Model Checking Probabilistic Real-Time Properties for Service-Oriented Systems with Service Level Agreements

Christian Krause*
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
christian.krause@hpi.uni-potsdam.de

Holger Giese
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
holger.giese@hpi.uni-potsdam.de

The assurance of quality of service properties is an important aspect of service-oriented software engineering. Notations for so-called *service level agreements* (SLAs), such as the Web Service Level Agreement (WSLA) language, provide a formal syntax to specify such assurances in terms of (legally binding) contracts between a service provider and a customer. On the other hand, formal methods for verification of probabilistic real-time behavior have reached a level of expressiveness and efficiency which allows to apply them in real-world scenarios. In this paper, we suggest to employ the recently introduced model of Interval Probabilistic Timed Automata (IPTA) for formal verification of QoS properties of service-oriented systems. Specifically, we show that IPTA in contrast to Probabilistic Timed Automata (PTA) are able to capture the guarantees specified in SLAs directly. A particular challenge in the analysis of IPTA is the fact that their naive semantics usually yields an infinite set of states and infinitely-branching transitions. However, using symbolic representations, IPTA can be analyzed rather efficiently. We have developed the first implementation of an IPTA model checker by extending the PRISM tool and show that model checking IPTA is only slightly more expensive than model checking comparable PTA.

1 Introduction

One of the key tasks in engineering service-oriented systems is the assurance of quality of service (QoS) properties, such as 'the response time of a service is less than 20ms for at least 95% of the requests'. *Service level agreements* (SLAs) provide a notation for specifying such guarantees in terms of (legally binding) contracts between a service provider and a service consumer. A specific example for an SLA notation is the Web Service Level Agreement (WSLA) [8, 3] language, which provides a formal syntax to specify such QoS guarantees for web services. The compliance of a service implementation with an SLA is commonly checked at runtime by means of monitoring them.

However, due to the fact that an application or service may itself make use of other services, guaranteeing probabilistic real-time properties can be difficult. The problem becomes even harder, when the service is not bound to a specific service provider but linked dynamically. Statistical testing of the service consumer together with all currently possible service providers can provide some evidence that the required probabilistic real-time properties hold. However, each time a new service provider is connected or in situations when a known service provider slightly changes the characteristics of the offered service, the test results are no longer representative.

In the last couple of years, formal methods for verification of probabilistic real-time behavior have reached a level of expressiveness and efficiency that allows to apply them to real-world case studies

*Supported by the research school in 'Service-Oriented Systems Engineering' at the Hasso Plattner Institute (HPI).

F. Yu and C. Wang: The 13th International Workshop on
Verification of Infinite State Systems (INFINITY'11)
EPTCS 73, 2011, pp. 64–78, doi:10.4204/EPTCS.73.8

in various application domains, including communication and multimedia protocols, randomized distributed algorithms and biological systems (cf. [15, 17]). Therefore, it is a natural step to investigate also their suitability to address the outlined challenges for guaranteeing QoS properties of service-oriented systems. In particular, dynamically linking of services in service-oriented systems introduces major difficulties concerning the analysis of their QoS properties.

In this paper, we suggest to employ the recently introduced model of Interval Probabilistic Timed Automata [18] (IPTA) which extend Probabilistic Timed Automata [12] (PTA) by permitting to specify intervals, i.e., lower and upper bounds for probabilities, rather than exact values. The contributions of this paper can be summarized as follows: (1) We show that IPTA (in contrast to PTA) are able to capture the guarantees specified in SLAs directly. The notion of probabilistic uncertainty in IPTA allows modeling and verifying service-oriented systems with dynamic service binding, where one can rely only on the guarantees stated in the SLA and no knowledge about the actual service implementation is available. (2) To the best of our knowledge, we present the first implementation of an IPTA model checker and show that it can analyze IPTA nearly as fast as comparable PTA. (3) We show that a naive analysis using sampling of PTA does not yield the correct results as predicted by IPTA. Furthermore, we provide evidence that checking equivalent PTA has a worse performance than checking the IPTA directly.

Organization

The rest of this paper is organized as follows. Section 2 demonstrates that IPTA naturally permit to capture the guarantees of an SLA when modeling the behavior of a service provider. Section 3 introduces the syntax and semantics of interval probabilistic timed automata. Section 4 discusses symbolic PTCTL model checking and the probabilistic reachability problem. In Section 5 we present our tool support. In Section 6 we show that IPTA checking is only slightly more expensive than PTA checking. We show that using sampling of the probability values in the intervals to derive a representative set of PTA does neither scale as good as IPTA checking nor does it work correctly. Finally, we demonstrate that also an encoding of IPTA in form of a PTA does not scale as good as IPTA checking. In Section 7 we discuss related work. Section 8 contains conclusions and future work.

2 Quality of Service Modeling

Since in the service-oriented paradigm, compositionality is employed to construct new services and applications, the interaction behavior of a service-oriented system can be captured by a set of communicating finite state automata. For instance, a simple service-oriented system can consist of a service provider and a service consumer, both represented as automata, which communicate according to a specific protocol, given by a service contract.

The QoS of a service-oriented application is often as important as its functional properties. Validation of QoS characteristics usually requires models, which capture probabilistic aspects as well as real-time properties. Probabilistic Timed Automata [12] (PTA) are an expressive, compositional model for probabilistic real-time behavior with support for non-determinism. However, a limitation of PTA is the fact that only fixed values for probabilities can be expressed. In practice, it is often only possible to approximate probabilities with guarantees for lower and upper bounds. For this reason, Interval Probabilistic Timed Automata [18] (IPTA) generalize PTA by allowing to specify intervals of probabilities as opposed to fixed values. This feature is particularly useful to model guarantees for probabilities as commonly found in service level agreements (SLAs).

