A Rewriting Based Model for Probabilistic Distributed Object Systems

Nirman Kumar, Koushik Sen, José Meseguer, Gul Agha
Department of Computer Science,
University of Illinois at Urbana-Champaign.
{nkumar5,ksen,meseguer,agha}@cs.uiuc.edu

Abstract. Concurrent and distributed systems have traditionally been modelled using nondeterministic transitions over configurations. The nondeterminism provides an abstraction over scheduling, network delays, failures and randomization. However a probabilistic model can capture these sources of nondeterminism more precisely and enable statistical analysis, simulations and reasoning. We have developed a general semantic framework for probabilistic systems using probabilistic rewriting. Our framework also allows nondeterminism in the system. In this paper, we briefly describe the framework and its application to concurrent object based systems such as actors. We also identify a sufficiently expressive fragment of the general framework and describe its implementation. The concepts are illustrated by a simple client-server example.

Keywords: Rewrite theory, probability, actors, Maude, nondeterminism.

1 Introduction

A number of factors, such as processor scheduling and network delays, failures, and explicit randomization, generally result in nondeterministic execution in concurrent and distributed systems. A well known consequence of such nondeterminism is an exponential number of possible interactions which in turn makes it difficult to reason rigorously about concurrent systems. For example, it is infeasible to use techniques such as model checking to verify any large-scale distributed systems. In fact, some distributed systems may not even have a finite state model: in particular, *networked embedded systems* involving *continuously* changing parameters such as time, temperature or available battery power are infinite state.

We believe that a large class of concurrent systems may become amenable to a rigorous analysis if we are able to quantify some of the probabilities of transitions. For example, network delays can be represented by variables from a probabilistic distribution that depends on some function of the system state. Similarly, available battery power, failure rates, etc., may also have a probabilistic behavior. A probabilistic model can capture the statistical regularities in such systems and enable us to make probabilistic guarantees about its behavior.

We have developed a model based on rewriting logic [11] where the rewrite rules are enriched with probability information. Note that rewriting logic provides a natural model for object-based systems [12]. The local computation of each object is modelled by rewrite rules for that object and one can reason about the global properties that result from the interaction between objects: such interactions may be asynchronous as in actors, or synchronous as in the π-calculus. In [9] we show how several well known models of probabilistic and nondeterministic systems can be expressed as special cases of probabilistic rewrite theories. We also propose a temporal logic to express properties of interest in probabilistic systems. In this paper we show how probabilistic object systems can be modelled in our framework. Our *probabilistic rewriting* model is illustrated using a client-server example. The example also shows how nondeterminism, for which we do not have the probability distributions, is represented naturally in our model. Nondeterminism is eventually removed by the system *adversary* and converted into probabilities in order to define a probability space over computation paths.

The *Actor* model of computation [1] is widely used to model and reason about object-based distributed systems. Actors have previously been modelled as rewrite theories [12]. Probabilistic rewrite theories can be used to model and reason about actor systems where actors may fail and messages may be dropped or delayed and the associated probability distributions are known (see Section 3).

The rest of this paper is organized as follows. Section 2 provides some background material on membership equational logic [13] and rewriting [11] as well as probability theory. Section 3 starts by giving an intuitive understanding of how a step of computation occurs in a probabilistic rewrite theory. We then introduce an example to motivate the modelling power of our framework and formalize the various concepts. In Section 4 we define an important subclass of probabilistic rewrite theories, and in Section 5, we describe its Maude implementation. The final section discusses some directions for future research.

2 Background and Notation

A *membership equational theory* [13] is a pair (Σ, E), with Σ a *signature* consisting of a set K of *kinds*, for each $k \in K$ a set S_k of *sorts*, a set of *operator* declarations of the form $f : k_1 \ldots k_n \to k$, with $k, k_1, \ldots, k_n \in K$ and with E a set of *conditional Σ-equations* and *Σ-memberships* of the form

$$(\forall \overrightarrow{x})\ t = t' \Leftarrow u_1 = v_1 \wedge \ldots \wedge u_n = v_n \wedge w_1 : s_1 \wedge \ldots \wedge w_m : s_m$$
$$(\forall \overrightarrow{x})\ t : s \Leftarrow u_1 = v_1 \wedge \ldots \wedge u_n = v_n \wedge w_1 : s_1 \wedge \ldots \wedge w_m : s_m$$

The $\overrightarrow{x}$ denote *variables* in the terms t, t', u_i, v_i and w_j above. A membership $w : s$ with w a Σ-term of kind k and $s \in S_k$ asserts that w has sort s. Terms that do not have a sort are considered *error* terms. This allows membership equational theories to specify partial functions within a total framework. A *Σ-algebra* B consists of a K-indexed family of sets $X = \{B_k\}_{k \in K}$, together with

1. for each $f : k_1 \ldots k_n \to k$ in Σ a function $f_B : B_{k_1} \times \ldots \times B_{k_n} \to B_k$

2. for each $k \in K$ and each $s \in S_k$ a subset $B_s \subseteq B_k$.

We denote the algebra of terms of a membership equational theory by T_Σ. The *models* of a membership equational theory (Σ, E) are those Σ-algebras that satisfy the equations E. The inference rules of membership equational logic are *sound* and *complete* [13]. Any membership equational theory (Σ, E) has an *initial algebra* of terms denoted $T_{\Sigma/E}$ which, using the inference rules of membership equational logic and assuming Σ *unambiguous* [13], is defined as a quotient of the term algebra T_Σ by

- $t \equiv_E t' \quad \Leftrightarrow \quad E \vdash (\forall\emptyset)\ t = t'$
- $[t]_{\equiv_E} \in T_{\Sigma/E,s} \quad \Leftrightarrow \quad E \vdash (\forall\emptyset)\ t : s$

In [2] the usual results about *equational simplification*, *confluence*, *termination*, and *sort-decreasingness* are extended in a natural way to membership equational theories . Under those assumptions a membership equational theory can be executed by equational simplification using the equations from left to right, perhaps modulo some *structural* (e.g. associativity, commutativity and identity) axioms A. We denote the algebra of terms simplified by equations and structural axioms as $T_{\Sigma,E\cup A}$ and the isomorphic algebra of equivalence classes modulo axioms A, of equationally simplified terms by $Can_{\Sigma,E/A}$. The notation $[t]_A$ represents the equivalence class of a term t fully simplified by the equations.

