The Demand for Reliability in Probabilistic Verification*

Ralf Wimmer Alexander Kortus Marc Herbstritt Bernd Becker
Albert-Ludwigs-University Freiburg im Breisgau, Germany
{wimmer,kortus,herbstri,becker}@informatik.uni-freiburg.de

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

First we present an example, for which several state-of-the-art probabilistic model checking tools give completely wrong results due to inexact computations. This motivates the investigation at which points inaccuracies are introduced during the model checking process. We then give ideas how, in spite of these inaccuracies, reliable results can be obtained or at least the user be warned about potential problems: (1) to introduce a "degree of belief" for each sub-formula, (2) to use exact (rational) arithmetic, (3) to use interval arithmetic to obtain safe approximations of the actual probabilities, and (4) to provide certificates which testify that the result is correct.

1 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, probabilistic 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, first successful applications in industrial settings have been reported.

*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.

There are several academic tools available for probabilistic model checking. But 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. Also, to the best of our knowledge, we are not aware of papers that discuss the impact of using inexact floating-point arithmetic on the correctness of the result. 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.

The main topic of this paper is therefore to discuss the impact of using inexact arithmetic while model checking probabilistic models. We will demonstrate that even state-of-the-art tools have the problem that sometimes completely wrong results are produced. In Section 3 we will present an example for which two very popular model checkers compute the probability 1.0 instead of the correct result 0.0. These results clearly reveal the demand for *reliable results*, either by using exact or interval arithmetic or—if possible—by providing certificates testifying the correctness of the result.

This paper consists of the following parts: First, we review the basic definitions of discrete-time Markov chains (DTMCs), the logic PCTL, and the algorithm 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 investigation at which points of conventional model checkers inaccuracies are introduced. We will point out ideas how these inaccuracies can be avoided and how reliable results can be obtained in spite of inexact computations. The paper closes with a conclusion and an outlook to future work.

2 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.

2.1 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, $P : S \times S \to [0, 1]$ the matrix of transition probabilities, and $L : S \to 2^{AP}$ 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

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

We illustrate the main concepts in the following example:

Example 1 *In Figure 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.*

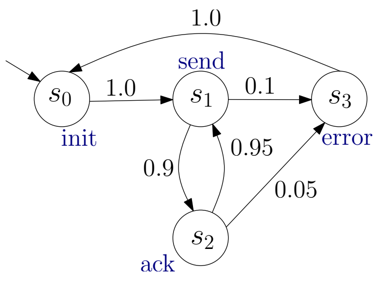

Figure 1: A discrete-time Markov chain

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 $\mathrm{Pr}_{s_0}(\pi) = 1.0 \cdot 0.9 \cdot 0.95 \cdot 0.9 \cdot 0.95 = 0.731025$.

2.2 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 [6].

In the following, we 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:

$$\Psi ::= \mathrm{X}\Phi \mid \Phi \,\mathrm{U}\, \Phi \mid \Phi \,\mathrm{U}^{\leq k}\, \Phi.$$

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:

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

2.3 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 at which points inaccuracies are introduced. For more details on PCTL model checking see e. g. [7, 8].

Model checking for PCTL is based on recursively traversing the syntax tree of the formula bottom-up and computing the set $\text{Sat}(\phi) = \{s \in S \mid s \models \phi\}$ for each state sub-formula ϕ. This can be done for state formulae 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):

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

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

2.3.1 Next Quantifier (X)

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

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

2.3.2 Bounded Until Quantifier ($\mathrm{U}^{\leq k}$)

We can characterize the bounded until operator with the following recursive equation system:

$$\Pr(s, \phi_1 \,\mathrm{U}^{\leq k}\, \phi_2) = \begin{cases} 1 & \text{if } s \in \text{Sat}(\phi_2) \\ 0 & \text{if } s \notin \text{Sat}(\phi_1) \text{ and } s \notin \text{Sat}(\phi_2) \\ 0 & \text{if } k = 0 \text{ and } s \notin \text{Sat}(\phi_2) \\ \sum_{s' \in S} P(s, s') \cdot \Pr(s', \phi_1 \,\mathrm{U}^{\leq k-1}\, \phi_2) & \text{otherwise.} \end{cases}$$

The intuition behind this recursive computation is the following: if s satisfies ϕ_2, all paths starting in s fulfill $\psi := \phi_1 \,\mathrm{U}^{\leq k}\, \phi_2$; 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.