Listing 1: A response time guarantee in WSLA

```
<Metric name="NormalResponsePercentage" type="float" unit="Percentage">
  <Source>ServiceProvider</Source>
  <Function resultType="float" xsi:type="wsla:PercentageLessThanThreshold">
    <Metric>ResponseTime</Metric>
    <Value>
      <LongScalar>20</LongScalar>    <!-- Normal responses take less than 20ms -->
    </Value>
  </Function>
</Metric>

<Obligations>
  <ServiceLevelObjective name="ResponseTimeGuarantee">
    <Obliged>ServiceProvider</Obliged>
    <Expression>
      <Predicate xsi:type="GreaterEqual">
        <SLAParameter>NormalResponsePercentage</SLAParameter>
        <Value>0.95</Value>          <!-- At least 95% normal responses -->
      </Predicate>
    </Expression>
  </ServiceLevelObjective>
<Obligations>
```

As a concrete example of an SLA, Listing 1 contains an adaption of a WSLA specification presented in [8]. In the upper part, a metric called `NormalResponsePercentage` is defined which contains the percentage of response events which took less than 20ms. The actual service level agreement is defined in the lower part in terms of a service provider obligation called `ResponseTimeGuarantee`. This obligation assures that the percentage of responses that take less than 20ms is at least 95%.

Figure 1 depicts a PTA for a client/server application in which the server guarantees a probability of (exactly) 95% for response times of less than 20ms. The client is modeled as another PTA which synchronizes with the server using the *request* and *response* actions. Note that the client model is actually just a Timed Automata (TA), because no probabilities are employed. However, in the cases where probabilities also matter, we would need exact knowledge of them to be able construct a proper PTA.

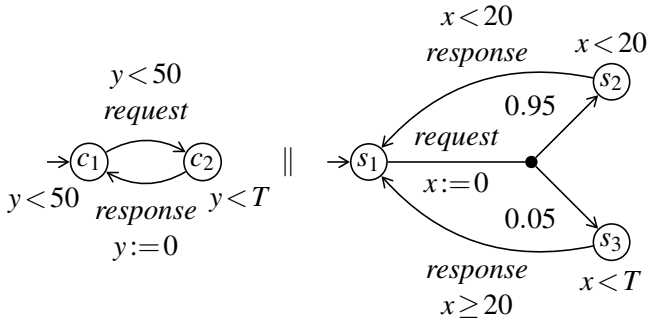

Figure 1: PTA for a client (left) and a server (right)

This small example shows that PTA, similarly to other automata models, consist of set of states (or

locations) and transitions (or *edges*). The time related behavior is specified using clocks (as x in the server) which can be reset ($x := 0$), tested in conditions for transitions ($x \geq 20$) and also state invariants ($x < 20$ for s_2). Note that we use the constant T to denote a constant timeout value in the invariant $x < T$. A clock such as x simply increases with progressing time, unless it is explicitly reset. The conditions block the transition until the clock constraint is fulfilled. Moreover, the state invariant ensures that (1) no transition leads to this state when this would result in invalidating the state invariant, and (2) the automaton can no longer stay in this state when this would also lead to a violation of the invariant. In addition to purely non-deterministic behavior, i.e. when multiple transitions with the same action are enabled, probabilities can be associated with transitions, e.g. 0.95 for the *request* transition leading to the state s_2, and 0.05 leading to the state s_3. Note that for probabilistic transitions, all alternative branches must sum up to 1. Formally, the target of a transition in a PTA is not a single state, but a discrete probability distribution over the set of all states. Thus, in addition to purely nondeterministic choice, PTA allow to specify the likelihood of an event. Note also that the existence of a parallel operator (written as $\mathscr{P}_1 \parallel \mathscr{P}_2$ where $\mathscr{P}_1$ and $\mathscr{P}_2$ are PTA) moreover allows to synchronize two automata via shared actions, which enables compositional modeling.

However, the Interval Probabilistic Timed Automaton (IPTA) of a server in Figure 2 additionally allows to capture the 'at least 95%' semantics of the SLA in Listing 1. The difference to the PTA model is that in IPTA it is possible to specify probabilistic behavior with a level of uncertainty. Specifically, IPTA allow to specify intervals of probabilisties as opposed to the exact probabilities used in PTA. The semantics of intervals in contrast to exact values is that each time a probabilistic decision is necessary, any of the usually uncountable many probability distributions which lie within the lower and upper bounds of the intervals denote a valid behavior. Therefore, probability intervals match better with the guarantees commonly found in SLAs, such as 'with at least 95% a request is answered within 20ms'. Note that we did not model the client as another IPTA here, but just as the TA in Figure 1. However, similarly to the modeled server, we can also model uncertain probabilistic behavior in the client, such as 'with at least 75% a request is made within 50ms'. The parallel composition of IPTA then allows to derive a model of the complete system.

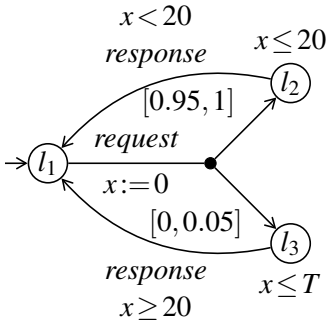

Figure 2: IPTA for a simple server

Given such models in form of PTA or IPTA, we can now employ model checking to verify probabilistic real-time properties for the composed system, specified in an appropriate probabilistic real-time logic. In our case, we might be interested in the property 'the probability that 1 out of 10 responses is too slow is at most 5%'. As we will demonstrate later in this paper, there are important differences between the outcome of such an analysis depending on whether we employ the PTA using exact probabilities or the IPTA which allows to specify only lower and upper bounds. In particular, no sample set of PTA

derived from the IPTA by choosing values from the interval is in general sufficient to derive the same result as the analysis of the IPTA.

3 Interval Probabilistic Timed Automata

Interval probabilistic timed automata (IPTA) [18] integrate the probabilistic real-time modeling concepts of probabilistic timed automata (PTA) [12] and the idea of probabilistic uncertainty known from interval Markov chains [16]. Thus, they not only provide a way to distinguish between purely probabilistic and nondeterministic (timed) behavior, but also allow to specify uncertain probabilities using lower and upper bounds. These ingredients make IPTA a suitable formal model for the specification and verification of QoS assurances that can be commonly found in SLAs.

3.1 Preliminaries

Discrete probability distributions

For a finite set S, $Dist(S)$ is the set of *discrete probability distributions* over S, i.e., the set of all functions $\mu : S \to [0,1]$, with $\sum_{s \in S} \mu(s) = 1$. The *point distribution* $\mu_s^\bullet$ is the unique distribution on S with $\mu(s) = 1$.