In a standard *rewrite theory* [11], transitions in a system are described by labelled rewrite rules of the form

$$l : t(\overrightarrow{x}) \longrightarrow t'(\overrightarrow{x})\ \textit{if}\ C(\overrightarrow{x})$$

Intuitively, a rule of this form specifies a *pattern* $t(\overrightarrow{x})$ such that if some fragment of the system's state matches that pattern and satisfies the condition C, then a local transition of that state fragment, changing into the pattern $t'(\overrightarrow{x})$ can take place. In a *probabilistic rewrite rule* we add probability information to such rules. Specifically, our proposed probabilistic rules are of the form,

$$l : t(\overrightarrow{x}) \longrightarrow t'(\overrightarrow{x}, \overrightarrow{y})\ \textit{if}\ C(\overrightarrow{x})\ \textit{with probability}\ \pi(\overrightarrow{x}).$$

In the above, the set of variables in the left-hand side term $t(\overrightarrow{x})$ is $\overrightarrow{x}$, while some new variables $\overrightarrow{y}$ may be present in the term $t'(\overrightarrow{x}, \overrightarrow{y})$ on the right-hand side. Of course it is not necessary that *all* of the variables in $\overrightarrow{x}$ occur in $t'(\overrightarrow{x}, \overrightarrow{y})$. The rule will match a state fragment if there is a substitution θ for the variables $\overrightarrow{x}$ that makes $\theta(t)$ equal to that state fragment and the condition $\theta(C)$ is true. Because the right-hand side $t'(\overrightarrow{x}, \overrightarrow{y})$ may have new variables $\overrightarrow{y}$, the next state is not *uniquely* determined: it depends on the choice of an additional substitution ρ for the variables $\overrightarrow{y}$. The choice of ρ is made according to the probability function $\pi(\theta)$, where π is not a fixed probability function, but a *family* of functions: one for each match θ of the variables $\overrightarrow{x}$.

The Maude system [4,5] provides an execution environment for membership equational and rewrite theories. The Full Maude [6] library built on top of the Core Maude environment allows users to specify object oriented modules in a convenient syntax. Several examples in [12,5] show specifications of object based systems in Maude. The code for our example in Section 3 is written in the syntax of Maude 2.0 [5].

To succinctly define probabilistic rewrite theories, we use a few basic notions from axiomatic probability theory. A σ-algebra on a set X is a collection $\mathcal{F}$ of subsets of X, containing X itself and closed under complementation and finite or countably infinite unions. For example the power set $\mathcal{P}(X)$ of a set X is a σ-algebra on X. The elements of a σ-algebra are called *events*. We denote by $\mathcal{B}_{\mathbb{R}}$ the smallest σ-algebra on $\mathbb{R}$ containing the sets $(-\infty, x]$ for all $x \in \mathbb{R}$. We also remind the reader that a *probability space* is a triple $(X, \mathcal{F}, \pi)$ with $\mathcal{F}$ a σ-algebra on X and π a *probability measure function*, defined on the σ-algebra $\mathcal{F}$ which evaluates to 1 on X and distributes by addition over finite or countably infinite union of disjoint events. For a given σ-algebra $\mathcal{F}$ on X, we denote by $\mathit{PFun}(X, \mathcal{F})$ the set

$$\{\pi \mid (X, \mathcal{F}, \pi) \text{ is a probability space}\}$$

Definition 1 ($\mathcal{F}$-cover). *For a σ-algebra $\mathcal{F}$ on X, an $\mathcal{F}$-cover is a function $\alpha : X \to \mathcal{F}$, such that $\forall x \in X \;\; x \in \alpha(x)$.*

Let π be a probability measure function defined on a σ-algebra $\mathcal{F}$ on X, and suppose α is an $\mathcal{F}$-cover. Then notice that $\pi \circ \alpha$ naturally defines a function from X to $[0, 1]$. Thus, for example, for $X = \mathbb{R}$ and $\mathcal{F} = \mathcal{B}_{\mathbb{R}}$, we can define α to be the function that maps the real number x to the set $(-\infty, x]$. With X a finite set and $\mathcal{F} = \mathcal{P}(X)$, the power set of X, it is natural to define α to be the function that maps $x \in X$ to the singleton $\{x\}$.

