

Probabilistic Rewrite Theories

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

We propose probabilistic rewrite theories as a general semantic framework supporting high-level specification of probabilistic systems that can be massively concurrent. We give the definition and semantics of probabilistic rewrite theories and discuss the mappings between different classes of theories and models. We then define the semantics of probabilistic temporal formulae for a given probabilistic rewrite theory. We explain how real-time probabilistic systems whose time is discrete can be expressed as probabilistic rewrite theories without any extension. Finally we give our design ideas for PMaude, an implementation of probabilistic rewrite theories on top of Maude 2.0. We shall report a running prototype of PMaude in the final version of the paper.

1 Introduction

We propose a natural extension of rewrite theories, called *probabilistic rewrite theories*, as a general high-level formalism to specify probabilistic systems of the kind used in performance and reliability modelling, and in distributed randomized algorithms. We are particularly interested in specifying next-generation large networks of embedded hybrid systems. Such systems are distributed, highly asynchronous, have essential real-time features, and have unreliable communication media and faults. The behavior of these systems can be modelled probabilistically by replacing nondeterminism due to unreliability and faults by probabilities.

There has been considerable research on models, logics, and model checking for probabilistic systems. Work in this area includes, among others, probabilistic process algebra approaches such as [12, 11, 25], probabilistic Petri nets [18, 19], the hybrid I/O automata by Lynch [15], probabilistic automata by Segala and Lynch [27], and probabilistic nondeterministic systems by Bianco and de Alfaro [8]. Theories related to timed automata were developed by Alur and Dill [3], and Alur and Dill [26] presents a survey of real-time logics, including RTL and MTL. Marta Kwiatkowska and others combined these works to develop the theory of probabilistic timed automata and tools related to verification of such systems [14]. The logics underlying such tools include Computational Stochastic Logic (CSL), Probabilistic Computational Tree Logic (PCTL) and Probabilistic Timed Computation Tree Logic (PTCTL). The PRISM tool [13] of Kwiatowska et al. has been developed to model check systems based on these models for the above mentioned logics. It implements efficient symbolic model checking techniques as proposed in [5].

The situation with models at present is that, either they build in some existing model of concurrency, such as a variant of CCS, Petri nets, or I/O automata, or they are low-level automaton like models such as Markov chains or probabilistic nondeterministic systems (PNSs). In this context,

our proposed probabilistic rewrite theories can serve as a flexible *semantic framework*, which can express all the above models and supports a high-level specification approach which:

- can explicitly support a wide range of concurrency models, including *asynchronous object models*;
- allows the definition of *parameterized* probabilistic transitions, whose probability may depend on their actual parameters;
- supports *execution* and symbolic simulation of specifications in languages such as the PMAude language proposed in this paper;
- can be naturally integrated with *probabilistic temporal logics* to express high-level properties, and to model check them for finite state systems, as explained in this paper;
- supports algebraic definition of general data types and a rich language of state predicates based on the underlying equational logic of the given rewrite theory.

The rest of the paper is organized as follows. Section 2 summarizes background concepts on PNSs and probabilistic temporal logics. Section 4 defines probabilistic rewrite theories and their semantics, presents a randomized distributed algorithm example, discusses mappings between different classes of theories and models, and explains how the semantics of probabilistic temporal formulae is defined for a given probabilistic rewrite theory. Section 5 explains how real-time probabilistic systems whose time is discrete can be expressed in our framework without any need for further extensions and illustrate such systems with an example. We finish the paper with some conclusions and directions for future work in Section 7.

2 Background

2.1 Probabilistic-Nondeterministic Systems (PNS)

Probabilistic nondeterministic systems (also known as *probabilistic automata*), as described in [8, 24, 23, 27], are widely used to model systems in which probabilistic and nondeterministic behavior coexists. The motivation for such co-existence is that probabilities may not be known or estimable for all types of transitions. Following the approach of [8] with the slight variant of ignoring the initial state, we give a formal definition of PNS. We first introduce the definition of *next-state probability distribution*.

Definition 1 (Next-state probability distribution) If S is the state space of a system, a *next-state probability distribution* is a function $p : S \mapsto [0, 1]$ such that $\sum_{s \in S} p(s) = 1$. For $s \in S$, $p(s)$ represents the probability of making a direct transition to s from the current state.

A PNS is defined as:

Definition 2 (PNS) A PNS is a *triple* $\Pi = (S, V, \tau)$, where,

- S is the denumerable or finite state space of the system,

- $V : S \rightarrow \mathcal{P}(AP)$ is a labelling function that associates with each $s \in S$ the set $V(s) \subseteq AP$ of atomic propositions that are true in s , and
- τ is a function which associates with each $s \in S$ a finite set $\tau(s) = \{p_1^s, \dots, p_{k_s}^s\}$ of next-state probability distributions from s .

The next state of s in a computation is chosen in two steps:

1. A next-state probability distribution $p_i^s \in \tau(s)$ is chosen nondeterministically from the set $\tau(s)$,
2. Then, a successor state $t \in S$ is chosen with probability $p_i^s(t)$.

2.2 Temporal Logics for Probabilistic Systems

Various logics can be used for the specification of temporal and probabilistic properties. Such logics are in general adequate extensions of some temporal logic with a *probability* operator that specifies probability values of formulas. The semantics of formulas in these extended logics is used to express properties such as “some *bad event* happening with probability at most p ” or “that some safety or liveness condition is satisfied with sufficiently large probability”. The *linear time* framework uses standard linear time temporal logic with an interpretation over the states of a probabilistic system. The truth value of a formula φ at state s is a value $p_s(\varphi)$ in the interval $[0,1]$ and can be viewed as the probability that the formula φ holds when the system starts in state s .

Among the branching time logics we have various logics such as pCTL/pCTL* [8] (probabilistic CTL/CTL* for *concurrent Markov chains*), PCTL/PCTL* [27] (probabilistic CTL/CTL* for *sequential Markov chains*), TPCTL [10] (Timed probabilistic CTL), PBTL [6] (Probabilistic branching time logic) in use. We shall present the syntax of PCTL* and pCTL* and will briefly go over their semantics. For more detailed information the reader is encouraged to follow the references [8, 27, 6].

2.2.1 PCTL* and pCTL*

These logics are used to express properties for sequential and concurrent Markov chains. They can express properties like “This process will enter the critical section within the next three steps with probability at least 0”. They make use of *state formulas* and *path formulas*. Path formulas make statements about the execution paths, whereas the state formulas make statements that express the branching time nature and the various possibilities in a state.

