REVIEW ARTICLES

Performance Evaluation and Model Checking Join Forces

By Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen
Communications of the ACM, September 2010, Vol. 53 No. 9, Pages 76-85
10.1145/1810891.1810912
Comments (2)

Timeline log-scaled from 2010 backward.

Consider a major news Web site like BBC or CNN. Typically, such a site is equipped with a number of machines serving as front-ends to receive incoming requests together with some application servers such as database engines to handle these requests. When a new request arrives, to which server does the dispatcher have to route it? To the machine with the shortest queue; that is, the queue with the minimal number of outstanding requests? This might be the best decision most times, but not in cases where some of the requests in the shortest queue happen to require a very long service time, for example, because they involve very detailed queries. And what to do when the servers differ in computational capabilities? And what to do when multiple hosts have the same queue length? Well, the "join-the-shortest queue" policy might be adequate in most cases, but surely not in all. Its adequacy also depends on what quantityor measureone is interested in. This may be the mean delay of service requests, the mean queue length of waiting requests, rejection rates for waiting requests, and so on.

The effect of queue-selection policies on measures of interest or on decisions on how many servers are needed to reduce the waiting time by a given percentage, are answered by *performance evaluation* techniques. This branch of computer (system) science studies the perceived performance of systems based on an architectural system description and a workload model. Prominent techniques to obtain the aforementioned measures of interest are mathematical analysis that is typically focused on obtaining closed-form expressions, numerical evaluation that heavily relies on methods from linear algebra, and (discrete-event) simulation techniques that are based on statistical methods. The study and description of stochastic processes, most notably *Markov chains*, is pivotal for these techniques.

A complementary issue to performance is correctness. The central question is whether a system is conforming to the requirements and does not contain any flaws. Typically updates to our news Web site are queued, and it is relevant to know whether such buffers may overflow, giving rise to losingperhaps headlinenews items. Can such situations ever occur? Is there a possible scenario in which the dispatcher and application server are mutually waiting for each other, thus effectively halting the system? If such situations make the CNN news site unreachable on a presidential Election Day, this has far-reaching consequences. And what if the content of Web pages unexpectedly depend on the ordering of seemingly unrelated events in the application servers? Such "race conditions" should, if possible, be avoided.

A prominent discipline in computer science to assure the absence of errors, or, complementarily, to find errors ("bug hunting") is *formal verification*. The spectrum of key techniques in this field ranges from runtime verification, such as checking properties while executing the system, to deductive techniques such as theorem proving, to model checking. The latter is a highly automated model-based technique assessing whether a system model, that is, the possible system behavior, satisfies a property describing the desirable behavior. Typically, properties are expressed in temporal extensions of propositional logic, and system behavior is captured by Kripke structures, that is, finite-state automata with labeled states. Traditionally, such models do not incorporate quantitative information like timing or likelihoods of event occurrences.

The purpose of this article is to report on combining performance evaluation with model checking. Although these fields have been developed by different research communities in the past, over the last decade we have seen an integration of these two techniques for system analysis. Significant merits of this trend are a major increase of the applicability to real cases, and an impulse in the further development for both fields.

Back to Top

A Historic Account

To appreciate the benefits of combining performance evaluation and model checking, it is worthwhile to reflect on past and recent developments. We aim to shed light on the hidden assumptions associated with these developments. For more details on performance evaluation we refer to Bolch et al.[10] and Jain,[22] for details on model checking we refer to Baier and Katoen[7] and Clarke et al.[11]

Single queues. Performance evaluation dates back to the early 1900s, when Erlang developed models to dimension the number of required lines in analogue telephone switches, based on the calculation of call loss probabilities. In fact, he used a queueing model, in which a potentially infinite supply of customers (callers) competes for a limited set of resources (the lines). The set of models and the theory that evolved from there is known as queueing theory. It has found, through the last century, wide applicability especially in telecommunications. Characteristic for most models is the competition for a single scarce resource at a time, leading to models with a single queue.

A large variety of modeling assumptions were made, for example, regarding the number of available servers (lines), buffering facilities, scheduling strategies, job discrimination, and the timing involved. The timings were assumed to follow some continuous-time distribution, most often a negative exponential distribution, leading to (what we now call) Markovian models. These models were subsequently analyzed, using calculus, to obtain such quantities as mean number of customers queued, mean delay, sometimes even the delay distribution, or the call blocking probability ("hearing a busy signal"). Many of these measures are available in closed form; at other times, numerical recipes were proposed, for example, to derive such measures from explicit expressions in the Laplace domain. Important to note is that model construction, as well as solution, was (and still is) seen as a craft, only approachable by experts.

