Advances in Symbolic Probabilistic Model Checking with PRISM*

Extended version (2016-02-18)

Joachim Klein, Christel Baier, Philipp Chrszon, Marcus Daum,
Clemens Dubslaff, Sascha Klüppelholz, Steffen Märcker, and David Müller

Institute of Theoretical Computer Science
Technische Universität Dresden, 01062 Dresden, Germany

Abstract. For modeling and reasoning about complex systems, symbolic methods provide a prominent way to tackle the state explosion problem. It is well known that for symbolic approaches based on binary decision diagrams (BDD), the ordering of BDD variables plays a crucial role for compact representations and efficient computations. We have extended the popular probabilistic model checker PRISM with support for automatic variable reordering in its multi-terminal-BDD-based engines and report on benchmark results. Our extensions additionally allow the user to manually control the variable ordering at a finer-grained level. Furthermore, we present our implementation of the symbolic computation of quantiles and support for multi-reward-bounded properties, automata specifications and accepting end component computations for Streett conditions.

1 Introduction

One prominent approach to cope with the well-known state-explosion problem in model checking is the use of symbolic methods based on binary decision diagrams (BDDs) [8,31]. Various BDD-variants have been studied and implemented in tools for the quantitative analysis of probabilistic systems, see, e.g.,[17,4,18,34,19,32,23,28,10]. The prominent probabilistic model-checker PRISM [34,26,35] uses symbolic approaches relying on a multi-terminal binary decision diagram (MTBDD) [3,15] representation of the model. Among others, PRISM provides support for modeling and the analysis of discrete-time Markov chains (DTMC) and Markov decision processes (MDP) as well as continuous-time Markov chains (CTMC) against temporal logical specifications. While the behavior of Markov chains is purely probabilistic, MDPs exhibit both probabilistic and nondeterministic choices. The typical task of the analysis of MDPs is to compute a scheduler for resolving the nondeterminism that maximizes or minimizes the probability for a given path property or an expectation. The symbolic implementation of PRISM comes in three flavors: a purely symbolic engine MTBDD

* The authors are supported by the DFG through the collaborative research centre HAEC (SFB 912), the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED and Institutional Strategy), the Research Training Groups QuantLA (GRK 1763) and RoSI (GRK 1907), the DFG/NWO-project ROCKS, and Deutsche Telekom Stiftung.

and two semi-symbolic engines, called HYBRID and SPARSE. The MTBDD engine performs all computations using MTBDDs, while the HYBRID engine combines the MTBDD-based representation of the transition matrix of the model with an explicit representation of the solution state value vector in the computation [24]. The latter is motivated by the observation that the MTBDD representation of such state value vectors can be of substantive size even for models with compact MTBDD representation during probabilistic model-checking algorithms. The SPARSE engine constructs an explicit, sparse matrix from the MTBDD-based transition matrix for numerical computations and performs computations using this explicit representation. In addition to the three symbolic engines, which rely internally on the infrastructure of the C-based CUDD library [37] for MTBDD storage and manipulation, the fourth engine, EXPLICIT, is fully implemented in Java, builds an explicit representation of the reachable state space of the model and carries out all analysis on this explicit representation. Depending on the concrete model structure and size, each of the four engines has situations where it can show its particular strength.

It is well known that the variable order of the BDD variables plays a crucial role for obtaining a compact representation of the model and for model checking performance. PRISM provides limited influence on the variable order, mainly during modeling by the order in which individual modules are placed in the model file and the order of the individual state variables inside a module. While care has been taken to use a sensible variable order derived from the structure of elements in the model file [34], PRISM lacks any support for automatically identifying a good variable order using techniques such as sifting [36,33], which are routinely employed in symbolic model checkers for non-probabilistic systems (e.g., [11]). In our previous work on complex case studies, we have reached several times the point where we had to resort to manually swapping the module and variable definitions in the model file to try and find a better ordering, in particular for models where explicit approaches were infeasible (e.g., [13]).

