Model Checking: One Can Do Much More Than You Think!

Joost-Pieter Katoen[1,2]

[1] RWTH Aachen University, Software Modelling and Verification Group, Germany
[2] University of Twente, Formal Methods and Tools, The Netherlands

Abstract. Model checking is an automated verification technique that actively is applied to find bugs in hardware and software designs. Companies like IBM and Cadence developed their in-house model checkers, and acted as driving forces behind the design of the IEEE-standardized temporal logic PSL. On the other hand, model checking `C`-, `C#`- and .NET-program code is an intensive research topic at, for instance, Microsoft and NASA. In this short paper, we briefly discuss three non-standard applications of model checking. The first example is taken from systems biology and shows the relevance of probabilistic reachability. Then, we show how to determine the optimal scheduling policy for multiple-battery systems so as to optimize the system's lifetime. Finally, we discuss a stochastic job scheduling problem that —thanks to recent developments— can be solved using model checking.

1 Introduction

Despite the scepticism in the early eighties, it is fair to say that model checking is scientifically a big success. Prizes such as the Paris Kanellakis Award 1998 awarded to Bryant, Clarke, Emerson and McMillan for their invention of "symbolic model checking, the Gödel prize 2000 —the equivalent of the Nobel prize in Mathematics— was awarded to Vardi and Wolper for their work on model checking with finite automata, and last but not least, the Nobel prize in Computer Science, the ACM Turing Award 2007, was granted to the inventors of model checking, Clarke, Emerson and Sifakis. The impact of model checking tools is clearly demonstrated by the ACM System Software Award 2001, granted to Holzmann, for his model checker SPIN, "a popular open-source software tool, used by thousands of people worldwide, that can be used for the formal verification of distributed software systems". Other winners of this prestigious award are, e.g., `TeX`, Postscript, `unix`, TCP/IP and `Java`, to mention a few.

Model checking is based on an exhaustive state space search; in fact, checking whether a set of target states is reachable from a given state is at the heart of various model-checking algorithms. The prime usage of model checking [6, 2, 8] is bug hunting: finding flaws in software programs, hardware designs, communication protocols, and the like. The feature of model checkers to generate a counterexample in case a property is refuted is extremely useful and turns model checking into an intelligent and powerful debugging technique. This feature combined with an abstraction-refinement loop is currently main stream in software verification. Success stories include the demonstration of conceptual bugs in an international standard proposal for a cache coherence protocol, catching a fatal flaw in the Needham-Schröder authentication protocol, but also the usage of model checking in designing device drivers in recent Microsoft operating systems, and highly safety-critical NASA space missions. The fact that the Property Specification Language (PSL), basically a derivative of linear temporal logic enriched with regular expressions, has become an IEEE standard since 2005 for specifying properties or assertions about hardware designs, is a clear sign that formal verification techniques such as model checking has significantly gained popularity and importance.

Model checking can however be applied to various problems of a completely different nature. It can be used for instance to solve combinatorial puzzles such as the famous Chapman puzzle [7] and Sudoku problems. In the rest of this short paper, we will discuss three non-standard applications of model checking. The first example is taken from systems biology and shows the relevance of probabilistic reachability. Then, we show how to determine the optimal scheduling policy for multiple-battery systems. Finally, we discuss a stochastic scheduling problem that—thanks to quite recent developments—can be solved using model checking. All examples share that the models and properties that we will check are *quantitative*. This is an important deviation from traditional model checking that focuses on functional correctness and models. It is our firm belief that quantitative model checking will gain importance in the (near) future and will become a technique that is highly competitive in comparison to standard solution techniques for quantitative problems.

2 Systems biology: enzyme kinetics

Enzyme kinetics investigates of how enzymes (E) bind substrates (S) and turn them into products (P). About a century ago, Henri considered enzyme reactions to take place in two stages. First, the enzyme binds to the substrate, forming the enzyme-substrate complex. This substrate binding phase catalyses a chemical reaction that releases the product. Enzymes can catalyse up to several millions reactions per second. Rates of kinetic reactions are obtained from enzyme assays, and depend on solution conditions and substrate concentration. The enzyme-substrate catalytic substrate conversion reaction is described by the stoichiometric equation:

