Formal Methods in System Design manuscript No.
(will be inserted by the editor)

Model Checking for Probabilistic Timed Automata

Gethin Norman · David Parker · Jeremy Sproston

Received: date / Accepted: date

Abstract Probabilistic timed automata (PTAs) are a formalism for modelling systems whose behaviour incorporates both probabilistic and real-time characteristics. Applications include wireless communication protocols, automotive network protocols and randomised security protocols. This paper gives an introduction to PTAs and describes techniques for analysing a wide range of quantitative properties, such as "the maximum probability of the airbag failing to deploy within 0.02 seconds", "the maximum expected time for the protocol to terminate" or "the minimum expected energy consumption required to complete all tasks". We present a temporal logic for specifying such properties and then give a survey of available model-checking techniques for formulae specified in this logic. We then describe two case studies in which PTAs are used for modelling and analysis: a probabilistic non-repudiation protocol and a task-graph scheduling problem.

1 Introduction

Automated verification techniques, such as model checking, provide powerful methods for rigorously analysing the correctness of systems. Increasingly, this analysis must also take into account *quantitative* aspects of the systems being verified, including both *real-time* characteristics and *probabilistic* behaviour. Embedded systems, for example, whether in communication and multimedia devices or in automotive and avionic control systems, often operate under timing constraints. The need for automated formal verification techniques in this domain is clear, as evidenced by the take-up of timed automata verification tools such as UPPAAL [13]. On the other hand, many real-life systems also exhibit stochastic behaviour,

Gethin Norman
School of Computing Science, 18 Lilybank Gardens, University of Glasgow, Glasgow, G12 8RZ
E-mail: gethin.norman@glasgow.ac.uk

David Parker
School of Computer Science, University of Birmingham, Edgbaston, Birmingham, B15 2TT
E-mail: d.a.parker@cs.bham.ac.uk

Jeremy Sproston
Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy
E-mail: sproston@di.unito.it

due, for example, to component failures, unreliable communication media or the use of randomisation. Probabilistic verification tools such as PRISM [51] and MRMC [45] have been widely used to analyse many systems with stochastic behaviour. Another vital ingredient in system modelling is *nondeterminism*, which is often used to capture concurrency between parallel components and to under-specify or abstract certain aspects of a system.

Probabilistic timed automata (PTAs) [35,54,11] are a modelling formalism for systems that exhibit probabilistic, nondeterministic *and* real-time characteristics. In many application domains, all three aspects need to be modelled; these include wireless communication protocols such as Bluetooth or Zigbee, automotive network protocols such as FlexRay, randomised security protocols, e.g. for anonymity or non-interference, and many others. The interplay between these different aspects can be subtle, making automated verification techniques and tool support essential. Verification of PTAs permits analysis of a wide range of quantitative properties, from reliability to performance, e.g.:

- "the maximum probability of an airbag failing to deploy within 0.02 seconds";
- "the minimum probability that a packet is correctly delivered with 1 second";
- "the maximum expected time for the protocol to terminate".

PTAs can also be augmented with additional quantitative information in the form of *costs* or *rewards*. The resulting model is sometimes referred to as *priced probabilistic timed automata* and allows reasoning about a wide range of additional properties, e.g.:

- "the maximum expected number of lost packets within the first hour";
- "the minimum expected energy consumption for completion of all tasks";
- "the maximum number of queued requests after 10 seconds of operation".

This paper provides an introduction to PTAs and the techniques that have been developed to specify and verify properties such as those listed above.

Paper structure. This paper is organised as follows. After background material in Section 2, Section 3 introduces the model of PTAs and discusses various issues relating to their use and analysis. Then, Section 4 presents a probabilistic temporal logic to represent properties of PTAs and Section 5 surveys the various techniques that can be used to perform model checking of this logic. Section 6 describes two case studies illustrating the usage of PTAs and their associated model-checking algorithms: a probabilistic non-repudiation protocol and a task-graph scheduling problem. Throughout the paper we give pointers to the relevant literature on PTAs and describe related work.

2 Background

We use $\mathbb{R}_{\geq 0}$ to denote the set of non-negative real numbers, $\mathbb{Q}_{\geq 0}$ for the set of non-negative rationals and $\mathbb{N}$ for the set of natural numbers. A discrete probability *distribution* over a countable set Q is a function $\mu : Q \rightarrow [0,1]$ such that $\sum_{q \in Q} \mu(q) = 1$. For a function $\mu : Q \rightarrow \mathbb{R}_{\geq 0}$ we define $\mathit{Support}(\mu) = \{q \in Q \mid \mu(q) > 0\}$. Then, for an arbitrary set Q, we define $\mathit{Dist}(Q)$ to be the set of functions $\mu : Q \rightarrow [0,1]$ such that $\mathit{Support}(\mu)$ is a countable set and μ restricted to $\mathit{Support}(\mu)$ is a distribution. For $q \in Q$, let μ_q be the *point distribution at q* which assigns probability 1 to q. Let AP be a set of *atomic propositions*, which we assume to be fixed throughout the paper.

Markov decision processes (MDPs) are a widely used formalism for modelling systems that exhibit both nondeterministic and probabilistic behaviour. In this paper, we will use *timed probabilistic systems* (TPSs) [52,56], an extension of MDPs in which transitions are labelled with either an action or a time duration.

Definition 1 (TPS) A *timed probabilistic systems* T is a tuple $(S, \bar{s}, Act, Steps_{\mathsf{T}}, lab)$ where S is (possibly infinite) a set of *states*, $\bar{s} \in S$ an *initial state*, Act a (finite) set of *actions*, $Steps_{\mathsf{T}} : S \times (Act \cup \mathbb{R}_{\geq 0}) \rightarrow Dist(S)$ a (partial) *probabilistic transition function* and $lab : S \rightarrow 2^{AP}$ a *labelling function.*

A TPS T starts in the initial state $\bar{s}$ and, when in state $s \in S$, there is a nondeterministic choice between one or more available actions or time durations $a \in Act \cup \mathbb{R}_{\geq 0}$ (those for which $Steps_{\mathsf{T}}(s,a)$ is defined). After the choice of an available action or time duration a, a successor state s' is selected at random according to the probability distribution $Steps_{\mathsf{T}}(s,a)$. We use the notation $s \xrightarrow{a,\mu} s'$ for such a transition, i.e., to denote that $Steps_{\mathsf{T}}(s,a)$ is defined and equal to μ, and that $s' \in Support(\mu)$. We assume, for each state $s \in S$, there exists at least one available action or time duration. An MDP M is a special case of a TPS where time is omitted from the transition function, i.e., the probabilistic transition function takes the form $Steps_{\mathsf{M}} : S \times Act \rightarrow Dist(S)$.

A path of a TPS represents a particular resolution of both the nondeterminism and probability present in the system. Formally, a path is a finite or infinite sequence of probabilistic transitions alternating between time durations and actions, for example:

$$\omega = s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} s_2 \xrightarrow{a_2,\mu_2} \cdots$$

where $a_{2i} \in \mathbb{R}_{\geq 0}$ and $a_{2i+1} \in Act$ for $i \in \mathbb{N}$. We denote by $\omega(i)$ the $(i{+}1)$th state s_i of ω and the accumulated duration up until this state $\omega(i)$ is defined by:

$$dur_{\omega}(i) \overset{\text{def}}{=} \textstyle\sum_{0 \leq j < i \wedge a_j \in \mathbb{R}_{\geq 0}} a_j .$$

A *position* of ω is a pair $(i,t) \in \mathbb{N} \times \mathbb{R}_{\geq 0}$ such that $t \leq dur_{\omega}(i{+}1) - dur_{\omega}(i)$. We say that the position (j,t') precedes the position (i,t), written $(j,t') \prec (i,t)$, when $j{<}i$ or $j{=}i$ and $t'{<}t$.

To reason about the probabilistic behaviour of a TPS T, we use the notion of an *adversary*, which is a possible resolution of nondeterminism only. Formally, an adversary is a function from finite paths with an even number of transitions to available time durations, and from finite paths with an odd number of transitions to available actions). For a fixed adversary σ and state s, we can define a probability measure $Pr^{\sigma}_{\mathsf{T},s}$ over the set $Path^{\sigma}_{\mathsf{T},s}$ of infinite paths starting in s corresponding to σ [47]. For a real-valued random variable f over $Path^{\sigma}_{\mathsf{T},s}$, we let $\mathbb{E}^{\sigma}_{\mathsf{T},s}(f)$ denote the expected value of f with respect to $Pr^{\sigma}_{\mathsf{T},s}$.

We restrict our attention to *time-divergent* (or *non-Zeno*) adversaries, i.e., we do not consider executions in which time does not advance beyond a certain point. These can be ignored on the grounds that they do not correspond to actual, realisable behaviour of the system being modelled [2,40]. Formally, an adversary σ of a TPS T is time divergent if:

$$Pr^{\sigma}_{\mathsf{T},s}(\{\omega \in Path^{\sigma}_{\mathsf{T},s} \,|\, \forall c \in \mathbb{N}. \exists i \in \mathbb{N}. dur_{\omega}(i) {>} c\}) = 1 \,.$$

for all states s of T. We denote by Adv_{T} the set of all time-divergent adversaries of T. This issue is discussed in more depth in Section 3.

We next introduce *rewards* (or, equivalently, *costs* or *prices*) for TPSs. These are used to represent additional information about the system that the TPS represents, e.g., the number of packets sent or requests lost, the time spent in a particular state or the energy consumed.

Definition 2 (Reward structure) A *reward structure* for a TPS $\mathsf{T}{=}(S, \bar{s}, Act, Steps_{\mathsf{T}}, lab)$ is a pair $r{=}(r_S, r_{Act})$ where $r_S : S \rightarrow \mathbb{R}_{\geq 0}$ is a *state reward function* and $r_{Act} : (S \times Act) \rightarrow \mathbb{R}_{\geq 0}$ is an *action reward function.*

For a reward structure $r{=}(r_S, r_{Act})$ and state s, the value $r_S(s)$ defines the *rate* (per time unit) at which reward is accumulated when in state s. On the other hand, for state s and action a, the value $r_{Act}(s,a)$ defines the reward acquired when the action a is taken in state s. More formally, for any infinite path $\omega = s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \cdots$, the reward accumulated during the transition of ω from state s_i to s_{i+1} is defined by:

$$r(\omega,i) \overset{\text{def}}{=} \begin{cases} r_S(s_i){\cdot}a_i & \text{if } a_i \in \mathbb{R}_{\geq 0} \text{ (or, equivalently, if } i \bmod 2 = 0) \\ r_{Act}(s_i,a_i) & \text{otherwise.} \end{cases}$$

Alternatively, we can also interpret state rewards as defining a reward at a particular time instant. An example of this usage would be a reward structure that represents the number of messages stored in a queue at a particular time instant. When using this interpretation, action reward values are not considered.

3 Probabilistic Timed Automata

Probabilistic timed automata (PTAs) [42,54,11] model real-time behaviour in the same fashion as classical timed automata [4], using *clocks*. Clocks are variables whose values range over the non-negative reals and which increase at the same rate as time. Throughout this paper, we assume a finite set of clocks $\mathscr{X}$. A function $v : \mathscr{X} {\to} \mathbb{R}_{\geq 0}$ is referred to as a *clock valuation* and the set of all clock valuations is denoted by $\mathbb{R}_{\geq 0}^{\mathscr{X}}$. For any $v \in \mathbb{R}_{\geq 0}^{\mathscr{X}}$, $t \in \mathbb{R}_{\geq 0}$ and $X \subseteq \mathscr{X}$, we use $v{+}t$ to denote the clock valuation which increments all clock values in v by t and $v[X{:=}0]$ for the clock valuation in which clocks in X are reset to 0. We use $\mathbf{0}$ to denote the clock valuation that assigns 0 to all clocks in $\mathscr{X}$.

The set of *clock constraints* over $\mathscr{X}$, denoted $CC(\mathscr{X})$, is defined by the syntax:

$$\chi ::= \texttt{true} \mid x \leq d \mid c \leq x \mid x{+}c \leq y{+}d \mid \neg\chi \mid \chi \wedge \chi$$

where $x,y \in \mathscr{X}$ and $c,d \in \mathbb{N}$. A clock valuation v satisfies a clock constraint χ, denoted by $v \models \chi$, if χ resolves to `true` when substituting each occurrence of clock x with $v(x)$. The set of valuations satisfying a clock constraint is called a *zone*. Clock constraints will be used in the syntactic definition of PTAs and for the specification of properties.