Contribution. The main purpose of this paper is to present several refinements of PRISM's symbolic engines. First, we added support for the *automated variable reordering* of the MTBDD-based model representation by enabling CUDD's implementation of group sifting and by extensions of PRISM's input modeling language that allow to rearrange and interleave the orders of the bits of state variables within the same module as well as (the bits) of state variables of different modules. The impact of the automated reordering has been evaluated using the examples from the PRISM benchmark suite [27] and in the context of the symbolic quantile computations. Our second contribution are symbolic implementations of computation schemes for *cost-* or *reward-bounded reachability properties* in discrete Markovian models (DTMCs or MDPs) and corresponding *quantiles*.[1] The latter are, e.g., useful to compute the minimal energy budget required to ensure a 90% chance for completing a list of jobs. Algorithms for the computation

[1] While PRISM supports the computation of expected costs or rewards and probabilities for step-bounded properties, it does not contain implementations of algorithms for computing probabilities for reachability conditions with cost/reward constraints.

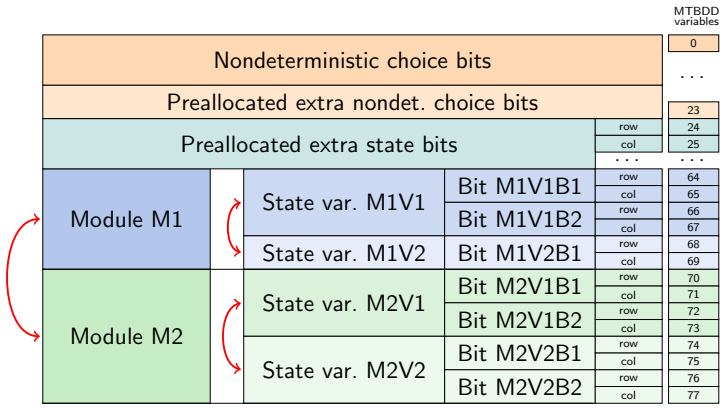

Fig. 1. Schema for the standard variable ordering used by PRISM. The arrows indicate the effect of syntactic reordering in the PRISM model file on the variable order.

of quantiles have been presented in [38,5] and prototypically implemented using (non-symbolic) explicit representations of the model. Within this paper, we report on the results of comparative experimental studies of the explicit and the new symbolic implementation. The third contribution are enhancements of PRISM's engines for the *automata-based analysis* of DTMCs and MDPs. This includes the treatment of Streett acceptance conditions in MDPs (PRISM only offers engines for Rabin acceptance and its generalized variant) and an extension of PRISM's property syntax for automata-specifications (rather than LTL-specifications).

Outline. Section 2 presents our new approaches for variable reordering in PRISM. Section 3 summarizes the main features of our implementations for cost/reward-bounded properties and quantiles, while Section 4 presents the automata-based extensions. For further details (implementation, experiments) and an extended version [21] see `http://wwwtcs.inf.tu-dresden.de/ALGI/PUB/TACAS16/`. We are collaborating with PRISM's authors to integrate our extensions into the main PRISM version and would like to thank David Parker for fruitful discussions.

2 Automatic variable reordering in PRISM

Here, we will briefly describe the relevant infrastructure in PRISM for dealing with variable ordering. The MTBDD variable ordering of the symbolic model representation is determined by the order of module and variable definitions in the PRISM model file. Fig. 1 sketches the general schema.[2] In a first block,

[2] The depicted scheme corresponds to the default ordering for the Hybrid and Sparse engines. There are subtle differences when using the Mtbdd engine, discussed in appendix A. Additionally, standard PRISM preallocates only extra *state* variables, mainly for the product with deterministic automata. To support generic symbolic model transformations, we also preallocate choice variables, i.e., for fresh actions in the transformed MDP.

