Probabilistic Real-Time Rewrite Theories and Their Expressive Power

Lucian Bentea and Peter Csaba Ölveczky

Department of Informatics, University of Oslo

Abstract. Unbounded data structures, advanced data types, and/or different forms of communication are often needed to model large and complex probabilistic real-time systems such as wireless sensor networks. Furthermore, it is often natural to model such systems in an object-oriented style, using subclass inheritance and dynamic object and message creation and deletion. To support the above features, we introduce probabilistic real-time rewrite theories (PRTRTs), that extend both real-time rewrite theories and probabilistic rewrite theories, as a rewriting-logic-based formalism for probabilistic real-time systems. We then show that PRTRTs can be seen as a unifying model in which a range of other models for probabilistic real-time systems—including probabilistic timed automata, stochastic automata, deterministic and stochastic Petri nets, as well as two probabilistic timed transition system models with under-specified probability distributions—can naturally be represented.

1 Introduction

In this paper we introduce *probabilistic real-time rewrite theories* (PRTRTs) to support the formal specification of probabilistic real-time systems in rewriting logic [18]. Rewriting logic is a logic for concurrent systems that emphasizes expressiveness and ease of specification over algorithmic decidability of key properties. In rewriting logic, the state space and data types of a system are defined by an algebraic equational specification, and the system's transitions are defined by labeled conditional rewrite rules $l : t \longrightarrow t'$ **if** *cond*, where t and t' are terms that may contain universally quantified mathematical variables. Rewriting logic supports the specification of any computable data type, and distributed systems can be naturally modeled in an object-oriented style, with class inheritance and dynamic creation and deletion of objects and messages. Simulation, reachability analysis, and LTL model checking for rewriting logic is provided by the high-performance Maude tool [8]. (Since properties are in general undecidable, such analyses may not always terminate.)

The Real-Time Maude tool [23] and its real-time rewrite theory formalism [22] extend rewriting logic and Maude to the formal modeling and analysis of real-time systems. Its expressiveness has made it possible to apply the tool to several large applications (see [21]) that are beyond the scope of most model checkers for real-time systems. However, some of those applications, including the LMST

U. Fahrenberg and S. Tripakis (Eds.): FORMATS 2011, LNCS 6919, pp. 60–79, 2011.

and OGDC wireless sensor network algorithms [13,25] and the AER/NCA and NORM multicast protocols [24,17], include probabilistic features—e.g., nodes may exhibit random behavior by design to break symmetries in a network, or the environment may interact with the system in a probabilistic manner—that can only be treated in an *ad hoc* way in Real-Time Maude.

Rewriting logic has also been extended to *probabilistic rewrite theories* to specify probabilistic behaviors [14]. Such theories combine nondeterministic and probabilistic behaviors, and the main idea is that the variables in the righthand side t' of a rewrite rule that do not occur in the lefthand side t are instantiated probabilistically. The VeStA tool [26] can be used for both statistical model checking and estimating numerical values in such theories, and has been used to analyze a DoS resistant TCP/IP protocol [1] and a model of the above mentioned LMST algorithm [13] in which its real-time behavior is treated in an *ad hoc* way.

PRTRTs can be seen as an extension of both real-time rewrite theories and probabilistic rewrite theories. However, PRTRTs are a *proper* extension of probabilistic rewrite theories even when time is ignored. In our case, the new variables in the righthand side of a rule are divided into *nondeterministic* and *probabilistic* variables. Each rewrite rule is equipped with a *family* of probability distributions used to instantiate the probabilistic variables; namely, there is one probability distribution for each substitution of the variables in the lefthand side of the rule *and* each choice of values for the nondeterministic variables. Regarding time, although the duration of/between events may be given probabilistically, time itself does not advance in a probabilistic way in our formalism.

After giving some background into rewriting logic and its probabilistic and real-time extensions (Section 2), we define our formalism and its semantics (Section 3) and the probability of reaching a certain state in a certain time (Section 5). We then demonstrate the expressiveness of PRTRTs—as well as their suitability as a *unifying semantic framework* for probabilistic real-time systems in which different models for such systems can naturally be represented and understood—by showing how a range of well known formal models for probabilistic real-time systems can be seen as PRTRTS (Section 6). Due to lack of space, we refer to the longer technical report [6] for formal proofs of correctness for the representations. To illustrate our formalism, we describe how a simple probabilistic round trip time protocol can be modeled as a PRTRT (Section 4). This system cannot be modeled as an automaton, since the number of messages in the state can grow beyond any bound.

We also explain in the accompanying report [6] how the state-of-the-art OGDC algorithm [30] for wireless sensor networks can be defined as a PRTRT. A main feature of the algorithm is that a sensor node becomes *active* depending on how close it is to an "ideal" position w.r.t. the already active nodes, and that it turns itself off when its sensing area is covered by the sensing areas of other active nodes. The OGDC algorithm therefore requires computing with data types for sensing areas and sophisticated functions including distances, angles, computing overlaps of areas, etc., which seem to be beyond the capability of formalisms that do not support the definition of new data types and advanced functions.

2 Preliminaries

In rewriting logic [18], the static parts of a system (functions, data types, etc.) are defined as an algebraic equational specification, and the transitions of a system are specified by labeled rewrite rules of the form $l : t \longrightarrow t'$ **if** *cond*, where t and t' are *terms* constructed by typed variables and function symbols in a type-consistent way, l is a rule label, and *cond* is a (possibly empty) conjunction of equalities, sort memberships, and rewrites. Such a rule specifies a local transition from an instance of the term t to the corresponding instance of the term t', provided that the condition *cond* is satisfied by the substitution instance.

Formally, given a set K of *kinds*, a *many-kinded signature* σ contains a set of function declarations of the form $f : k_1 \dots k_n \to k$, where $n \geq 0$ and $k_1, \dots, k_n, k \in K$. In *membership equational logic* (MEL) [19], each kind k has an associated set of sorts S_k. A *MEL theory* consists of a MEL signature $\Sigma = (K, \sigma, \{S_k \mid k \in K\})$ and a set E of (possibly) conditional equations $(\forall \vec{x})\ \ t = t'$ **if** *cond* and membership axioms $(\forall \vec{x})\ \ t : s$ **if** *cond*, where t and t' are Σ-terms of the same kind k, s is a sort of kind k, *cond* is a conjunction of equalities and sort memberships, and $\vec{x}$ denotes the set of variables in these axioms. We write $vars(t)$ for the set of variables occurring in a term t; if $vars(t) = \emptyset$, then t is called a *ground term*. If $(\Sigma, E \cup A)$ is a MEL theory, where A is a collection of *structural axioms* specifying properties of function symbols, like commutativity, associativity, etc., and E is terminating, confluent and sort-decreasing modulo A, then $Can_{\Sigma,E/A}$ denotes the algebra of fully simplified ground terms, or "normal forms," with respect to the set of axioms E, modulo A. We denote by $[t]_A$ the A-equivalence class of a fully simplified term t. An *E/A-canonical ground substitution* for a set of variables $\vec{x}$ is a function $[\theta]_A : \vec{x} \to Can_{\Sigma,E/A}$ that assigns a fully simplified ground term to each variable in $\vec{x}$. We denote by $CanGSubst_{E/A}(\vec{x})$ the set of all such functions. We use the same notation $[\theta]_A$ for the homomorphic extension of $[\theta]_A$ to Σ-terms.

Definition 1. *A* generalized rewrite theory *[7] is a tuple $\mathcal{R} = (\Sigma, \varphi, E, L, R)$, where Σ is a MEL signature, φ is a function that maps each function symbol $f : k_1 \dots k_n \to k$ in Σ its* frozen argument positions *$\varphi(f) \subseteq \{1, \dots, n\}$, (Σ, E) is a MEL theory, and R is a set of labeled conditional rewrite rules*

$$(\forall \vec{x})\ \ l : t \longrightarrow t' \ \ \textbf{if} \ \ cond, \tag{1}$$

where $l \in L$ is a label, t and t' are terms of the same kind, cond is a conjunction of equalities, memberships and rewrites, and $\vec{x} = vars(t) \cup vars(t') \cup vars(cond)$.

