Model Checking meets Performance Evaluation[1]

Christel Baier [2], Boudewijn R. Haverkort [3], Holger Hermanns [4], Joost-Pieter Katoen [5]

Abstract

Markov chains are one of the most popular models for the evaluation of performance and dependability of information processing systems. To obtain performance measures, typically long-run or transient state probabilities of Markov chains are determined. Sometimes the Markov chain at hand is equipped with rewards and computations involve determining long-run or instantaneous reward probabilities.

This note summarises a technique to determine performance and dependability *guarantees* of Markov chains. Given a precise description of the desired guarantee, all states in the Markov chain are determined that surely meet the guarantee. This is done in a fully automated way. Guarantees are described using logics. The use of logics yields an expressive framework that allows to express well-known measures, but also (new) intricate and complex performance guarantees. The power of this technique is that no matter how complex the logical guarantee, it is *automatically* checked which states in the Markov chain satisfy it. Neither manual manipulations of Markov chains (or their high-level descriptions) are needed, nor the knowledge of any numerical technique to analyze them efficiently. This applies to any (time-homogeneous) Markov chain of any structure specified in any high-level formalism.

1 Logical performance guarantees

Let us first explain how performance guarantees are specified using a temporal extension of logics. These logics are intensively used for specifying functional properties of systems (such as absence of deadlocks or reachability). We discuss by example their usage for specifying performance guarantees over discrete-time, continuous-time and Markov reward models. The logic used for the discrete-time case is due to Hansson & Jonsson [18].

1.1 Discrete-time Markov chains

Consider a discrete-time Markov chain with many states, millions say, among which some illegal (or: forbidden) states and some goal states. In a multi-processor system with failure-prone components, a illegal state could, for instance, characterise a configuration in which more than a certain number of components has failed. A goal state could be a configuration in which a majority of the components is operational. Suppose we are interested in determining the Markov chain states from which a goal state may be reached with a high probability, say at least 0.92, while never visiting an illegal state before reaching its goal. We thus consider scenarios in which the system starts in some state, visits any number of states that are not illegal, while finally ending up in some goal state. The requirement is formulated by the logical formula:

$$\mathcal{P}_{\geqslant 0.92}\,(\neg\, \mathit{illegal}\ \mathsf{until}\ \mathit{goal}) \quad . \tag{1}$$

The subscript in this formula describes the bound on the probability that we want to establish, while the part between parentheses is characterizing the set of *paths* of interest. A path in a Markov chain is simply a traversal (a sequence of states) through the graph underlying it, i.e., the graph obtained by removing the transition probabilities. The path $s_0\, s_1\, s_2 \ldots$ that starts in state s_0 satisfies the formula $(\neg\, \mathit{illegal}\ \mathsf{Until}\ \mathit{goal})$ if and only if there is some goal state s_j in the path ($j \geqslant 0$) while all states s_i (with $i < j$) prior to s_j are not illegal. The probability of a path is simply the product of the transition probabilities, i.e., $\mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots$. A state s now satisfies the guarantee (1) if at least 92% of the paths starting in s satisfy $(\neg\, \mathit{illegal}\ \mathsf{Until}\ \mathit{goal})$.

If, in addition, the number of steps to reach a goal state is limited, say at most 137, then this additional constraint on the number of steps can easily be accommodated in formula (1):

$$\mathcal{P}_{\geqslant 0.92}\left(\neg\, \mathit{illegal}\ \mathsf{Until}^{\leqslant 137}\ \mathit{goal}\right) \quad .$$

The difference with formula (1) is that now the part between parentheses is characterizing the set of paths that reach a goal state via only visiting non-illegal states, and do so in at most 137 jumps.

