Probabilistic Model Checking and Reliability of Results*

Ralf Wimmer Alexander Kortus Marc Herbstritt Bernd Becker

Institute of Computer Science,
Albert-Ludwigs-University,
79110 Freiburg im Breisgau, Germany
{wimmer | kortus | herbstri | becker}@informatik.uni-freiburg.de

***Abstract*—In formal verification, *reliable results* are of utmost importance. In model checking of digital systems, mainly incorrect implementations of the model checking algorithms due to logical errors are the source of wrong results. In probabilistic model checking, however, numerical instabilities are an additional source for inconsistent results.**

We motivate our investigations with an example, for which several state-of-the-art probabilistic model checking tools give completely wrong results due to inexact computations. We then analyze, at which points inaccuracies are introduced during the model checking process. We discuss first ideas how, in spite of these inaccuracies, reliable results can be obtained or at least the user be warned about potential correctness problems: (1) usage of exact (rational) arithmetic, (2) usage of interval arithmetic to obtain safe approximations of the actual probabilities, (3) provision of certificates which testify that the result is correct, and (4) integration of a "degree of belief" for each sub-formula into existing model checking tools.

I. Introduction

While for traditional decision problems arising in computer science a binary true/false answer is intuitively understandable, in the domain of probabilistic systems the interpretation of probability values can be very cumbersome. But our nature is inherently stochastic and hence we have to cope with stochastic systems by modelling them, e. g., with discrete-time Markov chains (DTMCs). At the very end of any method for analyzing stochastic systems, there has to be a representation of probabilistic values within the computer. Typically, the architecture of today's computers provide a *floating-point* representation for real numbers, most prominently established by the IEEE 754 standard specification [1].

Model checking as a verification method enables the separation of the system model from the properties that specify the correct behavior of the system. Model checking has been investigated very deeply in the last 25 years and has become a mature verification methodology pushing forward the frontiers for both large industrial systems (e. g., microprocessors) and novel academic models (e. g., hybrid systems). In the last 10 years, stochastic model checking has been in the focus of intense research and besides enormous advances w. r. t. probabilistic models and logics that can be handled algorithmically, it has also reached the usage within industrial applications.

There are several academic tools available for stochastic model checking. We are only aware of two publications about computing probabilities in a reliable way: [2] computes regular expressions in a symbolic way from which the probabilities can be derived using only addition and multiplication, which can be performed with rational arithmetic. However, this approach cannot cope with nested PCTL formulae and we do not expect it to scale well. The second paper [3] deals with performance analysis for compositional probabilistic I/O automata, but without comparison to inexact arithmetic.

All other state-of-the-art tools like PRISM [4] and MRMC [5], to name only two of the most popular ones, rely on inexact floating-point arithmetic. Especially in the context of probabilistic model checking this topic is often euphemized by stating that the probability values of the model are derived from natural observations which itself are inherently stochastic. But this argument does not give the permission to allow inadequate computations *after* the probability values of the model were agreed to be the most accurate values available. Recently, Haverkort et al. [6] gave a detailed experimental analysis of the numerous effects caused by the usage of inexact arithmetic for computing steady-state probabilities of large Markov chains. But they do not analyze PCTL model checking and they do not provide any solutions to the problems they encountered.

The main topic of this paper is therefore to discuss the impact of inexact computations on the result of model checking. We will give an example for which at least two state-of-the-art tools compute completely wrong probabilities—1.0 instead of 0.0. These results clearly reveal the demand for *reliable results* either by exact computations, safe approximations of the correct probabilities or—if possible—by providing certificates for the correctness.

This paper consists of the following parts: First, we review the basic definitions of discrete-time Markov chains (DTMCs), the logic PCTL, and the algorithms for model checking PCTL formulae on DTMCs. We motivate our investigation by providing an example for which inexact arithmetic is definitively inappropriate. The next section is devoted to the analysis, at which points of the model checking process inaccuracies

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See www.avacs.org for more information.

are introduced, and try to give ideas how these inaccuracies can either be avoided or how reliable results can be obtained in spite of inexact computations. The paper closes with a conclusion and an outlook to future work.

II. Foundations of DTMC Model Checking

In this section we will briefly recall the definitions of discrete-time Markov chains (DTMCs), the model we will focus on in this paper, and the logic PCTL, which is commonly used for the specification of properties. We will also sketch the algorithms for checking if a DTMC exhibits a property specified in PCTL.

