Performance Analysis of Real-Time Rewriting Models

Jounaidi Ben Hassan*, Osman Hasan§, Tarek Sadani* and Sofiène Tahar*
*Department of Electrical & Computer Engineering,
Concordia University,
1455 de Maisonneuve W., Montreal, Quebec, H3G 1M8, Canada
Email: {jounaidi,o_hasan,tsadani,tahar}@ece.concordia.ca
§School of Electrical Engineering and Computer Science,
National University of Sciences and Technology,
Sector H-12, Islamabad, 44000, Pakistan
Email: osman.hasan@seecs.nust.edu.pk

***Abstract*—Real-time systems usually involve a subtle interaction of a number of distributed components and have a high degree of parallelism, which makes their performance analysis quite complex. Thus, traditional techniques, such as simulation, fail to produce reasonable results. Formal methods pose an interesting solution but they usually lack the capabilities to reason about quantitative time and probabilistic properties, which play a vital role in performance analysis. This paper addresses this issue by presenting a formal approach for assessing the performance of a real-time system. To describe the evolution of the system, we use a real-time rewriting logic, in which we mechanize the extraction of quantitative information from a timed model. To evaluate the performance, we first consider the set of runs obtained from different initial input values that are not equivalent modulo the equational theory associated with the model. The overall performance of the system is then evaluated as the performance of each run weighted by its probability mass function. In order to illustrate the practical effectiveness of the proposed approach, we present the formal modeling and performance analysis of a simple search engine.**

Keywords: Performance Analysis, Probabilistic Analysis, Real-Time Systems, Rewriting Logic, RT-Maude.

I. Introduction

Real-time systems can be characterized as systems for which the correctness of an operation is dependant not only on its logical correctness but also on the time taken. Some commonly used real-time system applications include embedded systems, digital circuits with uncertain delays and communication protocols. Due to the increased usage of real-time systems in safety critical and extremely sensitive applications such as medicine, military and space travel, their correctness and performance has become imperative. The functional verification and performance evaluation tasks in this domain are quite challenging as the present age real-time systems usually involve a subtle interaction of a number of concurrent and distributed components and have a high degree of parallelism. The level of sophistication found in current real-time systems demands formal specification of requirements and thus traditional techniques, like simulation, fail to provide the necessary level of assuredness about what is being built is actually what was intended.

On the other hand, formal methods offer a promising solution. A number of elegant approaches for the formal verification of real-time systems can be found in the open literature using state-based or theorem proving techniques (e.g. [6], [3]). However, most of these tools are only capable of specifying and verifying hard deadlines, i.e., properties where a late response is considered to be incorrect. For example, while proving the message delivery characteristic of a communication protocol, we can only check if the message is delivered for sure within a specific duration in time, say 100s, for all possible scenarios. Even though this information is quite useful for establishing functional correctness, it usually is found to be quite insufficient for performance analysis, where the designers are more interested to determine the maximum (or minimum) possible values or the probabilities associated with the parameters of interest.

In this paper, we propose a formal approach for the analysis of the above mentioned performance analysis related properties for real-time systems. Our approach is based on rewriting logic [13], which is a simple and intuitive formalism that is well suited to be used as a semantics framework in which one can express different models of computation. The primary motivation of using rewriting logic is to leverage upon its naturalness and simplicity for expressing concurrent and distributed real-time systems. Rewriting logic has been successfully used for the functional verification of many concurrent models (e.g. [14], [16]). The main contribution of this paper is to utilize this formalism to analyze performance related properties of a real-time system. We propose three novel techniques in this direction.

Firstly, we propose algorithms for the computation of quantitative timing information, such as exact minimum and maximum delays, between two specified rewrite rules. This is probably the most crucial timing measurement for real-time systems as it allows designers to ensure that the timing objectives are being satisfied. For example, our algorithms can be used to measure the time taken from one specified point in a piece of code to another or the time between the occurrence of an external event and the system response to that event.

Secondly, we provide counting algorithms that compute the bounds on the number of states that satisfy a property between two specified rewrite rules. This feature provides more detailed information compared to the traditional formal techniques when addressing classical problems, such as schedulability analysis. For example, rather than just determining whether some tasks will always meet their deadlines, when executed under a given scheduling strategy, the likelihood of a *priority inversion* situation may be assessed by counting the number of reachable states where actions of lower priority processes can be executed during the execution of a high priority process. On the other hand, priority inversion time can be determined by looking at the fraction of time spent in such states.

Thirdly, we propose algorithms to compute the probability of a property being satisfied for all inputs of the given system. This capability allows us to optimize the system's performance over all inputs specially when we are not given an initial state assignment. This is a key feature of the proposed approach, since most of the available formal approaches are only capable of analyzing the truth value of a given property and do not provide any probabilistic information.

In order to illustrate the utilization and practical effectiveness of the proposed algorithms, we utilize them to evaluate the performance of a simple search engine system. The system is first formally modeled in RT-MAUDE [19], which is a language and tool that supports formal specification and analysis of real-time and hybrid systems. The proposed algorithms, also modeled in RT-MAUDE, are then used to precisely evaluate the minimum and maximum delays associated with a given run of the systems model. Similarly, the proposed probabilistic time computation approach is also used to formally achieve some interesting probabilistic information about the given search engine system.