Intuitively, if i is a frozen position of a function symbol f, i.e., if $i \in \varphi(f)$, then $f(\dots, t_i, \dots)$ does *not* rewrite to $f(\dots, t_i', \dots)$ when t_i rewrites to t_i'. A *context* is a Σ-term $\mathbb{C}$ with a single occurrence of a single variable, denoted $\odot$ and called the *hole*. Two contexts $\mathbb{C}$ and $\mathbb{C}'$ are A-equivalent if $A \vdash (\forall \odot)\ \mathbb{C}(\odot) = \mathbb{C}'(\odot)$. A context $f(t_1, \dots, t_n)$ has a hole in a frozen position if the hole occurs in some argument t_i, and either $i \in \varphi(f)$ or t_i has a hole in a frozen position.

Let Ω be a nonempty set. If Ω is countable, a *probability mass function*, or *probability distribution* over Ω is any mapping $p : \Omega \to [0, 1]$ with the property

that $\sum_{\omega \in \Omega} p(\omega) = 1$; we denote by $Dist(\Omega)$ the set of all probability distributions over the set Ω. A *cumulative distribution function* (CDF) is a function $\varphi : \mathbb{R} \to [0,1]$ defining the probability $\varphi(x)$ that a real-valued *random variable* is less than $x \in \mathbb{R}$; we write $Dist(\mathbb{R})$ for the set of all CDFs.

In [14] rewrite theories are extended to *probabilistic rewrite theories*. Intuitively, in such theories the righthand side t' of a rewrite rule $l : t \longrightarrow t'$ **if** *cond* may contain variables $\vec{p}$ that do not occur in t. These new variables are assigned values according to a probability distribution taken from a *family* of probability distributions—one for each instance of the variables in t—associated with the rule. Formally, a probabilistic rewrite theory is a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is a rewrite theory[1] and π is a function which assigns to each rule $r \in R$ of the form (1), with $vars(t) = \vec{x}$ and $vars(t') \setminus vars(t) = \vec{p}$, a mapping[2]

$$\pi_r : [\![cond(\vec{x})]\!] \to Dist\left(CanGSubst_{E/A}(\vec{p}) \right),$$

where $[\![cond(\vec{x})]\!]$ is the set of all E/A-canonical ground substitutions for $\vec{x}$ that satisfy the condition *cond*. That is, for each substitution θ of the variables in t which satisfies *cond*, we get a probability distribution $\pi_r([\theta]_A)$ that defines how the new variables $\vec{p}$ are instantiated. A rewrite rule $r \in R$ of the form (1) with $vars(t') \setminus vars(t) \neq \emptyset$, together with its associated probability distribution function π_r is called a *probabilistic rewrite rule* and is written $l : t \longrightarrow t'$ **if** *cond* **with probability** π_r.

In [22], rewrite theories are extended to real-time systems by *(i)* considering ordinary rewrite rules to be *instantaneous* transitions, and *(ii)* by adding *tick rewrite rules* that model time elapse in a system. Formally, a *real-time rewrite theory* [22] is a pair $(\mathcal{R}, \tau)$ where $\mathcal{R}$ is a generalized rewrite theory and τ is an assignment of a *duration* term τ_l of sort `Time` to rewrite rules of the form $l : \{t\} \longrightarrow \{t'\}$ **if** *cond*, where `{_}` encloses the entire state. The term τ_l specifies the amount of time that elapses with the application of the rewrite rule. If $\tau_l \neq 0$, the rule is called a *tick rewrite rule* and is written $l : \{t\} \xrightarrow{\tau_l} \{t'\}$ **if** *cond*.

We also use the Maude [8] syntax to specify rewrite rules, so that a conditional tick rule with duration y is written `crl [`l`]: {`t`} => {`t'`} in time` y `if` *cond*, where the label l may be omitted. In object-oriented Maude specifications [8], the state of the system is a term of sort `Configuration` denoting a multiset of objects and messages, where multiset union is denoted by juxtaposition. Each object is represented as a term `<` o `:` c `|` att_1 `:` val_1, ..., att_n `:` val_n `>`, where o is the object's identifier of sort `Oid`, c is the object's class, and where $val_1, \ldots, val_n$ are the values of the object's attributes $att_1, \ldots, att_n$. For example, the rule

```
rl [l]: m(O, w)  < O : C | a1 : x, a2 : O', a3 : z >   =>
                 < O : C | a1 : x + w, a2 : O', a3 : z >  dly(m'(O'), x) .
```

[1] A (standard) rewrite theory is a generalized rewrite theory with $\varphi(f) = \emptyset$ for all its function symbols f, i.e., a rewrite theory with no frozen operators.

[2] In [14] the definition of probabilistic rewrite theories is based on the more general case of *probability measures* on a σ-algebra over $CanGSubst_{E/A}(\vec{p})$. However, to simplify the exposition, we only consider probability mass functions. Extending our definitions to probability measures is straightforward, as shown in our report [6].

defines a family of transitions in which a message `m`, with parameters `O` and `w`, is read and consumed by an object `O` of class `C`. The transitions change the attribute `a1` of `O` and send a new message `m'(O')` with delay `x`. "Irrelevant" attributes (such as `a3` and the righthand side occurrence of `a2`) need not be mentioned.

3 Probabilistic Real-Time Rewrite Theories

This section defines probabilistic real-time rewrite theories (PRTRTs) and their semantics. PRTRTs extend both probabilistic rewrite theories and real-time rewrite theories to support the formal specification of real-time systems with probabilistic features. The definitions in this section are mostly extensions of similar definitions in [14] for (untimed) probabilistic rewrite theories.

Definition 2. *A* probabilistic real-time rewrite theory *(PRTRT) $\mathcal{R}_{\pi,\tau}$ is a tuple $(\mathcal{R}, \pi, \tau)$, where $\mathcal{R} = (\Sigma, \varphi, E \cup A, L, R)$ is a generalized rewrite theory in which the rules in R have no rewrites in their conditions, $(\mathcal{R}, \tau)$ is a real-time rewrite theory, and π is a function that takes each rewrite rule $r \in R$ of the form (1), with $vars(t) = \vec{x}$, $vars(t') \setminus vars(t) = \vec{y} \uplus \vec{p}$, and assigns to it a mapping*

$$\pi_r : [\![cond(\vec{x} \cup \vec{y})]\!] \to Dist\left(CanGSubst_{E/A}(\vec{p})\right)$$

such that, for each substitution $[\theta]_A \in CanGSubst_{E/A}(\vec{x} \cup \vec{y})$ that satisfies the condition cond, $\pi_r([\theta]_A)$ is a probability mass function over the set of ground substitutions $CanGSubst_{E/A}(\vec{p})$. If r is a tick rule, then $\vec{p} \cap vars(\tau_l) = \emptyset$. Probabilistic tick rewrite rules are written $l : \{t\} \xrightarrow{\tau_l} \{t'\}$ **if** *cond* **with probability** π_r.

Although the duration of/between events may be given probabilistically, time itself does not advance in a probabilistic way. The duration term τ_l therefore does not contain variables that are substituted probabilistically, hence $\vec{p} \cap vars(\tau_l) = \emptyset$. Apart from adding timed behaviors, PRTRTs also extend probabilistic rewrite theories in two ways necessitated by the way tick rules are usually defined:

i) PRTRTs allow new variables that are not assigned a probability distribution in the righthand side of a rewrite rule. The reason is that tick rules—in particular for dense time domains—allow time to advance by *any* amount less than a certain bound; they therefore have a new (non-probabilistic) variable in their righthand sides that defines the duration of the rewrite.
ii) The tick rules involve functions on the state, such as a function defining the effect of time elapse on a state, that are *frozen* operators; we therefore have used *generalized* rewrite theories as the underlying formalism.

