# Further Steps Towards Efficient Runtime Verification: Handling Probabilistic Cost Models

Antonio Filieri
*Dipartimento di Elettronica e Informazione*
*Politecnico di Milano*
*Milan, Italy*
*Email: filieri@elet.polimi.it*

Carlo Ghezzi
*Dipartimento di Elettronica e Informazione*
*Politecnico di Milano*
*Milan, Italy*
*Email: carlo.ghezzi@polimi.it*

***Abstract*—We consider high-level models that specify system behaviors probabilistically and support the specification of cost attributes. Specifically, we focus on Discrete Time Markov Reward Models (D-MRMs), i.e. state machines where probabilities can be associated with transitions and rewards (costs) can be associated with states and transitions. Through probabilities we model assumptions on the behavior of environment in which an application is embedded. Rewards can instead model the cost assumptions involved in the system's operations. A system is designed to satisfy the requirements, under the given assumptions. Design-time assumptions, however, can turn out to be invalid at runtime, and therefore it is necessary to verify whether changes may lead to requirements violations. If they do, it is necessary to adapt the behavior in a self-healing manner to continue to satisfy the requirements. We have previously presented an approach to support efficient runtime probabilistic model checking of DTMCs for properties expressed in PCTL. In this paper we extend the approach to D-MRMs and reward properties. The benefits of the approach are justified both theoretically and empirically on significant test cases.**

## I. Introduction

Software engineering research has been increasingly focusing on supporting design of software that can self-adapt to changes that are discovered dynamically. This would allow systems to continue to operate even after the initial assumptions on the behavior of the environment in which they are embedded are violated, and such violations would potentially lead to violation of system's requirements [1].

In our recent research, we studied different aspects concerning the design of self-adaptive systems, from the formalization of their requirements to design-time and run-time verification of their satisfaction to adaptation mechanisms (e.g. [2], [3], [4], [5]). The needs for self-adaptation typically arise from the uncertainty about environmental assumptions that must be made at design time and also from the intrinsic variability of the context in which the application is embedded. An example of uncertainty may be the user profile in terms of submission rate of requests for a certain service, which may affect certain design choices. Other examples are unpredictable physical conditions or the interaction with third-party components that may undergo changes while they are used. Uncertainty needs to be taken care of in the design phase. Whenever possible, it may be formalized in quantitative mathematical terms via probability theory.

In our previous work, we have shown that in order to deal with these issues verification must extend to runtime and must be executed continuously. The environment must be continuously monitored to detect the relevant changes and verification must be performed to check if the changes may lead to undesirable violations of the requirements. Runtime verification in general requires different approaches and tools than the ones suitable for design time; in particular, it has to be performed within strict time bounds. In fact, if verification fails, i.e. a violation of requirements is detected, the system should immediately react and try to adapt its behavior to continue to meet the requirements. Our goal is in fact to achieve a self-healing adaptation that would prevent failures from being perceived by the system's clients ([3]). If runtime verification would take too long, then reactions might take place too late. What exactly "too late" means depends on the specific system we are dealing with. In general, however, we may safely state that the mainstream verification approaches suitable for design-time analysis cannot be transferred directly to runtime, because they do not comply with the timing requirements of runtime verification.

Although our approach is motivated by the needs of runtime verification and adaptation, it can also be used to support an agile approach to design. The various design alternatives to explore can often be modeled in the same way as the runtime changeable aspects.

This paper focuses on requirements that concern some notion of cost (such as the average execution time or energy consumption of a certain operation). During design, it is possible to model a system as a state machine, and associate costs to states and/or transitions. Probabilities may also be associated with transitions to express uncertainty about certain environment phenomena in a quantitative way. Formally, the resulting state-transition model is a *Discrete Time Markov Reward Model* (D-MRM). The requirements we wish our system to satisfy may be expressed using some logic language. Verification that the model satisfies the

requirements may be accomplished through *model checking*. State-of-the-art model checkers exist to support this approach, such as PRISM [6]. These tools provide a perfectly adequate support to the kinds of reasoning a designer needs to perform at design time. Model checkers, however, are not meant for on-line execution at runtime or continuous changes in the model.