Networks of queues. In the late 1960s, computer networks, networked computer systems, and time-sharing computer systems came into play. These systems have the distinguishing feature that they serve a finite customer population; however, they comprise multiple resources. This led to developments in the area of queueing networks, in which customers travel through a network of queues, are served at each queue according to some scheduling discipline, and are routed to their next point of service, and so on, until returning to the party that originated the request. Efficient algorithms to evaluate networks of queues to obtain a set of "standard measures" such as mean delays, throughputs, and mean queue lengths, were developed in the 1970s.[40,51] A variety of software tools emerged supporting these algorithms that typically have a polynomial complexity in the number of queues and customers.

Stochastic Petri nets. In early 1980s, new computer architectures asked for more expressive modeling formalisms. In particular, parallel computers motivated modeling notions to spawn customers and to recombine smaller tasks into larger ones (fork/join queues). Moreover, the simultaneous use of multiple resources needed to be studied. Clearly, these concepts could not be expressed using queueing networks. This led to the proposal to extend Petri netsoriginally developed to model concurrencywith a notion of time, leading to (generalized) stochastic Petri nets (SPNs).[2] Here, the tokens can either play the role of customers or of resources. Two observations are important. First, due to the increase in expressivity, specialized algorithms, such as those available for queueing networks, are typically no longer used. Instead, the SPN models must be mapped to an underlying stochastic process, a Markov chain that is solved by numerical means. Hence, the state space of the model must be generated explicitly, and the resulting Markov chain has to be solved numerically (linear equation solvers).

The computational complexity of these state-based methods is polynomial in the number of states, but this often is, in turn, (often) a high-degree polynom in the SPN size. Secondly, as a result of the new solution trajectory, tool support became a central issue. Results achieved in this area also inspired new numerical algorithms for extended queueing network models. With hindsight, SPNs can be considered as the first "product" of the marriage between the field of performance evaluation and the field of formal modeling. In the 1990s, this trend continued and led to probabilistic variants of guarded command languages and of process algebras, the latter focusing on compositionality.

Nondeterminism. All of the models mentioned here are full stochastic models; that is, at no point in the model can some behavioral alternatives be left unspecified. For instance, the join-the-shortest-queue strategy leaves it open as to how to handle the case of several equally short queues. This choice cannot be left open with the methods noted earlier; leaving such a choice is regarded as under-specification. What typically happens is that these cases are dealt with probabilistically, for example, by assigning probabilities to the alternatives. That is, nondeterminism is seen as a problem that must be removed before analysis can take place. This is important especially for modeling formalisms as SPNs; tools supporting the evaluation of these models will either detect and report such nondeterminism through a "well-specified check" or will simply insert probabilities to resolve it. In this case, analysis is carried out under a hidden assumption, and there is no guarantee that an actual implementation will exhibit the assumed behavior, nor that the performance derived on the basis of this assumption is achieved.

Trends. The last 20 years have seen a variety of developments in performance evaluation, mostly related to specific application fields, such as the works on effective bandwidth,[42] network calculus,[46] self-similar traffic models,[47] and traffic (and mobility) models[43] (for communication network dimensioning purposes). A more general concept has been the development of fluid models to avoid the state space explosion problem (for example, Horton et al.[45]) by addressing a large denumerable state space as a single continuous state variable. Furthermore, queueing network models have been extended with layering principles to allow for the modeling

of software phenomena.[52] Finally, work on matrix geometric methods[48] has led to efficient analysis methods for large classes of queueing models.

Back to Top

Model Checking

Proof rules. The fundamental question "when and why does software not work as expected?" has been the subject of intensive research since the early days of computer science. Software quality is typically based on peer review, such as manual code inspection, extensive simulation, and testing. These rather ad hoc validation techniques have severe limitations and restrictions. Research in the field of formal verification has led to complementary methods aimed at establishing software correctness with a very high level of confidence. The origins of a sound mathematical approach toward program correctnessat a time where programs were described as flow diagramscan be traced back to Turing in the late 1940s. Early attempts to assess the correctness of computer programs were based on mathematical proof rules that allow to reason in a purely syntax-based manner. In the 1960s, these techniques were developed for sequential programs, whereas about a decade later, this approach was generalized toward concurrent programs, in particular shared-variable programs.