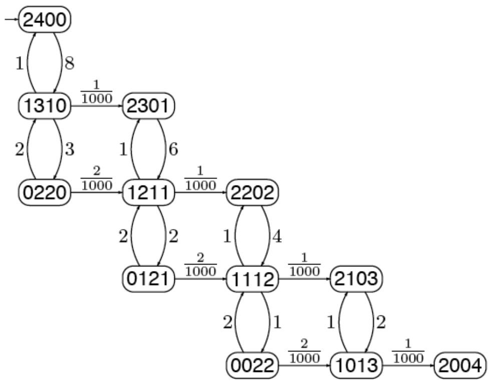
$$E + S \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C \xrightarrow{k_3} E + P$$

where k_i is the Michaelis-Menten constant, which is the substrate concentration required for an enzyme to reach one-half of its maximum reaction rate. Now let suppose we have N different types of molecules that randomly collide. The state $X(t)$ of the biological system at time instant $t \in \mathbb{R}_{\geqslant 0}$ is given by $X(t) =$

$(x_1, \ldots, x_N)$ where x_i denotes the number of species of sort i. In the enzyme-catalytic substrate conversion case, $N{=}4$ and $i \in \{C, E, P, S\}$. Let us number the types of reaction, e.g., $E{+}S \to C$ and $C \to E{+}S$ could be the first and second reaction, respectively. The reaction probability of reaction m within the infinitesimally small time-interval $[t, t{+}\Delta)$ with $\Delta\mathbb{R}_{\geqslant 0}$ is given by:

$$\alpha_m(\boldsymbol{x}) \cdot \Delta \;=\; \Pr\{\text{reaction } m \text{ in } [t, t{+}\Delta) \mid X(t) = \boldsymbol{x}\}$$

where $\alpha_m(\boldsymbol{x}) = k_m \cdot$ the number of possible combinations of reactant molecules in $\boldsymbol{x}$. For instance, in state (x_E, x_S, x_C, x_P) where $x_i > 0$ for all i, the reaction $E + S \to C$ happens with rate $\alpha_m(\boldsymbol{x}) = k_1 {\cdot} x_E {\cdot} x_S$ and yields the state $(x_E{-}1, x_S{-}1, x_C{+}1, x_P)$. This stochastic process possesses the Markov property, i.e., its future is completely described by the current state of the system. Moreover, it is time-homogeneous, i.e., its behaviour is invariant with respect to time shifts. In fact, it is a *continuous-time Markov chain* (CTMC, for short).



States:	*init*	*goal*
enzymes	2	2
substrates	4	0
complex	0	0
products	0	4

Fig. 1. CTMC for enzyme-catalytic substrate conversion for initially 2 enzyme and 4 substrate species with $k_1 = k_2 = 1$ and $k_3 = 0.001$. The transition labels are rates of exponential distributions, i.e., the reciprocal of the average duration of a reaction.

Let us now consider the following question: given a certain concentration of enzymes and substrates, what is the likelihood that after four days all substrates have engaged in a catalytic step and resulted in products? In terms of the CTMC, this boils down to determining the probability that starting from the state $(x_E, x_S, 0, 0)$ we can reach a state of the form $(x_E, 0, 0, x_P)$ within four days. This is a so-called *time-bounded reachability* property that we can tackle by model checking thanks to the following result:

Theorem 1. *[3] The following reachability problem is efficiently computable:*

Input: a finite CTMC, a target state, accuracy $0 < \epsilon < 1$, and deadline $d \in \mathbb{R}_{\geqslant 0}$
Output: an ϵ-approximation of the probability to reach the target in d time.

This result suggests to use an off-the-shelf probabilistic model checker for CTMCs such as `prism` [14] or `mrmc` [12]. Due to the large difference between the rates in the CTMC —the rates between states within one column is about a factor 1,000 times larger than the rates between columns— many iterations are needed to obtain results for a reasonable ϵ, say 10^{-4} or 10^{-6}. Verifying a configuration with 200 substrates and 20 enzymes yielding a CTMC of about 40,000 states, e.g., takes many hours. In order to deal with this problem, we apply aggressive abstraction techniques that are based on partitioning the state space. This manual step is guided by the following rule of thumb: group states that are quickly connected, i.e., group the states in a column-wise manner. This yields a chain structure as indicated in Fig. 2. Now the next step of the abstraction is to take

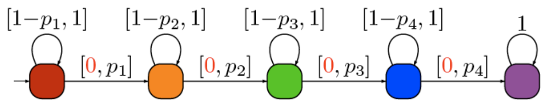


Fig. 2. Abstract CTMC for enzyme-catalytic substrate conversion for 2 enzyme and 4 substrate species after a state partitioning. The transition labels are probability intervals. Rates are omitted, as the residence times of all states has been normalised prior to abstraction, cf. [12].