Definition 3 (PTA syntax) A *probabilistic timed automaton* (PTA) is defined by a tuple $\mathsf{P}{=}(L, \bar{l}, \mathscr{X}, Act, inv, enab, prob, \mathscr{L})$ where:

- L is a finite set of *locations* and $\bar{l} \in L$ is an *initial location*;
- $\mathscr{X}$ is a finite set of *clocks*;
- Act is a finite set of *actions*;
- $inv : L{\to}CC(\mathscr{X})$ is an *invariant condition*;
- $enab : L{\times}Act{\to}CC(\mathscr{X})$ is an *enabling condition*;
- $prob : L{\times}Act{\to}Dist(2^{\mathscr{X}}{\times}L)$ is a (partial) *probabilistic transition function*;
- $\mathscr{L} : L{\to}2^{AP}$ is a *labelling function* mapping each location to a set of atomic propositions.

A *state* of a PTA is a pair $(l,v) \in L{\times}\mathbb{R}_{\geq 0}^{\mathscr{X}}$ such that $v \models inv(l)$. In any state (l,v), either a certain amount of time $t \in \mathbb{R}_{\geq 0}$ elapses, or an action $a \in Act$ is performed. If time elapses, then the choice of t requires that the invariant $inv(l)$ remains continuously satisfied while time passes. The resulting state after this transition is $(l,v{+}t)$ and, to ease notation, we denote this state by $(l,v){+}t$. In the case where an action is performed, an action a can only be chosen if it is *enabled*, i.e., if the clock constraint $enab(l,a)$ is satisfied by v. Once an enabled action a is chosen, a set of clocks to reset and a successor location are selected at random, according to the distribution $prob(l,a)$. We call each element $(X,l') \in 2^{\mathscr{X}} \times L$ in the support of $prob(l,a)$ an *edge* and use $edges(l,a)$ to denote the set of such edges.

We assume that PTAs are *well-formed*, meaning that, for each state (l,v) and action a such that v satisfies $enab(l,a)$, every edge $(X,l') \in edges(l,a)$ results in a transition to a valid state, i.e., we have $v[X{:=}0] \models inv(l')$. A PTA can be transformed into one that is well-formed by incorporating the invariant associated with the target location into the enabling condition of each location-action pair (see [56]).

Definition 4 (PTA semantics) Let $\mathsf{P}{=}(L,\bar{l},\mathscr{X},Act,inv,enab,prob,\mathscr{L})$ be a PTA. The semantics of P is defined as the (infinite-state) TPS $[\![\mathsf{P}]\!] = (S,\bar{s},Act,Steps_{\mathsf{P}},lab)$ where:

- $S = \{(l,v) \in L \times \mathbb{R}_{\geq 0}^{\mathscr{X}} \mid v \models inv(l)\}$ and $\bar{s} = (\bar{l},\mathbf{0})$;
- for any $(l,v) \in S$ and $a \in Act \cup \mathbb{R}_{\geq 0}$, we have $Steps_{\mathsf{P}}((l,v),a) = \lambda$ if and only if either:
  Time transitions. $a \in \mathbb{R}_{\geq 0}$, $v{+}t' \models inv(l)$ for all $0{\leq}t'{\leq}a$, and $\lambda = \mu_{(l,v+a)}$;
  Action transitions. $a \in Act$, $v \models enab(l,a)$ and for each $(l',v') \in S$:

$$\lambda(l',v') = \sum \{\!|\, prob(l,a)(X,l') \mid X \in 2^{\mathscr{X}} \wedge v' = v[X{:=}0] \,|\!\}\,,$$

- for any $(l,v) \in S$ we have $lab(l,v) = \mathscr{L}(l)$.

Example of a PTA. In Figure 1, we present a PTA modelling a simple communication protocol. We adopt the standard conventions for the graphical representation of timed automata. Distributions are represented by an arc connecting edges at their source and by probability labels attached to edges (omitted for edges taken with probability 1). The PTA has two clocks x and y, which start with the value 0. In the location init, the system waits for at least 1 time unit (represented by the enabling condition $x{\geq}1$ on the outgoing distribution of action *send*) and at most 2 time units (represented by conjunct $x{\leq}2$ of the invariant condition), before sending a message. With probability 0.9 the message is received correctly (edge to done); otherwise, with probability 0.1, the message is lost (edge to lost). In the latter case, once clock x reaches 8, the PTA returns to init where another attempt to send the message can be made. If, in total, at least 20 and at most 25 time units have elapsed since the start of system execution, the PTA performs a timeout and moves to location fail.

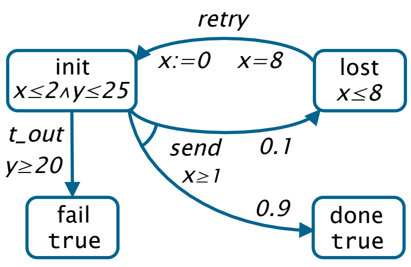


Fig. 1 Example of a PTA

Rewards for PTAs. We can also define *rewards* for a PTA. Often, these model usage of some resource and are equivalently referred to as *costs* or *prices*. A *reward structure* at the level of a PTA P is defined using a pair $\mathbf{r}{=}(\mathbf{r}_L,\mathbf{r}_{Act})$, where $\mathbf{r}_L : L \to \mathbb{R}_{\geq 0}$ is a function assigning to each location the rate at which rewards are accumulated as time passes in that location and $\mathbf{r}_{Act} : L{\times}Act{\to}\mathbb{R}_{\geq 0}$ is a function assigning the reward of executing each action in each location. The corresponding reward structure of the TPS $[\![\mathsf{P}]\!]$ is given by $r{=}(r_S,r_{Act})$ where $r_S(l,v){=}\mathbf{r}_L(l)$ and $r_{Act}((l,v),a){=}\mathbf{r}_{Act}(l,a)$ for $(l,v) \in L{\times}\mathbb{R}_{\geq 0}^{\mathscr{X}}$ and $a \in Act$. PTAs equipped with reward structures are a probabilistic extension of linearly-priced timed automata (also known as weighted timed automata) [14,5].

Modelling with PTAs. We now summarise a variety of extensions to the standard definition of PTAs that facilitate high-level modelling using this formalism.

Parallel composition. It is often useful to define complex systems as the *parallel composition* of several interacting components. The definition of the parallel composition operator $\|$ for PTAs [55] uses ideas from (untimed) probabilistic automata [62] and classical timed automata [4]. Let $\mathsf{P}_i = (L_i, \bar{l}_i, Act_i, \mathscr{X}_i, inv_i, enab_i, prob_i, \mathscr{L}_i)$ for $i \in \{1,2\}$ and assume that $\mathscr{X}_1 \cap \mathscr{X}_2 = \emptyset$. Given $\mu_1 \in Dist(2^{\mathscr{X}_1} \times L_1)$ and $\mu_2 \in Dist(2^{\mathscr{X}_2} \times L_2)$, we let $\mu_1 \otimes \mu_2 \in Dist(2^{\mathscr{X}_1 \cup \mathscr{X}_2} \times (L_1 \times L_2))$ be such that $\mu_1 \otimes \mu_2(X_1 \cup X_2, (l_1, l_2)) = \mu_1(X_1, l_1) \cdot \mu_2(X_2, l_2)$ for $X_i \subseteq \mathscr{X}_i$, $l_i \in L_i$ and $i \in \{1,2\}$. The *parallel composition* of PTAs P_1 and P_2 is the PTA:

$$\mathsf{P}_1 \| \mathsf{P}_2 = (L_1 \times L_2, (\bar{l}_1, \bar{l}_2), \mathscr{X}_1 \cup \mathscr{X}_2, Act_1 \cup Act_2, inv, enab, prob, \mathscr{L})$$

such that, for each location pair $(l_1, l_2) \in L_1 \times L_2$ and action $a \in Act_1 \cup Act_2$:

- the invariant condition is given by $inv(l_1, l_2) = inv_1(l_1) \wedge inv_2(l_2)$;
- the enabling condition is given by:

$$enab((l_1,l_2),a) = \begin{cases} enab_1(l_1,a) \wedge enab_2(l_2,a) & \text{if } a \in Act_1 \cap Act_2 \\ enab_1(l_1,a) & \text{if } a \in Act_1 \setminus Act_2 \\ enab_2(l_2,a) & \text{if } a \in Act_2 \setminus Act_1; \end{cases}$$

- the probabilistic transition function is given by:

$$prob((l_1,l_2),a) = \begin{cases} prob_1(l_1,a) \otimes prob_2(l_2,a) & \text{if } a \in Act_1 \cap Act_2 \\ prob_1(l_1,a) \otimes \mu_{(\emptyset,l_2)} & \text{if } a \in Act_1 \setminus Act_2 \\ \mu_{(\emptyset,l_1)} \otimes prob_2(l_2,a) & \text{if } a \in Act_2 \setminus Act_1; \end{cases}$$

- the labelling function is given by $\mathscr{L}(l_1, l_2) = \mathscr{L}_1(l_1) \cup \mathscr{L}_2(l_2)$.

If PTA P_i has associated reward structure $(\mathbf{r}^i_L, \mathbf{r}^i_{Act})$, then the reward structure $\mathbf{r}{=}(\mathbf{r}_L, \mathbf{r}_{Act})$ for $\mathsf{P}_1 \| \mathsf{P}_2$ is such that, for $(l_1,l_2) \in L_1 \times L_2$ and $a \in Act_1 \cup Act_2$, we have $\mathbf{r}_L(l_1,l_2){=}\mathbf{r}^1_L(l_1){+}\mathbf{r}^2_L(l_2)$, $\mathbf{r}_{Act}((l_1,l_2),a) = \mathbf{r}^1_{Act}(l_1,a){+}\mathbf{r}^2_{Act}(l_2,a)$ if $a \in Act_1 \cap Act_2$, $\mathbf{r}_{Act}((l_1,l_2),a) = \mathbf{r}^1_{Act}(l_1,a)$ if $a \in Act_1 \setminus Act_2$ and $\mathbf{r}_{Act}((l_1,l_2),a) = \mathbf{r}^2_{Act}(l_2,a)$ if $a \in Act_2 \setminus Act_1$.

Discrete variables. When modelling systems with (probabilistic) timed automata, it is often convenient to augment the model with discrete variables [13,65]. We restrict ourselves to the case in which a finite number of variables, each with a finite domain, are added to the PTA framework; enabling conditions can then refer to the current value of the variables, and the probabilistic transition relation is extended to allow updating variable values. Such an extended PTA model can be represented in the standard PTA framework presented above in the following way. Let L be the set of locations of the extended PTA, and suppose the extended PTA has n bounded integer variables. The locations of the standard PTA are tuples comprising $n{+}1$ elements: the first element is a location from L, while the remaining n elements are values of the bounded integer variables. Enabling conditions are obtained by resolving partially the enabling conditions of the extended PTA, using the variable values corresponding to the locations and the probabilistic transition function is obtained by encoding variable updates into target locations of edges.

Urgency. When modelling real-time systems, it is often necessary to express the fact that a particular action should be taken immediately, without letting time pass. In this way, we can model, for example, an instantaneous system event comprising several atomic actions. A number of mechanisms for modelling such situations have been introduced for timed automata, for example in the system-description language of the UPPAAL model checker [13]; here, we describe how they are adapted to PTAs.

Firstly, an *urgent location* of a PTA is a location in which no time can pass. Urgent locations can be represented in the PTA framework by introducing an additional clock, which is reset on entry to an urgent location, and by including a conjunct in the invariant condition of the location to specify that the value of the clock should be equal to 0 in the location.

Secondly, a *committed location* of a PTA is, like an urgent location, a location in which no time can pass, but also must be left before any other system component makes a transition. We adapt to PTAs the method of [65] for encoding committed locations in the standard timed automata framework. First a global Boolean variable *atom* is added to the PTA. Now, consider a PTA which is to be composed in parallel with other PTAs. A committed location of the PTA is subject to the constructs added in the case of urgent locations and, in addition, *atom* is set to `true` on entry to the committed location, is set to `false` on exiting the committed location, and all enabling conditions of the PTA except those corresponding to committed locations have the conjunct requiring that *atom* is `false`.

Finally we mention *urgent actions* [13,30]. Informally, an action is urgent if it must be chosen as soon as it is enabled. Urgent actions can be introduced to the PTA syntax simply by identifying the subset Act_u of actions which are interpreted as urgent. The presence of urgent actions necessitates the following modifications to the semantics of a PTA. For PTA $\mathsf{P}{=}(L,\bar{l},Act,\mathscr{X},inv,enab,prob,\mathscr{L})$ with urgent actions Act_u, $[\![\mathsf{P}]\!]$ is the TPS $(S,\bar{s},Act,Steps_{\mathsf{P}},lab)$ where S, $\bar{s}$, lab and $Steps_{\mathsf{P}}((l,v),a)$ for $(l,v)\in S$ and $a\in Act$ are as in Definition 4, while for $(l,v)\in L$, $t\in\mathbb{R}_{\geq 0}$, we have $Steps_{\mathsf{P}}((l,v),t)=\mu_{(l,v+t')}$ if and only if:

- $v{+}t'\models inv(l)$ for all $0{\leq}t'{\leq}t$;
- for all $0{\leq}t'{<}t$ and $a'\in Act$, if $v{+}t'\models enab(l,a')$ then $a'\notin Act_u$.

