Specification and Analysis of Automata-based Designs

Jeremy Bryans[1] and Lynne Blair[2] and Howard Bowman[1] and John Derrick[1]

[1] Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent, CT2 7NF, UK,
[2] Computing Department, Faculty of Applied Sciences, Lancaster University, Lancaster, LA1 4YR, UK

Abstract. One of the results of research into formal system specification has been the large number of notations which have been developed. Of these notations, *automata* have emerged as a promising vehicle for the specification, and particularly the analysis, of systems. This is especially so when the systems under consideration include timing requirements, and *timed* automata model such systems as a finite set of states with timed transitions between them. However, not all specifications involve deterministic timing, and *stochastic* automata can be used in these circumstances.
In this paper we consider both timed and stochastic automata, and demonstrate how they can be used in the same design. We will also consider what analysis of the specification can then be performed. In particular, we will describe how to translate stochastic to timed automata, and look at two approaches to model checking the stochastic components of an integrated design.

Keywords: Timed automata, stochastic automata, model checking.

1 Introduction

One of the results of research into formal system specification has been the large number of notations which have been developed. There are now many notations which can be used to specify and design systems. Potential problems with this are that specifiers working on the same system may be familiar with different notations, and that different notations may be better suited for different parts of the same design.

Even within notations there can be variants, and in this paper we will confine ourselves to automata. We will demonstrate how different automata notations can be used in the same design, and how analysis of the specification can be performed according to the particular notation used. This means that designers need not be restricted to a monolithic notation, and that the most convenient notation can be chosen to describe each component within the design.

In this paper we will focus on timed automata with deadlines and stochastic automata. Timed automata are now well established as a specification notation, and there has been extensive work on analysis techniques for them, and in particular *model checking* algorithms. Stochastic automata are a relatively new extension to timed automata, where the emphasis has been shifted from deterministic timing to timings picked from a probabilistic distribution, thus enabling a new range of systems to be specified.

The structure of the paper is as follows. In Section 2 we present the automata-based notations that we will use throughout the paper, and in particular timed automata with deadlines and stochastic automata. Section 3 presents an example using these notations which models cars arriving at a port wishing to board a ferry. Section 4 looks at possible ways to analyse such a specification, and compares them with each other.

Specifically we are interested in timed vs stochastic analysis. For the former there is a wide range of techniques available and therefore we concentrate on how we can integrate the stochastic components into this analysis. To this end we show how a stochastic automata can be translated into a timed automata with deadlines, as this allows the integrated specification to be interpreted within a single simpler notation.

However this clearly involves a loss of some stochastic information, and to perform stochastic analysis we look at two approaches to model checking the stochastic components of an integrated design. Finally in Section 5 we draw some conclusions, and mention ongoing and possible future work.

2 Notations

For the purposes of this paper, we choose automata as a “base” notation, and we will use the timed automata with deadlines (TAD) of [BST98], and the stochastic automata (SA) of [DKB98] as necessary. Although different versions of timed automata exist, we chosen TAD over the others because of the ease of translating from SA to TAD (see Section 4.1.) Both TAD and SA are extensions of ordinary automata, and we give definitions for them now.

Definition 1 In this paper, a TAD is:

- A discrete labelled transition system $(\mathcal{U}, \rightarrow, A)$ where
  - $\mathcal{U}$ is a finite set of discrete states
  - A is a finite set of actions
  - $\rightarrow \subseteq \mathcal{U} \times A \times \mathcal{U}$ 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: $h(u, a, u') = (u, (a, g, d, r), u')$ where
  - g and d are the *guard* and *deadline* of the transition. Guards and deadlines are predicates p defined by the following grammar:

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

    where $x \in X$, $w \in \mathbb{R}_{\geqslant 0}$ and $\# \in \{\leqslant, <, >, \geqslant\}$.

– r is the set of clocks which are reset to zero when the transition takes place.

A transition may occur only when the guard is true, and must occur if the deadline is true. (The definition of the grammar for defining the guard and deadline predicate is slightly modified from the one found in [BST98].)

The clocks in TAD always begin counting at zero and count upwards. This is in contrast to the clocks in SA, which are set to some value in $\mathbb{R}_{\geqslant 0}$ according to their probability distribution function, and count downwards.

As an example, consider the TAD depicted in Figure 1. From state u_0, the action a may occur provided the clock x_1 is greater than 2, and must occur if it is equal to 4. When it does occur, the clock x_2 is reset and the automaton moves to state u_1. From here, the action b may occur provided clock x_1 is in the range $[6, 8]$ and clock x_2 is greater than 3. The deadline imposes no restriction, and when the action does occur no clocks are reset.

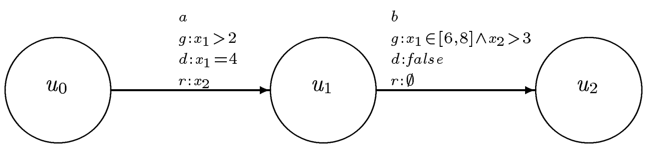

Fig. 1. A Timed Automaton with Deadlines

Stochastic automata are an extension of timed automata, in which the time at which actions occur may be a random variable. In this paper we use the stochastic automata defined in [DKB98], which are presented below.

Definition 2 A *stochastic automaton* is a structure $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, \dashrightarrow, \kappa, F)$ 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*.