Syntax: The syntax of PCTL* formulas can be specified by the following production rules. In what follows ϕ denotes a state formula and φ denotes a path formula.

$$\begin{aligned}\phi &= \mathbf{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbb{P}_{\bowtie p}(\varphi) \\ \varphi &= \phi \mid X\varphi \mid \varphi \mathcal{U} \varphi \mid \varphi \mathcal{U}^{\leq k} \varphi \mid \varphi \wedge \varphi \mid \neg\varphi\end{aligned}$$

In the above formulas the X , \mathcal{U} have their usual meanings of the operators next, until, respectively, while a represents an atomic proposition which is drawn from a set AP of atomic propositions, and the probabilistic operator $\mathbb{P}_{\bowtie p}(\varphi)$ is used to express the *quantity* of paths satisfying a given formula φ , where \bowtie stands for $<$, \leq , \geq , $>$, and $p \in [0, 1]$.

The syntax of pCTL* formulas can be specified by the following production rules.

$$\begin{aligned}\phi &= \mathbf{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{A}\phi \mid \mathbf{E}\phi \mid \mathbb{P}_{\geq p}(\varphi) \\ \varphi &= \varphi \mathcal{U} \varphi \mid \square\varphi \mid \diamond\varphi\end{aligned}$$

Here \mathbf{A} , \mathbf{E} , \square , \diamond have the usual meanings of the operators for all paths, there exists a path, always, and eventually, respectively.

Semantics: The semantics of the above formulas defines the satisfaction relation $\Pi, s \models \phi$ stating that the state formula ϕ holds in Π at state s . This definition uses the probability space (X, A) where X is the set of infinite execution paths starting at s and A is the σ -algebra on X generated by the sets

$$\{\sigma \in X \mid \sigma_0 = s, \sigma_1 = s_1, \dots, \sigma_n = s_n\}.$$

In the above definition the subscript i on σ denotes the i th state of the path σ . In the case of a PNS, an *adversary* decides at each state of a path σ which of the nondeterministic probability distributions to choose. Note that a sequential Markov chain is a special case of PNS. In such sequential systems there is just a trivial adversary who chooses the (only) nondeterministic choice with probability 1 at each step of the path. In the case of PCTL* the formula means that the (unique) measure of the set of paths satisfying φ is greater than p . For the complete semantics of pCTL* and PCTL* formulas the user is encouraged to follow the references [8, 27, 6].

3 General notion of probabilistic rewrite theories

Definition 3 (*E/A*-canonical ground substitution) An *E/A*-canonical ground substitution is a substitution $\theta : \vec{X} \rightarrow T_\Sigma$ such that $\forall x \in \vec{X} [\theta(x)]_A \in \text{Can}_{\Sigma, E/A}$.

Definition 4 (*A*-equivalent substitution) Given two *E/A*-canonical ground substitution $\theta, \rho : \vec{X} \rightarrow T_\Sigma$, they are called *A*-equivalent if and only if $\forall x \in \vec{X} [\theta(x)]_A = [\rho(x)]_A$. We use $[\theta]_A$ for equivalence classes.

Let

$$\text{CanGSubst}_{E/A}(\vec{X}) = \{[\theta]_A \mid \theta : \vec{X} \rightarrow T_\Sigma \text{ is an } E/A\text{-canonical ground substitution}\}$$

Definition 5 (General probabilistic rewrite theory) A *general probabilistic rewrite theory* is a quintuple $\mathcal{R} = (\Sigma, E \cup A, L, R, \Pi)$, with $(\Sigma, E \cup A, L, R)$ a rewrite theory and for a rule $r \in R$, r is of the form

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

where

1. $l \in L$ is the label of the rule r
2. \vec{x} is the set of all variables in t and $\vec{x} \cup \vec{y}$ is the set of variables in t' . Let $\vec{x} = x_1 : s_1 \dots x_n : s_n$ and $\vec{y} = y_1 : u_1 \dots y_m : u_m$,
3. C is of the form $(\bigwedge_j u_j = u'_j) \wedge (\bigwedge_k v_k : s_k)$, that is, condition C is a conjunction of equations and memberships.

4. $\pi_r \in \Pi : \llbracket C \rrbracket \rightarrow \mathbf{PDist}(CanGSubst_{E/A}(\vec{y}))$, where

$$\llbracket C \rrbracket = \{[\mu]_A \in CanGSubst_{E/A}(\vec{x}) \mid E \cup A \vdash (\forall \emptyset) \mu(C)\}$$

and \mathbf{PDist} is the set of all probability distribution functions on $CanGSubst_{E/A}(\vec{y})$.

We denote the class of general probabilistic rewrite theories as \mathbf{GPRTh} .

3.1 Semantics of general probabilistic rewrite theory

We assume that for $\mathcal{R} = (\Sigma, E \cup A, L, R, \Pi)$

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

Definition 6 (Context) A context \mathbb{C} is a Σ -term with a single occurrence of a single variable, \odot , called the *hole*. Given two contexts \mathbb{C}, \mathbb{C}' , they are called equivalent modulo A if and only if $A \vdash (\forall \emptyset) \mathbb{C} = \mathbb{C}'$. We use $\llbracket \mathbb{C} \rrbracket_A$ for such equivalence classes.

Definition 7 (R/A -matches) Given $[u]_A \in Can_{\Sigma, E/A}$ its R/A -matches are tuples,

$$([\mathbb{C}]_A, r, [\theta]_A),$$

where, if r is the rule

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

then $[\theta]_A \in \llbracket C \rrbracket$ and $[u]_A = \llbracket \mathbb{C}(\odot \leftarrow \theta(t)) \rrbracket_A$.