MTBDD variables for nondeterministic choices are allocated. This includes a unary encoding of the synchronizing actions (i.e., one MTBDD variable for each action), scheduling variables (one MTBDD variable indicating that a given module is active) as well as several bits for representing local choices, e.g., between alternative commands for the same synchronizing action. Then, two blocks of extra variables are preallocated to serve in later model transformations, e.g., during a product construction with a deterministic ω-automaton for LTL model checking. For each individual bit of a state variable in the model, two MTBDD variables are allocated, one serving in the representation of the rows and one for the columns of the transition matrix. The MTBDD variables for representing the possible values of the (integer-valued) state variables are allocated in the order in which they appear in the PRISM model file, with each state variable forming a block of row/column pairs. The bits for each state variable are ordered from most-significant to least-significant. Global state variables are treated as if they were contained in a single module located before the "real" modules.

The arrows in Fig. 1 indicate the extent of the influence that can be applied to the variable ordering by syntactically reordering the PRISM source file: At the highest level, the order of the modules can be changed. Additionally, inside each module, the order of the definition of the state variables can be changed. Note that such changes of the ordering in the PRISM model file do not lead to any semantic changes in the model, but can lead to cosmetic changes, e.g., in the order of the states for exported models. To complement the manual, trial-and-error approach for finding a good order in the model file, we detail our automatic approach in the next section.

2.1 Automatic variable reordering using group sifting

PRISM internally relies on the CUDD (MT)BDD library [37] for the management of a set of BDDs that arise during probabilistic model checking. CUDD provides implementations of several heuristics for (dynamic) variable reordering which in principle should be available to be used by PRISM. Unfortunately, the implementation of PRISM heavily relies on the assumption that the variable ordering of the MTBDD does not change at all. The order of the MTBDD variables is assumed to correspond with the order of the respective variables in the underlying PRISM model, i.e., that the variable index (logical index) and the variable level (index in the current variable order) need to agree. Eliminating this restriction on the variable order would require a substantial refactoring of PRISM's infrastructure, touching many parts of the implementation.

Our approach presented in this section makes automatic variable reordering available to a PRISM user while avoiding any substantial refactoring of PRISM's infrastructure. First, a symbolic, MTBDD-based representation of the model is built by PRISM as usual. After the model is built, we trigger the group sifting reordering heuristic [36,33] via the CUDD library, using several variable grouping constraints that will be detailed later. After this reordering, the MTBDD-based model representation violates PRISM's assumptions, which renders further computations in PRISM impossible. Thus, we perform an analysis of the variable

ordering found by group sifting and translate the changes in variable locations back to the source level of the PRISM model. This way, we obtain a syntactically reordered PRISM model, where the placement of the PRISM modules und state variables reflects the calculated variable ordering. Our implementation then allows using this reordered model directly after the reordering computation via the following trick: After reordering, we delete the MTBDDs of the model and reset the variable ordering in CUDD to the one that PRISM expects, where each variable index corresponds to the variable level in the BDD. Then, we build the BDDs for the model a second time, this time using the syntactically reordered PRISM model. We thus obtain the reordered model again, but now with the underlying assumptions of PRISM intact, allowing to use the full PRISM machinery. This approach provides transparent and convenient access to the reordering functionality to the user. Additionally, we also support exporting the reordered model to a file, which can then be used in future PRISM runs. This way, the time for reordering can be amortized over multiple model-checking runs.

For this approach to work, it is crucial that we are able to seamlessly convert between the reordered variable ordering obtained after sifting and the variable order that is induced by syntactically reordering the elements of the PRISM model file. To achieve this, we introduce appropriate groups of MTBDD variables represented by a tree structure and used in the groups sifting. The grouping reflects the structure of the given model file: Each PRISM module forms a group of BDD variables that can be reordered as a block. This corresponds to syntactically changing the order of modules in the model file. Additionally, inside each module, the MTBDD variables for each state variable form another group. Reordering those groups corresponds to changing the order of the variable declarations inside a PRISM module. The remaining variables, e.g., those for nondeterministic choices remain in fixed positions. Hence, the above approach allows for creating all variable orders that can result from permutations of modules and state variables within the PRISM model file. In the next section we show how a more fine-grained control can be achieved.

