Exact Quantitative Probabilistic Model Checking Through Rational Search

Matthew S. Bauer*, Umang Mathur*, Rohit Chadha†, A. Prasad Sistla‡ and Mahesh Viswanathan*
*University of Illinois at Urbana-Champaign, †University of Missouri,‡University of Illinois at Chicago

***Abstract*—Model checking of systems formalized using probabilistic models such as discrete time Markov chains (DTMCs) and Markov decision processes (MDPs) can be reduced to computing constrained reachability properties. Linear programming methods to compute reachability probabilities for DTMCs and MDPs do not scale to large models. Thus, model checking tools often employ iterative methods to approximate reachability probabilities. These approximations can be far from the actual probabilities, leading to inaccurate model checking results. In this article, we present a new algorithm and its implementation that improves approximate results obtained by scalable techniques like value iteration to compute exact reachability probabilities.**

I. Introduction

Probabilistic models such as discrete time Markov chains (DTMCs) and Markov decision processes (MDPs) are often used to describe systems in many application areas such as distributed systems [16], [35], hardware communication protocols [17], reliability engineering in circuits [11], [22], [32], [33], dynamic power management [10], [34], networking [28], [29] and security [13]. Probabilistic transitions in these models are used to capture random faults, uncertainty of environment and explicit randomization used in algorithms. The key verification tasks for such systems are often accomplished through Probabilistic Computation Tree Logic (PCTL) model checking [36]. The logic PCTL extends the temporal logic CTL with operators that provide the ability to reason quantitatively. For example, given $\bowtie \in \{\leq, <, \geq, >\}$, the formula $\mathcal{P}_{\bowtie p}[\psi]$ expresses the property that the measure of computation paths satisfying ψ is $\bowtie p$. PCTL model checking proceeds by recursively computing the set of states that satisfy subformulas of a given formula. Each recursive step, in turn, reduces to *constrained quantitative reachability*, wherein, given a DTMC/MDP, a set of good states G and a target set of states T, the goal is to compute the measure of the paths that reach T while remaining in G. If the model is decorated with *costs* or *rewards*, one may also be interested in computing the expected cost/reward of reaching T. It is well known that the constrained quantitative reachability problem for DTMCs and MDPs can be solved in polynomial time by reducing to linear programming [8], [36].

Despite its low asymptotic complexity, linear programming, unfortunately, doesn't scale well to large models and is rarely used in practice to solve the quantitative reachability problem. Instead, probabilistic model checkers [14], [15], [21], [24], [25], [30], typically compute *approximations* to the exact reachability probabilities through an iterative process. The results computed by these tools can be incorrect, primarily due to two sources of imprecision. The first is the use of finite precision arithmetic and floating point numbers to carry out calculations. The second is the use of approximate techniques like *value iteration*, where the exact reachability probabilities may only be approached in the limit. It is common practice for model checking tools to terminate value iteration in a finite number of steps, based on several different criteria, such as, when the change in the computed reachability probability between successive iterations is "small". This approximation step may lead to unsound results, for example, in systems where high magnitude changes in value iteration are preceded by periods of stability that cause iteration to terminate prematurely. Inaccuracies in model checking can get further compounded by the presence of nested probability operators in PCTL formulas when the sets of good states G and target states T are not correctly computed in the recursive step (see Example 3 on in Section III).

Contributions

In this article, we present a new algorithm and its implementation that *sharpens* approximate solutions computed by value iteration, to obtain the *exact* constrained reachability probability, allowing one to obtain accurate and reliable model checking results. The starting point of our approach is the observation that when transition probabilities in the model are rational numbers, the exact solution is also a rational number of polynomially many bits. The second ingredient in our technique is an algorithm due to Kwek and Mehlhorn [26], which, given a "close enough" approximation to a rational number, finds the rational number efficiently. Our algorithm works roughly as follows. We use value iteration to compute an approximate solution and then apply the Kwek-Mehlhorn algorithm to find a close candidate rational solution. Since the approximate solution we start with is of unknown quality, the candidate rational solution obtained may not be the exact answer. Therefore, we check if the candidate is the unique solution to the linear program that describes the system. This allows one to confirm the correctness of the candidate rational solution. If it is not correct, the process is repeated, starting with an approximate solution of improved precision. Precise details of the algorithm are given in Section IV.

We gratefully acknowledge the support of the following grants— Matthew S. Bauer was partially supported by NSF CNS-1314485; Umang Mathur was partially supported by NSF CCF-1422798; Rohit Chadha was partially supported by NSF CNS-1314338 and NSF CNS-1553548; A. Prasad Sistla was partially supported by CNS-1314485, CCF-1319754 and CCF-1564296; and Mahesh Viswanathan was partially supported by NSF CNS-1329991.

We have implemented this approach as an extension of the PRISM model checker, called RATIONALSEARCH. Our tool computes exact constrained reachability probabilities and exact expected rewards for model checking DTMCs and MDPs against PCTL specifications. Evaluation of our implementation against a large set of examples from the PRISM benchmark suite [6] and case studies [7] shows that our technique can be applied to a wide array of examples. In many cases, our tool is orders of magnitude faster than the exact model checking engines implemented in state-of-the-art tools like PRISM [30] and STORM [15].

Related Work

The work closest in spirit to ours is [19], which presents an approach to obtain exact solutions for reachability properties for MDPs and discounted MDPs. The basic idea there is to interpret the scheduler obtained for an approximate solution, as a *basis* for the linear program corresponding to the verification question. By examining the optimality of the solution associated with this basis, the exact solution can be obtained by improving the scheduler using the Simplex algorithm. This is significantly different from our approach. In particular, for DTMCs (where there is no scheduler), the approach of [19] reduces to solving a linear program, which is known to be not scalable. Since the implementation from [19] is not available, we could not experimentally compare with this approach.

To overcome the convergence problems with value iteration, techniques like *interval iteration* [9], [20], [38], which utilize two simultaneous value iteration procedures converging to the exact probabilities values from above and below, have been proposed. This allows one to bound the error produced by approximation techniques. Additionally, several tools [15], [30] implement exact quantitative model checking as an extension of *parametric model checking*, which synthesizes symbolic algebraic expressions over parameters of the model, representing quantitative properties of a system. These expressions can be evaluated under a concrete instantiation of the parameters to produce exact solutions.