Temporal logic. These syntax-based approaches are based on an interpretation of programs as input/output transformers and serve to prove partial correctness (such as soundness of output values for given inputs, provided the program terminates) and termination. Thanks to a key insight in the late 1970s by Pnueli, one recognized the need for concurrent programs to not only make assertions about the starting and final state of a program, but also about the states during the computation. This led to the introduction of temporal logic in the field of formal verification.[50] Proofs, however, were still conducted mainly by hand along the syntax of programs. Proofs for programs of realistic size, though, were rather lengthy and required a good dose of human ingenuity. In the field of communication protocols, the first techniques appeared toward automated checking of elementary properties.[53]

Model checking. In the early 1980s, an alternative to using proof rules was proposed that checks systematically whether a (finite) model of a program satisfies a given property.[7,11] The pioneers Clarke, Emerson, and Sifakis, received the ACM Turing Award 2007 for this breakthrough; it was the first step toward the fully automated verification of concurrent programs. How does model checking work? Given a model of the system (the possible behavior) and a specification of the property to be considered (the desirable behavior), model checking is a technique that systematically checks the validity of the property in the model. Models are typically nondeterministic finite-state automata, consisting of a finite set of states and a set of transitions that describe how the system evolves from one state into another. These automata are usually composed of concurrent entities and are often generated from a high-level description language such as Petri nets, process algebras, Promela, or Statecharts. Properties are specified in temporal logic such as Computation Tree Logic (CTL), an extension of propositional logic that allows one to express properties that refer to the relative order of events. Statements can either be made about states or about paths, such as sequences of states that model system evolution.

The strength of model checking is not in providing a rigorous correctness proof, but rather the ability to generate diagnostic feedback in the form of counterexamples in case a property is refuted.

The backbone of the CTL model checking procedure is a recursive descent over the parse tree of the formula under consideration where temporal conditions (for example, a reachability for an invariance condition) are checked using fixed point computations. The class of path properties expressible in CTL is restricted to local conditions on the current states and its direct successors, constrained reachability conditionsis a goal state reachable by not visiting certain states before?and their duals.

More complex path properties such as repeated reachability or progress properties, which, for example, can state that whenever a request enters the news Web site, it is served eventually, can be specified in Linear Temporal Logic (LTL). The rough idea of model checking LTL specifications is to transform the formula at hand into an automaton (recognizing infinite words) and then to analyze the product of this automaton with the system model by means of graph algorithms.

The strength of model checking is not in providing a rigorous correctness proof, but rather the ability to generate diagnostic feedback in the form of counterexamples (such as error traces) in case a property is refuted. This information is highly relevant to find flaws in the model and in the real system.

Taming state space explosion. The time and space complexity of these algorithms is linear in the size of the finite-state automaton describing the system. The main problem is this size may grow exponentially in the number of program and control variables, and in the number of components in a multithreaded or distributed system.

Since the birth of model checking, effective methods have been developed to combat this state explosion problem. Prominent examples of such techniques are: symbolic data structures,[39] partial-order reduction,[49] casting model checking as SAT-problems,[38] or abstraction techniques.[11] Due to these techniques, together

with unremitting improvements of underlying algorithms and data structures and hardware technology improvements, model checking techniques that only worked for simple examples a decade ago, are now applicable to more realistic designs. State-of-the-art model checkers can handle state spaces of about 10^9 states using off-the-shelf technology. Using clever algorithms and tailored data structures, much larger state spaces (up to 10^{120} states[41]) can be handled for specific problems and reachability properties.

Quantitative aspects. From the early 1990s on, various extensions of model checking have been developed to treat aspects such as time and probabilities. Automata have been equipped with clock variables to measure the elapse of time (resulting in timed automata), and it has been shown that despite the infinite underlying state space of such automata, model checking of a timed extension of CTL is still decidable.[37] LTL has been interpreted over (discrete) probabilistic extensions of automata, focusing on the probability that an LTL formula holds, and probabilistic variants of CTL have been developed, as we will elaborate in more detail later on. For an overview, see Baier and Katoen.[7] The combination of timing aspects and probabilities started about two decades ago and is highly relevant for this article.

Various software tools have been developed that support model checking. Some well-known model checking tools are: `SPIN` for LTL, `NuSMV` for CTL (and LTL), `Uppaal` for timed CTL, and `PRISM` for probabilistic CTL.

Back to Top

Let's Join Forces

Developments in performance evaluation lean toward more complex measures of interest, and focus on more complex system behavior. However, quantitative aspects such as timing and random phenomena are becoming more important in the field of model checking. Performance evaluation and model checking have thus grown in each other's direction, simply because from either end, it was felt that the methods in isolation did not answer the questions that were at stake. Let us discuss the reasons for this, and the benefits of combining these methods.