3 Probabilistic Rewrite Theories

A probabilistic rewrite theory has an interleaving execution semantics. A step of computation changes a term $[u]_A$ to $[v]_A$ by the application of a single rewrite rule on some subterm of the given *canonical* term $[u]_A$. Recall the form of a probabilistic rewrite rule as described in the previous section. Firstly, all context, rule, substitution (for the variables $\overrightarrow{x}$) triples arising from possible applications of rewrite rules (see definition 6) to $[u]_A$ are computed. One of them $([\mathbb{C}]_A, r, [\theta]_A)$ (for the justification of the A subscript see definitions 2, 3 and 5) is chosen *nondeterministically*. This step essentially represents the nondeterminism in the system. After that has been done, a particular substitution $[\rho]_A$ is chosen *probabilistically* for the new variables $\overrightarrow{y}$ and $[\rho]_A$ along with $[\theta]_A$, is applied to the term $t'(\overrightarrow{x}, \overrightarrow{y})$ and placed inside the context $\mathbb{C}$ to obtain the term $[v]_A$. The choice of the new substitution $[\rho]_A$ is from the set of possible substitutions for $\overrightarrow{y}$. The probabilities are defined as a *function* of $[\theta]_A$. This gives the framework great expressive power. Our framework can model both nondeterminism and probability in the system. Next we describe our example, model it as an object based rewrite theory and indicate how the rewrite rules model the probabilities and nondeterminism.

```
pmod QOS-MODEL is
        ...
        vars L N m1 m2: Nat.
        vars Cl Sr Nw: Oid.
        var i: Bit.
        vars C Q: Configuration.
        var M: Msg.
        op _ ← _: Oid Nat → Msg.
        class Client    |sent:Nat, svc1:Nat, svc2:Nat.
        class Network  |soup:Configuration.
        class Server    |queue:Configuration.
        ops H Nt S1 S2: → Oid.
        ops acq1 acq2: → Msg.
        prl [req]:⟨Cl:Client|sent:N, svc1:m1, svc2:m2⟩⟨Nw: Network|soup:C⟩⇒
         ⟨Cl: Client|sent:(N + 1), svc1:m1, svc2:m2⟩⟨Nw: Network|soup:C (Sr ← L)⟩.
        cprl [acq]:⟨Cl:Client|svc1:m1, svc2:m2⟩⟨Nw:Network|soup:M C⟩⇒
         ⟨Cl:client|svc1:m1 + δ(i, M, 1), svc2:m2 + δ(i, M, 2)⟩⟨Nw:Network|soup:C⟩
          if acq(M).
        prl [deliver]:⟨Nw:Network|soup:(Sr ← L) C⟩⟨Sr:Server|queue:Q⟩⇒
         ⟨Nw:Network|soup:C⟩⟨Sr:Server|queue:Q M⟩.
        prl [process]:⟨Sr:Server|queue:(Sr ← L) Q⟩⟨Nw:Network|soup:C⟩ ⇒
         ⟨Sr:Server|queue:Q⟩⟨Nw:Network|soup:C M⟩.
endpm
```

Fig. 1. A client-server example

A Client-Server Example : Our example is a situation where a client is sending computational jobs to servers across a network. There are two servers S_1 and S_2. S_1 is computationally more powerful than S_2, but the network connectivity to S_2 is better (more reliable) than that to S_1 and packets to S_1 may be dropped without being delivered, more frequently than packets to S_2. The servers may also drop requests if the load increases beyond a certain threshold. The computationally more powerful server S_1 drops packets with a lower probability than S_2. We would like to reason about a good randomized policy for the client. The question here is: which server is it better to send packets to, so that a larger fraction of packets are processed rather than dropped? Four objects model the system. One of them, the client, sends packets to the two server objects deciding probabilistically before each send which server to send the packet to. The other object models a network, which can either transmit the packets correctly, drop them or deliver them out of order. The remaining two objects are server objects which either drop a request or process it and send an acknowledgement message. The relevant fragment of code specifying the example is given in Figure 1. The client object named H maintains the total number of requests sent in a variable *sent* and those which were successfully processed by servers S_1, S_2 in variables svc_1 and svc_2 respectively. Notice that for example in $svc_1 : m_1$ the m_1 is the *value* of the variable named svc_1. An example term representing a possible system state is

$\langle$ H : Client | sent : 3, svc_1 : 1, svc_2 : 0 $\rangle$ $\langle$ Nt : Network | soup : $(S_1 \leftarrow 10)$ $\rangle$
$\langle$ S_1 : Server | queue : nil $\rangle$ $\langle$ S_2 : Server | queue : $(S_2 \leftarrow 5)$ $\rangle$

The term above of sort *Configuration* (collection of objects and messages) represents a multiset of objects combined with an empty syntax (juxtaposition) multiset union operator that is declared associative and commutative. The client has sent 3 requests in total, out of which one has already been serviced by S_1, one is yet to be delivered and one request is yet pending at the server S_2. The numbers 10 and 5 represent the measure of the loads in the respective requests.

We discuss the rules labelled *req* and *acq*. Henceforward we refer to a rule by its label. Though not shown in Figure 1, a probabilistic rewrite theory associates some functions with the rules, defining the probabilities. The rule *req* models the client sending a request to one of the servers by putting a message into the network object's variable *soup*. The rule involves two new variables Sr and L on the right-hand side. Sr is the name of the server to which the request is sent and L is the message *load*. A probability function $\pi_{req}(Cl, N, m_1, m_2, Nw, C)$ associated with the rule *req* (see definition 4) will decide the distribution of the new variables Sr and L, and thus the randomized policy of the client. For example, it can assign higher probability values to substitutions with $Sr = S_1$, if it finds that $m_1 > m_2$; this would model a heuristic policy which sends more work to the server which is performing better. In this way the probabilities can depend on the values of m_1, m_2 (and thus the state of the system). In the rule labelled *acq* there is only one new variable i on the right-hand side. That variable can only assume two values $0, 1$ with nonzero probability. 0 means a message drop, so that $\delta(0, M, 1) = \delta(0, M, 2) = 0$, while if $i = 1$ then the appropriate *svc* variable is incremented. The distribution of i as decided by the function $\pi_{acq}(\ldots, M)$ could depend on M, effectively modelling the network connectivity. The network drops messages more frequently for $M = acq_1$ (an acq message from server S_1) than it does for $M = acq_2$. Having the distribution of new variables *depend* on the substitution gives us the ability to model general distributions. The associativity and commutativity attribute of the juxtaposition operator for the sort *Configuration* essentially allows nondeterminism in the order of message delivery by the network (since it chooses a message to process, from the associative commutative *soup* of messages) and the order of messages processed by the servers.