Consider the following probabilistic tick rule r (written in an intuitive way):

$$\{f(x)\} \xrightarrow{y} \{g(x, y, z_1, z_2)\} \ \textbf{if} \ y \leq 10$$
$$\textbf{with probability} \ z_1 := \begin{pmatrix} h(x,y) & f(y) \\ 2/3 & 1/3 \end{pmatrix} \ \textbf{and} \ z_2 := \begin{pmatrix} 0 & 1 \\ 1/2 & 1/2 \end{pmatrix}.$$

The righthand side term $\{g(x, y, z_1, z_2)\}$ contains variables y, z_1 and z_2 that do not occur in the rule's lefthand side $\{f(x)\}$. Let $\{f(t)\}$ be the state of the system when the rule is applied. The variable y is then instantiated *nondeterministically* with any value t' less than or equal to 10. The variables z_1 and z_2 are then instantiated probabilistically, where z_1 is assigned the value $[h(t, t')]_A$ with probability $2/3$ and the value $[f(t')]_A$ with probability $1/3$. Formally, the mapping $\pi_r : \mathit{CanGSubst}_{E/A}(\{x, y\}) \to \mathit{CanGSubst}_{E/A}(\{z_1, z_2\})$ associated to the above rule r is given by $\pi_r([\theta]_A)(\{z_1 \mapsto [h(\theta(x), \theta(y))]_A,\ z_2 \mapsto i\}) = 1/3$ and $\pi_r([\theta]_A)(\{z_1 \mapsto [f(\theta(y))]_A,\ z_2 \mapsto i\}) = 1/6$, for $i \in \{[0]_A, [1]_A\}$.

Let $\mathcal{R}_{\pi,\tau} = (\Sigma, \varphi, E \cup A, L, R, \pi, \tau)$ be a PRTRT. Intuitively, an R/A-match contains the complete information on how and in which context the current system state is matched against a particular rewrite rule in the specification of that system. We extend the definition of R/A-matches in [14] as follows:

Definition 3. *Given a fully simplified term $[u]_A \in \mathit{Can}_{\Sigma,E/A}$, its* generalized R/A-matches *are triples $([\mathbb{C}]_A, r, [\theta]_A)$ where $\mathbb{C}$ is a context whose hole is not in a frozen position, $r \in R$ is a rewrite rule, $[\theta]_A \in \mathit{CanGSubst}_{E/A}(\vec{x} \cup \vec{y})$ is a substitution such that $E \cup A \vdash \theta(\mathit{cond})$, and $[u]_A = [\mathbb{C}(\odot \leftarrow \theta(t))]_A$ is the A-equivalence class of the term obtained by applying the substitution θ to t and placing the result into $\mathbb{C}$.*

The definition of a single (instantaneous and tick) transition of a PRTRT describes how the system state evolves when applying a rewrite rule to it:

Definition 4. *Given terms $[u]_A, [v]_A \in \mathit{Can}_{\Sigma,E/A}$, an* E/A-canonical one-step rewrite *from $[u]_A$ to $[v]_A$ is a labelled transition $[u]_A \xrightarrow[\tau]{([\mathbb{C}]_A, r, [\theta]_A, [\rho]_A)} [v]_A$, where $([\mathbb{C}]_A, r, [\theta]_A)$ is a generalized R/A-match for $[u]_A$ selected nondeterministically, $[\rho]_A \in \mathit{CanGSubst}_{E/A}(\vec{p})$ is a substitution selected with probability $\pi_r([\theta]_A)([\rho]_A)$, the duration τ is 0 if r is not a tick rule and is $\theta(\tau_l)$ otherwise, and $[v]_A = [\mathbb{C}(\odot \leftarrow t'(\theta(\vec{x}, \vec{y}), \rho(\vec{p})))]_A$ is the result of the one-step rewrite.*

4 Example: A Simple Round Trip Time Protocol

To illustrate our formalism, we specify in an object-oriented way a simple *round trip time* (RTT) protocol that computes the time it takes for a message to go from one node to another, and back, and where the message transmission time follows a probability distribution that depends on the distance between the nodes.

The initiator object `O` starts the protocol by sending an `rttReq` message to its neighbor `O'`, with a time stamp `T` which is the current value of `O`'s local clock (rule `start`). When `O'` receives this message, it immediately sends back a reply to `O` with the original time stamp with probability $3/4$ and ignores the request with probability $1/4$ (rule `rttResp`). When the initiator `O` receives the reply, it computes its RTT value w.r.t. `O'` by subtracting the original time stamp `T` from its current clock value `T'` (rule `treatResp`). However, if the message takes so long that `T' - T` $\geq$ `MAX_RTT`, then it is just ignored (rule `ignoreOld`). The

initiator uses a retransmission timer to start a new round of the protocol every MAX_RTT time units until it has computed a good RTT value. When the timer expires, O sends another RTT request to O' (rule tryAgain) with probability $1/(\mathtt{N}+1)$, which decreases with the number N of unresolved RTT requests of O. We represent each node by an object

< o : Node | nbr : o', rtt : r, clock : t, timer : ti, tries : n >

where o is the node's identifier, o' is the neighbor to which o wants to compute its round trip time, r is the value of the round trip time, if computed, or INF otherwise, t is the current value of the node's clock, ti is its current timer value, which has the value INF if the timer is switched off, and n is the number of unsuccessful attempts that o has made to compute the RTT. Messages have the form findRtt(o), which triggers a run of the RTT protocol for node o, rttReq(o', o, t), which sends a request from node o to node o' with t the time stamp, and rttResp(o, o', t), which sends a reply message from node o' to node o with the original time stamp t. We assume that a function dist that computes the distance dist(o, o') between two nodes is defined. In the rules start, rttResp and tryAgain we specify the transmission delay of the rttReq and rttResp messages as a variable D which is probabilistically substituted according to a probability distribution F(x) that mimics a truncated normal distribution $\mathcal{N}(\mu, \sigma^2)$ [12] with minimum value MIN_DELAY, and depends on the distance x between o and o', where μ and σ are positive constants representing the average and the standard deviation of the transmission delay, respectively.

The following instantaneous rewrite rules describe our simple RTT protocol.

```
vars O O': Oid. vars T T' D: Time. var N: Nat. var B: Bool. var CF: Configuration.

rl [start] :
   findRtt(O)  < O : Node | clock : T, nbr : O' >
 =>  < O : Node | timer : MAX_RTT >  dly(rttReq(O', O, T), D)
     with probability D := F(dist(O, O')) .

rl [rttResp] :
   rttReq(O, O', T)  < O : Node | >
 =>  if B then < O : Node | >  dly(rttResp(O', O, T), D)  else < O : Node | >  fi
     with probability B := ( true  false )  and D := F(dist(O, O')) .
                          (  3/4   1/4  )

crl [treatResp] :
    rttResp(O, O', T)  < O : Node | clock : T' >
 =>  < O : Node | rtt : T' - T, timer : INF >  if T' - T < MAX_RTT .

crl [ignoreOld] :
    rttResp(O, O', T)  < O : Node | clock : T' >  =>  < O : Node | >  if T' - T >= MAX_RTT.

rl [tryAgain] :
    < O : Node | timer : 0, clock : T, nbr : O', tries : N >
 =>  if B then  < O : Node | timer : MAX_RTT, tries : N + 1 >  dly(rttReq(O, O', T), D)
     else < O : Node | timer : MAX_RTT >  fi
     with probability B := (   true      false   )  and D := F(dist(O, O')) .
                          ( 1/(N+1)  N/(N+1) )
```

Time elapse is modeled by the tick rule

```
crl [tick] : {CF}  =>  {delta(CF, T)} in time T  if T <= mte(CF) .
```

where `delta` is a frozen function that specifies the effect of time elapse on the system by decreasing the timers and increasing the clock values of each node. The frozen function `mte` gives the maximum amount of time that can elapse before a node must perform an instantaneous transition. More precisely, time cannot advance past the expiration of a timer or the moment when a message arrives. See [6] for their formal definition.

It is worth noticing that the number of messages in the state can grow beyond any bound, since: *i)* the message delays could be arbitrarily large (with non-zero probability), and *ii)* the initiator node will periodically send requests until it receives a good RTT value. Therefore, even this simple protocol seems to be beyond the scope of automaton-based formalisms.