In [4] we proposed a paradigm, called *Working Mom (WM)*, which assumes a system to be described through a Discrete Time Markov Chain (DTMC) and its desired properties expressed in the probabilistic temporal logic PCTL [7]. Properties describe a requirement to be evaluated through probabilistic model checking. In the WM approach, at design time we consider transitions to be labeled by either numeric or symbolic values. Numeric values derive from the formalization of known and stable phenomena involved with the execution of our system. Variables instead formalize phenomena that are either unknown at design time and/or subject to change. A PCTL formula describing a requirement is therefore evaluated at design time on a DTMC that can contain both numeric and symbolic transitions. The evaluation produces a *parametric* result, i.e. a formula whose truth value depends on the value of the parameters. Such formula can be evaluated as soon as the real values become available at runtime to be substituted to the parameters. The approach can also be used to explore design alternatives. Formula evaluation is extremely efficient, compared to the exhaustive state space exploration of classical model checkers.

DTMCs and PCTL are valuable formalisms to model and reason about systems from a reliability analysis standpoint, but are not able to express some other properties, such as those that refer to an abstract notion of *cost* or *reward* associated with states and/or transitions. Rewards can express the average duration or the energy consumption of a task (modeled by a state). Modeling average execution time is important to reason about performance. Likewise, modeling energy consumption can be important for applications that run on battery operated devices and, more generally, to deal with green computing concerns.

This paper illustrates how the WM approach can be extended to deal with D-MRMs. We describe the theoretical underpinnings of the extended WM approach, illustrate it on examples, and discuss—both qualitatively and quantitatively—the benefits in terms of verification efficiency.

## II. Reference Models

Here we introduce D-MRMs and R-PCTL and we discuss how they can be extended to represent changeable behaviors.

### A. Discrete Time Markov Reward Models

A D-MRM [8] is a DTMC augmented with rewards, through which one can quantify a benefit (or loss) due to the residence in a specific state or the move along a certain transition. A D-MRM has an underlying DTMC, through which designers can provide a high-level model for the system's control flow by abstracting the execution state space into a finite set of abstract states relevant to the verification[1].

A reward is a non-negative value assigned to a state or a transition. Rewards can represent information such as average execution time, power consumption, number of I/O operations, or even cost of an outsourced operation.

A D-MRM is a tuple $(S, S_0, P, L, \rho, \iota)$ where:

- $S$ is a finite set of states,
- $S_0 \subseteq S$ is a set of initial states,
- $P : S \times S \to [0,1]$ is a stochastic matrix ($\sum_{s' \in S} P(s,s') = 1 \quad \forall s \in S$). An element $P(s_i, s_j)$ represents the probability that the next state of the process will be $s_j$ given that the current state is $s_i$,
- $L : S \to 2^{AP}$ is a labeling function which assigns to each state the set of *Atomic Propositions* that are true in the state,
- $\rho : S \to \mathbb{R}_{\geq 0}$ is a *state reward* function assigning to each state a non-negative real number,
- $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a *transition reward* function assigning a non-negative real number to each transition.

To understand how rewards are gained, we need to precisely state how the system modeled by the D-MRM evolves over a sequence of time steps. At step 0 the system enters the initial state $s_0$. At step 1, the system gains the reward $\rho(s_0)$ associated with the initial state and moves to a new state (say, $s1$), gaining also the reward $\iota(s_0, s_1)$. The cumulated reward when the system enters state $s1$ is $\rho(s_0) + \iota(s_0, s_1)$. At step 2, it gains the reward $\rho(s_1)$ associated with state $s1$, and when exiting it gains also the reward associated with the chosen transition, and so on. In summary, the state reward is acquired if the D-MRM resides in state $s_i$ for one time step. The reward associated with a transition $\iota(s_i, s_j)$ is gained as the process makes a move from state $s_i$ to state $s_j$.

A state $s \in S$ is an *absorbing state* if $P(s,s) = 1$. If a D-MRM contains at least one absorbing state, the D-MRM itself is *absorbing*. If the absorbing states are reachable, in any number of time steps, from transient ones, it can be shown that any execution will eventually be absorbed with probability 1 (as proved for DTMCs in [9]). We assume D-MRMs to be well-formed, i.e. all states are reachable from the initial state and for all non absorbing states it is possible to reach a least one absorbing state.

