Advances and Challenges of Probabilistic Model Checking

Marta Kwiatkowska, Gethin Norman, David Parker

Advances and Challenges of Probabilistic Model Checking

Marta Kwiatkowska
Computing Laboratory
Oxford University
Oxford, UK
Email: marta.kwiatkowska@comlab.ox.ac.uk

Gethin Norman
Department of Computing Science
University of Glasgow
Glasgow, UK
Email: gethin@dcs.gla.ac.uk

David Parker
Computing Laboratory
Oxford University
Oxford, UK
Email: david.parker@comlab.ox.ac.uk

***Abstract*—Probabilistic model checking is a powerful technique for formally verifying quantitative properties of systems that exhibit stochastic behaviour. Such systems are found in many domains: probabilistic behaviour may arise, for example, due to failures of unreliable components, communication across lossy media, or through the use of randomisation in distributed protocols. In this paper, we give a short overview of probabilistic model checking and of PRISM (www.prismmodelchecker.org), currently the leading software tool in this area. We then mention some of the limitations of these techniques, describe some of the advances that are being made to overcome them, and outline key challenges that remain in this research area.**

I. Introduction

Computerised systems can now be found in almost all aspects of everyday life, from the complex control systems found in today's cars and planes, to communication and multimedia devices such as mobile phones. There is a growing need for rigorous, formal techniques to verify the correctness of such systems. Furthermore, to ascertain the correctness of complex devices operating in unknown environments, it also becomes important to analyse *quantitative* properties such as reliability, responsiveness or resource usage. For this, it is essential to consider the inherently *probabilistic* nature of real systems: components in an embedded device may be prone to failure; or messages sent across communication networks may get lost. Furthermore, wireless technologies such as Bluetooth and ZigBee use randomisation to establish networks of devices efficiently and reliably.

Probabilistic model checking is a formal verification method for analysing quantitative properties of systems that exhibit stochastic behaviour. The basic idea is to construct a mathematical model that captures the system's behaviour, and then use it to analyse formally-specified quantitative properties. These could include, for example, "the probability of an airbag failing to deploy within 0.02 seconds", "the expected time for a network protocol to send a packet" or "the expected power consumption of a sensor network during 1 hour of operation". Considerable progress in the field of probabilistic model checking has been made in recent years. Tools such as PRISM [1] and MRMC [2] have been developed and successfully applied to the quantitative analysis of a diverse range of systems, from wireless communication protocols to biological signalling pathways.

In this paper, we give an overview of probabilistic model checking and of the software tool PRISM. We discuss some of the research directions in which progress is being made to advance these techniques and highlight some of the key challenges that remain.

II. Probabilistic Model Checking & PRISM

Probabilistic model checking is based on the construction and analysis of a probabilistic model. It has been applied to a variety of different types of model, typically variants of Markov chains and Markov decision processes. We begin this section by describing the process of probabilistic model checking on the simplest of these models, discrete-time Markov chains, and give a simple, illustrative example. We then describe how this basic model can be extended in several directions, including the addition of costs and rewards, nondeterminism and continuous-time. Finally, we give a brief overview of the PRISM model checker, which provides support for all of these types of models.

A. Discrete-Time Markov Chains

Discrete-time Markov chains (DTMCs) model systems whose behaviour at each point in time can be described by a discrete probabilistic choice over several possible outcomes. This might represent, for example, an electronic coin toss, as used to implement a randomised algorithm, or transmission of a message over an unreliable channel, which is known to fail with a certain probability. Essentially, a DTMC can be thought of as a labelled state-transition system in which each transition is annotated with a probability value indicating the likelihood of its occurrence.

Definition 1. A discrete-time Markov chain (DTMC) is a tuple $\mathsf{D} = (S, \bar{s}, \mathbf{P}, AP, L)$ where:

- S is a set of *states*;
- $\bar{s} \in S$ is an *initial state*;
- $\mathbf{P} : S \times S \to [0,1]$ is a *transition probability matrix* such that $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ for all $s \in S$;
- AP is a set of atomic propositions;
- $L : S \to 2^{AP}$ is a *labelling function* that assigns, to each state $s \in S$, a set $L(s)$ of atomic propositions.

For a state $s \in S$ of a DTMC D, the probability of moving to a state $s' \in S$ in one discrete step is given by $\mathbf{P}(s, s')$. Each state in D represents one possible configuration of the system being modelled; each transition represents the possibility to evolve from one configuration to another. A *path* of D, which gives one possible evolution of the Markov chain, is a sequence of states $s_0 s_1 s_2 \ldots$ such that $s_0 = \bar{s}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$.

Classically, analysis of DTMCs often focusses on *transient* or *steady-state* behaviour, i.e. the probability of being in each state of the chain at a particular instant in time or in the long-run, respectively. Probabilistic model checking adds to this the ability to reason about path-based properties, which can be used to specify constraints on the probability that certain desired behaviours are observed. Formally, this is done by defining a probability space over the set of all paths through the model [3]. Properties are then expressed using temporal logic. For DTMCs, specifications can be written in PCTL (Probabilistic Computation Tree Logic) [4], a probabilistic extension of the temporal logic CTL.

Definition 2. The syntax of PCTL is given by:

$$\begin{array}{rcl} \Phi & ::= & \mathit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathrm{P}_{\sim p}[\,\phi\,] \\ \phi & ::= & \mathrm{X}\,\Phi \mid \Phi\,\mathrm{U}^{\leq k}\,\Phi \mid \Phi\,\mathrm{U}\,\Phi \end{array}$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0, 1]$ and $k \in \mathbb{N}$.