A. Discrete-time Markov Chains

One of the simplest models in probabilistic model checking are discrete-time Markov chains. They are essentially transition systems in which the transitions are labelled with the probability to walk from its source state to the target state. This probability is independent of the way the source state was reached (so-called Markov property).

Definition 1 *Let AP be a fixed set of atomic propositions. A* discrete-time Markov chain *(DTMC) is a tuple $M = (S, P, L)$ such that S is a finite, non-empty set of states, $P : S \times S \rightarrow [0, 1]$ the matrix of transition probabilities, and $L : S \rightarrow 2^{AP}$ a labelling function which assigns each state the set of propositions which are satisfied in that state.*

P has to be a stochastic matrix, i. e., for each state $s \in S$ the condition $\sum_{s' \in S} P(s, s') = 1$ has to be satisfied.

A path of M is a finite or infinite sequence $\pi = s_0 s_1 s_2 \cdots$ of states such that $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. We denote the i-th state of π by π^i (i. e., $\pi^i = s_i$) and the i-th prefix by $\pi{\uparrow}^i = s_0 s_1 \cdots s_i$. The number of states on a finite path π is $|\pi|$. Path_s is the set of infinite paths of M starting in state s.

Following Markov chain theory, we now define the probability space on the set of paths starting in state $s_0 \in S$ as the unique measure on the σ-algebra where the basic cylinders are induced by the finite paths starting in s_0 and the probability measure by

$$\mathrm{Pr}_{s_0}(\{\underbrace{\pi \in \mathrm{Path}_{s_0} \mid \pi{\uparrow}^n = s_0 s_1 \cdots s_n}_{\text{basic cylinder of } s_0 s_1 \cdots s_n}\}) = \prod_{i=0}^{n-1} P(s_i, s_{i+1}).$$

We now illustrate the main concepts in the following example:

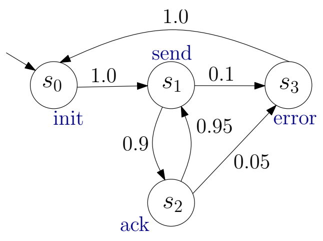

Fig. 1. A discrete-time Markov chain

Example 1 *In Fig. 1, you can see a DTMC modelling a very simple communication protocol. First, an initialization is performed, then data blocks are sent and the process waits for an acknowledgment. This normal operation can be interrupted by an error, which occurs with probability 0.1 (when sending) and 0.05 (when waiting for an acknowledgment). After an error, the initialization has to be repeated.*

Let us consider the finite path $\pi = s_0 s_1 s_2 s_1 s_2 s_1$ which is taken if two data packets are transmitted without being interrupted by an error. Its probability is $1.0 \cdot 0.9 \cdot 0.95 \cdot 0.9 \cdot 0.95 = 0.731025$.

B. Probabilistic Computation Tree Logic

After the formal introduction of the models, we still need a formal language to describe the properties which we want to verify. The most common language is probabilistic computation tree logic (PCTL), which was introduced by Hansson and Jonsson in [7].

In the following, we will briefly define syntax and semantics of PCTL before we turn to the model checking algorithms.

Definition 2 (Syntax of PCTL) *Let AP be a fixed set of atomic propositions, $a \in AP$, $\bowtie \in \{<, \leq, >, \geq\}$, $k \in \mathbb{N}$, and $p \in [0, 1]$. PCTL state formulae are then given by*

$$\phi ::= \mathrm{true} \mid a \mid \neg\phi \mid (\phi \wedge \phi) \mid \mathcal{P}_{\bowtie p}(\psi)$$

where ψ is a path formula. PCTL path formulae are created by the following grammar:

$$\psi ::= \mathcal{X}\,\phi \mid \phi\,\mathcal{U}\,\phi \mid \phi\,\mathcal{U}^{\leq k}\,\phi.$$