The probabilistic rewrite happens in $Can_{\Sigma, E/A}$ as follows:

Let

$$[u]_A \in Can_{\Sigma, E/A}$$

- consider the set of all R/A -matches of $[u]_A$.
- Pick one of them, say $([\mathbb{C}]_A, r, [\theta]_A)$, nondeterministically. Let r be the rule

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

- Pick a particular $[\rho]_A \in CanGSubst_{E/A}(\vec{y})$ according to distribution $\pi_r([\theta]_A)$ and get one step of probabilistic rewriting of the form

$$[u]_A \rightarrow Can_{E/A}(\llbracket \mathbb{C}(\odot \leftarrow \{\theta, \rho\}(t'(\vec{x}, \vec{y}))) \rrbracket)$$

3.2 PRwTh is a subset of GPRTh

Given a probabilistic rewrite theory $\mathcal{R} = (\Sigma, E, L, R, \rho)$, we can express it as a generalized probabilistic theory, say \mathcal{G} , in the following way. We encode each group of rule in R with label l of the form

$$\begin{aligned} l : t(\vec{x}) &\rightarrow t'_1(\vec{x}) \text{ if } C_1(\vec{x}) \text{ [rate } r_1(\vec{x})] \\ &\dots \\ l : t(\vec{x}) &\rightarrow t'_n(\vec{x}) \text{ if } C_n(\vec{x}) \text{ [rate } r_n(\vec{x})] \end{aligned}$$

to a single rewrite rule

$$t(\vec{x}) \rightarrow \text{projection}(y, (t'_1(\vec{x}), \dots, t'_n(\vec{x}))) \text{ if } C_1(\vec{x}) \vee \dots \vee C_n(\vec{x}) == \text{ true with probability } \pi$$

4 Probabilistic Rewrite Theories

Definition 8 (Probabilistic rewrite theory) A *probabilistic rewrite theory* is a quintuple $\mathcal{R} = (\Sigma, E, L, R, \rho)$, with (Σ, E, L, R) a rewrite theory and $\rho : R \rightarrow T_{\Sigma/E}(X)_{\text{PosReal}}$ a function associating each rewrite rule in R with a term of sort **PosReal**, where **PosReal** is a sort in (Σ, E) corresponding to the positive fragment of a computable subfield of the real numbers. This term represents the rate associated with the rule R . Let $l : t \rightarrow t'$ if C be a rule in R and ρ maps the rule to the term $r(\vec{x})$. Then we use the notation

$$l : t \rightarrow t' \text{ if } C \text{ [rate } r(\vec{x})]$$

for the rule. Furthermore, we require that for each label $l \in L$ all rules labelled by l have the same lefthand side and are of the form

$$\begin{aligned} l : t &\rightarrow t'_1 \text{ if } C_1 \text{ [rate } r_1(\vec{x})] \\ &\dots \\ l : t &\rightarrow t'_n \text{ if } C_n \text{ [rate } r_n(\vec{x})] \end{aligned}$$

where

1. \vec{x} is the set of all free variables in t, t'_1, \dots, t'_n , i.e. $\vec{x} = \text{fvars}(t) \cup \bigcup_{i \in [1, n]} \text{fvars}(t'_i)$,
2. C_i is of the form $(\bigwedge_j u_j = u'_j) \wedge (\bigwedge_k v_k : s_k)$, that is, condition C_i is a conjunction of equations and memberships.

We denote the class of probabilistic rewrite theories as **PRwTh**.

4.1 Semantics of a Probabilistic Rewrite Theory

Definition 9 (Match decompositions) Let (Σ, E) be an equational theory, and let $[u]$ be an equivalence class of term of the initial algebra $T_{\Sigma/E}$. By a *match decomposition* of $[u]$ we mean a triple, $(\mathbb{C}, t, \theta)_{[u]}$ with \mathbb{C} a Σ -term, called a *context*, with a single occurrence of a single variable, \odot , called the *hole*, with t a $\Sigma(X)$ -term, called the *pattern*, and with $\theta : X \rightarrow T_{\Sigma}$ a ground substitution with $\text{fvars}(t) \subseteq X$, such that,

$$E \vdash (\forall \emptyset) \mathbb{C}(\odot \leftarrow \theta(t)) = u.$$

An *equivalence relation* between match decompositions of $[u]$ is defined as follows:
 $(\mathbb{C}, t, \theta)_{[u]} \equiv (\mathbb{C}', t', \theta')_{[u]}$ iff:

1. $E \vdash (\forall \odot) \mathbb{C} = \mathbb{C}'$,
2. $t = t'$ (i.e syntactically identical),
3. $\text{dom}(\theta) = \text{dom}(\theta')$, and $(\forall x \in \text{dom}(\theta)) E \vdash (\forall \emptyset) \theta(x) = \theta'(x)$.

Let $[(\mathbb{C}, t, \theta)_{[u]}]$ represent the equivalence class of $(\mathbb{C}, t, \theta)_{[u]}$ modulo the equivalence relation defined above. For a term $[u] \in T_{\Sigma/E}$, let $\mathcal{D}_{[u]}^{\mathcal{R}}$ be the set of all match decompositions (modulo \equiv) of the form $(\mathbb{C}, t, \theta)_{[u]}$ with t a lefthand side in \mathcal{R} . Then the next-step rewrite takes place in two stages:

1. An element $[(\mathbb{C}, t, \theta)_{[u]}]$ is nondeterministically chosen from the set $\mathcal{D}_{[u]}^{\mathcal{R}}$ and then a label l whose lefthand side is t and which has some enabled rule for θ (see below) is picked nondeterministically,
2. Let

$$\begin{aligned} l : t &\rightarrow t'_{i_1} \text{ if } C_{i_1} \text{ [rate } r_{i_1}(\vec{x})] \\ &\dots \\ l : t &\rightarrow t'_{i_j} \text{ if } C_{i_j} \text{ [rate } r_{i_j}(\vec{x})] \\ &\dots \\ l : t &\rightarrow t'_{i_m} \text{ if } C_{i_m} \text{ [rate } r_{i_m}(\vec{x})] \end{aligned}$$

be the nonempty set of rewrite rules that are *enabled* by θ , i.e., such that for $1 \leq j \leq m$ the condition $\theta(C_{i_j})$ is true and $r_{i_j}(\theta(\vec{x})) > 0$. Then, in general, $\{\theta(t'_{i_1}), \dots, \theta(t'_{i_m})\}$ is *not* a set but a multiset. Let us group the repetitions in this multiset as families of the form

$$\{\theta(t'_{i_{q_1}}), \dots, \theta(t'_{i_{qn_q}})\}, \quad 1 \leq q \leq k.$$

Then for each q , $1 \leq q \leq k$, we have a *one-step transition* $[u] \rightarrow [\mathbb{C}(\odot \leftarrow \theta(t'_{i_{q_1}}))]$ with probability p_q defined as the sum of the rates in the family divided by the sum of all the rates in the multiset such that:

$$p_q = \frac{\sum_{v \in [1, n_q]} r_{i_{qv}}(\theta(\vec{x}))}{\sum_{k \in [1, m]} r_{i_k}(\theta(\vec{x}))}.$$