PCTL formulae are interpreted over the states of a DTMC. We say that a state $s \in S$ *satisfies* a PCTL formula Φ, denoted $s \models \Phi$, if it is true for s. The key operator in PCTL is $\mathrm{P}_{\sim p}[\,\phi\,]$ which means that the probability of a *path formula* ϕ being true in a state satisfies the bound $\sim p$. As path formulae (specified separately on the second line of Definition 2), we allow $\mathrm{X}\,\Phi$ ("Φ is satisfied in the *next* step") $\Phi_1\,\mathrm{U}^{\leq k}\,\Phi_2$ ("Φ_2 is satisfied within k steps and Φ_1 is true *until* that point") and $\Phi_1\,\mathrm{U}\,\Phi_2$ ("Φ_2 is eventually satisfied and Φ_1 is true *until* then"). A simple example is:

$$\mathrm{P}_{\leq 0.15}\,[\ \neg \mathit{fail}_A\ \mathrm{U}\ \mathit{fail}_B\ ]$$

which states that "the probability that component B fails before component A is at most 0.15". Here, fail_A and fail_B are atomic propositions, used to label states of a DTMC in which a property of interest (e.g. "component A has failed") is true. In practice, it is common to take a more *quantitative* approach, instead writing formulae of the following kind:

$$\mathrm{P}_{=?}\,[\ \neg \mathit{fail}_A\ \mathrm{U}\ \mathit{fail}_B\ ]$$

which asks simply "*what is* the probability that component B fails before component A?".

Several other useful operators can be derived from the basic PCTL syntax given above. This includes, for example, $\mathrm{F}\,\Phi \equiv \mathit{true}\,\mathrm{U}\,\Phi$ ("*eventually* Φ becomes true"), $\mathrm{G}\,\Phi \equiv \neg\mathrm{F}\neg\Phi$ ("Φ is *always* true") and time-bounded variants of these. Examples of properties using these operators are:

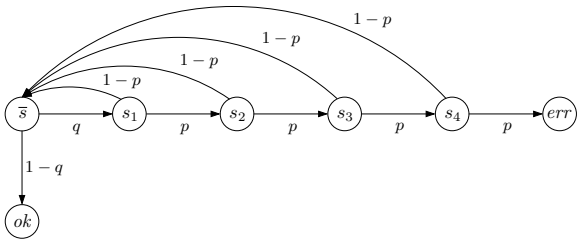

Figure 1. Simple DTMC model of the Zeroconf protocol.

- $\mathrm{P}_{=?}\,[\,\mathrm{F}\,(\mathit{fail}_A \vee \mathit{fail}_B)\,]$ - "the probability that either component A or B fails at some point";
- $\mathrm{P}_{=?}\,[\,\mathrm{G}^{\leq 3600}\,\neg(\mathit{fail}_A \vee \mathit{fail}_B)\,]$ - "the probability of no failures occurring in the first hour".

Model checking a PTCL formula over a DTMC requires a combination of graph-based algorithms and numerical solution techniques. For the latter, the most common problems (e.g. computing the probabilities that a U, F or G path formula is satisfied), reduce to solving a system of linear equations. For scalability reasons, in implementations of probabilistic model checking this is usually done with *iterative* numerical methods such as Gauss-Seidel, rather than more classical *direct* methods such as Gaussian elimination. A wider range of properties can be expressed by using more expressive logics such as LTL or PCTL*, however model checking becomes slightly more expensive.

B. Example: The Zeroconf Protocol

We now describe a small, but illustrative, example of a system modelled as a DTMC. In fact, this is a simplified version of a real probabilistic model checking case study: the *Zeroconf* dynamic network configuration protocol [5]. Zeroconf provides a distributed, 'plug-and-play' solution for the problem of configuring IP addresses in a small ad-hoc network where address allocation is managed by the individual devices connecting to the network.

The basic idea of the protocol is as follows. On connection to the network, a device first randomly select an IP address from a pool of 65024 available addresses, specifically allocated for this purpose (169.254.1.0–169.254.254.255). It then broadcasts four ARP (Address Resolution Protocol) packets, called *probes*, to the other devices in the network. These probes contain the chosen IP address and constitute requests to start using this address. If another device is already using the IP address, it should respond with an ARP reply packet, asserting its claim to the address. In this case, the original device will restart the protocol, i.e. randomly select a new address and then send new probes.

Figure 1 shows a simple DTMC model of the protocol operating. The state space S is $\{\bar{s}, s_1, s_2, s_3, s_4, \mathit{err}, \mathit{ok}\}$, with $\bar{s}$ representing the initial state in which a new device is about to join a network. States are drawn as circles and the transition probability matrix $\mathbf{P}$ by probability-labelled

arrows between states. The two possible transitions from $\bar{s}$ represent the possible outcomes when the device picks a randomly selected IP address. Assuming that there are M other devices already in the network, $q = M/65024$ gives the probability that the address chosen is already in use.

In the case where the address *is* unique (with probability $1-q$), we choose not to model the sending of probes, moving directly to state ok. In the case where the joining devices fails to select a unique address, the state s_i in the DTMC models the ith probe being sent out. We assume that, with probability p, a probe does not get a reply (due e.g. to a message being lost or a receiving device being busy). If a reply is received, the protocol is restarted and the DTMC returns to state $\bar{s}$; if all four probes fail to receive responses, we move to state err, indicating that the joining device will start using an invalid address.