Note that transient probabilities are "easy" variants of these formulae. For instance, a state satisfies the guarantee

$$\mathcal{P}_{\geqslant 0.92}\left(\mathtt{true}\ \mathsf{Until}^{=137}\ \mathit{goal}\right) \quad , \tag{2}$$

if the probability to be in a goal state after exactly 137 jumps is at least 0.92. Whereas in (1) the part $\neg\, \mathit{illegal}$ forbids to visit an illegal state, `true` allows any state to be visited, and thus puts no constraint on the states visited prior to reaching the goal state. As the construction `true` Until occurs frequently, this is also abbreviated by $\Diamond$ (pronounce "eventually"). The previous formula can thus equivalently be written as:

$$\mathcal{P}_{\geqslant 0.92}\left(\Diamond^{=137}\ \mathit{goal}\right)$$

[1] This work has been supported by the Dutch and German research council as part of their bilateral cooperation program.

[2] Institut für Informatik I, University of Bonn, Germany.
[3] Dept. of Comp. Science, Univ. of Twente, The Netherlands.
[4] Dept. of Comp. Science, Saarland University, Germany.
[5] Dept. of Comp. Science, RWTH Aachen University, Germany.

The dual of $\Diamond$ is $\Box$ (pronounce "always"), e.g., the formula $\Box \mathit{goal}$ is equivalent to $\neg \Diamond \neg \mathit{goal}$. In the setting of Markov chains we have:

$$\mathcal{P}_{\geqslant 0.92}\left(\Box \mathit{goal}\right) \;\equiv\; \mathcal{P}_{\leqslant 0.08}\left(\Diamond \neg \mathit{goal}\right)$$

Until so far, we have implicitly assumed that we could characterise the illegal and goal states by the simple predicates *illegal* and *goal*, respectively. In many practical situations, though, the characterisation of such states is unfortunately not so straightforwar but may depend on their probabilistic evolution. For example, the goal states could be configurations from which it is guaranteed that the system remains operational with the majority of the components (not necessarily the same ones!) during the next 13 jumps with an extremely high probability (0.9999, say). By using nesting of formulae, (1) is now generalised into

$$\mathcal{P}_{\geqslant 0.92}\left(\neg \mathit{illegal}\ \mathsf{Until}\ \mathcal{P}_{\geqslant 0.9999}\left(\Box^{=13}\ \mathit{goal}\right)\right)$$

Stating this more involved property in natural language may easily lead to ambiguities and misinterpretations.

As a variation on the latter performance guarantee, suppose that we are interested in quickly initialising the system (within 10 steps, say) such that a set of states is reached that guarantee the system to be operational with a majority of its components on the long run. That is, we want to quickly reach states that, when starting from there, the system is almost entirely operational in, 99.99 % of the cases, say, once the system is considered on the long run. As before, we do not allow visiting illegal states. This is expressed as:

$$\mathcal{P}_{\geqslant 0.92}\left(\neg \mathit{illegal}\ \mathsf{Until}^{\leqslant 10}\, \mathcal{L}_{\geqslant 0.9999}\left(\mathit{goal}\right)\right)$$

The new operator $\mathcal{L}$ refers—as opposed to $\mathcal{P}$ that refers to transient measures—to long-run measures. The formula $\mathcal{L}_{\geqslant 0.9999}\left(\mathit{goal}\right)$ holds in state s whenever the Markov chain when starting from s can guarantee to be in a goal state with probability at least 0.9999 on the long run. For an aperiodic Markov chain this means that the chain is in a goal state (with probability 0.9999) whenever it has become stationary (when starting from s).

As a last variation, we might want to characterise the states that when starting from either of them, the system guarantees on the long run that an illegal state can (almost surely) never be reached in the next 15 jumps. This is expressed as:

$$\mathcal{L}_{\geqslant 0.9999}\left(\mathcal{P}_{\geqslant 1}\left(\Diamond^{\leqslant 15} \neg \mathit{illegal}\right)\right)$$

Now that we have provided some flavor of defining performance guarantees on DTMCs in a logical way, let's consider the continuous-time setting.