2.2 Bit-level control over the variable order using views

Although it is well known that for some operators, e.g., the addition of two integers, an efficient representation relies on the interleaving of the individual bit-variables, there is no way of interleaving the individual bits of multiple state variables in PRISM up to now.

Our implementation provides the option of syntactically "exploding the bits" of all the state variables in a PRISM model file: Each multi-bit state variable s is replaced with the appropriate number of single-bit variables s_i. To keep this transformation simple and transparent to the user we introduce a syntactic enhancement of the PRISM modeling language called a *view*. A view forms a virtual variable s over bit variables s_j. This virtual variable can be used in guards and updates of transition definitions just as ordinary variables. Hence, exploding the bits does not affect any of the transition definitions given in the model file.

```
module M
   s_bit_2 : [0..1];
   s_bit_1 : [0..1];
   s_bit_0 : [0..1];
   s : view (s_bit_2,s_bit_1,s_bit_0) <=> [2..7] init 3;
   [inc] s<7 -> 1:(s'=s+1);
endmodule
```

Fig. 2. Defining a view `s` with data domain $(2, 7)$ from three single-bit state variables.

As an example, consider the PRISM module in Fig. 2. Here, the virtual state variable `s` with an integer data domain of $2 \leqslant \mathtt{s} \leqslant 7$ requires three bits to represent all values, as internally integer variables are encoded by first subtracting the lower bound of the data domain (2 is internally represented as 0, etc.). The actual storage is provided by the three single-bit state variables `s_bit_`i. The order of the single-bit state variables in the view definition determines their use in the encoding, with the most-significant bit appearing first. As can be seen, the virtual view variable `s` is being used just like a standard PRISM state variable.

Note that "exploding the bits" of a PRISM model file alone will not change the variable ordering and MTBDD representation, as the encoding and ordering of the newly introduced single-bit state variables correspond to the standard encoding used for the original variables. When applying the automatic reordering detailed in the previous section to an "exploded" model file, the individual bits of the state variables can be now be sifted and interleaved, as their grouping is removed. However, the MTBDD variables are still restricted from crossing module boundaries. We detail how to remove this restriction in the next section.

2.3 Interleaving state variables of different modules

To overcome the limitation that state variables cannot be interleaved across modules our implementation provides the option of "globalizing" all state variables in a PRISM model file: Each state variable inside a PRISM module is moved from the module to become a global variable, while keeping the order they appeared in the original model file. Realizing this requires to loosen some restrictions on the use of global variables imposed by PRISM. In standard PRISM, global variables cannot be updated in synchronous actions, as this has the potential of resulting in conflicting updates from multiple modules. We removed this restriction, as in our setting only the "previous owner" of a variable, i.e., the module in which the variable was initially declared, will update the global variable in the transformed model. This ensures that there can be no conflicting updates introduced by globalizing variables. Our implementation supports such global variable updates for similar situations as well, i.e., where it is apparent by a syntactic inspection that no conflicting updates can happen.

The options for exploding the bits and globalizing the variables can be used separately and in a combined fashion (cf. Fig. 3) and the resulting model yields

```
module M1
  x : [0..3] init 0;
 [a] true ->   0.5:(x'=0)
             + 0.5:(x'=y);
endmodule

module M2
  y : [0..3] init 0;
 [a] true -> 1:(y'=0);
 [b] y<3  -> 1:(y'=y+1);
endmodule
```

```
global x_bit_1 : [0..1];
global x_bit_0 : [0..1];
global y_bit_1 : [0..1];
global y_bit_0 : [0..1];
global x :
  view (x_bit_1,x_bit_0) <=> [0..3] init 0;
global y :
  view (y_bit_1,y_bit_0) <=> [0..3] init 0;

module M1
  [a] true -> 0.5:(x'=0) + 0.5:(x'=y);
endmodule

module M2
 [a] true -> 1:(y'=0);
 [b] y<3  -> 1:(y'=y+1);
endmodule
```