The more frequently the rewrite rules for the network object are applied (which allow it to process the messages in it soup), the more frequently the *acq* messages will be delivered. Likewise, the more frequently the rewrite rules for a particular server are applied, the more quickly will it process its messages. Thus, during a computation the values m_1, m_2, which determine the client's randomized policy, will actually depend not only on the probability that a server correctly processes the packets and the network correctly delivers requests and acknowledgments, but also on how frequently the appropriate rewrite rules are applied. However, the exact frequency of application depends on the *nondeterministic* choices made. We can now see how the nondeterminism effectively influences the probabilities in the system. As explained later, the nondeterminism is removed (converted into probabilities) by what is called an *adversary* of the system. In essence the adversary is like a *scheduler* which determines the rate of progress of each component. The choice of adversary is important for the behavior of the system. For example, we may assume a *fair* adversary that chooses between its nondeterministic choices equally frequently. At an intuitive level this would mean that the different parts of the system compute at the same rate. Thus, it must be understood that the model defined by a probabilistic rewrite theory is parameterized on the adversary. The system modeler must define the adversary based on an understanding of how frequently different objects in the system advance. Model checking of probabilistic systems quantifies over adversaries, whereas a simulation has to fix an adversary.

We now define our framework formally.

Definition 2 (E/A-canonical ground substitution). *An E/A-canonical ground substitution is a substitution $\theta : \overrightarrow{x} \to T_{\Sigma,E\cup A}$.*

Intuitively an E/A-canonical ground substitution represents a substitution of ground terms from the term algebra T_Σ for variables of the corresponding sorts, so that all of the terms have already been reduced as much as possible by the equations E and the structural axioms A. For example the substitution 10×2 to a variable of sort *Nat is not* a canonical ground substitution, but a substitution of 20 for the same variable is a canonical ground substitution.

Definition 3 (A-equivalent substitution). *Two E/A-canonical ground substitution $\theta, \rho : \overrightarrow{x} \to T_{\Sigma,E\cup A}$ are A-equivalent if and only if $\forall x \in \overrightarrow{x}\ [\theta(x)]_A = [\rho(x)]_A$.*

We use $CanGSubst_{E/A}(\overrightarrow{x})$ to denote the set of all E/A-canonical ground substitutions for the set of variables $\overrightarrow{x}$. It is easy to see that the relation of A-equivalence as defined above is an equivalence relation on the set $CanGSubst_{E/A}(\overrightarrow{x})$. When the set of variables $\overrightarrow{x}$ is understood, we use $[\theta]_A$ to denote the equivalence class containing $\theta \in CanGSubst_{E/A}(\overrightarrow{x})$.

Definition 4 (Probabilistic rewrite theory). *A probabilistic rewrite theory is a 4-tuple $\mathcal{R} = (\Sigma, E \cup A, R, \pi)$, with $(\Sigma, E \cup A, R)$ a rewrite theory with the rules $r \in R$ of the form*

$$l : t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y})\ \mathit{if}\ C(\overrightarrow{x})$$

where

- *$\overrightarrow{x}$ is the set of variables in t.*
- *$\overrightarrow{y}$ is the set of variables in t' that are not in t. Thus t' might have variables coming from the set $\overrightarrow{x} \cup \overrightarrow{y}$ but it is not necessary that all variables in $\overrightarrow{x}$ occur in t'.*
- *C is a condition of the form $(\bigwedge_j u_j = v_j) \wedge (\bigwedge_k w_k : s_k)$, that is, C is a conjunction of equations and memberships;*

and π is a function assigning to each rewrite rule $r \in R$ a function

$$\pi_r : [\![C]\!] \to PFun(CanGSubst_{E/A}(\overrightarrow{y}), \mathcal{F}_r)$$

where $[\![C]\!] = \{[\mu]_A \in CanGSubst_{E/A}(\overrightarrow{x}) \mid E \cup A \vdash \mu(C)\}$ is the set of E/A-canonical substitutions for $\overrightarrow{x}$ satisfying the condition C, and $\mathcal{F}_r$ is a σ-algebra on $CanGSubst_{E/A}(\overrightarrow{y})$. We denote a rule r together with its associated function π_r, by the notation

$$l : t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y}) \text{ if } C(\overrightarrow{x}) \text{ with probability } \pi_r(\overrightarrow{x})$$

We denote the class of probabilistic rewrite theories by **PRwTh**. Notice the following points in the definition

1. Rewrite rules may have new variables $\overrightarrow{y}$ on the right-hand side.
2. The condition $C(\overrightarrow{x})$ on the right-hand side depends only on the variables $\overrightarrow{x}$ occurring in the term $t(\overrightarrow{x})$ on the left-hand side.
3. The condition $C(\overrightarrow{x})$ is simply a conjunction of equations and memberships (but no rewrites).
4. $\pi_r(\overrightarrow{x})$ specifies, for each substitution θ for the variables $\overrightarrow{x}$, the probability of choosing a substitution ρ for the $\overrightarrow{y}$. In the next section we explain how this is done.

3.1 Semantics of Probabilistic Rewrite Theories

Let $\mathcal{R} = (\Sigma, E \cup A, R, \pi)$ be a probabilistic rewrite theory such that:

1. E is confluent, terminating and sort-decreasing modulo A [2].
2. the rules R are coherent with E modulo A [4].

We also assume a choice for each rule r of an $\mathcal{F}_r$-cover $\alpha_r : CanGSubst_{E/A}(\overrightarrow{y}) \to \mathcal{F}_r$. This $\mathcal{F}_r$-cover will be used to assign probabilities to rewrite steps. Its choice will depend on the particular problem under consideration.