Back to Top

Individual Shortcomings

Why is a performance (or a dependability) evaluation of a system in itself not good enough? And why is a formal verification of a system insufficient to validate its usefulness? These questions are best answered by taking a simple system design example, for instance a reliable data transmission protocol such as TCP. Such a protocol relies on a number of ingredients that, when suitably combined, result in the desired behavior: reliable, end-to-end in-order delivery of packets between communicating peers. These ingredients comprise timers, sequence numbers, retransmissions, and error-detecting codes.

A typical performance model will take into account the TCP timing and retransmission aspects, whereas the error correction will mostly be included as a random phenomenon.

For the sake of simplicity, sequence numbers are neglected, which results in a model that can be analyzed using either a closed-form formula or some numerical technique that, under the assumption the model is functionally correct, gives a certain mean performance, measured as throughput or mean packet delay. However, the obtained quantities do not say anything about the question of whether the packets do arrive correctly at all, hence, whether the protocol is correct. Conversely, a classical functional model of the sketched protocol here will most likely result in a correctness statement of the form "all packets will eventually arrive correctly." But this gives no information about perceived delays and throughputs. Needless to say, one cannot simply "add up" the results of both analyses, as they result from two differentand possibly quite unrelatedmodels.

The key challenge lies in developing an integrated model. Preferably, the user, such as the system architect or design engineer, just provides a single model (as engineering artifact) that forms the basis for both types of analysis. To improve the efficiency, additional property-dependent abstraction techniques can be applied to abstract away from all details of the model that are irrelevant for the property to be checked. For example, checking whether a purely functional property holds for a Markov model requires an analysis of the underlying graph structure, and one can ignore all stochastic information.

Back to Top

Benefits

Modeling and measure specification. An important advantage of using temporal logics (or automata) to specify properties of interestin fact guarantees on measures of interestis the possibility to describe properties at the same abstraction level as the modeling of the stochastic process. Up to now, it has been tradition to specify measures of interest such as "what is the probability to fail within deadline *d*?" at state level, that is, in terms of the states and their elementary properties (logically speaking, atomic propositions). Sometimes reward structures have been added at state level to quantify the use of resources such as queue occupancies and the like. This stands in sharp contrast with the description of the models themselves, which is mostly done using high-level modeling formalisms such as queueing networks, SPNs, stochastic automata networks, or stochastic process algebra. Temporal logics close this paradigm gap between high-level and state-based modeling as they allow to specify properties in terms of the high-level models, for example, in terms of the token distribution among places in a Petri net. By the use of temporal logics, modeling and measure specification become treated at an equal footing.

An example logic with semantics interpretation is illustrated in Figure 1. Instances of this generic logic arise by considering special types of stochastic processes, for example, for an interpretation over discrete-time Markov chains (DTMC), $T = N$ and we obtain probabilistic computation tree logic (PCTL).[18] For continuous-time Markov chains (CTMC), the time domain is $T = R$, and continuous stochastic logic (CSL) is obtained.[4,6] Figure 3 presents a small representative example[3,9] with some typical logical formulae.

Expressivity and flexibility. The use of logics offers, in addition, a high degree of expressiveness. Simple performance and dependability metrics such as transient probabilitieswhat is the probability of being in a failure state at time *t*?and long-run likelihoods (when the system is observed long enough) can readily be expressed. Most standard performance measures are easily captured, see Table 1 for a selection of properties. More importantly, the use of logics offers an enormous degree of flexibility. Nesting formulas yields a simple mechanism to specify complex measures in a succinct manner. A property like "the probability to reach a state within 25 seconds that almost surely stays safe for the next 10 seconds, via legal states only exceeds 1/2" boils down to

$$\mathbb{P}_{>\frac{1}{2}}\left(legal\, \mathcal{U}^{\leqslant 25}\, \mathrm{P}_{=1}(\Box^{\leqslant 10}\, safe)\right)$$

This immediately pinpoints another advantage: given the formal semantics of the temporal logic, the meaning of the above formula is precise. That is to say, there is no possibility that any confusion might arise about its meaning. Unambiguous measure specifications are of utmost importance. Existing mathematical measure specifications are rigorous too of course, but do not offer the flexibility and succinctness of logics. Temporal logic provides a framework that is based on just a few basic operators.

