Stochastic Bounded Model Checking: Bounded Rewards and Compositionality *

Bettina Braitling Ralf Wimmer Bernd Becker
Albert-Ludwigs-University Freiburg, Germany
{braitlin,wimmer,becker}@informatik.uni-freiburg.de

Erika Ábrahám
RWTH Aachen, Germany
abraham@cs.rwth-aachen.de

Abstract

We extend the available SAT/SMT-based methods for generating counterexamples of probabilistic systems in two ways: First, we propose bounded rewards, which are appropriate, e. g., to model the energy consumption of autonomous embedded systems, and show how to extend the SMT-based counterexample generation to handle such models. Second, we describe a compositional SAT encoding of the transition relation of Markov models, which exploits the system structure to obtain a more compact formula, which in turn can be solved more efficiently. Preliminary experiments show promising results in both cases.

1. Introduction

The formal *verification* of embedded systems has gained great importance both in research and in industry. Model checking proves or refutes automatically (i. e., without user interaction) that a system exhibits some given properties [4]. One of the most useful features is that model checking provides helpful diagnostic information [7]: in case of a defective system a *counterexample* in form of a witnessing run is returned.

The usage of symbolic representations like ordered binary decision diagrams (OBDDs) [6] made model checking applicable to many kinds of large systems. However, there are classes of practically relevant systems for which even these OBDD-based methods fail due to space restrictions. To fill this gap, *bounded model checking* (*BMC*) was developed [8]. Thereby the existence of a path of a fixed length that refutes the property under consideration is formulated as a satisfiability (SAT) problem. The size of the SAT problem is always linear in the size of the embedded system to be analyzed (in contrast to the size of the state space, which may be exponentially larger). As modern SAT solvers have strongly evolved over the last 15 years, it is not surprising that the BMC approach is very successful.

To model real-life scenarios it is often necessary to cope with specific uncertainties using *probabilities*. As an example consider a sensor node which transmits its data over an unreliable wireless

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) and the DFG-Project "CEBug – CounterExample Generation for Stochastic Systems using Bounded Model Checking"

connection. Properties involving probabilities of events can be formulated in probabilistic computation tree logic (PCTL) [13] for models called *discrete-time Markov chains* (*DTMCs*). The model checking approach for DTMCs is based on solving a linear equation system [13]. However, it lacks the generation of counterexamples, since the solution of the equation system yields only the mere probabilities without further information.
To provide a user with diagnostic information for probabilistic systems, a lot of research efforts have been made during the last years [3, 12, 19, 2, 1, 20, 16]. Contrary to, e. g., LTL model checking for digital systems, counterexamples for a PCTL property in general consist of a large number of paths to reach certain probability bounds. Various techniques have been proposed to obtain compact representations of these paths: computing smallest path sets [12, 2], reducing the strongly connected components of a DTMC [3, 1], using regular expressions to describe sets of paths [12], and using small subsystems of DTMCs as counterexamples [2, 18, 20]. All these techniques rely on an explicit representation of the state space.
Bounded model checking for DTMCs was introduced in [19] to make couterexample generation applicable to large models. The refutation of a probabilistic safety property is shown by using a SAT-solver to find paths leading to a safety-critical state such that the joint probability of the paths exceeds a given bound. The input of the solver are propositional formulae that are satisfied iff the assignment of the variables corresponds to a sequence of states of the DTMC that represents a path to a target state. This procedure was extended by SMT-solving [5] instead of SAT. This allows to find more probable paths first, as well as to handle Markov reward models (MRMs).
In this paper we present two extensions to the BMC-based counterexample generation for Markov models: In the first part we present a new variant of MRMs: In ordinary MRMs the reward of states is typically restricted to non-negative values and the accumulated reward may grow unboundedly during the execution. In our variant, the *bounded Markov reward model* (BMRM), the reward is only developing within a certain interval. This is well suited to model, e. g., rechargeable batteries. The electric charge cannot raise above a certain maximal level, nor can it sink below zero. We modify the SMT-BMC procedure to generate counterexamples for BMRMs.
In the second part of this paper we improve the SAT-BMC procedure by taking the modular structure of a composed system into account. Since our encoding avoids the explicit composition of the modules, we obtain in many cases considerably smaller SAT problems to solve, speeding up the computation and reducing the memory requirement.

Organization of the paper. At first, we give a brief introduction to the foundations of DTMCs, counterexamples, and Markov reward models and sketch the BMC-based counterexample method. In Section 3 we introduce bounded Markov reward models and show how to compute counterexamples for them using the SMT-BMC procedure. In Section 4 we present our SAT-based approach for compositional path search. An experimental evaluation showing the effectiveness of our approaches is given in both sections. Section 5 concludes our paper and gives an outlook to future work.