$\dashrightarrow \subseteq \mathcal{S} \times (\mathbf{A} \times \mathcal{C}) \times \mathcal{S}$ is the set of *edges*. If s and s' are states, a is an action and C is a finite subset of $\mathcal{C}$, then we denote the edge $(s, a, C, s') \in \dashrightarrow$ by $s \overset{a,C}{\dashrightarrow} s'$ and we say that C is the *trigger set* of action a. We use $s \overset{a}{\dashrightarrow} s'$ as a shorthand notation for $\exists\, C.s \overset{a,C}{\dashrightarrow} s'$. In this paper we will associate only a single clock with each action.

$\kappa : \mathcal{S} \to \mathbb{P}_{fin}(\mathcal{C})$ is the *clock setting function*, and indicates which clocks are to be set in which states, where $\mathbb{P}_{fin}(\mathcal{C})$ is the finite powerset of clocks.

$F : \mathcal{C} \to (\mathbb{R} \to [0, 1])$ 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. Each clock $x \in \mathcal{C}$ is a random variable with distribution F_x.

In this paper we will assume that clocks are only used on transitions emanating from the states in which they are set. We will also find it easier to refer to probability density functions (pdf's), which are the derivatives of the distribution functions. We will use P_x for the pdf of F_x.

As an example, of a stochastic automaton, consider Figure 2. This is written $(\{s_0, s_1\}, s_0, \{x, y\}, \{a, b\}, \twoheadrightarrow, \kappa, \{\mathsf{P}_x, \mathsf{P}_y\})$ where $\twoheadrightarrow = \{(s_0, a, \{x\}, s_1), (s_0, b, \{y\}, s_0)\}$, and the pdf's for clocks x and y are

$$\begin{array}{ll} \mathsf{P}_x(t) = 4 - 2t, \text{if } t \in [1,2] & \mathsf{P}_y(t) = 2t - 2, \text{if } t \in [1,2] \\ \quad = 0, \text{otherwise} & \quad = 0, \text{otherwise} \end{array}$$

as depicted. The horizontal axis measures time, and the vertical axis measures the probability of the clock being set to a value less than that time.

The SA starts in location s_0, and both clocks x and y are set according to the functions F_x and F_y. If clock x expires first, then action a is triggered and the automaton moves to location s_1. This location has no outgoing transitions, and so nothing further happens. If clock y expires first, then action b is triggered and the automaton returns to state s_0. The clocks are reset according to their distributions, and the process is repeated.

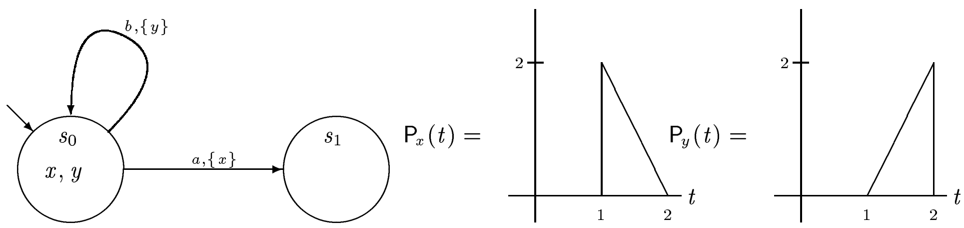

Fig. 2. A Stochastic Automaton

In the following Section we will show how we can combine both stochastic and timed automata using a larger example.

3 Example - A car ferry

To illustrate these ideas, we will specify a system consisting of a number of cars at a port, trying to get on to a ferry (see Figure 3.) The cars enter the port at the traffic lights, and join the queue in the middle of the port. When they reach the front of the queue they move to the next free kiosk, where they are processed, and then they go on to join the ferry.

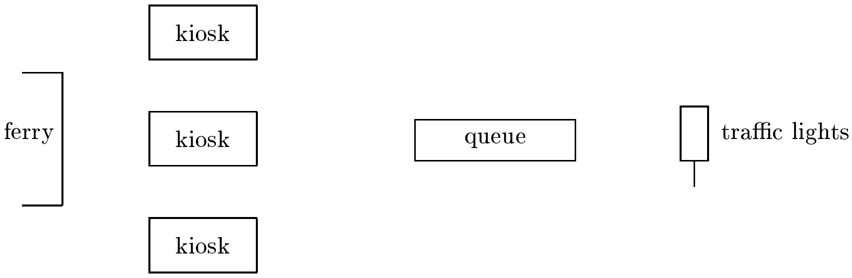

Fig. 3. The port

In this example, there are two parts of the model over which we do not have direct control. One is the arrival of the cars into the queue, and the other is the rate at which individual kiosk workers work. For both of these we use stochastic automata to model the inherent uncertainty.

We will consider that actions synchronise with other actions of the same name, and that in a parallel composition the intersection of the alphabets synchronise (as in [Hoa85].) For the car arrivals, we use the distribution shown in Figure 4. This can be thought of as modelling the behaviour resulting from a set of nearby traffic lights: If one car arrives it is quite likely that another will arrive very shortly afterwards, (between 5 and 10 seconds). If no car arrives in this time then the lights will turn red, and no car will be able to arrive until 30 seconds have passed. Whether or not this function is an accurate representation of the environment in which the system will have to operate can only be determined by observing the actual behaviour of the cars.