Fig. 3. Example of both "exploding bits" and "globalizing variables" for a PRISM model file (before on the left, after on the right).

a starting point for group sifting. This way, fine-grained control of the variable ordering for all state variables in the model becomes possible. Within the following section we will evaluate our implementation by means of a number of case studies.

2.4 Benchmarking automatic variable reordering of PRISM models

To explore the effect of automatic variable reordering using our implementation, we performed benchmarks using the DTMC, CTMC and MDP models in the PRISM benchmark suite [27]. The models are parametrized in various parameters, affecting both the number of states and the size of the MTBDD representation. In total, we performed benchmarks with 208 model instances (70 DTMCs, 70 CTMCs, 68 MDPs). We present here statistics for the "top" initial variable ordering [34] used by default in the HYBRID engine. Results using the default variable ordering of the MTBDD engine were roughly similar.

Fig. 4 presents statistics for the basic case, i.e., reordering without any syntactic transformations beforehand. Similar plots for reordering with the "globalize variables" (Sec. 2.3) and "explode bits" (Sec. 2.2) transformations being applied can be found in the appendix.[3] In the plots, the model instances are grouped by their base model. The size of the MTBDD refers to the number of nodes in the shared MTBDD structure storing the various individual MTBDDs. Those individual MTBDDs represent the model in PRISM, i.e., its transition matrix, a 0/1-version of the transition matrix representing the underlying graph structure

[3] The benchmarks for reordering were carried out on a machine with two Intel Xeon L5630 4-core CPUs at 2.13GHz and 192GB RAM, with a timeout of 1 hour and a CUDD memory limit of 10GB. The max-growth factor of CUDD was set to 2, i.e., allowing a doubling in MTBDD size before sifting is abandoned.

of the model, the set of reachable states, representations for the transition and state rewards.

As can be seen in the second plot from the top in Fig. 4, the automatic reordering was able to achieve significant reductions for many of the model instances. As a particularly striking example, the reordering was very effective for the "mapk-cascade" model, a CTMC: For the instance with parameter $N = 8$, the MTBDD size was reduced from 1,478,511 nodes to 96,718 nodes, a reduction of more than 90%. The time for building the symbolic representation of this model instance was reduced from 174 s to 2 s for the reordered model. Most of the time, the reduction in the MTBDD size is accompanied by a reduction in the time needed for building the MTBDDs for the reordered model. The major outlier to this are several instances of the "crowds" model, where the time for building the reordered model was substantially worse compared with the original model. Our investigation revealed that this is due to the point in time at which our reordering is performed, i.e., after the symbolic transition matrix has been restricted to the reachable part of the state space, which is the symbolic representation that is then used for the actual model checking. The reordering heuristic thus produced a variable order tailored for this state space and which is not particularly suitable for the representation of the individual, not yet restricted parts of the model used during the building phase. This is a classic example of the case where an asynchronous reordering, i.e., continuously adapting the variable ordering during the construction phase, would be helpful.

In general, the time for reordering tends to be related to the size of the MTBDD before reordering, as expected. As noted above, even substantial reordering times might be worthwhile, as the reordered model can be stored and subsequently reused multiple times, profiting, e.g., from the reduced build time and more compact symbolic representation.

There were three models ("brp", "nand" and "poll"), where instances exhibited an overall reduction in the size of the MTBDD, but an increase in the size of the MTBDD for the transition matrix alone (in all cases the increase was less than 10%). This is explained by the fact that the reordering operates on the whole shared MTBDD data structure and thus does not necessarily optimize all the individual MTBDD functions that are stored.