2. Foundations

In this section we take a brief look at the basics of discrete-time Markov chains and discrete-time Markov reward models.

2.1. Discrete-Time Markov Chains

Definition 1 *Let* AP *be a set of atomic propositions. A* ***discrete-time Markov chain*** *(DTMC) is a tuple* $M = (S, s_I, P, L)$ *such that* S *is a finite set of states,* $s_I \in S$ *is an initial state,* $P : S \times S \to [0,1]$ *specifies the one-step transition probabilities such that* $\sum_{s' \in S} P(s,s') = 1$ *for all* $s \in S$*, and* $L : S \to 2^{\mathrm{AP}}$ *is a labeling function that assigns to each state the set of atomic propositions which hold in that state.*

A (finite or infinite) *path* π is a (finite or infinite) sequence $\pi = s_0 s_1 \ldots$ of states from S such that $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A finite path $\pi = s_0 s_1 \ldots s_n$ has *length* $|\pi| = n$; for infinite paths we set $|\pi| = \infty$. For $i \leq |\pi|$, π^i denotes the i^{th} state of π, i. e., $\pi^i = s_i$. The i^{th} prefix of a path π is denoted by $\pi{\uparrow^i} = s_0 s_1 \ldots s_i$. The set of finite (infinite) paths starting in state $s \in S$ is called $\mathrm{Paths}_s^{\mathrm{fin}}$ ($\mathrm{Paths}_s^{\mathrm{inf}}$). In order to obtain a probability measure for sets of infinite paths we define a probability space for DTMCs as follows:

Definition 2 *Let* $M = (S, s_I, P, L)$ *be a DTMC and* $s \in S$*. We define a* ***probability space*** $\Psi_s = (\mathrm{Paths}_s^{\mathrm{inf}}, \Delta_s, \Pr_s)$ *such that*

- Δ_s *is the smallest* σ*-algebra generated by* $\mathrm{Paths}_s^{\mathrm{inf}}$ *and the set of basic cylinders of the paths in* $\mathrm{Paths}_s^{\mathrm{fin}}$*. Thereby, for a finite path* $\pi \in \mathrm{Paths}_s^{\mathrm{fin}}$*, the basic cylinder of* π *is defined as* $\mathrm{Cyl}(\pi) = \{\lambda \in \mathrm{Paths}_s^{\mathrm{inf}} \mid \pi \text{ is a prefix of } \lambda\}$.
- $\Pr_s$ *is the uniquely defined probability measure that satisfies* $\Pr_s(\mathrm{Cyl}(ss_1s_2\ldots s_n)) = P(s,s_1) \cdot P(s_1,s_2) \cdot \cdots \cdot P(s_{n-1},s_n)$ *for all basic cylinders* $\mathrm{Cyl}(ss_1s_2\ldots s_n)$.

The properties we want to consider are formulated in PCTL [13] and are of the form $\mathscr{P}_{\leq p}(a \mathsf{U} b)$ with $a, b \in \mathrm{AP}$. This means that the probability to walk along a path from the initial state to a state in which b holds, with all intermediate states satisfying a, is at most p. More formally: A path π satisfies $a \mathsf{U} b$, written $\pi \vDash a \mathsf{U} b$, iff $\exists i \geq 0 : \big(b \in L(\pi^i) \wedge \forall\, 0 \leq j < i : a \in L(\pi^j)\big)$. Note that the all paths sets of this form are measurable w. r. t. the probability measure defined above. A state $s \in S$ satisfies the formula $\mathscr{P}_{\leq p}(a \mathsf{U} b)$ (written $s \vDash \mathscr{P}_{\leq p}(a \mathsf{U} b)$) iff $\Pr_s\big(\{\pi \in \mathrm{Paths}_s^{\mathrm{inf}} \mid \pi \vDash a \mathsf{U} b\}\big) \leq p$. Let us assume that such a formula $\mathscr{P}_{\leq p}(a \mathsf{U} b)$ is violated by a DTMC M. That means $\Pr_{s_I}\big(\{\pi \in \mathrm{Paths}_{s_I}^{\mathrm{inf}} \mid \pi \vDash a \mathsf{U} b\}\big) > p$. In this case we want to compute a *counterexample* which certifies that the formula is indeed violated. Such a counterexample is given by a set of finite paths that start in s_I, satisfy $a \mathsf{U} b$, and whose joint probability exceeds the bound p.

Definition 3 *Let* $M = (S, s_I, P, L)$ *be a discrete-time Markov chain for which the property* $\varphi = \mathscr{P}_{\leq p}(a \mathsf{U} b)$ *is violated in state* s_I*. An* ***evidence*** *for* φ *is a finite path* $\pi \in \mathrm{Paths}_{s_I}^{\mathrm{fin}}$ *such that* $\pi \vDash a \mathsf{U} b$*, but no proper prefix of* π *satisfies this formula. A* ***counterexample*** *is a set* $C \subseteq \mathrm{Paths}_{s_I}^{\mathrm{fin}}$ *of evidences such that* $\Pr_{s_I}(C) > p$.