Clocks, valuations and constraints

Let $\mathbb{R}_+$ denote the set of non-negative reals. Let $\mathscr{X} = \{x_1, \ldots, x_n\}$ be a set of variables in $\mathbb{R}_+$, called *clocks*. An $\mathscr{X}$*-valuation* is a map $v : \mathscr{X} \to \mathbb{R}_+$. For a subset $X \subseteq \mathscr{X}$, $v[X := 0]$ denotes the valuation v' with $v'(x) = 0$ if $x \in X$ and $v'(x) = v(x)$ if $x \notin X$. For $d \in \mathbb{R}_+$, $v + d$ is the valuation v'' with $v''(x) = v(x) + d$ for all $x \in \mathscr{X}$. A *clock constraint* ζ on $\mathscr{X}$ is an expression of the form $x \bowtie c$ or $x - y \bowtie c$ such that $x, y \in \mathscr{X}$, $c \in \mathbb{R}_+$ and $\bowtie \in \{\leq, <, >, \geq\}$, or a conjunction of clock constraints. A clock valuation v satisfies ζ, written as $v \triangleright \zeta$ if and only if ζ evaluates to true when all clocks $x \in \mathscr{X}$ are substituted with their clock value $v(x)$. Let $CC(\mathscr{X})$ denote the set of all clock constraints over $\mathscr{X}$.

3.2 Syntax

Before defining IPTA formally, we introduce a syntactical and, thus, finite notion of probability interval distributions.

Definition 3.1 (Interval distribution) *Let S be a finite set. A* probability interval distribution λ *on S is a pair of functions $\lambda = \langle \lambda^\ell, \lambda^u \rangle$ with $\lambda^\ell, \lambda^u : S \to [0,1]$, such that $\lambda^\ell(s) \leq \lambda^u(s)$ for all $s \in S$ and furthermore:*

$$\sum_{s \in S} \lambda^\ell(s) \leq 1 \leq \sum_{s \in S} \lambda^u(s) \tag{1}$$

The set of probability interval distributions over S is denoted by $IntDist(S)$. The support *of λ is defined as $Supp(\lambda) = \{s \in S \mid \lambda^u(s) > 0\}$. Let $\lambda_s^\bullet$ be the unique interval distribution that assigns $\langle 1,1 \rangle$ to s, and $\langle 0,0 \rangle$ to all $t \in S, t \neq s$.*

A probability interval distribution λ is a symbolic representation of the non-empty, possibly infinite set of probability distributions that are conform with the interval bounds: $\{\, \mu \in \mathit{Dist}(S) \mid \forall s \in S : \lambda^{\ell}(s) \leq \mu(s) \leq \lambda^{u}(s) \,\}$. If clear from the context, we may abuse notation and identify λ with this set and also write $\mu \in \lambda$ if and only if μ respects the bounds of λ. Note that interval distributions are also used (in a slightly different syntax) in the notion of *closed interval specifications* in [5]. However, the explicit definition using lower and upper interval bounds in our model enables a syntactical treatment of interval distributions which is useful, e.g., in the following notion of *minimal interval distributions*.

Definition 3.2 (Minimal interval distribution) *An interval distribution λ on S is called* minimal *if for all $s \in S$ the following conditions hold:*

1. $\lambda^{u}(s) + \sum_{t \in S, t \neq s} \lambda^{\ell}(t) \leq 1$
2. $\lambda^{\ell}(s) + \sum_{t \in S, t \neq s} \lambda^{u}(t) \geq 1$

Minimal interval distributions have the property that the bounds of all intervals can be reached (but not necessarily at the same time). Although minimality is formally not needed in the properties that we consider here, it is often a desirable requirement since it can serve as a sanity check for a specification. For instance, the interval distribution $\lambda = \{\, s \mapsto \langle 0.4, 0.5 \rangle, t \mapsto \langle 0.4, 0.5 \rangle \,\}$ is not minimal because condition 2 is violated. Here, the lower bounds of 0.4 can never be reached. In fact, the only probability distribution that is conform with the interval bounds is $\mu = \{\, s \mapsto 0.5, t \mapsto 0.5 \,\}$. Thus, the minimality condition is a useful requirement which allows to verify the validity of interval bounds. Note also that it is always possible to derive a minimal interval distribution from a non-minimal one by *pruning* the interval bounds, e.g., by setting $\lambda^{u}(s) := 1 - \sum_{t \in S, t \neq s} \lambda^{\ell}(t)$ if condition 1 is violated for the state s.

Definition 3.3 (Interval probabilistic timed automaton) *An* interval probabilistic timed automaton *is a tuple $\mathscr{I} = (L, L^0, \mathscr{A}, \mathscr{X}, \mathit{inv}, \mathit{prob}, \mathscr{L})$ consisting of:*

- *a finite set of locations L with $L^0 \subseteq L$ the set of initial locations,*
- *a finite set of action $\mathscr{A}$,*
- *a finite set of clocks $\mathscr{X}$,*
- *a clock invariant assignment function $\mathit{inv} : L \to \mathit{CC}(\mathscr{X})$*
- *a probabilistic edge relation $\mathit{prob} \subseteq L \times \mathit{CC}(\mathscr{X}) \times \mathscr{A} \times \mathit{IntDist}(2^{\mathscr{X}} \times L)$, and*
- *a labeling function $\mathscr{L} : L \to 2^{AP}$ assigning atomic propositions to locations.*

Note that for more flexibility and a clear separation between communication and state invariants, our IPTA model contains both actions on transitions and atomic propositions for states. This approach is also in line with our tool support based on an extended version of PRISM (see Section 5).

As an example, we consider the IPTA model of a simple server depicted in Figure 2, where we denote interval distributions by small black circles. The set of actions is $\mathscr{A} = \{\mathit{request}, \mathit{response}\}$, and the clocks are $\mathscr{X} = \{x\}$. For simplicity, we do not include atomic propositions here. Moreover, we associate the interval $[1,1]$ with edges that have a support of size 1. The server modeled by this IPTA responds to an incoming request within 20ms with a probability between 95% and 100%. These lower and upper bounds can arise in scenarios where the exact probabilities are unknown or cannot be given precisely, e.g., due to implementation details. For instance, one can imagine that the server relays all requests to an heterogeneous, internal server farm, in which the success probability depends on the currently chosen server.