4.2 Example: A Randomized Byzantine Agreement Protocol

We illustrate probabilistic rewrite theories by specifying the randomized algorithm for *Byzantine agreement problem* introduced by Ben-Or [7] as a theory in **PRwTh**. Byzantine agreement requires a set of parties in a distributed environment to agree on a value, even if some of the parties are corrupted. The fundamental impossibility result [17] shows that there is no *deterministic* algorithm for achieving agreement in the asynchronous setting with benign failures. [7] overcomes this problem by using *randomization*. The protocol uses random assignments and hence its outcome is probabilistic. The protocol works in a completely asynchronous setting in presence of an adversary scheduler who knows all about the system. The result in [7] shows that if the number of faulty

processes, t , satisfies $5t < n$, where n is the total number of processes, then complete asynchronous agreement is possible with the randomized algorithm.

The basic assumptions are as follows. A set of n asynchronous processes wish to agree about a binary value. Each process P starts with a binary input x_P , and they all must decide on a common value. A process *decides* by sending a single message `decide(x_P)` to the observer. The processes communicate by means of messages. The whole system is represented by a *configuration* consisting of the processes and the messages sent but not yet delivered. We assume a totally asynchronous system; however, every message sent is eventually delivered.

We discuss below the specification of the randomized algorithm as a probabilistic rewrite theory, using Maude 2.0 syntax enriched with rate information. The rate information is provided as metadata associated with a rewrite rule. The entire specification is given in Appendix 9. In the protocol every process performs rounds of exchange of information. The current round of a process is stored in the attribute `round : MachineInt`. In a given round a process goes through four steps and the current step is represented by the attribute `step`. The value of x_P is stored in the attribute `val : Binary` of a process. The other auxiliary attributes of a process are used to keep track of the number of messages it received, the number of messages having value 0 or 1, a list of acquaintances of the process, etc.

```
class Process | val : Binary , step : Steps , round : MachineInt ,
  msgcnt : MachineInt , cnt0 : MachineInt , cnt1 : MachineInt ,
  cntD0 : MachineInt , cntD1 : MachineInt , flag : Flag , acq : OidSet .
```

In the first step of a given round r each process P broadcasts the value of x_P to all other processes (attribute `acq` contains the list of the other processes).

```
rl [send-step1] : < P : Process | val : X , step : 1 , round : R , acq : A >
  => < P : Process | step : 2 > bcast(A,m1(R,X)) [metadata "rate: 1"] .
```

In the second step, P waits till it receives $n - t$ (n being number of processes) messages of type `m1(R,*)`.

```
cr1 [receive-step2] : (P <- m1(R,X)) < P : Process | step : 2 ,
  round : R , msgcnt : N , cnt0 : Z , cnt1 : 0 >
  => if X == 0 then < P : Process | msgcnt : N+1 , cnt0 : Z+1 >
  else < P : Process | msgcnt : N+1 , cnt1 : 0+1 > fi
  if (n - t) > n [metadata "rate: 1"] .
```

If more than $(n + t)/2$ messages have the same value v , it broadcasts the message `mD(R,v)` to all other processes.

```
cr1 [sendD0-step2] : < P : Process | step : 2 , round : R ,
  msgcnt : (n - t) , cnt0 : Z , cnt1 : 0 , acq : A >
  => < P : Process | step : 3 , msgcnt : 0 , cnt0 : 0 ,
  cnt1 : 0 > bcast(A,mD(R,0))
  if 2 * Z > (n + t) [metadata "rate: 1"] .
cr1 [sendD1-step2] : < P : Process | step : 2 , round : R ,
  msgcnt : (n - t) , cnt0 : Z , cnt1 : 0 , acq : A >
  => < P : Process | step : 3 , msgcnt : 0 > bcast(A,mD(R,1))
  if 2 * 0 > (n + t) [metadata "rate: 1"] .
```

Else it broadcasts the message $m?(R)$.

```

crl [send-step2] : < P : Process | round : R , msgcnt : (n - t) ,
    cnt0 : Z , cnt1 : 0 , step : 2 , acq : A >
=> < P : Process | step : 3 , msgcnt : 0 > bcast(A,m?(R))
    if (n + t) >= 2 * 0 and (n + t) >= 2 * Z [metadata "rate: 1"] .

```

In the third step, P waits till it receives $n - t$ messages of type $mD(R,*)$ and $m?(R)$. If there are more than t messages of type $mD(R,v)$ with same value v , it sets x_P to v . If there are more than $(n + t)/2$ $mD(R,*)$ messages then it sends $\text{decide}(v)$ message to the observer.

```

crl [decide-step3] : < P : Process | val : X , step : 3 ,
    msgcnt : (n - t) , cntD0 : D0 , cntD1 : D1 , flag : true >
=> < P : Process | step : 4 > decide(X)
    if (2 * D0 + 2 * D1) > (n + t) [metadata "rate: 1"] .

```

Else it sets x_P to 1 or 0 each with probability $\frac{1}{2}$.

```

rl [choose-x-step3] : < P : Process | val : X , step : 3 , flag : false >
=> < P : Process | val : 0 , step : 4 > [metadata "rate: 0.5"] .

rl [choose-x-step3] : < P : Process | val : X , step : 3 , flag : false >
=> < P : Process | val : 1 , step : 4 > [metadata "rate: 0.5"] .

```

In step four, P increments its round and goes back to step one.

4.3 Mappings between Theory Classes

Each probabilistic rewrite theory $\mathcal{R} = (\Sigma, E, L, R, \rho) \in \mathbf{PRwTh}$ has an obvious underlying ordinary rewrite theory $W(\mathcal{R}) = (\Sigma, E, L, R) \in \mathbf{RwTh}$ obtained by forgetting the ρ .

Conversely, let \mathbf{RwTh}_o be the class of rewrite theories [20, 21] such that different rewrite rules have different labels, and the rules have only equational conditions (conjunctions of equations and memberships). Then we can define a function $J : \mathbf{RwTh}_o \rightarrow \mathbf{PRwTh}$ embedding rewrite theories into probabilistic rewrite theories in such a way that if $\mathcal{R}_o = (\Sigma, E, L, R) \in \mathbf{RwTh}_o$, then $J(\mathcal{R}_o) = (\Sigma, E, L, R, \rho) \in \mathbf{PRwTh}$ with ρ mapping each rewrite rule in R to 1.