2.3.3 Unbounded Until Quantifier (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 $\Pr(s, \phi_1 \,\mathrm{U}\, \phi_2) = 1$, if $s \in S^1$, and 0, if $s \in S^0$. A state s that is contained in the set $\mathrm{Sat}(\phi)$ of states that satisfy formula ϕ is denoted as ϕ-state. 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 as follows [7]:

$$\begin{aligned}
S_f &= \{s \in S \mid s \notin \mathrm{Sat}(\phi_1) \wedge s \notin \mathrm{Sat}(\phi_2)\} \\
S_i &= \{s \in S \mid s \in \mathrm{Sat}(\phi_1) \wedge s \notin \mathrm{Sat}(\phi_2)\} \\
S^0 &= S_f \cup \{s \in S_i \mid \text{there exists no path in } S_i \text{ from } s \text{ to any } s' \in \mathrm{Sat}(\phi_2)\} \\
S^1 &= \mathrm{Sat}(\phi_2) \cup \{s \in S_i \mid \text{there exists no path in } S_i \text{ from } s \text{ to any } s' \in S^0\} \\
S^? &= S \setminus (S^0 \cup S^1)
\end{aligned}$$

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

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

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 into account a bound on the number of steps.

Such linear equation systems are usually solved using iterative methods like Jacobi, Gauß-Seidel or over-relaxation methods. In general, starting with an initial estimation x^0 of the solution, these methods 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 that are optimized for sparse matrices. The application of direct numerical solution methods destroys the sparseness of the probability matrix. Symbolic model checkers, as described below, rely on an implicit state space representation and use iterative methods because they can exploit the compact symbolic representation in a better way than direct methods which have to modify single elements of the matrix. Furthermore, Gaussian elimination is known not to be numerically robust, i. e., numerical errors can cumulate and cause severe correctness problems.

2.4 Symbolic Methods for PCTL Model Checking

Algorithms that rely on an explicit representation of the state space 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) [9]. We assume some familiarity of the reader with BDDs. For further information see e. g. Wegener's monograph [10] on decision diagrams.

Since the details on how to modify model checking algorithms, such that the symbolic representation is exploited, are only of little importance for the understanding of the effects of inexact arithmetic, we refer the reader to Parker's thesis [8] about probabilistic model checking.

PRISM

```
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])]
```

MRMC

```
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
```

Figure 3: Input files for the DTMC shown in Fig. 2 (with $\gamma = 10^{-6}$)

3 PCTL Model Checking with Inexact Arithmetic

It is well-known that the usage of inexact arithmetic for numerical computations can pose severe problems. 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 ubiquitous in (almost) *all* state-of-the-art tools for probabilistic model checking. We show in the following 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 along a path starting the initial state s_1 of the DTMC that satisfies the formula

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

This formula means that c has to hold until we reach a state that satisfies $\mathcal{P}_{\leq 1/2}(a \,\mathrm{U}\, b)$. Since $\Pr(s_1, a \,\mathrm{U}\, b) = 1/2 + \gamma^3 > 1/2$ and no state fulfills property c, the formula above is satisfied in state s_1 with probability 0.