Non-standard clock resets. In the PTA framework, as in the standard definition of timed automata, clocks can only be reset to value 0 when taking a probabilistic transition. It may be useful, though, to also allow clocks to be reset to any non-negative integer value. If $\mathbb{N}_\perp=\mathbb{N}\cup\{\perp\}$, v is a clock valuation and $\theta:\mathscr{X}{\to}\mathbb{N}_\perp$, then define the clock valuation $v[\theta]$ such that $v[\theta](x){=}v(x)$ if $\theta(x){=}\perp$ and $v[\theta](x){=}\theta(x)$ otherwise. Then, a *PTA with extended resets* is defined as a PTA in Definition 3, except that the probabilistic transition function is now defined as $prob:L{\times}Act{\to}Dist((\mathscr{X}{\to}\mathbb{N}_\perp){\times}L)$. The semantics of a PTA with extended resets P is defined as in Definition 4, except that for $(l,v)\in S$ and $a\in Act$, we have $Steps_{\mathsf{P}}((l,v),a){=}\lambda$ if and only if $v\models enab(l,a)$ and, for each $(l',v')\in S$:

$$\lambda(l',v')=\sum\{\!|\,prob(l,a)(\theta,l')\mid\theta\in\mathscr{X}{\to}\mathbb{N}_\perp\wedge v'{=}v[\theta]\,|\!\}\,.$$

A PTA with extended resets can be translated into a PTA with the standard restriction of clock resets to 0, by adapting an analogous construction for (non-probabilistic) timed automata [26]. However, this construction can give an exponential blow-up in the size of the model, which is unavoidable [25]. Instead, model-checking algorithms for PTAs can be developed which incorporate the extended definition of resets directly.

Channels. The definitions of actions and parallel composition presented here can be extended to allow for channels and the sending and receiving (to either single or multiple recipients) of messages along them, as in UPPAAL [13]. Such behaviour can be encoded in the action names of a standard PTA.

Time divergence. An important issue with regard to the verification of models of real-time systems is that of time divergence. As explained in Section 2, such behaviour does not correspond to that of the real system, and hence the verification technique used must be able to disregard such behaviour during analysis. We use the notion of time divergence of Section 2, i.e., we restrict our attention to the adversaries $Adv_{[\![\mathsf{P}]\!]}$ (those adversaries for which the probability of time passing beyond any bound is 1).

Note that a PTA may feature states from which time cannot diverge for any adversary; such states correspond to a probabilistic generalisation of *timelocks* in the timed automata setting [40], and are considered to indicate modelling errors. The set of timelock states can be identified using (extensions of) the analysis methods that we present in Section 5, and removed from the model by modifying the invariant and enabling conditions.

For some PTAs, all adversaries will be time-divergent by construction. We give a syntactic and compositional condition, derived from analogous results on timed automata [66, 67], which guarantees that all adversaries are time-divergent: a PTA is *structurally divergent* if, for every sequence $(l_0,a_0),(X_0,l_1),\ldots,(l_n,a_n),(X_n,l_{n+1})$ such that $(X_i,l_{i+1}) \in$ $edges(l_i,a_i)$ for $0{\le}i{<}n$ and $l_{n+1}{=}l_0$, there exists $x \in \mathscr{X}$ and $0 \le i,j \le n$ such that $x \in X_i$ and $enab(l_j,a_j) \Rightarrow (x{\ge}1)$ (i.e. $enab(l_j,a_j)$ contains a conjunct of the form $x{\ge}c$ for $c{\ge}1$). If a PTA is not structurally divergent, verification algorithms can be adjusted to disregard non-divergent adversaries, as we will describe in Section 5.

Finally we note that a more restrictive notion of divergent adversary of a PTA, namely that of a *strictly divergent* adversary, has been presented in [63]. An adversary is strictly divergent if *all* of its paths entail time passing beyond any bound. In many contexts, strictly divergent adversaries are more realistic than divergent adversaries, e.g., if we regard the edge traversals of a PTA as corresponding to the change in a physical state of the system.

Alternative models. We conclude this section with a brief discussion of some alternative probabilistic models that also incorporate nondeterministic, probabilistic and timed aspects. One example, which has recently attracted increased interest in the context of probabilistic verification, is *continuous-time Markov decision processes*, along with closely related models such as *interactive Markov chains* [41] and *Markov automata* [31]. These extend classical (discrete-time) Markov decision processes with real-time delays modelled by exponential distributions. Thus, such models can alternatively be viewed as extensions of *continuous-time Markov chains*, which also model probabilistic real-time systems using exponentially-distributed delays, but which do not exhibit nondeterministic behaviour. Other proposed models in the literature include [1], which is based on generalised semi-Markov processes, and [53], which defines PTAs with continuously distributed random delays.

4 Property Specification for PTAs

In this section, we present a temporal logic for the formal specification of quantitative properties of PTAs. The basis for this is the temporal logic PCTL [37,20], a probabilistic extension of the logic CTL [29] which has been proposed for specifying properties of both MDPs [20] and discrete-time Markov chains [37]. We augment the logic with operators to reason about rewards (or costs or prices), in the style of the logic in [32] for MDPs (and as used in the probabilistic model checker PRISM [51]). We also discuss extensions to more expressive logics such as PTCTL and PCTL*. In the next section, we will survey the available techniques for model checking of PTAs against properties specified in our logic.

Definition 5 (Syntax) The syntax of our logic is given by the following grammar:

$$\begin{aligned}
\phi &::= \mathtt{true} \;\big|\; \mathsf{a} \;\big|\; \chi \;\big|\; \phi \wedge \phi \;\big|\; \neg\phi \;\big|\; \mathtt{P}_{\bowtie p}[\psi] \;\big|\; \mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho] \\
\psi &::= \phi \;\mathtt{U}^{\le k}\; \phi \;\big|\; \phi \;\mathtt{U}\; \phi \\
\rho &::= \mathtt{I}^{=k} \;\big|\; \mathtt{C}^{\le k} \;\big|\; \mathtt{F}\; \phi
\end{aligned}$$

where $\mathsf{a} \in AP$ is an atomic proposition, $\chi \in CC(\mathscr{X})$ is a clock constraint, $\bowtie \in \{\leq,<,\geq,>\}$, $p \in \mathbb{Q} \cap [0,1]$, $q \in \mathbb{Q}_{\geq 0}$, $\mathbf{r}$ is a reward structure and $k \in \mathbb{N}$.

This logic extends propositional logic with a *probabilistic operator* ($\mathtt{P}$) and a *reward operator* ($\mathtt{R}$). Informally, a property of the form $\mathtt{P}_{\bowtie p}[\psi]$ states that the probability of *path formula* ψ being true always satisfies the bound $\bowtie p$. A property of the form $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ means that the *expected* value of *reward function* ρ on *reward structure* $\mathbf{r}$ meets the bound $\bowtie q$.

Formulae in the logic are always *state formulae*, i.e., those formed by the production ϕ in the grammar above. These are evaluated over the states of a PTA P (or, more precisely, over the states of the TPS $[\![\mathsf{P}]\!]$ representing its semantics). For state s and formula ϕ, we write $s \models \phi$ to denote that ϕ is satisfied in s. The syntax also includes *path formulae* (ψ) and *reward operators* (ρ), which appear only as subformulae of the $\mathtt{P}$ and $\mathtt{R}$ operators.

We include two types of path formulae: *time-bounded until* ($\phi_1\ \mathtt{U}^{\leq k}\ \phi_2$) and (unbounded) *until* ($\phi_1\ \mathtt{U}\ \phi_2$). Formula $\phi_1\ \mathtt{U}\ \phi_2$ means that a state satisfying ϕ_2 is eventually reached and that, at every time-instant prior to that, ϕ_1 is satisfied. The time-bounded variant has the same meaning, with the additional constraint that the occurrence of ϕ_2 must occur within time k. We can derive several useful operators, such as $\mathtt{F}\ \phi \equiv \mathtt{true}\ \mathtt{U}\ \phi$, which means that ϕ is eventually satisfied, and $\mathtt{F}^{\leq k}\ \phi \equiv \mathtt{true}\ \mathtt{U}^{\leq k}\ \phi$, which means that ϕ is satisfied within time k. We also have $\mathtt{G}\ \phi \equiv \neg(\mathtt{F}\ \neg\phi)$, which means that ϕ is always satisfied, and $\mathtt{G}^{\leq k}\ \phi \equiv \neg(\mathtt{F}^{\leq k}\ \neg\phi)$ which means that ϕ is continuously satisfied for time k. Although the $\mathtt{G}$ and $\mathtt{G}^{\leq k}$ operators cannot be derived from the basic syntax of the logic since there is no negation of path formulae, it can be shown that $\mathtt{P}_{\leq p}[\neg\psi] \equiv \mathtt{P}_{\geq 1-p}[\psi]$, $\mathtt{P}_{<p}[\neg\psi] \equiv \mathtt{P}_{>1-p}[\psi]$, $\mathtt{P}_{\geq p}[\neg\psi] \equiv \mathtt{P}_{\leq 1-p}[\psi]$ and $\mathtt{P}_{>p}[\neg\psi] \equiv \mathtt{P}_{<1-p}[\psi]$. PCTL-style logics often include a *next* ($\mathtt{X}$) operator but this is of less use in timed models and omitted.

The reward operator $\mathtt{I}^{=k}$ refers to the reward of the current state at time instant k, $\mathtt{C}^{\leq k}$ to the total reward accumulated up until time point k, and $\mathtt{F}\ \phi$ to the total reward accumulated until a state satisfying ϕ is reached. Formally, we define the semantics of the logic as follows.

Definition 6 (Semantics) Let P be a PTA, $[\![\mathsf{P}]\!] = (S, \bar{s}, Act, \mathbb{T}, Steps_{\mathsf{P}}, lab)$ be its semantics, and let r denote the reward structure over $[\![\mathsf{P}]\!]$ corresponding to a reward structure $\mathbf{r}$ over P. For state $s{=}(l,v) \in S$, the satisfaction relation $\models$ is defined inductively by:

$$\begin{array}{lcl}
s \models \mathtt{true} & & \text{always} \\
s \models \mathsf{a} & \iff & \mathsf{a} \in lab(s) \\
s \models \chi & \iff & v \models \chi \\
s \models \phi_1 \wedge \phi_2 & \iff & s \models \phi_1 \wedge s \models \phi_2 \\
s \models \neg\phi & \iff & s \not\models \phi \\
s \models \mathtt{P}_{\bowtie p}[\psi] & \iff & Pr^{\sigma}_{[\![\mathsf{P}]\!],s}(\{\omega \in Path^{\sigma}_{[\![\mathsf{P}]\!],s} \mid \omega \models \psi\}) \bowtie p \text{ for all } \sigma \in Adv_{[\![\mathsf{P}]\!]} \\
s \models \mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho] & \iff & \mathbb{E}^{\sigma}_{[\![\mathsf{P}]\!],s}(rew(r,\rho)) \bowtie q \text{ for all } \sigma \in Adv_{[\![\mathsf{P}]\!]}
\end{array}$$

where, for any infinite path ω of $[\![\mathsf{P}]\!]$:

$$\begin{array}{lcl}
\omega \models \phi_1\ \mathtt{U}^{\leq k}\ \phi_2 & \Leftrightarrow & \text{there exists a position } (i,t) \text{ of } \omega \text{ such that } \omega(i){+}t \models \phi_2 \text{ and } dur_{\omega}(i){+}t{\leq}k \\
& & \text{and } \omega(j){+}t' \models \phi_1 {\vee} \phi_2 \text{ for all positions } (j,t'){\prec}(i,t) \text{ of } \omega \\
\omega \models \phi_1\ \mathtt{U}\ \phi_2 & \Leftrightarrow & \text{there exists a position } (i,t) \text{ of } \omega \text{ such that } \omega(i){+}t \models \phi_2 \\
& & \text{and } \omega(j){+}t' \models \phi_1 {\vee} \phi_2 \text{ for all positions } (j,t'){\prec}(i,t) \text{ of } \omega
\end{array}$$

and, for reward structure $r{=}(r_S, r_{Act})$ over $[\![\mathsf{P}]\!]$, the random variable $rew(r,\rho)$ over infinite paths of $[\![\mathsf{P}]\!]$ is defined as follows:

$$
\begin{aligned}
rew(r, \mathtt{I}^{=k})(\omega) &= r_S(\omega(j_k)) \\
rew(r, \mathtt{C}^{\leq k})(\omega) &= \textstyle\sum_{i=0}^{j_k-1} r(\omega,i) + (k - dur_\omega(j_k)) \cdot r_S(\omega(j_k)) \\
rew(r, \mathtt{F}\ \phi)(\omega) &= \begin{cases} \sum_{i=0}^{j_\phi-1} r(\omega,i) + t_\phi{\cdot}r_S(\omega(j_\phi)) & \text{if } (j_\phi,t_\phi) \text{ exists} \\ \infty & \text{otherwise} \end{cases}
\end{aligned}
$$

where $j_0 = 0$, $j_k = \max\{i \mid dur_\omega(i){<}k\}$ for $k{>}0$ and, when it exists, (j_ϕ, t_ϕ) is the minimum position under the ordering $\prec$ such that $\omega(j_\phi){+}t_\phi \models \phi$.