several transitions into account. For instance, the lower bound probability of moving from the leftmost abstract state to the one-but-leftmost state is 0, as the state 2400 cannot move to any state of the form $(x_E, x_S, x_C, 1)$ in one step, i.e., by taking a single transition. This yields rather course lower bounds. To overcome this deficiency, we consider several steps. That is to say, in addition to the state partitioning, we consider an abstraction of sequences of transitions. The resulting structure is sketched in Fig. 3 where the most important change is the amendment of the lower bounds in the probability intervals, and the addition of transitions. The length k of the sequences that are abstracted from is a

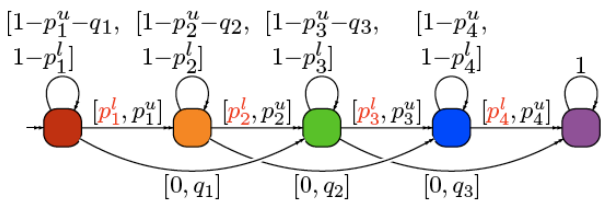
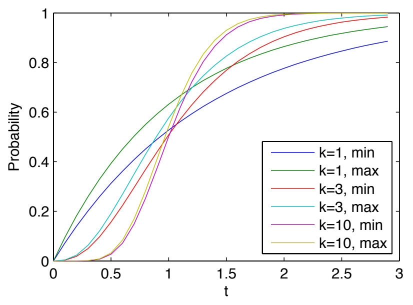
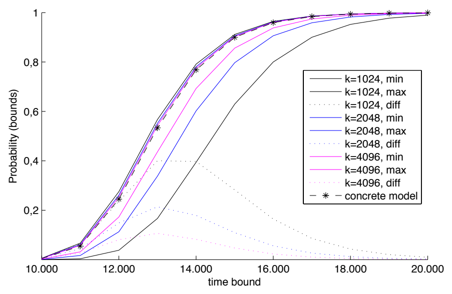


Fig. 3. Abstract CTMC for enzyme-catalytic substrate conversion for 2 enzyme and 4 substrate species after a state and transition sequence abstraction. The transition labels are probability intervals. State residence times now are Erlang distributions.

parameter of the abstraction procedure. The state residence times now become sequences of (equal) exponential distributions, i.e., they become Erlang distributions of length k. As a result of the intervals on the transition probabilities, the analysis of the abstract CTMC yields lower and upper bounds of the real probability. On increasing the parameter k, the difference between these bounds becomes smaller. This effect is illustrated in Fig. 4(a). Our method is accurate



(a) The influence of k on the accuracy of bounds.



(b) Time-bounded reachability bounds for enzyme-catalysed substrate conversion.

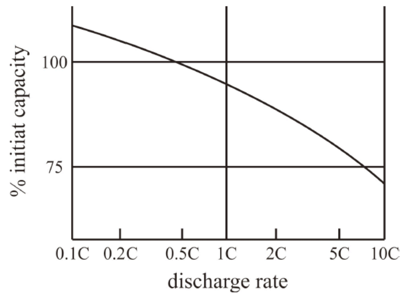
as the obtained intervals are small, e.g., for $x_S = 200$, $k = 2^{12}$, and time-bound $t = 14,000$, the relative interval width between the lower and upper bounds is about 10%. The column-wise abstraction results in a state space reduction by a factor 20 and reduces the run-times with several orders of magnitude. For further details on this case study we refer to [11]. The results have been obtained using the `mrmc` model checker [12].

To conclude, model checking combined with novel aggressive abstraction techniques yield a powerful technique to check interesting properties of biological systems. The technique is highly competitive with existing techniques such as solving chemical master equations and Monte carlo simulation. Recent experiments indicate that these techniques are also very helpful for a completely different application area—queueing theory. By means of abstraction we were able to analyse timed reachability properties for so-called tree-based quasi-birth-death processes with state spaces of up to 10^{278} states by abstractions of about 1,2 million states with an accuracy of $\epsilon = 10^{-6}$, see [13]. To our knowledge, this was the first time ever that tree-shaped Markov models of this size have been analysed numerically.