Transitions can be defined through a matrix $P$; $P(s_i, s_j)$ is the probability associated with the transition from state $s_i$ to state $s_j$. Let us consider two distinct states $s_i$ and $s_j$. The probability of moving from $s_i$ to $s_j$ in 2 steps is $\sum_{s_x \in S} P(s_i, s_x) \cdot P(s_x, s_j)$. Generalizing to a k-steps path and

[1] The adoption of an underlying Markov model implies that the modeled system meets, with some tolerable approximation, the Markov property, according to which the probability distribution of future states depend only on the current state.

recalling the definition of matrix product, the probability of moving from any state $s_i$ to any other state $s_j$ in $k$ steps corresponds to the entry $(s_i, s_j)$ of the matrix $P^k$. As a natural generalization, we can define $P^0$ (representing the probability of moving from a state $s_i$ to a state $s_j$ in 0 steps) as the identity matrix, whose elements are 1 iff $s_i = s_j$ [9].

Variability can be modeled quite simply in D-MRMs. We assume that variability does not affect the structure of the models, only parameters. In our case, it only affects the possible values used to label transition probabilities and rewards. This is usually expressive enough to accommodate changes in the environment that affect our system.

### B. Extending PCTL with Rewards

R-PCTL is a logic language to express properties of a D-MRM. it is defined as follows [8]:

$$\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg \Phi \mid \mathscr{P}_{\bowtie p}(\Psi) \mid \mathscr{R}_{\bowtie r}(\Theta)$$
$$\Psi ::= X\ \Phi \mid \Phi\ U\ \Phi \mid \Phi\ U^{\leq t}\ \Phi$$
$$\Theta ::= I^{=k} \mid C^{\leq k} \mid F\Phi$$

Formulae $\Phi$ are named *state formulae* and can be evaluated over a boolean domain (true, false) in each state. Formulae $\Psi$ are named *path formulae* and describe a pattern that can be matched over the set of all possible paths originating in a given state. Symbol $\bowtie$ stands for a relational operator in the set $\{\leq, <, \geq, >\}$, $p \in [0,1]$ is a probability bound, $r \in \mathbb{R}_{\geq 0}$, and $k \in \mathbb{Z}_{\geq 0}$. $trueU\Phi$ can be shortened by the *eventually* operator $F\Phi$, with exactly the same semantics. The expressions defined by $\Theta$ support the specification of *reward patterns*.

Let us now informally discuss the semantics of R-PCTL, first ignoring reward formulae. The intuitive meaning of the formula $\mathscr{P}_{\bowtie p}(x)$ evaluated in a state $s$, where $x$ is a path formula, is: the probability for the set of paths originating from $s$ and satisfying $x$ meets the bound expressed as $\bowtie p$. More precisely, the satisfaction relation for (non-reward) state formulae is defined for a state $s$ as:

$$s \models true$$
$$s \models a \quad \text{iff} \quad a \in L(s)$$
$$s \models \neg\Phi \quad \text{iff} \quad s \not\models \Phi$$
$$s \models \Phi_1 \wedge \Phi_2 \quad \text{iff} \quad s \models \Phi_1 \text{ and } s \models \Phi_2$$
$$s \models \mathscr{P}_{\bowtie p}(\Psi) \quad \text{iff} \quad Pr(s \models \Psi) \bowtie p)$$

A formal definition of how to compute $Pr(s \models \Psi)$ is presented in [7]. The intuition is that its value corresponds to the probability of taking a path that satisfies $\Psi$, among all the, possibly infinite, paths originating in $s$. The satisfaction relation for a path formula with respect to a path $\pi$ originating in $s$ ($\pi[0] = s$) is defined as:

$$\pi \models X\Phi \quad \text{iff} \quad \pi[1] \models \Phi$$
$$\pi \models \Phi U \Psi \quad \text{iff} \quad \exists j \geq 0.(\pi[j] \models \Psi \wedge (\forall 0 \leq k < j.\pi[k] \models \Phi))$$
$$\pi \models \Phi U^{\leq t} \Psi \quad \text{iff} \quad \exists 0 \leq j \leq t.(\pi[j] \models \Psi \wedge (\forall 0 \leq k < j.\pi[k] \models \Phi))$$

Let us now focus on the semantics of the rewards fragment of R-PCTL. We intuitively define how a state $s$ can satisfy a formula $\mathscr{R}_{\bowtie r}(x)$ depending on the way the reward expression $x$ is formulated.

- $\mathscr{R}_{\bowtie r}(I^{=k})$ is true in state $s$ if the expected state reward to be gained in the state entered at step $k$ along the paths originating in $s$ meets the bound $\bowtie r$.
- $\mathscr{R}_{\bowtie r}(C^{\leq k})$ is true in state $s$ if, from state $s$, the expected reward *cumulated* after $k$ steps meets the bound $\bowtie r$.
- $\mathscr{R}_{\bowtie r}(F\Phi)$ is true in state $s$ if, from state $s$, the expected reward cumulated before a state satisfying $\Phi$ is reached meets the bound $\bowtie r$.