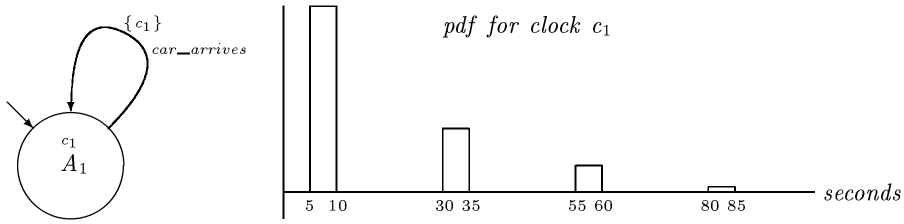

Fig. 4. Modelling car arrivals as an SA

We have a little more control over the behaviour of the kiosks, in that we can choose how many are open at a time. However, we cannot determine the rate at which the individual operators work, and so this must be represented as a stochastic function.

An individual kiosk is modelled as an SA, as shown in Figure 5. We model the kiosk as opening immediately, and twelve seconds after a kiosk opens, a car

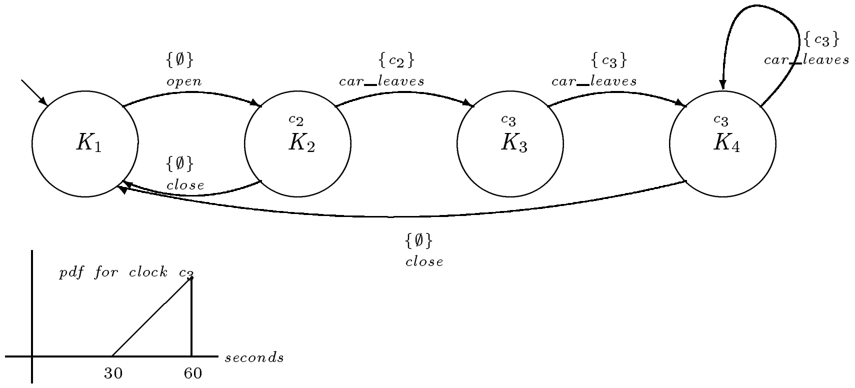

Fig. 5. Modelling a single kiosk as an SA

leaves the queue to be processed. This is modelled by the clock c_2, which is deterministically set to 12 seconds every time state K_2 is entered.

The processing takes between 30 and 60 seconds, and this continues until the kiosk is closed. This is modelled by the clock c_3, set to a value between 30 and 60 seconds according to the pdf:

$$\begin{aligned} \mathsf{P}_x(t) &= \tfrac{t-30}{450}, \text{ if } t \in [30, 60] \\ &= 0, \quad \text{otherwise} \end{aligned}$$

Here we are in fact modelling the impact on the queue (using the *car_leaves* actions) rather than on the kiosk directly.

We include the state $K4$ in order to be able to distinguish the state in which the first car has been processed (and the second one has left the queue.) We will make use of this later in the analysis of the kiosk.

The queue that the cars form is essentially passive. It does not instigate either the *car_arrives* or the *car_leaves* actions, and it therefore needs no time deadlines (as TAD) or clocks (as SA) and can be modelled as a simple automaton. This is shown in Figure 6 (where states 3 and 4 and the transitions between them have been elided).

Notice that quite general distributions are allowed in our stochastic automata. Here we have used combinations of uniform and triangular distributions, and in general arbitrary distributions are allowed.

4 Analysing the integrated specifications

In order to analyse a specification defined using a number of different notations, we have two possibilities. We can either re-interpret all the components within one notation (and then use whatever analysis that notation permits) or we can analysis the components of the specification. Here we briefly consider both approaches which are illustrated using the car ferry example.

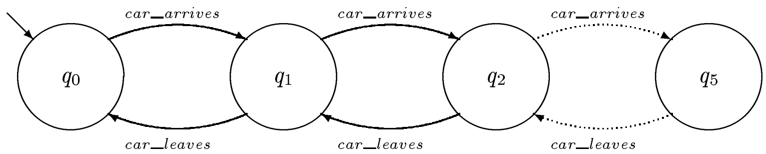

Fig. 6. Modelling the queue as an automaton

4.1 Translating SA to TAD

In this Section we consider the first approach and show how to interpret stochastic automata in terms of timed automata with deadlines. The interpretation must preserve the behaviour of the SA within a TAD as far as possible, so for each SA we must be able to generate the TAD which is capable of exactly the same set of runs[1] as the SA. However, since a TAD cannot represent probabilistic information, this translation will necessarily lose all probabilistic information.

To illustrate the ideas we begin by deriving timed automata with deadlines from the stochastic automata in the example, and then give the formal definition of the translation.

Consider the SA (Figure 4) that models the arrivals of cars at the car ferry. The clock c_1 is set (as we may deduce from the pdf of clock c_1) to some value in $[5,10] \cup [30,35] \cup [55,60] \cup [80,85]$, and then proceeds to count down. The action *car_arrives* occurs when this clock expires.

We derive an corresponding TAD (Figure 7) which must therefore be capable of performing the action *car_arrives* at any time in the range $[5,10] \cup [30,35] \cup [55,60] \cup [80,85]$, and the action must be performed by (or at) time 85. We use x_1 to correspond to clock c_1, and set the guard (the permitted occurrence times) to $x_1 \in [5,10] \cup [30,35] \cup [55,60] \cup [80,\infty)$ and the deadline to $x_1 \geqslant 85$. Setting the deadline to greater than or equal to 85 means that if this state is entered when $x_1 \geqslant 85$ the action must occur.