Definition 3 (Semantics of PCTL) *Let $M = (S, P, L)$ be a DTMC, $a \in AP$, $\bowtie \in \{<, \leq, >, \geq\}$, $s \in S$, and $\phi, \phi_1, \phi_2, \psi$ PCTL (state/path) formulae. We define the satisfaction relation $\models$ recursively as follows:*

$$\begin{array}{ll}
s \models \mathrm{true} & \\
s \models a & \textit{iff } a \in L(s) \\
s \models \neg\phi & \textit{iff } s \not\models \phi \\
s \models (\phi_1 \wedge \phi_2) & \textit{iff } s \models \phi_1 \textit{ and } s \models \phi_2 \\
s \models \mathcal{P}_{\bowtie p}(\psi) & \textit{iff } \mathrm{Pr}_s\big(\{\pi \in \mathrm{Path}_s \mid \pi \models \psi\}\big) \bowtie p \\
\pi \models \mathcal{X}\,\phi & \textit{iff } \pi^1 \models \phi \\
\pi \models \phi_1\,\mathcal{U}\,\phi_2 & \textit{iff } \exists i \geq 0 : (\pi^i \models \phi_2 \wedge \forall j < i : \pi^j \models \phi_1) \\
\pi \models \phi_1\,\mathcal{U}^{\leq k}\,\phi_2 & \textit{iff } \exists i \leq k : (\pi^i \models \phi_2 \wedge \forall j < i : \pi^j \models \phi_1).
\end{array}$$

C. Model Checking PCTL

Up to now, we have introduced discrete-time Markov chains as our system models and the logic PCTL for the description of desired properties. In this section we will show how to compute the states which satisfy a given PCTL formula. We will concentrate on the main principles which are necessary to understand where inaccuracies are introduced. For more details on PCTL model checking see e. g. [8], [9].

Like model checking for CTL, model checking for PCTL is based on recursively traversing the syntax tree of the formula bottom-up and computing the set $\mathrm{Sat}(\phi) = \{s \in S \mid s \models \phi\}$ for each state sub-formula ϕ. This can be done as follows (a denotes an atomic proposition; ϕ, ϕ_1, and ϕ_2, PCTL state formulae; ψ, a PCTL path formula; and $p \in [0,1]$, a real number):

$$\begin{aligned}\mathrm{Sat}(\mathrm{true}) &= S\\ \mathrm{Sat}(a) &= \{s \in S \mid a \in L(s)\}\\ \mathrm{Sat}(\neg\phi) &= S \setminus \mathrm{Sat}(\phi)\\ \mathrm{Sat}\big((\phi_1 \wedge \phi_2)\big) &= \mathrm{Sat}(\phi_1) \cap \mathrm{Sat}(\phi_2)\\ \mathrm{Sat}\big(\mathcal{P}_{\bowtie p}(\psi)\big) &= \{s \in S \mid \Pr(s,\psi) \bowtie p\}\end{aligned}$$

Hereby, $\Pr(s,\psi)$ denotes the probability $\Pr_s\big(\{\pi \in \mathrm{Path}_s \mid \pi \models \psi\}\big)$. The remaining task is consequently the computation of $\Pr(s,\psi)$. Depending on the path quantifier ($\mathcal{X}$, $\mathcal{U}^{\leq k}$, $\mathcal{U}$), we distinguish three cases.

1) Next Quantifier ($\mathcal{X}$): Given the set $\mathrm{Sat}(\phi)$, the probability $\Pr(s, \mathcal{X}\,\phi)$ can be computed as follows:

$$\Pr(s, \mathcal{X}\,\phi) = \sum_{s' \in \mathrm{Sat}(\phi)} P(s,s').$$

2) Bounded Until Quantifier ($\mathcal{U}^{\leq k}$): We can characterize the bounded until operator with the following recursive equation system:

$\Pr(s, \phi_1\,\mathcal{U}^{\leq k}\,\phi_2) = 1$ if $s \in \mathrm{Sat}(\phi_2)$, $\Pr(s, \phi_1\,\mathcal{U}^{\leq k}\,\phi_2) = 0$ if $s \notin \mathrm{Sat}(\phi_1)$ and $s \notin \mathrm{Sat}(\phi_2)$ or if $k = 0$ and $s \notin \mathrm{Sat}(\phi_2)$. In all other cases, we have

$$\Pr(s, \phi_1\,\mathcal{U}^{\leq k}\,\phi_2) = \sum_{s' \in S} P(s,s') \cdot \Pr(s', \phi_1\,\mathcal{U}^{\leq k-1}\,\phi_2).$$

The intuition behind this equation system is the following: if s satisfies ϕ_2, all paths starting in s fulfill $\psi := \phi_1\,\mathcal{U}^{\leq k}\,\phi_2$; hence, the probability is 1. If s satisfies neither ϕ_1 nor ϕ_2, ψ cannot be satisfied on any path starting in s; accordingly, the probability is set to 0. The same holds if s does not satisfy ϕ_2 and $k = 0$, such that no further steps may be taken. Otherwise, we may walk one step and fulfill the formula in one step less.