The third construct can be used, for example, to state the global costs of the running systems, that is, until the execution reaches a *completion* state, usually modeled by an absorbing state because of its definitive nature.

A formal semantics for the reward fragment of R-PCTL can be found in [8]. Intuitively, the expected reward $\mathscr{R}(\theta)$ for all possible paths exiting a given state $s$ and satisfying the pattern $\theta$ can be computed as the sum of the rewards for each of those paths, weighted by the probability for that path to be taken. Even in case the set of paths originating from $s$ is infinite, the resulting infinite series can be proved to converge [7]. Note that the probability for a path to be taken is the joint probability of all its transitions to fire, which can be computed as the product of the probabilities associated with the transitions thanks to the Markov assumption[9]. The expected value $X$ for the reward can be computed for a given path $\omega = s_0 s_1 s_2 \ldots$ ) and for a given pattern:

$$X_{I^{=k}}(\omega) = \rho(s_k) \tag{1}$$

$$X_{C^{\leq k}} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} \rho(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases} \tag{2}$$

$$X_{F\Phi} = \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i\ s_i \not\models \Phi \\ \sum_{i=0}^{min\{j|s_j \models \Phi\}-1} & \\ \qquad \rho(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases} \tag{3}$$

In Section IV we will show how the value of $X_\Theta$ can be computed with algebraic techniques taking into account the presence of both numeric values and variable parameters in the D-MRM model.

## III. Example

We introduce a running example, taken from [3]. Figure 1 shows an activity diagram which describes a workflow of a typical e-commerce application that sells on-line merchandise to its customers by integrating the following components, exposed by third-party service providers: (1) *Authentication Service*, (2) *Payment Service*, and (3) *Shipping Service*. The diagram provides an abstract operational specification of the system to be implemented. The *Authentication Service* manages the identity of users, via

a *Login* and a *Logout* operations. The *Payment Service* provides a secure transactional payment via a *CheckOut* operation. The *Shipping Service* is in charge of shipping goods. It provides two different operations: *Normal Shipping* and *Express Shipping*. The former is a standard shipping functionality while the latter represents a faster and more expensive alternative. The system also classifies the logged users as *Returning Customers* (RC) or *New Customers* (NC), in order to provide targeted quality of service.

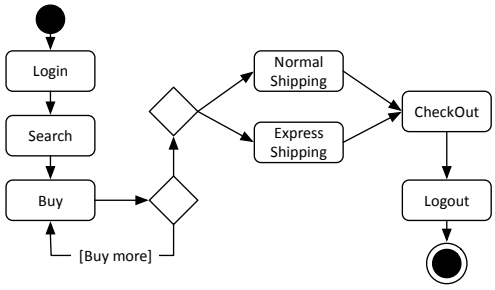


Figure 1. Activity diagram of the example application.

Figure 2 shows a D-MRM model for the example, derived from the activity diagram in Figure 1 (rewards are omitted for readability reasons, and will be introduced later.) The model describes both the possible sequences of operations supported by the application and the environment assumptions that concern user profiles (e.g., the probability that a user is a returning customer, the probability that new customers choose express shipping, etc.)or the behavior of the external components (e.g., the probability of failure of the authentication service.) These environment assumptions are annotated as probabilities on the transitions of the D-MRM. Notice that some probabilities are indicated as parametric, to indicate that their value is subject to change. Additional environment assumptions may concern costs.

Let us assume that the application will run on a platform-as-a-service cloud environment, such as the Google AppEngine, whose pricing policies (used to derive the rewards annotating the D-MRM states) are based on factors such as *average cpu time* and *average bandwidth*[2].

To compute the rewards, we may assume that *login* and *logout* run on a front-end instance, *profiler* and *normal shipping* run on a back-end of class B1, *express shipping* and *checkout* on a back-end of class B2. In addition, all the other operations concerning new customers run on class B1 instances, while the ones concerning returning customers run on instances B2. From these assumptions, the designer may easily derive the rewards to be associated with each state of the D-MRM in Figure 2. Precisely, the reward may be computed by the formula $e \cdot c_c + b \cdot c_b$, where:

[2]From http://code.google.com/appengine/docs/billing.html we have taken the following costs: outgoing bandwidth 0.12\$/*Gb*, Frontend instance (F2) 0.16\$/*h*, Backend Instance (B1) 0.08\$/*h*, Backend Instance (B2) 0.16\$/*h*.