The paper is organized as follows: Section II provides a review of the related work. Then, in the next three section, we provide the details associated with the above mentioned three contributions, i.e., the algorithms for computing bounds on delays for an interval, bounds on the number of times a property is verified for an interval and probabilistic properties, respectively. The search engine case study is given in Section VI. Finally, Section VII concludes the paper.

II. RELATED WORK

The minimum delay problem was first formulated by [9] within the Markovian decision theory. The authors of [9] addressed the problem of intercepting in minimum expecting time a target that moves randomly among a finite number of states. In [2] the minimum delay problem is formulated as a shortest stochastic path problem.

A large number of state-based formal approaches have been proposed that offer the capability to analyze the performance of real-time systems. They can be broadly classified into two categories, i.e., using probabilistic model-checking (e.g.,[11], [12]) and Petri nets (e.g. [4], [1]). However, all these approaches share the same inherent limitation, which is the reduced expressive power of their automata based or Petri net based specification formalism, when compared to the proposed rewriting logic formalism. For example, a verification methodology consisting of selective quantitative timing analysis and interval model checking has been proposed in [5]. The model of the system is a Kripke structure. The selective quantitative algorithms compute minimum and maximum delays over a selected subset of system executions. However, this approach is based on the assumption that the delay between two system's state is constant and equal to one time unit. Also, the selection of executions set is based on a linear-time temporal logic [22] formula and not on the environment behavior, which can also be probabilistic. Moreover, this approach lacks a mechanism which permits to specify suspension and resumption of actions without losing the temporal context. Hence, it seems to be appropriate only for systems that can be executed in a *lock-step*, which is usually not the case for real-time systems. Rewriting logic, which is the fundamental technique in the proposed approach, allows us to compute delays without any additional assumptions and to select executions based on a probabilistic model of the environment, as will be seen in this paper.

Besides the state-based formal techniques, interactive theorem proving based on higher-order logic has also been explored for conducting the performance analysis of real-time systems. The formal performance analysis of the Stop-and-Wait protocol, which is a classical example of a real-time system, is presented in [10]. Unlike the state-based formal methods, due to the high expressive nature of higher-order logic, this approach is capable of formally specifying any real-time system that can be described mathematically and reason about all sorts of statistical and probabilistic performance characteristics. But, it requires an enormous amount of user effort for interactively verifying the intended properties in the theorem prover (e.g., the Stop-and-Wait protocol analysis required 300 man-hours [10]), which is not the case in the proposed rewriting logic based technique.

To the best of our knowledge, there is no ad hoc approach to analyze the performance of real-time rewriting models. The most that has been achieved in this area so far is the "manual" analysis by defining execution strategies and comparing output results. For example, the specification and analysis of an AFR/NCA active network protocol, which is a real-time system, is done using rewriting techniques in [17]. To evaluate its performance, some variables have been added to the original model and the specification of the system was executed with different strategies. However, this approach highlights a serious drawback especially if many performance criteria have to be measured. In fact, continuous adding of variables to the initial system can introduce errors or modify the original system behavior in unexpected ways. Also, there is no guarantee that the selected execution strategies will provide the best estimation of the properties to be verified. To the best of our knowledge, the proposed approach presents an automated performance analysis of real-time rewriting models for the first time and thus tends to overcome the above mentioned limitations.

We utilize RT-MAUDE [19] to implement the proposed algorithms and conduct our case study. RT-MAUDE complements conventional timed-automata based model checkers, such as UPPAAL [7] and KRONOS [8], by the full generality of the specification language, but most importantly by its simplicity and clarity. The tool offers a wide range of analysis techniques including timed rewriting for simulation purposes, and time-bounded linear temporal logic model checking. It has been used to model and analyze sophisticated communication protocols [20], state of the art wireless sensor networks [21] and scheduling algorithms [18]. The main strength of RT-MAUDE lies in the modeling and analysis of complex systems that cannot be naturally expressed as, e.g., finite timed (or hybrid) automata or timed petri nets. This gain in expressive power is appreciable. However, it is balanced by the undecidability of many system properties.

III. REAL-TIME REWRITING SYSTEMS DELAYS

In this section, we first provide a brief review of rewriting logic and its usage in expressing real-time systems. This will be followed by the proposed algorithms for the computation of delay bounds.

Rewriting logic theory [13] expresses an essential equivalence between logic and computation. System's states are in bijective correspondence with formulas, and concurrent computations are in bijective correspondence with proofs. Given this equivalence between computation and logic, a rewrite rule of the form $t \longrightarrow t^{'}$ can be read in the following two different ways:

- Computationally: it is interpreted as a local transition which expresses how parts of system's state can change concurrently;
- Logically: it is interpreted as an inference rule which says that we can derive the formula *t'* form the formula *t*.

The computational and the logical viewpoints are not exclusive; they complement each other and can be seen as two different sides of the same coin.