II. BACKGROUND

A common technique in the analysis of systems is to model them as *state transitions systems* where states describe information about the system at a point in time and transitions describe how the system evolves from one state to another. When this evolution is governed by random phenomena, such state transition systems can then be enriched to capture probabilistic behavior. The resulting model is known as a DTMC, in which every state is mapped to a distribution over the successor states. MDPs generalize DTMCs, in that, the distribution over the successor states is non-deterministically chosen. We next formalize DTMCs and MDPs.

Discrete time Markov chains (DTMCs)

A DTMC is a tuple $\mathcal{M} = (Z, \Delta, \mathbf{C}, L)$ where Z is a set of states, $\Delta : Z \to \mathsf{Dist}(Z)$ is the *probabilistic transition function* that maps every state to a probability distribution over Z, $\mathbf{C} : Z \times Z \mapsto \mathbb{Q}^{\geq 0}$ is a cost (or reward) structure and $L : Z \to 2^{\mathsf{AP}}$ is a labeling function that maps states to subsets of AP, the set of atomic propositions. We will restrict our attention to DTMCs with a finite number of states. For each $z \in Z$, $\Delta(z)$ defines a discrete probability distribution over Z, that is, $\Delta(z)(z') \geq 0$ for all $z' \in Z$, and $\sum_{z' \in Z} \Delta(z)(z') = 1$. We will henceforth denote $\Delta(z)(z')$ by $\Delta(z, z')$. A path ρ of $\mathcal{M}$ is a sequence of states $z_0 \to z_1 \to \cdots$ such that $\Delta(z_i, z_{i+1}) > 0$. We write $\rho(i)$ to denote the i^{th} state z_i in ρ. We denote the set of all infinite paths of $\mathcal{M}$ by $\mathsf{Paths}(\mathcal{M})$ and the set of all infinite paths of $\mathcal{M}$ starting from state z by $\mathsf{Paths}_z(\mathcal{M})$. For a finite path $\rho_{\text{fin}} = z_0 \to \cdots \to z_m$ we associate a probability measure $\mathsf{prob}(\rho_{\text{fin}}) = \prod_{i=0}^{m-1} \Delta(z_i, z_{i+1})$. The cylinder set of ρ_{fin} is $\mathsf{Cyl}(\rho_{\text{fin}}) = \{\rho \in \mathsf{Paths}(\mathcal{M}) \mid \rho_{\text{fin}} \text{ is a prefix of } \rho\}$ and its associated probability measure is $\mathsf{prob}(\mathsf{Cyl}(\rho_{\text{fin}})) = \mathsf{prob}(\rho_{\text{fin}})$, which can be extended to a unique probability measure over the smallest σ-algebra containing all cylinder sets. The cost associated with ρ_{fin} is $\mathsf{cost}(\rho_{\text{fin}}) = \sum_{i=0}^{m-1} \mathbf{C}(z_i, z_{i+1})$. Let $F \subseteq Z$. For a path $\rho \in \mathsf{Paths}(\mathcal{M})$, the cost of reaching F, denoted $\mathsf{cost}(F)(\rho)$, is the cost of the shortest prefix of ρ that reaches F, and is ∞ if no such prefix exists. The expected cost incurred for reaching F starting from z is given by $\mathsf{E}[\mathsf{cost}_z(F)] = \sum_{\rho \in \mathsf{Paths}_z(\mathcal{M})} \mathsf{prob}(\rho) \cdot \mathsf{cost}(F)(\rho)$.

Example 1: Consider an embedded control system [27] comprised of an input processor, a main processor, an output processor and a bus. In each cycle of the system, the input processor collects data from a set of n sensors $S_1, S_2, \ldots, S_n$. The main processor polls the input processor and passes instructions to the output processor controlling a set of m actuators $A_1, A_2, \ldots A_m$. Communication between processors occurs over the bus. The system is designed to tolerate failures in a limited number of components. If the input processor reports that the number of sensor failures exceeds some threshold MAX_FAILURES, then the main processor shuts the system down. Otherwise, it activates the actuators, which again, are prone to failure. When the probabilities, with which each of these components fail, are known, one can model the system's reliability using a DTMC. In Figure 1, we give a DTMC that models a single cycle of such a system with $n = 2$ sensors and $m = 1$ actuator. For simplicity, we assume that each sensor fails with probability E_s and each actuator fails with probability E_a. States of the model are labeled with $e_1^s, ..., e_n^s \in \{0, 1\}$ and $e_1^a, ..., e_m^a \in \{0, 1\}$, where $e_i^s = 1$ denotes the failure of sensor S_i and $e_i^a = 1$ denotes the failure of actuator A_i. In Figure 1, we omit labels if they are not relevant in a particular state.

Markov decision processes (MDPs)

An MDP is a tuple $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$ where Z is a finite set of states, Act is a set of actions, $\Delta : Z \times \mathsf{Act} \hookrightarrow \mathsf{Dist}(Z)$ is the *probabilistic transition function* that maps pairs of states and actions to probability distributions over Z, $\mathbf{C} : Z \times \mathsf{Act} \times Z \to \mathbb{Q}^{\geq 0}$ is a cost (or reward) structure and $L : Z \to 2^{\mathsf{AP}}$ is a labeling function. The set $\mathsf{enabled}(z) =$

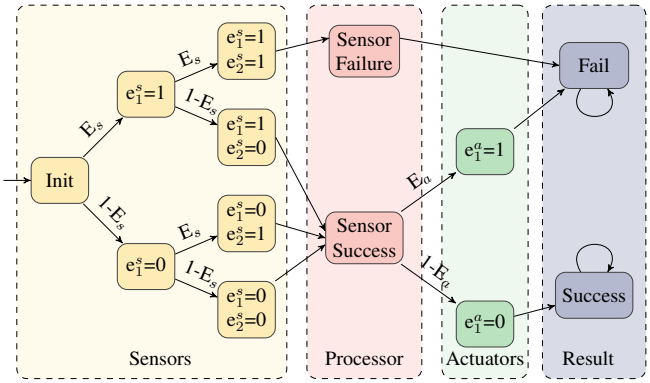

Fig. 1. Markov chain for a simple embedded control system with two sensors and one actuator tolerating a single sensor fault.

