Stochastic Specification and Verification

Jeremy Bryans and John Derrick
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Phone: + 44 1227 764000, Email: J.Derrick@ukc.ac.uk.)

Abstract

Modern distributed systems include a class of applications in which non-functional requirements are important. In particular, these applications include multimedia facilities where real time constraints are crucial to their correct functioning. In order to specify such systems it is necessary to describe that events occur at times given by probability distributions. Stochastic process algebras have emerged as a useful technique by which such systems can be specified and verified.

However, stochastic descriptions are very general, in particular they allow the use of general probability distribution functions, and therefore their verification can be complex. In this paper we define a translation from stochastic process algebras to timed automata. By doing so we aim to use the simpler verification methods for timed automata (e.g. reachability properties) for the more complex stochastic descriptions.

Keywords: Distributed systems; stochastic process algebras; timed automata.

1 Introduction

In this paper we define and verify a translation from a stochastic process algebra to timed automata. The reason for doing so is to support the specification and verification of non-functional properties in distributed multimedia systems.

The advent of distributed multimedia applications such as video conferencing, radio over the internet etc, place great demands on the specification and design of such systems because of the need to describe and verify non-functional requirements. These non-functional requirements typically involve real time constraints such as placing bounds on end-to-end latency, and are often called *Quality of Service (QoS)* requirements because they reflect the overall quality of delivery as opposed to the functional aspects.

In order to specify and verify such constraints it is necessary not only to be able to describe deterministic timing concerns but also probabilistic and stochastic systems. That is, in practice timings cannot be assumed to be fixed (deterministic timings) but events can occur at different times with particular probabilities. Therefore it is necessary to describe timings that occur according to certain *probability distributions*. For example, in a network specification it is not sufficient to assume that the packet deliveries arrive at fixed predetermined times, instead we need to model the system where they might might arrive at times determined by an exponential distribution.

There are now a number of techniques which can be used to describe such systems, e.g. Petri-nets [13], generalised semi-Markov processes [8] and stochastic process algebras etc. In this paper we will consider stochastic process algebras, for which a number of formalisms, techniques and tools are available, e.g. PEPA [10], TIPP [9], EMPA [3] PA_{GS} [11] and SPADES [7]. Stochastic process algebras offer a number of advantages over other techniques such as Petri-nets. In particular, stochastic process algebras are compositional and allow a specification to be built as a number of smaller components composed together (e.g. using the parallel composition operator). Such compositionality is important if the specification and verification techniques are to scale. In addition, stochastic process algebras allow the study of both functional and non-functional requirements within the same description, giving a more realistic view of overall performance than, say, a queueing theory description of the problem.

In this paper we focus on the stochastic process algebra SPADES. The reason for this is twofold. The primary motivation is that SPADES supports not just exponential distributions but general distributions. The issue here is the following. A stochastic process algebra associates a distribution function F with an action a so that $a_F;\ P$ describes an action prefix where the probability of the time delay after which a can happen is determined by the distribution function F. Because of the interleaving semantics of most process algebras and the low complexity of verification (exponential distributions are memoryless, this greatly simplifies verification of properties) most process algebras restrict themselves to allowing only exponential distributions for F.

However, this is unrealistic in practice and it is necessary in general for F to range over any distribution (e.g. uniform, gamma, deterministic etc). For example, it is often assumed that packet lengths are exponentially distributed. However, in reality this is not the case, rather they are either of constant length (as in ATM cells [14]) or they are uniformly distributed with minimum and maximum size (as in Ethernet frames [14]). SPADES allows this generality. It does so by using *stochastic automata* as its underlying model, and this forms the second motivation for its use here. Stochastic automata are based on timed automata [2], which enable a link to be formed into the extensive work on verification for timed automata. This is useful for the following reason. Although general distributions are important for describing complex distributed systems, they do make a range of verification tasks more complex. In particular, the move from memoryless exponential distributions to arbitrary general distributions has profound implications for the complexity of verification techniques such as model checking.

However, there are a collection of properties, such as reachability properties, which can be verified for general stochastic systems by looking at a simplified underlying model. It is for this reason that we define a translation from stochastic process algebras to timed automata. By doing so we aim to support the verification of a range of properties by using techniques developed for timed automata.

The structure of the paper is as follows. In Section 2 we introduce SPADES and stochastic automata illustrated by a simple specification of a multimedia stream. We also briefly review timed automata and the particular model we use, timed automata with deadlines [5]. In Section 3 we define the translation of stochastic automata into timed automata with deadlines. This translation is verified in Section 4, and we conclude in Section 5.

2 Stochastic specification and verification

Multimedia distributed applications are now commonplace. They are also difficult to specify and verify. One reason for this is a number of new requirements that now have to be considered when building such a system. For example, interactions between components in such systems are often continuous, e.g. the flow of packets in a radio over the internet application. These continuous interactions lead to QoS constraints which, for example, place acceptable bounds on the various timing aspects of this packet flow (latency, jitter, throughput etc).

Stochastic process algebras offer a promising method by which we can specify and verify these systems in a compositional manner. In this paper we will use the SPADES process algebra because it provides support for general distribution functions as well as providing a link into (timed) automata. We will use the latter as our verification strategy.

This is attractive because there are a range of verification tools available for timed automata, e.g. the Kronos and UPPAAL model checkers, whilst verification for stochastic techniques with generalised distribution functions has proved difficult. In particular, the non-memoryless nature of generalised distribution functions means that stochastic model checking is currently feasible for only very small state spaces. With a non-memoryless distribution recalculations of conditional probabilities need to be undertaken at each state explored in the state space, placing severe bounds on the feasibility with the current techniques.

The link with automata also allow stochastic process algebras to be used within other multi-paradigm specification frameworks. For example, timed automata have been used as an underlying formalism to bring together a variety of specification approaches [4]. The use of stochastic automata as a basis for stochastic process algebras, and the link developed in this paper between stochastic automata and timed automata should allow the integration of SPADES into such multi-paradigm approaches, this offers clear advantages when specifying complex systems.

2.1 Stochastic process algebras and automata

The stochastic process algebra SPADES and its underlying stochastic automata model use *clocks* to trigger events. Clocks are variables which take a value set according to a given probability distribution function. After being set a clock counts down, and when the clock reaches zero it enables certain events (or transitions) in the model. To model this situation the syntax of SPADES includes the following (where a is an action, p is a process and C is a set of clocks.)

$$p ::= \mathit{stop} \quad | \quad a;\ p \quad | \quad C \mapsto p \quad | \quad p + p \quad | \quad \{\!|\, C \,|\!\}p \quad | \quad p \parallel_A p$$

This is a standard process algebra extended with a clock setting operation and a triggering condition. The former, $\{\!|\, C \,|\!\}p$, sets the clocks in C to a value chosen according to their respective probability distribution functions. $C \mapsto p$ is the triggering condition: the process p becomes enabled as soon as all the clocks in C expire.

2.1.1 A multimedia example in SPADES

As an example, consider the specification of a simple multimedia stream. It has three components: a *Source* process, a *Sink* process and a *Channel*. The *Source* generates a sequence of packets, which are transmitted by the *Channel* to the *Sink*, which displays them. We assume that the *Channel* is unreliable, and may lose packets. We also assume that the *Sink* process is impatient, and if it does not receive packets at a sufficiently fast rate it will timeout.