In addition to the basic syntax of Definition 6, we allow, in the style of the PRISM model checker, *quantitative* (numerical) queries yielding the minimum or maximum probability or expected reward value from a state s. We use $\mathtt{P}_{\mathtt{min}=?}[\psi]$, $\mathtt{P}_{\mathtt{max}=?}[\psi]$, $\mathtt{R}^{\mathbf{r}}_{\mathtt{min}=?}[\rho]$ and $\mathtt{R}^{\mathbf{r}}_{\mathtt{max}=?}[\rho]$, which give, respectively:

$$
\begin{aligned}
Pr^{\min}_{\mathsf{P},s}(\psi) &\overset{\text{def}}{=} \inf_{\sigma\in Adv_{[\![\mathsf{P}]\!]}} Pr^\sigma_{[\![\mathsf{P}]\!],s}(\{\omega \in Path^\sigma_{[\![\mathsf{P}]\!],s} \mid \omega \models \psi\}) \\
Pr^{\max}_{\mathsf{P},s}(\psi) &\overset{\text{def}}{=} \sup_{\sigma\in Adv_{[\![\mathsf{P}]\!]}} Pr^\sigma_{[\![\mathsf{P}]\!],s}(\{\omega \in Path^\sigma_{[\![\mathsf{P}]\!],s} \mid \omega \models \psi\}) \\
\mathbb{E}^{\mathbf{r},\min}_{\mathsf{P},s}(\rho) &\overset{\text{def}}{=} \inf_{\sigma\in Adv_{[\![\mathsf{P}]\!]}} \mathbb{E}^\sigma_{[\![\mathsf{P}]\!],s}(rew(r,\rho)) \\
\mathbb{E}^{\mathbf{r},\max}_{\mathsf{P},s}(\rho) &\overset{\text{def}}{=} \sup_{\sigma\in Adv_{[\![\mathsf{P}]\!]}} \mathbb{E}^\sigma_{[\![\mathsf{P}]\!],s}(rew(r,\rho)).
\end{aligned}
$$

Some typical examples of PTA properties, specified in this logic are:

- $\mathtt{P}_{\geq 0.8}[\mathtt{F}^{\leq k}\ \mathsf{ack}_n]$ – “the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8”;
- $\mathsf{trigger} \rightarrow \mathtt{P}_{<0.0001}[\mathtt{G}^{\leq 20}\ \neg\mathsf{deploy}]$ – “the probability of the airbag failing to deploy within 20 milliseconds of being triggered is strictly less than 0.0001”,
- $\mathtt{P}_{\mathtt{max}=?}[\neg\mathsf{sent}\ \mathtt{U}\ \mathsf{fail}]$ – “what is the maximum probability of a failure occurring before message transmission is complete?”;
- $\mathtt{R}^{\mathbf{time}}_{\mathtt{max}=?}[\mathtt{F}\ \mathsf{end}]$ – “what is the maximum expected time for the protocol to terminate?”;
- $\mathtt{R}^{\mathbf{pwr}}_{<q}[\mathtt{C}^{\leq 60}]$ – “the expected energy consumption during the first 60 seconds is $< q$”.

Property reductions. We now describe how model checking for several of the operators included in our logic can be reduced to checking satisfaction of a simpler formula on a modified PTA. Consider first a time-bounded until property $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}^{\leq k}\ \phi_2]$ on a PTA P. If we augment P with an additional clock z, then it follows that a state (l,v) of P satisfies the formula $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}^{\leq k}\ \phi_2]$ if and only if the state (l,v') of the augmented PTA where $v'(x){=}v(x)$ for all clocks x of P and $v'(z){=}0$ satisfies $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}\ (\phi_2 \wedge (z{\leq}k))]$ (see [56]). Second, given an until property $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}\ \phi_2]$, if we modify the PTA such that, upon reaching a state not satisfying ϕ_1, only a transition to a sink state is possible, then a state of the PTA satisfies $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}\ \phi_2]$ if and only if a state of the modified PTA satisifies $\mathtt{P}_{\bowtie p}[\mathtt{F}\ \phi_2]$.

Next, we show how to reduce checking a property of the form $\mathtt{P}_{\geq p}[\mathtt{F}^{\leq k}\ \phi]$, to a formula of the form $\mathtt{P}_{\leq 1-p}[\mathtt{F}\ \mathsf{a}_{\mathsf{exc}}]$ (the former requires computation of minimum probabilities, whereas the the latter needs maximum probabilities, which are sometimes easier to compute). The reduction is correct only for states satisfying $\mathtt{P}_{\geq 1}[\mathtt{F}\ \phi]$, i.e. ϕ is always reachable with probability 1. We augment P with an extra clock z and then modify it such that states satisfying ϕ are forced to make a transition to a sink-location and, in all other states, we add a transition to a different, sink-location $\mathsf{exceeded}$, enabled when $z{>}k$. Then $\mathtt{P}_{\geq p}[\mathtt{F}^{\leq k}\ \phi]$ is

	Region graph	Boundary region graph	Digital clocks (for closed PTAs)	Backwards reachability	Stochastic games
Single $\mathtt{P}_{\bowtie p}[\psi]$ operator	✓	✓	✓	✓	✓
Single $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ operator	×	✓	✓	Open	Open
Logic without $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$	✓	✓	×	✓	Open
Full logic	×	×	×	Open	Open

Table 1 Summary of PTA model checking techniques and their applicability.

true in a state of the original model if and only if the corresponding state of the modified PTA with z=0 satisfies $\mathtt{P}_{\leq 1-p}[\mathtt{F}\ \mathsf{a}_{\mathsf{exc}}]$, where $\mathsf{a}_{\mathsf{exc}}$ is true only in location $\mathsf{exceeded}$ (see [55]).

Finally, we show how to reduce properties of the form $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{C}^{\leq k}]$ or $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{I}^{=k}]$ to $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{F}\ \phi]$. In both cases, we add an extra clock z to P. For $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{C}^{\leq k}]$, it suffices to check $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{F}\ (z{=}k)]$ on the augmented PTA. For $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{I}^{=k}]$, we add the conjunct $z{\leq}k$ to all invariants and a transition to a new sink location done (labelled $\mathsf{a}_{\mathsf{done}}$) with enabling condition $z{=}k$ to all locations, while changing the enabling conditions of all other transitions so that they are not enabled when $z{=}k$. It then suffices to check $\mathtt{R}^{r}_{\bowtie q}[\mathtt{F}\ \mathsf{a}_{\mathsf{done}}]$, where the only non-zero rewards are action rewards on the new transitions, set to the location reward of the source location.

More expressive logics. The next section of this paper will discuss techniques for model checking the properties expressible in the logic given above. A variety of more expressive logics have also been considered for PTAs. PTCTL [54] is a probabilistic extension of the timed temporal logic TCTL. In particular, it includes a *freeze quantifier* (or *reset quantifier*) $z.\phi$, which introduces a formula clock z, reset to zero, that can be referred to in the subformula ϕ. Although not considered further in this paper, PTCTL can be model checked using the region graph construction and backwards reachability method discussed in Section 5. Formulae of the logics LTL and PCTL*, originally proposed for discrete-time probabilistic systems, can be verified on PTAs using a Rabin automaton product construction and the model checking algorithm of [64].

5 Model Checking for PTAs

We now consider the problem of model checking a PTA P with respect to a property ϕ of the logic presented in Section 4, i.e., determining $Sat(\phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \phi\}$, where S is the set of states of $[\![\mathsf{P}]\!]$. We will survey the various PTA model checking techniques that have been proposed in the literature, which support different fragments of the logic. We will cover:

- the region graph construction [54];
- the boundary region graph [43];
- the digital clocks method [52];
- backwards reachability [56];
- abstraction refinement with stochastic games [50].

The first two approaches, which are based on the concept of the *region graph* [2,4], are used primarily to establish the decidability and complexity of model checking, rather than for practical implementations. The others provide efficient methods for model checking particular fragments of the logic. Table 1 provides a summary of the methods and their applicability. We omit the *forwards reachability* algorithm described in [54] since it only computes bounds on probabilities of system behaviour, rather than the exact values.

Unless otherwise stated, we also assume that P has no timelocks and is structurally divergent (structurally non-Zeno). In Section 5.1, we *do* explain how to treat PTAs that are not structurally divergent. Similar adaptations can also be applied to the other methods that we discuss in this section.

5.1 The Region Graph

We first discuss the *region graph* construction for a PTA [54], which is based on the classic construction for timed automata [2,4]. This approach provides a way to model check the fragment of the logic from Section 4 that excludes $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ formulae (in the next section, we will relax this restriction). The region graph of a PTA P and formula ϕ takes the form of a finite-state MDP whose states are *regions* of the form (l,α), where l is a location and α is an equivalence class of clock valuations according to the equivalence defined below. Let c be the maximal constant to which any clock is compared in the clock constraints of P and ϕ. Then clock valuations v and v' are equivalent if and only if they satisfy the following conditions:

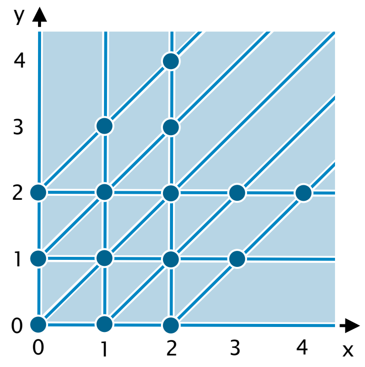


Fig. 2 Clock equivalence classes for two clocks x and y ($c=2$)

- for any $x \in \mathscr{X}$, either $v(x)>c$ and $v'(x)>c$, or $v(x)$ and $v'(x)$ agree on their integer parts;
- for any $x,x' \in \mathscr{X}$, either $v(x)-v(x') > c$ and $v'(x)-v'(x') > c$, or $v(x)-v(x')$ and $v'(x)-v'(x')$ agree on their integer parts,

where two values $q,q' \in \mathbb{R}_{\geq 0}$ agree on their integer parts when $\lfloor q \rfloor = \lfloor q' \rfloor$ and $\lfloor q \rfloor - q = 0$ if and only if $\lfloor q' \rfloor - q' = 0$. Figure 2 shows the set of possible equivalence classes of clock valuations for clocks x,y when $c=2$. Note that equivalent clock valuations satisfy the same clock constraints of the PTA. The set of regions needed for the region graph, denoted R, is the set of regions (l,α) such that there exists $v \in \alpha$ such that $v \models inv(l)$. The size of R is bounded by $|L| \cdot (2c+2)^{(|\mathscr{X}|+1)^2}$ (see [23]).

A region $(l,\alpha) \in R$ may have a *time-successor*, defined as follows. If $v{+}t \in \alpha$ for all $v \in \alpha$ and $t \in \mathbb{R}_{\geq 0}$, then the time-successor of (l,α) is (l,α) itself. Otherwise, there exists the unique region $(l,\beta) \neq (l,\alpha)$ for which there are $v \in \alpha$ and $t \in \mathbb{R}_{\geq 0}$ such that $v{+}t \in \beta$ and $v{+}t' \in \alpha \cup \beta$ for all $0 \leq t' \leq t$. If additionally we have $v{+}t' \models inv(l)$ for all $0 \leq t' \leq t$, then (l,β) is the time-successor of (l,α), otherwise (l,α) has no time-successor.

The region graph for P and ϕ is the finite-state MDP $\mathsf{Reg}[\mathsf{P},\phi] = (R,(\bar{l},[\mathbf{0}]),Act \cup \{\tau\},Steps,lab)$, where for each $(l,\alpha) \in R$ and $a \in Act \cup \{\tau\}$, we have $Steps((l,\alpha),a) = \lambda$ if and only if either:

Time transitions. $a=\tau$, $\lambda = \mu_{(l,\beta)}$ and (l,β) is the time-successor of (l,α);
Action transitions. $a \in Act$, there is a $v \in \alpha$ with $v \models enab(l,a)$ and, for each $(l',\beta) \in R$:

$$\lambda(l',\beta) = \sum\{\!|\, prob(l,a)(X,l') \mid X \in 2^{\mathscr{X}} \wedge \beta = \alpha[X{:=}0] \,|\!\}\,;$$

and $lab(l,\alpha) = \mathscr{L}(l)$ for all $(l,\alpha) \in R$.