Many measures, one algorithm. The above concerns the measure specification. The main benefit though is the use of model checking as a fully algorithmic approach toward measure evaluation. Even better, it provides a single computational technique for any possible measure that can be written. This applies from simple properties to complicated, nested, and possibly hard-to-grasp formulas. For the example logic this is illustrated in Figure 2. This is radically different from common practice in performance and dependability evaluation where tailored and brand new algorithms are developed for "new" measures. One might argue that this will have a high price, that is, the computational and space complexity of the exploited algorithms must be extremely high. No! On the contrary, in the worst case, the time complexity is linear in the size of the measure specification (logic formula), and polynomial (typically of order 2 or 3, at most) in the number of states of the stochastic process under consideration. As indicated in Figure 4, the verification of bounded reachability probabilities in DTMCs and CTMCsoften the most time-consuming ones is a matter of a few seconds even for millions of states: The space complexity is quadratic in the number of states in the worst case. In fact, as for other state-based performance evaluation techniques this polynomial complexity is an issue of concern as the number of states may grow rapidly.

Perhaps the largest advantage of model checking for performance analysis is that all algorithmic details, all detailed and non-trivial numerical computation steps are hidden to the user. Without any expert knowledge on, say, numerical analysis techniques for CTMCs, measure evaluation is possible. Even better: the algorithmic analysis is measure-driven. That is to say, the stochastic process can be tailored to the measure of interest prior to any computation, avoiding the consideration of parts of the state space that are irrelevant for the property of interest. In this way, computations must be carried out only on the fragments of the state space that are relevant to the property of interest. In fact, this generalizes the ideas put forward by Sanders and Meyer on variable-driven state space generation in the late 1980s.[33]

Dependability evaluation. This measure-driven aspect is even more beneficial in the field of system dependability evaluation, a field tightly related to performance evaluation, but especially concerned with evaluating service continuity of computer systems. Questions like "under which system faults can a given service still be provided adequately?" are addressed, and typical measures of interest are system reliability and availability, as illustrated in Table 1. Since the beginning of the 1980s this field has matured significantly, due to the introduction of state-oriented models and the invention of uniformization.[44] This facilitated the efficient analysis of time-dependent properties such as reliability or availability evaluation, in combination with high-level model specification techniques such as SPNs. The models that one could analyze now went well above the "standard models" based on reliability block diagrams or fault-trees.

The measures of interest in this field often involve costs, modeling the usage of resources. Extensions of stochastic processes with cost (or reward) functions give rise to a logic where in addition to, for example, time bounds, conditions about the accumulated reward along an execution path can be imposed. Model checking still goes along the lines of Figure 2, but involves computational procedures that are more time-consuming.

One for free. Is that all? Not quite. An important problem with performance modeling regardless of whether one aims at numerical evaluation or at simulation, is to check the functional correctness of the model. For a stochastic Petri net specification, place and transition invariants are exploited to check for deadlocks and liveness, among others.

For a Markov chain model, graph-based algorithms are used to check elementary properties. The good news is when employing model checking we get this functionality for free. Using the same machinery for validating the measures of interest, functional properties can be checked. Probabilistic model checking provides two for the price of one: both performance/dependability analysis and checking functional properties. This forces the user to construct models with a high precision as any tiny inconsistency will be detected. Compare this to simulation model construction in NS2 or OPNET!

Nondeterminism. Sometimes this need for precision might seem as a burden, but it is a vehicle to force the modeler to make hidden assumptions explicitor to leave them out. For instance, we have discussed the nondeterminism inherent in the join-the-shortest-queue idea, whichun-less made concreteimplies that the underlying model is not a stochastic process. Stochastic models with non-determinism are usually referred to as stochastic decision processes. In these models the future behavior is not always determined by a unique probability distribution, but by selecting one from a set of them. Temporal logics and verification technology have been extended to this type of models with relative ease for CTL[8] and LTL.[14,36] In fact, they constitute the genuine supermodel that comprises both the model checking and performance evaluation side as special cases: When transition systems are paired with Markov chains or Markov reward models, the model is known as Markov decision processes. Here, performance model checking is still possible, but the checker now computes bounds on the performance, in the sense that however the nondeterminism is concretized, the concrete performance figure will stay within the calculated bounds. Whereas for the discrete time setting, efficient model checking algorithms have been developed, this field is still relatively open in the continuous-time setting.

Back to Top

Appealing Application Areas

Several stochastic model checking tools have been developed since 2005, of which PRISM[20] is by far the most widely used. A number of well-known tools from the performance and dependability evaluation area, like tools for SPNs and stochastic process algebras, have been extended with stochastic model checking features. All these tools automatically generate a Markovian model of some sort, either using symbolic or sparse data structures.

With these tools, a wide variety of case studies have been carried out, amongst others, in application areas such as communication systems and protocols, embedded systems, systems biology, hardware design, and security, as well as more "classical" performance and dependability studies.