We can likewise view \mathbf{PNS} s as a special low-level case of probabilistic rewrite theories by defining a map $R : \mathbf{PNS} \rightarrow \mathbf{PRwTh}$ defined as follows. Given a \mathbf{PNS} $\Pi = (S, V, \tau)$ we define $R(\Pi) = (\Sigma_\Pi, E_\Pi, L_\Pi, R_\Pi, \rho_\Pi)$ with Σ_Π an unsorted signature consisting only of the states $s \in S$ as constants. For each $s \in S$ and for each $p_i^s \in \tau(s)$, let $\{s_1, \dots, s_m\}$ be the set of next states such that $\sum_{j \in [1, m]} p_i^s(s_j) = 1$. For that s and p_i^s , L_Π contains the label $l(s, p_i^s)$ and R_Π contains the following set of rewrite rules,

$$\begin{aligned}
l(s, p_i^s) : s &\rightarrow s_1 \text{ [rate } p_i^s(s_1)] \\
&\dots \\
l(s, p_i^s) : s &\rightarrow s_m \text{ [rate } p_i^s(s_m)]
\end{aligned}$$

and ρ_Π maps each such rule $l(s, p_i^s) : s \rightarrow s_j$ to $p_i^s(s_j)$.

Conversely, given a probabilistic rewrite theory $\mathcal{R} = (\Sigma, E, L, R, \rho)$, and a sort $State$ in Σ such that if $[u] \in T_{\Sigma/E, State}$, then all probabilistic one-step transitions $[u] \rightarrow [v]$ as defined above are such that $[v] \in T_{\Sigma/E, State}$, we can associate to \mathcal{R} and $State$ an underlying PNS, $U(\mathcal{R}, State) = (S_{\mathcal{R}}, V_{\mathcal{R}}, \tau_{\mathcal{R}})$ with:

1. $S_{\mathcal{R}} = T_{\Sigma/E, State}$,
2. $V_{\mathcal{R}} : T_{\Sigma/E, State} \rightarrow \mathcal{P}(SPred_0(\Sigma, State))$, where $SPred_0(\Sigma, State)$ is the set of first-order formulas, P , called *state predicates* in the first-order language with equality $FOL(\Sigma)$ that have a single free variable x of sort $State$, that is, $fvars(P) = \{x\}$. The function $V_{\mathcal{R}}$ then maps each state $[u] \in T_{\Sigma/E, State}$ to the set of state predicates that hold in $[u]$, that is, $V_{\mathcal{R}}([u]) = \{P \in SPred_0(\Sigma, State) \mid T_{\Sigma/E} \models_{FOL} P(x \leftarrow u)\}$.
3. $\tau_{\mathcal{R}}$ associates to each state $[u] \in T_{\Sigma/E, State}$ the set $\tau([u])$ consisting of all probability distributions of the form $p([(C, t, \theta)_{[u]}, l])$ where $(C, t, \theta)_{[u]} \in \mathcal{D}_{[u]}^{\mathcal{R}}$ is a matching decomposition and the set of enabled rules with label l for θ

$$\begin{aligned} l : t &\rightarrow t'_{i_1} \text{ if } C_{i_1} \text{ [rate } r_{i_1}(\vec{x})] \\ &\dots \\ l : t &\rightarrow t'_{i_m} \text{ if } C_{i_m} \text{ [rate } r_{i_m}(\vec{x})] \end{aligned}$$

is nonempty. The probability distribution $p([(C, t, \theta)_{[u]}, l])$ is the function associating $p_{[v]}$ to each next state $[v]$ as defined above, and zero to every other state.

Let $*/\mathbf{PRwTh}$ be the class of pairs $(\mathcal{R}, State)$, with $\mathcal{R} = (\Sigma, E, L, R, \rho) \in \mathbf{PRwTh}$, and $State$ a sort in Σ such that if $[u] \in T_{\Sigma/E, State}$, and $[u] \rightarrow [v]$ is a one-step probabilistic transition, then $[v] \in T_{\Sigma/E, State}$. The above mapping U is then a function $U : */\mathbf{PRwTh} \rightarrow \mathbf{PNS}$. There is also an obvious projection function $\pi_1 : */\mathbf{PRwTh} \rightarrow \mathbf{PRwTh}$.

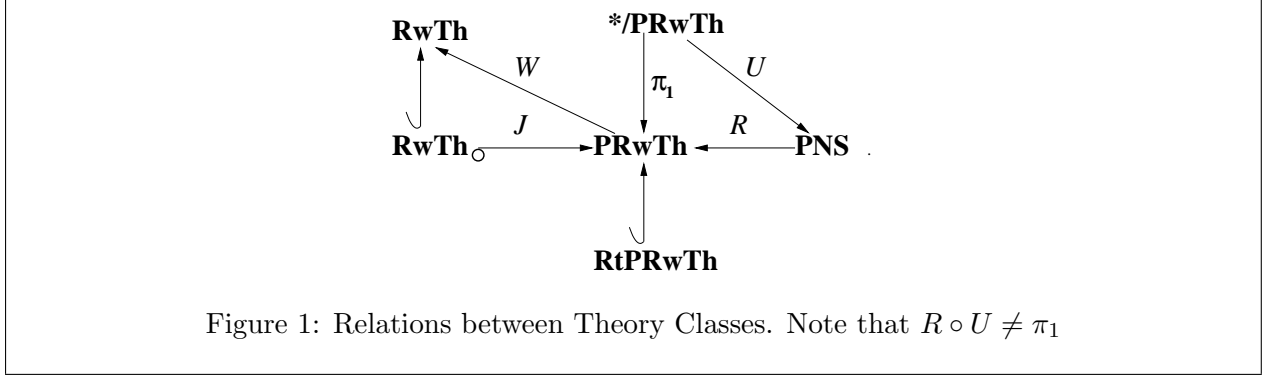
Note that the mapping U gives us a way of associating a temporal logic semantics to a probabilistic rewrite theory \mathcal{R} . Indeed, given a probabilistic rewrite theory $\mathcal{R} = (\Sigma, E, L, R, \rho)$ with a sort $State$, a state $[u] \in T_{\Sigma/E, State}$, and a state formula Φ in the temporal logic language with atomic propositions in $SPred_0(\Sigma, State)$, the satisfaction relation, $U(\mathcal{R}, State), [u] \models \Phi$, states that (the PNS associated to the) probabilistic rewrite theory \mathcal{R} and the initial state $[t]$ satisfy the state property Φ . When the sort $State$ can be left implicit, we use the simpler notation $\mathcal{R}, [u] \models \Phi$ as an abbreviation for $U(\mathcal{R}, State), [u] \models \Phi$.