5 Reachability Probabilities in PRTRTs

In this section we define the probability of reaching a given state in a certain time in a PRTRT. PRTRTs combine probabilistic and nondeterministic behaviors, and we must therefore assign "probabilities" also to the nondeterministic choices to define the probability of reaching a state t_2 from a state t_1 in time τ. This is done by "adversaries," so that the probability of reaching t_2 in time τ is defined relative to a given adversary.

A *computation* of a PRTRT $\mathcal{R}_{\pi,\tau}$ is an infinite sequence of E/A-canonical rewrite steps, with zero-time self-loops from deadlock states. Formally, a computation of $\mathcal{R}_{\pi,\tau}$ is an *infinite* sequence $\Pi = [u_1]_A \xrightarrow[\tau_1]{\alpha_1} [u_2]_A \xrightarrow[\tau_2]{\alpha_2} \cdots$ where $[u_i]_A \xrightarrow[\tau_i]{\alpha_i} [u_{i+1}]_A$ is either a E/A-canonical one-step rewrite, or $[u_i]_A$ cannot be rewritten using the rules in $\mathcal{R}_{\pi,\tau}$, in which case $[u_i]_A = [u_{i+1}]_A$, $\alpha_i = !$, and $\tau_i = 0$, where '$!$' is a new label. To each computation Π we associate the *infinite timed computation path* obtained by removing the labels above the transition arrows. A *finite timed computation path* is a prefix of an infinite timed computation path. We denote by $\Pi_n = [u_1]_A \xrightarrow[\tau_1]{} \cdots \xrightarrow[\tau_{n-1}]{} [u_n]_A$ a finite timed computation path. The nondeterministic choices of the generalized R/A-matches $([\mathbb{C}]_A, r, [\theta]_A)$ for a state $[u_k]_A$ prohibit us from defining the probability of reaching $[u_n]_A$ in a certain time. Therefore, all nondeterministic choices must be resolved by means of an *adversary*, which extends the notion of adversary in [14] as follows:

Definition 5. *An R/A-*match adversary *of a PRTRT $\mathcal{R}_{\pi,\tau}$ is a function $\mathcal{A}$ that maps each finite timed computation path Π_n of $\mathcal{R}_{\pi,\tau}$, ending with a term $[u_n]_A$, to a probability distribution on the set of generalized R/A-matches for $[u_n]_A$.*

Given an R/A-match adversary $\mathcal{A}$, the following formula gives the probability of performing a rewrite from a "non-deadlock" term $[u_n]_A$ to a term $[u']_A$ in one step in time τ, provided that $[u_n]_A$ is obtained via Π_n,

$$\mathbb{P}_{\mathcal{A}}\left([u_n]_A \xrightarrow[\tau]{} [u']_A \mid \Pi_n\right) = \sum \Big[\mathcal{A}(\Pi_n)\,([\mathbb{C}]_A, r, [\theta]_A) \cdot \pi_r([\theta]_A)\,([\rho]_A)\Big],$$

where the sum ranges over all $[u_n]_A \xrightarrow[\tau]{([\mathbb{C}]_A, r, [\theta]_A, [\rho]_A)} [u']_A$. If $[u_n]_A$ is a deadlock state then $\mathbb{P}_{\mathcal{A}}\left([u_n]_A \underset{\tau}{\longrightarrow} [u']_A \mid \Pi_n\right) = 1$ if and only if $[u']_A = [u_n]_A$ and $\tau = 0$, and is 0 otherwise. Also, the probability of the finite timed computation path Π_n to occur in a PRTRT is given by:

$$\mathbb{P}_{\mathcal{A}}(\Pi_n) = \prod_{i=1}^{n-1} \mathbb{P}_{\mathcal{A}}\left([u_i]_A \underset{\tau_i}{\longrightarrow} [u_{i+1}]_A \;\middle|\; [u_1]_A \underset{\tau_1}{\longrightarrow} \cdots \underset{\tau_{i-1}}{\longrightarrow} [u_i]_A\right).$$

The probability of reaching state $[u']_A$ from state $[u]_A$ in time τ is given by $\sum_{\Pi} \mathbb{P}_{\mathcal{A}}(\Pi)$, where Π ranges over all finite timed computation paths Π_n with $[u_n]_A = [u']_A$ and $\sum_{i=1}^{n-1} \tau_i = \tau$, and such that there is no $j < n$ with $[u_j]_A = [u']_A$ and $\sum_{i=1}^{j-1} \tau_i = \tau$.

6 The Expressive Power of PRTRTs

In this section we show the expressiveness of PRTRTs—and its suitability as a unifying semantic framework for probabilistic real-time systems in which different models of such systems can be represented—by explaining how a range of models of probabilistic real-time systems can naturally be seen as PRTRTs. More details about the mappings and their correctness proofs are given in [6].

Since probabilistic rewrite theories are a proper subclass of PRTRTs, any probabilistic rewrite theory can also be represented as a PRTRT. In [14] mappings are provided from probabilistic nondeterministic systems, generalized semi-Markov processes, and continuous-time Markov chains into probabilistic rewrite theories. That paper also claims that the same method can be used for representing the PEPA [11] language and various Petri net formalisms, such as stochastic reward nets, generalized stochastic Petri nets [3], and stochastic Petri nets with generally distributed firing times, as probabilistic rewrite theories.

6.1 Probabilistic Timed Automata

The *probabilistic timed automaton* (PTA) model [27] combines nondeterministic and probabilistic behaviors, and extends timed automata [5] by allowing a probabilistic choice of both the *next state* and the *set of clocks* to be reset in a "transition." PTA are supported by the probabilistic model checker PRISM [15].

A *clock* is a variable ranging over the real numbers that increases its value according to the elapsed time. A *zone* of a set of clocks $\mathcal{X}$ is a convex subset of $\mathbb{R}^{|\mathcal{X}|}$ defined by a conjunction of constraints over $\mathcal{X}$. Let $Z_{\mathcal{X}}$ be the set of all zones of $\mathcal{X}$. A PTA is then a tuple $(S, s_0, \mathcal{X}, \text{inv}, \text{prob}, \{\tau_s\}_{s \in S})$, where: S is a finite set of *states* with $s_0 \in S$ the *start* state; $\mathcal{X}$ is a finite set of *clocks*; $\text{inv} : S \to Z_X$ is a function that assigns an *invariant* to each state; $\text{prob} : S \to \mathcal{P}(Dist\,(S \times \mathcal{P}(\mathcal{X})))$ is a function that *assigns a set of probability distributions* on $S \times \mathcal{P}(\mathcal{X})$ to each state; and $\{\tau_s\}_{s \in S}$ is a family of functions where, for each $s \in S$, $\tau_s : \text{prob}(s) \to Z_{\mathcal{X}}$ assigns an *enabling condition* to each $p \in \text{prob}(s)$.

A PTA in state s may *nondeterministically* select any enabled probability distribution p in prob(s). The probability that the automaton then makes a transition to state s' and resets all the clocks $X \subseteq \mathcal{X}$ to 0 is $p(s', X)$. Following [16,27], we assume that the enabling condition $\tau_s(p)$ implies the invariant of all possible successor states s' with $p(s', X) > 0$, *after* the clocks in X are reset.

Figure 1 (a) shows a PTA that starts in state s_0 with its clock x initialized to 0. When $x \in [5, 7]$ in state s_0 the automaton can nondeterministically choose between the two probability distributions $\pi_1 = \begin{pmatrix} (s_1, \emptyset) & (s_2, \emptyset) \\ 0.3 & 0.7 \end{pmatrix}$ and $\pi_2 = \begin{pmatrix} (s_2, \emptyset) \\ 1 \end{pmatrix}$, and when $x \in [3, 8] \setminus [5, 7]$ it can only take π_2. A probabilistic choice is then made according to the selected probability distribution, and the corresponding transition is performed. Likewise, the probabilistic choice from s_1 is made by sampling from the probability distribution $\pi_3 = \begin{pmatrix} (s_1, \emptyset) & (s_2, \{x\}) \\ 0.5 & 0.5 \end{pmatrix}$; if the automaton makes a transition to state s_2, which happens with probability 0.5, then x is reset to 0.