It is shown in [11] that if $\mathscr{P}_{\leq p}(a \mathsf{U} b)$ is violated then there exists a finite counterexample.
We concentrate on this form of property because it allows to formalize probabilistic safety conditions. Such conditions often concern the probability (p) of reaching an error state (b) when starting in a legal state (a), which is basically the definition of the operators $\mathscr{P}_{\leq p}$ and U.
DTMCs can be extended with *rewards* to model costs of certain operations, to count steps, or to measure given gratifications. The variant of rewards used in [5] are *state rewards*, i. e., a reward is granted when leaving a certain state. Another possibility are *transition rewards*, which grant a

reward upon taking a certain transition in the MRM. We restrict ourselves to state rewards, but the extension to transition rewards is straightforward.

Definition 4 *A discrete-time **Markov reward model** (MRM) is a pair $(M,\boldsymbol{R})$ where $M = (S, s_I, P, L)$ is a DTMC and $\boldsymbol{R} : S \to \mathbb{R}^{\geq 0}$ a real-valued reward function.*

For properties regarding the cost of paths, we define the *accumulated reward* of a path.

Definition 5 *Let $(M,\boldsymbol{R})$ be an MRM, s a state of M and $\pi \in \mathrm{Paths}_s^{\mathrm{fin}}$ a finite path of M. The **accumulated reward** of π is given by $\boldsymbol{R}_{\mathrm{acc}}(\pi) = \sum_{i=0}^{|\pi|-1} \boldsymbol{R}(\pi^i)$.*

We extend the until operator of PCTL to take the accumulated reward of paths into account. This way we can formalize properties like *"The probability to reach a target state with costs larger than c is at most p"*.

Definition 6 *For a (possibly unbounded) interval $\mathscr{I} \subseteq \mathbb{R}$, a path $\pi \in \mathrm{Paths}_{s_0}^{\mathrm{inf}}$ satisfies the property $a \,\mathsf{U}^{\mathscr{I}}\, b$, written $\pi \vDash a \,\mathsf{U}^{\mathscr{I}}\, b$, if there is an $i \geq 0$ such that $b \in L(\pi^i)$, $\boldsymbol{R}_{\mathrm{acc}}(\pi{\uparrow^i}) \in \mathscr{I}$, and $a \in L(\pi^j)$ for all $0 \leq j < i$. A state $s \in S$ satisfies $\mathscr{P}_{\leq p}(a \,\mathsf{U}^{\mathscr{I}}\, b)$ if $\Pr_s\big(\{\pi \in \mathrm{Paths}_s^{\mathrm{inf}} \mid \pi \vDash a \,\mathsf{U}^{\mathscr{I}}\, b\}\big) \leq p$.*

2.2. Stochastic Bounded Model Checking (SBMC)

Stochastic bounded model checking (*SBMC*) was introduced in [19] and extended using SMT (SSBMC) in [5]. We briefly review the basic ideas here.
In symbolic model checking, the probability matrix P of the DTMC under consideration is typically represented as an MTBDD such that the leaf reached by following the path induced by two states s, s' contains the probability $P(s,s')$. This MTBDD can either be translated into a Boolean predicate $T_{\mathrm{SAT}}(s,s')$ such that $T_{\mathrm{SAT}}(s,s')$ is satisfied if $P(s,s') > 0$ [19]. Alternatively, it can be translated into an SMT-formula $T_{\mathrm{SMT}}(s,s',\hat{p})$ over linear real arithmetic such that $T_{\mathrm{SMT}}(s,s',\hat{p})$ is satisfied if $P(s,s') > 0$ and $\hat{p} = \log P(s,s')$ [5]. The logarithm is used to turn the multiplication of probabilities along paths into a summation. In a similar way, the BDDs used to encode the initial state and the labeling with propositions can be translated into formulae $I(s)$ and $L_a(s)$ for $a \in \mathrm{AP}$ such that $I(s)$ is true if s is the initial state, and $L_a(s)$ is true if s is labeled with a.
For computing a counterexample that refutes $\mathscr{P}_{\leq p}(a \mathsf{U} b)$ we remove all out-going transitions of states satisfying $\neg a \vee b$. Then each path ending in a b-state is a valid evidence. The propositional logic formula $\mathrm{SBMC}(k)$ is satisfied for every path $s_0 s_1 \ldots s_k$ of length k that starts in s_I and ends in a b-labeled state:

$$\mathrm{SBMC}(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T_{\mathrm{SAT}}(s_i, s_{i+1}) \wedge L_b(s_k)\ . \tag{1}$$