We summarize all the mappings defined in this section in Figure 1, where $\mathbf{RtPRwTh}$ is the class of discrete-time probabilistic rewrite theories defined in the following section.

5 Discrete-Time Probabilistic Rewrite Theories

In a discrete-time probabilistic rewrite theory we assume that discrete real-time acts on the global state. Time is incremented by a special rewrite rule *tick*. To ensure that the system has performed all the necessary computations between two ticks, we require that the *tick* rule is enabled in a state only when none of the other rewrite rules are enabled.

Definition 10 (Discrete-time probabilistic rewrite theory) *A discrete-time probabilistic rewrite theory is just a probabilistic rewrite theory $\mathcal{R} = (\Sigma, E, L, R, \rho)$ such that there is a sort, called $State$,*



to represent the state of the whole system, and another sort, called *ClockedState*, associating to the state a *global time* (wall clock) and having a constructor

$$op \langle _, _ \rangle : State Nat \rightarrow ClockedState.$$

where $\langle _, _ \rangle$ is the only operator that constructs terms of the *ClockedState* kind. We assume that *Nat* (natural numbers) is the sort used to represent discrete-time and that (Σ, E) is a protecting extension of the theory *NAT* of the natural numbers. For the *tick* label, \mathcal{R} contains a single rule

$$[tick] : \langle S, t \rangle \rightarrow \langle tick(S), t + 1 \rangle \text{ if } C \text{ [rate 1]} .$$

where the function $tick : State \rightarrow State$ defines the action of one time instant on the whole state by suitable equations. Given a ground substitution θ , we require that, whenever the predicate $\theta(C)$ is true, none of the other rewrite rules in \mathcal{R} , except *tick*, are enabled for the term $\theta(\langle S, t \rangle)$. We denote by **RtPRwTh** the subclass of **PRwTh** given by the discrete-time probabilistic rewrite theories as defined above.

Let us consider as an example a system having two digital clocks. The clocks are running and can break down probabilistically, depending on the charge left in each battery and the length of time for which each clock has been running. Each clock can also be reset by an external source. The behavior of such system can be easily specified as a theory in **RtPRwTh** as follows:

```

sort Clock Flag Status .
ops current next : -> Flag .
ops running broken : -> Status .
*** clock(Battery Charge, local time, status, ticking enabled) -> Clock
op clock : Float Nat Status Flag -> Clock [ctor] .
*** Two clocks form the global state
op state : Clock Clock -> State [ctor] .
op <_,_> : State Nat -> ClockedState .
op enabled : ClockedState -> Bool .

op tick : State -> State .
[tick]: <S,t> => <tick(S),t+1> if enabled(<S,t>) /= true [rate 1] .

[internal]: clock(B,T,running,current) =>
            clock(B-delta,T+1,running,next) [rate r1(B,T)] .
[internal]: clock(B,T,running,current) =>
            clock(B,T,broken,current) [rate r2(B,T)] .

```

```

[reset]:  clock(B,T,running,current) =>
          clock(B,0,running,next) [rate 1] .

eq init = state(clock(100.0,7,running,current), clock(200.0,0,running,current)) .

```

The rules `internal` state that a clock can either continue to run by incrementing its local time by 1 with rate $r1(B,T)$ or it can stop running (because of low battery charge or because it is running for long time) with rate $r2(B,T)$. Note that both rates are dependent on the battery charge left and the time for which the clock is running, and they can be defined equationally. The rule `reset` is a nondeterministic possibility which can happen if somebody manually resets the time of a clock to 0. The sort `Status` has two constants: `running` and `broken` to represent if the clock is running or broken. The sort `Flag` has two constants: `current` and `next` to represent if a rewrite rule can be applied to a clock or not. The predicate `enabled` in rule `tick` is computed using these flags as follows:

```

eq enabled(<state(clock(B,T,s,current),C2), t>) = true .
eq enabled(<state(C1,clock(B,T,s,current)), t>) = true .

```

The operator `tick` can be equationally defined to reset the flag of each clock to `current` and thus enable the rules `internal` and `reset`.

A probabilistic discrete-time model of sensor networks based on actors [1, 2] and meta-actors [4] has been specified as a theory in **RtPRwTh**. The rewrite specification of actors in rewriting logic [28] is used as a submodule. In this model of sensor networks the nondeterminism in message delay due to transmission failure, collision, and message loss, is replaced by probabilities. The elements of such a network follow a global clock, but in a probabilistic manner.

Thus theories \mathcal{R} in **RtPRwTh** can model real-time probabilistic systems such as communication protocols, embedded systems, and control systems. The properties of interest for these kinds of systems are properties such as:

- *The clocks will run without stopping for 356 days with probability 0.98,*
- *There is 95% chance that a packet will be delivered within 3ms.*

Such properties can be easily expressed in pCTL by treating global time t as a normal state variable and time bounds as normal state predicates. However, considering global time as a state variable results in a system with an infinite number of reachable states and hence in general cannot be model checked. But if we are interested in system properties such as those mentioned above that are bounded by a finite global time b , we can associate to \mathcal{R} a theory \mathcal{R}_b which involves a finite number of time instants. If $\mathcal{R} = (\Sigma, E, L, R, \rho)$, then $\mathcal{R}_b = (\Sigma, E, L, R_b, \rho_b)$ with R_b and ρ_b identical to R and ρ , except that the `tick` rule is now replaced by the rule,

$$[tick] : \langle S, t \rangle \rightarrow \langle tick(S), t + 1 \rangle \text{ if } \text{cond}(\langle S, t \rangle) \text{ and } t < b \text{ [rate 1]} .$$

where b is the finite bound on global real-time. In practice many such properties of interest are bounded by a finite global time. The point is that given a time-bounded property φ with bound b , and given an initial state $[u]$, we have an equivalence,

$$\mathcal{R}, [u] \models \varphi \Leftrightarrow \mathcal{R}_b, [u] \models \varphi .$$

If the set of states reachable from $[u]$ in \mathcal{R}_b is finite, then we can model check the property by model checking φ for the underlying PNS, $U(\mathcal{R}_b, State)$ with an appropriate decision procedure.