1.2 Continuous-time Markov chains

The modular and flexible way in which performance guarantees can be specified for discrete-time Markov chains, also applies to CTMCs [10]. To put it shortly, all operators that we have seen so far can be used in the continuous-time setting as well, with the extra possibility that we can now also refer to *continuous time*. For instance,

$$\mathcal{P}_{\geqslant 0.92}\left(\neg \mathit{illegal}\ \mathsf{Until}^{\leqslant 133.4}\ \mathit{goal}\right)$$

identifies those states from which a goal state can be reached within 133.4 time units (while only visiting non-illegal states prior to that). Like before, the part between parentheses characterizes a set of paths. But a path is no longer a simple sequence of states $s_0\, s_1\, s_2 \ldots$, but an alternating sequence $s_0\, t_0\, s_1\, t_1\, s_2\, t_2 \ldots$ where t_j denotes the residence time in state s_j. A timed path satisfies $\neg\, \mathit{illegal}\ \mathsf{Until}^{\leqslant 133.4}\ \mathit{goal}$ if there exists an index j such that s_j is a goal state, all states prior to that on the path are not illegal, and s_j is reached within 133.4 time units, i.e., $t_0 + t_1 + \ldots t_{j-1} \leqslant 133.4$ and $t_0 + t_1 + \ldots t_j > 133.4$.

If we are interested in reaching a goal state in the interval $[61.1, 133.4]$ the above formula can easily be adapted to:

$$\mathcal{P}_{\geqslant 0.92}\left(\neg \mathit{illegal}\ \mathsf{Until}^{[61.1,133.4]}\ \mathit{goal}\right)$$

This guarantee states that one should visit non-illegal states prior to reaching a goal state, which should be reached between 61.1 and 133.4 time units.

1.3 Markov reward models

Various performance and dependability measures are obtained by equipping Markov models with cost or benefit information. These structures are known as *rewards*. In the continuous-time setting, they come in two flavors: impulse rewards are attached to transitions, while state rewards are associated with states. On taking a transition from state s to s' with impulse reward $r(s, s')$ (for convenience, let all rewards be non-negative), a reward $r(s, s')$ is earned. On visiting state s with reward $r(s)$ for t time units, a reward $r(s) \cdot t$ is earned.[1] In the multiprocessor system referred to before, a state reward could indicate the number of jobs processed per time unit, while an impulse reward could indicate the loss of a number of jobs when a component fails. The accumulated reward plotted versus the elapsed time is thus a piecewise continuous function, where discontinuities occur at transition epochs, and slopes of the continuous fragments are determined by the state rewards.

The logical framework can be extended with accumulated rewards in a rather straightforward manner [5]. To that end, the Until-operator is equipped with an extra parameter, indicated as a subscript, that puts a constraint on the total accumulated reward. For instance,

$$\mathcal{P}_{\geqslant 0.92}\left(\neg \mathit{illegal}\ \mathsf{Until}^{\leqslant 133.4}_{\leqslant 62}\ \mathit{goal}\right)$$

identifies those states from which a goal state can be reached within 133.4 time units (while only visiting non-illegal states

[1] In discrete-time stochastic processes the distinction between impulse and state rewards is irrelevant.

prior to that) while the accumulated reward does not exceed 62. Note the similarity with the previous formula: only the constraint on the accumulated reward has been added. In order to decide whether a timed path satisfies a formula like $\neg$ *illegal* $\mathsf{Until}^{\leqslant 133.4}_{\leqslant 62}$ *goal*, the accumulated reward of a path needs to be determined. For timed path $s_0\, t_0\, s_1\, t_1\, s_2\, t_2 \ldots$ this equals

$$r(s_0){\cdot}t_0 + r(s_0, s_1) + r(s_1){\cdot}t_1 + r(s_1, s_2) + \ldots$$

1.4 Couldn't we specify these guarantees before?

One could argue that the logical framework introduced so far is nothing else than an alternative formalism to write down (requirements on) performance and dependability measures. And, of course, one could use mathematics to precisely specify the measure-of-interest, being it a transient, steady-state, or reward one. This is a point, but not the point we want to emphasize. True, the logical framework—as mathematics—allows one to specify measures in an unambiguous manner. The main distinguishing feature, however, is not this preciseness (although this is essential), but its flexibility and conciseness. Besides, a single (just one!) algorithm suffices to check these guarantees. This is explained in the following section.