We can capture some interesting properties of the Zeroconf protocol using PCTL. Assuming that states err and ok are labelled with atomic propositions of the same name, we can express for example:

- $\mathrm{P}_{=?}[\,\mathrm{F}\ err\,]$ - "the probability that the protocol results in an invalid IP being used";
- $\mathrm{P}_{=?}[\,\mathrm{F}^{\leq k}\,(ok \vee err)\,]$ - "the probability that the protocol completes within k time units".

C. Adding Costs & Rewards

DTMCs (and other probabilistic models) can be augmented with *reward* (or *cost*) information, which allows reasoning about a wide range of additional quantitative measures. Formally, for a DTMC $\mathsf{D} = (S, \bar{s}, \mathbf{P}, AP, L)$, we define a *reward* structure $(\underline{\rho}, \boldsymbol{\iota})$ that allows specification of two distinct types of rewards: *state rewards*, which are assigned to states by means of the reward function $\underline{\rho} : S \to \mathbb{R}_{\geq 0}$, and *transition rewards*, which are assigned to transitions by means of the reward function $\boldsymbol{\iota} : S \times S \to \mathbb{R}_{\geq 0}$. The state reward $\underline{\rho}(s)$ is the reward acquired in state s per time step, i.e. a reward of $\underline{\rho}(s)$ is incurred if the DTMC is in state s for 1 time step and the transition reward $\boldsymbol{\iota}(s, s')$ is acquired each time a transition between states s and s' occurs.

Reward structures can be used to represent a variety of different aspects of a system model, for example "the number of messages successfully transmitted by a protocol" or "the amount of time that a server spends operational". Although we consistently use the terminology "rewards" here, these values can equally be considered as "costs", e.g. to model "power consumption" or "number of message packages dropped".

To express reward-based properties of DTMCs, the logic PCTL can be extended [6] with additional operators:

$$\mathrm{R}_{\sim r}[\,\mathrm{C}^{\leq k}\,] \;\big|\; \mathrm{R}_{\sim r}[\,\mathrm{I}^{=k}\,] \;\big|\; \mathrm{R}_{\sim r}[\,\mathrm{F}\ \Phi\,] \;\big|\; \mathrm{R}_{\sim r}[\,\mathrm{S}\,]$$

where $\sim \in \{<, \leq, \geq, >\}$, $r \in \mathbb{R}_{\geq 0}$, $k \in \mathbb{N}$ and Φ is a PCTL formula. The R operators properties about the *expected* value of rewards. The formula $\mathrm{R}_{\sim r}[\,\psi\,]$, where ψ denotes one of the four possible operators given in the grammar above, is satisfied in a state s if, from s, the expected value of reward ψ meets the bound $\sim r$. The possibilities for ψ are: $\mathrm{C}^{\leq k}$, which refers to the reward *cumulated* over k time steps; $\mathrm{I}^{=k}$, the state reward at time *instant* k (i.e. after exactly k time steps); $\mathrm{F}\ \Phi$, the reward cumulated before a state satisfying Φ is *reached*; and S, the long-run (*steady-state*) rate of reward accumulation. As for the P operator, we often use properties of the form $\mathrm{R}_{=?}[\,\psi\,]$, with the meaning "what is the expected reward?".

Returning to the Zeroconf case study from the previous section, we now give two examples of rewards structures and accompanying temporal logic properties. Consider first the reward structure $(\underline{\rho}_1, \boldsymbol{\iota}_1)$ which assigns $\underline{\rho}_1(\bar{s}) = 1$, $\underline{\rho}_1(s) = 0$ to all other states s and $\boldsymbol{\iota}_1(s, s') = 0$ to all transitions $s \to s'$. Second, consider the reward structure $(\underline{\rho}_2, \boldsymbol{\iota}_2)$ which assigns $\underline{\rho}_2(s) = 0$ to all states s, $\boldsymbol{\iota}_2(s_i, s_{i+1}) = \boldsymbol{\iota}_2(s_4, err) = 1$ and $\boldsymbol{\iota}_2(s, s') = 0$ to all other transitions $s \to s'$. Using the formula $\mathrm{R}_{=?}[\,\mathrm{F}\,(ok \vee err)\,]$ with each reward structure would yield the expected number of random IP addresses generated and the expected number of message failures, respectively, for the duration of the protocol.

Like for the basic operators of PCTL, model checking for the reward operators above reduces to a combination of graph algorithms and linear equation system solution. See e.g. [6] for further details.

D. Adding Nondeterminism: Markov Decision Processes

DTMCs represent a *fully probabilistic* model of a system, i.e. in each state of the model, the exact probability of moving to each other state is always known. In many instances, this does not realistically model a system's behaviour, because it behaves in a *nondeterministic* fashion. Nondeterminism can be useful to model, for example: *concurrency* between system components operating in parallel; *unknown* behaviour of a system's environment; *underspecification* of certain system parameters; or *abstraction* of a complex system using a simpler one. *Markov decision processes* (similar to *probabilistic automata*) are one of the most common formalisms for modelling systems with both probabilistic and nondeterministic behaviour.