In the SA, clock c_1 is reset every time the state A_1 is entered, so in the TAD r (which is the set of clocks being reset to zero when the transition occurs) is set to $\{x_1\}$.

This turns out to be an automaton with just one state, however, not all translations are this simple, for example the kiosk description (Figure 5) becomes the TAD in Figure 8.

Using the ideas illustrated in these examples we can formalise the full definition of the translation of stochastic automata to timed automata with deadlines as follows.

Definition 3 Translating an SA into a TAD.

[1] A run is a (finite or infinite) sequence of timed actions.

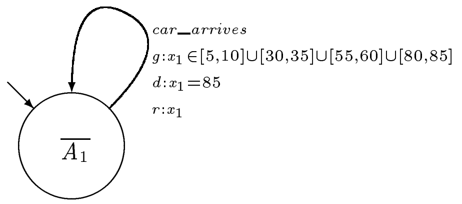

Fig. 7. The translation of the car arrivals into a TAD

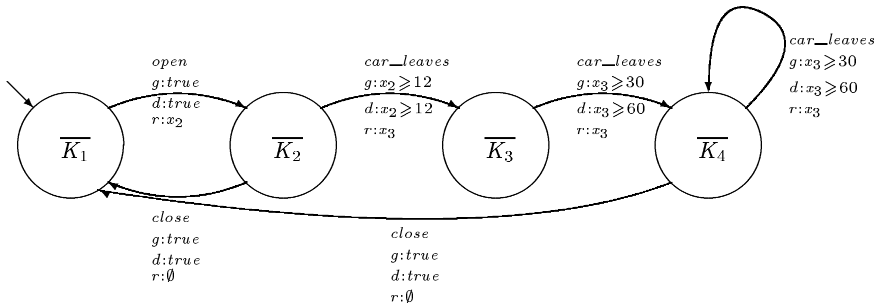

Fig. 8. The translation of a single kiosk into a TAD

Let $(\mathcal{S}, s_0, \mathcal{C}, \mathbf{A}, -\!\blacktriangleright, \kappa, F)$ 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 $-\!\blacktriangleright$ with the clocks removed, i.e.
  $\longrightarrow_T \subseteq Z \times A \times Z$ where
  $\longrightarrow_T = \{(z, a, z') \mid \exists\, C_a.(s, a, C_a, s') \in -\!\blacktriangleright \wedge\ s = z \wedge s' = z'\}$
- The set X contains (non-negative real-valued) clock variables, labelled x_i and indexed as the SA variables.
  $\forall\, i.x_i \in X \Leftrightarrow c_i \in C$
- $h(s, a, s') = (s, (a, g, d, r), s')$ where C_a is the *trigger set* for action a and
  - $g = (\bigwedge_{c_i \in C_a} x_i \geqslant \min(c_i)$
    $\wedge$
    $\bigvee_{c_i \in C_a} x_i \in \mathrm{ran}(c_i))$
    $\vee$
    $\bigwedge_{c_i \in C_a} x_i \geqslant \max(c_i)$
  - $d = \bigwedge_{c_i \in C_a} x_i \geqslant \max(c_i)$
    where $\min(c_i)$, $\max(c_i)$ and $\mathrm{ran}(c_i)$ are the minimum, maximum and range respectively of the pdf of clock c_i.
  - $r = \kappa(s')$

We are endebted to Pedro D'Argenio [D'A] for this definition, and it is discussed in more detail in [BD99].

With the stochastic components turned into timed automata with deadlines, temporal observations of the system can be made, for example to address questions such as "Is a particular throughput of cars possible?", or "With only one kiosk open, what is the minimum/ maximum time before the queue overflows?". To support this task, work has started on extending the LUSCETA tool [JB99]. It currently supports the creation and editing of timed automata and timed automata with deadlines. It also supports the compositon of either type of automata providing all automata are of the same type (the composition rules for timed automata with deadlines are presented in [BS98]). However, the simulator currently only supports timed automata; work is still required to extend this to timed automata with deadlines.

This translation provides us with the ability to analyse the temporal properties of a specification that originally included stochastic information. However, this translation has the obvious drawback that the exact stochastic properties of the specification can no longer be investigated. We move on to consider this in the next Section.

4.2 Model Checking Stochastic Automata

The alternative to translating SA to TAD (and thereby forfeiting the stochastic information) is to keep the stochastic information by retaining the SA, and performing more complex analysis only on the stochastic components. Much of the work done in stochastic modelling and performance evaluation uses the assumption that the random times at which actions occur are drawn from exponential distributions. While this allows many performance evaluation results to be derived, in practice it is unrealistic to consider only exponential distributions, and it is necessary for arbitrary distributions to be considered.

The analysis technique we consider is model checking [CGP99]. This has proved very successful in many applications, and applying it to stochastic systems opens up several new research issues.

In this Section we discuss two approaches to model checking stochastic automata. The first calculates exact answers for stochastic automata involving arbitrary distributions. However, the cost of this precision is the complexity of the algorithm and we also describe a further algorithm which uses a discretisation to reduce this complexity and is discussed in more detail in [BBD00].