2 Analysis Algorithms

Suppose we are confronted with a (discrete-time or continuous-time) Markov chain originating from some high-level formalism such as a stochastic Petri net, a stochastic activity network or a Markovian queueing network, and a performance guarantee formulated in the logical way explained above. How does one compute the set of states satisfying this guarantee? The basic computational procedure is a simple *recursive descent* over the logical formula. This means basically that the formula is broken down into its sub-formulae, that the computation starts with the most simple sub-formulae, and once this step is completed, considers the one-but most-simple formulae, and so on, until the entire formula is captured. Considering the parse tree of the formula, this computation is just a bottom-up traversal over the parse tree, where at each node (representing a sub-formula) a single algorithm is invoked. In this way, formulae of arbitrary complexity can be treated in a uniform manner. This recursive descent mechanism is adopted from *model-checking* algorithms [13].

The main difference with traditional model-checking algorithms where all computations involve graph algorithms, fixed-point computations and the like, is that in our setting *numerical* algorithms are needed to reason about the probabilities and (in case of CTMCs) real-time aspects. To achieve this, well-known techniques for solving systems of linear equations, determining long-run probabilities, and transient probabilities (e.g., uniformization) are embedded in the tree traversal as sub-routines.

Let's explain the computational procedure in a bit more detail by means of an example. Consider the formula:

$$\mathcal{P}_{\geqslant 0.92}\left(\neg\, \textit{illegal}\ \mathsf{Until}^{\leqslant 11.2}\ \mathcal{L}_{\geqslant 0.9999}\,(\textit{goal})\right)$$

in a continuous-time setting, i.e., the superscript $\leqslant 11.2$ refers to the amount of time passed rather than the number of discrete jumps taken. We are thus interested in computing the states in a CTMC that can reach certain states within 11.2 time units (without ever visiting a non-illegal state) with at least probability 0.92. The states that need to be reached should guarantee that when starting from there, the system—when in equilibrium—is in a goal state at least 99% percent of time. Before continuing the explanation of our computational procedure, we like you to take a few moments to consider how to determine the required states.

Once you are convinced of the fact that it is not easy to determine the required states, let's start with explaining our algorithm. The above formula has the following sub-formula: *goal*, *illegal*, $\neg$*illegal*, $\mathcal{L}_{\geqslant 0.9999}\,(\textit{goal})$, and, of course, the entire formula itself.

The computation starts with the most simple sub-formulae, i.e., *goal* and *illegal*. As these are the most elementary formulae of our logical framework, it is assumed that their validity in any state of the model can directly be determined. In case of a stochastic Petri net, for instance, *illegal* and *goal* states could be states with a certain number of tokens in a certain place, whereas in a Markovian queueing network these states could refer to the number of customers in certain queues. The computation of the set of states satisfying *goal*, indicated as $\mathsf{Sat}(\textit{goal})$, is therefore straightforward. The same applies to computing $\mathsf{Sat}(\textit{illegal})$. The parse tree traversal proceeds by considering the one-but-most-simple sub-formulae, i.e., $\neg$ *illegal*. This set is simply obtained by complementing $\mathsf{Sat}(\textit{illegal})$, i.e.,

$$\mathsf{Sat}(\neg\, \textit{illegal}) = S - \mathsf{Sat}(\textit{illegal}) \quad ,$$

where S is the entire set of states in the Markov model under consideration.

The next step is to compute the states satisfying $\mathcal{L}_{\geqslant 0.9999}\,(\textit{goal})$, that is, the set of states from which the system can be started and that guarantee the system to be in a goal state with at least 99.9% fraction of time, when in equilibrium. As we follow a recursive descent procedure, the set $\mathsf{Sat}(\textit{goal})$ has already been computed. In case the CTMC is strongly connected, it suffices to compute the steady-state probabilities by standard means (e.g., Power method, Gauß-Seidel, or the like), to sum up all these probabilities for states in $\mathsf{Sat}(\textit{goal})$, and to check whether this sum is at least 0.9999 or not. In the former case, all states satisfy $\mathcal{L}_{\geqslant 0.9999}\,(\textit{goal})$, otherwise none of the states do. This is the simple case, in which the CTMC is strongly connected, and for which the initial probability distribution is not of any relevance for determining the steady-state probabilities. The more interesting case appears when the CTMC is not