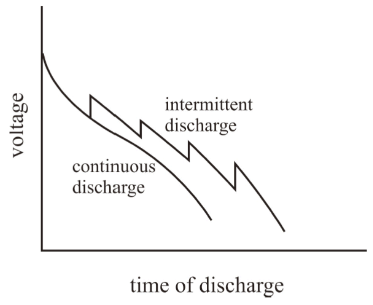
3 Optimal battery scheduling

As argued in the introduction, an important feature of model checking is the possibility to generate counterexamples in case a property is refuted. For instance, for the property $\Box(x > 2)$, expressing that along a path any state should satisfy $x > 2$ for integer variable x, a counterexample is a finite path reaching a state for which $x \leqslant 2$. Counterexamples can be used for scheduling problems in the following way. Suppose that we are interested in finding a schedule that steers a system from a starting to a target state, G, say. Then we model the possible non-deterministic moves of the system by means of a finite transition system, and check whether the property $\neg\Diamond G$, or equivalently, $\Box\neg G$, holds. If there exists a schedule leading to G, the model checker will refute the property $\Box\neg G$ and yields a finite schedule as counterexample. A similar strategy can be applied to real-time systems extended with costs where schedules are sought that minimize the total costs. This will be briefly illustrated in the following example where we will use costs to model energy consumption.

It is well-known that the battery lifetime determines system uptime and heavily depends on the battery capacity, the level of discharge current, and the usage profile. We consider the following problem: given a number of batteries and a usage profile, what is the optimal policy to empty the batteries such that the multi-battery system's lifetime is maximized. It is certainly far from optimal to solve this off-line scheduling problem by emptying the batteries in a sequential fashion due to the recovery effect: during idle periods, the battery regains some of its capacity, cf. Fig. 4(d). There is an electro-chemical explanation for this recovery effect. Ions have to diffuse from the anode to the cathode of the battery. At high currents, the internal diffusion is too slow and the reaction sites at the cathode surface get blocked. During idle periods, ions get time to diffuse again and accordingly the battery's capacity increases. Alternative scheduling strategies that can exploit this recovery during idel periods are round-robin (empty the batteries according to fixed total order), or best-of-N strategies (use the mostly charged battery among the available N ones). We will show that optimal scheduling policies can be obtained using model checking of *priced timed automata*.



(c) The rate-capacity effect: the battery capacity (y-axis) drops for high discharge currents (x-axis). A discharge rate of 0.5 C means that the total discharge takes 2 hours.



(d) The recovery effect: battery regains capacity during idle periods. This yields the raw-tooth curve.

A second non-linear effect of batteries that has to be taken into account is the so-called rate-capacity effect, see Fig. 4(c). One would think that the ideal capacity would be constant for all discharge currents, and all energy stored in the battery would be used. However, in reality for a real battery the voltage slowly drops during discharge and the effective capacity is lower for high discharge currents. The discharge rate in Fig. 4(c) is given in terms of C rating, a C rating of $2C$ means that the battery is discharged in 1/2 hour. The measured capacities are given relatively to the capacity at the 2 hour discharge rate, 0.5 C.

The battery model we use is based on the kinetic battery model for lead-acid batteries as developed by Manwell & McGowan [15]. In this model, the charge of the battery is distributed over two wells, the available charge with height h_1 and the bound charge with height h_2, see Fig. 4. The available charge represents the charge that is currently available for usage. Discharging leads to a decrease of h_1. The battery is empty if and only if $h_1 = 0$. When the battery is idle, i.e., not being discharged, charge flows from the bound charge to the available charge. The speed depends on the height difference $h_2 - h_1$ and the resistance k between the two wells. This models the recovery effect. The rate capacity effect is captured by the fact that at higher discharge levels, there is less time to recover. Let y_1 be the volume of the available charge well and y_2 the volume of the boundary charge well. The behaviour of the kinetic battery model is captured

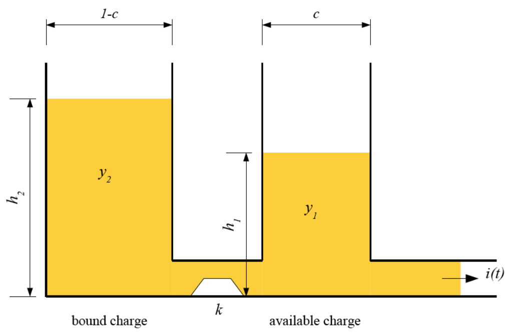


Fig. 4. The kinetic battery model with a boundary and available charge well of height $h_1(t)$ and $h_2(t)$ at time t, respectively. The discharge $i(t)$ at time point t is depicted on the right and will lead to a decrease of the available charge. Recovery is modelled by a charge flow between the boundary and available well when $i(t) = 0$.