Examples of the latter category, for which CTMCs are a very natural model, include the analysis of various classes of traditional queuing networks and even infinite-state variants thereof, fault-tolerant workstation clusters, and wireless access protocols such as IEEE 802.11. Also system survivability, that is the ability of a system (for example, military or aircraft) to recover predefined service levels in a timely manner after the occurrence of disasters, has been precisely captured using a logic similar to that introduced before, and has been verified for Google-like file systems.[12] The evaluation of a wireless access protocol for ad hoc networks using model checking could be carried out at far lower cost than using discrete-event simulations.[32]

The popularity of Markovian models is rapidly growing due to their application potential in systems biology; the timing and probabilistic nature of CTMCs naturally reflect the operations of biological mechanisms such as molecular reactions. In fact, various biological systems have been studied by CTMC model checking in recent years.[26] Prominent examples include ribosome kinetics, signaling pathways, cell cycle control in Eukaryotes, and enzyme-catalyzed substrate conversion. In particular, the possibility to compute time-bounded reachability probabilities is of great importance here as traditional studies focus on steady-state behavior.

Another application area for CTMC model checking is embedded systems where the timeliness of communication between sensor and actuator devices, for example, within cars or between high-speed trains, is of utmost importance. Stochastic model checking techniques allow us to address the timeliness and the protocols' correctness from a single model. One example is dynamic power management in relation to job scheduling.[31]

Examples for the discrete-time setting include several studies of the IPv4 Zeroconf protocol are illustrated in Figure 3, where next to the probability of eventually obtaining an unused IP address, extensions have been studied with costs, addressing issues such as the number of attempts needed to obtain such address. Security protocols are another important class of systems in which discrete randomness is exploited, for example, by applying random routing to avoid information leakage. An interesting case is the Crowds protocol,[34] a well-known security protocol that aims to hide the identity of Web-browsing stations. Checking Markovian models with up to 10^7 states did provide important information on quantifying the increase of confidence of an adversary when observing an Internet packet of the same sender more than once. A novel case study in the field of nano-technology applies stochastic model checking to quantify the reliability of a molecular switch with increasing memory array sizes.[13] Other natural cases for discrete-time probabilistic models are randomized protocolsin which probabilities are used to break tiessuch as consensus and broadcast protocols, and medium access mechanisms such as Zigbee.

An important problem with performance modeling regardless whether one aims at numerical evaluation or at simulation, is to check the functional correctness of the model.

To conclude, an interesting case study using DTMCs with non-determinism is the analysis of the Firewire protocol (IEEE 1394). This protocol has been developed to allow "plug-and-play" network connectivity for

multimedia consumer electronics in the home environment. A key component in IEEE 1394 is a leader election protocol (the "root contention protocol") that exploits a coin-tossing mechanism to break ties. Stochastic model checking revealed that using a biased coin instead of the typically used unbiased coin, speeds up the leader election process. This confirmed a conjecture in Stoelinga.[35] This insight would not have been found through "classical" qualitative verification.

Back to Top

Current Trends and Challenges

Edmund M. Clarke, a co-recipient of the 2007 ACM A.M. Turing Award, points out that probabilistic model checking is one of the brands of verification that requires further developments.[41] Here, we note some of the current trends and major research challenges.

One of the major practical obstacles shared by model-based performance evaluation and model checking is the state space explosion problem. To combat the state space explosion problem, various techniques have been developed and successfully applied for model checking Kripke structures[11] (and the literature mentioned there).

For stochastic models the state space explosion problem is even more severe. This is rooted in the fact that the model checking algorithms for stochastic models rely on a combination of model checking techniques for non-stochastic systems, such as graph algorithms, but also mathematical, often numerical methods for calculating probabilities, such as linear equation solving or linear programming.

Many of the advanced techniques for very large non-stochastic models have been adapted to treat stochastic systems, including variations of decision diagrams to represent large state spaces symbolically.[30] Complementary techniques attempt to abstract from irrelevant or redundant details in the model and to replace the model with a smaller, but "equivalent" one. Some of them rely on the concept of lumpability for stochastic processes, which in the formal verification setting is known as bisimulation quotienting, and where states with the same probabilistic behavior are collapsed into a single representative.[16,27]

Other advanced techniques to fight the state-explosion problem include symmetry exploitation,[24] partial order reduction,[5] or some form of abstraction[28], possibly combined with automatic refinement.[15,19] All these approaches take inspiration in classical model checking advances, which often get much more intricate to realize, and raise interesting theoretical and practical challenges. All together, they have advanced the field considerably in the ability to handle cases as the ones discussed earlier.