We now consider how $\mathsf{Reg}[\mathsf{P},\phi]$ can be used to verify P against ϕ (recall that we consider the fragment of the logic without the $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\cdot]$ operator). For simplicity, we first assume that P is structurally divergent. Model checking proceeds in standard fashion (for a branching-time logic), recursing over subformulae ϕ' of ϕ and computing $Sat(\phi')$. Identifying states that satisfy atomic propositions, clock constraints or Boolean connectives is straightforward and time-bounded properties are dealt with using the reduction given in Section 4. Hence, we focus on formulae of the form $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}\ \phi_2]$. If the state sets satisfying ϕ_1 and ϕ_2 have already been computed and the regions of $\mathsf{Reg}[\mathsf{P},\phi]$ corresponding to these sets are labelled a_1 and a_2, respectively, then the states in $[\![\mathsf{P}]\!]$ satisfying $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}\ \phi_2]$ are simply those in the regions corresponding to states in the MDP $\mathsf{Reg}[\mathsf{P},\phi]$ that satisfy $\mathtt{P}_{\bowtie p}[\mathsf{a}_1\ \mathtt{U}\ \mathsf{a}_2]$.

Thus, the region graph yields an algorithm for model checking structurally divergent PTAs against properties of the logic without the $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\cdot]$ operator. The algorithm runs in exponential time, because verifying properties of the form $\mathtt{P}_{\bowtie p}[\mathsf{a}_1\ \mathtt{U}\ \mathsf{a}_2]$ on MDPs can be done in polynomial time [20,10], and the size of the region graph $\mathsf{Reg}[\mathsf{P},\phi]$ is exponential in the size of P (the size of P is the sum of the number of locations and clocks, the size of the binary encoding of the constants used in invariant and enabling conditions, and the size of the encoding of its transition probabilities, which are expressed as a ratio between two natural numbers, each in binary). The problem is EXPTIME-complete, where an EXPTIME lower bound can be obtained even for the restricted case in which the PTA has only two clocks (for PTAs with one clock, the model-checking problem for certain restricted classes of properties, such as PCTL properties or time-bounded until properties with probability thresholds 0 or 1 only, is PTIME-complete) [44].

We now consider the case where P is not structurally divergent. For $\mathtt{P}_{\leq p}[\phi_1\ \mathtt{U}\ \phi_2]$ or $\mathtt{P}_{<p}[\phi_1\ \mathtt{U}\ \phi_2]$ (i.e., where maximum probabilities are needed), the method given above can be applied without further modification [63]. The case of $\mathtt{P}_{\geq p}[\phi_1\ \mathtt{U}\ \phi_2]$ or $\mathtt{P}_{>p}[\phi_1\ \mathtt{U}\ \phi_2]$ (which needs minimum probabilities) is more involved. Intuitively, adversaries which can avoid reaching states satisfying ϕ_2 only because they perform some non-divergent behaviour can result in the minimum probability of satisfying $\phi_1\ \mathtt{U}\ \phi_2$ computed over all adversaries being lower than the probability computed over divergent adversaries only. An algorithm that resolves this problem, based on the computation of the maximum probability of satisfying the dual path property $\neg(\phi_1\ \mathtt{U}\ \phi_2)$, is presented in [63]. The model-checking problem remains EXPTIME-complete in this case. Details on model-checking algorithms for the strictly divergent adversaries described in Section 3 are also given in [63].

Finally, we note that region equivalence is an example of a time-abstracting bisimulation, a relation which combines time-abstracting bisimulation [3,59] and (probabilistic) bisimulation [58,62]. As with region equivalence, time-abstracting bisimilar PTA states satisfy the same formulae (without $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ formulae). An algorithm for computing a time-abstracting equivalence relation of a PTA, which may be coarser than region equivalence, has been presented in [28]. This approach is described as being applicable to formulae of the logic without $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ or $\mathtt{P}_{\bowtie p}[\phi_1\ \mathtt{U}^{\leq k}\ \phi_2]$; however, as explained in Section 4, time-bounded until properties can be reduced to unbounded until properties on a modified PTA.

5.2 The Boundary Region Graph

The region graph construction presented above is not sufficient for verifying reward properties. In particular, we note that, for a particular region (l,α), the values $\mathbb{E}^{\mathbf{r},\min}_{\mathsf{P},s}(\rho)$ and $\mathbb{E}^{\mathbf{r},\max}_{\mathsf{P},s}(\rho)$ will generally not be uniform on states $s \in (l,\alpha)$. We now briefly describe a generalisation of the region graph, called the *boundary region graph* [43], which is a finite MDP equipped with a reward structure on which we can decide whether a particular state s of a PTA satisfies an $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{F}\ \phi]$ property, under the restriction that there is no nesting of the $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\cdot]$ operator and the bound $\bowtie$ is non-strict. For $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{C}^{\leq k}]$ and $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{I}^{=k}]$ properties, the reduction given in Section 4 can be used under the same restrictions.

The boundary region graph construction is an extension of the *corner point abstraction* for timed automata [24]. The underlying idea is that optimal behaviour (resulting in minimum or maximum expected rewards) corresponds to the case in which edge transitions of the PTA are taken either in a clock equivalence class in which the value of at least one clock equals an integer, or close to a boundary of a clock equivalence class in which the fractional parts of all clocks are positive: in this case, for computing the accumulation of rewards over time, it is necessary to distinguish which of the class's boundaries is considered. Furthermore, the exact position on the clock equivalence boundaries is determined by the values of the clocks in the state s, and must also be encoded in the boundary region graph. The reward structure for the boundary region graph is derived directly from that of the PTA. From the boundary region graph (a finite-state MDP), we can compute the minimum or maximum expected accumulated reward to a target set (see, e..g [32]). We can then obtain either $\mathbb{E}^{\mathbf{r},\min}_{\mathsf{P},s}(\mathtt{F}\ \phi)$ or $\mathbb{E}^{\mathbf{r},\max}_{\mathsf{P},s}(\mathtt{F}\ \phi)$ and hence decide whether s satisfies $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{F}\ \phi]$.

5.3 Digital Clocks

Region graph-based approaches are not usually practically applicable since the region graphs are generally of a prohibitive size. Thus, various other PTA model checking approaches have been developed. We first describe the *digital clocks* method [52], which restricts the standard continuous-time semantics of a PTA so that only time transitions of duration 1 occur. This means that clocks take only integer, rather than real, values. Using this fact, and knowing there is a maximal constant c_x to which each clock x is compared in PTA P and property ϕ, we can again build and analyse a finite-state MDP. This approach builds on the use of digital clocks for (non-probabilistic) timed automata verification [39,8,19].

The digital clocks method is applicable to properties of the form $\mathtt{P}_{\bowtie p}[\psi]$ and $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ *without* nesting of further $\mathtt{P}_{\bowtie p}[\cdot]$ and $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\cdot]$ operators within the subformulae ψ and ρ (see [52] for an explanation as to this limitation). It can only be used to determine satisfaction in states where all clocks take natural-numbered values, and we will restrict our attention to checking satisfaction in the initial state. The correctness of the digital clocks method also relies on the assumption that P and ϕ are *closed*, meaning that all clock constraints of the form $x{\le}d$ or $d{\le}x$ are contained within an even number of negations. Furthermore, all invariant and enabling conditions of P are assumed to be *diagonal-free*, meaning that constraints of the form $x{+}c \le y{+}d$ are not permitted. Any PTA can be transformed into one containing only diagonal-free constraints by applying the construction of [15] (the construction is presented for timed automata, and requires minor modifications for PTAs).

For a digital clock valuation $v \in \mathbb{N}^{\mathscr{X}}$, let $v{\oplus}1$ be the clock valuation such that $(v{\oplus}1)(x) = \min\{v(x){+}1, c_x{+}1\}$ for all $x \in \mathscr{X}$. The *digital clock semantics* of P and ϕ is defined as for the standard semantics, except that the rule for time transitions restricts durations to 1, and each clock x can increase to at most $c_x{+}1$. Formally, the digital clock semantics of a closed PTA P is defined as the finite-state MDP $\mathsf{Dgt}(\mathsf{P},\phi) = (S, (\bar{l},\mathbf{0}), Act \cup \{1\}, Steps, lab)$ where:

- $S = \{(l,v) \in L{\times}\mathbb{N}^{\mathscr{X}} \mid v \models inv(l) \wedge (\forall x \in \mathscr{X}.\, v(x) \le c_x{+}1)\}$;
- $Steps((l,v),a) = \lambda$ if and only if either:
  Time transitions. $a{=}1$, $v{\oplus}1 \models inv(l)$ and $\lambda = \mu_{(l,v\oplus 1)}$;
  Action transitions. $a \in Act$, $v \models enab(l,a)$, and, for any $(l',v') \in S$:
$$\lambda(l',v') = \textstyle\sum\{\!|\, prob(l,a)(X,l') \mid X \in 2^{\mathscr{X}} \wedge v' = v[X{:=}0] \,|\!\}\,;$$
- $lab(l,v) = \mathscr{L}(l)$ for each $(l,v) \in S$.

The number of states of the digital clock semantics of P is bounded by $|L| \cdot \prod_{x \in \mathscr{X}} (c_x+1)$.

Model checking for formulae of the form $\mathtt{P}_{\bowtie p}[\psi]$ and $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ without nesting can then be carried out directly on the finite MDP from the digital clock semantics. For $\mathtt{P}_{\bowtie p}[\psi]$ formulae, we proceed as in the case of the region graph in Section 5.1. For a formula $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\mathtt{F}\ \phi]$ formulae and PTA reward structure $\mathbf{r}{=}(\mathbf{r}_{Act}, \mathbf{r}_L)$, we proceed as follows. We construct the reward structure $r{=}(r_S, r_{Act})$ where $r_S(l,v){=}0$, $r_{Act}((l,v),1){=}\mathbf{r}_L(l)$ and $r_{Act}((l,v),a){=}\mathbf{r}_{Act}(l,a)$ for all $(l,v) \in S$ and $a \in Act$. We then use standard algorithms for MDPs to compute the minimum or maximum expected reward to reach the set of states $Sat(\phi)$. The cases for $\rho{=}\mathtt{C}^{\leq k}$ and $\rho{=}\mathtt{I}^{=k}$ use the reductions presented in Section 4.

5.4 Backwards Reachability

The next method we consider is *backwards reachability* [56], which provides model checking for properties without $\mathtt{R}^{\mathbf{r}}_{\bowtie q}[\rho]$ operators. This is based on the repeated application of a predecessor operation that, given a set of states S', returns the set of states that can reach S' by performing an action and then letting time pass. Sets of states are represented by *symbolic states*, pairs $\mathsf{z}{=}(l,\zeta)$ comprising a location l and a clock constraint ζ over $\mathscr{X}$, representing the set of states $\{(l,v) \,|\, v \models \zeta\}$.

This approach is an adaptation of the algorithm in [40] for model checking timed automata. Whereas the latter just requires iteration of a generic predecessor operation, for PTAs it is necessary to retain information about the probabilities of the PTA edges used along paths. First, the predecessor operation is parameterised by actions and edges of the PTA. Then, as the predecessor iteration proceeds, a graph is constructed, where the nodes are the generated symbolic states, and an edge is added from symbolic state z to symbolic state z' if z was generated from z' by a predecessor operation. The edge (z,z') is labelled by the corresponding action and PTA edge. The symbolic states generated in this manner do *not* form a partition of the state space of the PTA, unlike the region graph or time-abstracting bisimulation approaches described in Section 5.1.

After the iteration of predecessor operations terminates (which is guaranteed because a symbolic state corresponds to a union of regions), the obtained graph can then be used as a basis for the construction of a finite MDP. To build the probabilistic transition function, information has to be combined from different symbolic states in order to obtain the exact combination of PTA edges (corresponding to a particular action) available in states of the PTA. This is done by computing the conjunctions of symbolic states which have at least one outgoing edge labelled with the same action, and adding the corresponding graph edges to the newly generated symbolic states. For more information, see [56].

The approach outlined above only applies to $\mathtt{P}_{\bowtie p}[\psi]$ operators where $\bowtie \in \{\leq, <\}$, i.e. it computes only maximum probabilities for the PTA. For the case $\bowtie \in \{\geq, >\}$, which needs minimum probabilities, the method needs to be adapted [56]. Like the reduction described in Section 5.1 for the region graph approach on non-structurally-divergent PTAs, the solution taken in [56] works by considering the dual formula.