- $e$ is the average execution time (in hours) of the task modeled by the state
- $c_c$ is the hourly cost of the instance running it
- $b$ is the average bandwidth consumed in *Gb*
- $c_b$ is the cost of outgoing bandwidth per *Gb*

Concerning variability, we assume that the values for $e$ and $b$ are subject to change; $c_c$ might also change, corresponding to different deployment choices which may occur at runtime on this kind of platforms.

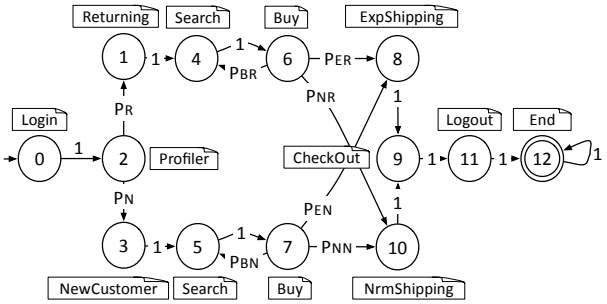


Figure 2. D-MRM of the example.

Our goal is to support dynamic verification, as changes are detected which deviate from the numeric values assumed at design time. As we mentioned, the fact that certain values may change is represented by using parameters, instead of numeric values. For example, in Figure 2, we use parameters to indicate that user profiles may change, e.g. the probability that a user is a returning customer.

The next section describes how the evaluation of a ePCTL formula, expressing a property of a parametric D-MRM, can be evaluated to produce a verification formula that can be efficiently evaluated at runtime once the real values of the parameters become available.

## IV. Algebraic Rewards Analysis

Equations (1), (2), and (3) formalize the meaning of rewards formulae. In this section we provide an algebraic formulation and discuss its complexity.

We start by showing how to compute the expected reward along the set of all the possible paths originating in a state $s_i$ of a D-MRM. In Section II-B it has been defined as the sum of the rewards for each path originating in $s_i$, weighted by the probability for that path to be taken. Such a sum may contains infinite terms as the number of paths originating in $s_i$ may be infinite. Due to the Markov assumption, the computation of such expected reward can be restated in a compact form, better suited for algebraic solution. Indeed, the expected reward for a (non empty) path originating in $s_i$ can be seen as the sum of the reward gained in state $s_i$ plus the expected reward to be gained in each of the possible next states, weighted by the probability of moving toward it. The resulting formulation implemented by state-of-the-art model checkers (e.g. [7], [8]), follows:

$$r_i = \rho(s_i) + \sum_{s_j \in S} p_{ij} * (\iota(s_i, s_j) + r_j) \quad (4)$$

where $r_i$ is the expected reward from $s_i$ and $p_{ij}$ is the probability of moving from state $s_i$ to state $s_j$.

To simplify the exposition, let us make two preliminary assumptions. First, we assume that for formulae $F\Phi$, $\Phi$ can only be a (boolean composition of) atomic propositions, that is it the operators $\mathscr{P}$ and $\mathscr{R}$ do not appear in $\Phi$. An extension to generic $\Phi$ can be done similarly to the treatment of nested formulae in [4]. Second, we focus on state rewards only. Transition rewards can always be mapped into state rewards of a modified D-MRM automatically.

The expected reward for formulae (1), (2), and (3) can be computed by specializing the equations system (4) in convenient linear algebraic procedures discussed below.

From Equation (1), $X_{I=k}$ is computed as the sum of the rewards of every state reached in exactly $k$ time steps, weighted by the probability of reaching it. The probability of being in state $s_j$ after exactly $k$ steps, given that the process started from state $s_i$, is the entry $(s_i, s_j)$ of the matrix $P^k$ [9]. Hence, let $\bar{\rho}$ be the column vector $[\rho(s_0), \rho(s_1), \rho(s_2), \ldots]$, the expected reward for a formula $R_{I=k}$ can be computed as:

$$X_{I=k} = P^k \cdot \bar{\rho} \Big\|_0 \quad (5)$$

where $X\|_i$ is the $i$-th element of $X$; 0 is the initial state.