An important feature of model checkers for non-stochastic systems is the generation of counterexamples for properties that have been refuted by the model checker. The principal situation is more difficult in the stochastic setting, as for probabilistic properties, say the requirement that a certain un-desired event will appear with probability at most 10^3, single error traces are not adequate. The generation and representation of counterexamples is therefore a topic of much increasing attention[17,29] within the community.

To overcome the limitation to finite state spaces, much work has been done to treat infinite-state probabilistic systems, in many different flavors.[1,23]

Another topic of ongoing interest lies in combining probabilistic behavior with continuous dynamics as in timed[25] or hybrid automata, but more work on the tool side is needed to assess the merits of these approaches faithfully. Theorem-proving techniques for analyzing probabilistic systems[21] are also a very promising direction. One of the major open technical problems is the treatment of models with nondeterminism and continuous distributions. Initial results are interesting but typically subject to (severe) restrictions.

As a final item, we mention the need to tailor the general-purpose probabilistic model checking techniques to special application areas. This covers the design of special modeling languages and logics that extend or adapt classical modeling languages and temporal logics by adding features that are specific for the application area.

Back to Top

Acknowledgments

We thank Andrea Bobbio, Gianfranco Ciardo, William Knottenbelt, Marta Kwiatkowska, Evgenia Smirni, and the anonymous reviewers for their valuable feedback.

Back to Top

References

Due to space limitations, a comprehensive list of all references cited in this article can be found at the authors' Web sites.

1. Abdulla, P., Bertrand, N., Rabinovich, A. and Schnoebelen, P. Verification of probabilistic systems with faulty communication. *Inf. and Comp. 202*, 2 (2007), 141165.

2. Ajmone Marsan, M., Conte, G., Balbo, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst. 2*, 2 (1984), 93122.

3. Andova, S., Hermanns, H., and Katoen, J.-P. Discrete-time rewards model-checked. *FORMATS, LNCS 2791*, (2003), 88104.

4. Aziz, A., Sanwal, K., Singhal, V. and Brayton, R.K. Model checking continuous-time Markov chains. *ACM TOCL 1*, 1 (2000), 162170.

5. Baier, C., Größer, M., and Ciesinski, F. Partial order reduction for probabilistic systems. *Quantitative Evaluation of Systems*. IEEE CS Press, 2004, 230239.

6. Baier, C., Haverkort, B.R., Hermanns, H., and Katoen, J-P. Model checking algorithms for continuous-time Markov chains. *IEEE TSE 29*, 6 (2003), 524541.

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

8. Bianco, A. and De Alfaro, L. Model checking of probabilistic and non-deterministic systems. *Foundations of Softw. Technology and Theor. Comp. Science. LNCS 1026* (1995), 499513.

9. Bohnenkamp, H., van der Stok, P., Hermanns, H., and Vaandrager, F.W. Cost optimisation of the IPv4 zeroconf protocol. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks*. IEEE CS Press. 2003, 531540.

10. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S. *Queueing Networks and Markov Chains*. Wiley Press, 1998.

11. Clarke, E.M., Grumberg, O., and Peled, D. *Model Checking*. MIT Press, 1999.

12. Cloth, L. and Haverkort, B.R. Model checking for survivability. *Quantitative Evaluation of Systems*. IEEE CS Press (2005), 145154.

13. Coker, A., Taylor, V., Bhaduri, D., Shukla, S., Raychowdhury, A., and Roy, K. Multijunction fault-tolerance architecture for nanoscale crossbar memories. *IEEE Trans. on Nanotechnology 7*, 2 (2008), 202208.

14. Courcoubetis, C. and Yannakakis, M. The complexity of probabilistic verification. *JACM 42*, 4 (1995), 857907.

15. D'Argenio, P.R. Jeannet, B., Jensen, H., and Larsen, K.G. Reduction and refinement strategies for probabilistic analysis. *LNCS 2399* (2002), 335372.

16. Derisavi, S., Hermanns, H., and Sanders, W.H. Optimal state-space lumping in Markov chains. *Inf. Proc. Letters 87*, 6 (2003), 309315.

17. Han, Y., Katoen, J.-P. and Damman, B. Counter-examples in probabilistic model checking. *IEEE TSE 36*, 3 (2010), 390408.

18. Hansson, H. and Jonsson, B. A logic for reasoning about time and reliability. *Formal Aspects of Comp. 6*, 5 (1994), 512535.

19. Hermanns, H., Wachter, B., and Zhang, L. Probabilistic CEGAR. *Computer-Aided Verification LNCS 5123* (2008), 162175.