by the following set of linear differential equations:

$$\begin{aligned} h_1(t) &= \tfrac{y_1(t)}{c} & \dot{y_1}(t) &= -i(t) + k\cdot(h_2(t) - h_1(t)) \\ h_2(t) &= \tfrac{y_2(t)}{1-c} & \dot{y_2}(t) &= -k\cdot(h_2(t) - h_1(t)) \end{aligned}$$

with initial conditions $y_1(0) = c\cdot C$ and $y_2(0) = (1-c)\cdot C$ where C is the total capacity and $0 < c < 1$ for constant c. Here, $i(t)$ represents the discharge process.

The kinetic battery model can naturally be described by a network of *priced timed automata*. Intuitively speaking, clocks in timed automata are used to model the advancement of time t, whereas cost variables are used to model the battery charge (in fact, the reverse). A timed automaton is in fact a finite-state automaton equipped with real-valued clocks that can be used as timers to measure the elapse of time. Constraints on these clocks can be used to guard state-transitions, and clocks can be set to zero while taking a transition. In priced timed automata, states are equipped with a cost rate r such that the accumulated cost in that state over a time period d grows with $r \cdot d$.

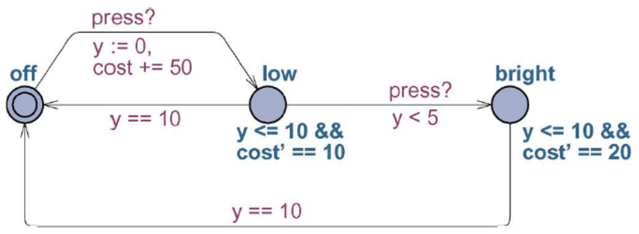


Fig. 5. Example priced timed automaton of a lamp. The cost rate is 0 in state *off*, 10 in state *low* and 20 in state *bright*. Cost represents energy consumption.

We now model the battery scheduling problem as:

$$\underbrace{(DC_1 \,||\, RC_1)}_{\text{battery 1}} \,||\, \ldots\ldots \,||\, \underbrace{(DC_n \,||\, RC_n)}_{\text{battery } n} \,||\, Load \,||\, Scheduler$$

where DC_i describes the discharging process of the battery i, RC_i the recovery effect during idle periods of battery i, $Load$ the usage profile and $Scheduler$ an automaton that non-deterministically selects one of the batteries for discharging once the usage profile demands a discharge. Then we exploit the following result:

Theorem 2. *[4, 1] The following reachability problem is effectively computable:*

Input: a priced timed automaton, an initial state, and a target state
Output: the minimum cost of runs from the initial state to the target.

As a by-product of the computation of the minimal cost run, an optimal schedule is obtained that achieves this minimal-cost run.

Our objective is to minimize the bound charge levels (of all batteries) once all batteries are empty, i.e., once all available charges are empty. Table 1 presents the results for two batteries for several usage profiles (the rows) and several battery scheduling disciplines (columns). The last column presents the battery lifetimes obtained by model checking our priced timed automaton. These results have been obtained using the `uppaal cora` model checker [1]. The recovery effect becomes clearly apparent when comparing, e.g., the rows for the usage profiles ILs_250 and ILℓ_250. Both profiles have a peak charge of 250 Amin and peak with equal duration, but the idle time between successive discharging periods is small and long, respectively. This almost doubles the battery lifetime. A similar phenomenon appears for profiles ILs_500 and ILℓ_500. The optimal battery

[1] www.uppaal.com

test load	sequential lifetime (min)	round robin lifetime (min)	best-of-two lifetime (min)	optimal lifetime (min)
CL_250	9.12	11.60	11.60	12.04
CL_500	4.10	4.53	4.53	4.58
CL_alt	5.48	6.10	6.12	6.48
ILs_250	22.80	38.96	38.96	40.80
ILℓ_250	45.84	76.00	76.00	78.96
ILs_500	8.60	10.48	10.48	10.48
ILℓ_500	12.94	15.96	15.96	18.68
ILs_alt	12.38	12.82	16.30	16.91
ILs_r1	12.80	16.26	16.26	20.52

Table 1. Lifetimes of a multi-battery system under various usage profiles (first column) and various scheduling disciplines (second to fourth column). The optimal lifetimes obtained by model checking are listed in the last column.

lifetimes obtained by model checking (last column) clearly outperform round-robin and best-of-two scheduling. Note that best-of-two is not much better than round-robin, and requires the ability to measure the remaining capacity of the batteries. Sequential scheduling is far from attractive. An example schedule that is obtained by model checking (in red), and compared to a best-of-two schedule (in blue) for a given usage profile (uppermost block curve, in black) is provided in Fig. 6.