A rewrite theory [13] can be formally defined as a 4-tuple $\mathcal{R} = (\Sigma, E, L, R)$ [15], where

- (Σ, E) is the equational theory modulo which we rewrite.
- L is a set of labels.
- R is a set of labeled rewrite rules.

A commonly occurring form for the equational theory (Σ, E) is $E = E' \cup A$, where A is a set of equational axioms for which there exists a matching algorithm, and E' is a set of terminating equations modulo A.

Now, a real-time system may be described in rewrite theory by $\mathcal{M} = (\Sigma, E, L, R)$, where R, in this case, can be viewed as the union of two disjoint sets R_I and R_T. R_I is the set of *instantaneous* rewrite rules $\ell : t \longrightarrow t^{'}$ that are applied without delay and R_T represents the set of *tick* rewrite rules. A tick rewrite rule $\ell : t \xrightarrow{\tau_\ell} t^{'}$ is applied after the specified delay τ_ℓ.

A run ρ of $\mathcal{M}$ represents a set of rewriting derivations of $\mathcal{M}$. Whereas, a rewriting derivation $\mathbf{deriv}(t_i, t_{i+1}, \tau_\ell)$ corresponds to one application of a rewrite rule $\ell : t_i \xrightarrow{\tau_\ell} t_{i+1}$. When ℓ is instantaneous, we assume that the derivation step is delayed by 0, i.e., $\tau_\ell = 0$. The starting term t_0 of a run ρ can be obtained by using the $head$ function as $\mathbf{head}(\rho)$. A run ρ can also be viewed as the union of some sequences of derivation steps. In this case, each sequence defines a path π from the starting term and has the form:

$$\pi = t_0 \xrightarrow{\tau_{0,1}} t_1 \xrightarrow{\tau_{1,2}} t_2 \xrightarrow{\tau_{2,3}} t_3 \xrightarrow{\tau_{3,4}} \dots$$

Given a path, the derivation step $\mathbf{deriv}(t_i, t_{i+1}, \tau_{i,(i+1)})$ starts at the global time $\sum_{j=0}^{j=i-1} \tau_{j,(j+1)}$. Given a term t in the run ρ, we define $\mathbf{step}(t)$ as the function that associates to t the set of all possible derivation steps that can be applied over t. $\mathsf{Tails}(\rho)$ denotes the set of terms in the run ρ that cannot be rewritten by any of the rewrite rules of the model.

Now, utilizing the above mentioned terminology, we present the proposed algorithms for the computation of minimum and maximum delays associated with real-time rewriting models. In the minimum delay computation, our main goal is to be able to predict how quickly can we start from a given state and reach a given set of states in a real-time rewriting model. We solve this minimum delay problem by reducing it to a *shortest path* (in terms of time delays) problem.

The proposed minimum delay computation algorithm is given in Algorithm 1. The function mintime accepts two variables: a starting term t_i and a set of final terms $T_f \neq \varnothing$, whereas typically, $T_f = \mathsf{Tails}(\rho)$. It returns the shortest time delay between t_i and T_f. The loop, in the algorithm, intuitively scans through the delays associated with the set of all possible terms in the real-time rewriting model of the system by executing each derivation step and recursively calling the function mintime for each one of them. If at any point, we encounter a delay shorter than the current delay, we update the latter by the new value. The correctness of Algorithm 1 relies on the hypothesis that there exists at least one path from some initial term reaching some final term.

Algorithm 1 Minimum Delay

function mintime(t_i, T_f)

```
1. if t_i ∈ T_f then return 0
2.    else mindelay := ∞
3.         for all t_k ∈ step(t_i) do
4.              min_{t_k} := mintime(t_k, T_f) + τ_{i,k}
5.              if mindelay > min_{t_k} then
6.                 mindelay := min_{t_k}
7.              end if
8.         end do
9.         return mindelay
10.   end if
```

It is important to note that technically, there may not be a *shortest path* in a real-time rewriting model, i.e., it is possible that every path can be shortened by an infinitesimal amount. So in order to be precise, we seek the greatest lower bound

on the *mindelay*. Moreover initial and final terms must be understood in a broad sense. For a terminating process these can be the terms in which execution starts and ends. Whereas, for a non-terminating process, they can be the terms in which a resource is requested and in which it has later been allocated, respectively.

We now consider the maximum delay computation in a real-time rewriting model. Similar to the minimum delay problem, we solve this problem by reducing it to the calculation of the *longest path* (in terms of time delays) between a term and a set of terms. Algorithm 2 accepts two parameters: t_i and $T_f \neq \varnothing$, where t_i is the starting term and T_f is the set of final terms. It returns the longest time delay between t_i and T_f. This algorithm is basically a recursive function that performs a backward search. The function $\mathbf{step}^{-1}(t)$ defines the set of terms $t^{'}$ such that $\exists\ \pi_i \in \rho$ satisfying $\mathbf{step}(t^{'}, t) \in \pi_i$. Intuitively, this algorithm explores the delays between each term $t_f \in T_f$ and the starting term t_i and it retains the maximum value of the delay encountered. The correctness of Algorithm 2 relies on two hypotheses. Firstly, that there exists at least one path from the initial term to some final term. Secondly, that there is no infinite cyclic path leading towards some final term that never backtracks to the initial term.