Definition 3. A Markov-decision process (MDP) is a tuple $\mathsf{M} = (S, \bar{s}, Act, \mathbf{Steps}, AP, L)$ where:

- S, $\bar{s} \in S$, AP and $L : S \to 2^{AP}$ are as for DTMCs;
- Act is a set of *action labels*;
- $\mathbf{Steps} : S \times Act \to Dist(S)$ is the (partial) *transition probability function*, with $Dist(S)$ denoting the set of all discrete probability distributions over S.

In each state s of an MDP, the successor state is decided in two steps: first, nondeterministically selecting an available action $a \in Act$ (i.e. one for which $\mathbf{Steps}(s, a)$ is defined); and, second, randomly choosing the successor according to the probability distribution $\mathbf{Steps}(s, a)$.

To reason formally about the behaviour of MDPs, we use the notion of *adversaries*, which resolve all of the nondeterministic choices in a model. In the case, for example, where nondeterminism is used to model concurrency between components, an adversary represents one possible scheduling of the components over the lifetime of the system. Under a particular adversary, the behaviour of an MDP is fully probabilistic and, as for DTMCs, we can define a probability space over the possible paths through the model. We can then reason about the best- or worst-case system behaviour by quantifying over all possible adversaries: for example, we can compute the minimum or maximum probability that some event occurs.

To formally specify properties, we again use the logic PCTL, with identical syntax to the DTMC case, but with an implicit quantification over adversaries. The $\mathrm{P}_{=?}$ operator used for DTMCs is replaced with two variants $\mathrm{P}_{\min=?}$ and $\mathrm{P}_{\max=?}$. Example properties include:

- $\mathrm{P}_{\geq 1}\,[\,\mathrm{F}\;\mathit{end}\,]$ - "under all possible adversaries, the algorithm always terminates with probability 1";
- $\mathrm{P}_{\max=?}\,[\,\mathrm{F}\;\mathit{lost}\,]$ - "the maximum probability, across all possible schedulers, of the protocol losing a message".

Model checking MDPs requires solution of linear optimisation problems, rather than linear equation systems. In practice, this is often done using dynamic programming [7].

E. Adding Time: Continuous-Time Markov Chains

DTMCs and MDPs are both *discrete-time* models: the progress of time is modelled by discrete time steps, one for each transition of the model. For many systems, it is preferable to use a *continuous* model of time, where the delays between transitions can be arbitrary real values. One popular model is *continuous-time Markov chains*, in which transition delays are assumed to be modelled by exponential distributions. These are in common use across many fields, for example in performance evaluation, to study the reliability of computer networks or communication systems, and in systems biology, to model cellular signalling pathways.

Definition 4. A continuous-time Markov chain (CTMC) is a tuple $\mathsf{C} = (S, \bar{s}, \mathbf{R}, AP, L)$ where:

- S, $\bar{s} \in S$, AP and $L : S \to 2^{AP}$ are as for DTMCs;
- $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is the *transition rate matrix*.

The matrix $\mathbf{R}$ assigns a *rate* to each pair of states in the CTMC. A transition can only occur between states s and s' if $\mathbf{R}(s,s')>0$ and, in this case, the delay before the transition can occur is modelled as an exponential distribution with rate $\mathbf{R}(s,s')$, i.e. the probability of this transition being triggered within t time-units is $1 - e^{-\mathbf{R}(s,s')\cdot t}$. Usually, in a state s, there is more than one state s' for which $\mathbf{R}(s,s')>0$, which is referred to as a *race condition*. The first transition to be triggered determines the next state of the CTMC.

As for DTMCs, a probability space over the paths through a CTMC can be defined [8] and we can reason about the probability of certain events occurring. To specify such properties, the logic CSL [9], [8] has been proposed, which extends PCTL with continuous versions of the time-bounded operators, such as $\mathrm{P}_{=?}\,[\,\mathrm{F}^{\leq t}\;\Phi\,]$, and a steady state operator S. Examples of CSL properties are:

- $\mathrm{P}_{=?}\,[\,\mathrm{F}^{\leq 1.5}\mathit{fail}_A\,]$ - "the probability of component A failing within 1.5 hours";
- $\mathrm{S}_{=?}\,[\,\neg\mathit{fail}_A \wedge \neg\mathit{fail}_B\,]$ - "the long-run probability that both components A and B are operational".

Model checking for CTMCs is similar to DTMCs, but also uses techniques from performance evaluation, in particular *uniformisation*, an efficient iterative numerical method for computing transient probabilities of CTMCs.

F. The PRISM Tool

PRISM [1], [10] is a probabilistic model checker developed initially at the University of Birmingham and now at the University of Oxford. It currently has direct support for DTMCs, MDPs and CTMCs. As will be described in the following sections, it is currently being extended with support for other models such as probabilistic timed automata. All of these models are specified using the PRISM modelling language, a single high-level language for model description based on guarded command notation.

PRISM provides model checking for the various types of properties discussed in the previous section, namely those expressible in the logics PCTL, CSL, LTL and PCTL*, as well as extensions for quantitative properties (e.g. the $\mathrm{P}_{=?}$ operator) and for costs and rewards. For model checking, multiple efficient engines are included. These are primarily based on *symbolic* techniques (see Section III-D) but PRISM also makes extensive use of *explicit* storage schemes such as sparse matrices and arrays. Approximate computations, based on Monte Carlo techniques, are also supported.

PRISM's graphical user interface provides a model editor, a simulator for model debugging and graph-plotting functionality. The latter is particularly useful in combination with PRISM's notion of *experiments*, which is a way of automating multiple instances of model checking. This allows the user to investigate the value of quantitative model checking queries as one or more parameters of the model or property are varied, and is often a very useful way of identifying interesting patterns or trends in the behaviour of a system.