3) Unbounded Until Quantifier ($\mathcal{U}$): The unbounded until operator can be characterized in a similar way as the bounded operator. But in this case the characterization leads to a linear equation system:

We partition the state space into three sets S^0, S^1, and $S^?$ such that $\Pr(s, \phi_1\,\mathcal{U}\,\phi_2) = 1$, if $s \in S^1$, and 0, if $s \in S^0$. S^0 contains all those states from which no path leads to a ϕ_2 state while passing only ϕ_1-states. Conversely, S^1 is the set of ϕ_1-states which do not have a path to a state in S^0. $S^?$ contains all states not in S^0 or S^1. These sets can be computed efficiently using graph traversal algorithms, see e. g. [8].

The probability that in state s a path π is taken with $\pi \models \phi_1\,\mathcal{U}\,\phi_2$ is then given by the unique solution of the following system of linear equations:

$$\Pr(s, \phi_1\,\mathcal{U}\,\phi_2) = \begin{cases} 1 & \text{if } s \in S^1\\ 0 & \text{if } s \in S^0\\ \sum\limits_{s' \in S} P(s,s') \cdot \Pr(s', \phi_1\,\mathcal{U}\,\phi_2) & \text{if } s \in S^?. \end{cases}$$

The intuition behind this equation system is the same as in the case of bounded until. The difference is that we do not have to take a bound on the number of steps into account.

Such linear equation systems are usually solved using iterative methods like Jacobi, Gauß-Seidel, or over-relaxation methods. Starting with an initial estimation x^0 of the solution, they iteratively compute more precise approximations until some convergence criterion is satisfied, e. g. until $\|x^{(i)} - x^{(i-1)}\| < \varepsilon$ for some $\varepsilon > 0$ and a norm $\|\cdot\|$.

The reasons for using iterative methods instead of e. g. Gaussian elimination are the following: Stochastic model checkers that rely on an explicit state space representation use data structures for the probability matrix which are optimized for sparse matrices. The application of direct numerical solution methods destroys the sparseness of the probability matrix. Symbolic model checkers rely on an implicit state space representation and use iterative methods because they can exploit the compact symbolic representation more effectively than direct methods, which have to modify single elements of the matrix. Furthermore, Gaussian elimination is known to be not numerically robust, i. e., rounding errors can cumulate during the solution process, which is not the case for most iterative methods.

D. Symbolic Methods for PCTL Model Checking

Algorithms which rely on an explicit representation of the system are naturally restricted to quite a small number of states. A method to overcome this problem is the usage of symbolic data structures. Their advantage is that their size is not directly related to the size of the represented state space. For many practical examples, the size of the symbolic representation is much smaller than the explicit representation such that larger systems can be handled.

One of the most prominent symbolic data structures are binary decision diagrams (BDDs) [10]. Since the details of how the model checking algorithms can be modified to exploit the symbolic representation efficiently are only of little importance for the understanding of the effects of inexact arithmetic, we refer the reader to [11] and Parker's PhD thesis [9] about probabilistic model checking.

III. Motivation

It is well-known that the usage of inexact arithmetic for numerical computations can be problematic. For efficiency reasons and because it is common sense that the models are quite stable such that "nothing will go wrong", IEEE 754 floating-point arithmetic [1] is nevertheless used by (almost) *all* state-of-the-art tools for probabilistic model checking. We will show now that this assumption is not always justified.

Fig. 2 shows a DTMC. Let $0 < \gamma \leq 1/2$. We want to compute the probability with which we walk a path starting in the initial state s_1 of the DTMC that satisfies the PCTL path formula

$$c\,\mathcal{U}\,\mathcal{P}_{\leq 1/2}(a\,\mathcal{U}\,b)$$

with the meaning that c has to hold until we reach a state that satisfies $\mathcal{P}_{\leq 1/2}(a\,\mathcal{U}\,b)$. Since $\Pr(s_1, a\,\mathcal{U}\,b) = 1/2 + \gamma^3 > 1/2$