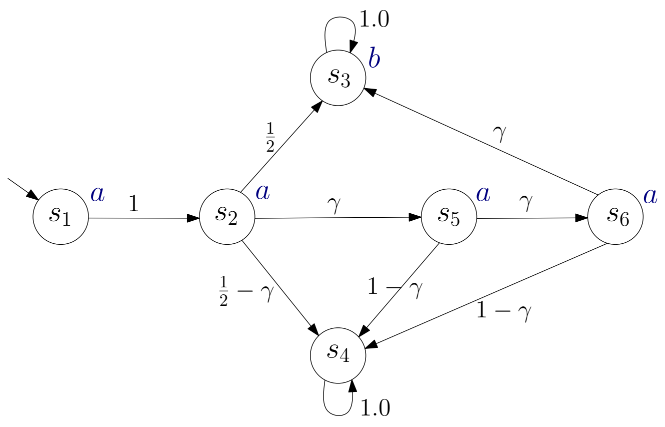

Figure 2: A discrete-time Markov chain

We evaluated 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 and the property from above. The corresponding input files for PRISM and MRMC are given in Fig. 3.

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

The reason for this incorrect result is due to the usage of 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 is required 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 $P_{\leq 1/2}(a \,\mathrm{U}\, b)$ is satisfied in state s_1. This, however, implies that the probability to take a path from s_1 that satisfies $c \,\mathrm{U}\, \mathcal{P}_{\leq 1/2}(a \,\mathrm{U}\, b)$ is exactly 1.

A further problem that does not occur in this example is that the structure of the system may be changed when transition probabilities are so small that they are mapped to 0 when using inexact arithmetic. For more details on this issue, we kindly refer the reader to our technical report [11].

In the following we investigate in more detail where these rounding problems are introduced and discuss several concepts how reliability could be increased or even ensured.

4 Fighting the Problem of Inaccuracy

In this section we first investigate at which points inaccuracy is introduced during the model checking process. Then, we present several ideas how to obtain reliable results in spite of these inaccuracies.

4.1 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 probabilistic model checkers.

A. 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, then the nearest representable number is chosen if it is unique. If the result lies exactly inbetween 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 number works.

B. For symbolic model checking, another reason for inaccuracy is located in the BDD package. In popoular 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.

C. The termination criterion for solving the linear equation systems for the unbounded-until quantifier. For the Jacobi method, for example, PRISM supports two termination criteria: The iteration terminates either if $\|x^{(k)} - x^{(k-1)}\|_\infty < \varepsilon$ for a given constant $\varepsilon > 0$ or if $\frac{\|x^{(k)} - x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty} < \varepsilon$. PRISM uses the second criterion and $\varepsilon = 10^{-6}$ as default. MRMC uses the Gauss-Seidel method to solve linear equation systems. The termination criterion is similar to the one of PRISM.

D. For model checking time-bounded until operator on the continuous-time variant of DTMCs, continuous-time Markov chains (CTMCs), uniformization is applied [14]. The computation of the probability that a path formula $\phi_1 \,\mathrm{U}^{\leq t}\, \phi_2$ holds can then be reduced to the evaluation of an infinite sum. Since an exact evaluation is impossible, an error is introduced by the truncation of the sequence. The size of the error can be estimated using a theorem by Fox and Glynn [15].

4.2 Degree of Belief

An idea to increase the trust in the results, which can easily be integrated into current tools, is the computation of a "degree of belief", i. e., a measure that gives the user at least a hint about the risk that the truth value of a sub-formula flips due to inexact computations. We define the measure as follows:

$$db_{\mathcal{P}_{\bowtie p}(\psi)} := \min\bigl\{ |\Pr(s,\psi) - p| \bigm| s \in S \bigr\}.$$

If db_ϕ is close to 0 for some sub-formula ϕ, the result returned by the model checker has to be taken with care, since already small errors may then change the probability of the outermost formula arbitrarily. In our motivational example from section 3, we have $db_{\mathcal{P}_{\leq 0.5}(a \,\mathrm{U}\, b)} = 0$, warning the user about potentially wrong results.

4.3 Exact Arithmetic

By using exact arithmetic, we can eliminate the sources A and B. Source C can only be eliminated by using a direct solution method like Gaussian elimination for linear equation systems. These direct methods, however, 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 sizes of the sparse matrix representation and of the MTBDDs, resp., explode. Source D, which occurs only in the context of CTMCs, also cannot be eliminated by using exact arithmetic. But here, the Fox-Glynn approximation provides us at least with an estimation of the maximal error introduced by truncating the infinite sum.