PRISM is free and open-source (released under the GPL license), and runs on most major operating systems. The website [10] provides downloads of the tool and its source code, as well as an online manual, tutorials and publications. The site also includes an extensive case study repository, with more than 50 examples. These cover a broad range of application domains: PRISM has been applied to the analysis of wireless communication protocols such as Bluetooth and

Zigbee, randomised security protocols for anonymity and contract signing, biological signalling pathways, dynamic power management schemes and many others [10].

III. Recent Advances

Probabilistic model checking has already shown itself to be a powerful and widely applicable method. The ultimate challenge, however, is to extend and improve these techniques such that they can handle the scale and complexity of real-world systems and become a valuable tool for the designers of such systems. In this section, we survey some of the recent advances in the field of probabilistic model checking, focussing in particular on existing and ongoing developments to the PRISM tool.

The topics we discuss fit into two distinct categories: (i) extending the range of models to which probabilistic model checking can be applied; (ii) improving the scalability of the techniques. The directions are of course closely linked: simpler models, such as those described in the previous section, yield more tractable verification procedures, but at the expense of a less precise system model. More expressive models can better capture the real behaviour of a system: here, we focus on combining both nondeterminism and timing characteristics into probabilistic models. With regards to scalability, we discuss two topics: *abstraction* and *compositional verification*. For the former, we discuss both the different ways of defining abstractions, and the use of *abstraction refinement* to automate their construction. For the latter, we describe one particular approach, based on the *assume-guarantee* paradigm.

A. Model Checking for Probabilistic Real-time Systems

Section II described how MDPs and CTMCs, respectively, extend DTMCs with support for nondeterminism and a continuous-model of time. Several different approaches exist to combine both of these factors. One prominent model in this area is *probabilistic timed automata* (PTAs) [11], [12], [13]. These support probability, nondeterminism and real-time. They can be seen as MDPs, extended with a set of real-valued clocks or, alternatively, an extension of the well-known *timed automata* [14] formalism with discrete probabilistic choice.

The underlying semantics of a PTA is an MDP with an infinite state space. This necessitates specialised techniques for their analysis. Perhaps the simplest is the *digital clocks* [15] approach, which, for a (slightly) restricted class of PTAs, performs a conversion to a finite-state MDP in which clocks can only take integer values. By manually translating PTAs in this way directly into the PRISM modelling language, a variety of PTA case studies have been successfully studied [10]. Automated translations also exist [16] and are being developed for inclusion in PRISM.

Other techniques for PTA analysis make use of *zones*, which capture sets of clock valuations symbolically in a way that can be implemented efficiently with data structures such as difference-bound matrices (DBMs). Two such zone-based approaches include *forwards reachability* [13] and *backwards reachability* [17]. Another is the technique of [18], which is built upon the quantitative abstraction-refinement methods discussed in the next section. Recently, there has been increased interest in the development of tools and techniques for model checking PTAs [19], [18], [16], [20]. The next version of PRISM will include native support for PTAs, via the techniques of [18] and [15].

An alternative way to combine probability, nondeterminism and a continuous model of time is to use models such as *continuous-time Markov decision processes* (CTMDPs) [7] and *interactive Markov chains* (IMCs) [21]. The former are in fact well studied in the context of optimisation problems on, for example, communication networks or queueing systems, but are now attracting attention in the probabilistic verification community. Both models incorporate random timing delays in the style of CTMCs; CTMDPs allow nondeterminism between several such behaviours, in the style of MDPs, and IMCs maintain a separation between the two types of behaviour, permitting a more compositional approach to modelling and analysis.

B. Quantitative Abstraction Refinement

The use of *abstraction*, in which certain aspects of a system model are hidden to create a less precise but more manageable model, has proved to be essential for the scalability of non-probabilistic model checking. Substantial research in this area has resulted in the development of both elegant mathematical frameworks to reason about abstraction, and state-of-the-art model checking tools that have implemented the techniques and applied them to large real-life systems. In the field of probabilistic model checking, abstraction is a more complex topic and is thus less well developed, but it is recognised as an essential research direction.

One important question is how to define the abstraction of a given probabilistic model. The situation is more complicated than the non-probabilistic case because: (a) there are several different types of model in common use; and (b) there are different ways of abstracting each one. For DTMCs, it has been proposed to define an abstraction as an *abstract Markov chain* [22] in which transitions between (abstract) states are labelled with intervals of probabilities. In fact, these abstractions can also be thought of as MDPs. For CTMCs, a similar approach can be taken, treating abstractions as CTMDPs [23]. In both cases, the DTMC or CTMC can be related to its abstraction using the notion of probabilistic simulation, which preserves some important classes of properties, such as probabilistic safety properties.