5.5 Abstraction Refinement using Stochastic Games

The final model checking technique we discuss for PTAs is *abstraction refinement* using *stochastic games*, which can verify $\mathtt{P}_{\bowtie p}[\psi]$ properties via computation of reachability probabilities. This approach uses the game-based notion of abstraction put forward for MDPs in [49] and the corresponding refinement techniques proposed in [46]. The method can be applied to PTAs containing diagonal-free constraints only, because one of its procedures is a (forwards) reachability exploration and, by the results of Bouyer [22], such a procedure is only correct if the considered PTA is diagonal-free.

The idea of [49] is to build an abstraction of a large or infinite-state MDP based on a finite partition of its state space. Abstractions take the form of stochastic two-player games, a generalisation of MDPs in which there are two distinct types of nondeterminism, each controlled by a separate player. In this case, player 1 controls nondeterminism introduced by the abstraction, and player 2 controls nondeterminism from the original MDP. An analysis of the *optimal probabilities* in the stochastic game (e.g. the maximum probability that player 1 can achieve for some objective, assuming that player 2 aims to minimise it) yields lower and upper bounds on reachability probabilities for the original MDP.

The abstraction-refinement framework of [46] provides a way to automatically construct stochastic game abstractions, by iteratively refining abstractions until the difference between the lower and upper bounds produced is below some desired level of precision ε. In [50], this technique is adapted to an algorithm to compute exact (minimum or maximum) reachability probabilities for PTAs. First, a *reachability graph* is constructed, based on a successor operation that returns the set of states that can be reached by performing an action and then letting time pass. This works in a similar style to the classic approach to verifying (non-probabilistic) timed automata. From this, a stochastic game abstraction is created, over *symbolic states* of the same form as in Section 5.4. The abstraction is repeatedly analysed and refined until the exact required probabilities are obtained (i.e. ε=0). The iterative refinement process is guaranteed to terminate thanks to the fact that there is an underlying finite time-abstracting bisimulation quotient, namely the region graph.

5.6 A Comparison of the Methods

We briefly summarise the relative merits of the three practical approaches to PTA model checking discussed above. In terms of applicability, digital clocks is currently the only method for computing expected rewards, but handles only closed (non-strict) and diagonal-free clock constraints. The other limitations of the remaining two methods are that abstraction refinement only applies to PTAs with diagonal-free clock constraints, and backwards reachability requires (non-trivial) adaptation to compute minimum probabilities.

In terms of efficiency and scalability, the digital clocks approach has proved to work well in practice, but performance suffers for large numbers of clocks or when large constants appear in the clock constraints; in these cases, the other two methods described have been shown to work better. The techniques based on parameter synthesis of [7] can be used to reduce the size of the constants of clock constraints for subclasses of PTAs and for probabilistic properties without time bounds. Another good approach to improving performance is the use of symbolic (binary decision diagram based) implementations. The original implementation of backward reachability [56] showed that it yielded relatively small MDPs, but that the algorithm can be expensive to implement. Experimental results for abstraction refinement [50] later demonstrated better performance in all cases. However, subsequent optimisations presented for backwards reachability [18] have led to much better performance, improving on abstraction refinement in many cases.

5.7 Implementations and Tool Support

Thanks to increasing interest in the verification of probabilistic real-time systems, a variety of related software tools have recently been developed. The probabilistic model checker PRISM [51], for example, which provides verification of Markov chains and MDPs, now also supports PTAs, via the digital clocks, backwards reachability and abstraction-refinement methods. A second tool is `mcpta` [38], which applies the digital clocks method to translate a subset of the modelling language Modest [21] directly into the PRISM modelling language.

Fortuna [18] is a tool that focuses on PTAs augmented with prices (called rewards in this paper). In particular, it implements the semi-algorithm of [17] for computing the maximum probability of reaching a target while accumulating a reward below a given threshold (this problem is shown to be undecidable in [16]). Since this algorithm generalises the backwards reachability method of [56], computation of (maximum) reachability probabilities is also supported. Several optimisations of the basic algorithm are also implemented. Finally, UPPAAL PRO [68] is an extension of the popular timed automaton verifier UPPAAL. It computes the maximum probability of reaching a set of target states of a PTA, by progressively partitioning the state space, constructing and solving a finite MDP at each step.

6 Case Studies

PTAs have been used for the modelling and analysis of a wide variety of systems, including communication protocols [55,33], aviation security systems [34], streaming download protocols [69] and service level agreements [48]. In this section, we give an illustration of the PTA model checking techniques described in this paper by presenting two case studies: a non-repudiation protocol and a task-graph scheduling problem.

6.1 Markowitch & Roggeman's Non-Repudiation Protocol

This case study analyses Markowitch & Roggeman's non-repudiation protocol for information transfer [61]. Our models extend those presented previously in [57]. One party, the *originator*, sends information to a second party, the *recipient*. *Repudiation* is defined as the denial of either party of having participated in all or part of the information transfer. For example, in electronic commerce, if the information represents the transfer of a service, then *non-repudiation* ensures the client (the recipient) cannot deny receiving the service as a reason for non-payment.

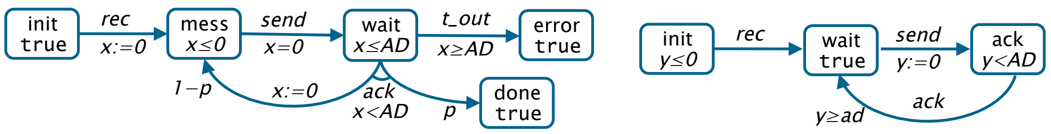
The protocol of Markowitch & Roggeman is probabilistic and does not require a trusted third party. It achieves the following non-repudiation properties:

- "*ε-fair*": at each step of the protocol run, either both parties receive their expected items, or the probability that a cheating party gains any valuable information about its expected items and the other party gains nothing is at most ε;
- "*time-bounded*": if at least one party behaves correctly, then the protocol will complete within a finite amount of time (with probability 1);
- "*viable*": if both parties behave correctly and finish the protocol, then they both receive their expected items at the end of the protocol (with probability 1).

The steps of the protocol are outlined in Figure 3. First, to prevent replay attacks, the recipient R selects a date D which it sends, along with a request, to the originator O. Next,

The recipient chooses the date D	
1.	$R \to O : \mathrm{Sign}_R(request, R, O, D)$
The originator checks D, chooses n and computes functions $f_1, \dots, f_n$	
2.	$O \to R : \mathrm{Sign}_O(f_n(message), O, R, D)$
3.	$R \to O : \mathrm{Sign}_R(ack_1)$
...	...
2n.	$O \to R : \mathrm{Sign}_O(f_1(message), O, R, D)$
$2n{+}1$.	$R \to O : \mathrm{Sign}_R(ack_n)$

Fig. 3 The steps of Markowitch & Roggeman's non-repudiation protocol



(a) Originator (b) Honest recipient

Fig. 4 PTAs used to model the non-repudiation protocol

O randomly selects an integer n representing the number of steps of the protocol (which is never revealed to R during the execution) and computes functions f_i such that their composition satisfies the following:

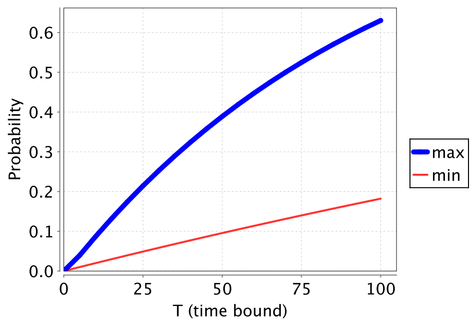
$$f_n(message) \circ (f_{n-1}(message) \circ (\cdots \circ (f_2(message) \circ f_1(message)) \cdots)) = message$$

The composition operator $\circ$ needs to be non-commutative to ensure that the recipient cannot start to compute the message until the final message $f_1(message)$ has been received.

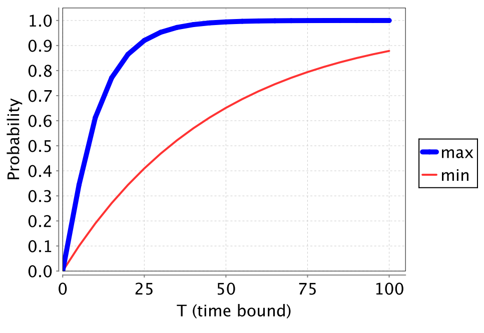
To prevent the recipient from gaining an advantage, if the originator does not receive an acknowledgement within a certain time bound (denoted AD), the protocol is stopped and and the originator states that the recipient is trying to cheat. The time bound is chosen such that it is greater than the time it takes for a recipient to send a reply, but is not sufficient for the recipient to be able to compute the composition $f_n(message) \circ (\cdots \circ f_i(message) \cdots)$ for any $i<n$, i.e., the recipient does not have time to check if it has received the complete message before sending an acknowledgement.

We consider two different versions of the protocol. In the first, both the originator and recipient act *honestly*, while in the second the recipient can act *maliciously* (i.e., stop early by not returning an acknowledgement). For each, we assume that the choice of n is made by the recipient according to a geometric distribution with parameter p and the minimum time for the recipient to send an acknowledgement is ad. For the malicious version, we consider two variants. The first corresponds to the one described in [61], where the only malicious behaviour corresponds to stopping early. In the second, we introduce a more powerful malicious recipient, which has access to a method that takes time less than AD and with probability $\frac{1}{4}$ correctly computes the composition while with probability $\frac{3}{4}$ fails to compute the composition.

Each model is the parallel composition of two PTAs, one representing the originator and one the recipient. These synchronise on the actions *rec*, *send* and *ack*, corresponding to the recipient initiating the transaction, the originator sending messages to the recipient and the recipient sending acknowledgements back. The PTAs for the originator and (honest) recipient are shown in Figure 4. The probabilistic choice in the originator correctly models selection of n according to a geometric distribution with parameter p, since the probability of each message being the last one to be sent is p. Notice that, in the PTA for the honest recipient, an acknowledgement is sent after between ad and AD time units.

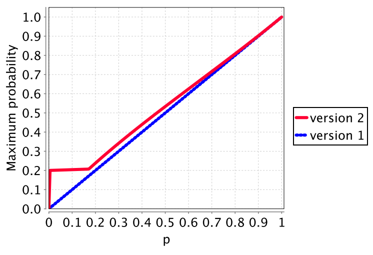


(a) p=0.01

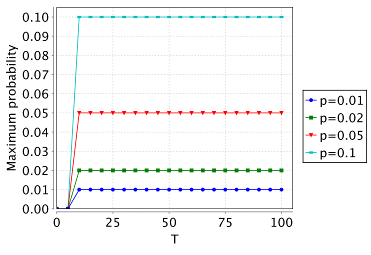


(b) p=0.1

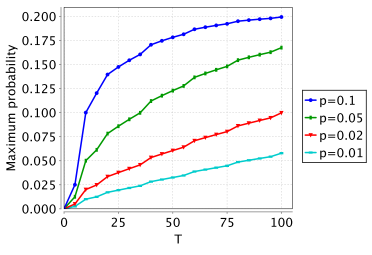
Fig. 5 Probability that the protocol terminates successfully by time T (honest version)



(a) Gains knowledge



(b) Gains knowledge by time T (variant 1)



(c) Gains knowledge by time T (variant 2)

Fig. 6 Maximum probability that the recipient gains knowledge (malicious versions)

We analysed these PTA models in PRISM using the stochastic games technique (since the originator PTA contains strict inequalities, the digital clocks method is not applicable). For the analysis, we assume AD=5 and ad=1.

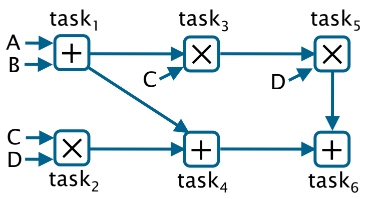
For the honest version of the protocol, the first property we consider is "*time-bounded*". More precisely, we check the formula $\mathtt{P}_{\geq 1}[\mathtt{F}\ \mathtt{done}]$, stating that the minimum probability of the protocol terminating correctly is 1. Next, we investigate the performance of the protocol with the quantitative properties $\mathtt{P}_{\mathtt{min}=?}[\mathtt{F}^{\leq T}\mathtt{done}]$ and $\mathtt{P}_{\mathtt{max}=?}[\mathtt{F}^{\leq T}\mathtt{done}]$, i.e., the minimum and maximum probability of termination by time T. Figure 5 plots these values for T between 0 and 100, with p=0.01 and p=0.1. We see that increasing the parameter p improves the performance of the protocol when the parties behave honestly. However, as we shall see below, when the recipient behaves maliciously, increasing this parameter comes at a cost since it also increases the likelihood that the recipient gains an advantage.