This formula ignores the actual transition probabilities. They can be taken into account using the following SMT-formula, which is satisfied for evidences of length k with probability of at least p_t:

$$\mathrm{SSBMC}(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T_{\mathrm{SMT}}(s_i, s_{i+1}, \hat{p}_i) \wedge L_b(s_k) \wedge \left(\sum_{i=0}^{k-1} \hat{p}_i \geq \log p_t \right)\ . \tag{2}$$

For reward properties of the form $\mathscr{P}_{\leq p}(a \,\mathsf{U}^{\mathscr{I}}\, b)$ we need to compute the accumulated reward $\mathbf{R}_{\text{acc}}$ within the formula. We translate the MTBDD which represents the reward function into a predicate $R(s,\hat{r})$ such that $R(s,\hat{r})$ is satisfied if state s has reward $\hat{r}$. In the end we have to check whether $\mathbf{R}_{\text{acc}} \in \mathscr{I}$. Then the BMC formula reads as follows:

$$\text{R-SSBMC}(k) := \text{SSBMC}(k) \wedge \bigwedge_{i=0}^{k-1} R(s_i,\hat{r}_i) \wedge \left[\min(\mathscr{I}) \leq \left(\sum_{i=0}^{k-1} \hat{r}_i\right) \leq \max(\mathscr{I})\right] . \qquad (3)$$

Each of these formulae is given to a SAT- (in case of (1)) or SMT-solver (in case of (2) and (3)). A satisfying assignment corresponds to a path of length k which is an evidence for $a \,\mathsf{U}\, b$ or $a \,\mathsf{U}^{\mathscr{I}}\, b$. We collect all found evidences and compute their probability mass. As long as it does not exceed the allowed bound p, we exclude the found path from the search space by adding an additional clause to the clause database of the solver and restart it. If the formula becomes unsatisfiable before exceeding the bound, we increase the value of k and restart. If the property is violated, this procedure terminates after a finite number of steps, yielding a counterexample.

3. Bounded Reward Models

A *discrete-time bounded Markov Reward Model* (*BMRM*) is an MRM with certain modifications: First, it allows also negative rewards. Second, the accumulated reward does not rise above (sink below) a certain threshold *ub* (*lb*). Instead, saturation occurs at the bound.

Definition 7 *A discrete-time* ***bounded Markov reward model*** *(BMRM) is a tuple $(M,\boldsymbol{R},\mathscr{I}_r,r_I)$ such that $M=(S,s_I,P,L)$ is a DTMC, $\boldsymbol{R}: S \to \mathbb{R}$ is a reward function, $\mathscr{I}_r=[lb,ub]$ is a non-empty real interval, and $r_I \in \mathscr{I}_r$ is the initial reward.*

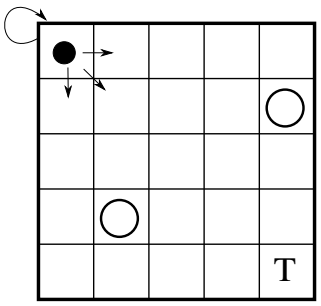

Figure 1: GridBot for $N=5$

An example for such a BMRM is a rechargeable battery: Neither is it possible to charge the battery beyond its maximum capacity nor can its electric charge sink below zero. The initial charging level of the battery corresponds to the initial value r_I of the accumulated reward, which can be interpreted as a continuous charging and discharging process. For a finite path $\pi = s_0 s_1 \ldots s_n$ in a BMRM we define the *accumulated reward* $\mathbf{R}_{\text{acc}}$ inductively as follows: $\mathbf{R}_{\text{acc}}(s_0) = r_I$ and $\mathbf{R}_{\text{acc}}(s_0 \ldots s_i s_{i+1}) = \min\{ub, \max\{lb, \mathbf{R}_{\text{acc}}(s_0 \ldots s_i) + \mathbf{R}(s_i)\}\}$.
For demonstration purposes we created a variant of the well-known GridBot [21], a robot with a rechargeable battery moving in an $N \times N$-grid.

Example 1 *A robot moves through an $N \times N$-grid. It can move vertically, horizontally or diagonally to any adjacent square or stay where it is. The starting square of the robot is in the top left corner, the target T is in the bottom right corner. The probability to move from square to square or to stay is equally distributed, i. e., the probability for staying in a corner or moving to an adjacent square is $1/4$ each, for each square at the boundary of the grid (without the corners) it is $1/6$ and for each interior square $1/9$. Figure 1 shows an illustration of the GridBot model for $N=5$.*

The robot has a rechargeable battery with a capacity of ub. Therefore we define the reward bounds as $\mathscr{I}_r = [0, ub]$. Every step, i. e., moving or staying, costs one unit of energy. Certain randomly chosen squares are charging stations, where the robot can recharge itself as long as it stays there. The energy level increases there by one unit per time-step. In Figure 1 the charging stations are marked by a circle. The starting square is a charging station as well.