These components can be written in SPADES as

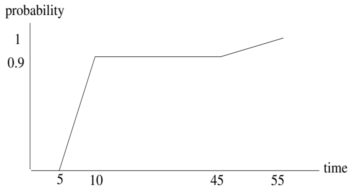
$$\begin{array}{rcl}
\mathit{Source} & = & \{\!|\, tr \,|\!\}\{tr\} \mapsto \mathit{trans};\ \mathit{Source} \\
\mathit{Sink} & = & \{\!|\, d, t' \,|\!\}(\mathit{receive};\ \{d\} \mapsto \mathit{display};\ \mathit{Sink}\ +\ \{t'\} \mapsto \mathit{timeout};\ \mathit{Stop}) \\
\mathit{Channel} & = & \mathit{Channel}_0 \\
\mathit{Channel}_0 & = & \{\!|\, t \,|\!\}\{t\} \mapsto \mathit{trans};\ \{\!|\, t, r, l \,|\!\}\mathit{Channel}_1 \\
\mathit{Channel}_i & = & \left\{ \begin{array}{l}
\{t\} \mapsto \mathit{trans};\ \{\!|\, t \,|\!\}\mathit{Channel}_{i+1} \\
+ \\
\{r\} \mapsto \mathit{receive};\ \{\!|\, r, l \,|\!\}\mathit{Channel}_{i-1} \\
+ \\
\{l\} \mapsto \mathit{loss};\ \{\!|\, r, l \,|\!\}\mathit{Channel}_{i-1}
\end{array} \right.
\end{array}$$

and the *Stream* can then be given as the parallel composition of these components:

$$\mathit{Stream} = \mathit{Source} \parallel_{\{\mathit{trans}\}} \mathit{Channel} \parallel_{\{\mathit{receive}\}} \mathit{Sink}$$

The *Source* process begins by setting the clock tr according to the probability function F_{tr} and defined as (the unit of time being *ms*):

$$
\begin{aligned}
F_{tr}(t) &= 0; && \text{if } t \leqslant 5\\
&= 0.18(t-5); && \text{if } t \in (5,10]\\
&= 0.9; && \text{if } t \in (10,45]\\
&= 0.01(t-45)+0.9; && \text{if } t \in (45,55]\\
&= 1; && \text{otherwise}
\end{aligned}
$$



This represents a situation where messages made up of several packets are being transmitted. If a packet is generated it is reasonably likely that a further packet will be generated in the near future, (between 5 and 10ms, in this example) but if the following packet is not generated (if, for example, the previous packet marked the end of a message) then there will be a time interval of approximately 50ms (between 45 and 55ms) before the start of the next message.

When the clock *tr* expires, the action *trans* is enabled, and when it fires the process repeats itself.

In the process *Sink*, the clocks d and t' are set initially. The clock d controls the rate of display of packets, and t' controls the timeout behaviour. The distributions F_d and F_t' are given by

$$
\begin{aligned}
F_d(t) &= 0;\ \text{if } t < 25 \\
&= 1;\ \text{otherwise}
\end{aligned}
\qquad\qquad
\begin{aligned}
F_{t'}(t) &= 0;\ \text{if } t < 50 \\
&= 1;\ \text{otherwise}
\end{aligned}
$$

Both these functions represent *deterministic* timing. Function F_d says that a packet cannot be displayed until precisely 25ms after the previous one. At any time less that 25ms a *display* action is not possible; at every time greater that or equal to 25ms the *display* action is enabled. If a *display* action is performed, the process will recurse again to *Sink*.

In the *Sink* process, the *receive* action does not have any clock dictating its rate, and therefore if the *Sink* process was operating on its own the *receive* action would happen immediately. However, this action has to synchronise with the receive action in *Channel*, and the clock r will then influence its rate.

If, after displaying a packet, the *Sink* does not receive another packet within 50ms, it will *timeout* and stop. It will also *timeout* if it does not receive a packet within 50ms of initialisation.

The *Channel* process contains the three clocks t, r and l, which control the rates of the *trans*, *receive* and *loss* actions respectively. The process is presented as a parameterised set of recursive equations. The functions F_t and F_l are defined as

$$
\begin{aligned}
F_t(t) &= 0;\ \text{if } t \leqslant 9 \\
&= 0.5(t-9),\ \text{if } t \in [9,11] \\
&= 1;\ \text{otherwise}
\end{aligned}
\qquad\qquad
\begin{aligned}
F_l(t) &= \tfrac{t}{200};\ \text{if } t \leqslant 200 \\
&= 1;\ \text{otherwise}
\end{aligned}
$$

and the probability distribution F_r is a bounded normal curve, centred at 50, truncated at 25 and 75, and normalised.

The function F_t represents the time *Channel* takes to initialise, and the "recovery time" necessary after each *trans* action. Initially, when no packets are contained in *Channel*, the clock t is set according to F_t, and the *trans* action is enabled when clock t has expired. When the *trans* action fires, each of the three clocks are set. The function F_r represents the time taken for the *Channel* to move a message from *Source* to *Sink*.

The loss behaviour is modelled by function F_l. A packet may be lost from *Channel* at any point up to 200ms after it has been received. $\square$

2.1.2 Stochastic Automata

To understand this specification we can think of it as defining a stochastic automaton. Stochastic automata generalise timed automata by using stochastic clock settings instead of strictly deterministic timings in a timed automaton. In fact, stochastic automata and SPADES are equally expressive and [7] gives the mapping between SPADES and stochastic automata in detail. Therefore for the remainder of this paper it suffices to work at the level of stochastic automata, for any process algebraic specification has an equivalent stochastic automata representation.

Definition 1 A *stochastic automaton* is a structure $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \twoheadrightarrow, \kappa, F)$ where:

- $\mathcal{S}$ is a set of *locations* with $s_0 \in \mathcal{S}$ being the *initial location*, $\mathcal{C}$ is the set of all *clocks*, and $\mathbf{A}$ is a set of *actions*.
- $\twoheadrightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \mathcal{C}) \times \mathcal{S}$ is the set of *edges*. If s and s' are states, a is an action and C is a subset of $\mathcal{C}$, then we denote the edge $(s, a, C, s') \in \twoheadrightarrow$ by $s \xrightarrow{a,C} s'$ and we say that C is the *trigger set* of action a. We use $s \xrightarrow{a} s'$ as a shorthand notation for $\exists\, C.s \xrightarrow{a,C} s$.
- $\kappa : \mathcal{S} \rightarrow \mathbb{P}_{\text{fin}}(\mathcal{C})$ is the *clock setting function*, and indicates which clocks are to be set in which states, where $\mathbb{P}_{\text{fin}}(\mathcal{C})$ is the powerset of clocks.
- $F : \mathcal{C} \rightarrow (\mathbb{R} \rightarrow [0, 1])$ assigns to each clock a *distribution function* such that, for any clock x, $F(x)(t) = 0$ for $t < 0$; we write F_x for $F(x)$ and thus $F_x(t)$ states the probability that the value selected for the clock x is less than or equal to t.

Notice that each clock $x \in \mathcal{C}$ is a random variable with distribution F_x. □

This is the definition of stochastic automata presented in [7], but in this paper we will be less concerned with determining probabilities, and more concerned with determining whether or not a particular value is possible when a clock is initially set. For this reason we consider the derivatives of these functions F'_x, since if $F'_x(t) > 0$, then t is a possible initial value for the clock x.

For simplicity, we consider only functions whose derivative is made up of a finite number of left/right closed intervals; that is, we consider only functions F such that

$$\{t \mid F'(t) > 0\} = \bigcup_{1 \leqslant j \leqslant n} [g_j, h_j]$$

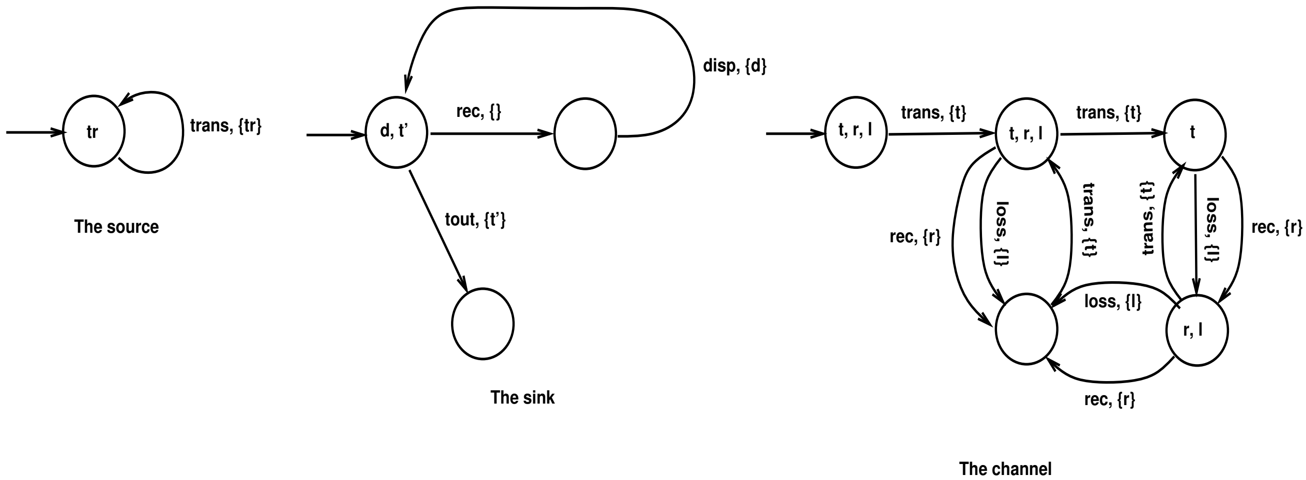
where $[g_j, h_j]$ is a left/right closed interval and n is the number of intervals in the derivative. In practice this is not a severe restriction (e.g. [1] imposes the same restriction).