Definition 5 (Context). *A* context *$\mathbb{C}$ is a Σ-term with a single occurrence of a single variable, $\odot$, called the* hole*. Two contexts $\mathbb{C}$ and $\mathbb{C}'$ are A-equivalent if and only if $A \vdash (\forall \odot)\ \mathbb{C} = \mathbb{C}'$.*

Notice that the relation of A-equivalence for contexts as defined above, is an equivalence relation on the set of contexts. We use $[\mathbb{C}]_A$ for the equivalence class containing context $\mathbb{C}$. For example the term

$\odot$ ⟨ Nt : Network | soup : ($S_1 \leftarrow 10$) ⟩
⟨ S_1 : Server | queue : nil ⟩ ⟨ S_2 : Server | queue : ($S_2 \leftarrow 5$) ⟩

is a context.

Definition 6 (R/A-matches). *Given $[u]_A \in Can_{\Sigma,E/A}$, its R/A-*matches *are triples $([\mathbb{C}]_A, r, [\theta]_A)$, where if $r \in R$ is a rule*

$$l : t(\overrightarrow{x}) \to t'(\overrightarrow{x}, \overrightarrow{y}) \text{ if } C(\overrightarrow{x}) \text{ with probability } \pi_r(\overrightarrow{x})$$

then $[\theta]_A \in [\![C]\!]$, that is $[\theta]_A$ satisfies condition C and $[u]_A = [\mathbb{C}(\odot \leftarrow \theta(t))]_A$, so $[u]_A$ is the same as θ applied to the term $t(\overrightarrow{x})$ and placed in the context.

Consider the canonical-term

$\langle$ H : Client | sent : 3, svc_1 : 1, svc_2 : 0 $\rangle$ $\langle$ Nt : Network | soup : $(S_1 \leftarrow 10)$ $\rangle$
$\langle$ S_1 : Server | queue : nil $\rangle$ $\langle$ S_2 : Server | queue : $(S_2 \leftarrow 5)$ $\rangle$

Looking at the code in Figure 1, one of the R/A-matches for the equivalence class of the above term is the triple $([\mathbb{C}]_A, req, [\theta]_A)$ such that

$\mathbb{C} = \odot$ $\langle$ Nt : Network | soup : $(S_1 \leftarrow 10)$ $\rangle$
$\langle$ S_1 : Server | queue : nil $\rangle$ $\langle$ S_2 : Server | queue : $(S_2 \leftarrow 5)$ $\rangle$

and θ is such that

$$\theta(Cl) = H, \theta(N) = 3, \theta(m_1) = 1, \theta(m_2) = 0.$$

Definition 7 (E/A-canonical one-step $\mathcal{R}$-rewrite). *An E/A-canonical one-step $\mathcal{R}$-rewrite is a labelled transition of the form,*

$$[u]_A \xrightarrow{([\mathbb{C}]_A, r, [\theta]_A, [\rho]_A)} [v]_A$$

where

1. $[u]_A, [v]_A \in Can_{\Sigma, E/A}$
2. $([\mathbb{C}]_A, r, [\theta]_A)$ *is an R/A-match of* $[u]_A$
3. $[\rho]_A \in CanGSubst_{E/A}(\overrightarrow{y})$
4. $[v]_A = [\mathbb{C}(\odot \leftarrow t'(\theta(\overrightarrow{x}), \rho(\overrightarrow{y})))]_A$, *where* $\{\theta, \rho\}|_{\overrightarrow{x}} = \theta$ *and* $\{\theta, \rho\}|_{\overrightarrow{y}} = \rho$.

We associate the probability $\pi_r(\alpha_r(\rho))$ with this transition. We can now see why the $\mathcal{F}_r$ cover α_r was needed. The nondeterminism associated with the choice of the R/A-match must be removed in order to associate a probability space over the space of computations (which are infinite sequences of canonical one step $\mathcal{R}$-rewrites). The nondeterminism is removed by what is called an *adversary* of the system, which defines a probability distribution over the set of R/A-matches. In [9] a probability space is associated over the set of computation paths. To do this, an adversary for the system is fixed. We have also shown in [9] that probabilistic rewrite theories have great expressive power. They can express various known models of probabilistic systems like Continuous Time Markov Chains [8], Markov Decision Processes [10] and even Generalized Semi Markov Processes [7]. We also propose a temporal logic, to express properties of interest in probabilistic systems. The details can be found in [9].

Probabilistic rewrite theories can be used to model probabilistic actor systems [1]. Actors, which are inherently asynchronous, can be modelled naturally using object based rewriting. In probabilistic actor systems we may be interested in modelling message delay distributions among other probabilistic entities. However because time acts as a global synchronization parameter the natural encoding using objects, computing by their own rewrite rules is insufficient. The technique of *delayed* messages helps us to correctly encode time in actors. Actor failures and message drops can also be encoded. Due to space constraints we do not indicate our encoding in this paper. The file at http://maude.cs.uiuc.edu/pmaude/at.maude presents our technique.

A special subclass of **PRwTh**, called *finitary probabilistic rewrite theories*, while fairly general, are easier to implement. We describe them below.

4 Finitary Probabilistic Rewrite Theories

Observe that there are two kinds of nondeterministic choice involved in rewriting. First, the selection of the rule and second, the exact substitution-context pair. Instead of having to think of nondeterminism from both these sources, it is easier to think in terms of rewrite rules with same left-hand side term as representing the computation of some part of the system, say an object, and thus representing one nondeterministic choice. Of course the substitution and context also have to be chosen to fix a nondeterministic choice. After nondeterministically selecting a rewrite rule, instantiated with a given substitution in a given context, different probabilistic choices arise for different right-hand sides of rules having the same left-hand side as that of the chosen rule, and which can apply in the chosen context with the chosen substitution. To assign probabilities, we assign *rate* functions to rules with the same left-hand side and normalize them. The rates, which depend on the chosen substitution, correspond to the *frequency* with which the RHS's are selected. Moreover, not having new variables on the right-hand sides of rules makes the implementation much simpler. Such theories are called *finitary* probabilistic rewrite theories. We define them formally below.