20. www.prismmodelchecker.org.

21. Hurd, J., McIver, A., and Morgan, C. Probabilistic guarded commands mechanized in HOL. *Theor. Comp. Sc. 346*, 1 (2005), 96112.

22. Jain, R. *The Art of Computer System Performance Analysis*. Wiley, 1991.

23. Kucera, A., Esparza, J., and Mayr, R. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science 2*, 1 (2006).

24. Kwiatkowska, M.Z., Norman, G., and Parker, D. Symmetry reduction for probabilistic model checking. *Computer-Aided Verification LNCS 4144* (2008), 238248.

25. Kwiatkowska, M.Z., Norman, G., Parker, D., and Sproston, J. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design 29*, 11 (2006), 3378.

26. Kwiatkowska, M.Z., Norman, G., and Parker, D. Probabilistic model checking for systems biology. *Symbolic Systems Biology*, 2010.

27. Larsen, K.G. and Skou, A. Bisimulation through probabilistic testing. *Inf. & Comp., 94*, 1 (1989), 128.

28. McIver, A. and Morgan, C. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

29. McIver, A., Morgan, C., and Gonzalia, C. Proofs and refutation for probabilistic systems. *Formal Methods LNCS 5014* (2008), 100115.

30. Miner, A. and Parker, D. Symbolic representation and analysis of large probabilistic systems. *Validation of Stochastic Systems. A Guide to Current Research. LNCS 2925* (2005), 296338.

31. Norman, G., Parker, D., Kwiatkowska, M.Z. Shukla, S.K., Gupta, R. Using probabilistic model checking for dynamic power management. *Formal Asp. Comp. 17*, 2 (2005), 160176.

32. Remke, A., Haverkort, B.R. and Cloth, L. A versatile infinite-state Markov reward model to study bottlenecks in 2-hop ad hoc networks. Quantitative Evaluation of Systems *IEEE CS Press*, 2006, 6372.

33. Sanders, W.H. and Meyer, J.F. Reduced base model construction methods for stochastic activity networks. *IEEE J. on Selected Areas in Comms. 9*, 1 (1991), 2536.

34. Shmatikov, V. Probabilistic model checking of an anonymity system. *J. Computer Security 12*, (2004) 355377.

35. Stoelinga, M. Fun with FireWire: A comparative study of formal verification methods applied to the IEEE 1394 root contention protocol. *Formal Asp. Comp., 14*, 3 (2003), 328337.

36. Vardi, M.Y. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th IEEE Symp. on Foundations of Comp. Science*. IEEE CS Press (1985), 327338.

Back to Top

Authors

Christel Baier (baier@tcs.inf.tu-dresden.de) is a professor at TU Dresden, Germany.

Boudewijn R. Haverkort (brh@cs.utwente.nl) is a professor at the University of Twente, and scientific director of the Embedded Systems Institute, Eindhoven, The Netherlands.

Holger Hermanns (hermanns@cs.uni-sb.de) is a professor at Saarland University, Saarbrücken, Germany.

Joost-Pieter Katoen (katoen@cs.rwth-aachen.de) is a professor at RWTH Aachen University, Aachen, Germany.

Back to Top

Footnotes

DOI: http://doi.acm.org/10.1145/1810891.1810912

Back to Top

Figures

Figure 1. A logic for quantitative properties: syntax and semantics.

Figure 2. Schema for model checking stochastic processes.

Figure 3. A simple model checking example: The Zeroconf protocol.

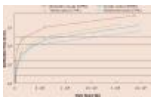

Figure 4. Efficiency of computing reachability probabilities versus the state space size.

Figure. Timeline log-scaled from 2010 backward.

Back to Top

Tables

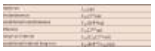

Table 1. Availability measures and their logical specification.

Back to Top

Sidebar: key insights

Performance engineers and verification engineers are currently facing very similar modeling and analysis challenges.

A joint consideration is possible, practical, beneficial, and is supported by effective tools.

Quantitative model checkers are applicable to a broad spectrum of applications ranging from sensor networks to security and systems biology.

Comments

Anonymous

June 15, 2011 11:38

Nice review but it seems that none of the authors were keen to put the full list of the references on his/her website.

Anonymous

June 16, 2011 03:56

The full list of the references can be founded in http://www-i2.informatik.rwth-aachen.de/pub/index.php?type=download&pub_id=423&location=cacm2009.pdf-52c412b7294c6a6cbf99157bbcbf0a5c.pdf.

Displaying **all 2** comments