The *range* of a clock x is given by the set $\{t \mid F'_x(t) > 0\}$. We also define $\max(x)$ as the maximum possible initial value of x, given by h_n, where n is the number of intervals in the range of x and which may be infinity. The minimum possible initial value $(\min(x))$ is given by g_1 and must be finite. Note that the upper and lower bounds of an interval may be equal, thus allowing deterministic timing. We assume that all clocks are initially set to some value within their range.

We can now give the stochastic automata for the example presented above.

2.1.3 The multimedia example as a stochastic automata

The stochastic automata for this example are depicted below. Like the SPADES description above the overall automaton is composed of three parts: a sink, a source and a channel. These are first given separately, followed by their composition. In the figures, the locations are given as circles with the clocks reset at that location contained within each circle. The edges of the automata are given as arrows between the locations, with the initial state represented by the small ingoing arrow.



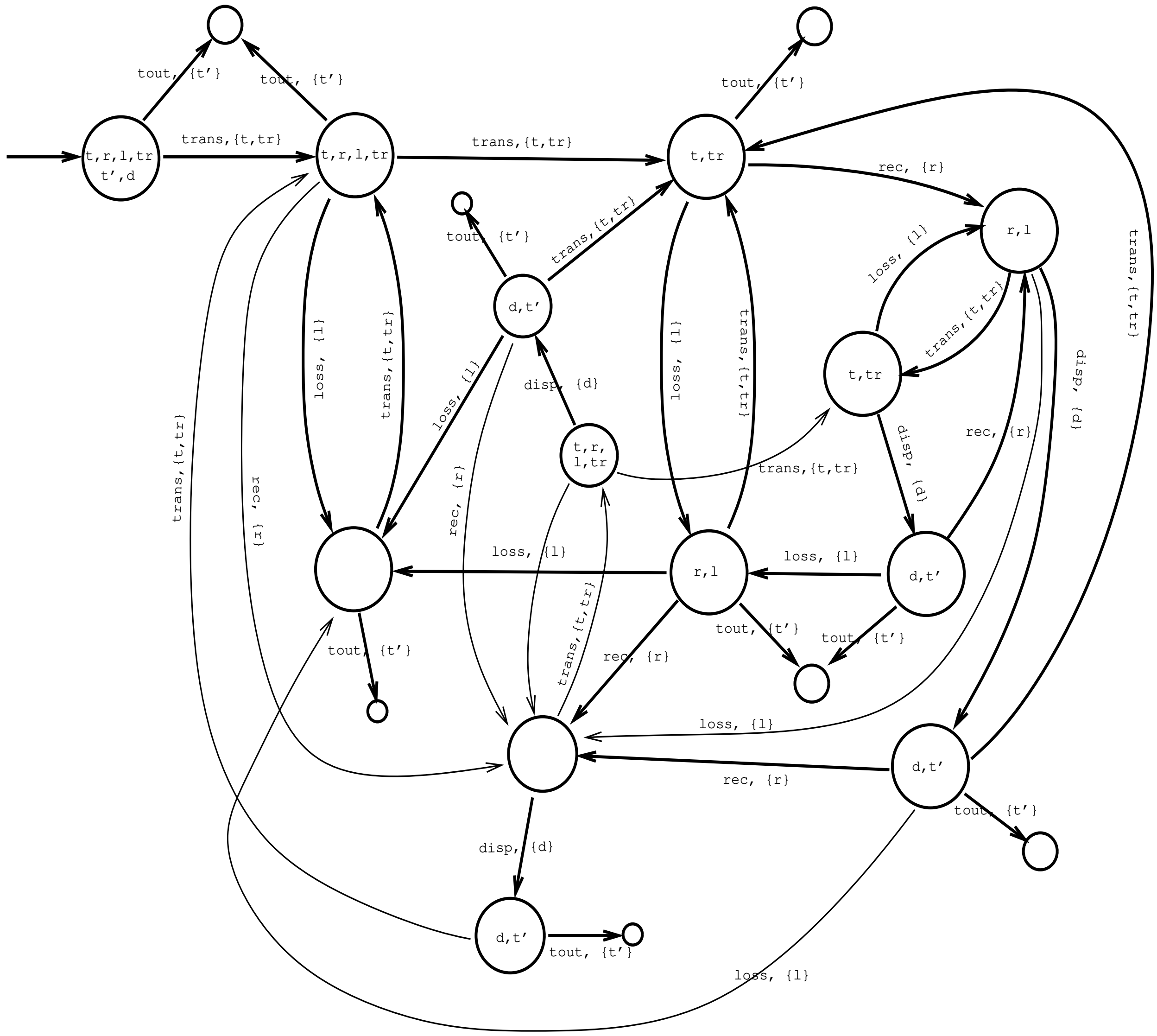
The *source* automaton generates the packets which make up the stream, and are separated according to the probability function F_{tr}.

The *channel* automaton synchronises with the *trans* action to accept a packet. This happens according to the probability distribution F_t. When a single packet resides in the channel: it can be passed on to the *sink* automaton via the *rec* action; lost via the *loss* action; or joined by another packet via the *trans* action. When two packets reside in the channel then either one of them will be lost, or one will be passed on to the *sink* automaton.

The *sink* automaton can receive a packet (by synchronising with *channel* on *rec*)in which case it displays it, or it can timeout.

The composition of stochastic automata is similar to the composition of timed automata, and is explained in detail in [7]. Component automata may proceed independently, performing the actions for which they alone are responsible, or they may synchronise on combined actions. With independent actions, the subsequent state resets only the clocks reset by the component automaton. With combined actions, the subsequent state resets the clocks reset by all participating automata.

The composition of the three automata above is given by the automaton below. Note that the smaller circles, entered after a timeout, are actually capable of performing further actions, since the *Channel* itself does not timeout, and may continue to perform *trans* and *loss* actions. However, we do not incorporate that extra behaviour in the diagram, to avoid cluttering the presentation.



2.2 Timed automata

Timed automata, as typified by the UPPAAL tool [12], represent one of the major approaches to specifying real time systems. Like a stochastic automaton, a timed automaton consists of a set of clocks and locations. The clocks proceed at the same rate and measure the amount of time that has elapsed since they were reset. Locations (or states) can have invariants attached to them. If a timed automaton is in a particular location, the invariant must be true. This property can be used to make actions *urgent*, by insisting that the automaton must have exited the state by a certain time.

In this paper we use timed automata with deadlines (TAD), as presented in [5], which differ slightly from the standard presentation of timed automata. The essential approach in timed automata with deadlines is to associate deadlines with transitions instead of placing invariants on states. Therefore transitions consist of 4-tuples (a, g, d, r), comprising of an action a, guard g, deadline d and set of clocks r. Guards and deadlines are predicates parameterised by the values of the clocks. The guard states when a transition is enabled (i.e. when it may be taken), and if the deadline is true then the transition must be taken. When the transition occurs the clocks in r are reset to zero. The constraint $d \Rightarrow g$ is assumed to hold. This means that if in a state time cannot progress (because a deadline is true), then the action is also enabled. This prevents timelocks from occurring. This issue is discussed in detail in [5]. We can give the following definition.

Definition 2 Formally a timed automaton with deadlines (TAD) consists of the following.