A probabilistic real time temporal logic The basic approach we take to model checking is to try to show that a temporal logic property is satisfied by a stochastic automaton description of the system. Here we use a simple probabilistic real-time temporal logic. The purpose of the logic is to express properties that we wish to check the stochastic automaton against and the logic we define allows us to check a range of such properties.

The syntax of our logic is

$$\psi ::= \mathsf{tt} \mid \mathsf{ap} \mid \neg\, \psi \mid \psi_1 \wedge \psi_2 \mid [\phi_1\, \mathcal{U}_{\sim c}\, \phi_2] \simeq p$$
$$\phi ::= \mathsf{tt} \mid \mathsf{ap} \mid \neg\, \phi \mid \phi_1 \wedge \phi_2$$

Here ap is an atomic proposition, $c \in \mathbb{N}$ (natural numbers), $p \in [0,1]$ is a probability value and $\simeq, \sim \in \{<, >, \leqslant, \geqslant\}$. The temporal aspects are described by $[\phi_1\, \mathcal{U}_{\sim c}\, \phi_2] \simeq p$ which is an *"until" formula*. In general we would associate sets of atomic propositions with states of automata; however here it will be sufficient to assume a single distinct proposition for each state, (effectively identifying states and propositions) so that each state A_x models the set of propositions $\{A_x, \mathsf{tt}\}$. Using this logic we can also define a number of derived operators, for details see [BBD00].

To understand an "until" formula, it is simplest to begin with an untimed, non-probabilistic version. Intuitively, $\phi_1\, \mathcal{U}\, \phi_2$ reads as: ϕ_1 holds until ϕ_2 does. The subscript $\sim c$ is the time restriction — eg. if $\sim$ is $\leq$ then ϕ_2 must hold before (or at) time point c. The addition $\simeq p$ is a probability restriction — e.g. if $\simeq$ is $>$ then $\phi_1\, \mathcal{U}_{\sim c} \phi_2$ must be true with probability greater than p.

The until formulae can only be used at the top level — they cannot be nested. This is because the model checking algorithms we discuss can only evaluate until formulae from the initial state; this is a necessary restriction of our current approach.

With this syntax, an example of a valid formula that we can check against the stochastic automaton in Figure 5 would be $[\mathsf{tt}\mathcal{U}_{\leq 60}\, K_4] > 0.3$. This states that the probability of reaching the state K_4 (and therefore having processed the first car) within 60 seconds is greater than 0.3.

It should be clear that since we do not allow the until formulae to be nested we can use the following recipe in order to model check a formula ψ of our logic against a stochastic automaton A.

1. For each until subformula (i.e. of the form $[\phi_1 \mathcal{U}_{\sim c} \phi_2] \simeq p$) in ψ perform an individual model check to ascertain whether

   $$A \models [\phi_1 \mathcal{U}_{\sim c} \phi_2] \simeq p$$

2. Replace each until formula in ψ by tt if its corresponding model check was successful, or ff ($\neg\, \mathsf{tt}$) otherwise.
3. Replace each atomic proposition in ψ by tt or ff depending upon its value in the initial location of A.
4. ψ is now a ground term, i.e. truth values combined by a propositional connective ($\neg$ and $\wedge$). Thus, it can simply be evaluated to yield a truth value. The automaton is a model of ψ if this evaluation yields tt, and is not otherwise.

We assume that when we wish to model check a property against an automaton, we are also given an *adversary* [BK98] to resolve the nondeterminism within the automaton. Without this adversary, enumerative analysis would not be possible; the provision of an adversary is a prerequisite of model checking. To understand the notion of an adversary here, we must explain in a little more detail our conceptual model of automata. We consider an automaton to operate within an *environment*, and for this environment (if unspecified) to be the most general environment possible, and to permit all behaviours of the automaton[2].

[2] This follows closely the CSP [Hoa85] notion of process and environment, where the environment, if unspecified, is taken to be the most nondeterministic process.

We can then think of an adversary as an environment which is deterministic with respect to the automaton, and therefore resolves all nondeterminism within it.

If, for example, we were checking a property of a single kiosk, we could choose the adversary to perform the *open* action immediately, and to *close* the kiosk again after three hours.

This recipe employs standard techniques apart from the individual checking that $A \models [\phi_1 \mathcal{U}_{\sim c} \phi_2] \simeq p$ and this is what our two algorithms address.

The first algorithm – using region trees The first algorithm (called the *region tree algorithm*) has similarities to the region graph construction of [AD94], [ACD93], [KNSS00]. In general for this algorithm, we must assume that the clock distribution functions are continuous within the range[3] of the function, in order to ensure that the probability of two clocks expiring at the same time is zero.

The algorithm works by unfolding the automaton to construct a *region tree*, and at each stage in the unfolding using the temporal logic formula to construct a probabilistic region tree. The regions are formed using the notion of valuation equivalence. A valuation records the values of all the clocks in a particular state at a particular moment in time. The unique clock $a \in \mathcal{C}$, which we add to the set of clocks, is used to facilitate the model checking. It keeps track of the total time elapsed in the execution of the stochastic automaton, but plays no part in the behaviour of the automaton.

Definition 4 A *valuation* is a function $v : \mathcal{C} \bigcup \{a\} \to \mathbb{R} \bigcup \{\perp\}$ such that $v(x) = \perp$ or $v(x) \leq x_{max}$, where x_{max} is the maximum value to which clock x can be set. If $d \in \mathbb{R}_{\geq 0}$, $v - d$ is defined by $\forall\, x \in \mathcal{C} \bigcup \{a\}.(v-d)(x) \stackrel{\text{def}}{=} v(x) - d$. The function $\min(v)$ returns the value of the smallest defined clock.