Much of the work on abstraction for probabilistic models has focussed, not on Markov chains, but on MDPs. Here, two distinct approaches have proved popular: representing an abstraction of an MDP either as another MDP [24] or

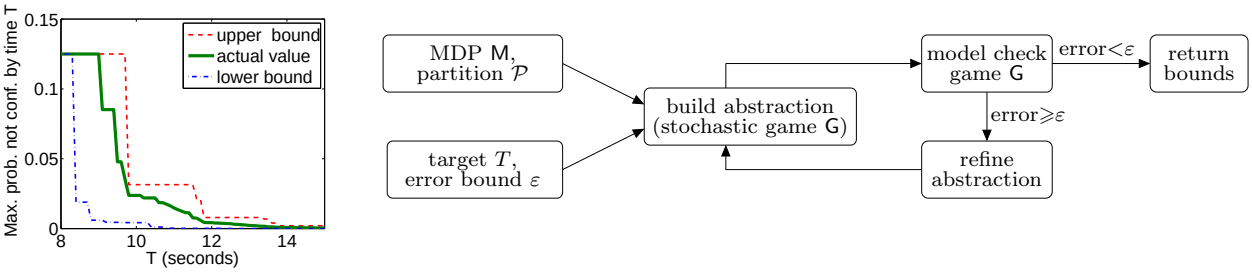

Figure 2. Quantitative abstraction refinement. Left: quantitative results obtained from an abstraction of a model of the Zeroconf network protocol. Right: Illustration of the quantitative abstraction-refinement loop for Markov decision processes.

as a stochastic two-player game [25]. The former is usually cheaper to construct but yields less information. A stochastic game G, abstracting an MDP M, gives both lower and upper bounds on either the minimum or maximum probability of a path formula being satisfied (e.g. on the results of $\mathrm{P}_{\min=?}[\,\mathrm{F}\ T\,]$ or $\mathrm{P}_{\max=?}[\,\mathrm{F}\ T\,]$). An MDP abstraction [24], by contrast, gives only a one-sided bound on each property. The plot in Figure 2 (left) shows an example of lower and upper bounds obtained from a stochastic game abstraction of a large MDP model of the Zeroconf protocol.

The other key question in this area is how to construct "good" abstractions. In the work cited above, an abstraction is defined based on a partition of the states of the original model (called *concrete* states) into subsets (called *abstract* states). The challenge is therefore to find a partition that results in abstract model that is small enough to be tractable but yields useful quantitative results, e.g. in the case of stochastic games, tight lower and upper bounds on the property of interest. A popular approach is to use *abstraction refinement*, which takes an existing abstraction and produces a finer (more precise) one. This can form the basis of fully automated abstraction generation, starting with a simple coarse abstraction and then iteratively refining until the abstraction is sufficiently precise. Such techniques were first proposed for MDPs in [24]. In [26], a probabilistic version of the classic CEGAR (counterexample-guided abstraction refinement) approach is presented, which combines several different verification technologies: predicate abstraction, interpolation and probabilistic counterexample generation.

We describe here in more detail an alternative approach called *quantitative abstraction refinement* [27], [28], [18]. To date, this has been applied primarily to the stochastic game abstractions of MDPs described above, but the approach is equally applicable to Markov chains. The basic idea, illustrated in Figure 2 (right), is to build an initial abstraction (say, from an MDP M and a coarse partition of its states) and then iteratively refine it based on verification results for some quantitative property (say, the maximum probability of reaching a target T). In particular, a stochastic game abstraction of M gives lower/upper bounds on this property. The difference between the bounds (the "error") indicates how precise the abstraction is. The bounds also identify which parts of the model need to be refined. This process is repeated iteratively until the error falls below some pre-specified tolerance ε. Quantitative abstraction refinement has been applied to the verification of PTAs [18], a probabilistic extension of ANSI-C [28] and PRISM models [29].

C. Compositional Probabilistic Model Checking

Another direction of research to attack the scalability problem of model checking is *compositional* techniques, where the process is subdivided into separate verification tasks for each component of the system being analysed. In non-probabilistic verification, the *assume-guarantee* paradigm has been a success. As an example, consider the problem of verifying property G on a two-component system $M_1 \| M_2$. An assume-guarantee-style approach to this would be to establish a reduction to (i) checking that, under the *assumption* that some property A holds, M_2 is guaranteed to satisfy G, denoted $\langle A \rangle\ M_2\ \langle G \rangle$; and (ii) checking that M_1 always satisfies the assumption A under any context.

In recent work [30], assume-guarantee techniques have been developed for MDPs. Building on underlying existing theories [31] for compositional reasoning about MDPs, [30] proposes a framework based on assume-guarantee queries of the form $\langle A \rangle_{\geq p_A}\ M\ \langle G \rangle_{\geq p_G}$. Here, the assumption $\langle A \rangle_{\geq p_A}$ and guarantee $\langle G \rangle_{\geq p_G}$ about the MDP M are probabilistic safety properties, represented by finite automata. Informally, this query states that "whenever M is part of a system satisfying property A with probability at least p_A, then the system will guarantee property G with probability at least p_G". Several proof rules are developed to support compositional probabilistic model checking of MDPs.

The framework of [30] is also efficient in practice. Model checking queries such as $\langle A \rangle_{\geq p_A}\ M\ \langle G \rangle_{\geq p_G}$ is reduced to multi-objective probabilistic model checking [32], which requires solution of an LP problem. These techniques are implemented in an extension of PRISM and successfully applied to several large case studies, including examples where non-compositional probabilistic model checking fails. In subsequent work [33], the process of generating an appropriate assumption $\langle A \rangle_{\geq p_A}$ is also automated, using algorithmic learning techniques based on the L* algorithm.

D. Other Advances and Directions

We also mention some other developments in probabilistic model checking that have shown promising results.