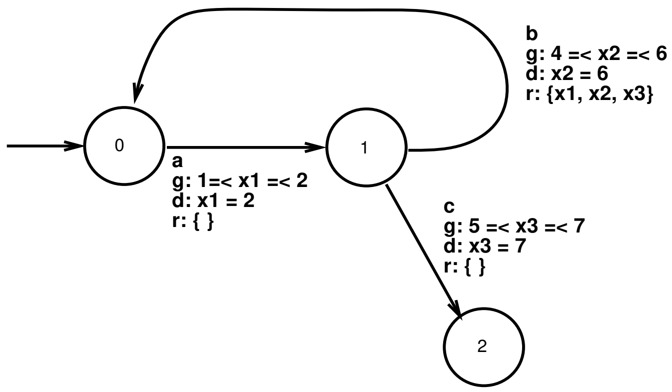
- A discrete labelled transition system $(Z, \rightarrow, A)$ where
  - Z is a finite set of discrete states
  - A is a finite set of actions
  - $\rightarrow \subseteq Z \times A \times Z$ is an untimed transition relation
- A set $X = \{x_1, \ldots, x_n\}$ of non-negative real valued variables called *clocks*.
- A labelling function h mapping untimed transitions into timed transitions: $h(z, a, z') = (z, (a, g, d, r), z')$ where
  - g and d are the *guard* and *deadline* of the transition. Guards and deadlines are predicates p defined by the following grammar:

    $$p ::= x\#c \mid p \wedge p \mid p \vee p$$

    where $x \in X$, $c \in \mathbb{R}_{\geqslant 0}$ and $\# \in \{\leqslant, \geqslant\}$. We require $d \Rightarrow g$.
- r is a set of clocks to be reset to zero. □

The clocks in a TAD are always reset to zero and count upwards at the same rate. This is in contrast to the clocks in stochastic automata, which are set to some value in $\mathbb{R}_{\geqslant 0}$ according to their probability distribution function, and count downwards. We assume in this work that all clocks in a TAD are initially set to zero. This is a simplification of the work in [5] where clocks may take any values initially, but we do not need that generality in defining a correct translation.

As an example of a timed automaton with deadlines consider the following specification.



The timed automaton depicted begins in state 0. All clocks x_i are initially set to zero. In state 0 the transition a becomes possible as soon as x_1 reaches 1, and remains possible until x_1 reaches 2. At that time the deadline becomes true, and action a is forced to happen if it hasn't already been taken.

When state 1 is entered, clock x_2 will already have some value between 1 and 2, since it has been counting upwards in synchronisation with clock x_1. Clock x_2 allows action b between time 4 and time 6, and insists on it (via the deadline) at time 6. However, at time 5, clock x_3 enables action c, and so between times 5 and 6 the choice between b and c is non-deterministic. If action b does occur, the three clocks x_1, x_2 and x_3 are reset to zero, and the process begins again from state 0. If action c occurs the timed automaton enters state 2 and no further transitions are possible. Note that clock x_3 cannot progress beyond time 6, because at that time action b is forced and x_3 is reset to zero. □

A range of tools and techniques are available for timed automata which can support various verification activities. One of the most successful techniques has been model checking, where a system can be checked to see if a particular property holds. This is achieved by representing the property as a formula in a propositional temporal logic (e.g. CTL), and the model checker automatically compares this with a state-transition graph of the systems behaviour. Model checkers now exist for a range of specification paradigms including both real-time and probabilistic systems.

UPPAAL is a good example of a real-time model checker, where the system under consideration is represented as a timed automaton. UPPAAL includes both a simulator and a model checker which can check the reachability properties of the system. If a property does not hold in a given system, UPPAAL provides an example trace which can be fed into the simulator for further analysis. There are many alternative model checking tools and techniques, e.g. Kronos [15].

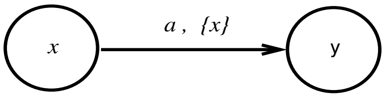
By providing a mapping from stochastic automata to timed automata, we aim to reuse this existing technology to enable reachability analysis to be performed on stochastic automata specifications.

3 Translating stochastic to timed automata

In this section we define the mapping from stochastic to timed automata. The mapping is relatively straightforward and intuitive, and we illustrate the translation with our running example. The proof of correctness of the mapping is more technical, and is verified in Section 4.

The translation mapping is designed to preserve precisely the behaviour that is necessary to verify reachability properties, and remove the remaining redundant information. In particular, because the reachability analysis looks for reachable states and not the probability of reaching those states, we can remove the probabilistic element and replace it by non-deterministic timing information.

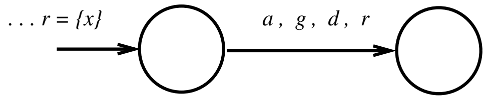
For example, given a stochastic automaton fragment



where

$$\begin{aligned} F_x(t) &= 0; \text{ if } t < 1 \\ &= 2t - t^2; \text{ if } t \in [1, 2] \\ &= 1; \text{ otherwise} \end{aligned}$$

the pertinent information is that the clock x can be set to any value between 1 and 2. The actual probabilities do not matter, so that for our purposes an equivalent timed automaton would be the following:



In this description we have used the same action name a, and $g : x \in [1, 2]$, $d : x = 2$, and $r : \{y\}$.

This timed automaton has the following behaviour. The guard g specifies the times at which the transition may be taken, which here are the values between 1 and 2. It must be taken by the deadline $d : x = 2$. When it is actually taken is non-deterministically chosen between these limits. It is in this sense that the stochastic information in the probabilistic distribution function has been replaced by a non-deterministic choice of actual time determined by the interplay of the guards and deadlines. In general, the guards and deadlines are determined by more complex formulae than suggested by this small fragment. The reasons for this are given after the definition.

From this example it should be clear how we will define the translation. Each stochastic automaton will be mapped to a timed automaton with the same number of locations and the same action label set. Each transition in the stochastic automaton will be mapped to a transition in the timed automaton. For each stochastic automaton clock there is a corresponding timed automaton clock. However, the stochastic information represented in the probability distribution associated with each stochastic automaton clock becomes embedded as deadlines and guards. Finally, an appropriate initial valuation for the timed automaton has to be given, and this is drawn from a possible initial valuation in the stochastic automaton.

The definition of the mapping can thus formally be given by the following [1].

Definition 3 *Translating a SA into a TAD*

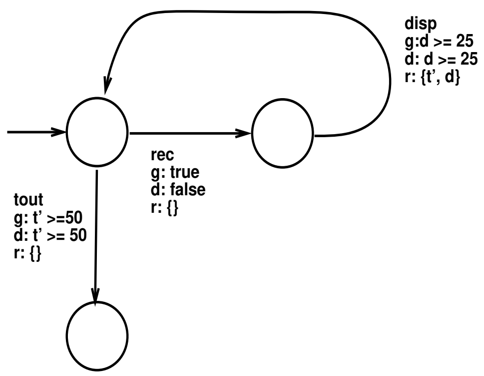
Let $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \dashrightarrow, \kappa, F)$ be a stochastic automaton. This automaton is mapped to the timed automaton $(Z, \longrightarrow_T, A)$ where

- $Z = \mathcal{S}$
- $A = \mathbf{A}$
- $\longrightarrow_T$ is the transition relation $\dashrightarrow$ with the clocks removed, i.e.
  $\longrightarrow_T \subseteq Z \times A \times Z$ where $\longrightarrow_T = \{(z, a, z') \mid \exists\, C_a.(s, a, C_a, s') \in \dashrightarrow \wedge\, s = z \wedge s' = z'\}$
- The set X contains clock variables, labelled x_i and indexed as the SA variables.
  $X = \{x_i \mid \forall\, c_j \in \mathcal{C}.\, \exists\, x_i.i = j\}$
- $h(z, a, z') = (z, (a, g, d, r), z')$ where C_a is the *trigger set* for action a and
  - $g = (\bigwedge_{c_i \in C_a} u(i) \geqslant \min(c_i)$
    $\wedge$
    $\bigvee_{c_i \in C_a} u(i) \in \mathrm{ran}(c_i))$
    $\vee$
    $\bigwedge_{c_i \in C_a} u(i) \geqslant \max(c_i)$
  - $d = \bigwedge_{c_i \in C_a} u(i) \geqslant \max(c_i)$
  - $r = \kappa(s')$
- The initial valuation for the TAD (u_0) is the valuation with each clock set to zero.

$$\forall\, x_i \in X.u_0(i) = 0$$

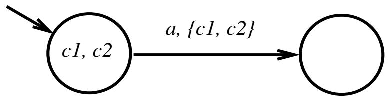
To see this in practice let us consider the example stochastic automata given in Section 2.1.3. The Sink stochastic automaton will translate to the following timed automaton.

[1] This definition was formulated by Pedro D'Argenio [6].