Another drawback of exact arithmetic are its costs: As our experiments with an exact PCTL model checker have shown [11], the usage of rational arithmetic is very time and memory consuming, and hence is not a viable way for the verification of large models.

4.4 Interval Arithmetic

Another way to go is to use interval arithmetic for the computations. That means that we compute an interval $[l, u]$ which contains the exact probability with which a path formula holds in a state. A formula $\mathcal{P}_{\bowtie p}(\psi)$ is then satisfied for sure, if $\forall x \in [l, u] : x \bowtie p$. Conversely, the formula is violated for sure if $\forall x \in [l, u] : x \not\bowtie p$. Otherwise nothing can be said about the validity of the formula. We have therefore to cope with a three-valued logic as it was introduced, e. g., in [16].

The usage of interval arithmetic can eliminate the sources A, B, and D of inaccuracy. The elimination of source C is only possible if a direct solution method is used or if the iterative method provides us with a bound for the error.

In opposite to exact arithmetic, interval arithmetic is not prohibitively more expensive than standard floating-point arithmetic. Since two numbers have to be stored instead of one, the memory consumption of the probability vectors will roughly double. The runtime will be higher by a small factor since we have to derive lower and upper bounds for the probability requiring two model checking runs per sub-formula.

4.5 Certificates for the Correctness of the Result

A certificate is an output of a model checker of a solver besides the yes/no answer which makes it easy to check that the decision of the tool is correct.

Example 2 *Consider the linear programming problem [17], where one has to decide whether a system of the form $Ax = b$, $x \geq 0$ is satisfiable for a matrix $A \in \mathbb{R}^{n\times m}$, a vector $b \in \mathbb{R}^m$, and variables $x \in \mathbb{R}^n$.*

If the system is satisfiable, the solver can provide a certificate by returning a satisfying variable assignment for x. The user can then easily check if it indeed satisfies all constraints.

If the system is unsatisfiable, Farkas' lemma [18] can be used to obtain a certificate. It says that exactly one of the systems

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

is satisfiable. Hence an assignment for y such that $y^T A \geq 0,\ y^T b < 0$ is a certificate in case of unsatisfiability.

Han et al. [19] provide a method to compute certificates for PCTL formulae of the form $\mathcal{P}_{>p}(a \,\mathrm{U}\, b)$ (and counterexamples for $\mathcal{P}_{\leq p}(a \,\mathrm{U}\, b)$, resp.) where a and b are atomic propositions. In this case, a certificate consists of a finite set of finite paths which satisfy the path formula $a \,\mathrm{U}\, b$ and whose probability mass exceeds the bound p. If such a certificate is given, it can be verified easily using exact arithmetic since only multiplication and addition are necessary to compute the probability.

Example 3 *Let us consider the motivational example from section 3 again. A possible certificate showing that the property $\mathcal{P}_{>0.5}(a \,\mathrm{U}\, b)$ is fulfilled in state s_1 of the DTMC of Fig. 2 is shown in Fig. 4. It consists of two paths: $s_1 s_2 s_3$ and $s_1 s_2 s_5 s_6 s_3$. Their probability is $1 \cdot {}^1\!/_2 + 1 \cdot \gamma \cdot \gamma \cdot \gamma > {}^1\!/_2$.*

To the best of our knowledge, up to now, there is no efficient method to generate certificates for properties of the form $\mathcal{P}_{\leq p}(a \,\mathrm{U}\, b)$ and for properties with nested PCTL subformulae. The approach that comes closest to general certificates for PCTL is described in a paper by Daws [2]. There, regular expressions are computed for DTMCs that describe the probability with which a path formula holds. Since there are no numerical computations involved in the construction of the regular expressions and since they can be evaluated using only multiplications and additions, the regular expressions can be seen as a kind of compact representation of the paths satisfying a path formula. But since the computation of the regular expressions is expensive, we do not expect the approach to scale beyond small-sized systems. Furthermore, it does not support nested PCTL sub-formulae, which are — as we have seen in section 3 — a major source of problems.