Algorithm 2 Maximum Delay

function maxtime(t_i, T_f)

1. if $t_i \in T_f$ then return 0
2. else maxdelay := 0
3. for all $t_f \in T_f$ do
4. for all $t_k \in \mathbf{step}^{-1}(t_f)$ do
5. maxdelay$_{t_k}$:= maxtime(t_i, $\{t_k\}$) + $\tau_{k,f}$
6. if maxdelay $<$ max$_{t_k}$ then
7. maxdelay := max$_{t_k}$
8. end if
9. end do
10. end do
11. return maxdelay
12. end if

IV. Counting Over a Run

In many performance related queries, we are interested not only in the length of a rewriting derivation from a starting term t_i to a non empty set of final terms T_f, but also the minimum or the maximum number of times a given property is satisfied on a rewriting derivation from the starting to final terms.

Now, we propose two algorithms that compute the minimum and maximum number of states where a given property is verified in a run ρ. Both algorithms accept three variables, i.e., the initial term t_i, a property p and a set of final terms T_f. The algorithms compute the minimum and the maximum number of terms (states) where p is true, over all finite paths of the rewriting derivations that are between a starting term t_i and some $t_f \in T_f$, respectively. The function v(t_k,p), used in these algorithms, returns 1 if the property p is verified on term t_k and 0 otherwise. To guarantee that the $mincounter$ and $maxcounter$ variables, in Algorithms 3 and 4, respectively, are well-defined, we assume that there exists at least one rewriting derivation path π_i beginning from t_i that reaches some term t_f in T_f after a finite number of steps. By definition of the rewriting derivation, those minimum and maximum are well defined when the starting term is the initial term of the run.

Algorithm 3 Minimum Property Counting

function minprop(t_i, p, T_f)

1. if $t_i \in T_f$ then return v(t_i,p)
2. else mincounter := ∞
3. for all $t_k \in \mathbf{step}(t_i)$ do
4. counter := minprop(t_k, p, T_f) + v(t_i,p)
5. if mincounter $>$ counter then
6. mincounter := counter
7. end if
8. end do
9. return mincounter
10. end if

Algorithm 4 Maximum Property Counting

function maxprop(t_i, p, T_f)

1. if $t_i \in T_f$ then return v(t_i,p)
2. else maxcounter := 0
3. for all $t_f \in T_f$ do
4. for all $t_k \in \mathbf{step}^{-1}(t_f)$ do
5. counter := maxprop(t_i, p, $\{t_k\}$) + v(t_k,p)
6. if maxcounter $<$ counter then
7. maxcounter := counter
8. end if
9. end do
10. end do
11. return maxcounter
12. end if

Variants of the above algorithms can be used to compute many other interesting characteristics of real-time rewriting models. This includes the minimum and maximum intermediate derivation steps in a run, by setting $p =$ True for all the "states" in the run. We can also combine delay and counting algorithms to deduce information about the bounds on the number of states in the rewriting derivations that satisfy a given property within an interval [mindelay, maxdelay].

V. Probabilistic Analysis

Probabilistic considerations play a vital role in the performance analysis of real-time systems as these systems contain a considerable amount of asynchronously initiated unpredictable events. Traditionally, formal verification techniques handle these random events by making worst-case assumptions in

order to avoid the tedious tasks of formally specifying random behaviors and formal reasoning about probabilistic properties. Due to the inherent nature of this approach, the resulting designs are developed based on excessively pessimistic criteria, which in turn reduces their overall performance. In order to solve this problem, we propose an approach that can be utilized to formally analyze probabilities associated with a property being satisfied for all possible inputs of a real-time rewriting model.

The probability of a discrete event A is usually defined in terms of its relative frequency [23].

$$\mathbb{P}(A) = \frac{\texttt{Number Of Ways Event A Can Occur}}{\texttt{Total Number Of Possible Outcomes}} \quad (1)$$

This definition can be utilized to determine the probability associated with a real-time system property p being satisfied for a rewriting model run as follows

$$\mathbb{P}(p = \mathsf{True}) \;=\; \frac{|\overline{\rho}_{t_k,p}|}{|\rho_{T_i}|} \quad (2)$$