To understand how the definitions for the guards and deadlines have been chosen, first consider a single transition as depicted in the stochastic automaton on page 9. The clock x in the resultant timed automaton with deadlines is set to zero when the clock x in the stochastic automaton is nondeterministically set to some value within its range. Thus when the stochastic automaton clock reaches zero, (and action a fires) the clock in the TAD will be at some value within the range of x, i.e. $u(x) \in \mathrm{ran}(x)$.

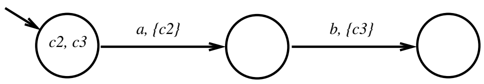
Now consider the following fragment



If $\mathrm{ran}(c_1) = [2,4] \cup [7,8]$ and $\mathrm{ran}(c_2) = [4,5] \cup [8,9]$ then it is not possible for action a to fire at time 3, since clock c_2 cannot have expired by this time. The earliest time at which a can fire is time 4; the greater of the two minimums, which we express in general as $\bigwedge_{c_i \in C_a} u(i) \geqslant \min(c_i)$, where C_a is the trigger set for each action a.

Also, note that it is not possible for a to fire at time 6, because neither clock can be set to this value. We must therefore insist that at least one clock be within range in order for the action to fire, i.e. $\bigvee_{c_i \in C_a} u(i) \in \mathrm{ran}(c_i)$.

Next, consider the fragment



where $\mathrm{ran}(c_3) = [2,3]$.

The clock c_3 will certainly have expired before the second state is reached, and so in the stochastic automaton the action b will fire as soon as the second state is entered. In the timed automaton, we therefore require the guard to continue to be true even if all clocks have passed their maximum values, and this gives us the full definition of the guard

$$
\begin{array}{l}
(\bigwedge_{c_i \in C_a} u(i) \geqslant \min(c_i) \\
\wedge \\
\bigvee_{c_i \in C_a} u(i) \in \mathrm{ran}(c_i)) \\
\vee \\
\bigwedge_{c_i \in C_a} u(i) \geqslant \max(c_i)
\end{array}
$$

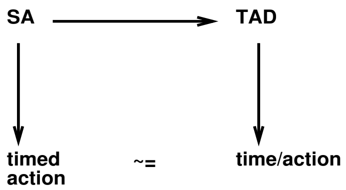
As for the deadline, we can only be certain that the stochastic automaton action will fire if all the clocks have expired. This translates to the timed automaton as insisting that all clocks have passed their maximum value, and so the deadline is

$$\bigwedge_{c_i \in C_a} u(i) \geqslant \max(c_i)$$

and we also have that $d \Rightarrow g$, as required.

4 Verification of the translation algorithm

In this section we verify the translation defined above. In order to verify the translation we have to show that any stochastic automata and its corresponding timed automaton with deadlines are equivalent. By equivalent we mean that they have the same meaning in some suitable semantic model. Because we are removing the stochastic element in the stochastic automata in order to verify reachability properties, the semantic model we choose is one that records just actions and their associated timings. In fact we use two slight variants in order to record this information: a timed action transition system and a time/action transition system. This is because the semantic model for stochastic automata uses timed actions, while timed automata with deadlines are mapped to time/action transition systems.

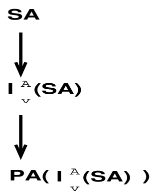


The verification of the translation then amounts to showing that the timed action semantics of a stochastic automaton is equivalent to the time/action semantics of a timed automaton with deadlines. The proof is in two parts. The first shows that any timed action trace arising from a stochastic automaton can also be performed by the timed automaton with deadlines (i.e. the trace is in the time/action semantics). The second part of the proof does the converse: that any time/action trace in the timed automaton with deadlines is a possible trace in the semantics of the stochastic automaton.

Before we give the proof we define timed action and time/action transition systems and show how to map the automata to their respective models.

4.1 Timed action transition systems: the semantics of SA

The timed action transition system results from a stochastic automaton where we abstract away from the stochastic information. It is defined as the end result of first taking the *interpretation* of a stochastic automaton, followed by its *probabilistic abstraction*. These are defined as follows.



In the interpretation I_v^A, A represents an *actual*, as opposed to *potentational* behaviour, and v is the initial valuation.

4.1.1 *The interpretation of a stochastic automaton*

The interpretation of a stochastic automaton is given by a *Probabilistic Transition System* (PTS). Probabilistic transition systems are explained in detail in [7], where they are used to give a semantic model for stochastic automata. Here, they are used simply as an intermediate step in generating the timed action system. If $SA = (\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \dashrightarrow, \kappa, F)$ is a stochastic automaton, then the interpretation is given by the probabilistic transition system $I^A_{v_0}(SA) = ((\mathcal{S} \times \mathcal{V} \times \{0\}), (\mathcal{S} \times \mathcal{V} \times \{1\}), (s_0, v_0, 0), \mathbf{A} \times \mathbb{R}_{\geqslant 0}, T, \longrightarrow)$ where T and $\longrightarrow$ are defined by the rules **Prob** and **Act**:

$$\frac{\overrightarrow{\kappa(s)} = \{x_1, \ldots, x_n\}}{T(s, v, 0) = \mathcal{D}^s_v(\mathcal{R}(F_{x_1}, \ldots, F_{x_n}))}\ \textbf{Prob} \qquad \frac{s \overset{a,C_a}{\dashrightarrow} s' \wedge d \in \mathbb{R}_{\geqslant 0} \wedge \forall x \in C_a.(v-d)(x) \leqslant 0 \quad \forall d' \in [0.d).\forall s \overset{b,C_b}{\dashrightarrow}.\exists y \in C_b.(v-d')(y) > 0}{(s, v, 1) \xrightarrow{a(d)} (s', (v-d), 0)}\ \textbf{Act}$$

The rule **Prob** corresponds to setting some clocks within the stochastic automaton. Essentially, $T(s, v, 0)$ is the probability space where the sample space is made up of the state s combined with all possible resultant clock settings, and the probability measure is derived from $F_{x_1}, \ldots, F_{x_n}$ in the obvious way. The annotations 0 and 1 are simply to indicate whether or not the clocks have been set, and we say $(s, v', 1) \in T(s, v, 0)$ if $(s, v, 0)$ is in the sample space of the probability space.

The rule **Act** specifies the conditions under which a stochastic automaton can perform an action a at time d.

4.1.2 *The probabilistic abstraction of a stochastic automaton*

The timed action system is produced from the PTS via the probabilistic abstraction function PA. As its name suggests, this function removes all probabilistic information from the PTS.

Let $((\mathcal{S} \times \mathcal{V} \times \{0\}), (\mathcal{S} \times \mathcal{V} \times \{1\}), (s_0, v_0, 0), \mathbf{A} \times \mathbb{R}_{\geqslant 0}, T, \longrightarrow)$ be the interpretation of the stochastic automaton. The probabilistic abstraction of the SA is given by

$$((\mathcal{S} \times \mathcal{V}), (s_0, v_0), \mathbf{A} \times \mathbb{R}_{\geqslant 0}, \longrightarrow_S)$$

where $((s, v), a(d), (s', v')) \in \longrightarrow_S$ if and only if

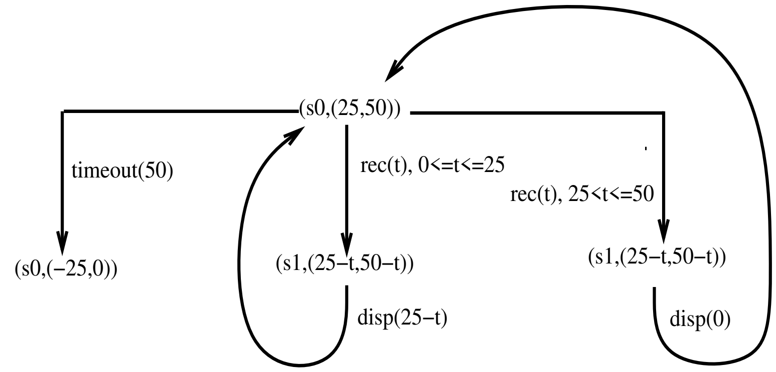
$$\begin{array}{l} \exists v'' \in \mathcal{V}.(s, v'', 1) \in T(s, v, 0) \\ \qquad\qquad \wedge \\ \qquad\qquad ((s, v'', 1), a(d), (s', v'' - d, 0)) \in \longrightarrow \end{array}$$

and $v'' - d = v'$.

Note that we use the subscript on the transition system arrow to distinguish it from the TAD transition relation, which will be introduced later.