Composition

An important aspect of the service-oriented paradigm is compositionality, i.e., the fact that new services can be built by composing existing ones. Therefore, it is also crucial to support composition at the modeling level. In our approach, a parallel operator for IPTA is used for this purpose. The parallel composition of IPTA is defined analogously to the one for PTA. However, we need to compose interval distributions instead of probability distributions.

Definition 3.4 (Parallel composition) *The* parallel composition *of two interval probabilistic timed automata* $\mathscr{I}_i = (L_i, L_i^0, \mathscr{A}_i, \mathscr{X}_i, inv_i, prob_i, \mathscr{L}_i)$ *with* $i \in \{1,2\}$ *is defined as:*

$$\mathscr{I}_1 \parallel \mathscr{I}_2 = (L_1 \times L_2, L_1^0 \times L_2^0, \mathscr{A}_1 \cup \mathscr{A}_2, \mathscr{X}_1 \cup \mathscr{X}_2, inv, prob, \mathscr{L})$$

such that

- $\mathscr{L}(\langle l_1, l_2 \rangle) = \mathscr{L}_1(l_1) \cup \mathscr{L}_2(l_2)$ *for all* $l_1 \in L_1, l_2 \in L_2$
- $inv(\langle l_1, l_2 \rangle) = inv_1(l_1) \wedge inv_2(l_2)$ *for all* $l_1 \in L_1, l_2 \in L_2$
- $\langle \langle l_1, l_2 \rangle, \zeta, a, \lambda \rangle \in$ *prob if and only if one of the following conditions hold:*
  1. $a \in \mathscr{A}_1 \setminus \mathscr{A}_2$ *and there exists* $\langle l_1, \zeta, a, \lambda_1 \rangle \in prob_1$ *such that* $\lambda = \lambda_1 \otimes \lambda^{\bullet}_{\langle \emptyset, l_2 \rangle}$
  2. $a \in \mathscr{A}_2 \setminus \mathscr{A}_1$ *and there exists* $\langle l_2, \zeta, a, \lambda_2 \rangle \in prob_2$ *such that* $\lambda = \lambda^{\bullet}_{\langle \emptyset, l_1 \rangle} \otimes \lambda_2$
  3. $a \in \mathscr{A}_1 \cap \mathscr{A}_2$ *and there exists* $\langle l_i, \zeta_i, a, \lambda_i \rangle \in prob_i$ *such that* $\lambda = \lambda_1 \otimes \lambda_2$ *and* $\zeta = \zeta_1 \wedge \zeta_2$

where for any $l_i \in L_i$, $X_i \subseteq \mathscr{X}_i$*:*

$$\lambda_1 \otimes \lambda_2(X_1 \cup X_2, \langle l_1, l_2 \rangle)^{\ell} \stackrel{\text{def}}{=} \lambda_1^{\ell}(X_1, l_1) \cdot \lambda_2^{\ell}(X_2, l_2)$$
$$\lambda_1 \otimes \lambda_2(X_1 \cup X_2, \langle l_1, l_2 \rangle)^{u} \stackrel{\text{def}}{=} \lambda_1^{u}(X_1, l_1) \cdot \lambda_2^{u}(X_2, l_2)$$

Thus, the product of two interval distributions is simply defined by the product of their lower and upper bounds. Note also that the parallel composition for IPTA synchronizes transitions via shared actions, and interleaves transitions via unshared actions.

3.3 Semantics

The semantics of IPTA can be given in terms of Timed Interval Probabilistic Systems (TIPS) [18], which are essentially infinite-state Interval Markov Decision Processes (IMDPs) [16].

Definition 3.5 (Timed interval probabilistic system) *A* timed interval probabilistic system *is a tuple* $\mathscr{T} = (S, S^0, \mathscr{A}, Steps, \mathscr{L})$ *consisting of:*

- *a set of states* S *with* $S^0 \subseteq S$ *the set of initial states,*
- *a set of actions* $\mathscr{A}$*, such that* $\mathscr{A} \cap \mathbb{R}_+ = \emptyset$*,*
- *a transition function* $Steps : S \to 2^{(\mathscr{A} \cup \mathbb{R}_+) \times IntDist(S)}$*, such that, if* $(a, \lambda) \in Steps(s)$ *and* $a \in \mathbb{R}_+$*, then* λ *is a point interval distribution, and*
- *a labeling function* $\mathscr{L} : S \to 2^{AP}$ *assigning atomic propositions to states.*

The operational semantics of a timed interval probabilistic system can be understood as follows. A *probabilistic transition*, written as $s \xrightarrow{a,\lambda,\mu} s'$, is made from a state $s \in S$ by:

1. nondeterministically selecting an action/duration and interval distribution pair $(a,\lambda) \in \mathit{Steps}(s)$,
2. nondeterministically choosing a probability distribution $\mu \in \lambda$,
3. making a probabilistic choice of target state s' according to μ.

A *path* of a timed interval probabilistic system is a non-empty finite or infinite sequence of probabilistic transitions:

$$\omega = s_0 \xrightarrow{a_0,\lambda_0,\mu_0} s_1 \xrightarrow{a_1,\lambda_1,\mu_1} s_2 \xrightarrow{a_2,\lambda_2,\mu_2} \ldots$$

where for all $i \in \mathbb{N}$ it holds that $s_i \in S$, $(a_i,\lambda_i) \in \mathit{Steps}(s_i)$, $\mu_i \in \lambda_i$ and $\mu_i(s_i) > 0$. We denote with $\omega(i)$ the $(i+1)$th state of ω, and with $\mathit{last}(\omega)$ the last state of ω, if it is finite. An *adversary* is a particular resolution of the nondeterminism in a timed interval probabilistic system $\mathscr{T}$. Formally, an adversary A for $\mathscr{T}$ is a function mapping every finite path ω of $\mathscr{T}$ to a triple (a,λ,μ), such that $(a,\lambda) \in \mathit{Steps}(\mathit{last}(\omega))$ and $\mu \in \lambda$. We restrict ourselves to *time-divergent* adversaries, i.e., we require that time has to advance beyond any given time bound. This is a common restriction in real-time models to rule out unrealizable behavior. The set of all time-divergent adversaries of $\mathscr{T}$ is denoted by $\mathit{Adv}_{\mathscr{T}}$.

