, i.e. \(N = 2\) the partition is illustrated in figure 6.

![Figure 6: Equivalence partition of clock space.](image)

The relation \(\rho\) is particularly interesting for timed automata since the following property is readily observed. If and only if \(c \rho c'\) we have the following: if there exists a real positive \(dt\) so that \(c + dt \in C_i\) then a real positive \(d\tau\) exists so that \(c' + d\tau \in C_i\). In other words related clock vectors visit the same equivalence classes when time elapses. Also if some logical expression (invariant or guard) \(L(c)\) is
true then $L(c')$ is also true and vice versa.

Also if for some discrete transition $c$ is reset to $R(c) \in C_j$ then $R(c') \in C_j$ also.

Consider two states $[s \ c]$ and $[s \ c']$ where $c, c' \in C_i$ then we wish to find the set of state reachable from these states, either by one discrete state transition or by letting time elapse. The reachable sets by one discrete transition are equal since guards are equivalently true for the to clock values.

Now if $[s \ c* \in C_j]$ is reachable from $[s]$ some time advance $dt$ brings $[s]$ to $[s \ c + dt \in C_j]$ and no invariant preclude clock values $c + t$ for $t \in [0, dt]$ in $s$. In other words $c + t$ only visited equivalence classes $C_i, ..., C_p, ..., C_j$ not precluded by invariants in $s$ on its way from $C_i$ to $C_j$. Now elapsing time from $[s]$ by some $d\tau$ so that $c' + d\tau \in C_j$ brings us through the same sequence $C_i, ..., C_p, ..., C_j$ of equivalence classes as shown for $N = 2$ in figure 7. Thus no invariants are violated during this travel and $[s \ c']$ may reach some $[s \ c'' \in C_j]$. For some timed automata $M$ we define the automata $\mathcal{M}$ having augmented states $[s \ C]$ where $C$ is an equivalence class of $\rho$. This is the so called region automate, where equivalence classes are called regions. Transitions are defined for $\mathcal{M}$ in the following way:

- A transition exists between states $[s \ C]$ and $[s' \ C']$ in $\mathcal{M}$ if a transition exists between $[s \ c]$
and \( [s' R(c') \in C'] \) for all \( c \in C \).

- A transition exists between states \([s C] \) and \([s C'] \) in \( \mathcal{M} \) if for all \( c \in C \), \( dt > 0 \) exists so that \([s c + dt \in C'] \) and no invariants are violated in \( s \) by \( c + t \) for \( t \in [0, dt] \).

- No other transitions exist in \( \mathcal{M} \)

Now we are in the position to express the following relation between the timed automata \( \mathcal{M} \) and the associated region automata \( \mathcal{M}_C \):

- Assume a path \([s_1 C_1], [s_2 C_2], \ldots, [s_n C_n] \) exists in \( \mathcal{M} \) then a path \([s_1 C_1], [s_2 C_2], \ldots, [s_n C_n] \), where \( c_i \in C_i \) exists in \( \mathcal{M} \).

- Assume a path \([s_1 c_1], [s_2 c_2], \ldots, [s_n c_n] \) exists in \( \mathcal{M} \) then a path \([s_1 C_1], [s_2 C_2], \ldots, [s_n C_n] \), where \( c_i \in C_i \) exists in \( \mathcal{M} \).

which is proved by the following arguments:

- if \( \ldots, [s_i C_i], [s_j C_j], \ldots \) where \( s_i = s_j \) and a path \([s_1 c_1], [s_2 c_2], \ldots, [s_n c_n] \) exists in \( \mathcal{M} \), then a time advance brings \([s_i C_i] \) to \([s_i C_j] \) for some \( C_j \in C_j \) and \([s_1 c_1], [s_2 c_2], \ldots, [s_j c_j] \) exists in \( \mathcal{M} \), if \( \ldots, [s_i C_i], [s_j C_j], \ldots \) where \( s_i \neq s_j \) and \([s_1 c_1], [s_2 c_2], \ldots, [s_i c_i] \) exists in \( \mathcal{M} \) then one discrete transition (with appropriate clock resets) brings \([s_i c_i] \) to \([s_j c_j = R(c_i)] \) for some \( c_j \in C_j \) and in turn \([s_1 C_1], [s_2 C_2], \ldots, [s_j c_j] \) exists in \( \mathcal{M} \).

- if \( \ldots, [s_i C_i], [s_j C_j], \ldots \) where \( s_i = s_j \) and \([s_1 C_1], [s_2 C_2], \ldots, [s_i C_i] \) exists in \( \mathcal{M} \), then a time advance brings then a time advance brings \([s_i C_i] \) to \([s_i C_j] \) for all \( C_i \in C_i \) and some \( C_j \in C_j \) so a transition exists between \([s_i C_i] \) and \([s_i C_j] \) in \( \mathcal{M} \) and in turn \([s_1 C_1], [s_2 C_2], \ldots, [s_j C_j] \) exists in \( \mathcal{M} \), if \( \ldots, [s_i C_i], [s_j C_j], \ldots \) where \( s_i \neq s_j \) and \([s_1 C_1], [s_2 C_2], \ldots, [s_i C_i] \) exists in \( \mathcal{M} \) then one discrete transition (with appropriate clock resets) brings \([s_i C_i] \) to \([s_j C_j = R(c_i)] \) for some \( C_j \in C_j \), so a transition between \([s_i C_i] \) to \([s_j C_j] \) exists in \( \mathcal{M} \) and in turn \([s_1 C_1], [s_2 C_2], \ldots, [s_j C_j] \) exists in \( \mathcal{M} \).

All together, if we only care about which discrete states and clock regions augmented states \([s C] \) visit during the life time of a timed automata, we may as well analyze its associated region automata. Thus reachability, deadlock, lifelock and progress properties of a timed automata are preserved in its associated region automata.
4 Tool Assistance

In general state machine system modelling results in a number of communicating state machines. As the state set of the parallel composition is generally the cartesian product of the individual state sets, the resulting number of states may be huge. Although the reachable set may only be a small fraction of the entire cartesian product, manually constructing the parallel composition and subsequently analyzing it is non feasible for almost all practically relevant systems. However a number of tools exist, which may be of assistance. Such tools typically offers some form of definition interface either graphical or textual, into which the state machines may be defined. After defining state machines the tools typically construct only so much of the reachable set as necessary to answer questions about deadlocks, livelocks and progress.

One such tool is UPPAAL, where the definition interface is graphical. Timed automata are drawn as shown above and various properties may be verified.