The number of charging stations within the grid as well as their respective position is randomly decided upon generation of the model instance. Besides the starting square, at least one square and at most $\lceil N^2/10 \rceil$ of the total number of squares are charging stations. For example, for a model instance with $N = 5$ the number of additional charging stations would lie within 1 and $\lceil 5^2/10 \rceil = 3$. The movement of the robot is completely random. Especially it might walk into a charging station and straight out of it even if its battery is almost empty. Even as the number of squares only increases quadratically with N, the number of possible paths increases exponentially. At the same time the probability of the paths decreases exponentially. This becomes clear if we consider the probability mass of the shortest and most probable path, which runs diagonally from the starting square to the target. Its probability can be computed as $\frac{1}{4 \cdot 9^{N-2}}$, which is an exponentially decreasing function.

3.1. Counterexamples for BMRMs

For BMRMs we are interested in ensuring safety properties like "*The probability is at most 0.1 to reach a non-target state in which the battery charge is zero before having visited a target state*". They cannot be formulated directly in PCTL. Therefore we allow new atomic propositions which compare the current accumulated reward $\mathbf{R}_{\text{acc}}$ with a constant value. The above-mentioned property can then be written as $\mathscr{P}_{\leq 0.1}\big(\neg T \,\mathsf{U}(\mathbf{R}_{\text{acc}} = 0 \wedge \neg T)\big)$, where $T \in \text{AP}$ marks all target states.
A counterexample for such a property is a set of paths each of which starts in the initial state, satisfies the until condition, is minimal in the sense that no path in the counterexample is a prefix of another path, and whose accumulated probability mass exceeds the allowed bound.
In order to generate counterexamples for properties $\mathscr{P}_{\leq p}(\varphi \mathsf{U} \psi)$ we can use the following BMC formula:

$$\begin{aligned}
\text{BR-SSBMC}(k) := \text{SSBMC}(k) \wedge (r_0^{\text{acc}} = r_I) \wedge \Bigg(& \bigwedge_{i=0}^{k-1} R(s_i, \hat{r}_i) \wedge && \text{(4a)}\\
& (r_i^{\text{acc}} + \hat{r}_i \leq ub \vee r_{i+1}^{\text{acc}} = ub) \wedge (r_i^{\text{acc}} + \hat{r}_i \geq lb \vee r_{i+1}^{\text{acc}} = lb) \wedge && \text{(4b)}\\
& (r_i^{\text{acc}} + \hat{r}_i > ub \vee r_i^{\text{acc}} + \hat{r}_i < lb \vee r_{i+1}^{\text{acc}} = r_i^{\text{acc}} + \hat{r}_i) \wedge && \text{(4c)}\\
& \varphi[\mathbf{R}_{\text{acc}} / r_i^{\text{acc}}, a / L_a(s_i)] \Bigg) \wedge \psi[\mathbf{R}_{\text{acc}} / r_k^{\text{acc}}, a / L_a(s_k)] \,. && \text{(4d)}
\end{aligned}$$

The notation $\varphi[\mathbf{R}_{\text{acc}} / r_i^{\text{acc}}, a / L_a(s_i)]$ means that each occurrence of symbol $\mathbf{R}_{\text{acc}}$ in the formula φ is replaced by r_i^{acc} and each occurrence of the atomic proposition $a \in AP$ by the formula $L_a(s_i)$.
The expression $R(s_i, \hat{r}_i)$ assigns the reward of state s_i to variable $\hat{r}_i$. The clauses (4b)–(4c) are responsible for assigning the accumulated reward up to state s_i to variable r_i^{acc}. The last part (4d) takes care that all intermediate states $s_0 \ldots s_{k-1}$ satisfy φ and the last state satisfies ψ.