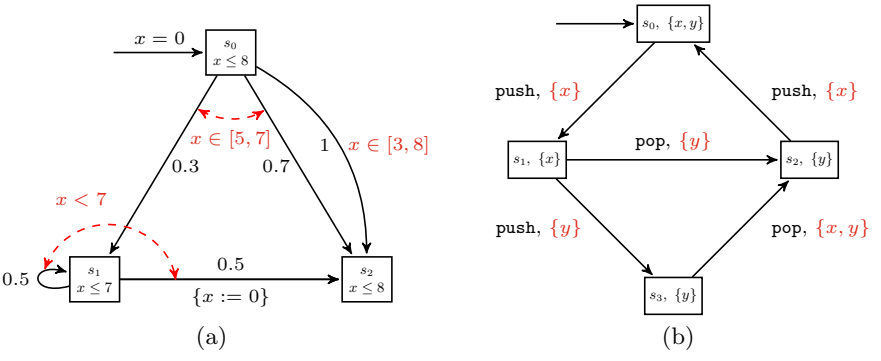

Fig. 1. (a) A probabilistic timed automaton (b) A stochastic automaton

A PTA $A = (S, s_0, \mathcal{X}, \text{inv}, \text{prob}, \langle \tau_s \rangle_{s \in S})$ with $\mathcal{X} = \{x_1, \ldots, x_n\}$ is represented as a PRTRT $\Psi_{PTA}(A)$ as follows (see [6] for more details). A "timed state" of A is represented as a term $\{s, r_1, r_2, \ldots, r_n\}$, with r_i denoting the current value of clock x_i. To each state $s \in S$ and each probability distribution $\pi : S \times \mathcal{P}(\mathcal{X}) \to [0, 1]$ in prob(s), we associate a probabilistic rewrite rule

```
crl [π]: {s, y1, ..., yn} => σ if (y1, ..., yn) ∈ τs(π) with probability σ := Γs(π)
```

where σ and the y_i are variables, and $\Gamma_s(\pi) : \mathit{CanGSubst}_{E/A}(\sigma) \to [0, 1]$ is a probability distribution over the set of E/A-canonical ground substitutions for σ defined by $\Gamma_s(\pi)(\sigma \mapsto \{s', t_1, \ldots, t_n\}) = \pi(s', X)$ for all $s' \in S$ and all $X \subseteq \mathcal{X}$, where t_j is 0 if $x_j \in X$ and y_j otherwise. A tick rule

```
crl [tick s]:
    {s, y1, ..., yn} => {s, y1 + y, ..., yn + y} in time y if (y1 + y, ..., yn + y) ∈ inv(s)
```

models time elapse for each state $s \in S$. Since $\tau_s(\pi)$ and $\text{inv}(s)$ are zones defined by conjunctions of inequality constraints over the clock values, the set memberships $(y_1, y_2, \ldots, y_n) \in \tau_s(\pi)$ and $(y_1 + y, y_2 + y, \ldots, y_n + y) \in \text{inv}(s)$ are actually translated into standard inequalities in $\Psi_{PTA}(G)$, as shown below.

The probabilistic timed automaton in Fig. 1 (a) is therefore represented by a PRTRT containing the following set of conditional tick rules

```
crl [tick s0]: {s0, x} => {s0, x + y} in time y if x + y <= 8 .
crl [tick s1]: {s1, x} => {s1, x + y} in time y if x + y <= 7 .
crl [tick s2]: {s2, x} => {s2, x + y} in time y if x + y <= 8 .
```

as well as the following instantaneous probabilistic rewrite rules

$$\texttt{crl [}\pi_1\texttt{]: \{}s_0\texttt{, }x\texttt{\} => }\sigma\texttt{ if }x\texttt{ >= 5 and }x\texttt{ <= 7 with probability }\sigma\texttt{ := }\begin{pmatrix}\texttt{\{}s_1\texttt{, }x\texttt{\}} & \texttt{\{}s_2\texttt{, }x\texttt{\}} \\ \texttt{0.3} & \texttt{0.7}\end{pmatrix}.$$

$$\texttt{crl [}\pi_2\texttt{]: \{}s_0\texttt{, }x\texttt{\} => }\sigma\texttt{ if }x\texttt{ >= 3 and }x\texttt{ <= 8 with probability }\sigma\texttt{ := }\begin{pmatrix}\texttt{\{}s_2\texttt{, }x\texttt{\}} \\ \texttt{1.0}\end{pmatrix}.$$

$$\texttt{crl [}\pi_3\texttt{]: \{}s_1\texttt{, }x\texttt{\} => }\sigma\texttt{ if }x\texttt{ < 7 with probability }\sigma\texttt{ := }\begin{pmatrix}\texttt{\{}s_1\texttt{, }x\texttt{\}} & \texttt{\{}s_2\texttt{, 0\}} \\ \texttt{0.5} & \texttt{0.5}\end{pmatrix}.$$

where σ, x and y are variables and the initial state is given by the term $\texttt{\{}s_0\texttt{, 0\}}$.

6.2 Stochastic Automata

A *stochastic automaton* (SA) [9] is an automaton where the transitions have the form $s \xrightarrow{a,X} s'$, with a an action and X a set of *timers*. A transition is enabled when all the timers in X have expired, and time cannot advance when there is an enabled transition. An SA may also have nondeterministic behaviors, since multiple transitions may become enabled at the same time. As the result of taking the transition $s \xrightarrow{a,X} s'$, each timer x which should be reset when arriving at s' is assigned a value sampled from its cumulative distribution function $F(x)$.

Figure 1 (b) shows an SA. Each state has a set of timers that have to be set when arriving at that state. The SA starts in state s_0 and uses the CDFs $F(x)$ and $F(y)$ to assign initial values to the timers x and y. The automaton makes a (`push`) transition to state s_1 as soon as the timer x expires. In the new state s_1, the timer x is assigned a new value sampled from the CDF $F(x)$. At this point, the SA makes a nondeterministic choice between the two possible transitions when the timer y expires, to s_2 with a `pop` action, or to s_3 with a `push` action.

Let $\mathcal{P}_{\text{fin}}(X)$ denote the set of *finite* subsets of a set X. A stochastic automaton is a tuple $(S, s_0, \mathcal{X}, \text{Act}, \longrightarrow, \kappa, F)$ where: S is a set of *states* and $s_0 \in S$ is the *initial state*; $\mathcal{X}$ is a set of *timers*; Act is a set of *actions*; $\longrightarrow \subseteq S \times (\text{Act} \times \mathcal{P}_{\text{fin}}(\mathcal{X})) \times S$ is the set of *transitions*, where we write $s \xrightarrow{a,X} s'$ iff $(s, a, X, s') \in \longrightarrow$; the function $\kappa : S \to \mathcal{P}_{\text{fin}}(\mathcal{X})$ is the *clock setting function*; and $F : \mathcal{X} \to \text{Dist}(\mathbb{R})$ assigns a CDF $F(x)$ to each timer $x \in \mathcal{X}$, with $F(x)(t) = 0$ if $t < 0$.