$\{\alpha \in \mathsf{Act} \mid \Delta(z,\alpha) \text{ is defined}\}$ of actions enabled from each state z is assumed to be non-empty for every $z \in Z$. An MDP, therefore, differs from a DTMC, in that, at each state z, there is a choice among several possible distributions. The choice of which distribution to *trigger* is resolved by a *scheduler* (or an attacker). Informally, an MDP $\mathcal{M}$ evolves as follows. It starts from some state $z_0 \in Z$. After i execution steps, if $\mathcal{M}$ is in state z, the scheduler chooses an action $\alpha \in \mathsf{enabled}(z)$, which then defines a unique probability distribution μ given by $\Delta(z,\alpha)$. The process then moves to state z' in step $(i+1)$ with probability $\Delta(z,\alpha)(z')$. We will write $\Delta(z,\alpha,z')$ to denote $\Delta(z,\alpha)(z')$ when $\alpha \in \mathsf{enabled}(z)$. A path ρ of an MDP $\mathcal{M}$ is a sequence $z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} \cdots$ such that for each $i \geq 0$, $\alpha_{i+1} \in \mathsf{enabled}(z_i)$ and $\Delta(z_i,\alpha_{i+1},z_{i+1}) > 0$.

Formally, a scheduler is a function $\mathfrak{S} : Z^+ \to \mathsf{Act}$ such that for every finite sequence $\tau = z_0 z_1 \ldots z_k \in Z^+$, we have $\mathfrak{S}(\tau) \in \mathsf{enabled}(z_k)$. A path $z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} \cdots$ is a $\mathfrak{S}$-path if $\mathfrak{S}(z_0 z_1 \ldots z_i) = \alpha_{i+1}$ for all $i \geq 0$. We will write $\mathsf{Paths}(\mathcal{M})$ for the set of infinite paths, $\mathsf{Paths}_z(\mathcal{M})$ for the set of infinite paths starting from z, $\mathsf{Paths}^{\mathfrak{S}}(\mathcal{M})$ for the set of infinite $\mathfrak{S}$-paths, and $\mathsf{Paths}^{\mathfrak{S}}_z(\mathcal{M})$ for the set of infinite $\mathfrak{S}$-paths starting from z. The set of all schedulers will be denoted by $\mathcal{S}$. A scheduler $\mathfrak{S} \in \mathcal{S}$ for MDP $\mathcal{M}$ induces a (potentially infinite) DTMC $\mathcal{M}^{\mathfrak{S}}$ such that $\mathsf{Paths}(\mathcal{M}^{\mathfrak{S}}) = \mathsf{Paths}^{\mathfrak{S}}(\mathcal{M})$. The definitions for the measure and cost associated with paths can then be naturally lifted from DTMCs. Interested readers should refer to standard texts such as [8], [37] for more details.

Probabilistic computation tree logic (PCTL)

Properties of DTMCs and MDPs can be expressed in the logic PCTL, which extends the temporal logic CTL with the ability to reason quantitatively. Let $a \in \mathsf{AP}$ be an atomic proposition, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0,1]$, $c \in \mathbb{Q}^{\geq 0}$ and $k \in \mathbb{N}$. Below, we begin by defining PCTL for DTMCs and then give the extension to MDPs.

Definition 1: The syntax of PCTL is

$$\phi ::= \mathsf{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{E}_{\bowtie c}[\phi]$$

where

$$\psi ::= \mathcal{X}\phi \mid \phi \mathcal{U} \phi$$

In Definition 1, ϕ is a state formula used to describe properties of states and ψ are path formulas used to model properties of paths. We now formalize the semantics of PCTL.

Definition 2: Let $\mathcal{M} = (Z, \Delta, \mathbf{C}, L)$ be a DTMC, ϕ, ϕ_1, ϕ_2 be state formulas and ψ be a path formula. The satisfaction relation $\models$ for PCTL state formulae is defined inductively as

$$\begin{array}{ll} z \models \mathsf{true} & \text{for all } z \in Z \\ z \models a & \Leftrightarrow a \in L(z) \\ z \models \neg\phi & \Leftrightarrow z \not\models \phi \\ z \models \phi_1 \wedge \phi_2 & \Leftrightarrow z \models \phi_1 \text{ and } z \models \phi_2 \\ z \models \mathcal{P}_{\bowtie p}[\psi] & \Leftrightarrow p_z(\psi) \bowtie p \\ z \models \mathcal{E}_{\bowtie c}[\phi] & \Leftrightarrow e_z(\phi) \bowtie c \end{array}$$

where $p_z(\psi) = \mathsf{prob}(\{\rho \in \mathsf{Paths}_z(\mathcal{M}) \mid \rho \models \psi\})$, $e_z(\phi) = \mathsf{E}[\mathsf{cost}_z(Z_\phi)]$ with $Z_\phi = \{z' \in Z | z' \models \phi\}$, and the satisfaction relation for paths and path formulae is defined inductively as

$$\begin{array}{ll} \rho \models \mathcal{X}\phi & \Leftrightarrow \rho(1) \models \phi \\ \rho \models \phi_1 \mathcal{U} \phi_2 & \Leftrightarrow \exists i \geq 0 : (\rho(i) \models \phi_2 \;\&\; \forall j < i : \rho(j) \models \phi_1) \end{array}$$

When the underlying model $\mathcal{M}$ is an MDP, the semantics of PCTL formulae stay the same, except for the semantics of $\mathcal{P}_{\bowtie p}[\psi]$ and $\mathcal{E}_{\bowtie c}[\phi]$, which now require a quantification over all schedulers. Let $p^{\mathfrak{S}}_z(\psi) = \mathsf{prob}(\{\rho \in \mathsf{Paths}^{\mathfrak{S}}_z(\mathcal{M}) | \rho \models \psi\})$. One can analogously define $e^{\mathfrak{S}}_z(\phi)$ for a scheduler $\mathfrak{S}$.

Definition 3: Let $\mathcal{M}$ be an MDP, ϕ be a state formula and ψ be a path formula. The satisfaction relation $\models$ for PCTL state formulae is defined identically to Definition 2, with the exception of the following cases.

$$\begin{array}{ll} z \models \mathcal{P}_{\bowtie p}[\psi] & \Leftrightarrow \forall \mathfrak{S} \in \mathcal{S},\, p^{\mathfrak{S}}_z(\psi) \bowtie p \\ z \models \mathcal{E}_{\bowtie c}[\phi] & \Leftrightarrow \forall \mathfrak{S} \in \mathcal{S},\, e^{\mathfrak{S}}_z(\phi) \bowtie c \end{array}$$

For a path formula ψ (resp. state formula ϕ), we write $\mathcal{P}_{=?}[\psi]$ (resp. $\mathcal{E}_{=?}[\phi]$) to represent the solution vector V, given by $\mathsf{V}(z) = p_z(\psi)$ (resp. $e_z(\phi)$) for all $z \in Z$. Strictly speaking, $\mathcal{P}_{=?}[\cdot]$ and $\mathcal{E}_{=?}[\cdot]$ are not part of PCTL syntax. However, we henceforth extend the PCTL syntax to allow $\mathcal{P}_{=?}[\cdot]$ and $\mathcal{E}_{=?}[\cdot]$ as the outermost operator.