To compute $X_{C \leq k}$, we need to sum the expected rewards over all possible paths up to $k$ time steps. For previous considerations, the expected reward at step $k$ is exactly $P^k \cdot \bar{\rho}$. Hence, the cumulated reward up to step $k$ excluded is:

$$X_{C \leq k} = \sum_{i=0}^{k-1} P^i \cdot \bar{\rho} \Big\|_0 \quad (6)$$

To evaluate $X_{F\Phi}$, recalling (3) and (4), we can define the computation of the expected value for the reward formula as follows:

$$r_i = \begin{cases} 0 & \text{if } s_i \models \Phi \\ \infty & \text{if } s_i \text{ is absorbing and } s_i \not\models \Phi \\ \rho(s_i) + \sum_{s_j \in S} p_{ij} \cdot r_j & \text{otherwise} \end{cases} \quad (7)$$

From (5), (6), and (7) we derive polynomials on variables that represents the parametric solution of the verification formulae. These polynomials can be used for fast verification by replacing parameters with actual values.

### A. Computational Aspects

The WM paradigm splits the verification process into a design-time and a run-time phase. At run time, verification of R-PCTL properties consist of evaluating polynomial formulae. Even though the number of terms of polynomials may become quite large, the actual time needed for evaluation has been shown (for DTMC and PCTL) to be orders of magnitude faster than model-checking procedure [4]. Here we focus on design-time complexity of the WM approach.

Concerning formulae $R[I^{=k}]$ and $R[C^{\leq k}]$, the core of the design-time phase consists of computing powers of a $n \times n$ matrix, where $n$ is the number of states of the D-MRM. The number of matrix multiplications is $O(log(k))$, for $R[I^{=k}]$, and $O(k)$, for $R[C^{\leq k}]$ [10]. Matrix multiplication has been extensively studied in computational algebra and its complexity upper-bound for dense matrices has been shown to be $O(n^{2.496})$ on a sequential machine [10]. The fastest published algorithm was proposed in 1990 by Coppersmith and Winograd and reaches $O(n^{2.376})$ [11]. However, in practice in our context, transition matrices are quite sparse, since software components often are connected to only a small fraction of the others. In this case the complexity of matrix multiplication can improve up to $O(m)$, where $m$ is the number of non zero elements [12]. The cost of each elementary operation in this matrix multiplication process depends on type of the matrix entries involved. Some of them are numeric, others symbolic. The two cases lead to different complexities. The cost of an operation between two numbers depends on their representation and thus on the desired precision. If a fixed (though high) precision is enough, the cost of a single operation is a constant. If the numbers are represented by infinite precision rationals, the complexity of operations among them, which depends on the number of digits $d$ required for their representation, is in the order of $O(log(d))$ [13].

The design-time partial evaluation of a $R[F\ \Phi]$ property on a D-MRM has been reduced to the solution of the linear equations system (7). The concrete execution time depends on several factors. One is the size of the equations system. Indeed the complexity for solving a $n \times n$ linear system of equations is $O(n^3)$ elementary operations between its coefficients on a sequential machine, even though it is heavily parallelizable [14]. As before, the complexity of each elementary operation instead depends on the type of the operands involved. Finally, specific matrix structure may support heuristics that reduce computational time even further. $R[F\ \Phi]$ may also benefit from the sparsity of the transition matrix [15].

## V. Empirical Evaluation

Here we focus on the analysis of $R[F\ \Phi]$ formulae because of their greater interest in many practical cases.

The solution of the equation system in (7), for the verification of $R[F\ \Phi]$, has been implemented in Maple 15, the implementation of our algorithm, and the test cases discussed in this section can be downloaded from http://home.dei.polimi.it/filieri/2012formsera. The sparsity of the linear system has been exploited to obtain a fast computation. For the solution of the system we used direct methods (e.g. [15]) instead of the iterative ones (popular in probabilistic model-checkers) because they do not loose precision and they can deal with

symbolic coefficients. We implemented a solver supporting symbolic parameters by extending Pierce's algorithm[3]. The algorithm is based on the combined application of two techniques: structured Gaussian elimination and Markowitz pivoting[15]. Structured Gaussian elimination is a variation of the popular method to triangularize linear systems, which allows the solution of a large sparse system to be reduced to the case of a very small dense one. Empirical studies show that extremely sparse systems will often collapse to corresponding very small dense systems, and this seems to be the case for most of the sample cases analyzed in this section. Markovitz pivoting is a strategy to select the order in which matrix elements will be eliminated in the Gaussian elimination in order to reduce the fill-in. Mathematical details are out of the scope of this paper. The interested reader may refer for example to[15].