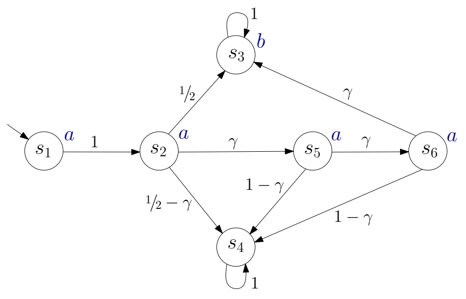

Fig. 2. A discrete-time Markov chain

```
probabilistic

const double gamma = 0.000001;

module sys
 s: [1..6] init 1;

 [] s=1 -> 1.0: (s'=2);
 [] s=2 -> 0.5: (s'=3) + gamma: (s'=5)
                    + (0.5-gamma): (s'=4);
 [] s=3 -> 1.0: (s'=3);
 [] s=4 -> 1.0: (s'=4);
 [] s=5 -> gamma: (s'=6) + (1-gamma): (s'=4);
 [] s=6 -> gamma: (s'=3) + (1-gamma): (s'=4);
endmodule

P=? [s=8 U (P<=0.5 [s=1 | s=2 | s=5 | s=6 U s=3])]
```

Fig. 3. PRISM input file for the DTMC shown in Fig. 2 (with $\gamma = 10^{-6}$)

```
STATES 6
TRANSITIONS 10
1 2 1.0                        #DECLARATION
2 3 0.5                        a b c
2 4 0.499999                   #END
2 5 0.000001                   1 a
3 3 1.0                        2 a
4 4 1.0                        3 b
5 4 0.999999                   5 a
5 6 0.000001                   6 a
6 3 0.000001
6 4 0.999999
```

Fig. 4. MRMC input files for the DTMC shown in Fig. 2 (with $\gamma = 10^{-6}$)

and no state is labeled with the property c, the formula above is satisfied in state s_1 with probability 0.

We took two state-of-the-art tools for probabilistic model checking, namely PRISM 3.1.1 [4] and MRMC 1.2.2 [5], and applied them to this DTMC. The input file for PRISM is given in Fig. 3, the MRMC input files in Fig. 4.

To our surprise, both tools returned the incorrect probability 1.0 for this DTMC with $\gamma = 10^{-6}$. Probabilities in the order of magnitude of 10^{-6} are not uncommon for real-world systems, e. g., when describing component failure probabilities.

The reason for this incorrect result is the floating-point arithmetic. It provides about 15 correct decimal digits. To represent $1/2 + \gamma^3 = 1/2 + 10^{-18}$, a precision of at least 18 decimal digits would be necessary to obtain a value that is strictly larger than $1/2$. Therefore $1/2 + 10^{-18}$ is rounded down to $1/2$. Now, the sub-formula $\mathcal{P}_{\leq 1/2}(a\,\mathcal{U}\,b)$ is satisfied in state s_1. This however implies that the probability to take a path from s_1 which satisfies $c\,\mathcal{U}\,\mathcal{P}_{\leq 1/2}(a\,\mathcal{U}\,b)$ is 1.

In the following we will investigate in more detail, where these inaccuracies are introduced during the model checking process, and present ideas how to avoid them and how to obtain reliable results in spite of inexact computations.

IV. Obtaining Reliable Results

We now turn our attention to the analysis of the problem described above. We will first point out where inaccuracy is introduced during the model checking process and how it can be avoided.

A. Sources of Inaccuracy

To identify at which points inaccuracy is introduced during the model checking process, we had a close look at the state-of-the-art model checkers PRISM [4], which supports explicit and symbolic model checking as well as a hybrid variant of these two approaches, and the explicit tool MRMC [5]. We have identified four major sources of inaccuracy. These are not restricted to a specific tool, but they are inherent to *all* state-of-the-art model checkers.