Example 2: Consider the DTMC modeling an embedded control system from Example 1. One can describe many important properties of this model using PCTL:

1) The probability of success.
$\mathcal{P}_{=?}$ [$\mathsf{true}\, \mathcal{U}$ "Sucess"]

2) The probability that there are no sensor failures.
$\mathcal{P}_{=?}$ [$\mathsf{true}\, \mathcal{U}\, (e^s_1 + \ldots + e^s_n = 0)$]

3) The probability that actuator A_1 does not fail given that sensor S_1 fails with probability $\bowtie 1/2$.
$\mathcal{P}_{=?}$ [$\mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\, \mathcal{U}\, (e^s_1{=}1)]\, \mathcal{U}\, \mathcal{P}_{\leq 0}[\mathsf{true}\, \mathcal{U}\, (e^a_1{=}1)]$]

PCTL model checking

Similar to the model checking algorithm for CTL, the PCTL model checking algorithm recursively computes the set of states satisfying a state sub-formula. We will begin by restricting our attention to DTMCs.

Let ϕ, ϕ' be state formulas. To compute $\mathcal{P}_{=?}[\phi\, \mathcal{U}\, \phi']$, one recursively computes the set of states Z_ϕ and $Z_{\phi'}$ satisfying

ϕ and ϕ' respectively. These can be used to derive, for every $z \in Z$, the quantity $p_z(\phi \,\mathcal{U}\, \phi')$ representing the probability of reaching the set $Z_{\phi'}$ while remaining in the set Z_ϕ, starting from the state z.

Now, $p_z(\phi \,\mathcal{U}\, \phi')$ is the unique solution to the following linear program:

$$p_z(\phi \,\mathcal{U}\, \phi') = \begin{cases} 0 & \text{if } z \in \mathsf{Prob0} \\ 1 & \text{if } z \in \mathsf{Prob1} \\ \sum\limits_{z' \in Z} \Delta(z, z') \cdot p_{z'}(\phi \,\mathcal{U}\, \phi') & \text{otherwise} \end{cases} \quad (1)$$

where $\mathsf{Prob0}, \mathsf{Prob1}$ can be determined via a pre-computation step that analyzes the underlying graph of the DTMC. For computing $\mathcal{P}_{\bowtie p}[\phi \;\mathcal{U}\; \phi']$, one computes $\mathcal{P}_{=?}[\phi \;\mathcal{U}\; \phi']$ and compares $p_z(z \,\mathcal{U}\, z') \bowtie p$ for every $z \in Z$. The computation for $\neg\phi$, $\phi \wedge \phi'$, $\mathcal{E}_{=?}[\phi]$ and $\mathcal{P}_{\bowtie c}[\mathcal{X}\phi]$ is similar.

When the underlying model is an MDP, the computation for $\mathcal{P}_{\bowtie p}[\phi \;\mathcal{U}\; \phi']$ reduces to solving the following linear optimization problem when $\bowtie \in \{<, \leq\}$

$$\begin{aligned} &\min \sum_{z \in Z} p_z(\phi \,\mathcal{U}\, \phi') \quad \text{subject to} \\ &p_z(\phi \,\mathcal{U}\, \phi') = 0 && \text{if } z \in \mathsf{Prob0} \\ &p_z(\phi \,\mathcal{U}\, \phi') = 1 && \text{if } z \in \mathsf{Prob1} \quad (2) \\ &p_z(\phi \,\mathcal{U}\, \phi') \geq \sum_{z' \in Z} \Delta(z, \alpha, z') \cdot p_{z'}(\phi \,\mathcal{U}\, \phi') \\ &\qquad \text{for each } \alpha \in \mathsf{enabled}(z) && \text{otherwise} \end{aligned}$$

When $\bowtie \in \{>, \geq\}$, the objective changes to maximization and the direction the last inequality is reversed.

Value iteration

One can equivalently express the system of equations described in (1) and (2) as (3) and (4) for DTMCs and MDPs respectively (for some appropriate matrix A and vector b),

$$\bar{x} = A\bar{x} + b \quad (3)$$

$$\bar{x}(z) = \max\{\Delta(z, \alpha, z') \cdot \bar{x} \,|\, \alpha \in \mathsf{enabled}(z)\} \quad (4)$$

An alternate approach to solving the above set of equations is *value iteration*, which iteratively computes the solution vector as the limit of the sequence $\{\bar{x}_i\}_{i \geq 0}$ given by $\bar{x}_{i+1} = A\bar{x}_i + \bar{b}$ starting with $\bar{x}_0(z) = 1$ if $z \in \mathsf{Prob1}$ and $\bar{x}_0(z) = 0$ otherwise, for the case of DTMCs. The iterative formulation for MDPs is also similar. Value iteration techniques remain the popular choice for industrial tools that analyze PCTL properties, because, when equipped with a suitable stopping criterion, value iteration beats state-of-the-art linear programming techniques, despite their theoretically better asymptotic complexity. State-of-the-art quantitative model checkers further enhance the performance of value iteration by performing arithmetic operations using *Multi-terminal binary decision diagrams* (MTBDDs) [18], [23]. MTBDDs generalize BDDs [12] by allowing terminal values to be different from 0 or 1. Similar to the role of BDDs in symbolic model checking [31], MTBDD based model checkers leverage the performance benefit due to the succinct representations of the data structures involved.