Model reduction techniques. In contrast to the abstraction techniques discussed above, which construct a smaller model with reduced precision, there exist a number of techniques to reduce a probabilistic model to a smaller, equivalent one without loss of precision. The key underlying notion is that of *bisimulation* which, roughly speaking, captures the fact that two models have the same step-wise behaviour. Well-known approaches to constructing a minimised, bisimilar model, based on a iterative splitting of a model's state space, can also be applied to probabilistic models and have been shown to be beneficial [34]. Other, more specific classes of reduction include exploiting *symmetry* [35], i.e. the existence of replication within a model, and *partial order reduction* [36], [37], which detects sequences of independent steps.

Efficient model representations. An alternative to reducing a model's size is to retain the full model, but devise efficient ways of storing and manipulating it. A particularly successful approach is the use of *symbolic* approaches, i.e. those that use data structures based on binary decision diagrams (BDDs). These provide compact representation and efficient manipulation of large, structured models by exploiting regularity from model descriptions in high-level modelling languages. Extensions of BDDs that have been successfully applied in the context of probabilistic verification include multi-terminal BDDs (MTBDDs) and matrix diagrams (see e.g. [38] for a survey). Other research directions in this vein include *disk-based* techniques, which store use slower but more plentiful hard-drive storage over RAM, and *parallelisation*, which aims to distribute the costs of storage or execution across multiple processors or machines.

Approximate probabilistic model checking. Yet another approach to avoiding the state space explosion problem is to generate *approximate* results to quantitative model checking queries based on Monte Carlo techniques, i.e. by sampling a suitably large number of simulated random paths through the model. Often referred to as *statistical probabilistic model checking*, there are two two distinct approaches in this area: *estimation* [39] and *hypothesis testing* [40]. The former attempts to estimate the result of a quantitative query such as $\mathrm{P}_{=?}[\,\phi\,]$, offering a probabilistic guarantee as to the precision of the result. The latter takes a boolean valued query such as $\mathrm{P}_{\sim p}[\,\phi\,]$ and stops generating sample paths through the models as soon as the result can be shown to be either true or false with high probability. Approximate techniques such as these, in contrast to traditional probabilistic model checking, scale to models of almost any size. They are, however, only applicable to fully probabilistic models (i.e. models without nondeterminism) and may require a large number of samples to obtain sufficiently accurate results.

Probabilistic counterexamples. Finally, we mention a direction of work that aims, not to increase the scalability of probabilistic model checking, but to improve the quality and usefulness of the results that it generates. In the non-probabilistic setting, *counterexamples* are one of the key reasons for the success of model checking. They provide, in the case where model checking shows a property to be false, evidence of this violation, typically in the form of a trace through the model. The complication in the probabilistic setting is that, to refute a property, typically a *set* of such traces is required. The key ideas were proposed in in [41], for DTMCs and PCTL properties of the form $\mathrm{P}_{\sim p}[\,\mathrm{F}^{\leq k}\,\Phi\,]$ or $\mathrm{P}_{\sim p}[\,\mathrm{F}\,\Phi\,]$. Subsequent and current research involves extending the applicability of counterexample generation (e.g. to other models such as CTMCs and MDPs, and to other types of properties, such as LTL formulae), increasing the efficiency of the generation process, and improving the way in which the information is presented to the user.

IV. Challenges

We conclude this paper by identifying some of the key future challenges for probabilistic model checking.

Real software. While modelling languages for tools such as PRISM are expressive enough for many purposes, it is clearly desirable to support probabilistic model checking directly for mainstream programming languages such as C or Java. To make progress in this direction [28], [42], developing good abstraction techniques will be essential.

Hybrid systems. Probabilistic model checking has clear potential in the domain of embedded systems, for example in quantitative verification of sensor networks or robotic applications. In this setting, the interaction of (discrete) computerised systems with their (continuous) environment becomes a crucial issue. Such *hybrid systems* (or *cyber-physical systems*) raise new challenges because they require more powerful models such as *stochastic hybrid automata*.

Synthesis. While probabilistic model checking can verify or analyse existing designs, *synthesis* has more ambitious goals, aiming to use verification techniques to contribute to the design process itself. This ranges from synthesising system parameters [43], [44] guaranteeing some property, to synthesising model components such as controllers.

Ubiquitous computing. The vision of *ubiquitous* or *pervasive* computing sees thousands of computerised devices integrating seamlessly in daily life. This emphasises the need for techniques to ensure their correctness, but also demands the development of new modelling formalisms and analysis techniques that can handle both the dynamic nature the enormous scale of these systems.

Acknowledgements. The authors are supported in part by EPSRC projects EP/D07956X and EP/F001096, EU FP7 project CONNECT and ERC Advanced Grant VERIWARE.

REFERENCES

[1] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. TACAS'06*, ser. LNCS, vol. 3920. Springer, 2006.

[2] J.-P. Katoen, E. Hahn, H. Hermanns, D. Jansen, and I. Zapreev, "The ins and outs of the probabilistic model checker MRMC," in *Proc. QEST'09*. IEEE Press, 2009.

[3] J. Kemeny, J. Snell, and A. Knapp, *Denumerable Markov Chains*. Springer, 1976.

[4] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *FAC*, vol. 6, no. 5, pp. 512–535, 1994.

[5] H. Bohnenkamp, P. van der Stok, H. Hermanns, and F. Vaandrager, "Cost-optimisation of the IPv4 Zeroconf protocol," in *Proc. IPDS'03*. IEEE, 2003.

[6] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, ser. LNCS, vol. 4486. Springer, 2007.

[7] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

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

[9] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," in *Proc. CAV'96*, ser. LNCS, vol. 1102. Springer, 1996, pp. 269–276.