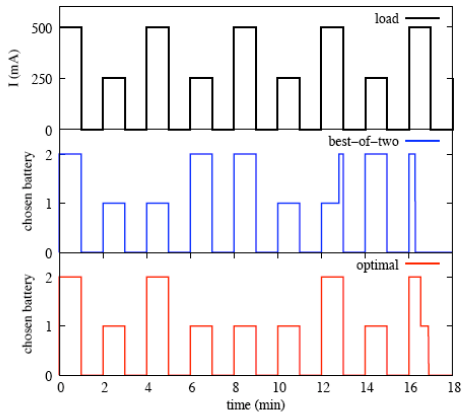


Fig. 6. Example of obtained optimal schedule for two batteries (lowermost curve) for a given usage profile (uppermost curve), compared to a best-of-two scheduling policy (middle curve).

To conclude, model checking allows for computing the optimal battery scheduling policy. Alternative techniques to obtain such policies are by solving non-linear optimisation problems. It is fair to say, that the obtained optimal schedules using this technique are not easily implementable in realistic battery-powered systems such as PDAs or sensor nodes. By means of model checking, one can however determine the quality of a given scheduling policy by comparing it to the optimal one. The above experiments show that round-robin scheduling is mostly behaving quite good. For further details on this case study we refer to [9, 10].

4 Stochastic scheduling

The third application example is slightly more theoretical, and aims to illustrate how state-of-the-art stochastic model checking techniques can be used to solve stochastic scheduling problems. Stochastic scheduling is important in the field of optimization [19], and is motivated by problems of priority assignment in various systems where jobs with random features, such as random durations, or arrival processes, are considered, or in which machines are present that are subject to random failures.

More concretely, we consider the scheduling of N jobs on K identical machines, where $K \ll N$. Every job has a random duration such that job i has a mean duration of $d_i > 0$ time units. The most appropriate stochastic approximation is to model the duration of job i by a negative exponential distribution with rate $\lambda_i = \frac{1}{d_i}$. (Technically speaking, given that only the mean of a random event is known, the probability distribution that maximizes the entropy is an exponential one with exactly this mean; intuitively, maximizing entropy minimizes the amount of prior information built into the probability distribution.) Jobs are scheduled on the machines such that job scheduling is pre-emptive. The pre-emptive scheduling allows us to assign each machine one of the n remaining jobs giving rise to $\binom{n}{K}$ possible choices. This means that on finishing of a job on machine j, every job on any other machine can be pre-empted. This scheme is illustrated by a decision tree for 4 jobs and 2 machines in Fig. 7. Every node in the tree is labelled with the set of remaining, i.e., unfinished jobs. The underlined job numbers are those that are selected for execution; if one of the jobs, i say, expires first in a situation where n jobs have not been processed yet, an event that happens with probability $\frac{\lambda_i}{\lambda_i+\lambda_j}$ (where j is the number of the other selected, but unfinished, job), finishes, $n-1$ jobs remain, and a new selection is made. The time that has elapsed is determined by the rate λ_i. Due to the memoryless property of the exponential distribution, the remaining execution time of the pre-empted job j remains exponentially distributed with rate λ_j.

It is well-known that the LEPT policy —the longest expected processing time-first policy— yields the minimal expected finishing time of the last job (also called the expected makespan), cf. [5]. As [5] however argues, "it is hard to calculate these expected values". We will show how probabilistic model checking can be applied to address a harder question, namely: which policy maximizes the probability to finish all jobs on time? (The alerted reader might argue that

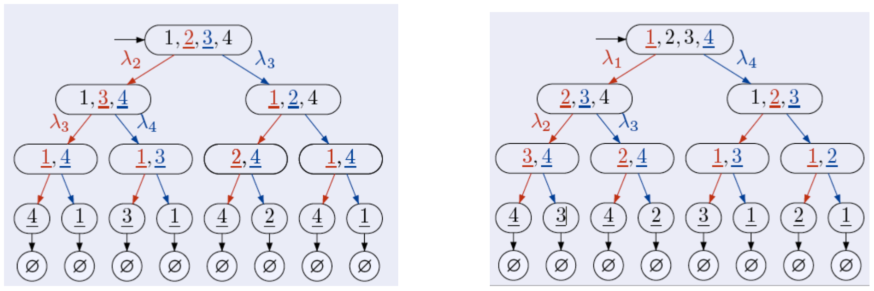


Fig. 7. Two possible schedules of 4 jobs on 2 machines with pre-emptive scheduling policy. In the left one, jobs 2 and 3 are selected first; in the right one, jobs 1 and 4 are initially picked.