For any $s \in S$ and adversary $A \in \mathit{Adv}_{\mathscr{T}}$, we let $\mathit{Paths}^A_{\mathit{finite}}(s)$ and $\mathit{Paths}^A_{\mathit{full}}(s)$ be the sets of all finite and infinite paths starting in s that correspond to A, respectively. Under a given adversary, the behavior of a timed interval probabilistic system is purely probabilistic. Formally, an adversary for a timed interval probabilistic system induces an infinite discrete-time Markov chain and, thus, a probability measure Prob^A_s over the set of paths $\mathit{Paths}^A_{\mathit{full}}(s)$ (cf. [9] for details). The semantics of an IPTA can be given by a TIPS as follows.

Definition 3.6 (TIPS semantics) *Given an IPTA $\mathscr{I} = (L, L^0, \mathscr{A}, \mathscr{X}, \mathit{inv}, \mathit{prob}, \mathscr{L})$. The* TIPS semantics *of $\mathscr{I}$ is the timed interval probabilistic system $\mathscr{T}_{\mathscr{I}} = (S, S^0, \mathscr{A}, \mathit{Steps}, \mathscr{L}')$ where:*

- $S \subseteq L \times \mathbb{R}_+^{\mathscr{X}}$, *such that $\langle l, v\rangle \in S$ if and only if $v \triangleright \mathit{inv}(l)$,*
- $S_0 = \{\, \langle l, v[\mathscr{X} := 0]\rangle \mid l \in L^0 \,\}$
- $\langle a,\lambda\rangle \in \mathit{Steps}(\langle l, v\rangle)$ *if and only if one of the following conditions holds:*
  - *Time transitions: $a = t \in \mathbb{R}_+$, $\lambda = \lambda^{\bullet}_{\langle l, v+t\rangle}$ and $v + t' \triangleright \mathit{inv}(l)$ for all $0 \le t' \le t$*
  - *Discrete transitions: $a \in \mathscr{A}$ and $\langle l, \zeta, \hat{\lambda}\rangle \in \mathit{prob}$ such that $v \triangleright \zeta$ and for any $\langle l', v'\rangle \in S$:*
    - $\lambda^{\ell}(l', v') = \sum_{X \subseteq \mathscr{X} \wedge v' = v[X:=0]} \hat{\lambda}^{\ell}(X, l')$
    - $\lambda^{u}(l', v') = \sum_{X \subseteq \mathscr{X} \wedge v' = v[X:=0]} \hat{\lambda}^{u}(X, l')$
- $\mathscr{L}'(\langle l, v\rangle) = \mathscr{L}(l)$ *for all $\langle l, v\rangle \in S$.*

4 Symbolic model checking

In this section, we recall the symbolic approach for PTCTL model checking as introduced for PTA in [13] and adapted for IPTA in [18]. Moreover, we discuss in more detail an iterative algorithm for computing the maximum and minimum probabilities for reaching a set of target states.

4.1 PTCTL – Probabilistic Timed Computation Tree Logic

Probabilistic Timed Computation Tree Logic (PTCTL) [12] can be used to specify combined probabilistic and timed properties. Constraints for probabilities in PTCTL are specified using the probabilistic threshold operator known from PCTL. Timing constraints in PTCTL are expressed using a set of *system*

clocks $\mathscr{X}$, which are the clocks from the automaton to be checked, and a set of *formula clocks* $\mathscr{Z}$, which is disjoint from $\mathscr{X}$. The syntax of PTCTL is given by:

$$\phi ::= a \mid \zeta \mid \neg\phi \mid \phi \vee \phi \mid z.\phi \mid \mathscr{P}_{\sim\kappa}[\phi \; \mathscr{U} \; \phi]$$

where:

- $a \in AP$ is an atomic proposition,
- $\zeta \in CC(\mathscr{X} \bigcup \mathscr{Z})$ is a clock constraint over all system and formula clocks,
- $z.\phi$ with $z \in \mathscr{Z}$ is a reset quantifier, and
- $\mathscr{P}_{\sim\kappa}[\_]$ is a probabilistic quantifier with $\sim \in \{\leq, <, >, \geq\}$ and $\kappa \in [0,1]$ a probability threshold.

As an example for the specification of a combined probabilistic and timed property, the requirement for a bounded response time, e.g. 'with a probability of at least 95% a response is sent within 20ms' can be formalized in PTCTL as the formula:

$$z.\mathscr{P}_{\geq 0.95}[\text{true} \; \mathscr{U} \; (\mathit{responseSent} \wedge z < 20)]$$

Furthermore, it is possible to specify properties over system clocks, e.g. the formula:

$$\mathscr{P}_{\leq 0.05}[(x \geq 4) \mathscr{U} (z = 8)]$$

represents the property 'with a probability of at most 5%, the system clock x exceeds 4 before 8 time units elapse'. For the formal semantics of PTCTL, we refer to [12].

4.2 Symbolic states

Since the timed interval probabilistic systems that are being generated as the semantics of an IPTA are in general infinite, it is crucial to find a finite representation which can be used for model checking. For this purpose, symbolic states are considered in [13, 18], which are formally given by a pair (l, ζ) of a location l and a clock constraint ζ, also referred to as *zone* in this context. A symbolic state (l, ζ) is a finite representation of the set of state and formula clock valuations $\{ \langle\langle l, v\rangle, \mathscr{E}\rangle \mid v, \mathscr{E} \triangleright \zeta \}$. Based on this finite representation using the notion of zones, PTCTL model checking is realized by recursively evaluating the parse tree of a given formula, computing the set of reachable symbolic states.

4.3 Probabilistic reachability

The probabilistic quantifier $\mathscr{P}_{\sim\kappa}[\_]$ can be evaluated by (i) computing the minimum and maximum probabilities for reaching a set of states, which is also referred to as the problem of *probabilistic reachability*, and (ii) comparing these probabilities with κ [13]. Formally, the problem of probabilistic reachability can be stated as follows. Let A be an adversary for a TIPS $\mathscr{T} = (S, s_0, \mathscr{A}, \mathit{Steps}, \mathscr{L})$, and $F \subseteq S$ be a set of target states. The probability of reaching F from a state $s \in S$ is defined as:

$$p_s^A(F) = \mathit{Prob}_s^A\{\omega \in \mathit{Paths}_{\mathit{full}}^A(s) \mid \exists i \in \mathbb{N} : \omega(i) \in F\}$$

Then, the *minimal and maximal reachability probabilities* of F are defined as:

$$p^{\min}(F) = \inf_{A \in Adv_{\mathscr{T}}} p_{s_0}^A(F) \qquad p^{\max}(F) = \sup_{A \in Adv_{\mathscr{T}}} p_{s_0}^A(F)$$

Iterative algorithm

The minimum and maximum probabilities for a set of target states in a TIPS can be computed using an iterative algorithm [16, 18] known as *value iteration*, which is used to solve the *stochastic shortest path problem* [1] for (interval) Markov decision processes.