We have also benchmarked the effect of our syntactic transformations on the automatic reordering and present here (Table 1) some notable examples. For further, detailed statistics we refer to the appendix. As already seen in Fig. 4, the "tandem" model has no reduction in MTBDD size when it is reordered as-is. However, when the state variables are "exploded", reordering becomes profitable, with additional reductions when combined with the "globalize variables" transformations. Globally, for every model instance from the benchmark suite, at least one of the variants achieved some reduction. As is to be expected, no variant is uniformly best. Consider the statistics for the "cluster" model in Table 1. For $N = 32$, "exploding" and "globalizing" are individually successful, but in combination lead to only minor reductions. For $N = 256$, "exploding" is in the lead, while for $N = 512$, "globalizing" by itself leads to the most reductions. For

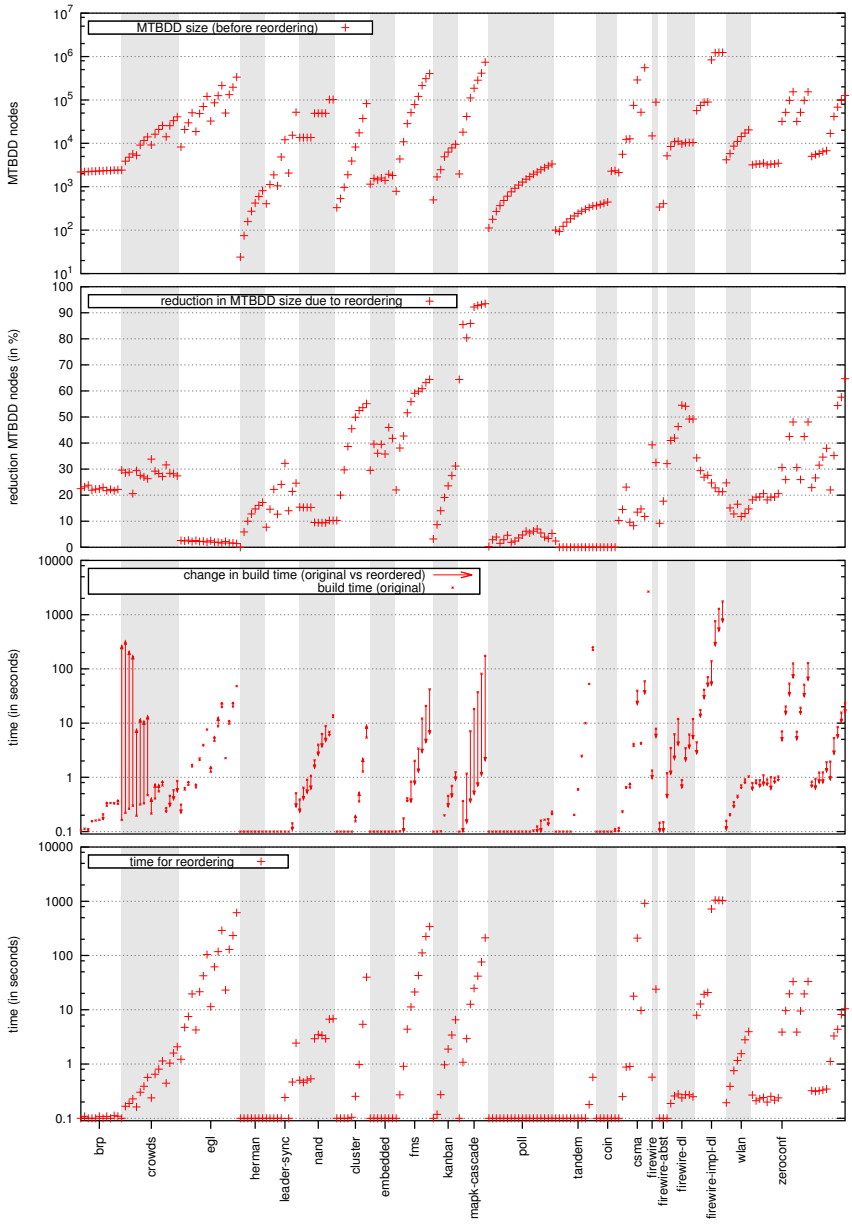

Fig. 4. Statistics for reordering without syntactic transformations: The number of MTBDD nodes before reordering, the reduction (larger numbers represent more reduction) in the number of MTBDD nodes, the change in time for building the model (before/after reordering) and the time spent reordering. Times below 0.1 seconds are clipped to 0.1 for visualization purposes. There was one timeout, reordering the "csma4_6" instance (45 minutes of the 1 hour timeout spent on building, with 3,589,198 nodes).

Table 1. Selected statistics for the reduction achieved using reordering on the standard model instance and where the "explode bits" and "globalize variables" transformations were applied. In the last column, both transformations are applied. For reference, the MTBDD size before reordering is included as well. For full details, see appendix.

	MTBDD	reduction in %			
Model instance	before	standard	explode	globalize	exp.+glob.
tandem c=255	4 917	0.0	26.0	0.0	35.1
tandem c=4095	103 233	0.0	35.7	0.0	64.3
cluster N=32	7 391	45.5	47.9	52.2	8.3
cluster N=256	61 749	53.6	58.7	24.2	41.4
cluster N=512	132 908	55.1	59.3	61.5	46.2
kanban t=6	14 001	27.5	34.0	1.2	32.3