In accordance with convention, we will write $(s, v) \xrightarrow{a(d)}_S (s', v')$ for $((s, v), a(d), (s', v')) \in \longrightarrow_S$, and therefore a timed action transition system consists of transitions with labels of the form $a(d)$, where a is a discrete action and d is a time value. A transition $(s, v) \xrightarrow{a(d)}_S (s', v')$ should be understood to mean that the state-valuation pair (s, v) can delay for a time d, before performing the action a and entering the state-valuation pair (s', v').

Below, we give the timed action transition system resulting from the stochastic automaton description of the Sink, specified on page 6. Because time is represented by the real numbers, any timed action graph quickly becomes infinite and so where we can we draw just one arrow, with a label to represent the range of possible time values.



4.2 Time/action transition systems: the semantics of TAD

The time/action transition system is the standard semantics for timed automata with deadlines. It consists of transitions labelled by elements of A, which correspond to state changes, and transitions labelled with non-negative reals, which correspond to time steps. The significant difference between timed action and time/action transition systems is that in the former the passing of time and the performing of an action are merged into one transition, while in the latter they are separate transitions.

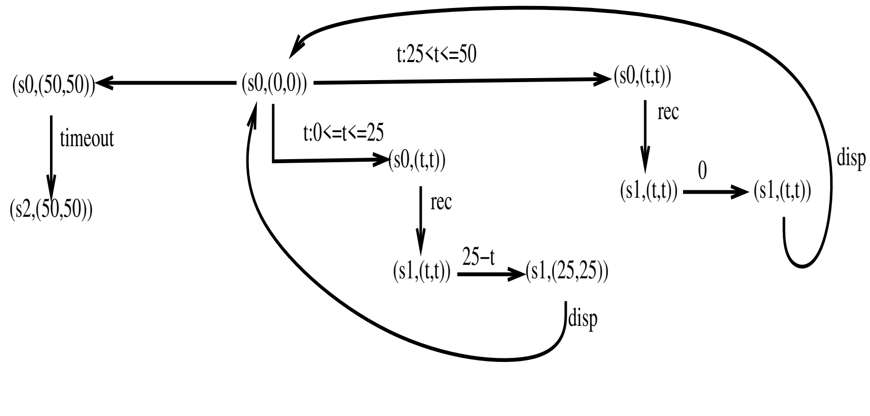
The time/action semantics of the TAD is given by the transition relation $\longrightarrow_T \subseteq (Z \times \mathbb{R}^n_{\geqslant 0}) \times (A \cup \mathbb{R}_{\geqslant 0}) \times (Z \times \mathbb{R}^n_{\geqslant 0})$. A *(state,valuation) pair* of a TAD is a pair (z, u) where $z \in Z$ is a discrete state and $u \in \mathbb{R}^n_{\geqslant 0}$ is a *clock valuation*.

Given $z \in Z$, if $\{(z, a_i, z_i)\}_{i \in I}$ is the set of all transitions issued from z and $h(z, a_i, z_i) = (z, (a_i, g_i, d_i, r_i), z_i)$ then the time/action transitions are defined by the following two clauses:

- $\forall\, i \in I. \forall\, u \in \mathbb{R}^m_{\geqslant 0}.(z, u) \xrightarrow{a_i} (z_i, u[r_i])$ if $g_i(u)$ where $u[r_i]$ is the valuation obtained from u when all the clocks in r_i are set to zero, and the others are left unchanged.

- $(z, u) \xrightarrow{t} (z, u + t)$ if $\forall\, t' < t.c_s(u + t')$ where $c_s = \neg \bigvee_{i \in I} d_i$.

For example, the time/action transition system resulting from the TAD specification of the Sink on page 10 is as below. Again, we parameterise time transitions where we can.



4.3 Proof

We are now in a position to give the proof of equivalence between the semantic models that arise from a stochastic automaton and its translation.

Outline: We will prove that the translation from stochastic automata to timed automata with deadlines (the function *sa2tad*) is correct with respect to the probabilistic abstraction function. In particular, we prove timed trace equivalence: the timed traces possible through the transition system $PA(I^A_{v_0}(SA))$ and the timed traces possible through the transition system $[\![sa2tad(SA)]\!]$ are equal, where SA is any stochastic automaton; $I^A_{v_0}(SA)$ is the interpretation of the stochastic automaton in terms of a probabilistic transition system; *PA* is the probabilistic abstraction function; *sa2tad* is the function from stochastic automata to timed automata with deadlines; and $[\![]\!]$ is the semantic interpretation of a TAD.

First, we introduce some auxiliary definitions between the two different types of valuations: stochastic automata valuations, denoted v in the sequel, and TAD valuations, denoted u. If n is the number of clocks in the valuation, then v is in $\mathbb{R}^n$ and u is in $\mathbb{R}^n_{\geqslant 0}$. We assume the set of clocks is ordered.

Example: As an example, consider the timed action and time/action transition systems given above. If the pair (t, a) represents the action a being observed at absolute time t, then the timed trace

$$\langle (0, rec), (25, disp), (45, rec), (50, disp), (100, tout) \rangle$$

is a possible behaviour of both systems. The proof works by showing that the set of all possible traces resulting from the conventional interpretation of the stochastic automaton and the set of all possible traces resulting from our TAD interpretation are equal.

4.3.1 SA traces occur in its TAD translation

The proof is in two parts. This part shows that any trace that the SA can perform can be performed by the TAD. We do this by showing that any *(state,valuation)* pair in the SA can be simulated by a *(state,valuation)* pair in the TAD. We do this by showing that if two *(state,valuation)* pairs *correspond*, in a manner to be defined, then a timed action possible for the SA has related time and action transitions within the TAD.

Definition 4 *Correspondence*
Two valuations v and u *correspond* (written $v \asymp u$) provided v is in $\mathbb{R}^n$, u is in $\mathbb{R}^n_{\geqslant 0}$ and

$$\forall\, i \leqslant n. v(i) + u(i) \in \mathrm{ran}(c_i)$$

We then define the correspondence between *(state,valuation)* pairs as follows.

$(s, v) \asymp (z, u)$ iff $s = z$ and $\forall\, \overline{v} \in \mathcal{D}^s_v(\mathcal{R}(\overrightarrow{\kappa(s)})).\overline{v} \asymp u.$ □

We now begin the proof. We assume that $(s, v) \asymp (z, u)$, and we wish to prove that

$$(s, v) \xrightarrow{a(d)}_S (s', v') \Rightarrow (z, u) \xrightarrow{d}_T (z'', u + d) \wedge (z'', u + d) \xrightarrow{a}_T (z', (u + d)[r_a])$$

By the PA function we deduce that

$$\begin{array}{l} \exists\, v''.(s, v'', 1) \in T(s, v, 0) \\ \quad \wedge \\ \quad (s, v'', 1) \overset{a(d)}{\to}_P (s', v'' - d, 0) \end{array}$$

where $v'_S = v'' - d$.

From the first clause it follows (by the rule **Prob**) that $(s, v'', 1) \in \mathcal{D}^s_v(\mathcal{R}(F_{x_1}, \ldots, F_{x_n}))$ where $\overrightarrow{\kappa(s)} = \{x_1, \ldots, x_n\}$. (We will write $\mathcal{D}^s_v(\mathcal{R}(\overrightarrow{\kappa(s)}))$ as an abbreviation for $\mathcal{D}^s_v(\mathcal{R}(F_{x_1}, \ldots, F_{x_n}))$.)

From the second clause it follows (from rule **Act**) that

$$
\begin{array}{l}
s \xrightarrow{a,C_a} s' \wedge \forall x \in C_a.(v'' - d)(x) \leqslant 0 \\
\forall d' \in [0, d). \forall s \xrightarrow{b,C_b} . \exists y \in C_b.(v'' - d')(y) > 0
\end{array}
$$

To show that $(z, u) \xrightarrow{d}_T (z'', u + d)$ we need to show that $\forall d' < d.c_s(u + d')$ where $c_s = \neg \bigvee_{i \in I} dead_i$, where I enumerates all the possible transitions from s, and $dead_i$ is the deadline for transition i.

We know that $(s, v) \asymp (z, u)$, and therefore that $s = z$ and that $\forall \overline{v} \in \mathcal{D}^s_v(\mathcal{R}(\overrightarrow{\kappa(s)})).\overline{v} \asymp u$. In particular, $v'' \asymp u$, so $\forall x.v''(x) + u(x) \in \mathrm{ran}(x)$.