Let $\mathscr{T} = (S, S^0, \mathscr{A}, Steps, \mathscr{L})$ be a timed interval probabilistic system and $F \subseteq S$ be a set of target states. Moreover, let $\overline{F} \subseteq S$ be the set of states from which F cannot be reached. We define $(p_n)_{n\in\mathbb{N}}$ as the sequence of probability vectors over S, such that for any $s \in S$:

- $p_n(s) = 1$ if $s \in F$ for all $n \in \mathbb{N}$,
- $p_n(s) = 0$ if $s \in \overline{F}$ for all $n \in \mathbb{N}$,
- $p_n(s)$ is computed iteratively if $s \in S \setminus (F \cup \overline{F})$ by:

$$p_0(s) = 0$$
$$p_{n+1}(s) = \max_{(a,\lambda)\in Steps(s)} \sum_{t\in Supp(\lambda)} \mu_\lambda^{\max}(t) \cdot p_n(t)$$

where we consider an ordering $t_1, t_2, \ldots t_N$ of the states $Supp(\lambda)$, such that the vector $p_n(t_1), p_n(t_2), \ldots, p_n(t_N)$ is in descending order, and $\mu_\lambda^{\max}$ is defined as follows with $m \in \{1, \ldots, N\}$:[1]

$$\mu_\lambda^{\max}(t_m) = \min\left(\lambda^u(t_m), \left(1 - \sum_{i=1}^{m-1} \mu_\lambda^{\max}(t_i) - \sum_{i=m+1}^{N} \lambda^\ell(t_i)\right)\right)$$

Then $p_n(s_0)$ converges to $p^{\max}(F)$ for $n \to \infty$. For a correctness proof of this algorithm we refer to [18]. Note also that except for the additional sorting of the support set, the complexity for computing the maximum and minimum probabilities for IPTA is the same as for PTA.

Note that PTCTL model checking (interval) probabilistic timed automata is EXPTIME-complete. However, for certain subclasses of PTCTL the model checking problem can be shown to be PTIME-complete (cf. [6]).

5 Tool Support

PRISM 4.0 [10] is the latest version of the probabilistic model checker developed at the University of Oxford. For various probabilistic models, including PTA, PRISM provides verification methods based on explicit and symbolic model checking, and discrete-event simulation.

We have extended PRISM 4.0 with support for IPTA.[2] Our implementation adds the new operator '$\sim$' to the PRISM language which can be used to specify probability intervals ($l \sim u : \ldots$) and not only exact probabilities ($0.95 : \ldots$). Moreover, we adapted the implementation for computing the minimum and maximum probabilities for reaching a set of target states based on the definitions in Section 4.3.

Listing 2 contains the PRISM code for the server IPTA in Figure 2 and an IPTA for a client which performs a fixed number of requests and then terminates. The constants `L` and `U` are used to declare the lower and upper interval bounds for a successful request, e.g. by setting `L=0.95` and `U=1` we obtain the IPTA in Figure 2. Note that we need to set the module type to `ipta` to be able to specify probability intervals. Fixed probabilities are also supported and interpreted as point intervals. Thus, any PTA model is also a valid IPTA model in our tool. Note also that the `invariant` section is used in PRISM 4.0 to associate clock invariants to locations, such as $x \leq 20$ for the state $s = 1$.

[1] Note that $\sum_{i=k}^{m} x \stackrel{\text{def}}{=} 0$ whenever $k > m$.

[2] Our IPTA extension of PRISM is available at `www.mdelab.org/?p=50`.

Listing 2: Client/Server system as a PRISM–IPTA

```
ipta                                                                                                    1
                                                                                                        2
const double L; // Lower probability for normal response                                                3
const double U; // Upper probability for normal response                                                4
const int REQUESTS; // Number of requests                                                               5
const int TIMEOUT = 30000; // Timeout value                                                             6
                                                                                                        7
module Server                                                                                           8
  s : [0..2] init 0;                                                                                    9
  w : [0..REQUESTS] init 0; // Number of slow responses                                                 10
  x : clock;                                                                                            11
  invariant                                                                                             12
    (s=0 ⇒ x≤100) & (s=1 ⇒ x≤20) & (s=2 ⇒ x≤TIMEOUT)                                                     13
  endinvariant                                                                                          14
                                                                                                        15
  [request] (s=0 & w<REQUESTS) → (L∼U):(s'=1)&(x'=0)                                                    16
                                 + ((1-U)∼(1-L)):(s'=2)&(w'=w+1)&(x'=0);                                17
  [response] (s=1 & x≤20) | (s=2 & x>20) → (s'=0)&(x'=0);                                                18
endmodule                                                                                               19
                                                                                                        20
module Client                                                                                           21
  t : [0..REQUESTS] init 0;                                                                             22
  y : clock;                                                                                            23
  invariant                                                                                             24
        (y<=TIMEOUT)                                                                                    25
  endinvariant                                                                                          26
                                                                                                        27
  [request] t<REQUESTS → (t'=t+1)&(y'=0);                                                               28
  []        t=REQUESTS → (y'=0);                                                                        29
endmodule                                                                                               30
                                                                                                        31
label "lessThan50PercentSlow" = (t=REQUESTS & w<REQUESTS/2);                                            32
```

Note also that we have extended the original server of the example in Figure 2 here by recording the number of slow responses that occurred so far using the variable `w`. Moreover, the client now performs only a pre-defined number of requests, given by the constant `REQUESTS`. This allows us to control and count the number of subsequent requests and (slow) responses and to reason about probabilities for specific scenarios, such as the probability that less than 50% of all requests will result in a slow response. This particular property is encoded using the label `lessThan50PercentSlow` in line 32. Note also that this definition of the client provides a convenient way to scale the size of the state space by increasing the number of requests, i.e. the constant `REQUESTS`. This is particularly useful for conducting benchmarks, e.g. for measuring the run-times of the model checker for different model sizes (cf. Section 6.3).