1) The floating-point arithmetic, which is used by all state-of-the-art model checkers for PCTL. The floating-point arithmetic is based on IEEE standard 754 [1] for 64 bit numbers. While the additions and multiplications are carried out with higher precision internally, the result of each arithmetic operation is rounded to fit into the 64-bit representation. About 15 decimal digits (51 binary digits) can be represented correctly. If the result is not representable as a floating-point number with that precision, the nearest representable number is chosen if it is unique. If the result lies exactly in the middle of two floating-point numbers, the one whose representation ends with "0" is chosen (round-to-nearest-even). We refer the reader to e. g. [12] for details on how the rounding for floating-point numbers works.
2) The termination criterion for solving the linear equation systems $Ax = b$ for the unbounded-until quantifier. There are mainly two kinds of criteria: The first one uses the difference between two successive approximations, i. e., for a given norm $\|\cdot\|$ on $\mathbb{R}^n$, the algorithm terminates if
$$\|x^{(k)} - x^{(k-1)}\| < \varepsilon$$
or if the relative difference is smaller than ε:
$$\frac{\|x^{(k)} - x^{(k-1)}\|}{\|x^{(k)}\|} < \varepsilon.$$
These criteria are supported by PRISM. The other type of termination condition uses the residual $Ax^{(k)} - b$. Then the iteration terminates if
$$\|Ax^{(k)} - b\| < \varepsilon.$$
3) For symbolic model checking, another reason for inaccuracy is located in the BDD package. In most packages like Cudd [13], which is used by PRISM, there is a constant $\delta > 0$ such that a new leaf with value v is only generated if there is no leaf with value v' and $|v - v'| \leq \delta$. The value of δ is chosen to be in the

order of the error by the floating-point arithmetic. Cudd uses $\delta = 10^{-12}$ as default.

4) For the continuous-time variant of DTMCs, continuous-time Markov chains (CTMCs), there is a further source of inaccuracies: To evaluate the time-bounded until-operator, uniformization is applied [14]. This reduces the computation of the probability, that in a state the formula $\phi_1 \mathcal{U}^{\leq t} \phi_2$ is satisfied, to the evaluation of an infinite sum. Since an exact evaluation is impossible, the sum is truncated after a finite number of steps. The size of the truncation error can be bounded using a theorem of Fox and Glynn [15].

We will now present a few ideas how these inaccuracies can be avoided or how correct answers can be obtained in spite of inexact computations.

B. Exact Arithmetic

By using exact arithmetic, we can eliminate the first and the third source of inaccuracy. The second source can only be eliminated by using a direct solution method like Gaussian elimination for the linear equation systems. As already discussed above, these direct methods are very badly suited for the solution of large sparse systems or when using a symbolic data representation. They destroy the sparseness of the matrix and require single entries in the matrix to be manipulated. Thereby the compact symbolic representation cannot be exploited and—making it still worse—the structure of the matrix gets lost such that the size of the sparse matrix representation and of the MTBDDs, resp., explodes.

In the setting of CTMCs, further inaccuracies remain: The evaluation of the time-bounded next-state quantifier $\mathcal{X}^{\leq t} \phi$ ("after one step, which has to occur within time t, ϕ has to hold") requires the computation of $e^{-\lambda t}$ [14] for some rational number $\lambda > 0$, since the transitions are governed by a negative exponential distribution. This value is irrational for almost all values of t. Consequently, this cannot be done exactly using rational arithmetic. Also the problem of computing an infinite sum for the time-bounded until operator cannot be solved using exact arithmetic.

We can conclude that the application of rational arithmetic cannot yield absolutely correct probabilities, but they can be approximated arbitrarily tightly. We have implemented a prototypic PCTL model checker which can switch from floating-point to exact arithmetic [16]. It uses the same algorithms as PRISM and with floating-point arithmetic, it produces results which are comparable with PRISM w. r. t. accuracy and runtimes. Our experiments have clearly shown that exact arithmetic is prohibitively expensive regarding computation time and memory consumption.

C. Interval Arithmetic

A different idea is to compute safe intervals for the probabilities of path formulae, i. e. intervals $I_\psi(s) = [l_\psi, u_\psi]$ such that $\Pr(s, \psi) \in I_\psi(s)$. Then a formula $\mathcal{P}_{\bowtie p}(\psi)$ holds for sure in state s if $\forall x \in I_\psi(s) : x \bowtie p$. Conversely, $\mathcal{P}_{\bowtie p}(\psi)$ is violated for sure if $\forall x \in I_\psi(s) : x \not\bowtie p$. In all other cases, nothing can be implied about the validity of the formula. Consequently, we have to cope with a three-valued interpretation of PCTL. This logic was introduced in another context in [17].

How can such safe intervals be derived? The error introduced by floating point arithmetic can easily be estimated using interval arithmetic, even for the evaluation of the exponential function. Also the uniformization in the setting of CTMCs is unproblematic, since the Fox-Glynn approximation provides us with an error bound.

The second source of inaccuracy (solution of linear equation systems) still remains. There are publications which investigate exactly this problem, see e. g. [18]. The proposed solutions are mostly based on estimating the approximation error from the residual, such that they are essentially independent of the solution method and the errors introduced by inexact arithmetic.