strongly connected. In this case, the steady-state probabilities depend on the initial state—certain states may even not be reachable depending on where we start. By a graph analysis, basically a depth-first traversal through the Markov chain, the strongly connected components (SCCs) are determined that are *terminal*. A terminal strongly connected component is a component in which each state can reach any other state in it, but no other state. Once such component is reached it can thus not be left anymore; one can only cycle through that component, but never escape. For each such component, the steady-state probabilities are determined by solving a system of linear equations. For terminal SCC B let $\pi_B(\mathit{goal})$ the steady-state probability for being in a goal state in B in equilibrium. In order now to determine whether

$$s \in \mathsf{Sat}\left(\mathcal{L}_{\geqslant 0.9999}\left(\mathit{goal}\right)\right)$$

we determine the probability $p_s(B)$ to reach the terminal SCC B from state s. This is done for all terminal SCCs and can be computed by solving the following system of linear equations:

$$p_s(B) = \begin{cases} 1 & \text{if } s \in B \\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot p_{s'}(B) & \text{otherwise} \end{cases}$$

Gathering these results, we obtain that $s \in \mathsf{Sat}\left(\mathcal{L}_{\geqslant 0.9999}\left(\mathit{goal}\right)\right)$ if and only if

$$p_s(B_1) \cdot \pi_{B_1}(\mathit{goal}) + \ldots p_s(B_k) \cdot \pi_{B_k}(\mathit{goal}) \geqslant 0.9999$$

where $k > 0$ is the number of terminal SCCs in the Markov chain.

As a result from the previous computational steps we have the following sets of states at our disposal:

$$L = \mathsf{Sat}(\neg\, \mathit{illegal}) \text{ and } G = \mathsf{Sat}\left(\mathcal{L}_{\geqslant 0.9999}\left(\mathit{goal}\right)\right)$$

Let $[G]$ be the characteristic function of G, i.e., for state s we have $[G](s) = 1$ if $s \in G$, and $[G](s) = 0$, otherwise. Similarly, $[L]$ is the characteristic function of L. As a final step in the verification process we now determine the set of states satisfying the formula:

$$\mathcal{P}_{\geqslant 0.92}\left(\underbrace{\neg\, \mathit{illegal}}_{[L]}\ \mathsf{Until}^{\leqslant 11.2}\ \underbrace{\mathcal{L}_{\geqslant 0.9999}\left(\mathit{goal}\right)}_{[G]}\right) \qquad (3)$$

This is done as follows. Consider a path $s_0\, s_1\, s_2\, \ldots$ through the CTMC. (For simplicity, the state residence times are omitted.) Once a state in G has been reached (via only states in L), it is of no relevance whatsoever to know which states will be visited afterwards. That is, when $s_k \in G$ and all preceding states $s_i \in L$ (with $i < k$), the fact whether later states s_j (for $j > k$) are in L or G does not matter for the path to satisfy $[L]\ \mathsf{Until}\ [G]$. It suffices to treat $s_k \in G$ as an *absorbing* state. This applies to all states in G. Therefore, as a first step we make all states in G in the CTMC absorbing. We do the same with all states that are neither in L nor in G. This is justified by the fact that once such state is reached, it is certain that the path can never fulfill the formula $[L]\ \mathsf{Until}\ [G]$, whatever states are visited afterwards. So, prior to doing any computational step, the CTMC M is changed into M' by making all states in G and all states in $S - (L \cup G)$ absorbing. The number of reachable states in M' is never larger than than in M. It is not difficult to see that state s in M satisfies (3) if and only if it satisfies

$$\mathcal{P}_{\geqslant 0.92}\left(\mathtt{true}\ \mathsf{Until}^{=11.2}\ [G]\right)$$