Since we assume that clocks are only used in the states in which they are set, there is no need to remember their value once the state has been exited. Only the clock a maintains its value; the rest are set to $\perp$. At the initialisation of a stochastic automaton, clock a is set to the time value of the temporal formula, and all other clocks are undefined. We define this initial valuation as $\mathbf{O}_n$, if $\mathbf{O}(a) = n$.

We also need a notion of equivalence between the valuations, which we will use to construct a finite number of regions at each node within the probabilistic region tree.

Definition 5 Two clock valuations v and v' are *equivalent* (denoted $v \cong v'$) provided the following conditions hold:

- For each clock $x \in \mathcal{C} \bigcup \{a\}$, either both $v(x)$ and $v'(x)$ are defined, or $v(x) = \perp$ and $v'(x) = \perp$.
- For every (defined) pair of clocks $x, y \in \mathcal{C} \bigcup \{a\}.v(x) < v(y) \Leftrightarrow v'(x) < v'(y)$.

[3] The *range* of a function F_x is given by the set $\{t \mid F'_x(t) > 0\}$.

The same clocks are defined in each valuation, and the order of the values of the defined clocks is all that is important, since the actions are triggered by the first clock to expire. Therefore we only need to know whether one clock is greater than or less than another.

In building the region tree, each level of unfolding comprises two steps. First, the regions within the region tree are formed by distinguishing the equivalence classes at each node, then the nodes which can be reached given these equivalence classes are calculated using the SA.

The probabilistic region tree records the resolution of the nondeterministic choices and the probabilities at the final nodes represent the chances of taking the particular sequence of actions that end in that node.

At each iteration, we update the information we have on the probability of a path satisfying the formula. To do this, we define three new propositions, and each node of the probabilistic region tree is labelled with p, f or u: p, if it has *passed* (it is the end of a path which models the bounded until formula ψ); f, if it has *failed* (it is the end of a path which cannot model ψ), or u, if it is *undecided*. We also have two global variables, $\Sigma\mathsf{p}$ and $\Sigma\mathsf{f}$, which keep running totals of the probabilities of the *pass* and *fail* paths.

The basic idea of the model checking algorithm is that we check the values of $\Sigma\mathsf{p}$ and $\Sigma\mathsf{f}$ at each stage, and if we cannot deduce from these the truth or falsity of the formula we are checking, we look more closely at the undecided nodes. That is, we extend the undecided paths by each possible subsequent action, label these new nodes p, f or u, and calculate their probabilities. We then add these probabilities to $\Sigma\mathsf{p}$ and $\Sigma\mathsf{f}$ and repeat.

To determine the probabilities on the arcs, we need to use probability density functions of the distribution functions, and integrate these in the order given by the valuation equivalence class. It is this integration that is the cause of the complexity in this region tree algorithm.

As an example, consider the formula mentioned earlier: $[\mathsf{tt}\ \mathcal{U}_{<60} K_4] > 0.3$, which states that the probability of the first car processed by the kiosk being processed within one minute from the kiosk opening is greater than 0.3. Even though the kiosk contains a deterministically set clock (c_2 is set to 12), we can analyse it using the region tree algorithm because no other clocks can expire at the same time.

An example of a nondeterministic region tree is shown in Figure 10. Consider first the SA in Figure 9. When the clock x fires, both transitions a and b are enabled, because both are governed by x. This gives rise to the nondeterministic region tree in Figure 10, and if we are to model check such a region tree, the nondetministic choice between a and b must be resolved by an adversary.

The region tree for this example is shown in Figure 11. Because there are no nondeterministic choices in this region tree, the probabilistic region tree will be structurally identical, the only difference being the labelling. For this reason, we do not present the probabilistic region tree.

Consider region K_1 first. Since we are interested in the behaviour of the kiosk after it opens, we have an adversary which makes the action *open* happen

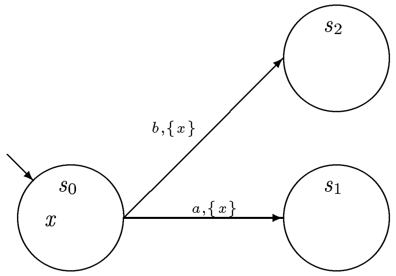

Fig. 9. A nondeterministic Stochastic Automaton

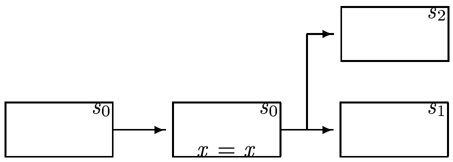

Fig. 10. A nondeterministic region tree

immediately. Thus the automaton moves to state K_2, where the clock c_2 is set. Since clock c_2 is deterministically set to 12, we only consider the valuation equivalence $c_2^1 < a^0$, and the region graph moves from region 0 to region 1. The superscript indicates that this is the first time the clock has been set.