Definition 8 (Finitary probabilistic rewrite theory). A *finitary probabilistic rewrite theory* is a 4-tuple $\mathcal{R}_f = (\Sigma, E \cup A, R, \gamma)$, with $(\Sigma, E \cup A, R)$ a rewrite theory and $\gamma : R \to T_{\Sigma,E/A,PosRat}(X)$ a function associating to each rewrite rule in R a term $\gamma(r) \in T_{\Sigma,E/A,PosRat}(X)$, with some variables from the set X, and of sort $PosRat$, where $PosRat$ is a sort in $(\Sigma, E \cup A)$ corresponding to the positive rationals. The term $\gamma(r)$ represents the *rate* function associated with rule $r \in R$. If $l : t(\overrightarrow{x}) \to t'(\overrightarrow{x})$ *if* $C(\overrightarrow{x})$ is a rule in R involving variables $\overrightarrow{x}$, then γ maps the rule to a term of the form $\gamma_r(\overrightarrow{x})$ possibly involving some of the variables in $\overrightarrow{x}$. We then use the notation

$$l : t(\overrightarrow{x}) \to t'(\overrightarrow{x}) \ \textit{if} \ C(\overrightarrow{x}) \ [\ \textit{rate} \ \gamma_r(\overrightarrow{x})\ ]$$

for the γ-annotated rule. Notice that t' does not have any new variables. Thus, all variables in t' are also variables in t. Furthermore, we require that all rules labelled by l have the *same* left-hand side and are of the form

$$\begin{array}{l} l : t \to t'_1 \ \textit{if} \ C_1 \ [\ \textit{rate} \ \gamma_{r_1}(\overrightarrow{x})\ ] \\ \qquad \cdots \\ l : t \to t'_n \ \textit{if} \ C_n \ [\ \textit{rate} \ \gamma_{r_n}(\overrightarrow{x})\ ] \end{array} \tag{1}$$

where

1. $\overrightarrow{x} = \mathit{fvars}(t) \supseteq \bigcup_{1\leq i\leq n} \mathit{fvars}(t'_i) \cup \mathit{fvars}(C_i)$, that is the terms t'_i and the conditions C_i do not have any variables other than $\overrightarrow{x}$, the set of variables in t.
2. C_i is of the form $(\bigwedge_j u_{ij} = v_{ij}) \wedge (\bigwedge_k w_{ik} : s_{ik})$, that is, condition C_i is a conjunction of equations and memberships.[1]

We denote the class of finitary probabilistic rewrite theories by **FPRTh**.

4.1 Semantics of Finitary Probabilistic Rewrite Theories

Given a finitary probabilistic rewrite theory $\mathcal{R}_f = (\Sigma, E \cup A, R, \gamma)$, we can express it as a probabilistic rewrite theory $\mathcal{R}^\bullet_f$, by defining a map $F_{\mathcal{R}} : \mathcal{R}_f \mapsto \mathcal{R}^\bullet_f$, with $\mathcal{R}^\bullet_f = (\Sigma^\bullet, E^\bullet \cup A, R^\bullet, \pi^\bullet)$ and $(\Sigma, E \cup A) \subseteq (\Sigma^\bullet, E^\bullet \cup A)$, in the following way. We encode each group of rules in R with label l of the form 1 above by a single probabilistic rewrite rule[2]

$$t(\overrightarrow{x}) \rightarrow \ \mathit{proj}(i, (t'_1(\overrightarrow{x}), \ldots, t'_n(\overrightarrow{x}))) \ \mathit{if} \ \widetilde{C}_1(\overrightarrow{x}) \ \mathit{or} \ \ldots \mathit{or} \ \widetilde{C}_n(\overrightarrow{x}) = \mathit{true} \\ \mathit{with\ probability}\ \pi_r(\overrightarrow{x})$$

in $R^\bullet$. Corresponding to each such rule, we add to $\Sigma^\bullet$ the sort $[1:n]$, with constants $1, \ldots, n :\rightarrow [1:n]$, and the projection operator $\mathit{proj} : \ [1:n]\ k \ldots k \rightarrow k$. We also add to $E^\bullet$ the equations $\mathit{proj}(i, t_1, \ldots, t_n) = t_i$ for each $i \in \{1, \ldots, n\}$. Note that the only new variable on the righthand side is i, and therefore $\mathit{CanGSubst}_{E/A}(i) \cong \{1, \ldots, n\}$. We consider the σ-algebra $\mathcal{P}(\{1, \ldots, n\})$ on $\{1, \ldots, n\}$. Then π_r is a function

$$\pi_r : \ [\![C]\!] \rightarrow \mathit{PFun}(\{1, \ldots, n\}, \mathcal{P}(\{1, \ldots, n\}))$$

defined as follows. If θ is such that $\widetilde{C}_1(\theta(\overrightarrow{x})) \ \mathit{or} \ \ldots \ \mathit{or} \ \widetilde{C}_n(\theta(\overrightarrow{x})) = \mathit{true}$, then $\pi_\theta = \pi_r(\theta)$ defined as

$$\pi_\theta(\{i\}) = \frac{?\gamma_{r_i}(\theta(\overrightarrow{x}))}{?\gamma_{r_1}(\theta(\overrightarrow{x})) + ?\gamma_{r_2}(\theta(\overrightarrow{x})) + \cdots + ?\gamma_{r_n}(\theta(\overrightarrow{x}))}$$

where, if $\widetilde{C}_i(\theta(\overrightarrow{x})) = \mathit{true}$, then $?\gamma_{r_i}(\theta(\overrightarrow{x})) = \gamma_{r_i}(\theta(\overrightarrow{x}))$ and $?\gamma_{r_i}(\theta(\overrightarrow{x})) = 0$ otherwise. The semantics of $\mathcal{R}_f$ computations is now defined in terms of its associated theory $\mathcal{R}^\bullet_f$ in the standard way, by choosing the singleton $\mathcal{F}$-cover $\alpha_r : \ \{1, \ldots, n\} \rightarrow \mathcal{P}(\{1, \ldots, n\})$ mapping each i to $\{i\}$.