6 A Prototype Interpreter: PMaude

A prototype interpreter PMaude is currently under development. We plan to report on a running prototype in the final version of the paper. The interpreter will provide a realistic simulation environment for a probabilistic rewrite theory, making use of the reflective features of Maude [9].

A probabilistic rewrite theory will be specified as an ordinary system module with some metadata associated with the rewrite rules. This facility of specifying metadata with rewrite rules will be supported in the next release of Maude 2.0 [16]. The probability information for a rewrite rule in such a module would look like:

```
cr1 [label] : t(x1,x2) => t'(x1,x2) if cond(x1,x2) [metadata "r(x1,x2)"] .
```

```
*** t(x1,x2), t'(x1,x2), cond(x1,x2), r(x1,x2) are terms definable in the module.
```

The PMaude interpreter will make use of the LOOP-MODE module and the above mentioned capability of associating metadata information with rewrite rules to provide an input/output environment where the user will enter a module, followed by a rewrite command. When the interpreter reads a rewrite command it is given an initial term to rewrite. At this stage the interpreter will compute all the nondeterministic choices possible for choosing a rewrite, as we have described in the semantics of a probabilistic rewrite theory in Section 4. The interpreter now has to make a decision of which nondeterministic choice to make, to actually proceed with the rewriting. In the current design this choice is limited and the interpreter chooses a nondeterministic choice fairly (with equal probability between all choices) among the various nondeterministic choices. Later we can also choose to make this user-defined. A particular nondeterministic choice then fixes a rewrite label, a matching context, and a substitution. A number of rewrite rules can get enabled at this moment. The interpreter also computes the probabilities associated with the various rewrites in the chosen nondeterministic choice, say n of them, by using the particular substitution selected in the probability expressions (which in general can have variables). At this moment the interpreter calls upon a random number generator to decide, based on the probabilities calculated, to actually decide which rewrite rule to apply. The above mentioned sub-step of choosing a probabilistic choice can be thought of as simulating a n -sided biased dice, with the probability of coming up a particular side equal to the probability of one of the n choices. The side which actually comes up in a throw will determine which rule to apply. After the choice is selected, a rewrite is actually applied. If there were no rewrites from the term to start with, or the interpreter has completed the maximum number of rewrites as specified in the command, the final term is just output to the output buffer.

7 Conclusions and Future Work

We have proposed probabilistic rewrite theories as a general semantic framework supporting high-level specification of probabilistic systems that can be massively concurrent and can have real-time features. We have also shown how these systems can be both symbolically simulated and analyzed, and can be model checked using existing algorithms when they are finite state.

Much work remains ahead, including:

- show in detail how other existing models besides PNSs, such as CCS, probabilistic Petri nets or I/O-automata-based models can be naturally expressed within our framework;

- develop the *deductive aspects*, both of probabilistic rewrite theories themselves, and of suitable probabilistic temporal logics;
- develop *tools* supporting our formalism, including the PMaude system, and model checking tools;
- generalize the existing approach to model concurrent probabilistic systems with *continuous real time* features, by combining the ideas presented here with those developed for *real-time rewrite theories* [22];
- gain greater experience through case studies applying our specification formalism in a wide range of areas.

8 Acknowledgment

The work is supported in part by the Defense Advanced Research Projects Agency (Contract numbers: F30602-00-2-0586 and F33615-01-C-1907) and the ONR MURI Project “A Logical Framework for Adaptive System Interoperability”. We would like to thank Narciso Martí-Oliet for reviewing previous versions of this paper and giving feedback.

References

- [1] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.
- [2] Gul Agha. *Actors: A Model of Concurrent Computation*. MIT Press, 1986.
- [3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [4] Mark Astley and Gul Agha. Customization and composition of distributed objects: Middleware abstractions for policy management. In *Proceedings Sixth International Symposium on the Foundations of Software Engineering (FSE-6, SIGSOFT '98)*, pages 1–9, 1998.
- [5] Christel Baier, Edmund M. Clarke, Vassili Hartonas-Garmhausen, Marta Z. Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming, ICALP*, pages 430–440, 1997.
- [6] Christel Baier and Marta Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.
- [7] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing (PODC)*, 1983.
- [8] Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 15, 1995.
- [9] Manuel Clavel and José Meseguer. Reflection in rewriting logic and its applications in the Maude language. In *IMSA '97*, pages 128–139. Information-Technology Promotion Agency, Japan, 1997.
- [10] Hansson H. Time and probability in formal design of distributed systems. volume 1, Amsterdam, 1994. Elsevier.
- [11] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
- [12] J. Hillston and M. Ribaud. Stochastic process algebras: A new approach to performance modeling. In J. Walrand K. Bagchi and G. Zobrist, editors, *Modeling and Simulation of Advanced Computer Systems*. Gordon Breach, 1998.
- [13] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker, 2002.
- [14] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *International Conference on Concurrency Theory*.
- [15] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. In *82*, page 16. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1995.
- [16] S. Eker P. Lincoln N. Martí-Oliet J. Meseguer M. Clavel, F. Durán and J.F. Quesada. Towards Maude 2.0. In *3rd International Workshop on Rewriting Logic and its Applications (WRLA '00)*, volume 36 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2000.
- [17] M. Paterson M. Fischer, N. Lynch. Impossibility of distributed consensus with one faulty process. In *JACM*, volume 32, pages 374–382, 1985.
- [18] M. Ajmone Marsan. Stochastic petri nets: An elementary introduction. *Lecture Notes in Computer Science; Advances in Petri Nets 1989*, 424:1–29, 1990. NewsletterInfo: 36.
- [19] M. Ajmone Marsan, A. Bobbio, and S. Donatelli. Petri nets in performance analysis: An introduction. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491:211–256, 1998.
- [20] José Meseguer. Rewriting as a unified model of concurrency. Technical Report SRI-CSL-90-02, SRI International, Computer Science Laboratory, February 1990. Revised June 1990.