Although we have not yet implemented this approach, we expect that the memory consumption of the probability vectors will roughly double, since two numbers for the interval bounds have to be stored instead of one number. The runtime will be higher by a small factor, because we need to derive lower and upper bounds for the intervals, requiring two model checking runs per sub-formula.

D. Certificates for the Correctness of the Result

One possibility to check whether a result is reliable or not is to use algorithms which provide certificates for the correctness of the result. A certificate is an output of the tool besides the yes/no-answer which allows the user to check whether the answer is correct. We want to illustrate the concept of certificates with an example from a different domain:

Example 2 *Consider the linear programming problem $Ax = b$, $x \geq 0$ with $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$, and variables $x \in \mathbb{R}^n$. We have to decide whether this system is satisfiable [19].*

If it is satisfiable, an assignment for x, which satisfies all constraints, is a certificate. Otherwise, we can use Farkas' lemma [20]. It says that exactly one of the systems

$$Ax = b,\ x \geq 0 \qquad \text{and} \qquad y^T A \geq 0,\ y^T b < 0$$

is satisfiable. So, in case of unsatisfiability, an assignment for y for which $y^T A \geq 0$ and $y^T b < 0$ hold is a certificate.

For a property of the form $\mathcal{P}_{>p}(\phi_1 \mathcal{U} \phi_2)$ a certificate consists of a finite set Π of finite paths satisfying $\phi_1 \mathcal{U} \phi_2$ such that $\Pr_{s_0}(\Pi) > p$ [21], [22]. The certificate can be verified easily using exact arithmetic, since only multiplication and addition are necessary.

Example 3 *A possible certificate for $\mathcal{P}_{>0.5}(a \mathcal{U} b)$ in state s_1 of the DTMC from Fig. 2 is shown in Fig. 5.*

Unfortunately, for properties of the form $\mathcal{P}_{\leq p}(\phi_1 \mathcal{U} \phi_2)$ no certificates are known up to now. On reason for this is that a finite set of finite paths cannot testify that there are no further paths satisfying $\phi_1 \mathcal{U} \phi_2$ such that the probability mass p is exceeded.

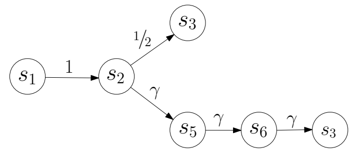

Fig. 5. Certificate showing that $\mathcal{P}_{>{}^1\!/_2}(a\,\mathcal{U}\,b)$ is satisfied in s_1 of Fig. 2

E. Increasing the Confidence into the Result

In our example from section III, if the user knew that $\mathcal{P}_{\leq 0.5}(a\,\mathcal{U}\,b)$ was satisfied in s_1, because the probability computed by the model checker was 0.5, it would be clear that the answer has to be taken with care, since an arbitrarily small change would flip the truth value.

For large systems with hundreds of thousands of states, it is not practical to provide the user with all probabilities. But instead a "degree of belief" can be introduced:

$$db_{\mathcal{P}_{\bowtie p}(\psi)} = \min\big\{ |\Pr(s,\psi) - p| \,\big|\, s \in S \big\}$$

The smaller db_ϕ is for a sub-formula ϕ of the formula under consideration, the more likely it is that inexactness influences the result of the model checking process. In our motivational example, we have $db_{\mathcal{P}_{\leq 0.5}(a\,\mathcal{U}\,b)} = 0$, which informs the user about an extremely unreliable intermediate result.

The introduction of the degree of belief does not remove any of the inaccuracies, but it is easy to integrate into existing solvers and it gives the user at least a hint about potential problems.

V. Conclusion

In this paper, we have shown that probabilistic model checking, which is performed with inexact computations by all state-of-the-art tools, can produce wrong results. We have seen a small Markov chain, for which at least two of the most popular model checking tools compute completely wrong probabilities.

We have analyzed at which points of the model checking process inaccuracies are introduced. We have then presented ideas how to obtain reliable results: (1) the usage of exact (rational) arithmetic does not solve all the problems and has costs which are too high for practical applications. (2) Interval arithmetic is a solution that can produce reliable results but there will be cases for which the model checker cannot give an answer. (3) Certificates, i. e. data which testifies that the answer of the tool is correct and which is easily checked by the user, can only be provided for formulae of the form $\mathcal{P}_{>p}(\phi_1\,\mathcal{U}\,\phi_2)$ where ϕ_1 and ϕ_2 do not contain probabilistic quantifiers and are thus restricted to a sub-logic of PCTL. (4) The introduction of a "degree of belief", which describes the risk that the truth value of a sub-formula flips in a state due to inexact computations, does not improve the results, but it can give the user at least a hint about potentially wrong results.