Since $\forall x.v''(x) + u(x) \in \mathrm{ran}(x)$, it follows that $\forall d' \in [0, d). \forall x.(v'' - d')(x) + (u + d')(x) \in \mathrm{ran}(x)$, and since $\forall d' \in [0, d). \forall s \xrightarrow{b,C_b} . \exists y \in C_b.(v'' - d')(y) > 0$, it follows that $\forall d' \in [0, d). \forall s \xrightarrow{b,C_b} \exists y \in C_b.(u - d')(y) < \max(y)$.

Thus for every time value less than d, and for every action b possible from s, there is some clock y in the trigger set C_b such that $u(y) < \max(y)$, and therefore the c_s condition above holds.

To show that $(z, u + d) \xrightarrow{a}_T (z', (u + d)[r_a])$, we must show that $g(u + d)$ holds. We divide this into two cases. First, we consider the case where the action a is not urgent, i.e. some of the clocks in C_a may still be running when the state s is entered, and so a is not forced to happen immediately.

Now we know from rule **Act** that $\forall d' \in [0, d). \exists x \in C_a.(v'' - d')(x) > 0$ and $\forall x \in C_a.(v'' - d)(x) \leqslant 0$. Therefore there must be a clock y in C_a such that $(v'' - d)(y) = 0$, and by correspondence $(u + d)(y) \in \mathrm{ran}(y)$ and therefore $g(u + d)$ holds, as required.

In the second case, action a is urgent: all clocks in C_a have expired strictly before the state s is entered. In this case, we cannot find a $d > 0$ such that $\forall d' \in [0, d). \exists x \in C_a.v(x) - d' > 0$, and therefore d must be zero. For each clock x in C_a, $ts_x + \max(x) < t_a$, where ts_x is the absolute time at which the clock x was last set. In the TAD, each clock is set to zero at time ts_x and counts up, therefore at time t_a all clocks in C_a must be greater than $\max(x)$, and therefore $g(u)$ holds, as required.

It remains to show that $(s', v') \asymp (z', u')$.

The timed action transition is $(s, v) \xrightarrow{a(d)}_S (s', v - d)$, and the time/action transitions performed are $(z, u) \xrightarrow{d}_T (z, u + d) \xrightarrow{a}_T (z', (u + d)[r_a])$, so we must show that $(s', v - d) \asymp (z', (u + d)[r_a])$.

By rule **Act** $s \xrightarrow{a,C_a} s'$, and since this is a transition of the stochastic automaton SA we can deduce that $z \xrightarrow{a,C_a} z'$, and therefore that $s' = z'$.

To show that the valuations match, we begin by noting that, by definition, $\overrightarrow{\kappa(s')} = r_a$.

Now, let $\overline{v}'$ be in $\mathcal{D}^{s'}_{v'}(\mathcal{R}(\overrightarrow{\kappa(s')}))$. Then for all clocks x

$$
\begin{array}{l}
x \in (\overrightarrow{\kappa(s')}) \Rightarrow \overline{v}'_s(x) \in \mathrm{ran}(x) \\
x \notin (\overrightarrow{\kappa(s')}) \Rightarrow \overline{v}'(x) = v'(x)
\end{array}
$$

and for all clocks x

$$x \in r_a \Rightarrow (u+d)[r_a](x) = 0$$
$$x \notin r_a \Rightarrow (u+d)[r_a](x) = (u+d)(x)$$

so, for all valuations $\overline{v}'(x)$ in $\mathcal{D}^{s'}_{v'}(\mathcal{R}(\kappa(\overrightarrow{s'})))$, and all clocks x, $\overline{v}'(x) + u(x) \in \mathrm{ran}(x)$, as required.

Initial states: Since all the clocks x in the initial SA valuation v_0 are initially set to a value within their range $\mathrm{ran}(x)$, and all the clocks in the initial TAD valuation u_0 are set to zero, and the initial state of the TAD is derived from the initial state of the SA, the two (state,valuation) pairs (s_0, v_0) and (z_0, u_0) clearly correspond.

This completes the first half of the proof. □

4.3.2 Any TAD trace is a valid SA trace

We now prove that given a translation of a stochastic automaton into a TAD, any trace possible for the TAD is possible for the stochastic automaton. We do this by induction on the length of the trace: provided $\langle \ldots, (t_{j-1}, a_{j-1})\rangle$ is a valid trace of both the SA and the TA, we show that if $\langle \ldots, (t_{j-1}, a_{j-1}), (t_j, a_j)\rangle$ is a valid trace of the TAD, it is also a valid trace of the SA. In order to simplify the presentation, we make the assumption that all clocks are associated with only one transition.

In the base case, if $\langle (t_1, a_1)\rangle$ is a trace of the TAD then from the definition of the guard we know that

$$\begin{array}{l}(\bigwedge_{i \in C_1} t_1 \geqslant \min(c_i) \\ \wedge \\ \bigvee_{k \in C_1} t_1 \in \mathrm{ran}(c_k)) \\ \vee \\ \bigwedge_{i \in C_1} t_1 > \max(c_i)\end{array}$$

and from the time-passing constraint we know that

$$\forall\, d' \in [0, t_1).\, \forall\, l \in L.\, \exists\, m \in C_l.\, \max(c_m) \leqslant d'$$

(where L enumerates all outgoing transitions from s_0).

In this instance the time-passing constraint is incompatible with the second clause of the guard, and therefore only the first clause can be true. The trace can be reproduced by setting clock c_k to t_1, and c_i to $\min(c_i)$ for each clock c_i, $i \neq k$. Since clocks can only be used once, setting clocks to their minimums will not interfere with any other possible transitions from s_0.

For the inductive step, consider a trace $\langle \ldots, (t_{j-1}, a_{j-1}), (t_j, a_j)\rangle$ from a TAD. Recall that t_j is the time at which event a_j occurs, C_j is the trigger set of action a_j, and ts_i is the latest time at which clock i was reset.

From the definition of the guard we know that

$$\begin{array}{l}(\bigwedge_{i \in C_j} t_j \geqslant \min(c_i) + ts_i \\ \wedge \\ \bigvee_{k \in C_j} t_j \in \mathrm{ran}(c_k) + ts_k) \\ \vee \\ \bigwedge_{i \in C_j} t_j > \max(c_i) + ts_i\end{array}$$

and from the time-passing constraint we know that

$$\forall\, d' \in [t_{j-1}, t_j).\, \forall\, l \in L.\, \exists\, m \in C_l.ts_m + \max(c_m) \leqslant d'$$