- [21] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
- [22] Peter Csaba Ölveczky and José Meseguer. Specification of real-time and hybrid systems in rewriting logic. To appear in *Theoretical Computer Science*, maude.csl.sri.com, 2000.
- [23] A. Pnueli and L. Zuck. Probabilistic verification by tableaux. In *Proceedings of the First Symposium on Logic in Computer Science, Cambridge*, pages 322–331, 1986.
- [24] A. Pnueli and L. Zuck. Probabilistic verification. In *Inf Comput*, volume 103, pages 1–29, 1993.
- [25] Corrado Priami. Stochastic π -calculus with general distributions. In *Proceedings of PAPM '96*, 1996.
- [26] Rajeev Alur and Thomas A. Henzinger. Logics and Models of Real-Time: A Survey. In *Real Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, 1991.
- [27] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In *International Conference on Concurrency Theory*, pages 481–496, 1994.
- [28] C. L. Talcott. An actor rewriting theory. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, number 4 in *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996.

9 Appendix

```
(pmod BYZANTINE is

protecting MACHINE-INT .
sorts Steps Flag Binary Contents OidSet .
subsort Binary < Steps .
subsort Bool < Flag .
subsort Oid < OidSet .

op null : -> Flag .
ops 0 1 : -> Binary .
ops 2 3 4 : -> Steps .
op nil : -> OidSet .

*** n is number of processors
*** t is maximum number of faulty processors
op n t : -> MachineInt .

op m1 : Nat Binary -> Contents .
op mD : Nat Binary -> Contents .
op m? : Nat -> Contents .
msg _<_ : Oid Contents -> Msg .
msg decide : Binary -> Msg .
op __ : OidSet OidSet -> OidSet [comm assoc id: nil] .
op bcast : OidSet Contents -> Configuration .

var C : Contents .
var A : OidSet .
var F : Flag .
var P : Oid .
vars R N Z O D0 D1 : MachineInt .
var X : Binary .

class Process | val : Binary , step : Steps , round : MachineInt ,
  msgcnt : MachineInt , cnt0 : MachineInt , cnt1 : MachineInt ,
  cntD0 : MachineInt , cntD1 : MachineInt , flag : Flag , acq : OidSet .

eq bcast(nil,C) = none .
eq bcast(P A,C) = (P <- C) bcast(A,C) .

rl [step0] : < P : Process | step : 0 , round : 0 >
=> < P : Process | step : 1 , round : 1 > [metadata "rate: 1"] .

rl [send-step1] : < P : Process | val : X , step : 1 , round : R , acq : A >
=> < P : Process | step : 2 > bcast(A,m1(R,X)) [metadata "rate: 1"] .

crl [receive-step2] : (P <- m1(R,X))
< P : Process | step : 2 , round : R , msgcnt : N , cnt0 : Z , cnt1 : O >
=> if x == 0 then < P : Process | msgcnt : N + 1 , cnt0 : Z + 1 >
else < P : Process | msgcnt : N + 1 , cnt1 : O + 1 > fi
if (n - t) > N [metadata "rate: 1"] .

crl [sendD0-step2] : < P : Process | step : 2 , round : R ,
```

```

    msgcnt : (n - t) , cnt0 : Z , cnt1 : 0 , acq : A >
=> < P : Process | step : 3 , msgcnt : 0 , cnt0 : 0 , cnt1 : 0 > bcast(A,mD(R,0))
    if 2 * Z > (n + t) [metadata "rate: 1"] .

crl [sendD1-step2] : < P : Process | step : 2 , round : R ,
    msgcnt : (n - t) , cnt0 : Z , cnt1 : 0 , acq : A >
=> < P : Process | step : 3 , msgcnt : 0 > bcast(A,mD(R,1))
    if 2*0 > (n + t) [metadata "rate: 1"] .

crl [send-step2] : < P : Process | round : R , msgcnt : (n - t) ,
    cnt0 : Z , cnt1 : 0 , step : 2 , acq : A >
=> < P : Process | step : 3 , msgcnt : 0 > bcast(A,m?(R))
    if (n + t) >= 2 * 0 and (n + t) >= 2 * Z [metadata "rate: 1"] .

crl [receiveD-step3] : (P <- mD(R,X))
< P : Process | step : 3 , round : R , msgcnt : N , cntD0 : D0 , cntD1 : D1 >
=> if X == 0 then < P : Process | msgcnt : N + 1 , cntD0 : D0 + 1 >
else < P : Process | msgcnt : N + 1 , cntD1 : D1 + 1 > fi
    if (n - t) > N [metadata "rate: 1"] .

crl [receive-step3] : (P <- m?(R))
< P : Process | step : 3 , round : R , msgcnt : N >
=> < P : Process | msgcnt : N + 1 >
    if (n - t) > N [metadata "rate: 1"] .

crl [set0-step3] : < P : Process | val : X , step : 3 ,
    msgcnt : (n - t) , cntD0 : D0 , flag : null >
=> < P : Process | val : 0 , flag : true > if D0 > t [metadata "rate: 1"] .

crl [set1-step3] : < P : Process | val : X , step : 3 ,
    msgcnt : (n - t) , cntD1 : D1 , flag : null >
=> < P : Process | val : 1 , flag : true > if D1 > t [metadata "rate: 1"] .

crl [set1-step3] : < P : Process | step : 3 , msgcnt : (n - t) ,
    cntD1 : D1 , cntD0 : D0 , flag : null >
=> < P : Process | flag : false > if t >= D0 and t >= D1 [metadata "rate: 1"] .

crl [decide-step3] : < P : Process | val : X , step : 3 , msgcnt : (n - t) ,
    cntD0 : D0 , cntD1 : D1 , flag : true >
=> < P : Process | step : 4 > decide(X)
    if 2 * (D0 + D1) > (n + t) [metadata "rate: 1"] .

rl [choose-x-step3] : < P : Process | val : X , step : 3 , flag : false >
=> < P : Process | val : 0 , step : 4 > [metadata "rate: 0.5"] .

rl [choose-x-step3] : < P : Process | val : X , step : 3 , flag : false >
=> < P : Process | val : 1 , step : 4 > [metadata "rate: 0.5"] .

rl [step4] : < P : Process | step : 4 , round : R , flag : F ,
    msgcnt : N , cnt0 : Z , cnt1 : 0 , cntD0 : D0 , cntD1 : D1 >
=> < P : Process | step : 1 , round : R + 1 , flag : null ,
    cnt0 : 0 , cnt1 : 0 , msgcnt : 0 , cntD0 : 0 , cntD1 : 0 > [metadata "rate: 1"] .
endpm)

```