We outline the PRTRT representation $\Psi_{SA}(A)$ of a finitary SA $A = (S, s_0, \mathcal{X}, \text{Act}, \longrightarrow, \kappa, F)$, where $\mathcal{X}$ is a finite set $\{x_1, \ldots, x_n\}$. The "timed state" of A is represented by a term $\texttt{\{}s, r_1, r_2, \ldots, r_n\texttt{\}}$, where s is a constant denoting the

current state of the SA, and r_i is the current value of the timer x_i. The following probabilistic rewrite rule randomly selects the initial timer values:

```
rl [init]: init => {s0, y1, ..., yn} with probability y1 := F1 and ... and yn := Fn .
```

where `init` is a constant and F_i mimics the CDF $F(x_i)$ of clock x_i. Each transition $s \xrightarrow{a,X} s'$ of the SA is translated to a labeled probabilistic rewrite rule

```
rl [a]: {s, r1, ..., rn} => {s', r1', ..., rn'} with probability r'j1 := Fj1 and ... and r'jl := Fjl.
```

where r_i is 0 if $x_i \in X$ and is a variable y_i otherwise, $\kappa(s') = \{x_{j_1}, x_{j_2}, \ldots, x_{j_l}\}$, and $r'_i = r_i$ if $x_i \notin \kappa(s')$. Time elapse is modeled by the following tick rule which can advance time until the next timer expires if no transition is enabled:

```
crl [tick]: {σ, y1, ..., yn} => {σ, max(y1 − y, 0), ..., max(yn − y, 0)} in time y
            if y <= nextTimerExpires(y1, ..., yn) and not transEnabled(σ, y1, ..., yn) .
```

where σ, y_i, and y are all variables, `nextTimerExpires(`$y_1, \ldots, y_n$`)` returns the smallest non-zero value of the y_i, and `transEnabled(`$\sigma, y_1, \ldots, y_n$`)` holds iff some transition is enabled in the given state. The latter is defined by an equation

```
eq transEnabled(s, r1, ..., rn) = true .
```

where r_i is 0 if $x_i \in X$ and is a variable y_i otherwise, for each transition $s \xrightarrow{a,X} s'$, and by having an equation that states that *otherwise* (i.e., if none of the above equations apply), `transEnabled(`$\sigma, y_1, \ldots, y_n$`)` is `false`:

```
eq transEnabled(σ, y1, ..., yn) = false [owise] .
```

The SA in Fig. 1 (b) is therefore represented as a PRTRT as follows

```
rl [init]: init => {s0 , y1 , y2} with probability y1 := F1 and y2 := F2 .
rl [push]: {s0 , 0, y2} => {s1 , y1', y2} with probability y1' := F1 .
rl [pop] : {s1 , y1 , 0} => {s2 , y1 , y2'} with probability y2' := F2 .
rl [push]: {s1 , y1 , 0} => {s3 , y1 , y2'} with probability y2' := F2 .
rl [push]: {s2 , 0, y2} => {s0 , y1', y2'} with probability y1' := F1 and y2' := F2 .
rl [pop] : {s3 , 0, 0} => {s2 , 0, y2'} with probability y2' := F2 .

crl [tick]: {σ, y1 , y2} => {σ, max(y1 - y, 0), max(y2 - y, 0)} in time y
            if y <= nextTimerExpires(y1 , y2) and not transEnabled(σ, y1 , y2) .
```

where `transEnabled` and `nextTimerExpires` are defined by:

```
eq transEnabled(s0 , 0, y2) = true .  eq transEnabled(s1 , y1 , 0) = true .
eq transEnabled(s2 , 0, y2) = true .  eq transEnabled(s3 , 0, 0) = true .
eq transEnabled(s, y1 , y2) = false [owise] .
eq nextTimerExpires(y1 , y2) = if y1 == 0 then (if y2 == 0 then INF else y2 fi)
                               else (if y2 == 0 then y1 else min(y1 , y2) fi) .
```

6.3 Deterministic and Stochastic Petri Nets

Deterministic and stochastic Petri nets (DSPNs) [2] are a fairly general class of timed Petri nets, where a transition can fire after having been continuously enabled for either a fixed (deterministic) or a random (exponentially distributed) amount of time. DSPNs are strictly more expressive than generalized stochastic Petri nets [3] (and, hence, stochastic Petri nets), which can be seen as DSPNs in which all deterministic transitions are instantaneous.

There are many variations of the basic model, including having inhibitor arcs, arc multiplicities that are functions of the marking, transition precedences, etc. To focus on the real-time and probabilistic aspects of the model, we assume a "standard" Petri net model extended with the above firing delays, and refer to [28] for the treatment of advanced Petri net features in rewriting logic. That is, a DSPN is a tuple (P, T, F, τ, R) where: P is a finite set of *places*; $T = T^D \uplus T^S$ is a finite set of *transitions*, partitioned into sets T^D and T^S of *deterministic* and *stochastic* transitions, respectively, and satisfying $T \cap P = \emptyset$; $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation; $\tau : T^D \to \mathbb{R}_{\geq 0}$ is a function that maps each *deterministic* transition t to its firing delay $\tau(t)$; if $\tau(t) = 0$ we call t an *instantaneous* transition; $R : T^S \to \mathbb{R}$ is a function that associates to each *stochastic* transition t the rate $R(t)$ of the exponential distribution of its firing delay. A transition must fire when it has been enabled continuously for the duration of its firing delay. We assume that the "enabled-time" of a transition is reset to zero when the transition is fired.

Figure 2 shows a DSPN specification, including its initial marking, of a client-server architecture. Stochastic (t_1), deterministic (t_3) and instantaneous (t_2) transitions are shown as empty rectangles, filled rectangles and thick lines, respectively. For the stochastic transitions we show the rate of the exponential distribution of their waiting times, while for the deterministic transitions we display the fixed amount of time associated with them.

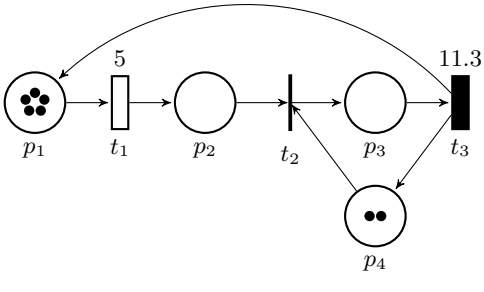

Fig. 2. A DSPN specifying a queueing model of a client-server architecture

Our representation follows the approach of [20], which sees a marking as a multiset of places and a transition as a multiset rewrite rule. In addition, for each transition $t \in T$, we associate a timer that denotes the remaining time during which the transition must be continuously enabled to fire. Such a timer

can be represented by a term `< t ; r >`, where t is the transition and r is its timer value. The global state of the system is therefore represented as a term `{m}`, where m is a multiset of places and transition timers, with multiset union denoted by juxtaposition. As a result of firing a transition, previously enabled transitions may be disabled, and vice versa. Therefore, we must recompute the transition timer values when a transition fires. For each transition t in the DSPN with pre-set $p_1\ p_2\ \ldots\ p_m$ and post-set $q_1\ q_2\ \ldots\ q_n$, we therefore have a rewrite rule

```
rl [apply-t] :
  {< t ; 0 >  p1  p2  ...  pm  REST}  =>  {recomputeTimers(< t ; INF >  q1  q2  ...  qn  REST)}.
```

which fires the transition t when its timer is 0. As a result, the pre-set is removed from the state, the post-set is added to it, t's timer is turned off (although it may be reset by `recomputeTimers` if the transition is still enabled, or re-enabled), and the function `recomputeTimers` is applied to the entire resulting state to recompute all transition timer values.