[1] The requirement $\mathit{fvars}(C_i) \subseteq \mathit{fvars}(t)$ can be relaxed by allowing new variables in C_i to be introduced in "matching equations" in the sense of [4]. Then these new variables can also appear in t'_i.

[2] By the assumption that $(\Sigma, E \cup A)$ is confluent, sort-decreasing, and terminating modulo A, and by a metatheorem of Bergstra and Tucker, any condition C of the form $(\bigwedge_i u_i = v_i \wedge \bigwedge_j w_j : s_j)$ can be replaced in an appropriate protecting enrichment $(\widetilde{\Sigma}, \widetilde{E} \cup A)$ of $(\Sigma, E \cup A)$ by a semantically equivalent Boolean condition $\widetilde{C} = \mathit{true}$.

5 The PMaude Tool

We have developed an interpreter called **PMaude**, which provides a framework for specification and execution of finitary probabilistic rewrite theories. The **PMaude** interpreter has been built on top of Maude 2.0 [4,3] using the Full-Maude library [6]. We describe below how a finitary probabilistic rewrite theory is specified in our implemented framework and discuss some of the implementation details.

Consider a finitary probabilistic rewrite theory with k distinct rewrite labels and with n_i rewrite rules for the i^{th} distinct label, for $i = 1, 2, \ldots, k$.

$$
\begin{array}{l}
l_1 : t_1 \rightarrow t'_{11} \quad \textit{if } C_{11} \ [\ \textit{rate } \gamma_{11}(\overrightarrow{x}) \quad] \\
\qquad \ldots \\
l_1 : t_1 \rightarrow t'_{1n_1} \ \textit{if } C_{1n_1} \ [\ \textit{rate } \gamma_{1n_1}(\overrightarrow{x}) \ ] \\
\qquad \ldots \\
l_k : t_k \rightarrow t'_{k1} \ \textit{if } C_{k1} \quad [\ \textit{rate } \gamma_{k1}(\overrightarrow{x}) \quad] \\
\qquad \ldots \\
l_k : t_k \rightarrow t'_{kn_k} \ \textit{if } C_{kn_k} \ [\ \textit{rate } \gamma_{kn_k}(\overrightarrow{x}) \ ]
\end{array}
$$

At one level we want all rewrite rules in the specification to have *distinct* labels, so that we have low level control over these rules, while at the conceptual level, groups of rules must have the same label. We achieve this by giving two labels: one, common to a group and corresponding to the group's label l at the beginning, and another, unique for each rule, at the end. The above finitary probabilistic rewrite theory can be specified as follows in **PMaude**.

$$
\begin{array}{l}
\textit{pmod FINITARY-EXAMPLE is} \\
\quad \textit{cprl } [l_1] : t_1 \Rightarrow t'_{11} \ \textit{if } C_{11} \quad [\textit{rate } \gamma_{11}(x_1, \ldots) \quad] \ [\textit{metadata } \text{“}l_{11} \ \ldots\text{”} \ ] \ . \\
\quad \ldots \\
\quad \textit{cprl } [l_1] : t_1 \Rightarrow t'_{1n_1} \textit{if } C_{1n_1} \ [\textit{rate } \gamma_{1n_1}(x_1, \ldots) \ ] \ [\textit{metadata } \text{“}l_{1n_1} \ \ldots\text{”} \ ] \ . \\
\quad \ldots \\
\quad \textit{cprl } [l_k] : t_k \Rightarrow t'_{k1} \ \textit{if } C_{k1} \quad [\textit{rate } \gamma_{k1}(x_1, \ldots) \quad] \ [\textit{metadata } \text{“}l_{k1} \ \ldots\text{”} \ ] \ . \\
\quad \ldots \\
\quad \textit{cprl } [l_k] : t_k \Rightarrow t'_{kn_k} \textit{if } C_{kn_k} \ [\textit{rate } \gamma_{kn_k}(x_1, \ldots) \ ] \ [\textit{metadata } \text{“}l_{kn_k} \ \ldots\text{”} \ ] \ . \\
\textit{endpm}
\end{array}
$$

User input and output are supported as in Full Maude using the `LOOP-MODE` module. **PMaude** extends the Full Maude functions for parsing modules and any terms entered later by the user for rewriting purposes. Currently **PMaude** supports four user commands. Two of these are low level commands used to change seeds of pseudo-random generators. We shall not describe the implementation of those two commands here. The other two commands are rewrite commands. Their syntax is as follows:

(*prew* t .)
(*prew*-$[n]$ t .)

The default module M in which these commands are interpreted is the last read probabilistic module. The *prew* command is an instruction to the interpreter to probabilistically rewrite the term t in the default module M, till no further rewrites are possible. Notice that this command may fail to terminate. The *prew*-$[n]$ command takes a natural number n specifying the maximum number of probabilistic rewrites to perform on the term t. This command always terminates in at most n steps of rewriting. Both commands report the final term (if *prew* terminates).