For the two modules defined in Listing 2, PRISM forms the system to be analyzed as the parallel composition of the server and the client, (cf. Definition 3.4). In the following section, we give an evaluation of our analysis approach and tool support using this example.

6 Evaluation

In this section, we compare the IPTA model in Listing 2 with PTA encodings of the same example. In particular, we show that PTA encodings either yield incorrect results (sampling with exact probabilities) or result in a blow-up of the model which causes a decay in the run-times of the model checker (equivalent model).

6.1 Difference to sampling

For an initial test, we have set the constants in our example to `L=0.7`, `U=0.8` and `REQUESTS=2`. Using the IPTA version of PRISM, we then calculated the minimum and maximum probabilities for the property that one out of two responses was slow: `(t=2 & w=1)`. The computed minimum and maximum probabilities are:

$$p^{\min}_{\texttt{ipta}} = 0.30, \qquad p^{\max}_{\texttt{ipta}} = 0.45$$

To illustrate the difference to approaches with fixed probabilities, we also encoded this example as a `pta` model, where we tested the following probabilities for normal response times: y=0.7, 0.75 and 0.8. For this model and the above property, we obtain the following probabilities:

$$p^{(y=0.7)}_{\texttt{pta}} = 0.42 \qquad p^{(y=0.75)}_{\texttt{pta}} = 0.375 \qquad p^{(y=0.8)}_{\texttt{pta}} = 0.32$$

It is obvious that these three samples are not sufficient to obtain the actual minimum and maximum probabilities as predicted using the IPTA model. In fact, no fixed value for y in the interval $[0.7, 0.8]$ produces the correct results, because the probability for the chosen property is minimal / maximal when y is chosen differently for each request. To illustrate this situation we computed the solutions analytically, depicted in the graph in Figure 3.

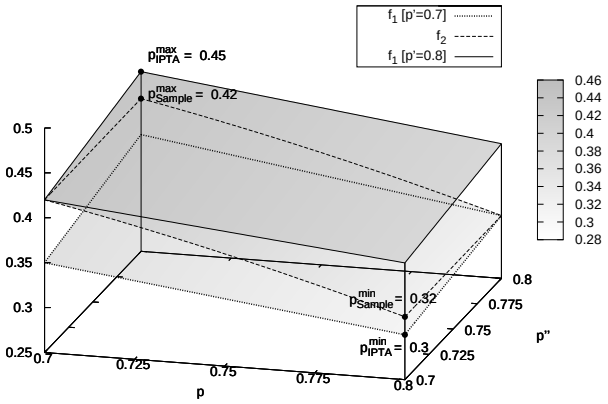

Figure 3: Analytic solutions for the property 'one out of two response is slow'

The plane in the middle represents the solution for the sampling-based `pta` approach, which reaches a minimum probability of 0.32 for y=0.7 and a maximum probability of 0.42 for y=0.8. The upper and lower plane depict the IPTA version which reaches a minimum and maximum probabilities of 0.3 and 0.45, respectively. Therefore, the sampling approach using PTA is not sufficient for determining the correct minimum and maximum probabilities in the original IPTA model.

6.2 Encoding IPTA as PTA

Although the semantics of an interval distribution, i.e., the set of all probability distributions that respect the bounds of its intervals, is in general infinite, it is still possible to encode any finite IPTA into an equivalent, finite PTA. This encoding, which we also refer to as PTA*, works as follows:[3]

- The actions, clocks and locations of the PTA are the same as in the IPTA.
- For every transition $s \xrightarrow{a} \lambda$ in the IPTA and any ordering of the set $Supp(\lambda)$ add the transition $s \xrightarrow{a} \mu_\lambda^{\max}$ to the PTA (cf. Section 4.3).

As an example, Figure 4 depicts the PTA* encoding of the server IPTA in Figure 2. From the construction, it is clear that this encoding preserves probabilistic reachability, i.e., the minimum and maximum probabilities for reaching a set of target states in this PTA is the same as for the original IPTA. However, the number of generated transitions in the PTA is exponential in the size of the support of the transition. Thus, there is a significant blow-up in the size of the model. Even in our simple example in Figure 2 where the support sets have a size of at most 2, the larger number of transitions in the PTA* encoding results in longer run-times of the model checker. To illustrate this, we increased the number of requests performed by the client in our running example and compared the run-times of PRISM.

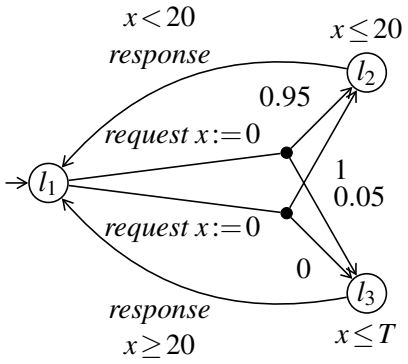

Figure 4: PTA* encoding of the server IPTA

6.3 Comparison of the run-times

Table 1 summarizes the run-times of our IPTA version of PRISM for three different encodings of the running example:

1. PTA: sampling approach where a single probability distribution in the interval distribution is tested;
2. IPTA: the original model as in Listing 2;
3. PTA*: the encoding of the original IPTA using $\mu_\lambda^{\max}$;

The checking of the PTA version was the fastest. However, we have shown above already that such a naive analysis using sampling does not produce the correct results. While the PTA* version yields the correct results, the numbers show that the direct checking of the IPTA is more efficient. This is due to the fact the number of transitions to be checked in PTA* encoding is higher than in the original IPTA. The actual numbers of the transitions in the example are listed in Table 2. Note that in our simple client/server example, the support sets of the transitions are very small (of size 1 or 2). We expect that with a greater branching of transitions, the performance loss using the PTA* encoding gets significantly worse.

[3]The PTA* encoding is similar to the MDP reduction of IMDPs in [16].