The function `recomputeTimers` is defined as follows: (*i*) if a transition is enabled and the corresponding timer is turned off (i.e., has the value `INF`), then the timer is reinitialized to the firing delay of the transition, otherwise the timer is left unchanged; and (*ii*) if a transition is not enabled, its timer is turned off. Case (*i*) can easily be defined by an equation defining `recomputeTimers` for *deterministic* transitions. However, an *equation* defining `recomputeTimers` cannot reset the timer of a stochastic transition, since the new timer value should be assigned probabilistically. Therefore, the timer of the stochastic transition is initialized to a new value `reset`, which will be replaced by a probabilistically chosen value in a *rewrite rule*. That is, for any *deterministic* transition t, case (*i*) above is defined by the following equation:

```
eq recomputeTimers(< t ; TI >  p1  p2  ...  pm  REST)
 = < t ; if TI == INF then τ(t) else TI fi >  recomputeTimers(p1  p2  ...  pm  REST) .
```

and for any *stochastic* transition t, case (*i*) is defined by the following equation:

```
eq recomputeTimers(< t ; TI >  p1  p2  ...  pm  REST)
 = < t ; if TI == INF then reset else TI fi >  recomputeTimers(p1  p2  ...  pm  REST) .
```

For each stochastic transition t with rate $R(t)$ we therefore have a rewrite rule

```
rl [set-stoc-timer]: < t ; reset >  =>  < t ; X > with probability X := ExpRate(R(t)) .
```

where the function `ExpRate(λ)` mimics the CDF of the exponential distribution with rate parameter $\lambda \in \mathbb{R}$. For case (*ii*), if the transition is not enabled, the following `owise` equation sets the corresponding timer to `INF`:

```
eq recomputeTimers(< T ; TI >  REST)  =  < T ; INF >  recomputeTimers(REST) [owise] .
```

where `T` is a variable. Finally, we add the tick rule

```
crl [tick]: {SYSTEM}  =>  {decreaseTimers(SYSTEM, Y)} in time Y if Y <= mte(SYSTEM).
```

where `decreaseTimers` decreases the value of each timer by the elapsed time `Y`, and `mte` gives the smallest timer value (or 0 if a timer has the value `reset`). Therefore, this tick rule may advance time until the next timer expires. Appendix A gives a detailed specification of the PRTRT representation of a DSPN.

The representation of the DSPN in Fig. 2 contains the instantaneous rules

```
rl [apply-t1] : {< t1 ; 0 > p1 REST}  =>  {recomputeTimers(< t1 ; INF > p2 REST)} .
rl [apply-t2] : {< t2 ; 0 > p2 p4 REST}  =>  {recomputeTimers(< t2 ; INF > p3 REST)} .
rl [apply-t3] : {< t3 ; 0 > p3 REST}  =>  {recomputeTimers(< t3 ; INF > p1 p4 REST)} .
```

together with the equations defining the `recomputeTimers` function

```
eq recomputeTimers(< t1 ; TI >  p1 REST)
 =  < t1 ; if TI == INF then reset else TI fi >  recomputeTimers(p1 REST) .
eq recomputeTimers(< t2 ; TI >  p2 p4 REST)
 =  < t2 ; if TI == INF then 0 else TI fi >  recomputeTimers(p2 p4 REST) .
eq recomputeTimers(< t3 ; TI >  p3 REST)
 =  < t3 ; if TI == INF then 11.3 else TI fi >  recomputeTimers(p3 REST) .
eq recomputeTimers(< t ; TI >  REST)  =  < t ; INF >  recomputeTimers(REST) [owise] .
ceq recomputeTimers(REST)  =  REST  if noTimers(REST) .
```

as well as the tick rule and the rule for setting the timer of the stochastic transition t_1 to a value sampled from the exponential distribution with rate 5:

```
rl [set-stoc-timer] : < t1 ; reset >  =>  < t1 ; X >  with probability X := ExpRate(5).
```

6.4 Handling Uncertainty in Probabilistic Transitions

We have identified two models for probabilistic real-time systems where the probability distribution associated with a transition is nondeterministically chosen from a *set* of probability distributions. We can represent these models as PRTRTs, which implies that PRTRTs are more expressive than (untimed) probabilistic rewrite theories in which the probability distribution is deterministically chosen. See [6] for details about the PRTRT representation of these models.

In *timed probabilistic transition systems* (TPTS) [29] the probability of making a transition belongs to an *interval*. This can be modeled in our formalism by exploiting the fact that, for a given rewrite rule r, π_r is a family of probability distributions, indexed both by the substitutions for the variables in the left-hand side of r and by the substitutions for the nondeterministically instantiated variables in the righthand side of r. In our PRTRT encoding we add the nondeterministically selected probability values to the state. Therefore, we specify the probabilistic transition from s_0 of the TPTS in Fig. 3 (a) by the following probabilistic tick rewrite rule, where σ, p and q are variables:

```
crl [tick] : {s0 ; q} => {σ ; p} in time 1 if p ∈ [0.9, 1]
                                 with probability σ := ( s1   s2  ) .
                                                       ( p  1 − p )
```

In *timed probabilistic systems* (TPS) [10], the time that a node waits in a location—after selecting an outgoing action, but before *performing* the action—is a random value, whose *average* is given, but whose probability distribution is

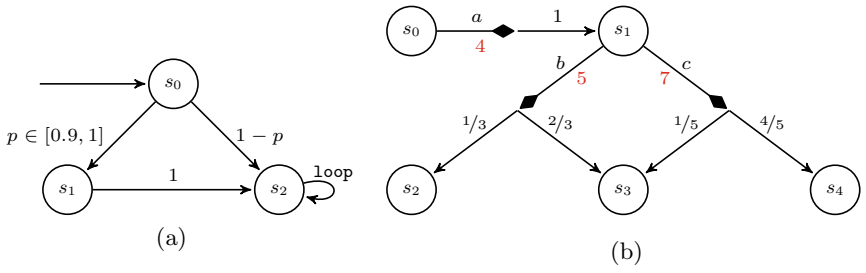

Fig. 3. (a) A timed probabilistic transition system (b) A timed probabilistic system

not specified. Figure 3 (b) shows an example of a TPS, where the actions are depicted as diamond-tipped arrows, the average time for performing an action is shown in red, and the probabilistic transitions are the arrows with probability values attached to them. We assume that the set of possible waiting times of action b is a finite[3] set $\{\varphi_1, \varphi_2, \ldots, \varphi_m\}$, where φ_i is selected with an *unknown* probability $p_i \in [0, 1]$. A state in the PRTRT representation of a TPS has the form $\{s, a, s', r, p_1; \ldots; p_n\}$, where s is the current state, a is the next action to perform, s' is the next state, r is the remaining time until the automaton performs the action a, and $p_1, \ldots, p_m$ are the nondeterministically chosen probability values as for TPTSs. The rule that performs action a when the timer expires and selects b as the next action is then

$$\texttt{crl } [a_b]\texttt{: } \{s_0\,,\, a\,,\, s_1\,,\, 0\,,\, q_1\,;\,\ldots\,;\, q_n\} \;\texttt{=>}\; \{s_1\,,\, b\,,\, \sigma\,,\, r\,,\, p_1\,;\,\ldots\,;\, p_m\}$$
$$\texttt{if } p_1 + \ldots + p_m = 1 \texttt{ and } p_1\varphi_1 + \ldots + p_m\varphi_m = 5$$
$$\texttt{with probability } \sigma := \begin{pmatrix} s_2 & s_3 \\ 1/3 & 2/3 \end{pmatrix} \texttt{ and } r := \begin{pmatrix} \varphi_1 & \ldots & \varphi_m \\ p_1 & \ldots & p_m \end{pmatrix} .$$

7 Concluding Remarks

We have defined the probabilistic real-time rewrite theory (PRTRT) formalism for modeling probabilistic real-time systems in rewriting logic, and have shown how PRTRTs can be seen as a unifying semantic framework in which a range of models for probabilistic real-time systems can be naturally represented, including systems with underspecified probability distributions. We have also given a PRTRT specification of a simple round trip time protocol that seems to be outside the class of systems that can be modeled using automaton-based formalisms, since the number of messages in a state can grow beyond any bound.

This work has provided the theoretical foundations for an analysis tool for probabilistic real-time systems in rewriting logic. In the future we must define property specification formalisms and implement suitable model checkers for PRTRTs. For this purpose, the statistical model checking approach seems very promising, since instead of performing exact probabilistic model checking—which often becomes unfeasible for large distributed systems—statistical model

[3] See [6] for the continuous case.

checking is typically much more efficient, although it only guarantees a property with a desired level of confidence. In particular, statistical model checking is based on evaluating a number of behaviors and is therefore easily parallelizable. Indeed, the PVeStA tool [4] provides a parallel statistical model checker for a subset of (untimed) probabilistic rewrite theories and could be a useful starting point for a future tool for PRTRTs.