this question is somehow related to the biology case study, and indeed it is. The difference is that the biology example is fully deterministic, that is, in fact an instance of the above case in which there is only a single possible choice in every node of the decision tree.)

This stochastic job scheduling problem naturally gives rise to a *continuous-time Markov decision process* (CTMDP, for short) [2]. This model is a generalisation of CTMCs, the model used in the first case study, with non-determinism. In every state, an action (ranged over by α) is selected non-deterministically, see Fig. 8. In our setting, an action corresponds to a scheduling decision of which jobs to process next. The residence time in a state is exponentially distributed. Fig. 8. The problem of determining the policy that maximizes the probability to finish all jobs within d time units now reduces to the following question: what is the maximal probability to reach the sink state within d time units? This can be solved by means of model checking using the following result.

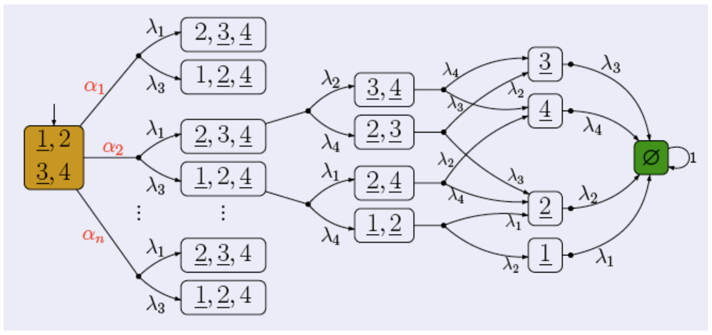


Fig. 8. Possible schedules for 4 jobs on 2 machines, modelled as a continuous-time Markov decision process.

Theorem 3. *[18] The following reachability problem is effectively computable:*

[2] In fact, a locally uniform continuous-time Markov decision process [17].

Input: a finite CTMDP, a target state, accuracy $0 < \epsilon < 1$, and deadline $d \in \mathbb{R}_{\geqslant 0}$
Output: an ϵ-approximation of the maximal (or dually, minimal) probability to reach the target in d time.

Importantly though is that as a by-product of determining this ϵ-approximation, one obtains an ϵ-optimal policy that yields this maximal probability (up to an accuracy of ϵ). The main complication of this timed reachability problem is that the optimal policies are time-dependent. This is an important difference with reachability questions for discrete-time Markov decision processes (MDPs) for which time-independent policies suffice, e.g., policies that in any state always take the same decision. The decisions of time-dependent policies may vary over time and may for instance depend on the remaining time until the deadline d. Their computation is done via a discretisation yielding an MDP on which a corresponding step-bounded reachability problem is solved using value iteration. The smallest number of steps needed in the discretised MDP to guarantee an accuracy of ϵ is $\frac{\lambda^2 \cdot d^2}{2\epsilon}$ where λ is the largest rate of a state residence time in the CTMDP at hand. In a similar way, minimal timed reachability probabilities can be obtained and their corresponding policies.

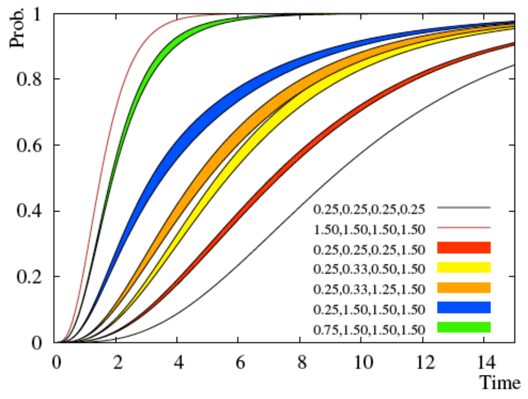


Fig. 9. Minimal and maximal reachability probabilities for finishing 4 jobs on 2 machines under a pre-emptive scheduling strategy.