III. Approximate model checking

As discussed above, solving quantitative properties of DTMCs and MDPs by a reduction to linear programming does not scale well enough to make it a viable solution technique in practice. As a result, techniques to approximate solutions using floating point arithmetic, such as value iteration, have been widely adopted. In addition to errors introduced by overflows in floating point numbers, several other sources of imprecision can arise in quantitative model checkers that employ approximate solution techniques.

a) Value iteration and convergence: Value iteration, discussed above, computes a sequence of vectors $\{v_i\}_{i \geq 0}$ that converge to the solution vector V for a PCTL formula. In many cases, the sequence does not converge in a finite number of steps, and therefore model checkers terminate the sequence when successive vectors v_k and v_{k+1} become "close enough". The choice of stopping criterion is based largely on heuristics. The PRISM model checker, for example, implements two criteria (i) *absolute criterion*, and (ii) *relative criterion*. Under the absolute criterion, value iteration terminates if $\|v_{k+1} - v_k\| < \epsilon$ for some $\epsilon > 0$. Under the relative criterion, termination occurs when $\frac{\|v_{k+1} - v_k\|}{\|v_k\|} < \epsilon$. Both of these convergence criteria can result in solutions that are very far from the actual answers. In [20], the authors give a DTMC and a PCTL property whose solution is $\frac{1}{2}$, yet PRISM reports 9.77×10^{-4} for the absolute criterion and 0.198 for the relative criterion.

b) Nested reachability and PCTL: State-of-the-art quantitative model checking tools employing floating point arithmetic often fail to produce accurate solutions to properties of the form $\mathcal{P}_{\bowtie p}(\psi)$ when the probability of satisfying ψ is very close to p. First observed in [39], we shall also demonstrate this phenomenon using the DTMC from Example 1. For the sake of illustration, let $\mathrm{E}_s = \frac{1}{2}$. Clearly, from the initial state, the probability of reaching a state where sensor 1 fails is exactly $\frac{1}{2}$ and hence the formula $\mathcal{P}_{<\frac{1}{2}}$ [$\mathsf{true}\ \mathcal{U}\ (e_1^s{=}1)$] evaluates to false for the initial state. However, PRISM returns true. Errors such as these can be compounded in PCTL formulas containing nested operators, wherein the recursive step of the model checking algorithm returns an incorrect set of states. This can lead to substantial logical errors in model analysis where the reported probabilities are very far from the actual ones.

Example 3: Let us instantiate the DTMC from Example 1 with $n = 14$ sensors, $m = 1$ actuator, MAX_FAILURES=1 and with $\mathrm{E}_s = \mathrm{E}_a = \frac{1}{2}$. Recall the third PCTL property of the embedded control system given in Example 2:

$$\mathcal{P}_{=?}\,[\,\mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\,\mathcal{U}\,(e_1^s{=}1)]\,\mathcal{U}\,\mathcal{P}_{\leq 0}[\mathsf{true}\,\mathcal{U}\,(e_1^a{=}1)]\,].$$

When $\bowtie$ is $\leq$, the PRISM model checker returns the value "0.7096993582589287" for the initial state. Using our tool RationalSearch, one can verify that the probability is actually $212895/229376$, or "0.9281485421316964". Further, when $\bowtie$ is $<$, PRISM again returns "0.7096993582589287". This time, the actual solution, is 0, and our tool concurs with this. This is because, PRISM incorrectly computes the

set of states satisfying $\mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\ \mathcal{U}\ (e_1^s{=}1)]$. This error in the recursive step results in an incorrect formulation of the constraints in the outermost constrained reachability problem.

IV. Exact model checking

As demonstrated in the previous section, approximate solution techniques can lead to unreliable results and to incorrect analysis of systems. To rectify this serious limitation, several approaches that attempt to give exact (or high-precision) solutions to the PCTL model checking problem have been proposed. One such technique, *interval iteration* [9], [20] carries out two simultaneous value iterations converging to the solution from above and below, allowing one to bound the error in the approximate solution. This approach is again vulnerable to floating point errors and it doesn't directly give exact answers. Another approach for computing exact answers is *parametric model checking*. In this approach, one synthesizes symbolic algebraic expressions representing the quantitative properties of the model, by treating the parameters in the underlying model (constants, transition probabilities and rewards) symbolically. These symbolic expressions evaluate to the exact arithmetic values when instantiated with the concrete values in the model. However, generating these expressions, in general, is an expensive task. See our experiential evaluation in Section V for a comparison with this technique.

Example 4: Again consider the DTMC modeling an embedded control system with the parameters given in Example 3, where it was demonstrated that approximate model checking techniques can lead to incorrect logical analysis of the system. To guarantee the correctness of one's analysis, exact solution techniques must be employed. Unfortunately, the exact model checking engines of PRISM and STORM do not scale well enough to analyze this example, which contains about 4.8 million states and about 44 million transitions. Under our test setup (see Section V), both tools reached a 30 minute timeout when trying to analyze the properties from Example 3. On the other hand, our tool RationalSearch found the exact answer to both the formulae in under 1 minute.

The Kwek-Mehlhorn algorithm

Given an ordered set of integers of bounded size, the classical binary search algorithm can be used to locate the smallest element larger than a given value. Kwek and Mehlhorn [26] extend this methodology to efficiently locate the rational number with the smallest size in a given interval. Here we present a novel application of this technique where approximate answers to quantitative model checking problems can be used to efficiently generate exact solutions.

Consider an interval $I = [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$ with rational end-points. It was established [26] that for any interval $I = [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$, there exists a unique rational $a_{\min}(I)/b_{\min}(I)$ such that for all rational numbers $\frac{a}{b} \in I$, $a_{\min}(I) \leq a$ and $b_{\min}(I) \leq b$. Further, this minimal fraction $a_{\min}(I)/b_{\min}(I)$ can be found using Algorithm 1 from [26].

Let $Q_M = \{p/q \mid p, q \in \{1, ..., M\}\} \cap [0, 1]$. For $\mu \in \mathbb{N}$, if $\frac{a}{b} \in Q_M$ is contained in the interval $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$ of length $\frac{1}{2M^2}$ then $\frac{a}{b}$ is the unique element of Q_M in $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$. It turns out that $\frac{a}{b}$ must also be the minimal element of $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$, meaning it can be found using the algorithm from Algorithm 1 in time $O(\log M)$.

Algorithm 1 Compute the minimal rational in $[\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$

```
function FindFraction(α, β, γ, δ):
    if ⌊α/β⌋ = ⌊γ/δ⌋ and α/β ∉ ℕ then
        b, a ← FindFraction(δ, γ mod δ, β, α mod β)
        return ⌊α/β⌋ b + a, b
    else
        return ⌈α/β⌉, 1
    end if
end function
```

Rational search

In this section, we explain our approach for exact quantitative model checking of PCTL formulas. The key insight we exploit is that value iteration typically converges very fast and produces a precise enough answer. Using this precise approximation, we can then effectively construct a small interval for which the Kwek-Mehlhorn algorithm can find the exact answer. In the following, we formalize this procedure.