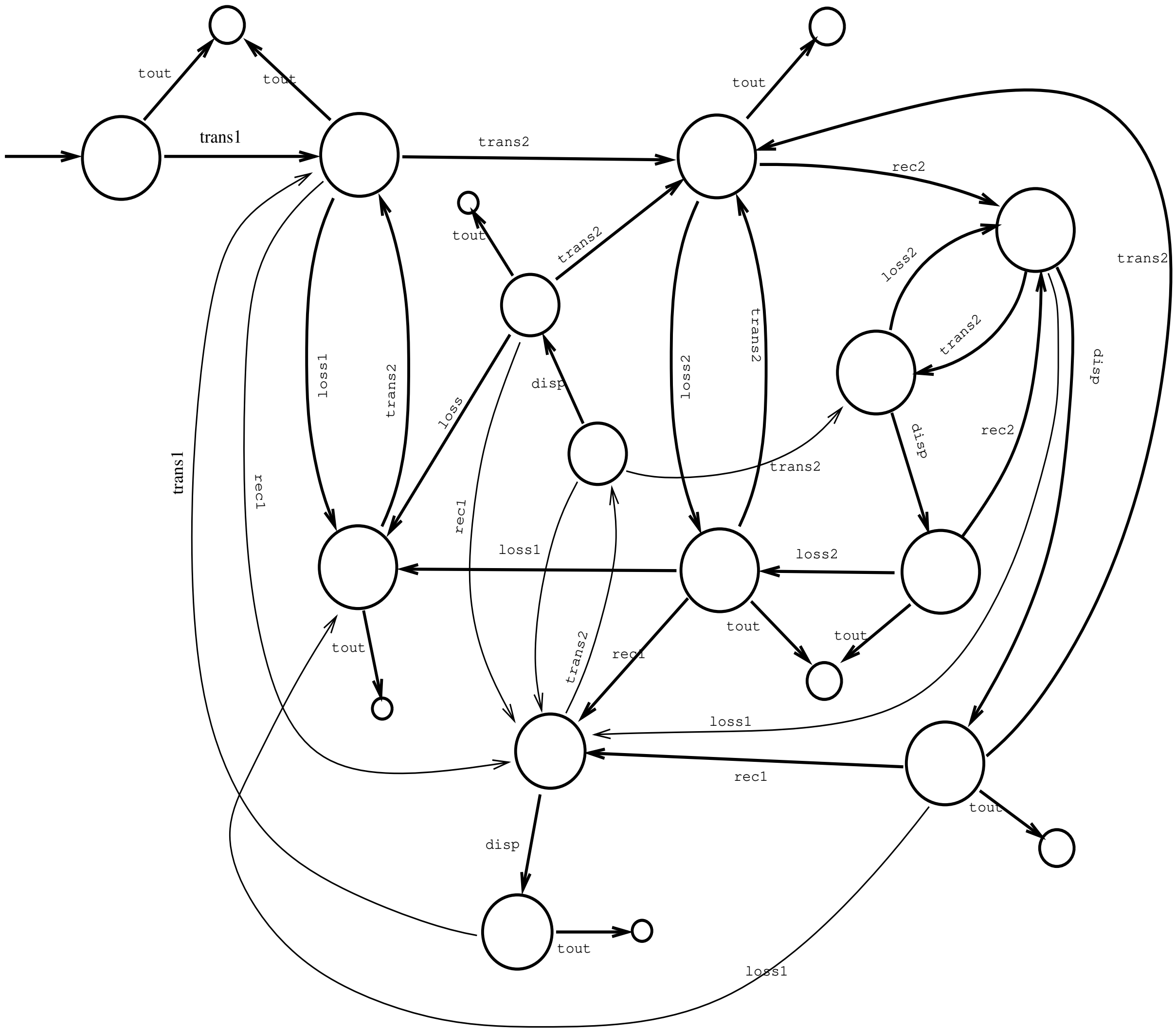
The definition of the guard gives rise to two cases. In the first, at least one of the clock variables must be within range ($\bigvee_{k \in C_j} . t_j \in \mathrm{ran}(c_k) + ts_k$) and all clocks must have have started ($\bigwedge_{i \in C_j} . t_j \geqslant \min(c_i) + ts_i$). The SA can reproduce this behaviour by setting clock k to $t_j - t_{j-1}$ and all other clocks to their respective minimums. Since clocks are only used once we are free to set all clocks here as we wish. In this first case, the time-passing constraint ensures that every other outgoing transition has at least one clock (say c_m) in its trigger set which may still be active ($ts_m + \max(c_m) \geqslant t_j$), and therefore no other transition is forced to have occurred previously.

In the second case, t_j is strictly greater than $\max(c_i) + ts_i$ for all clocks in i in the trigger set of a_j, but the time-passing constraint applied to transition a_j states that for all time between t_{j-1} and t_j there is a clock in the trigger set which is less than $\max(c_i) + ts_i$. This can only be resolved if no time elapses between t_{j-1} and t_j (i.e. that $t_{j-1} = t_j$), and this corresponds to the case in the SA where all clocks in the trigger set have already expired, and so action a_j fires as soon as action a_{j-1} does. $\square$

4.4 Example

The translation of our stochastic automaton representing the multimedia stream can now be derived. The result is the timed automaton given below. Due to space limitations within the diagram, the transitions are labelled representatively; the meaning of the labels is given below.

label	action	guard	deadline	reset
*loss*1	*loss*	$l \geqslant 0$	$l \geqslant 200$	$\{\}$
*loss*2	*loss*	$l \geqslant 0$	$l \geqslant 200$	$\{r, l\}$
tout	*tout*	$t' \geqslant 50$	$t' \geqslant 50$	$\{\}$
*rec*1	*rec*	$r \geqslant 25$	$r \geqslant 75$	$\{\}$
*rec*2	*rec*	$r \geqslant 25$	$r \geqslant 75$	$\{r, l\}$
disp	*disp*	$d \geqslant 25$	$d \geqslant 25$	$\{d, t'\}$



The *trans* actions have more complex guards and deadlines, since they are derived from two clocks.

label	action	guard	deadline	reset
*trans*1	*trans*	$((t \geqslant 9 \wedge tr \geqslant 5) \wedge$ $(t \in [9,11] \vee tr \in [5,10] \vee tr \in [45,55]))$ $\vee\ (t \geqslant 11 \wedge tr \geqslant 55)$	$t \geqslant 11 \wedge tr \geqslant 55$	$\{t, r, l, tr\}$
*trans*2	*trans*	$((t \geqslant 9 \wedge tr \geqslant 5) \wedge$ $(t \in [9,11] \vee tr \in [5,10] \vee tr \in [45,55]))$ $\vee\ (t \geqslant 11 \wedge tr \geqslant 55)$	$t \geqslant 11 \wedge tr \geqslant 55$	$\{t, tr\}$

The aim of this translation is to produce a timed automaton which we can check for reachability properties. Examples of reachability properties might include: can we reach a state where a timeout is possible? Can we reach a state where more than ten packets are in the channel? None of these properties requires precise probabilities to be determined, therefore we can check our stochastic automaton against these properties by using the translation into timed automaton[2].

[2]Lack of space means we cannot give a full account of this here, however it would involve turning the timed automaton with deadlines into a form suitable for input into one of the automata tools, e.g. UPPAAL.

5 Conclusions

In this paper we have defined a translation from stochastic automata to timed automata with deadlines. The aim of this was to enable reachability analysis to be performed on stochastic automata in a feasible manner. Although this is successful much remains to be done in terms of verification of stochastic specification. The greatest challenge in this respect lies in the area of stochastic model checking. There are extremely efficient model checkers for timed automata, yet model checking for stochastic automata with arbitrary distributions is currently hampered by problems due to the extensive calculations necessarily involved in building model checking procedures. Any work in this direction would represent a significant breakthrough.

Acknowledgements: The research presented here is supported by the UK Engineering and Physical Sciences Research Council under grant number GR/L95878 (A Specification Architecture for the Validation of Real-time and Stochastic Quality of Service). Thanks are due to co-workers on this project for their input into this work: Howard Bowman at Kent and Lynne and Gordon Blair from Lancaster University. We would also like to thank Pedro D'Argenio and Justin Pearson for their help and advice with this work.

References

[1] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for probabilistic real-time systems. In *Proceedings of 18th ICALP*, 1991.

[2] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] Marco Bernardo, Lorenzo Donatiello, and Roberto Gorrieri. Integrating performance and functional analysis of concurrent systems with EMPA. Technical Report UBLCS-95-14, Department of Computer Science, University of Bologna, Piazza di Porto S. Donato, 5, 40127 Bologna, September 1995. Revised March 1996.

[4] Lynne Blair and Gordon Blair. Composition in multiparadigm techniques. In Paolo Ciancarini, Alessandro Fantechi, and Roberto Gorrieri, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 401–417, February 1999.

[5] Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modelling urgency in timed systems. In *COMPOS'97*. LNCS, 1997.

[6] Pedro D'Argenio. Personal communication.

[7] Pedro R. D'Argenio, Joost-Pieter Katoen, and Ed Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, editors, *Proceedings of the IFIP Working Conference on Programming Concepts and Methods, PROCOMET'98*, pages 126–147. Chapman & Hall, 1998.

[8] P.W. Glynn. A GSMP formalism for discrete event simulation. In *Proceedings of the IEEE*, volume 77(1), pages 14–23, 1989.

[9] Holger Hermanns, Michael Rettlebach, and Thorsten Weiss. Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching. *The Computer Journal*, 38(7):530–541, 1995.

[10] Jane Hillston. *A Compositional Approach to Performance Modelling*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.

[11] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, Centre for Telematics and Information Technology, P.O. Box 217, 7500 AE Enschede The Netherlands, April 1996.

[12] K.G. Larsen, P. Pettersson, and W. Yi. Diagnostic model-checking for real-time systems. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, 1995.

[13] W. Reisig. *Petri Nets, An Introduction*. Springer-Verlag, 1982.

[14] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.

[15] S. Yovine. Kronos: A Verification Tool for Real Time Systems. *Springer International Journal of Software Tools for Technology Transfer*, 1(1/2), 1997.