where $\overline{\rho}_{t_k,p}$ denotes the set of runs, with initial term $t_k \in T_i$, that satisfy property p, ρ_{T_i} denotes the set of runs for all possible initial terms T_i and $|S|$ denotes the cardinality of a set S. Even though Equation 2 illustrates the fundamental concept behind the computation of the probability of success for a given property of a real-time system run, its usage for real world applications is not very straightforward due to two main reasons. First of all, executing all possible runs of a real-time systems, for the evaluation of the probability term in Equation 2, is quite time consuming, if not impossible. The situation gets worse when we consider the analysis of *real world* real-time systems where we often have to deal with a large set of parameters that affect the initial terms. For example, consider the case of an Internet routing controller that takes an IP address as its input and either directs the associated packet to its destination or sends it to another routing controller. Due to the wide range of initial terms in this case that depend on the number of possible IP addresses, it is practically impossible to consider all possible runs associated with each one of these IP addresses. The second challenge in using Equation 2 for practical purposes is to model the random input sequence from the environment of the given real-time system, which plays an important role in determining the set of runs that satisfy a given property. The range of input values is usually known but the probability of getting a given input at a particular time may or may not be equal to the probability of getting another one. It is very important to consider this kind of environmental randomness for the verification of performance related properties of real-time systems. For example, if we want to analyze the performance of a computer, it is important to take into account the type of applications that it will be required to run along with the frequency of each one. The performance measure will vary drastically depending on if it is mostly used for rough computations or for text processing. Formal modeling of this kind of random inputs from the environments, which are either very unpredictable or depend on many random choices, is usually not a very straightforward task. Now, we present an approach that allows us to use the proposed rewriting logic based approach for solving both of the above mentioned problems associated with the computation of the probability, given in Equation 2.

Generally, when we run a rewriting model, the number of possible runs depends on the number of initial terms t_i. Thus, our goal is to select a subset of initial terms in such a way that allows us to reduce the computation time without compromising the computation accuracy of $\mathbb{P}(p = True)$, given in Equation 2. We recall that a run ρ is obtained from an initial term by successive applications of the rewrite rules of a model $\mathcal{M} = (\Sigma, E \cup A, L, R)$. If we consider two runs ρ_{t_1} and ρ_{t_2} such that $[t_1]_{E\cup A} = [t_2]_{E\cup A} = \overline{t}$, where $[t_k]_{E\cup A}$ denotes the equivalence class term of t_k modulo $E \cup A$ and is called *candidate* initial term, then $\rho_{t_1} = \rho_{t_2}$ because t_1 and t_2 can be reduced to the same canonical form $\overline{t}$ with respect to the equations of $E \cup A$. This fact allows us to choose a subset of all possible initial terms for the computation of the probability in Equation 2 and thus solves the above mentioned problem. The cardinality of this subset would be a number, say N, which represents the number of disjoint equivalent classes of possible initial terms. Thus, the set $\{\rho_{t_k}\}_{1\leq k\leq N}$ is the set of runs such that every ρ_{t_k} denotes the run obtained from a particular acceptable value of an initial term t_k and corresponds to the run provided by the equivalence class term of $[t_k]_{E\cup A}$. If ρ_{t_i} and ρ_{t_j} are two runs obtained from the initial terms t_i and t_j, respectively, then $[t_i]_{E\cup A} \neq [t_j]_{E\cup A}$ for all $i, j \in \{1, \ldots, N\}$ such that $i \neq j$. The set of possible initial terms T_i can then be considered as the union of the N initial terms' canonical forms:

$$T_i \;=\; \bigcup_{k=1}^{N} [t_k]_{E\cup A} \quad (3)$$

If we consider the example of the Internet routing controller again and assume that this system treats IP addresses coming from the same server *equally*, then, the above reasoning allows us to consider one address input from each server in the network instead of all possible IP addresses. This, in turn, reduces the number of runs to be considered for the computation of success probability of a run property, significantly.

To solve the second problem with Equation 2, we propose to formally model the random input from the environment of the real-time system as a random variable. In the case of a real-time system, the set $\{\rho_{t_k}\}_{1\leq k\leq N}$ is countable because the set T_i of all possible inputs is countable, which allows us to model the random input behavior of the environment as a discrete random variable $X = \{\rho_{t_1}, \rho_{t_2}, \ldots, \rho_{t_N}\}$ over the set of runs. Now, the behavior of the environment can be integrated into our real-time rewriting model by associating a *probability mass function* (PMF) to the random variable X. The PMF gives the probability that a random variable *R* is exactly equal to some value *x*, i.e., $\hat{f}_R = \mathbb{P}(R = x)$. We assume that the PMF $\hat{f}_X$, associated with the random variable

X, reflects the behavior of passing all input values from the environment to the system model

$$\begin{array}{rccl} \hat{f}_X: & \{\rho_{t_k}\}_{1\leq k\leq N} & \longrightarrow & [0,1] \\ & \rho_{t_k} & \mapsto & \mathbb{P}(\rho=\rho_{t_k}) = \\ & & & \mathbb{P}(t_i=[t_k]_{E\cup A}) = \\ & & & f(t_k) \end{array}$$

where f is the input frequency distribution function that associates the probability of its occurrence to each input initial term.

$$\begin{array}{rccl} f: & \{t_i\}_{t_i\in T_i} & \longrightarrow & [0,1] \\ & t_k & \mapsto & \mathbb{P}(t_i=t_k) \end{array}$$