#Requests	#States	PTA	IPTA	PTA*
10	235	0.752	0.804	0.816
20	865	2.274	2.625	2.888
30	1,895	7.274	7.818	9.225
40	3,325	19.170	21.662	25.990
50	5,155	43.573	47.908	57.847

Table 1: Runtime in seconds for computing minimum probabilities for 'less than 50% slow responses'

#Requests	PTA	IPTA	PTA*
10	339	339	521
20	1,269	1,269	2,031
30	2,799	2,799	4,541
40	4,929	4,929	8,051
50	7,659	7,659	12,561

Table 2: Number of transitions for different encodings of the client/server example

7 Related work

Probabilistic reachability and expected reachability for PTA based on an integral model of time (digital clocks) is studied in [11]. A zone-based algorithm for symbolic PTCTL [13] model checking of PTA is introduced in [13]. A notion of probabilistic time-abstracting bisimulation for PTA is introduced in [2]. For an overview of tools that support verification of (priced) PTA we refer to the related tools section in [10]. Interval-based probabilistic models and their use for specification and refinement / abstraction have been studied already in '91 in [5]. PCTL model checking of interval Markov chains is introduced in [16]. Symbolic model checking for IPTA is presented in [18] based on the approaches in [13, 16]. However, no tool support or evaluation is given. Moreover, we show here that IPTA can also be encoded into PTA and provide some empirical data for comparing the differences in terms of correctness and run-times of our model checker.

Quality prediction of service compositions based on probabilistic model checking with PRISM is suggested in [4]. A comparison of different QoS models for service-oriented systems and an extension of the UML for quantitative models is given in [7]. A formal syntax for service level agreements of web services can be given using WSLA [8, 3]. A compositional QoS model for channel-based coordination of services is presented in [14].

8 Conclusions

We demonstrated in this paper how the recently introduced model of Interval Probabilistic Timed Automata [18] (IPTA) can be employed to model and verify quality of service guarantees, specifically, probabilistic real-time properties for service-oriented systems with dynamic service binding with contracts specified in service level agreements. We have shown that IPTA can capture the guarantees specified in the SLAs more naturally than PTA. To the best of our knowledge, our extension of the PRISM tool is the first implementation of an IPTA model checker. Moreover, we were able to show that IPTA can be analyzed nearly as fast as sample PTA and faster than a possible encoding of an IPTA in a finite PTA.

As future work, we plan to study refinement notions for IPTA which we hope will enable us to reason compositionally about QoS guarantees of service-oriented systems.

Acknowledgments

The authors of this paper are grateful to Dave Parker for his support with the IPTA implementation in PRISM.

References

[1] D. P. Bertsekas & J. N. Tsitsiklis (1991): *An Analysis of Stochastic Shortest Path Problems*. Mathematics of Operations Research 16(3), pp. 580–595, doi:10.1287/moor.16.3.580.

[2] T. Chen, T. Han & J. P. Katoen (2008): *Time-Abstracting Bisimulation for Probabilistic Timed Automata*. In: *TASE'08*, IEEE Comp. Soc., pp. 177–184, doi:10.1109/TASE.2008.29.

[3] A. Dan, R. Franck, A. Keller, R. King & H. Ludwig (2002): *Web Service Level Agreement (WSLA) Language Specification*. Available at `http://www.research.ibm.com/wsla/documents.html`.

[4] S. Gallotti, C. Ghezzi, R. Mirandola & G. Tamburrelli (2008): *Quality Prediction of Service Compositions through Probabilistic Model Checking*. In: *QoSA'08*, LNCS 5281, Springer, pp. 119–134, doi:10.1007/978-3-540-87879-7_8.

[5] B. Jonsson & K. G. Larsen (1991): *Specification and Refinement of Probabilistic Processes*. In: *LICS'91*, IEEE Comp. Soc., pp. 266–277, doi:10.1109/LICS.1991.151651.

[6] M. Jurdzinski, J. Sproston & F. Laroussinie (2008): *Model Checking Probabilistic Timed Automata with One or Two Clocks*. Log. Meth. in Comp. Sci. 4(3), doi:10.2168/LMCS-4(3:12)2008.

[7] I. Jureta, C. Herssens & S. Faulkner (2009): *A comprehensive quality model for service-oriented systems*. Software Quality Journal 17, pp. 65–98, doi:10.1007/s11219-008-9059-2.

[8] A. Keller & H. Ludwig (2003): *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services*. J. Netw. Syst. Manage. 11, p. 2003, doi:10.1023/A:1022445108617.

[9] J. Kemeny, J. Snell & A. Knapp (1976): *Denumerable Markov Chains*, 2nd edition. Springer.

[10] M. Kwiatkowska, G. Norman & D. Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-time Systems*. In: *CAV'11*, LNCS 6806, Springer, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[11] M. Kwiatkowska, G. Norman, D. Parker & J. Sproston (2006): *Performance Analysis of Probabilistic Timed Automata using Digital Clocks*. Form. Methods Syst. Des. 29, pp. 33–78, doi:10.1007/s10703-006-0005-2.

[12] M. Kwiatkowska, G. Norman, R. Segala & J. Sproston (2002): *Automatic verification of real-time systems with discrete probability distributions*. Theor. Comput. Sci. 282, pp. 101–150, doi:10.1016/S0304-3975(01)00046-9.

[13] M. Kwiatkowska, G. Norman, J. Sproston & F. Wang (2007): *Symbolic model checking for probabilistic timed automata*. Inf. Comput. 205, pp. 1027–1077, doi:10.1016/j.ic.2007.01.004.

[14] Y.-J. Moon, A. Silva, C. Krause & F. Arbab (2011): *A Compositional Model to Reason about end-to-end QoS in Stochastic Reo Connectors*. Science of Computer Programming (to appear) .

[15] *PRISM Case Studies*. `http://www.prismmodelchecker.org/casestudies`.

[16] K. Sen, M. Viswanathan & G. Agha (2006): *Model-Checking Markov Chains in the Presence of Uncertainties*. In: *TACAS'06*, LNCS 3920, Springer, pp. 394–410, doi:10.1007/11691372_26.

[17] *UPPAAL Case Studies*. `http://www.it.uu.se/research/group/darts/uppaal/examples.shtml`.

[18] J. Zhang, J. Zhao, Z. Huang & Z. Cao (2009): *Model Checking Interval Probabilistic Timed Automata*. In: *ICISE'09*, IEEE Comp. Soc., pp. 4936–4940, doi:10.1109/ICISE.2009.749.