"kanban" with $t = 6$, "globalizing" alone leads to worse reductions than reordering on the standard model. As can be seen, it remains an area of experimentation to select the reordering variant that is a good fit for a particular model and model instance. As a good first assumption, the time for model checking tends to be related in general to the compactness of the symbolic representation. We experimented with some of the properties in the benchmark suite (cf. appendix B for some examples). In the next section, we will additionally report on significant reductions in model-checking time in the context of quantile computations with reordered models.

3 Computing Quantiles for Markov Decision Processes

Models in PRISM can be annotated with rewards (non-negative values) specifying the costs or the gain for visiting certain states or taking certain transitions. PRISM provides implementations of algorithms for reasoning about expected rewards, but lacks support for computing the probabilities for reward-bounded path properties, unless for unit-reward functions that count the number of steps. We have extended PRISM with support for the computation of (extremal) probabilities of cost-/reward-bounded simple path formulas for DTMCs and MDPs with non-negative integer rewards, e.g., of $\Pr^{\max}(\Diamond^{\leq r}\, \Phi)$ for a reward bound r. This includes conjunctions of multiple reward bounds and step bounds [1], relying on a product transformation with a counter automaton tracking the accumulated reward. This is implemented for both the explicit and symbolic engines.

In our recent work [38,5], we addressed the computation of quantiles for probability constraints on reward-bounded reachability conditions and carried out experiments with a prototypical implementation based on PRISM's Explicit engine. In the mean time, this implementation has been refined and extended by a symbolic implementation. In what follows, we describe some details of the latter. We consider here MDP with a reward function $rew\colon S \times Act \to \mathbb{N}_{\geqslant 0}$, mapping state-action pairs (s, α) to the non-negative integer reward $rew(s, \alpha)$. The quantiles under consideration (for details we refer to [5]) are optimal reward