in the newly obtained CTMC M'. What does this bring us? Well, compare this formula with the one for transient measures, cf. equation (2). The shapes are very similar, and indeed we have transformed the verification of formula (3) into a (standard) transient measure calculation on another CTMC. Thus, we are left with determining the probability vector $\pi_s(11.2)$ (in M') that indicates the probability to be in a certain state after exactly 11.2 time units when starting in state s. This can be done using standard means such as uniformization. Now, it follows that

$$s \in \mathsf{Sat}\left(\mathcal{P}_{\geqslant 0.92}\left([L]\ \mathsf{Until}^{\leqslant 11.2}\ [G]\right)\right)$$

if and only if

$$\sum_{s' \in G} \pi_s(s', 11.2) \ \geqslant\ 0.92 \qquad (4)$$

Here, we emphasize that s is a state in CTMC M whereas the vector $\pi_s(11.2)$ is computed in the new chain M'.

2.1 Computational complexity

The computational procedure as exemplified before may seem time-consuming, but this is not the case [6]. As we follow a recursive descent over the formula representing the performance guarantee to be checked, the worst-case time complexity is linear in the size of the formula. This simply follows from the fact that for each sub formula a single computation suffices. For checking long-run guarantees (i.e., formulae of the form $\mathcal{L}(\ldots)$), the graph analysis to determine the terminal SCCs has the same time complexity as a depth-first search, i.e., linear in the number of states and linear in the number of non-zero elements in the transition probability (or: rate) matrix. Solving systems of linear equations can be done rather efficiently using well-known numerical approximation techniques such as Gauß-Seidel, successive over-relaxation or conjugate gradient squared methods. The complexity of checking formulae of the form $\mathcal{P}(\ldots\, \mathsf{Until}\, \ldots)$ is the same as that of performing uniformization. This yields an algorithm that is polynomial in the length of the formula and in the size of the Markov chain at hand. This also applies to Markov reward processes, although for these processes more involved numerical techniques are needed, in particular those for computing transient rewards [1, 19, 14].

Fig. 1 gives an indication of the computation time that is needed to check the validity of a formula of the form $\mathcal{P}(\ldots\, \mathsf{Until}\, \ldots)$. This is obtained with a prototypical implementation (in C) using standard sparse-matrix representations.

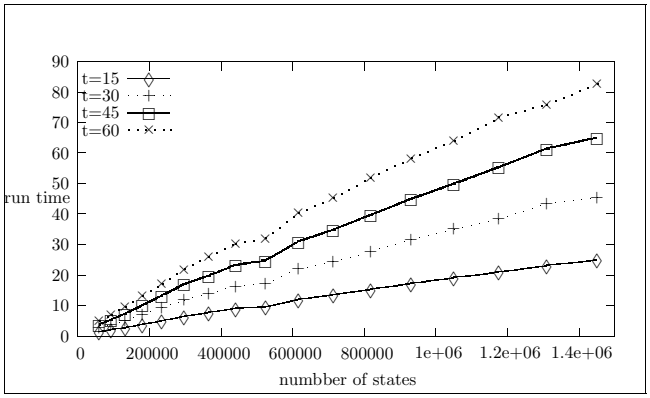

Figure 1: Computation time vs. number of states for checking $\mathcal{P}(\ldots \mathsf{Until} \ldots)$

2.2 Couldn't we analyze these guarantees before?

One could argue that the algorithms used as sub-routines are all well-known. True. This is, in fact, one of the strengths of this technique rather than one of its weaknesses. One could therefore argue that these measures could be computed already, so what's new? Indeed, our claim is not that specialists in performance analysis would not be able to check the performance guarantees that we logically describe. The methods provided in this note, make this process, however, much easier: there is one computational procedure for *all* measures one can write up in the logic, one does not need to know the underlying algorithms to check the guarantees, and finally, various guarantees can be assured for a single model without the need to manually adapt the model to the guarantee under consideration.