Suppose that we have a run ρ_{t_k} obtained from the candidate initial term $t_k \in T_i$. By definition, t_k represents an equivalence class of terms $t_{k_1},\ldots,t_{k_r}$ such that t_{k_i} can be reduced to t_k by the equational theory $E\cup A$ for all $1\leq i\leq r$. The term t_k exists and is unique because the equational theory $E\cup A$ is confluent and terminating. Thus, the frequency distribution of t_k can be expressed in terms of the frequency distributions of $t_{k_1},\ldots,t_{k_r}$ as:

$$f(t_k)=\sum_{i=1}^{r} f(t_{k_i})$$

Now, based on the above mentioned formalization, we are in the position of formally evaluating the performance metrics of real-time rewriting models using Equation 2. Consider a performance metric ϕ, such as minimum (maximum) delay or a minimum (maximum) property counting, to be measured for a real-time system. We define over the set of runs a function $\nu_\phi : \{\rho_{t_k}\}_{1\leq k\leq N} \longrightarrow \{0,1\}$ as follows:

$$\nu_\phi(\rho_{t_k}) = \begin{cases} 1 & \text{if } \rho_{t_k} \text{ verifies } \phi \\ 0 & \text{otherwise} \end{cases}$$

where $\{\rho_{t_k}\}_{1\leq k\leq N}$ is the set of runs and ρ_{t_k} denotes the run obtained from the candidate initial term $[t_k]_{E\cup A}\in T_i$. Now, we can formally evaluate the average value of the performance metric ϕ being satisfied in the overall system by adding the performance of each run weighted by its associated probability as follows:

$$\mathbb{E}(\phi)=\sum_{k=1}^{N} \hat{f}(\rho_{t_k})\times\nu_\phi(\rho_{t_k}) \tag{4}$$

The above results can be used to develop algorithms for performance analysis of a real-time system, based on probabilistic considerations. For instance, Algorithm 5 describes an approach for computing the probability that the maximum delay of the model does not exceed a given threshold Max. The other algorithms for computing the probability of minimum delay, minimum and maximum property counting can also be developed by making some modest modifications to the algorithm presented here. Intuitively, this algorithm computes the probability estimation provided by Equation 4. It accepts three inputs: the set of runs $\{\rho_{t_i}\}_{1\leq i\leq N}$, the frequency distribution f of an initial term and the threshold Max, and returns the probability that the maximum delay of the overall model does not exceed Max. First, it computes the frequency distribution $\hat{f}$ for every initial candidate term as the sum of the individual frequency distributions of all the initial terms that are equivalent modulo $E\cup A$ to the candidate term. Once the function $\hat{f}$ is computed, the algorithm calls the function maxdelay with Head(ρ_{t_i}) and Tail(ρ_{t_i}) as parameters to compute the maximum delay for every run of the overall system. If at any step, the maximum delay provided by ρ_{t_i} is less or equal to the threshold Max, then the probability value is incremented by the frequency distribution of ρ_{t_i}.

Algorithm 5 Probability Maximum Delay

function probmaxtime($\{\rho_{t_i}\}_{1\leq i\leq N}$, f, Max)

```
01. prob:= 0
02. for i=1 to N do
03.     t := head(ρ_{t_i})
04.     T_f := Tails(ρ_{t_i})
05.     f̂(ρ_{t_i}) := 0
06.     for all t_j ∈ [t]_{E∪A} do
07.         f̂(ρ_{t_i}) := f̂(ρ_{t_i}) + f(t_j)
08.     if maxdelay(t,T_f)≤ Max then
09.       prob := prob + f̂(ρ_{t_i})
10.     end if
11. end do
```

Algorithm 5 and its other counterparts for the minimum delay and minimum and maximum property counting can be used to analyze various interesting probabilistic properties about real-time rewriting models, which play a significant role in formally analyzing the performance of a real-time systems. To illustrate their utilization and effectiveness, we present the performance analysis of a simple search engine in the next section.

VI. Application: A Simple Search Engine

We present the performance analysis of a simple search engine system using RT-Maude. The considered system takes as input a query described as a string of characters, it looks for results that match the given string and then provides the matching results to the user.

We assume that the system memorizes the results of m different queries in its local server. In our case, we pose $m=2$ and suppose that results of query a and query b are saved in the local server of the system. The access to the local server takes a negligible amount of time, so we assume that it is performed instantly. If some query other than a or b is received, the search engine system S_0 sends it to two different exterior servers S_1 and S_2. In this case, each server processes the query and sends back the results to S_0. Since results can be different from both servers, the user is provided both sets of results by the system. We assume that the communication between S_0 and S_1 and between S_0 and S_2 requires a delay of 100 and 200 time units, respectively. The query processing

by S_1 and S_2 takes 20 and 2 time units, respectively; whereas the query processing of a and b by S_0 needs 30 and 50 time units, respectively. The RT-MAUDE description of the module is given as follows:

```
(tomod SEARCH-ENGINE is
  protecting QID.
  sort Query Answer.
  sort Server.
  subsort Query  < QID.
  subsort Answer < QID.
  op a : -> Query.
  op b : -> Query.
  op s0: -> Server.
  op s1: -> Server.
  op s2: -> Server.
  op request: Query -> Query[ctor].
  op send: Query Server -> Query Server.
  op process: Query Server -> Answer.
  op result: Answer -> Answer.
  var r: Query.}

  crl[req1] request(r) => send(r,s1) in time 100
                          if (r =/=a)/\ (r =/= b).
  crl[req2] request(r) => send(r,s2) in time 200
                          if (r =/=a) /\ (r =/=b).
  rl[reqa] request(a) => process(a,s0) in time 30.
  rl[reqb] request(b) => process(b,s0) in time 50.
  rl[sen1] send(r,s1) => process(r,s1) in time 20.
  rl[sen2] send(r,s2) => process(r,s2) in time 2 .
  rl[pro0] process(r,s0) => result(r).
  rl[pro1] process(r,s1) => result(r) in time 100.
  rl[pro2] process(r,s2) => result(r) in time 200.
 endtom)
```