Table 1: Counterexample generation for the GridBot model

N	$\mathscr{I}_r$	p	k	#paths	memory	time
3	$[0,3]$	0.3	7	2848	52.96	6.34
4	$[0,4]$	0.3	9	25255	376.75	346.76
5	$[0,5]$	0.2	10	40590	711.55	862.16
6	$[0,9]$	0.01	12	3881	150.25	34.15
7	$[0,10]$	0.01	14	13731	596.03	297.43
8	$[0,11]$	0.001	13	159	31.32	3.26
9	$[0,12]$	0.001	15	682	129.20	72.51
10	$[0,13]$	0.001	16	6247	297.36	111.41
11	$[0,14]$	0.001	18	20693	1121.33	801.55
12	$[0,15]$	0.0001	18	895	227.75	85.24

3.2. Experimental Results

We incorporated our modifications into the existing SBMC tool [5] and created instances of the GridBot model for different values of N and reward interval $\mathscr{I}_r$. These instances were analyzed under the property $\mathscr{P}_{\leq p}\big(\neg T \,\mathsf{U}\, (\mathbf{R}_{\text{acc}} = 0 \wedge \neg T)\big)$, with T being the target state in the bottom right corner, similar to Figure 1. The property asks whether the probability that the battery becomes empty before reaching the target state is at most p. The experiments were done on a Dual Core AMD Opteron processor with 2.4 GHz per core and 16 GB of memory.

The first two columns of Table 1 contain the values for N and the capacity interval $\mathscr{I}_r$. The initial charging level of the robot was $\max(\mathscr{I}_r)$ for all experiments. The third column contains the probability threshold p of the property, k in the fourth column designates the maximum length of the considered paths. The columns "#paths", "memory", and "time" contain the number of paths in the generated counterexample, the memory consumption in megabytes and the computation time in seconds.

As the value of N increases, the probabilities of the generated paths sink dramatically. This is reflected in the probability threshold p and in the fact that more and more paths are needed for exceeding p (s. column "#paths"). However, we are still able to generate a counterexample within an acceptable amount of time and memory.

4. Compositionality

In many description languages, e. g., the widely used language of the model checker PRISM [14], DTMCs are specified as a *composition* of several modules, each of these modules being a DTMC in its own right. For example, the leader election protocol [15] consists of N processors in a synchronous ring. Each of these processors is modeled as a separate module. An additional counter module measures how many rounds it takes until a leader is elected.

Model checking a modular DTMC typically requires to compute the composition first, yielding a single model for the whole system. Model checking then reduces to solving an equation system whose size is linear in the number of states and transitions of the composition. The SBMC tool [19, 5] also works on this large model to perform bounded model checking.

To enable synchronized execution of transitions, DTMCs are extended with *action labels*, i. e., a DTMC is now a tuple $M = (S, s_I, Act, P, L)$ with a finite set Act of action labels. The transition

probability function is $P : S \times Act \times S \to [0,1]$ such that $\sum_{\alpha \in Act} \sum_{s' \in S} P(s,\alpha,s') = 1$ for all $s \in S$. The composition of modules is defined in the usual CSP-like manner: Let $M_j = (S_j, s_{I,j}, Act_j, P_j, L_j)$ for $j = 1,\ldots,m$ be a set of DTMCs. If a transition labeled with α is executed in one module M_j, all modules M_i with $\alpha \in Act_i$ also have to execute a transition labeled with α. If this is not possible, a deadlock occurs, which is typically considered a modeling error. All other modules M_i with $\alpha \notin Act_i$ do not change their state.

In order to ensure that the composition is again a DTMC (and not a Markov decision process), all outgoing transitions of a specific state within a module have to carry the same action label. The nondeterminism introduced by the composition of the modules is resolved by taking a uniform distribution over all nondeterministic choices.

For example, let us take a look at the synchronous leader election protocol as it is presented on the PRISM website `http://www.prismmodelchecker.org`: All the processor modules draw randomly a number from a set $\{1,\ldots,K\}$. This choice is synchronized through the action label "pick" and takes place at the same time in every module. Within the modules, all outgoing transitions of a specific state are labeled with the same action label, i. e. if an outgoing transition of a state carries the action label "pick", then all other outgoing transitions of this state carry this label as well.

We now modify the SBMC procedure for modular DTMCs such that the formula is constructed from the modular description without building the composition beforehand. Assume we have DTMCs $M_j = (S_j, s_{I,j}, Act_j, P_j, L_j)$ for $j = 1,\ldots,m$ and that the transitions of each module M_j are partitioned according to the action labels. That means for each $j = 1,\ldots,m$ and each $\alpha \in Act_j$ we have a formula $T_{j,\alpha}$, which is satisfied if the transition from s to s' is labeled with α and has a positive probability. We set $Act = \bigcup_{i=1}^{m} Act_i$ and assume that $T_{j,\alpha}(s,s') = \text{false}$ for $\alpha \in Act \setminus A_j$. We modify each module such that for each action $\alpha \notin Act \setminus A_j$ we have a self-loop on each state, i. e.,

$$\hat{T}_{j,\alpha}(s,s') := \begin{cases} T_{j,\alpha}(s,s') & \text{if } \alpha \in Act_j \\ s = s' & \text{if } \alpha \notin Act_j\ . \end{cases} \tag{5}$$

This formula can be used to generate the BMC formula, describing the evidences of length k:

$$\text{C-SBMC}(k) := I(s_0) \wedge \bigwedge_{i=0}^{k-1} \big(L_a(s_i) \wedge \neg L_b(s_i)\big) \wedge L_b(s_k) \wedge \tag{6a}$$

$$\bigwedge_{i=0}^{k-1} \Big[\bigvee_{\alpha \in Act} \Big(\bigwedge_{j=1}^{m} \hat{T}_{j,\alpha}(s_{j,i}, s_{j,i+1}) \Big) \Big] \tag{6b}$$

s_i indicates the composite state for time frame i, whereas $s_{j,i}$ denotes the state for time frame i in module j.