Future work will consist of elaborating and implementing the presented ideas. An experimental evaluation will have to show their practical feasibility.

Acknowledgments

We thank Holger Hermanns from the Saarland University for his helpful comments.

References

[1] IEEE Computer Society Standards Committee. Working group of the Microprocessor Standards Subcommittee and American National Standards Institute, *IEEE standard for binary floating-point arithmetic*, ser. ANSI/IEEE Standard 754-1985. Silver Spring, MD 20910, USA: IEEE Computer Society, 1985.

[2] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *1st Int. Colloquium on Theoretical Aspects of Computing (ICTAC)*, ser. LNCS, vol. 3407. Guiyang, China: Springer, Sept. 2004, pp. 280–294.

[3] E. Stark and G. Pemmasani, "Implementation of a compositional performance analysis algorithm for probabilistic I/O automata," in *7th Int. Workshop on Process Algebra and Performance Modelling (PAPM)*, Sept. 1999, pp. 3–24.

[4] M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM 2.0: A tool for probabilistic model checking." in *1st Int. Conf. on Quantitative Evaluation of Systems (QEST)*. Enschede, The Netherlands: IEEE Computer Society, 2004, pp. 322–323.

[5] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *2nd Int. Conf. on Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 2005, pp. 243–244.

[6] A. Bell and B. R. Haverkort, "Untold horrors about steady-state probabilities: What reward-based measures won't tell about the equilibrium distribution," in *Fourth European Performance Engineering Workshop on Formal Methods and Stochastic Models for Performance Evaluation, (EPEW)*, ser. LNCS, vol. 4748. Springer, 2007, pp. 2–17.

[7] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[8] F. Ciesinski and M. Größer, "On probabilistic computation tree logic," in *Validation of Stochastic Systems*, ser. LNCS, vol. 2925. Springer, 2004, pp. 147–188.

[9] D. Parker, "Implementation of symbolic model checking for probabilistic systems," PhD thesis, University of Birmingham, Great Britain, 2002.

[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[11] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan, "Symbolic model checking for probabilistic processes," in *24th Int. Colloqium on Automata, Languages and Programming (ICALP)*, ser. LNCS, vol. 1256. Springer, 1997, pp. 430–440.

[12] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comp. Surv.*, vol. 23, no. 1, Mar. 1991.

[13] F. Somenzi, *CUDD: CU Decision Diagram Package Release 2.4.1*. University of Colorado at Boulder, 2005.

[14] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 524–541, 2003.

[15] B. L. Fox and P. W. Glynn, "Computing Poisson probabilities," *Commun. ACM*, vol. 31, no. 4, pp. 440–445, 1988.

[16] R. Wimmer, A. Kortus, M. Herbstritt, and B. Becker, "Symbolic model checking for DTMCs with exact and inexact arithmetic," Reports of SFB/TR 14 AVACS 30, 2007, ISSN: 1860-9821, `http://www.avacs.org`.

[17] H. Fecher, M. Leucker, and V. Wolf, "Don't know in probabilistic systems," in *13th Int. SPIN Workshop on Model Checking Software*, ser. LNCS, vol. 3925. Vienna, Austria: Springer, 2006, pp. 71–88.

[18] S. Oishi and S. M. Rump, "Fast verification of solutions of matrix equations," *Numer. Math.*, vol. 90, no. 4, pp. 755–773, 2002.

[19] A. Schrijver, *Theory of linear and integer programming*, ser. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., 1986, A Wiley-Interscience Publication.

[20] J. Farkas, "Theorie der einfachen Ungleichungen," *Journal für reine und angewandte Mathematik*, vol. 124, 1902.

[21] T. Han and J.-P. Katoen, "Counterexamples in probabilistic model checking," in *Proc. of TACAS*, ser. LNCS, vol. 4424. Springer, 2007, pp. 60–75.

[22] ——, "Providing evidence of likely being on time: Counterexample generation for CTMC model checking," in *5th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, ser. LNCS, vol. 4762. Springer, 2007, pp. 331–346.