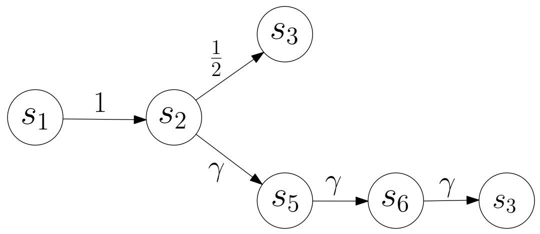

Figure 4: Certificate showing that $P_{>0.5}(a \,\mathrm{U}\, b)$ is satisfied in s_1 of Fig. 2

5 Conclusion

In this work we have discussed the impact of inexact numerical computations on the correctness of the results in the context of probabilistic model checking. We have presented an example for which several state-of-the-art probabilistic model checkers return completely wrong results due to inexact numerical computations. Motivated by this artefact, we have investigated at which points of the model checking process these inaccuracies are introduced.

We have discussed several approaches which try to circumvent the inexactness: (1) the introduction of a "degree of belief" for each sub-formula, the usage of (2) exact or (3) interval arithmetic for the numerical computations, and (4) the provision of certificates which testify that the result is correct.

Our considerations reveal the need for further investigations how to provide certificates for the correctness of the answer since not all problems can be solved using exact or interval arithmetic.

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, American National Standards Institute: IEEE standard for binary floating-point arithmetic. ANSI/IEEE Std 754-1985. IEEE Computer Society Press (1985)
2. Daws, C.: Symbolic and parametric model checking of discrete-time markov chains. In Liu, Z., Araki, K., eds.: 1st Int. Colloquium on Theoretical Aspects of Computing (ICTAC). Vol. 3407 of LNCS., Guiyang, China (2004) 280–294
3. Stark, E., Pemmasani, G.: Implementation of a compositional performance analysis algorithm for probabilistic I/O automata. In: 7th Int. Workshop on Process Algebra and Performance Modelling (PAPM). (1999) 3–24
4. Kwiatkowska, M.Z., Norman, G., Parker, D.: 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) 322–323
5. Katoen, J.P., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: 2nd Int. Conf. on Quantitative Evaluation of Systems (QEST), IEEE CS (2005) 243–244
6. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5) (1994) 512–535
7. Ciesinski, F., Größer, M.: On probabilistic computation tree logic. In: Validation of Stochastic Systems. Vol. 2925 of LNCS., Springer (2004) 147–188
8. Parker, D.: Implementation of Symbolic Model Checking for Probabilistic Systems. PhD thesis, University of Birmingham, Great Britain (2002)
9. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **35**(8) (1986) 677–691
10. Wegener, I.: Branching Programs and Binary Decision Diagrams – Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications. (2000)
11. Wimmer, R., Kortus, A., Herbstritt, M., Becker, B.: 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`.
12. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys **23**(1) (1991)
13. Somenzi, F.: CUDD: CU Decision Diagram Package Release 2.4.1. University of Colorado at Boulder (2005)
14. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time markov chains. IEEE Trans. Software Eng. **29**(6) (2003) 524–541
15. Fox, B.L., Glynn, P.W.: Computing poisson probabilities. Communications of the ACM **31**(4) (1988) 440–445
16. Fecher, H., Leucker, M., Wolf, V.: Don't know in probabilistic systems. In: 13th Int. SPIN Workshop on Model Checking Software. Vol. 3925 of LNCS., Springer (2006) 71–88
17. Schrijver, A.: Theory of linear and integer programming. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., Chichester (1986) A Wiley-Interscience Publication.
18. Farkas, J.: Theorie der einfachen Ungleichungen. Journal für reine und angewandte Mathematik **124** (1902)
19. Han, T., Katoen, J.P.: Counterexamples in probabilistic model checking. In: 13th Int. Conf. on Tools and Algorithms for Construction and Analysis of Systems (TACAS). Vol. 4424 of LNCS., Springer (2007) 60–75