C-SBMC(k) is satisfied for a path $s_0 s_1 \ldots s_k$ through the composed DTMC if it starts in the initial state and satisfies $a \mathsf{U} b$. This is ensured in line (6a). The condition that there is a transition from state s_i to s_{i+1} which follows the synchronization rules is given in (6b): We encode the transition relation for each unrolling separately; this leads to the left-most conjunction. In each step, one action label needs to be chosen. Since the transition sets $T_{j,\alpha}(s,s')$ are pairwise disjoint, a disjunction over all actions in Act suffices to encode the selection of an action label. Once an action has been determined, each of the modules M_j has to make a step: If the selected action $\alpha \in Act_j$, a transition

from $T_{j,\alpha}(s,s')$ has to be executed; otherwise the state of M_j does not change. This is exactly captured in the formula $\hat{T}_{j,\alpha}(s,s')$.
The probabilities of the transitions are not considered within Formula (6), as was the case in the original SBMC [19]. The probability of the path π can be computed by tracing the computed path through the individual components.
Using Formula (6) instead of (1) it is possible to reduce the number of clauses in the resulting CNF, which often reduces the memory consumption and speeds up the computation. We demonstrate this in the following on some experimental results.

4.1. Experimental Results

We incorporated our modifications in the SAT-based version of the SBMC tool from [19, 5] and compared the performance of our modified version with the original tool. For our experiments we used the following benchmarks:
(1) The *leader election protocol* [15] consists of N processors in a synchronous ring. Each processor chooses a random number from $\{0,\ldots,K\}$ and passes its number along the ring. The processor with the highest unique number becomes the leader if a unique number exists. If this fails, a new round starts. We provide a certificate that a leader will finally be elected.
(2) The *contract signing protocol* [9, 17] provides a fair exchange of signatures between two parties A and B over a network. If B has obtained the signature of A, the protocol ensures that A will receive the signature of B. For our experiments we examine the possibility that this property is violated.
(3) The *probabilistic broadcast protocol* [10] models information passing within an $N \times N$-grid. A source node broadcasts a message to its neighbors. The neighbors decide randomly whether to pass along the information or not. It is possible to include several mechanisms to influence the message passing: Message collision, lossy channels and random delays. We examine the possibility that the message is received by the node which is furthest away from the original sender.
All experiments were done on the same computer as in Section 3.2. Any computation which took longer than four hours ("– TO –") or needed more than 4 GB of memory ("– MO –") was aborted. Table 2 shows an excerpt of our results. The first two columns contain the name of the benchmark instance and the probability threshold p. Column three to six contain the result for our modified SBMC, the remaining columns contain the results for the original SBMC. We used the loop optimization from [19] in all experiments. "#paths" refers to the number of found paths, "#clauses" to the number of clauses in the CNF at the beginning of the path search. Note that the number of clauses increases during the iteration process. The columns labeled with "memory" and "time" state the memory consumption in megabytes and the running time in seconds.
For the contract benchmarks the overhead of the modular CNF generation unfortunately slows the computation down by a small factor. The reduction in the number of clauses does not fully compensate the additional effort for computing the modular formula. For the leader election and the broadcast protocol benchmarks, however, we can observe a considerable speed-up and reduction of the memory consumption. Some benchmarks instances are not even manageable otherwise with the available resources.