The implementation of these commands is as follows. When the interpreter is given one of these commands, all possible one-step rewrites for t in the default module M are computed. Out of all possible groups $l_1, l_2, .., l_k$ in which *some* rewrite rule applies, one is chosen, *uniformly at random*. For the chosen group l_i, all the rewrite rules $l_{i1}, l_{i2}, .., l_{in_i}$ associated with l_i, are guaranteed to have the same left-hand side $t_i(x_1, x_2, ..)$. From all possible canonical substitution, context pairs $([\theta]_A, [\mathbb{C}]_A)$ for the variables x_j, representing successful matches of $t_i(x_1, x_2, ..)$ with the given term t, that is, matches that also satisfy one of the conditions C_{ij}, one of the matches is chosen *uniformly at random*. The two steps above also define the exact adversary we associate to a given finitary probabilistic rewrite theory in our implementation. If there are m groups, $l_{i_1}, \ldots, l_{i_m}$, in which *some* rule applies and v_j matches in total for group l_{i_j} then the adversary chooses a match in group l_{i_j} with probability $\frac{1}{mv_j}$. To choose the exact rewrite rule l_{ij} to apply, use of the rate functions is made. The values of the various rates γ_{ip} are calculated for those rules l_{ip} such that $[\theta]_A$ satisfies the condition of the rule l_{ip}. Then these rates are normalized and the choice of the rule l_{ij} is made *probabilistically*, based on the calculated rates. This rewrite rule is then applied to the term t, in the chosen context with the chosen substitution. If the interpreter finds no successful matches for a given term, or if it has completed the maximum number of rewrites specified, it immediately reports that term as the answer. Since rates can depend on the substitution, this allows users to specify systems where probabilities are determined by the state.

PMaude can be used as a simulator for finitary probabilistic rewrite theories. The programmer must supply the system specification as a **PMaude** module and a start term to rewrite. To obtain different results the seeds for the random number generators must be changed at each invocation. This can be done by using a scripting language to call the interpreter repeatedly but with different seeds before each execution.

We have specified the client-server example discussed in Section 3 in **PMaude** with the following parameters: The client only sends two kinds of packets, loads 5 and 10, with equal probability. The request messages for S_1, S_2 are dropped with probabilities 2/7 and 1/6 respectively, while acknowledgement messages for S_1, S_2 are dropped with probabilities 1/6, 1/11 respectively. We also chose S_1 to drop processing of a request with probability 1/7 as opposed to 3/23 when its load was at least 100, while for S_2 the load limit was 40 but the probabilities of dropping requests were 1/7 and 3/19. We performed some simulations and, after a number of runs, we computed the ratio $(svc_1 + svc_2)/sent$ as a measure of the quality of service for the client. Simulations showed that among static policies, namely those where the client did not adapt sending probabilities on the fly, that of *sending twice as often* to server S_2 than to S_1 was better than most others.

The complete code for the **PMaude** interpreter, as well as several other example files, can be found at http:/maude.cs.uiuc.edu/pmaude/pmaude.html.

6 Conclusions and Future Work

Probabilistic rewrite theories provide a general semantic framework supporting high level probabilistic specification of systems; in fact, we have shown how various well known probabilistic models can be expressed in our framework [9]. The present work shows how our framework applies to concurrent object based systems. For a fairly general subclass, namely finitary probabilistic rewrite theories, we have implemented a simulator **PMaude** and have exercised it on some simple examples. We are currently carrying out more case studies. We have also identified several aspects of the theory and the **PMaude** tool that need further development.

On the more theoretical side, we feel research is needed in three areas. First, it is important to develop a general model of probabilistic systems with *concurrent* probabilistic actions, as opposed to the current *interleaving* semantics. Second, deductive and analytic methods for property verification of probabilistic systems, based on our current framework is an important open problem. Algorithms to translate appropriate subclasses to appropriate representations enabling use of existing model checkers, should be developed and implemented.

Third, we think that allowing the probability function π_r to depend not only on the substitution, but also on the context would give us more modelling power. Specifically, it would enable us to represent applications where the probability distributions of certain variables, such as message delays, depend on functions of the entire state of the system, for example, on the congestion in the network. Such situations can also be modelled in our current framework but at the expense of using rewrite rules at the *top-level*, whose substitutions capture global system parameters. Such rules can modify the entire system at once as opposed to just modifying local fragments, but at the cost of violating the modularity principle of concurrent objects or actors.

On the implementation side, an extension of the **PMaude** framework to enable specification of more general classes of probabilistic rewrite theories and adversaries is highly desirable. This will allow the generation of simulation traces for the system under consideration and can be used as a tool to implement the model independent Monte-Carlo simulation and acceptance sampling methods for probabilistic validation of properties [14]. As an application of our theory, we believe that it will be fruitful to model networked embedded systems, where apart from time, there are other continuous state variables, such as battery power or temperature, whose behavior may be stochastic. Moreover, the properties of interest are often a statistical aggregation of many observations.

7 Acknowledgement

The work is supported in part by the Defense Advanced Research Projects Agency (the DARPA IPTO TASK Program, contract number F30602-00-2-0586 and the DARPA IXO NEST Program, contract number F33615-01-C-1907) and the ONR Grant N00014-02-1-0715. We would like to thank Wooyoung Kim for reading a previous version of this paper and giving us valuable feedback and Joost-Pieter Katoen for very helpful discussions and pointers to references.

References

1. G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, 1997.
2. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1–2):35–132, 2000.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Towards maude 2.0. In K. Futatsugi, editor, *Electronic Notes in Theoretical Computer Science*, volume 36. Elsevier, 2001.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude 2.0 Manual, Version 1.0*, june 2003. `http://maude.cs.uiuc.edu/manual/maude-manual.pdf`.
6. F. Durán and J. Meseguer. Parameterized theories and views in full maude 2.0. In K. Futatsugi, editor, *Electronic Notes in Theoretical Computer Science*, volume 36. Elsevier, 2001.
7. P. Glynn. The role of generalized semi-Markov processes in simulation output analysis, 1983.
8. J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. *Lecture Notes in Computer Science*, 2165, 2001.
9. N. Kumar, K. Sen, J. Meseguer, and G. Agha. Probabilistic rewrite theories: Unifying models, logics and tools. Technical Report UIUCDCS-R-2003-2347, University of Illinois at Urbana-Champaign, May 2003.
10. M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker, 2002.
11. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.
12. J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.
13. J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, pages 18–61. Springer LNCS 1376, 1998.
14. H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235, Copenhagen, Denmark, July 2002. Springer.