For the two models with a malicious recipient, we find that the minimum and maximum probability of the protocol terminating correctly ($\mathtt{P}_{\mathtt{min}=?}[\mathtt{F}\ \mathtt{done}]$ and $\mathtt{P}_{\mathtt{max}=?}[\mathtt{F}\ \mathtt{done}]$) are 0 and 1, respectively. The minimum probability is achieved when the (malicious) recipient returns no acknowledgements at all and the maximum when it acts honestly.

Figure 6 presents results for the properties $\mathtt{P}_{\mathtt{max}=?}[\mathtt{F}\ \mathtt{unfair}]$ and $\mathtt{P}_{\mathtt{max}=?}[\mathtt{F}^{\leq T}\mathtt{unfair}]$, for both variants of the malicious recipient and several values of parameter p. These correspond to the maximum probability that the recipient (eventually, or within time bound T) gains an advantage. For the first variant (describing the scenario in [61]), Figure 6(a) shows that the protocol is indeed ε-fair with ε equal to p. Essentially, all this recipient can do to gain knowledge is to correctly guess which message is the last (which, when a message arrives, is true with probability p). As shown in Figure 6(b), the probability of gaining an advantage over time remains constant after the arrival of the first message: since each message has an equal chance of being the last, there is nothing to be gained by waiting for a later one. The

	P_1	P_2
+	2 picoseconds	5 picoseconds
$\times$	3 picoseconds	7 picoseconds
idle	10 Watts	20 Watts
active	90 Watts	30 Watts

(a) Processor specification



(b) Task graph

Fig. 7 Scheduling problem for computing the term $D\times(C\times(A{+}B))+((A{+}B)+(C{\times}D))$

figures also show that the malicious recipient in the second variant of the model (which has a chance of correctly decoding the message before the deadline AD) has a greater chance of gaining knowledge, thanks to its additional power.

6.2 Task-Graph Scheduling

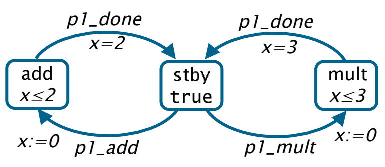
One common use of (non-probabilistic) timed automata is the formalisation and solution of scheduler optimisation problems. In this case study, we consider an extension of the *task-graph scheduling* problem described in [27]. We show how PTAs can be used to introduce uncertainty with regards to time delays and to consider the possibility of failures. We demonstrate how the digital clocks method can determine optimal schedulers for these problems, in terms of the (expected) time and energy consumption required to complete all tasks.

The Basic Model. First, we introduce the basic problem formalisation and model presented in [27]. This considers the example of evaluating the expression $D\times(C\times(A{+}B))+((A{+}B)+(C{\times}D))$, where its subterms are evaluated on two processors, P_1 and P_2. Processor P_1 is faster than P_2, but also consumes more energy. Figure 7(a) shows the specifications of the processors, in terms of their processing times and energy usage for addition and multiplication operations. Figure 7(b) presents the *task graph* for this example, illustrating the set of tasks (corresponding to subterms of the expression) and the dependencies that exist between tasks (in terms of their required order of evaluation).

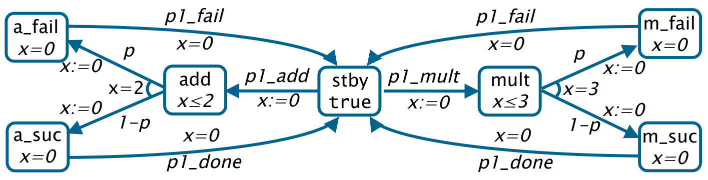
In [27], a (non-probabilistic) timed automaton model is built, consisting of the parallel composition of one automaton for each processor and one automaton for the scheduler which decides when tasks get performed and on which processors. The scheduler automaton includes integer-valued variables to indicate whether a task still has to be processed, is currently being processed or has been completed. To ensure that the restrictions of the task graph are met, there are conditions placed on enabling conditions such that a task cannot be scheduled until all of its dependencies have been computed.

The timed automaton for processor P_1 is given in Figure 8(a). The actions *p1_add* and *p1_mult* correspond to an addition and multiplication task being started on P_1, and *p1_done* indicates the completion of a task. The clock x is used to record the time that a task has been running and is initialised when a task is started. The automaton for P_2 is similar except that the action names change and the delays are modified to reflect the values in Figure 7(a).

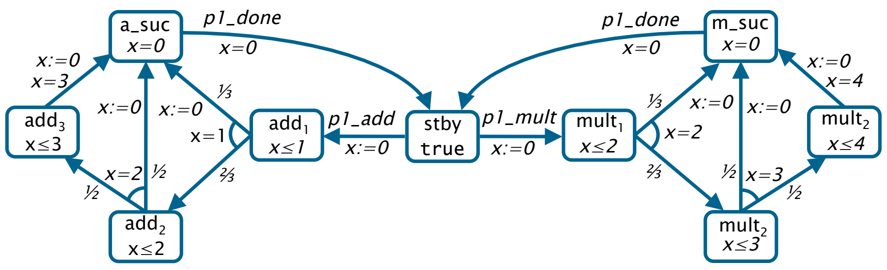
The time and energy consumed to complete all tasks are modelled as rewards (called prices in [27]). We introduce reward structures `time` and `energy`, respectively, to represent each of these quantities. For the case of time, only the scheduler automaton has a non-zero reward structure, where the location reward is 1 in each location and all action rewards are zero. For the case of energy consumption, the scheduler automaton has a zero reward



(a) Processor P_1



(b) Faulty version of processor P_1



(c) Processor P_1 with random delays

Fig. 8 PTAs for the task-graph scheduling case study

structure, while each processor has a location reward equal to the current rate of energy usage (as shown in Figure 7(a)) and has zero action rewards.

We built a PTA model for this case study using PRISM and, by applying the digital clocks method, calculated both the minimum (expected) time and energy consumption for completion of all tasks. For this, we used the two quantitative reward properties $\mathtt{R}^{\mathtt{time}}_{\mathtt{min}=?}[\mathtt{F\ complete}]$ and $\mathtt{R}^{\mathtt{energy}}_{\mathtt{min}=?}[\mathtt{F\ complete}]$. We also used PRISM to generate the corresponding schedulers that achieve these optimal values. The results agree with those reported in [27]. A scheduler that minimises the elapsed time requires 12 picoseconds to complete all tasks and schedules the tasks as follows:

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	task1		task3			task5			task4		task6									
P_2	task2																			

On the other hand, a scheduler optimising the energy consumption requires 1.3200 nanojoules (and 19 picoseconds) and makes the following scheduling decisions:

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	task1		task3			task4														
P_2	task2							task5							task6					

Due to the additional energy consumption of processor P_1, the first scheduler above, which optimises the time for task completion, requires 1.3900 nanojoules.

Random Task Execution Times. Now, we extend the formalisation of the task-graph problem, making the time required for each processor to perform a task probabilistic (in a more general setting, we can easily envisage situations where the exact time required to complete a task is unknown, but can be represented by some probability distribution). More precisely, we consider the following simple scenario. If, in the original problem the time for a processor to perform a task was $k \in \mathbb{N}$, we suppose now that the time taken is uniformly distributed between the delays $k-1$, k and $k+1$, e.g. the time for P_1 to perform a multiplication operation is either 1, 2 or 3 and the probability of each execution time is $\frac{1}{3}$.

The PTA for processor P_1 with random delays is presented in Figure 8(c) where, to ease notation, we have omitted action labels if they do not synchronise. Additional locations are added to encode the random delays. For example, in the case of multiplication, with probability $\frac{1}{3}$ the task completes after 2 time units; with probability $\frac{2}{3}$, the PTA moves to a location where, with probability $\frac{1}{2}$ the task completes after 1 additional time unit (i.e., of a total of 3 time units) or moves to a location where the task completes after 2 more time units (i.e., 4 time units in total). When the task completes, the PTA moves to a location where no time can pass (clock x is reset upon entering and the invariant of the location is $x{\leq}0$) and immediately notifies the scheduler the task is computed through action *p1_done*. To prevent the scheduler from seeing into the future when making decisions, the probabilistic choice for task completion is made on completion rather than on initialisation.

Analysing this model, we find that the optimal expected time and energy consumption to complete all tasks equals 12.226 picoseconds and 1.3201 nanojoules, respectively. This improves on the results obtained using the optimal schedulers for the original model, where the expected time and energy consumption equal 13.1852 picoseconds and 1.3211 nanojoules. Examining the optimal schedulers, we find that they change their decision based upon the delays of previously completed tasks. For example, for elapsed time, the optimal scheduler starts as for the non-probabilistic case, first scheduling task1 followed by task3 on P_1 and task2 on P_2. However, it is now possible for task2 to complete before task3 (if the execution times for task1, task2 and task3 are 3, 6 and 4 respectively), in which case the optimal scheduler now makes a different decision from the non-probabilistic case. Under one possible set of execution times for the remaining tasks, the optimal scheduling is as follows:

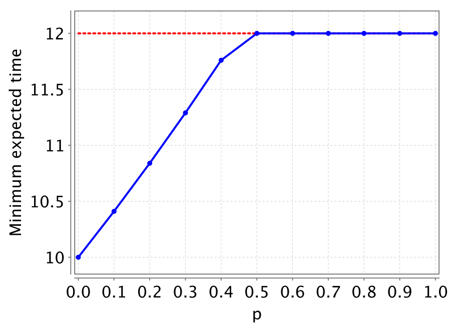
time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	task1			task3				task5				task6								
P_2	task2						task4													

Adding a Faulty Processor. As a second extension of the scheduling problem, we add a third processor P_3 which consumes the same energy as P_2 but is faster (addition takes 3 picoseconds and multiplication 5 picoseconds). However, this comes at a cost: there is a chance (probability p) that the processor fails and the computation must be rescheduled and performed again.

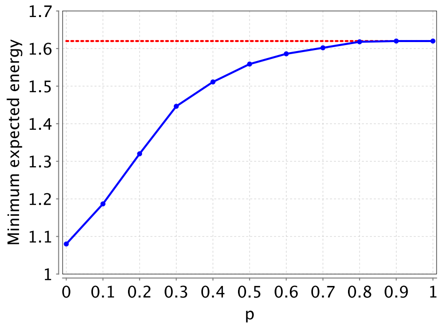
In Figure 8(b), we show the PTA for the faulty version of processor P_1. In this PTA, when a task completes, there is a probabilistic choice between moving to a location corresponding to successful completion and one to failure. In both cases, we move to a location where no time can pass and immediate notify the scheduler of either the success or failure of the computation. The automaton for the scheduler also changes for this model since it must react to the failure signals from the processors. In addition, the reward structure `energy` is extended to include the energy consumed by the additional processor.

The graphs in Figure 9 plot the optimal expected time and energy consumption for this extended model as the failure probability p varies. The dashed lines show the optimal results for the original model, i.e., when not using the processor P_3. As can be seen, once the probability of failure becomes sufficiently large, there is no gain in using the processor P_3 but, while when the probability of failure is small, it uses offers considerable gains in performance. To illustrate this fact, below we give a scheduler that optimises (minimises) the expected energy consumption when $p{=}0.5$. Dark boxes for tasks are used to denote processor P_3 failing to complete a task correctly, meaning that the task needs to be rescheduled.

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1				task3											task6					
P_2	task2							task5												
P_3	task1							task4												



(a) Expected time



(b) Expected energy consumption

Fig. 9 Optimal expected time and energy consumption as the failure probability of processor P_3 varies

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1				task1		task3			task5				task6							
P_2	task2							task4												
P_3	task1																			

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1				task3							task4				task6					
P_2	task2							task5												
P_3	task1							task4												

Notice that the scheduler uses the processor P_3 for task1 and, if this task is completed successfully, it later uses P_3 for task4. However, if task1 fails to complete, P_3 is not used again.

7 Conclusions

In this paper, we have presented an introduction to the model of probabilistic timed automata and summarised the various techniques developed to perform probabilistic model checking. Verification of probabilistic real-time systems is an active field of research and further progress is required in several important directions. Examples include the development of verification techniques for *probabilistic timed games* [43,6] and for *probabilistic hybrid automata* [64,36,9]. The former have proved, in the non-probabilistic setting, to being applicable to a variety of useful synthesis problems [12]. The latter provide essential modelling capabilities for domains such as embedded systems and cyber-physical systems; they represent a useful, but more tractable, subclass of the model of stochastic hybrid automata. Other important issues to investigate in the context of PTAs include robustness [7] and continuously-distributed time delays [53,1,60].

Acknowledgments. David Parker is part supported by ERC Advanced Grant VERIWARE. Jeremy Sproston is part supported by the project AMALFI (Advanced Methodologies for the AnaLysis and management of the Future Internet, Università di Torino/Compagnia di San Paolo). We thank the anonymous referees for various useful comments.