References

1. Agha, G., Greenwald, M., Gunter, C.A., Khanna, S., Meseguer, J., Sen, K., Thati, P.: Formal modeling and analysis of DoS using probabilistic rewrite theories. In: Proc. FCS 2005 (2005)
2. Ajmone Marsan, M., Chiola, G.: On Petri nets with deterministic and exponentially distributed firing times. In: Rozenberg, G. (ed.) APN 1987. LNCS, vol. 266, Springer, Heidelberg (1987)
3. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Trans. Comput. Syst. 2 (1984)
4. AlTurki, M., Meseguer, J.: PVeStA: A parallel statistical model checking and quantitative analysis tool. To appear in Proc. CALCO 2011 (2011)
5. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126 (1994)
6. Bentea, L., Ölveczky, P.C.: Probabilistic real-time rewrite theories and their expressive power (2011), `http://www.ifi.uio.no/~lucianb/publications/2011/prt-exp.pdf`
7. Bruni, R., Meseguer, J.: Semantic foundations for generalized rewrite theories. Theoretical Computer Science 360(1-3) (2006)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
9. D'Argenio, P.R., Katoen, J.P., Brinksma, E.: An algebraic approach to the specification of stochastic systems. In: PROCOMET 1998. Chapman & Hall, Ltd., Boca Raton (1998)
10. De Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University, USA (1998)
11. Gilmore, S., Hillston, J.: The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In: Computer Performance Evaluation (1994)
12. Johnson, N.L., Kotz, S., Balakrishnan, N.: Continuous Univariate Distributions, 2nd edn. Wiley Series in Probability and Statistics, vol. 1. Wiley-Interscience, Hoboken (1994)
13. Katelman, M., Meseguer, J., Hou, J.: Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In: Barthe, G., de Boer, F.S. (eds.) FMOODS 2008. LNCS, vol. 5051, pp. 150–169. Springer, Heidelberg (2008)
14. Kumar, N., Sen, K., Meseguer, J., Agha, G.: Probabilistic rewrite theories: Unifying models, logics and tools. Technical report UIUCDCS-R-2003-2347, Department of Computer Science, University of Illinois at Urbana-Champaign (2003)
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic model checking for performance and reliability analysis. ACM SIGMETRICS Performance Evaluation Review 36(4) (2009)

16. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. 282 (2002)
17. Lien, E., Ölveczky, P.C.: Formal modeling and analysis of an IETF multicast protocol. In: SEFM 2009. IEEE Computer Society, Los Alamitos (2009)
18. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96(1) (1992)
19. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, Springer, Heidelberg (1998)
20. Meseguer, J., Montanari, U.: Petri nets are monoids. Information and Computation 88(2) (1990)
21. Ölveczky, P.C., Meseguer, J.: The Real-Time Maude tool. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 332–336. Springer, Heidelberg (2008)
22. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theoretical Computer Science 285(2), 359–405 (2002)
23. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20 (2007)
24. Ölveczky, P.C., Meseguer, J., Talcott, C.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Formal Methods in System Design 29, 253–293 (2006)
25. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. Theoretical Computer Science 410(2-3), 254–280 (2009)
26. Sen, K., Viswanathan, M., Agha, G.A.: VESTA: A statistical model-checker and analyzer for probabilistic systems. In: QEST 2005. IEEE Computer Society, Los Alamitos (2005)
27. Sproston, J.: Model Checking for Probabilistic Timed and Hybrid Systems. Ph.D. thesis, School of Computer Science, University of Birmingham (2001)
28. Stehr, M.O., Meseguer, J., Ölveczky, P.C.: Rewriting logic as a unifying framework for petri nets. In: Ehrig, H., Juhás, G., Padberg, J., Rozenberg, G. (eds.) APN 2001. LNCS, vol. 2128, pp. 250–303. Springer, Heidelberg (2001)
29. Yi, W.: Algebraic reasoning for real-time probabilistic processes with uncertain information. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 680–693. Springer, Heidelberg (1994)
30. Zhang, H., Hou, J.: Maintaining Sensing Coverage and Connectivity in Large Sensor Networks. Ad Hoc & Sensor Wireless Networks 1(1-2) (2005)

A PRTRT Representation of DSPNs

We give below a more detailed specification, using Maude syntax, of the PRTRT representation of a DSPN.

```
sort Place .
ops p1 p2 p3 p4 ... : -> Place [ctor] .   --- one constant for each place

sort Transition .
ops t1 t2 t3 ... : -> Transition [ctor] . --- one constant for each transition
```

```
op reset : -> TimeInf [ctor] .            --- new 'timer' value

sort TransitionTimer .                    --- transition timers
op <_;_> : Transition TimeInf -> TransitionTimer [ctor] .

--- An extended marking is a multiset of places and transition timers:
sort ExtendedMarking .    subsort TransitionTimer Place < ExtendedMarking .
op none : -> ExtendedMarking [ctor] .     --- empty marking
--- assoc-comm multiset union operator:
op __ : ExtendedMarking ExtendedMarking -> ExtendedMarking
                                           [ctor assoc comm id: none] .

--- For EACH transition t we have a firing rule of the following form. Assume
--- that the preset of t is q1 ... qm and the postset of t is q1' ... qn', where
--- each qi and qk' is some place pj:

vars REST SYSTEM : ExtendedMarking .  var TI : TimeInf .  var T : Transition .
var X : Time .

rl [fire-t] :
   {< t ; 0 >  q1 ... qm REST}
   =>
   {recomputeTimers(< t ; INF >  q1' ... qn' REST)} .

op recomputeTimers : ExtendedMarking -> ExtendedMarking [frozen (1)] .

--- For EACH deterministic transition t, with the above pre- and postsets,
--- there is one equation as follows:

eq recomputeTimers(< t ; TI >  q1 ... qm REST)    --- t is enabled!
  =
   if TI == INF        --- t was previously disabled
     < t ; tau(t) >    --- initialize with value of firing delay
   else
     < t ; TI >        --- t was already enabled, do not change timer value
   fi
   recomputeTimers(q1 ... qm REST) .  --- recursively compute the other timers

--- For EACH stochastic transition t, with the above pre- and postsets,
--- there is one equation as follows, which is very similar to the
--- equation above, but resets the timer to 'reset':

eq recomputeTimers(< t ; TI >  q1 ... qm REST)    --- t is enabled
  =
   if TI == INF        --- t was previously disabled
     < t ; reset >     --- initialize with value 'reset'
   else
     < t ; TI >        --- t was already enabled, do not change timer value
   fi
   recomputeTimers(q1 ... qm REST) .  --- recursively compute the other timers

--- An owise equation matches when the transition T is not enabled.
--- Then we set the timer to 'INF', no mater its earlier value:
```

```
eq recomputeTimers(< T ; TI >  REST)  =  < T ; INF >  recomputeTimers(REST) .

--- Finally, when there are no transitions timers left to apply
--- recomputeTimers to, we are finished:

ceq recomputeTimers(REST)  =  REST  if noTimers(REST) .

--- This noTimers could also have been done with sorts, etc.
op noTimers : ExtendedMarking -> Bool .
eq noTimers(< T ; TI >  REST)  = false .
eq notimers(REST)  =  true [owise] .

--- Next, we instantiate the probabilistic variable; details
--- about the probability distribution omitted:

rl [instantiate-probabilistic-delay] :
   < T ; reset >  =>  < T ; X >  with probability X := distr(...R(T)...) .

--- Finally, the tick rule:
crl [tick] :
    {SYSTEM}  =>  {decreaseTimers(SYSTEM, X)}  in time X if X <= mte(SYSTEM) .

op decreaseTimers : ExtendedMarking Time -> ExtendedMarking [frozen (1)] .
eq decreaseTimers(< T ; TI >  REST, X)
  = < T ; TI - X >  decreaseTimers(REST, X) .
eq decreaseTimers(REST) = REST [owise] .    --- no more timers

--- The function mte gives the smallest timer value in the system:
op mte : ExtendedMarking -> TimeInf [frozen (1)] .
eq mte(< T ; TI >  REST) = min(if TI == reset then 0 else TI fi, mte(REST)) .
eq mte(REST) = INF [owise] .
```