3 Abstraction

Like for any model-based performance evaluation technique, model checking suffers from the infamous state-space explosion problem: the size of the Markov chain often grows exponentially with the size of the high-level description. To combat this problem, reduction techniques based on *lumpability* are often employed. This allows for the computation of steady-state and transient-state probabilities on the quotient of the Markov chain under lumping equivalence. It is an interesting result that for the logic presented in this note (in its full form known as Continuous Stochastic Logic [2, 10], or CSL) the following result has been established [6, 17]: *lumping equivalence coincides with CSL equivalence*. This means that any two lumping-equivalent Markov chains cannot be distibguished by any CSL-formula, since they satisfy exactly the same formulae. Using efficient algorithms to construct the quotient space under lumpability [16], a Markov chain can be lumped prior to the model checking while preserving the results. Another consequence of this result is that in order to disprove that two Markov chains are lumping equivalent, it suffices to provide a single logical formula that holds in one but not in the other chain.

Lumpabilioty is akin to the notion of bisimulation [24]. Whereas lumpability relates states that mutually mimic all individual steps, *weak simulation* requires that one state can mimic all stepwise behavior of the other, but not the converse, and—in contrast to strong simulation relations—only requires this for certain ("observable") transitions and not for other ("silent") transitions. This allows for a more radical reduction of the state space than using lumpability, while preserving non-trivial cumulative transient-state probabilities such as the probability to reach a set of goal states within a given time bound. More precisely, let $s, s' \in S$ be states of the CTMC such that s is weakly simulated by s'. Then, for any set G of goal states and positive real number d we have:

$$s' \text{ satisfies } \mathcal{P}_{\leqslant p}([L]\ \mathsf{Until}^{\leqslant d}\ [G])$$

implies

$$s \text{ satisfies } \mathcal{P}_{\leqslant p}([L]\ \mathsf{Until}^{\leqslant d}\ [G])$$

This result can be generalised towards many more formulae [9]. While strong simulation preorder can be computed by solving a network flow problem [4], a recent result shows that the weak simulation pre-order can be solved in polynomial time using a reduction to a linear programming problem [8]. Weak simulation thus provides an effective and aggressive abstraction technique for Markov chains.

4 Tools

The are of probabilistic and stochastic model checking, as the research field is called, has received quite some attention in the last decade. Most importantly, this has led to several dedicated software tools as well as the incorporation of this approach into existing performance evaluation tools. We briefly describe the most relevant implementations and their characteristics:

- ETMCC was the first CTMC model checker [20]. It uses a sparse-matrix representation, is based on the algorithms explained in this note, and has a simple input format that enables its usage as a back-end to existing performance modeling tools.
- PRISM [22] is a model checker for Markov chains as well as Markov decision processes. It uses a mixed representation: a binary-decision diagram for the probability (or rate) matrix and a sparse representation for the solution vector. This tool has been applied to several case studies from different application fields [23].
- VESPA [25] and YMER [27] support the checking of performance guarantees on Markov chains (and more general stochastic processes). These tools do not use the numerical algorithms as explained in this note, but use statistical hypothesis testing to decide on the validity of logical guarantees.

- GreatSPN [15] and the APNN Toolbox [12] are examples of longer-existing performance modeling tools that have adopted stochastic model checking as an integral part. Both tools use as input language stochastic Petri nets. The APNN Toolbox supports CSl model checking using Kronecker representation of composed Markov chains.

5 Conclusions

This note has surveyed the model-checking approach to discrete and continuous-time Markov (reward) models. We believe that the model-checking approach provides a useful technique for performance and dependability analysis. Logics are useful for specifying performance guarantees, and model-checking algorithms provide effective (and efficient) means for checking these guarantees. This is done in a fully automated way, and provides a *single* framework for checking performance measures as well as functional properties such as absence of deadlocks and reponsiveness.

Although the focus in this note was on discrete- and continuous-time Markov chains, the logical framework can also be applied to specify guarantees over stochastic processes that might exhibit non-determinism. Indeed, most of the algorithms in this note can be generalised towards such Markov decision processes [11, 7]. Other extensions consider more powerful path-based properties [3], impulse rewards [14], quasi-birth-death processes [26], or semi-Markov chains [21].

References

[1] S. Andova, H. Hermanns and J.-P. Katoen. Discrete-time rewards model-checked. *Formal Methods for Timed Systems (FORMATS)*, LNCS 2791: 88–104, 2003.