We begin the presentation of our exact model checking algorithm by first describing how a given approximate solution vector corresponding to a set of equations, like those in (1) and (2), can be refined to get the exact vector. This process is formalized in Algorithm 2, which takes as input the model $\mathcal{M}$, a maximum precision P and a state-indexed vector $\mathsf{V}^\dagger$ that approximates V (defined after Definition 2).

Algorithm 2 Sharpen

```
function Sharpen(M, P, V†):
    for all p ∈ {1, ..., P} do
        for all z ∈ Z do
            α, β, γ, δ ← Bounds(p, V†(z))
            V⋆(z) ← ⌊V†(z)⌋ + FindFraction(α, β, γ, δ)
        end for
        if FixPoint(M, V⋆) then
            return V⋆
        end if
    end for
    return null
end function
```

For a given precision p and state z, $\textsc{Bounds}(p, \mathsf{V}^\dagger(z))$ returns $\alpha, \beta, \gamma, \delta$ such that α is the first p decimal digits of the fractional part of $\mathsf{V}^\dagger(z)$, $\beta = 10^p$, $\gamma = \alpha + 1$ and $\delta = \beta$. Observe that α/β is the rational representation of the first p digits of the fractional part of $\mathsf{V}^\dagger(z)$. From this approximation, we identify a *sharpened* solution vector $\mathsf{V}^\star$ using the FindFraction procedure from Algorithm 1. The procedure FixPoint then tests if $\mathsf{V}^\star$ is the correct solution by checking if it satisfies (3) or (4), whichever is appropriate. The uniqueness of the solutions to the these equation systems (which follows from those of (1) and (2)) ensures that the fixpoint check is only satisfied by the desired solution vector.

If the input vector $\mathsf{V}^\dagger$ is not precise enough, then SHAPREN returns "null".

The guarantees of Algorithm 2 are formalized as follows. Let V^b satisfying $\mathsf{V}(z) - \mathsf{V}^b(z) \leq 10^{-b}$ for all $z \in Z$ be an approximate solution vector of precision b[1]. Then, Lemma 1 establishes that starting from a close enough approximation, Algorithm 2 finds the actual solution vector.

Lemma 1: Let $\mathcal{M}$ be an MDP, ψ be a PCTL path formula and V be the solution vector for $\mathcal{P}_{=?}[\psi]$. Let $b, P \in \mathbb{N}$ be such that $P \geq b$ and V^b is an approximate solution vector of precision b. If $\mathsf{V}(z) \in Q_{\lfloor\sqrt{10^b/2}\rfloor}$ for every $z \in Z$, then $\text{SHARPEN}(\mathcal{M}, P, \mathsf{V}^b) = \mathsf{V}$.

Proof (Sketch): Fix a state z and assume $\mathsf{V}(z) \in Q_M$ for $M = \lfloor\sqrt{10^b/2}\rfloor$. If $P \geq b$ then $\text{SHARPEN}(\mathcal{M}, P, \mathsf{V}^b)$ searches for $\mathsf{V}(z)$ in $I = [\alpha/\beta, \gamma/\delta]$ for $\alpha, \beta, \gamma, \delta = \text{BOUNDS}(b, \mathsf{V}^b(z))$. Now, $\mathsf{V}(z) \in I$ since $\mathsf{V}^b(z)$ satisfies $\mathsf{V}(z) - \mathsf{V}^b(z) \leq 10^{-b}$. Further, $|I| = 10^{-b} \leq \frac{1}{2M^2}$. Due to Kwek et. al. [26], we have that an interval of size $\frac{1}{2M^2}$ contains at most 1 element of Q_M. Clearly, $\text{FINDFRACTION}(\alpha, \beta, \gamma, \delta)$ returns $\mathsf{V}(z)$ which is the unique "minimal" element in $I \cap Q_M$. □

Building on the SHARPEN procedure, Algorithm 3 computes the values in $\mathcal{P}_{=?}[\phi_1 \mathcal{U} \phi_2]$ for state formulas of the form $\mathcal{P}_{\bowtie p}[\phi_1 \mathcal{U} \phi_2]$. It augments the value iteration phase from the standard PCTL model checking algorithm.

Algorithm 3 Rational Search

```
function RATIONALSEARCH(M, φ, ε0):
    V^init ← INIT(M, φ)
    ε ← ε0
    while true do
        V† ← VALUEITERATION(M, φ, V^init, ε)
        V* ← SHARPEN(M, ⌈log(1/ε)⌉, V†)
        if V* ≠ null then
            return V*
        end if
        V^init ← V†
        ε ← ε/10
    end while
end function
```

Algorithm 3 begins by running value iteration up to a given precision ϵ (where ϵ is used in the convergence criterion — absolute or relative — described in Section III) to determine an approximate solution vector $\mathsf{V}^\dagger$. Alternatively, value iteration could be replaced by an interval iteration algorithm and then ϵ would then represent a bound on the maximum error in the approximate solution vector. Once $\mathsf{V}^\dagger$ is computed, Algorithm 2 attempts to sharpen the approximate answer to an exact one. If it succeeds, the whole process terminates. Otherwise, $\mathsf{V}^\dagger$ is further refined by re-invoking value iteration with an increased ϵ precision and the sharpening process is repeated. When successive approximations in value iteration are computed using arbitrary precision arithmetic, Theorem 1 establishes the correctness of Algorithm 3.

Theorem 1: Let $\mathcal{M}$ be a MDP, ψ be a PCTL path formula and V be the solution vector for $\mathcal{P}_{=?}[\psi]$ and $\epsilon_0 \in \mathbb{Q}^{>0}$. Then, $\text{RATIONALSEARCH}(\mathcal{M}, \mathcal{P}_{\bowtie p}[\psi], \epsilon_0)$ terminates and returns the exact solution vector V.

Proof (Sketch): It is easy to see that there is a $b > 0$ such that, for every state z, $\mathsf{V}(z) \in Q_N$ for $N = \lfloor\sqrt{10^b/2}\rfloor$. Now, since value iteration converges in the limit, we have that the first b digits of $\mathsf{V}^\dagger(z)$ match that of $\mathsf{V}(z)$ for each state $z \in Z$, eventually. Also, in every iteration of the loop in Algorithm 3, SHARPEN is invoked with an incremented value of P and eventually $P \geq b$. □

While both Lemma 1 and Theorem 1 are stated for formulae of the kind $\mathcal{P}_{=?}[\psi]$, they can be easily re-factored to reason about formulas of the form $\mathcal{E}_{=?}[\phi]$.