The results of applying this discretisation on the example with 4 jobs and two machines is shown in Fig. 9 where the deadline d is given on the x-axis and the reachability probability on the y-axis. For equally distributed job durations, i.e., $\lambda_i = \lambda_j$ for all i, j, the maximal and minimal probabilities coincide. Otherwise, the probabilities depend on the scheduling policy. It turns out that the ϵ-optimal scheduler that maximizes the reachability probabilities adheres to the SEPT (shortest expected processing time first) strategy; moreover, the optimal ϵ-scheduler for the minimum probabilities obeys the LEPT strategy. These results have been obtained by a vanilla version of the model checker `mrmc` [12]. The case study is described in more detail in [16].

5 Concluding remarks

By means of three examples from different application fields, we have attempted to argue that model checking is applicable to problems of a quite different nature than what is typically considered as verification problems. All problems have a quantitative flavour, i.e., non-functional aspects such as timing, randomness, and costs (energy) are essential to adequately model the applications at hand. We belief that there is an increased need for quantitative model checking as the importance of non-functional aspects is growing at staggering rate. We stress that in the last two examples we used model checking to *synthesize* an optimal schedule.

The battery example can certainly also be handled with existing techniques such as mixed integer linear programming. Dynamic programming techniques using Bellman equations can be used to tackle the stochastic planning example. The systems biology example can be handled using the chemical master equation or by Gillespie's simulation algorithm. Truly so. Our take-home message is not that model checking is the best and most efficient technique to tackle the described problems here; it is a valuable and interesting alternative that in some cases might be well competitive with existing traditional solution techniques. Model checking is on its way to become ubiquitous!

Acknowledgement. I thank all co-workers on the discussed case studies: Henrik Bohnenkamp, Boudewijn Haverkort, Marijn Jongerden, Daniel Klink, Alexandru Mereacre, Martin Neuhäusser, Martin Leucker, Verena Wolf, and Lijun Zhang. Part of this work has been established in the context of the EU FP7 QUASIMODO project, the NWO-DFG bilateral ROCKS project and the DFG Research Training Group ALGOSYN.

References

1. R. Alur, S. L. Torre, and G. J. Pappas. Optimal paths in weighted timed automata. *Theor. Comput. Sci.*, 318(3):297–322, 2004.
2. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
3. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *10th Int. Conf. on Concurrency Theory (CONCUR)*, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 1999.
4. G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Hybrid Systems: Computation and Control (HSCC)*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001.
5. J. L. Bruno, P. J. Downey, and G. N. Frederickson. Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *J. ACM*, 28(1):100–113, 1981.
6. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
7. E. M. Clarke and H. Schlingloff. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning (Volume II)*, chapter 24, pages 1635–1790. 2000.

8. O. Grumberg and H. Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.
9. M. R. Jongerden, B. R. Haverkort, H. C. Bohnenkamp, and J.-P. Katoen. Maximizing system lifetime by battery scheduling. In *39th IEEE/IFIP Conf. on Dependable Systems and Networks (DSN)*, pages 63–72. IEEE Computer Society, 2009.
10. M. R. Jongerden, A. Mereacre, H. C. Bohnenkamp, B. R. Haverkort, and J.-P. Katoen. Computing optimal schedules for battery usage in embedded systems. *IEEE Trans. Industrial Informatics*, 5(3):276–286, 2010.
11. J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Abstraction for stochastic systems by Erlang's method of stages. In *19th Int. Conf. on Concurrency Theory (CONCUR)*, volume 5201 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2008.
12. J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, 2011.
13. D. Klink, A. Remke, B. R. Haverkort, and J.-P. Katoen. Time-bounded reachability in tree-structured QBDs by abstraction. *Perform. Eval.*, 68(2):105–125, 2011.
14. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: probabilistic model checking for performance and reliability analysis. *SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.
15. J. Manwell and J. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5):399–405, 1993.
16. M. R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. PhD thesis, RWTH Aachen University and University of Twente, 2010.
17. M. R. Neuhäußer, M. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *12th Int. Conf. on Foundations of Software Science and Computational Structures (FOSSACS)*, volume 5504 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2009.
18. M. R. Neuhäußer and L. Zhang. Time-bounded reachability probabilities in continuous-time Markov decision processes. In *7th Int. Conf. on the Quantitative Evaluation of Systems (QEST)*, pages 209–218. IEEE Computer Society, 2010.
19. J. Nino-Mora. Stochastic scheduling. In *Encyclopedia of Optimization*, volume V, pages 367–372. Springer, 2001.