After declaring two new sorts Query and Answer as subsorts of the typical sort of identifiers QID in RT-MAUDE, we declare an abstract sort Server. We define two constants a and b of sort Query that represent the two stored queries in the server and three constants s0, s1 and s2 of sort Server to represent the local server, the server S_1 and the server S_2, respectively. We also define four operators:

- request: represents the initial state, where the system reads the user's request.
- send: represents the sending of the request to servers S_1 and S_2.
- process: defines the state corresponding to the request processing.
- result: describes the state where the result is provided to the user.

The description of the module contains nine rewrite rules. The first two [req1] and [req2], are conditional rewrite rules that are applied only when the request is different from the stored ones (a and b). The first rewrite rule is delayed by 100 time units and the second by 200 time units, which represents the necessary time for the system to communicate with the servers S_1 and S_2, respectively. All other rewrite rules but [pro0] are tick rewrite rules (that model time elapsing in the system). The rewrite rule [pro0] is applied instantly and reflects our assumption regarding local server access.

Our goal is to analyze the performance of this real-time system based on timing delays. For this purpose, we assume that there is a set of N possible different queries, i.e., $Q = \{q_1, \ldots q_{N-2}, a, b\}$. We first utilize Algorithms 1 and 2, proposed in this paper, to deduce the minimum and maximum delays of a given run of the system. Table I summarizes our experimental results for the minimum and maximum delay of the runs when $N = 10$ for the given system. The set Q contains the following input ground terms $\{$q1 $\ldots$ q8, a, b$\}$ and the two parameters passed to both algorithms are the query and the result of the query provided by each run.

TABLE I
DELAYS FOR QUERIES IN THE SEARCH ENGINE

	run(q1) ... run(q8)	a	b
Minimum Delay	220	30	50
Maximum Delay	402	30	50

Next, we illustrate the process of analyzing probabilistic properties for this system. Consider a performance metric p for the system, according to which the result of a query should be provided no later than 200 time units. It is quite obvious that the overall system behavior does not support p since the minimum delay when the input is q0, for instance, is larger than 200 time units and thus the property fails, even though, this property is satisfied for the cases when the input is a or b. This fact clearly demonstrates the usefulness of using probability theory in the field of performance analysis as it allows us to obtain some information about these kind of cases where a failing property is satisfied. Using the formalization presented in Section V, we can rephrase our question as follows: "*what is the probability that the total behavior of the system satisfies p?*". The answer to this question depends on the probability distribution of the random variable that models the query generation for the search engine. If the query generation is modeled as a Uniform random variable, where the probability of acquiring each query is the same, then an obvious solution to the above question is two runs out of 10, which gives an overall system performance of 20%. Usually this is not the case for a search engine system, i.e., the queries are not equiprobable. This is when our environment modeling approach comes into account, as it allows us to model the query generation with a random variable with an appropriate PMF that assigns a frequency distribution to every query. Thus, we can compute the system's performance by applying Algorithm 5. We experimented with a frequency distribution of 0.5% and 1.5% for a and b, respectively. This gave us the overall system performance of 2%. On the other hand, with a frequency distribution of 30% each for both a and b, the overall performance of the system rose to 60%. The results of this exercise clearly indicate that the performance of the given search engine can be maximized if the results of most of the incoming queries are available in the local server.

The above example clearly demonstrates the practical effectiveness of the proposed algorithms in the formal performance analysis of rewriting models of real-time systems. We are able to formally compute delays and probabilistic quantities about a real-time rewriting model, which to the best of our knowledge cannot be done by any other existing formal method. Though,

the state-based techniques are capable of verifying probabilistic properties, as has been described in Section II of this paper, but the analysis is not as straightforward as has been the case with the proposed rewriting logic based infrastructure.

VII. Conclusions

This paper presents a novel approach for formally analyzing quantitative information about the performance of real-time systems. Our approach is based on rewriting logic and is thus expressive enough to model a variety of real-time systems. The main contributions of the paper can be classified into three categories. Firstly, we present algorithms that can be used to find the bounds on the length of the time interval between two specified rewrite rules in a rewriting model. Secondly, we present algorithms that facilitate the computation of the bounds on the number of times a property is verified in such intervals. Thirdly, based on the different runs of a rewriting model, we present algorithms that allow us to compute the probabilities associated with a property being verified for all possible input values. All these three characteristics play a vital role in performance analysis of real-time systems. Thus, the precise evaluation within a formal framework can prove to be very beneficial for the performance and reliability optimization of safety critical and highly sensitive real-time system application domains, such as medicine, military or space travel.