The automaton moves to state K_3 when clock c_2 expires, this is represented by the transition from region 1 to region 2. The clock a has not expired by this state, since it is greater than clock c_2, which has only just expired, but we have not yet reached state K_4, so in the probabilistic region tree we would label this state u (undecided). In this state clock c_3 is set, and there are two valuation equivalences(where a_1 is the value of clock a at the time of transition): $c_3 < a^1$ (represented by region 3) and $a^1 < c_3$ (represented by region 4). Both of these moves will move the automaton to state K_4 when clock c_3 expires, but in one instance (region 4 to region 6) it is too late, because clock a has already expired, and so more than 60 seconds have passed. Region 6 is therefore labelled f in the probabilistic region tree. In the other instance (region 3 to region 5) state K_4 has been reached within 60 seconds, and it is therefore labelled p.

To determine the exact probability of reaching region 5 (and any other regions labelled with p), we need to use the probability density functions associated with the clocks. In our example, since we know that the transition from state K_2 to state K_3 occurs at precisely 12 seconds, a_1 is 48, the problem reduces to solving the integration

$$\int_0^{48} \mathsf{P}_{c_3}\, dt$$

which, since $\mathsf{P}_{c_3}(t) = 0$ when t is less than 30, is equal to

$$\frac{1}{450}\int_{30}^{48}(t-30)\,dt$$

which evaluates to 0.36, and so the formula is true.

This method could easily be adapted to answer queries such as "What is the probability of reaching a certain state within a certain time?" and could return a precise answer.

When the time of occurrence of one event may be dependent on the time of occurrence of a large number of other events, all the probabilistic density functions must be considered in order to calculate the probability of occurrence of one event, and the integrals which result become very complex. In order to avoid this, we consider a second algorithm, which uses discretisation.

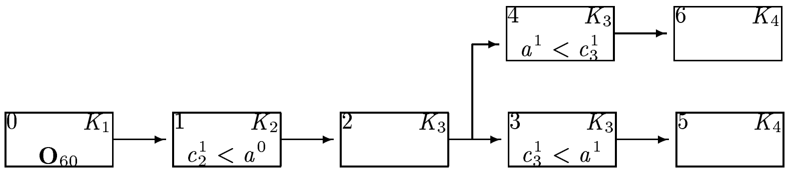

Fig. 11. Diagram for region tree algorithm

The second algorithm – approximations using discretisation This algorithm avoids the calculation of integrals that the region tree algorithm was forced to undertake. In order for the discretisation to be possible we need to make a number of assumptions. In particular, we assume that the range of the clock functions 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. For example, the distributions on the stochastic automata given in Figures 2 and 3 conform to this template. The template also allows deterministic timing since the upper and lower bounds of an interval may be of the formula $[\phi_0\,\mathcal{U}_{\leq c}\,\phi_1]$ being satisfied at this point.[4] To build the next snapshot, the algorithm picks out at each time point $n\delta$ the transitions that the automaton is capable of during the next interval of length δ. Because δ is less than the minimum of all the clock lower bounds, a maximum of one transition per path can occur in each interval. Recording all possible states of the automaton at each time point is therefore enough to record all the possible transitions.

4 We also require that $\exists\, n.n\delta = c$, which ensures that one of the snapshots will be at exactly time c.

A snapshot is built by deriving a matrix for each state s and time t (which is a rational number and calculated as $n\delta$), denoted $matrix(s,t)$, and placing in this matrix a record of the probabilities of the various combinations of clock values in state s at time t. Each matrix will have as many dimensions as its state has clocks.

Each entry in the matrix $matrix(s,t)$ is the probability that at time point t, the automaton is in state s, and each clock is within a particular time range. Thus, the value $matrix(s,t)[k_1 \dots k_n]$ is the probability that at time point t, the automaton is in state s, and $v(c_i) \in (\delta(k_i - 1), \delta k_i]$ for each clock c_i.

The algorithm stops when either enough information has been gathered to determine the truth or falsity of the formula, or enough time has passed so that $n\delta > c$, and allowing time to pass further will make no difference to the information we already have. In this case the result undecided is returned.

$(K_2, 0)$

0	1	
10	20	c_2

(K_2, δ)

1	0	
10	20	c_2

$(K_3, 2\delta)$

0	0	0	$\frac{1}{9}$	$\frac{3}{9}$	$\frac{5}{9}$	
10	20	30	40	50	60	c_3

$(K_3, 3\delta)$

0	0	$\frac{1}{9}$	$\frac{3}{9}$	$\frac{5}{9}$	0	
10	20	30	40	50	60	c_3

$(K_3, 4\delta)$

0	$\frac{1}{9}$	$\frac{3}{9}$	$\frac{5}{9}$	0	0	
10	20	30	40	50	60	c_3

$(K_3, 5\delta)$

$\frac{1}{9}$	$\frac{3}{9}$	$\frac{5}{9}$	0	0	0	
10	20	30	40	50	60	c_3

$(K_3, 6\delta)$

$\frac{3}{9}$	$\frac{5}{9}$	0	0	0	0	
10	20	30	40	50	60	c_3

$(K_4, 6\delta)$

0	0	0	$\frac{1}{81}$	$\frac{3}{81}$	$\frac{9}{81}$	
10	20	30	40	50	60	c_3

Fig. 12. matrices for the second algorithm

Consider again the formula $[\mathsf{tt}\, \mathcal{U}_{\leq 60} K_4] > 0.3$. We choose δ to be 10. The first matrix to be constructed would be $m(K_1, 0)$, but the state K_1 has no associated clocks, therefore the automaton moves immediately to state K_2, and $matrix(K_2, 0)$ is constructed (see Figure 12).