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for probabilistic real-time systems. In: Proc. 19th International Colloquium on Automata, Languages and Programming (ICALP'91), *LNCS*, vol. 510, pp. 115–136. Springer (1991)
2. Alur, R., Courcoubetis, C., Dill, D.: Model checking in dense real time. Information and Computation **104**(1), 2–34 (1993)

3. Alur, R., Courcoubetis, C., Halbwachs, N., Dill, D., Wong-Toi, H.: Minimization of timed transition systems. In: R. Cleaveland (ed.) Proc. 3rd Int. Conf. Concurrency Theory (CONCUR'92), *LNCS*, vol. 630, pp. 340–354. Springer (1992)
4. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**, 183–235 (1994)
5. Alur, R., La Torre, S., Pappas, G.: Optimal paths in weighted timed automata. Theoretical Computer Science **318**(3), 297–322 (2004)
6. Alur, R., Trivedi, A.: Relating average and discounted costs for quantitative analysis of timed systems. In: Proc. 11th Int. Conf. Embedded Software (EMSOFT'11), pp. 165–174. ACM (2011)
7. André, E., Fribourg, L., Sproston, J.: An extension of the inverse method to probabilistic timed automata. Formal Methods in System Design (2012). To appear
8. Asarin, E., Maler, O., Pnueli, A.: On discretization of delays in timed automata and digital circuits. In: D. Sangiorgi, R. de Simone (eds.) Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), *LNCS*, vol. 1466, pp. 470–484. Springer (1998)
9. Assouramou, J., Desharnais, J.: Analysis of non-linear probabilistic hybrid systems. In: M. Massink, G. Norman (eds.) Proc. 9th Workshop Quantitative Aspects of Programming Languages (QAPL'11), pp. 104–119 (2011)
10. Baier, C., Kwiatkowska, M.: Model checking for a probabilistic branching time logic with fairness. Distributed Computing **11**(3), 125–155 (1998)
11. Beauquier, D.: On probabilistic timed automata. Theoretical Computer Science **292**(1), 65–84 (2003)
12. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Proc. 19th International Conference on Computer Aided Verification (CAV'07), *LNCS*, vol. 4590, pp. 121–125. Springer (2007)
13. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Proc. 3rd Int. Conf. Quantitative Evaluation of Systems (QEST'06), pp. 125–126. IEEE (2006)
14. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for linearly priced timed automata. In: M.D. Benedetto, A. Sangiovanni-Vincentelli (eds.) Proc. 4th Int. Workshop Hybrid Systems: Computation and Control (HSCC'01), *LNCS*, vol. 2034, pp. 147–162. Springer (2001)
15. Bérard, B., Petit, A., Diekert, V., Gastin, P.: Characterization of the expressive power of silent transitions in timed automata. Fundamenta Informaticae **36**(2-3), 145–182 (1998)
16. Berendsen, J., Chen, T., Jansen, D.: Undecidability of cost-bounded reachability in priced probabilistic timed automata. In: J. Chen, S.B. Cooper (eds.) Proc. 6th Conf. Theory and Applications of Models of Computation (TAMC'09), *LNCS*, vol. 5532, pp. 128–137. Springer (2009)
17. Berendsen, J., Jansen, D., Katoen, J.P.: Probably on time and within budget: On reachability in priced probabilistic timed automata. In: Proc. 3rd Int. Conf. Quantitative Evaluation of SysTems (QEST'06), pp. 311–322. IEEE (2006)
18. Berendsen, J., Jansen, D., Vaandrager, F.: Fortuna: Model checking priced probabilistic timed automata. In: Proc. 7th Int. Conf. Quantitative Evaluation of SysTems (QEST'10), pp. 273–281. IEEE (2010)
19. Beyer, D.: Improvements in BDD-based reachability analysis of timed automata. In: J. Oliveira, P. Zave (eds.) Int. Symp. Formal Methods Europe, FME 2001: Formal Methods for Increasing Software Productivity, *LNCS*, vol. 2021, pp. 318–343. Springer (2001)
20. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: P. Thiagarajan (ed.) Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95), *LNCS*, vol. 1026, pp. 499–513. Springer (1995)
21. Bohnenkamp, H., D'Argenio, P., Hermanns, H., Katoen, J.P.: Modest: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Engineering **32**(10), 812–830 (2006)
22. Bouyer, P.: Untameable timed automata! In: H. Alt, M. Habib (eds.) Proc. 20th Int. Symp. Theoretical Aspects of Computer Science (STACS'03), *LNCS*, vol. 2607, pp. 620–631. Springer (2003)
23. Bouyer, P.: From qualitative to quantitative analysis of timed systems. Mémoire d'habilitation, Université Paris 7, Paris, France (2009)
24. Bouyer, P., Brinksma, E., Larsen, K.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design **32**(1), 3–23 (2008)
25. Bouyer, P., Chevalier, F.: On conciseness of extensions of timed automata. Journal of Automata, Languages and Combinatorics **10**(4), 393–405 (2005)
26. Bouyer, P., Dufourd, C., Fleury, E., Petit, A.: Updatable timed automata. Theoretical Computer Science **321**(2–3), 291–345 (2004)
27. Bouyer, P., Fahrenberg, U., Larsen, K., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. Communications of the ACM **54**(9), 78–87 (2011)
28. Chen, T., Han, T., Katoen, J.P.: Time-abstracting bisimulation for probabilistic timed automata. In: Proc. 2nd IEEE Int. Symp. Theoretical Aspects of Software Engineering (TASE'08), pp. 177–184. IEEE (2008)

29. Clarke, E., Emerson, A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: D. Kozen (ed.) Proc. Workshop Logic of Programs, *LNCS*, vol. 131. Springer (1981)
30. Daws, C., Yovine, S.: Two examples of verification of multirate timed automata with KRONOS. In: Proc. IEEE Real-Time Systems Symposium (RTSS'95), pp. 66–75. IEEE (1995)
31. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proc. 25th Annual IEEE Symposium on Logic in Computer Science (LICS'10), pp. 342–351. IEEE Computer Society (2010)
32. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: M. Bernardo, V. Issarny (eds.) Formal Methods for Eternal Networked Software Systems (SFM'11), *LNCS*, vol. 6659, pp. 53–113. Springer (2011)
33. Fruth, M.: Probabilistic model checking of contention resolution in the IEEE 802.15.4 low-rate wireless personal area network protocol. In: Proc. 2nd Int. Symp. Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'06) (2006)
34. Glässer, U., Rastkar, S., Vajihollahi, M.: Modeling and validation of aviation security. In: Intelligence and Security Informatics, vol. 135, pp. 337–355. Springer (2008)
35. Gregersen, H., Jensen, H.E.: Formal design of reliable real time systems. Master's thesis, Department of Mathematics and Computer Science, Aalborg University (1995)
36. Hahn, E.M., Norman, G., Parker, D., Wachter, B., Zhang, L.: Game-based abstraction and controller synthesis for probabilistic hybrid systems. In: Proc. 8th Int. Conf. Quantitative Evaluation of SysTems (QEST'11), pp. 69–78. IEEE (2011)
37. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5), 512–535 (1994)
38. Hartmanns, A., Hermanns, H.: A modest approach to checking probabilistic timed automata. In: Proc. 6th Int. Conf. Quantitative Evaluation of SysTems (QEST'09), pp. 187–196. IEEE (2009)
39. Henzinger, T., Manna, Z., Pnueli, A.: What good are digital clocks? In: W. Kuich (ed.) Proc. 19th Int. Colloq. Automata, Languages and Programming (ICALP'92), *LNCS*, vol. 623, pp. 545–558. Springer (1992)
40. Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation **111**(2), 193–244 (1994)
41. Hermanns, H.: Interactive Markov Chains and the Quest for Quantified Quality, *LNCS*, vol. 2428. Springer Verlag (2002)
42. Jensen, H.: Model checking probabilistic real time systems. In: B. Bjerner, M. Larsson, B. Nordström (eds.) Proc. 7th Nordic Workshop Programming Theory, Report 86, pp. 247–261. Chalmers University of Technology (1996)
43. Jurdziński, M., Kwiatkowska, M., Norman, G., Trivedi, A.: Concavely-priced probabilistic timed automata. In: M. Bravetti, G. Zavattaro (eds.) Proc. 20th Int. Conf. Concurrency Theory (CONCUR'09), *LNCS*, vol. 5710, pp. 415–430. Springer (2009)
44. Jurdzinski, M., Sproston, J., Laroussinie, F.: Model checking probabilistic timed automata with one or two clocks. Logical Methods in Computer Science **4**(3) (2008)
45. Katoen, J.P., Hahn, E.M., Hermanns, H., Jansen, D., Zapreev, I.: The ins and outs of the probabilistic model checker MRMC. In: Proc. 6th Int. Conf. Quantitative Evaluation of Systems (QEST'09), pp. 167–176. IEEE (2009)
46. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Formal Methods in System Design **36**(3), 246–280 (2010)
47. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains, 2nd edn. Springer-Verlag (1976)
48. Krause, C., Giese, H.: Model checking probabilistic real-time properties for service-oriented systems with service level agreements. In: Proc. 13th Int. Workshop Verification of Infinite-State Systems (INFINITY'11), *EPTCS*, vol. 73, pp. 64–78. Elsevier (2011)
49. Kwiatkowska, M., Norman, G., Parker, D.: Game-based abstraction for Markov decision processes. In: Proc. 3rd Int. Conf. Quantitative Evaluation of Systems (QEST'06), pp. 157–166. IEEE (2006)
50. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: J. Ouaknine, F. Vaandrager (eds.) Proc. 7th Int. Conf. Formal Modelling and Analysis of Timed Systems (FORMATS'09), *LNCS*, vol. 5813, pp. 212–227. Springer (2009)
51. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: G. Gopalakrishnan, S. Qadeer (eds.) Proc. 23rd Int. Conf. Computer Aided Verification (CAV'11), *LNCS*, vol. 6806, pp. 585–591. Springer (2011)
52. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods in System Design **29**, 33–78 (2006)
53. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Verifying quantitative properties of continuous probabilistic timed automata. In: C. Palamidessi (ed.) In Proc. 11th International Conference on Concurrency Theory (CONCUR'00), *LNCS*, vol. 1877, pp. 123–137. Springer (2000)

54. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theoretical Computer Science **282**, 101–150 (2002)
55. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. Formal Aspects of Computing **14**(3), 295–318 (2003)
56. Kwiatkowska, M., Norman, G., Sproston, J., Wang, F.: Symbolic model checking for probabilistic timed automata. Information and Computation **205**(7), 1027–1077 (2007)
57. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Automatic analysis of a non-repudiation protocol. In: Proc. Second Workshop Quantitative Aspects of Programming Languages (QAPL 2004), *ENTCS*, vol. 112, pp. 113–129 (2005)
58. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Information and Computation **94**, 1–28 (1991)
59. Larsen, K., Yi, W.: Time-abstracted bisimulation: Implicit specifications and decidability. Information and Computation **134**(2), 75–101 (1997)
60. Maler, O., Larsen, K., Krogh, B.: On zone-based analysis of duration probabilistic automata. In: Proc. 12th Int. Workshop Verification of Infinite-State Systems (INFINITY'10), *EPTCS*, vol. 39, pp. 33–46 (2010)
61. Markowitch, O., Roggeman, Y.: Probabilistic non-repudiation without trusted third party. In: Proc. 2nd Workshop Security in Communication Networks (1999)
62. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing **2**(2), 250–273 (1995)
63. Sproston, J.: Strict divergence for probabilistic timed automata. In: M. Bravetti, G. Zavattaro (eds.) 20th Int. Conf. Concurrency Theory (CONCUR 2009), *LNCS*, vol. 5710, pp. 620–636. Springer (2009)
64. Sproston, J.: Discrete-time verification and control for probabilistic rectangular hybrid automata. In: Proc. 8th Int. Conf. Quantitative Evaluation of SysTems (QEST'11), pp. 79–88. IEEE (2011)
65. Tripakis, S.: The analysis of timed systems in practice. Ph.D. thesis, Université Joseph Fourier, Grenoble (1998)
66. Tripakis, S.: Verifying progress in timed systems. In: J.P. Katoen (ed.) Proc. 5th Int. AMAST Workshop Real-Time and Probabilistic Systems (ARTS'99), *LNCS*, vol. 1601, pp. 299–314. Springer (1999)
67. Tripakis, S., Yovine, S., Bouajjani, A.: Checking timed Büchi automata emptiness efficiently. Formal Methods in System Design **26**(3), 267–292 (2005)
68. UPPAAL PRO web site. `http://people.cs.aau.dk/~arild/uppaal-probabilistic/`
69. Zhang, M., Hung, D.V.: Formal analysis of streaming downloading protocol for system upgrading. In: Proc. 4th Workshop Quantitative Aspects of Programming Languages (QAPL 06), *ENTCS*, vol. 164(3), pp. 205–224 (2006)