Example 5: Our experiments show that Algorithm 3 can make non-trivial improvements to solution quality. Consider the standard example of tossing N biased coins independently, where each coin yields heads with probability 1/3 and tails with probability 2/3. Analyzing the DTMC model to compute the probability of the event that 11 coins land heads, PRISM's floating-point model checker returned the decimal "0.000005645029269476758". Our tool was able to correctly determine the exact probability to be 1/177,147 by starting with the first 12 digits of this approximate answer. This is remarkable given that the period of this fraction (and hence its most succinct decimal representation) is almost 20,000 digits long. Moreover, the algorithm is able to simultaneously infer the reachability probabilities for *all* of the roughly 200,000 states of the model during a single fixpoint check. This illustrates another advantage of our technique; the algorithm is agnostic of the number of initial states in the system. The exact model checking engine of PRISM, on the other hand, currently only supports systems with a single initial state.

V. RESULTS

We have implemented Algorithm 3 in our tool RATIONALSEARCH as an extension of the PRISM model checker (version 4.3.1). RATIONALSEARCH is available for download at [5]. PRISM is comprised of four solution engines, three of which (MTBDD, HYBRID, SPARSE) are based on symbolic methods using compact data structures like MTBDDs. The fourth engine (EXPLICIT) manipulates sparse matrices, vectors and bit-sets directly. RATIONALSEARCH implements Algorithm 3 on top of all four engines. It intercepts PRISM's routine for solving constrained reachability probabilities and rewards, sharpening the probabilities every time it is invoked.

The EXPLICIT engine of PRISM is implemented in Java. To support this engine, our tool uses the libraries JScience [4] and Apfloat [1] to construct the transition matrix using rational entries, perform matrix-vector multiplications for the fixpoint

[1]Notice that we require $\mathsf{V}^b(z)$ to be a lower bound for $\mathsf{V}(z)$, instead of $|\mathsf{V}(z) - \mathsf{V}^b(z)| \leq 10^{-b}$. Such an approximation can indeed be obtained from, say, value iteration.

1	2	3	4	5	6	7	8	9	10	11
Model			RationalSearch						PRISM	STORM
Name	Parameter	States	Explicit		MTBDD		Hybrid			
			Time (s)	Overhead (%)	Time (s)	Overhead (%)	Time (s)	Overhead (%)	Time (s)	Time (s)
Biased Coins	11	177,147	23.1	336	0.125	179	0.178	225	1449.7	3.2
Dice	6	4,826,809	OOM	N/A	1.8	2.1	6.5	12	TO	63
Din. Cryptographers	8	187,457	18.9	197	0.278	70	0.364	105	356.2	2.4
Din. Philosophers	3	956	0.41	165	1.9	4.8	0.133	98	3.128	0.65
ECS	14	4,815,782	OOM	N/A	2.4	23	11.6	79	TO	TO
Fair Exchange	400	321,600	14.6	423	2.0	44	2.2	51	TO	1.1
Firewire	11,000	428,364	122.2	225	15.1	0.2	19.5	21	232.3	29.5
Leader Election	4	12,302	1.8	226	5.0	30	20.4	25	80	0.042
Virus Infection	3	809	0.5	165	2.8	52	0.17	93	0.98	0.032

Fig. 2. **Experimental Results:**. Columns 1-3 describe the benchmark examples. Columns 4-11 report the performance metrics for the various exact solution engines. Running times are reported in seconds, averaging over 5 measurements. For the RationalSearch engines [Explicit, MTBDD, Hybrid], we additionally report overhead percentages which are calculated by comparing the running times of PRISM's approximate engines with the corresponding extensions in RationalSearch. We use absolute convergence criterion ($\epsilon = 10^{-12}$) for the three engines in RationalSearch and the corresponding approx. engines in PRISM. A TO in columns 8, 10 and 11 represents a timeout (set to be 30 minutes). OOM indicates an out of memory exception.

check in Algorithm 3, and implement the Kwek-Mehlhorn algorithm (Algorithm 1). PRISM implements the remaining three engines using an extension of the CUDD library [2]. The off-the-shelf version of CUDD only supports floating point numbers at the terminals. RationalSearch enhances CUDD by allowing terminals to hold either floating points or arbitrary precision rational numbers provided by the GNU MP library [3]. Our extension allows the data type at the node to be easily interchanged and the full suite of MTBDD operations can be performed regardless of the data type. RationalSearch constructs the transition matrix as a rational MTBDD and uses Algorithm 2 to generate candidate solution vectors over rationals starting from approximate solution vectors represented as floating point MTBDDs given by PRISM's value iteration procedure.

Evaluation: We evaluated our tool against all of the examples involving quantitative reachability and rewards from the PRISM benchmark suite and case studies [6], [7] and compared the results with the exact parametric engines implemented in PRISM and STORM. Our tests were carried out on an Intel core i7 dual core processor @2.2GHz with 4Gb RAM running macOS 10.12.4. A summary of the performance on quantitative PCTL properties is given in Figure 2. The model checking times reported in the table include the time required to build the transition matrix using rational numbers. The reported times are an average of five runs for each engine/tool. We observed that, among the engines based on symbolic techniques, the SPARSE engine of PRISM was being consistently outperformed by the MTBDD and Hybrid engines. We, therefore, do not report its performance statistics.

Analysis of Results: The objective of our experimental evaluation is two-fold. First, we would like to compare our implementation against state-of-the-art tools for exact quantitative model checking (see Columns 4,6,8,10,11). The second objective is to analyze the performance overhead that our technique adds to approximate model checking techniques (see Columns 5,7,9). The overhead measures the additional time incurred by RationalSearch when compared to PRISM's engines that perform only value iteration using inexact floating point arithmetic.

Each implementation Explicit, MTBDD and Hybrid of RationalSearch significantly outperforms PRISM's exact engine; in many cases, by several orders of magnitude. We also found at least one class of examples (Biased Coins) where PRISM's exact engine gave incorrect probabilities.

The comparison with STORM is more competitive. On most examples with a large number of states (ECS, Biased Coins, Dice), the running times achieve by RationalSearch are much lower than those from STORM. On smaller examples, the times were more comparable, with RationalSearch running slightly faster on the majority of examples.