This matrix tells us that the probability of clock c_2 being somewhere between the values 10 and 20 at time zero is 1.

There are two different procedures for updating a matrix (that is, to derive $matrix(s, \delta(n+1))$ from the matrices referring to time $\delta(n)$), both of which correspond to different situations. The first corresponds to the situation within the stochastic automaton where time passes, but the state remains unchanged. In this case we must shift the clock configuration probabilities in the previous matrix down by one index step (which corresponds to δ time passing) and add the result to the matrix we are updating.

This is the situation here, and $matrix(K_2, 10)$ is formed as in Figure 12.

The second procedure is applied when new states can be reached from the current state during the δ time passing, and involves determining the probability of entering these states. We do this by looking at all the probability values in the matrix where at least one of the indices has the lowest possible value (10, in this example). If this is the case then we know that at least one clock will expire during the ensuing δ timestep.

If only one index in the configuration has the value 10 then only one clock can expire, and only one state can be entered from this clock configuration, and so the matrix for that state is built.

This is the case from $matrix(K_2, d)$, and so we get $matrix(K_3, 2\delta)$, which tells us that the probability of being within state K_3 at time 2δ with clock c_3 between values 30 and 40 is $\frac{1}{9}$, being within state K_3 at time 2δ with clock c_3 between values 40 and 50 is $\frac{3}{9}$ and being within state K_3 at time 2δ with clock c_3 between values 50 and 60 is $\frac{5}{9}$.

If more than one index has the value 10, then we simply do not explore that configuration any further, and the configuration probability is added to *error*. In the example we are considering, this possibility does not occur.

In our example, the matrices $matrix(K_3, 3\delta)$, $matrix(K_3, 4\delta)$, $matrix(K_3, 5\delta)$ and $matrix(K_3, 6\delta)$ can all be constructed simply by moving the clock configuration probabilities with the previous matrices.

The second way to update a matrix corresponds to a transition from one state to another within the automaton. For each matrix entry we calculate the clock configuration probability, multiply it by the probability of moving into this state at this time, and add it to the matrix entry we are updating. Thus, in the example, we get $matrix(K_4, 6\delta)$,

We have now reached the timepoint 6δ, which corresponds to 60 seconds, and so the sum of all the probability values in the matrix at this point ($\frac{1}{81} + \frac{3}{81} + \frac{5}{81} = \frac{1}{3}$) is a lower bound on the probability that state K_4 will have been reached by time 60. Thus, since we are interested in whether the probability is greater than 0.3, we can conclude that the formula is true. A smaller δ would produce a more accurate result, but we do not illustrate that here.

5 Conclusions

In this paper we have begun to tackle the problem of integrating various automaton based notations within a specification. Specifically, we have

- given a translation from stochastic automata to timed automata with deadlines and shown which properties are retained;
- presented two methods for model checking stochastic automata, the first of which builds regions from the automaton, and uses integration of the probability density functions and the second of which uses an approximation technique based on discretisation.

Translating stochastic automata to timed automata with deadlines means that, although the stochastic information is lost, we can analyse the composed specification for temporal properties to do with, for example, throughput within a certain time.

The model checking methods cannot consider the composed specification, and must be restricted to the individual components, although the effects of the environment may be represented by the adversary chosen.

The two model checking methods presented complement each other. The region method is best used when the size of the model to be explored is small, because the number of integrations to be performed goes up exponentially with the number of clocks. The discretisation method is more promising for larger models. It can produce upper and lower bounds on the probabilities, and is therefore best suited for queries such as "Does the probability of reaching a state s by a time t lie within the range $[a, b]$?"

We are currently seeking to implement the second algorithm, and to integrate it with the LUSCETA [JB99] tool. We would also like to consider how to model check more general stochastic automata, and in particular to allow clocks to be set and used in any state.

Acknowledgments We are indebted to Pedro D'Argenio for supplying us with Definition 3 [D'A].

References

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[AD94] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BBD00] Howard Bowman, Jeremy Bryans, and John Derrick. A model checking algorithm for stochastic systems. Technical Report 4-00, University of Kent at Canterbury, 2000.

[BD99] Jeremy Bryans and John Derrick. Stochastic specification and verification. In *Proceedings of Third Irish Workshop in Formal Methods*, 1999.

[BK98] Christel Baier and Marta Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, May 1998.

[BS98] Sébastien Bornot and Joseph Sifakis. On the composition of hybred systems. In *International Workshop on Hybred Systems: Computation and Control*, volume LNCS 1386, pages 49–63, April 1998.

[BST98] Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling urgency in timed systems. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *International Symposium: Compositionality — The significant difference*, volume 1536 of *LNCS*. Springer-Verlag, 1998.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[D'A] Pedro D'Argenio. Personal communication.

[DKB98] 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 Working Conference on Programming Concepts and Methods*. Chapman & Hall, 1998.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[JB99] Trevor Jones and Lynne Blair. A Tool-Suite for Simulating, Composing and Editing Timed Automata (LUSCETA: Users Manual Release 1.0). Technical Report MPG-99-24, Computing Department, Lancaster University, 1999.

[KNSS00] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying Quantitative Properties of Continuous Probabilistic Real-Time Graphs. In *CONCUR'00*, 2000. Also available as a University of Birmingham technical report: CSR-00-06.