To avoid any loss of precision, we used rational coefficients. All the test cases have been generated randomly. The algorithm to generate them is available online. Each test suite is identified by the random seed used to initialize the random generator, to make all cases replicable.

We will study the complexity trend with respect to two dimensions of the problem: number of states, and number of symbolic parameters. The former accounts for the *size* of the model, the latter for the intensity of symbolic computation. All test cases have exactly 2 absorbing states, corresponding to *successful completion* (S) and *unrepairable failure* (F), respectively. Each transient state has 5 outgoing transitions The reward value we considered is $\mathbb{R}[F\ state = S]$.

The execution environment is a Dual Intel(R) Xeon(R) CPU E5530 @ 2.40GHz with 8Gb of ram, equipped with GNU Linux Ubuntu server 11.04 64bit. All the tests considered in this section did not overran the available memory. All the plotted graphs show the average execution time with a thick black line and the maximum measured execution time in a dashed thin line.

Figure 3 shows the execution time with respect to the size of the model in number of states. For each model there are 15 parametric transitions and 5 parametric state rewards. The total number of test cases is 4132. Even for the largest systems, the WM maximum execution time did not exceed $50s$.

Figure 4 shows how design-time computation growths in complexity with the number of parameters in the model. All the test cases have 100 states. The number of parameters affects design-time performance; however, our algorithm has been able to perform the requested verification tasks within at most 5 minutes on a general purpose hardware.

We now provide a brief comparison with PARAM [16]. To the best of our knowledge, PARAM is the only other available parametric tool[4]. We disabled bisimulation abstrac-

[3]http://www.cecm.sfu.ca/~rpearcea/sge/sge.mpl

[4]We chose version 1.8.

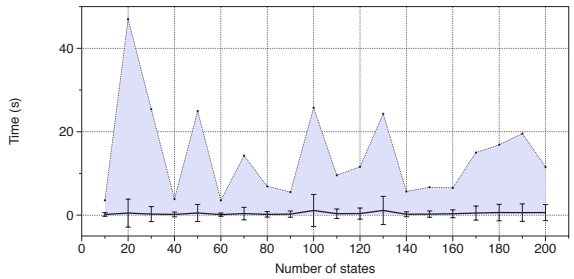


Figure 3. Design-time computation vs number of states.

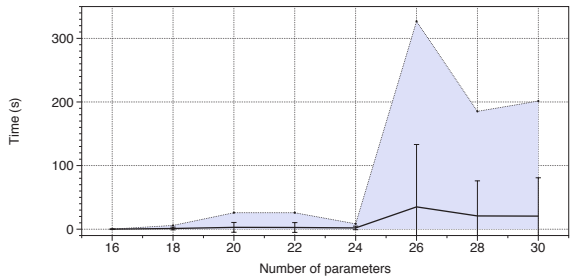


Figure 4. Design-time computation vs number of parameters.

tion in PARAM because we aim at comparing the pure parametric verification of the tool with respect to WM. The execution environment is the same as before and we used the compiled version of PARAM for GNU Linux 64bit[5].

Table I
WORKING MOM VS PARAM MAXIMUM EXECUTION TIMES (5 SAMPLES).

| | | States | |
|---|---|---|---|
| | | 50 | 100 |
| Pars | 13 | 24030.4 | 0.081887 |
| | 23 | 37602.8 | — |

*PARAM*

| | | States | |
|---|---|---|---|
| | | 50 | 100 |
| Pars | 13 | 0.11 | 0.09 |
| | 23 | 1.01 | 96.77 |

*WM*

Table I shows our preliminary results, in which WM performs much better than PARAM in terms of verification time. To be fair, PARAM is more efficient in terms of memory consumption, which could, in our experiments, be up to 10 times less (it never exceeded 500Mb).

In both cases, runtime computation is extremely efficient, since it is just the evaluation of a polynomial it actually takes $10^{-3}$ to $10^{-2}$ seconds [4]. Furthermore, it is simple enough to be implemented on low power devices such as mobile phones.