[10] "PRISM web site," www.prismmodelchecker.org.

[11] H. Jensen, "Model checking probabilistic real time systems," in *Proc. 7th Nordic Workshop on Programming Theory*, 1996.

[12] D. Beauquier, "Probabilistic timed automata," *TCS*, vol. 292, no. 1, pp. 65–84, 2003.

[13] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston, "Automatic verification of real-time systems with discrete probability distributions," *TCS*, vol. 282, pp. 101–150, 2002.

[14] R. Alur and D. Dill, "A theory of timed automata," *TCS*, vol. 126, pp. 183–235, 1994.

[15] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," *FMSD*, vol. 29, pp. 33–78, 2006.

[16] A. Hartmanns and H. Hermanns, "A Modest approach to checking probabilistic timed automata," in *Proc. QEST'09*. IEEE Press, 2009.

[17] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang, "Symbolic model checking for probabilistic timed automata," *Inf. and Comp.*, vol. 205, no. 7, pp. 1027–1077, 2007.

[18] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic games for verification of probabilistic timed automata," in *Proc. FORMATS'09*, ser. LNCS, vol. 5813. Springer, 2009.

[19] Uppaal-PRO, www.cs.aau.dk/ arild/uppaal-probabilistic/.

[20] J. Berendsen, D. Jansen, and F. Vaandrager, "Fortuna: Model checking priced probabilistic timed automata," in *Proc. QEST'10*. IEEE Press, 2010, to appear.

[21] H. Hermanns, *Interactive Markov Chains and the Quest for Quantified Quality*, ser. LNCS. Springer, 2002.

[22] H. Fecher, M. Leucker, and V. Wolf, "Don't know in probabilistic systems," in *Proc. SPIN'06*, ser. LNCS, vol. 3925. Springer, 2006.

[23] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf, "Three-valued abstraction for continuous-time Markov chains," in *Proc. CAV'07*, ser. LNCS, vol. 4590. Springer, 2007.

[24] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen, "Reachability analysis of probabilistic systems by successive refinements," in *Proc. PAPM/PROBMIV'01*, ser. LNCS, vol. 2165. Springer, 2001.

[25] M. Kwiatkowska, G. Norman, and D. Parker, "Game-based abstraction for Markov decision processes," in *Proc. QEST'06*. IEEE Press, 2006.

[26] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic CEGAR," in *Proc. CAV'08*, ser. LNCS, vol. 5123. Springer, 2008.

[27] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker, "A game-based abstraction-refinement framework for Markov decision processes," *Formal Methods in System Design*, vol. 36, no. 3, 2010.

[28] ——, "Abstraction refinement for probabilistic software," in *Proc. VMCAI'09*, ser. LNCS, vol. 5403. Springer, 2009.

[29] B. Wachter and L. Zhang, "Best probabilistic transformers," in *Proc. VMCAI'10*, ser. LNCS, vol. 5944. Springer, 2010.

[30] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume guarantee verification for probabilistic systems," in *Proc. TACAS'10*, ser. LNCS, vol. 6105. Springer, 2010.

[31] R. Segala, "Modelling and verification of randomized distributed real time systems," Ph.D. dissertation, MIT, 1995.

[32] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis, "Multi-objective model checking of Markov decision processes," *LMCS*, vol. 4, no. 4, pp. 1–21, 2008.

[33] L. Feng, M. Kwiatkowska, and D. Parker, "Compositional verification of probabilistic systems using learning," in *Proc. QEST'10*. IEEE Press, 2010, to appear.

[34] J.-P. Katoen, T. Kemna, I. Zapreev, and D. Jansen, "Bisimulation minimisation mostly speeds up probabilistic model checking," in *Proc. TACAS'07*, ser. LNCS. Springer, 2007.

[35] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking," in *Proc. CAV'06*, ser. LNCS, vol. 4114. Springer, 2006.

[36] P. D'Argenio and P. Niebert, "Partial order reduction on concurrent probabilistic programs," in *QEST'04*, 2004.

[37] C. Baier, M. Groesser, and F. Ciesinski, "Partial order reduction for probabilistic systems," in *Proc. QEST'04*, 2004.

[38] A. Miner and D. Parker, "Symbolic representations and analysis of large probabilistic systems," in *Validation of Stochastic Systems: A Guide to Current Research*, ser. LNCS (Tutorial Volume), vol. 2925. Springer, 2004.

[39] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, "Approximate probabilistic model checking," in *Proc. VMCAI'04*, ser. LNCS, vol. 2937. Springer, 2004.

[40] H. Younes and R. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *Proc. CAV'02*, ser. LNCS, vol. 2404. Springer, 2002.

[41] T. Han, J.-P. Katoen, and B. Damman, "Counterexample generation in probabilistic model checking," *IEEE TSE*, vol. 35, no. 2, pp. 241–257, 209.

[42] M. Kwiatkowska, G. Norman, and D. Parker, "A framework for verification of software with time and probabilities," in *Proc. FORMATS'10*, ser. LNCS. Springer, 2010, to appear.

[43] T. Han, J.-P. Katoen, and A. Mereacre, "Approximate parameter synthesis for probabilistic time-bounded reachability," in *Proc. RTSS'08*. IEEE Press, 2008.

[44] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric Markov models," in *Proc. SPIN'09*, ser. LNCS, vol. 5578. Springer, 2009.