On several examples with large state spaces, the Explicit engine fails due to an out of memory exception. This can be attributed to the fact that the implementation stores two copies of the transition matrix in memory. On all the examples where Explicit fails, the symbolic engines (MTBDD and Hybrid) find the solution quickly.

For the symbolic engines MTBDD and Hybrid, we found that RationalSearch can infer exact solutions from approximate ones, while typically not adding more than double overhead to approximate engines that are known to run

extremely fast. The EXPLICIT engine incurs a much higher overhead. This difference is due to the fact that MTBDD's perform symmetry reductions, storing a single copy of each possible terminal value. This allows our implementation to run the Kwek-Mehlhorn algorithm a single time for all states sharing the same approximate value. This luxury is not afforded by the EXPLICIT engine, with carries out the Kwek-Mehlhorn procedure for every state in the model. For nested PCTL properties (such as ECS), the SHARPEN procedure must compute multiple fixpoints, adding to the overhead time. We would note, however, that for this example RATIONALSEARCH is the only tool that found a solution without hitting a timeout.

VI. CONCLUSION

Techniques for exact model checking allow one to avoid logical errors in system analysis that can arise due to approximation techniques. We presented an algorithm and tool, RATIONALSEARCH, that computes the exact probabilities described by PCTL formulas for DTMCs and MDPs. Our tool works by sharpening approximate results obtained through value iteration, allowing it to benefit from the performance enhancements gained through approximation techniques. Our experimental evaluation concurs with this hypothesis, and shows that our approach often performs significantly better than existing exact quantitative model checking tools while also scaling to large model sizes. For future work, we plan to combine our algorithm with more precise approximation techniques such as interval iteration. We believe there are also performance enhancements that can be achieved by a tighter integration with the Kwek-Mehlhorn algorithm, wherein computations from previous iterations can be reused.

REFERENCES

[1] Apfloat. http://www.apfloat.org/.

[2] CUDD. http://vlsi.colorado.edu/~fabio/CUDD/html/.

[3] GNU Multiple Precision Arithmetic Library. https://gmplib.org/.

[4] JScience. http://jscience.org/.

[5] RationalSearch. https://publish.illinois.edu/rationalmodelchecker/.

[6] (2017) PRISM Benchmark Suite. http://www.prismmodelchecker.org/benchmarks/. [Online; accessed 1-January-2017].

[7] (2017) PRISM Case Studies. http://www.prismmodelchecker.org/casestudies/. [Online; accessed 1-January-2017].

[8] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[9] C. Baier, J. Klein, L. Leuschner, D. Parker, and S. Wunderlich, "Ensuring the reliability of your model checker: Interval iteration for markov decision processes," in *Computer Aided Verification*, 2017.

[10] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1999.

[11] D. Bhaduri, S. K. Shukla, P. S. Graham, and M. B. Gokhale, "Reliability analysis of large circuits using scalable techniques and tools," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, 2007.

[12] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, 1986.

[13] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of cryptology*, vol. 1, no. 1, 1988.

[14] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Abraham, "Prophesy: A probabilistic parameter synthesis tool," in *International Conference on Computer Aided Verification, CAV*, 2015.

[15] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *Computer Aided Verification: 29th International Conference, CAV 2017*.

[16] E. W. Dijkstra, "Self-stabilization in spite of distributed control," in *Selected writings on computing: a personal perspective*. Springer, 1982.

[17] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A formal analysis of bluetooth device discovery," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, no. 6, pp. 621–632, 2006.

[18] M. Fujita, P. C. McGeer, and J.-Y. Yang, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *Formal methods in system design*, vol. 10, no. 2-3, pp. 149–169, 1997.

[19] S. Giro, "Efficient computation of exact solutions for quantitative model checking," in *Proc. 10th Workshop on Quantitative Aspects of Programming Languages (QAPL'12)*, 2012.

[20] S. Haddad and B. Monmege, "Reachability in mdps: Refining convergence of value iteration," in *International Workshop on Reachability Problems*. Springer, 2014, pp. 125–137.

[21] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "PARAM: A model checker for parametric Markov models," in *International Conference on Computer Aided Verification (CAV'10)*, 2010.

[22] J. Han, H. Chen, E. Boykin, and J. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," *Microelectronics Reliability*, 2011.

[23] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier, "Spudd: Stochastic planning using decision diagrams," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 1999.

[24] B. Jeannet, P. D'Argenio, and K. Larsen, "Rapture: A tool for verifying Markov decision processes," in *Proc. Tools Day, affiliated to 13th Int. Conf. Concurrency Theory (CONCUR'02)*, 2002.

[25] J.-P. Katoen, M. Khattri, and I. Zapreevt, "A markov reward model checker," in *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*. IEEE, 2005.

[26] S. Kwek and K. Mehlhorn, "Optimal search for rationals," *Information Processing Letters*, vol. 86, no. 1, pp. 23–26, 2003.

[27] M. Kwiatkowska, G. Norman, and D. Parker, "Controller dependability analysis by probabilistic model checking," in *11th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'04)*, 2004.

[28] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol," in *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, 2002.

[29] ——, "Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol," *Formal Aspects of Computing*, vol. 14, no. 3, pp. 295–318, 2003.

[30] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 585–591.

[31] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[32] N. Mohyuddin, E. Pakbaznia, and M. Pedram, "Probabilistic error propagation in a logic circuit using the boolean difference calculus," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 2011, pp. 359–381.

[33] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla, "Evaluating the reliability of nand multiplexing with prism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[34] Q. Qiu, Q. Qu, and M. Pedram, "Stochastic modeling of a power-managed system-construction and optimization," *IEEE Transactions on computer-aided design of integrated circuits and systems*, 2001.

[35] M. Rabin, "Randomized Byzantine generals," in *Proc. Symposium on Foundations of Computer Science*, 1983, pp. 403–409.

[36] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[37] J. J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical techniques for analyzing concurrent and probabilistic systems*. American Mathematical Soc., 2004.

[38] R. St-Aubin, J. Hoey, and C. Boutilier, "Apricodd: Approximate policy construction using decision diagrams," in *Advances in Neural Information Processing Systems*, 2001, pp. 1089–1095.

[39] R. Wimmer, A. Kortus, M. Herbstritt, and B. Becker, "Probabilistic model checking and reliability of results," in *Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on*. IEEE, 2008, pp. 1–6.