## VI. RELATED WORK

This paper described a parametric model-checking approach for D-MRMs and R-PCTL. Parametric model-checking for Markov models has been originally introduced

[5]It has to be noticed that PARAM may still be the choice in case of reduced memory availability, since in all of our test cases its memory consumption never exceeded 3Gb while Maple did.

in [17]. This seminal paper focused on DTMCs and PCTL reachability formulae. The core of that algorithm resembles the synthesis of regular expressions for finite state automata. It produces stochastic regular expressions from DTMCs which yield a parametric solution of the verification problem. The length of the stochastic expression has been proved to be $O(n^{log(n)})$ ($n$ is the number of states) [17]. In its original formulation, Daws did not provide support for rewards. In [16], the authors provide an efficient implementation of Daws' algorithm which combines state space reduction techniques and early evaluation of the regular expression in order to improve the actual execution time when only a few variable parameters appear in the model. The improvement in [16] requires $n^3$ algebraic operations among polynomials, performing better than [17] in most practical cases, although still leading to a $O(n^{log(n)})$ long expression in the worst case. The approach in [16] has been implemented in the tool PARAM 1.8 . This tool does not allow for the nesting of temporal operators, but supports numeric and symbolic rewards and related formulae. Param has been briefly compared with WM in Section V. Further and more complete comparison needs to be performed and is subject of ongoing work.

In [18] a new version of PARAM has been presented, which supports Markov Decision Processes (MDPs), a superset of Markov chains where non-deterministic transitions are allowed too. PARAM 2 applies techniques for state space exploration in order to partition the domain of parameters into hyper-regions where the desired properties are true or false. The output of the tool is the set of regions where the property holds, not a mathematical formula.

## VII. Conclusions

In this paper we extended the WM run-time verification approach to D-MRMs, which model both variable transitions and variable rewards. We have shown that our implementation based on Maple is suitable for reasonably efficient design-time analysis even on general purpose hardware. The resulting polynomial form can be evaluated quite efficiently by replacing the parameters with their actual values and without requiring the execution of any complex mathematical routines. This enables efficient run-time verification even on low power mobile devices. Another application of the WM approach concerns agile design-space exploration to support analysis of the effect of different environmental assumptions. The implementation of the design-time solver in Maple 15 is available for use V.

The experimental assessment presented in Section V shows that the number of states and number of parameters do not satisfactorily characterize the performance of the WM verification algorithm. This is clear by looking at the difference between maximum and average execution time, which is an evidence of the large variance of the test results. This suggests that the dependency of design-time performance on the topology of the D-MRM should be further investigated.

## Acknowledgments

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom.

## References

[1] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems*, ser. LNCS, vol. 5525. Springer, 2009.

[2] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *ICSE*, 2009.

[3] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, pp. 1–24, 2011.

[4] ——, "Run-time efficient probabilistic model checking," in *ICSE*. New York, NY, USA: ACM, 2011, pp. 341–350.

[5] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements," in *ASE 2011*, nov. 2011, pp. 283 –292.

[6] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in *Computer Performance Evaluation: Modelling Techniques and Tools*, ser. LNCS. Springer, 2002, vol. 2324, pp. 113–140.

[7] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[8] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for Performance Evaluation*, ser. LNCS. Springer, 2007, vol. 4486, pp. 220–270.

[9] S. Ross, *Stochastic Processes*. Wiley New York, 1996.

[10] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical mathematics*. Springer Verlag, 2007, vol. 37.

[11] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251 – 280, 1990.

[12] R. Yuster and U. Zwick, "Fast sparse matrix multiplication," *ACM Trans. Algorithms*, vol. 1, pp. 2–13, July 2005.

[13] H. T. Kung, "The computational complexity of algebraic numbers," in *Proceedings of the fifth annual ACM symposium on Theory of computing*, ser. STOC '73. New York, NY, USA: ACM, 1973, pp. 152–159.

[14] A. Bojanczyk, "Complexity of solving linear systems in different models of computation," *SIAM Journal on Numerical Analysis*, vol. 21, no. 3, pp. 591–603, 1984.

[15] T. Davis, *Direct methods for sparse linear systems*. Society for Industrial Mathematics, 2006, vol. 2.

[16] E. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric markov models," *Model Checking Software*, pp. 88–106, 2009.

[17] C. Daws, "Symbolic and parametric model checking of discrete-time markov chains," *ICTAC*, pp. 280–294, 2005.

[18] E. Hahn, T. Han, and L. Zhang, "Synthesis for pctl in parametric markov decision processes," in *NASA FM*. Springer, 2011, vol. 6617, pp. 146–161.