[2] A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Model checking continuous time Markov chains. *ACM Trans. on Comp. Logic*, **1**(1): 162–170, 2000.

[3] C. Baier, L. Cloth, B.R. Haverkort, M. Kuntz, and M. Siegle. Model checking action- and state-labelled Markov chains. *Dependable Systems and Networks (DSN)*, pp. 701–710, IEEE CS Press, 2004.

[4] C. Baier, B. Engelen, and M. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. of Comp. and System Sc.*, **60**(1):187–231, 2000.

[5] C. Baier, B.R. Haverkort, H. Hermanns, J.-P. Katoen. On the logical specifi cation of performability properties. *Automata, Languages and Programming (ICALP)*, LNCS 1853: 780–792, 2000.

[6] C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. on Softw. Eng.*, **29**(6): 524–541, 2003.

[7] C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Effi cient computation of maximal timed reachability probabilities in uniform continuous-time Markov decision processes. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2988: 61–76, 2004.

[8] C. Baier, H. Hermanns and J.-P. Katoen. Probabilistic weak simulation is decidable in polynomial time. *Inf. Proc. Letters*, **89**(3): 123–130, 2004.

[9] C. Baier, H. Hermanns, J.-P. Katoen and V. Wolf. Comparative branching-time semantics for Markov chains. In R. de Simone and D. Lugiez (eds), *Concurrency Theory*, LNCS 2761, pp. 492–508, 2003.

[10] C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. *Concurrency Theory (CONCUR)*, LNCS 1664: 146–162, 1999.

[11] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distr. Comp.*, **11**: 125–155, 1998.

[12] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper. Model-checking large structured Markov chains. *J. of Logic and Alg. Progr.*, **56**:69–97, 2003.

[13] E. Clarke, O. Grumberg and D. Peled. *Model Checking*. MIT Press, 1999.

[14] L. Cloth, J.-P. Katoen, M. Khattri and R. Pulungan. Model-checking Markov reward models with impulse rewards. Submitted for publication, 2005.

[15] D. D'Aprile, S. Donatelli and J. Sproston. CSL model checking for the GreatSPN tool. *Int. Symp. on Computer and Information Sciences (ISCIS)*, LNCS 3280: 543–552, 2004.

[16] S. Derisavi, H. Hermanns and W.H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Proc. Lett.*, **87**(6): 309–315, 2003.

[17] J. Desharnais and P. Panangaden. Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes. *J. of Logic and Alg. Progr.*, **56**(1-2): 99–115, 2003.

[18] H.A. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.* **6**(5), (1994) 512–535.

[19] B.R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen and C. Baier. Model-checking performability properties. *Dependable Systems and Networks (DSN)*, pp. 103–113, IEEE CS Press, 2002.

[20] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. A tool for model checking Markov chains. *J. on Software Tools for Technology Transfer*, **4**(2):153–172, 2003.

[21] G.G. Infante-López, H. Hermanns and J.-P. Katoen. Beyond memoryless distributions: model checking semi-Markov chains. In *Process Algebra and Probabilistic Methods*, LNCS 2165:57–70, 2001.

[22] M. Kwiatkowska, G. Norman and D. Parker. Probabilistic symbolic model checking using PRISM: a hybrid approach. *J. on Software Tools for Technology Transfer*, **6**(2): 128–142, 2004.

[23] M. Kwiatkowska, G. Norman and D. Parker. Probabilistic model checking in practice: case studies with PRISM. This volume, 2005.

[24] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. and Comp.*, **94**(1): 1–28, 1992.

[25] K. Sen, M. Viswanathan and G. Agha. Statistical model checking of black-box probabilistic systems. *Computer-Aided Verification (CAV)*, LNCS 3114: 202–215, 2004.

[26] A. Remke, L. Cloth and B.R. Haverkort. Model-checking infi nite-state Markov chains. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, 2005 (to appear).

[27] H. Younes and R. Simmons. Probabilistic verifi cation of discrete event systems using acceptance sampling. *Computer-Aided Verification*, LNCS 2404: 223-235, 2002.