In order to illustrate the utilization and practical effectiveness of the proposed approach in analyzing real-time systems, we developed our algorithms in RT-Maude and used them to analyze a simple search engine model. This exercise allowed us to formally analyze many performance characteristics about the given real-time systems that, to the best of our knowledge, cannot be analyzed in such a straightforward manner using other available formal methods, such as model checking or theorem proving, due to their limited expressiveness or huge user interaction issues.

The next step is to extend our approach by developing algorithms for analyzing the performance of real-time rewriting models that are composed of many sub-models operating in parallel and exchanging data and messages. In such cases, the run of a sub-model depends not only on the input provided by the environment but also on the current behavior of the other sub-models. Also, we want to explore real-time models that interact with the environment. In this case, the input is not only a function of *static* probabilities, but also depends on the output of the model. We also plan to conduct more extensive case studies using the proposed infrastructure in this paper, such as soft real-time processing applications, where some modules are allowed to miss deadlines, and communication protocols, like the Stop-and-Wait protocol.

References

[1] B. Berthomieu and F. Vernadat. Time Petri Nets Analysis with TINA. In *International Conference on the Quantitative Evaluation of Systems*, pages 123–124. IEEE Computer Society, 2006.

[2] D.P. Bertsekas and J.N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16, 1991.

[3] D. Beyer, C. Lewerentz, and A. Noack. Rabbit: A Tool for BDD-based Verification of Real-Time Systems. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 122–125. Springer, 2003.

[4] G. Bucci, L. Sassoli, and E. Vicario. Correctness Verification and Performance Analysis of Real-Time Systems using Stochastic Preemptive Time Petri Nets. *IEEE Transactions on Software Engineering*, 31(11):913–927, 2005.

[5] S. Campos, E. M. Clarke, and O. Grumberg. Selective Quantitative Analysis and Interval Model Checking: Verifying Different Facets of a System. *Formal Methods in System Design*, 17(2):163–192, 2000.

[6] R. Cardell-Oliver. *The Formal Verification of Hard Real-time Systems*. PhD Thesis, University of Cambridge, Cambridge, UK, 1992.

[7] A. David, K.G. Larsen, G. Behrmann, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *International Conference on the Quantitative Evaluation of Systems*, pages 125–126. IEEE Computer Society, 2006.

[8] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, pages 208–219. Springer, 1995.

[9] J.H. Eaton and L.A. Zadeh. Optimal pursuit strategies in discrete state probabilistic systems. *Transactions of the ASME, D,J. Basic Eng*, 84:23–29, 1962.

[10] O. Hasan and S. Tahar. Performance Analysis and Functional Verification of the Stop-and-Wait Protocol in HOL. *Journal of Automated Reasoning*, 42(1):1–33, 2009.

[11] M. Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic Model Checking. In *Formal Methods for Performance Evaluation*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007.

[12] M. Z. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic Model Checking for Probabilistic Timed Automata. *Information and Computation*, 205(7):1027–1077, 2007.

[13] N. Marti-Oliet and J. Meseguer. Rewriting logic: Roadmap and Bibliography. *Theoretical Computer Science*, 285(2):121 – 154, 2002.

[14] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theorical Computer Science*, 96(1):73–155, 1992.

[15] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[16] J. Meseguer. Rewriting Logic as a Semantic Framework for Concurrency: a Progress Report. In *International Conference on Concurrency Theory*, pages 331–372. Springer, 1996.

[17] P. C. Ölveczky. *Specification and Analysis of Real-Time and Hybrid Systems in Rewriting Logic*. PhD thesis, University of Bergen, Norway, December 2000.

[18] P. C. Ölveczky and M. Caccamo. Formal Simulation and Analysis of the CASH Scheduling Algorithm in Real-Time Maude. In *Fundamental Approaches to Software Engineering*, volume 3922 of *LNCS*, pages 357–372. Springer, 2006.

[19] P. C. Ölveczky and J. Meseguer. Real-Time Maude 2.1. *Electronic Notes Theoretical Computer Science*, 117:285–314, 2005.

[20] P. C. Ölveczky, J. Meseguer, and Carolyn L. Talcott. Specification and Analysis of the AER/NCA Active Network Protocol Suite in Real-Time Maude. *Formal Methods in System Design*, 29(3):253–293, 2006.

[21] P. C. Ölveczky and S. Thorvaldsen. Formal Modeling and Analysis of the OGDC Wireless Sensor Network Algorithm in Real-Time Maude. In *Formal Methods for Open Object-Based Distributed Systems, Cyprus,2007*, volume 4468 of *LNCS*, pages 122–140. Springer, 2007.

[22] A. Pnueli. A Temporal Logic of Concurrent Programs. *Theoretical Computer Science*, 13(1):45–60, January 1981.

[23] S. Ross. *A First Course in Probability*. Prentice Hall, 2005.