Table 2: Counterexample generation for DTMCs with and without modules

		modular SBMC				**standard SBMC**			
Model	p	#paths	#clauses	memory	time	#paths	#clauses	memory	time
leader04_06	0.90	1167	36972	101.46	28.68	1167	177938	584.28	274.13
leader04_10	0.90	9001	238949	730.00	1663.69	– MO –			
leader04_12	0.90	18663	448194	1422.82	9219.34	– MO –			
leader05_05	0.90	2813	93084	244.57	288.78	2813	453684	2880.41	2516.22
leader05_06	0.90	6999	187982	539.07	1600.16	– MO –			
leader08_02	0.90	16	10208	32.49	91.97	16	24152	84.82	144.36
leader09_02	0.90	18	14039	47.83	271.97	18	35179	132.43	514.85
contract06_07	0.50	2049	59846	540.66	2311.00	2049	88810	959.77	2282.89
contract07_03	0.50	8193	27996	185.98	1068.51	8193	35201	257.16	681.74
contract07_07	0.50	8193	80474	900.75	14008.80	8193	118937	1498.91	11487.60
contract08_02	0.50	32769	29732	297.16	2661.18	32769	32769	294.07	1282.20
contract08_03	0.50	32769	33180	424.40	4036.42	32769	38228	473.64	3604.38
brp_no_coll_4	0.60	3530	2903	18.97	2.21	3415	5629	22.84	7.30
brp_no_coll_5	0.60	593934	20110	1362.68	6086.96	– TO –			
brp_coll_4	0.20	1148	5357	24.93	2.46	1217	10995	27.51	10.65
brp_coll_5	0.30	121033	87213	358.57	11111.80	– TO –			
brp_coll_del_3	0.30	536428	3143	943.94	764.18	691225	8024	1205.05	4640.65
brp_coll_los_3	0.50	498304	21819	1209.18	10110.30	– TO –			

5. Conclusion

In the first part of our paper we presented bounded reward models and showed how to generate counterexamples for them on the basis of the SSBMC procedure. For this we created a case study which we used for our experiments. These first results look promising.
In the second part of the paper we explored the possibilities of modular DTMCs and modified the existing SBMC tool in order to do a compositional path search. This reduces the memory consumption of the counterexample generation and also achieves a considerable speed-up for some benchmark sets, as we have shown with our experimental results.
In future we plan to do a more detailed experimental evaluation of our methods for BMRMs, a task which mostly requires more appropriate benchmarks. For the compositional counterexample generation we have to investigate the possibility to combine this with the SMT-based version of SBMC. At the moment only the path search is compositional, for the computation of the actual path probability we use the composition of all the modules. Finally we want to combine the SBMC tool with the critical subsystems of [16]. It should be possible to use the found paths to generate a critical subsystem of the DTMC under consideration. This subsystem would have a higher probability mass than the sum of the probabilities of the found paths, thus reducing the number of required paths.

References

[1] Erika Ábrahám, Nils Jansen, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. DTMC model checking by SCC reduction. In *Int'l Conf. on Quantitative Evaluation of Systems (QEST)*, pages 37–46. IEEE CS, 2010.

[2] Husain Aljazzar and Stefan Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Transactions on Software Engineering*, 36(1):37–60, 2010.

[3] Miguel E. Andrés, Pedro D'Argenio, and Peter van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Haifa Verification Conference*, volume 5394 of *LNCS*, pages 129–148. Springer, 2008.

[4] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[5] Bettina Braitling, Ralf Wimmer, Bernd Becker, Nils Jansen, and Erika Ábrahám. Counterexample generation for Markov chains using SMT-based bounded model checking. In *Int'l Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS) and Int'l Conf. on FORmal TEchniques for Networked and Distributed Systems (FORTE)*, volume 6722 of *LNCS*, pages 75–89. Springer, 2011.

[6] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[7] Edmund M. Clarke. The birth of model checking. In *25 Years of Model Checking – History, Achievements, Perspectives*, volume 5000 of *LNCS*, pages 1–26. Springer, 2008.

[8] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[9] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[10] Ansgar Fehnker and Peng Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *Int'l Conf. on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW'06)*, volume 4104 of *LNCS*, pages 128–141. Springer, 2006.

[11] Tingting Han and Joost-Pieter Katoen. Counterexamples in probabilistic model checking. In *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 72–86. Springer, 2007.

[12] Tingting Han, Joost-Pieter Katoen, and Berteun Damman. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, 35(2):241–257, 2009.

[13] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[14] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[15] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[16] Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. Hierarchical counterexamples for discrete-time Markov chains. In *Int'l Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 6996 of *LNCS*, pages 443–452. Springer, 2011.

[17] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[18] Ralf Wimmer, Bernd Becker, Nils Jansen, Erika Ábrahám, and Joost-Pieter Katoen. Minimal critical subsystems for discrete-time Markov models. In *Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7214 of *LNCS*, pages 299–314. Springer, 2012.

[19] Ralf Wimmer, Bettina Braitling, and Bernd Becker. Counterexample generation for discrete-time Markov chains using bounded model checking. In *Int'l Conf. on Verification, Model*

Checking, and Abstract Interpretation (VMCAI), volume 5403 of *LNCS*, pages 366–380. Springer, 2009.

[20] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for refuting ω-regular properties of Markov decision processes (extended version). Reports of SFB/TR 14 AVACS 88, 2012. ISSN: 1860-9821, available at `http://www.avacs.org`.

[21] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *Software Tools for Technology Transfer*, 8(3):216–228, 2006.