thresholds that guarantee that the maximal or minimal probability of a reward-bounded reachability path formula meets some probability bound. Examples are $\min\{\, r : \Pr^{\max}(\Diamond^{\leq r}\, \Phi) > p \,\}$ or $\max\{\, r : \Pr^{\min}(\Diamond^{\geq r}\, \Phi) > p \,\}$ where r can be seen as a parametric reward bound, Φ is a state formula and p a rational probability bound. Quantiles yield a useful concept for the cost-utility analysis, e.g., in terms of the minimal amount of energy r required to reach some goal Φ with probability at least p for some/all schedulers. The approach for computing quantiles as proposed in [5] consists of a two-step process. A precomputation step determines all states $s \in S$ for which the quantile exists, i.e., is finite. In the simplest case, this amounts to the computation of the maximal probability for unbounded reachability. In other cases, the computation requires the analysis of zero-reward and positive-reward end components [5]. For the remaining states, an iterative approach is used, which we illustrate here for a quantile of the form $\min\{\, r : \Pr^{\max}(\Diamond^{\leq r}\, \Phi) > p \,\}$ where we suppose the MDP has a unique initial state s_0. Successively, the values $x_{s,r} = \Pr_s^{\max}(\Diamond^{\leq r}\, \Phi)$ for $r = 1, 2, 3, \ldots$ are computed for all states $s \in S$ until some r with $x_{s_0,r} > p$ is reached, using the equation $x_{s,r} = \max\{A_s, B_s\}$ with

$$A_s = \max_{\alpha \in Act(s), rew(s,\alpha)=0} \sum\nolimits_{t \in S} \Pr(s,\alpha,t) \cdot x_{t,r}$$

$$B_s = \max_{\alpha \in Act(s), rew(s,\alpha)>0} \sum\nolimits_{t \in S} P(s,\alpha,t) \cdot x_{t,r-rew(s,\alpha)}$$

where $\Pr(s,\alpha,t)$ is the probability of reaching state t when action α is chosen in state s. For states satisfying Φ, $x_{s,r}$ is set to 1 for all r. The values A_s, handling the zero-reward actions, are computed using value iteration. The values B_s, handling the positive-reward actions, are determined by inserting the previously calculated values $x_{t,i}$ for $i < r$. For the other quantile variants, similar computations are performed [5]. The time complexity of this approach is exponential, meeting the complexity-theoretic optimum [16].

3.1 Symbolic computation of quantiles

We have extended PRISM with a symbolic implementation for the computation of quantiles, following the general approach outlined above. For the precomputation step, we rely on the PRISM machinery for the computation of maximal/minimal probabilities for unbounded path formulas and for the computation of (maximal) end components, adapted for identifying states in positive-reward end components and zero-reward end components by appropriate symbolic model transformations.

For the iterative computation of the values $x_{s,r}$ for $r = 1, 2, 3, \ldots$ until the probability threshold p is reached, the values $x_{s,r}$ are stored symbolically, using one MTBDD per bound r to represent the functions $x_r \colon S \to \mathbb{Q}$. The state-action pairs with positive reward are handled first, computing the MTBDD $B \colon S \to \mathbb{Q}$. Here, all state-action pairs with identical reward value are handled simultaneously. Consequently, this symbolic approach tends to be most efficient if there are many state-action pairs, but few distinct reward values in the model. To

subsequently handle the zero-reward state-action pairs, we symbolically transform the MDP. First, all positive-reward actions are stripped and replaced by a single fresh τ-action for each state. These τ-actions model the choice of choosing the "best" positive-reward action in a state s and go to a special *goal* trap state with probability $B(s)$ and to a *fail* trap state with probability $1 - B(s)$. The computation of $x_{s,i}$ then amounts to a standard maximal/minimal reachability probability computation in the transformed model by means of value iteration, relying on the computation engine chosen by the user, i.e., either the MTBDD, HYBRID or SPARSE engine. As the state value vectors x_r are stored symbolically in all cases and the use of the semi-symbolic techniques of the HYBRID and SPARSE engines is thus limited, we denote these engines as SEMIHYBRID and SEMISPARSE in the context of our symbolic quantile implementation.

3.2 Benchmarks for quantile computations

To perform benchmarking of our implementation, we have reused several models and quantile queries that were first considered in [5] for benchmarking our quantile implementation for PRISM's EXPLICIT engine. We present here (Table 2) statistics for some noteworthy model instances, for further statistics and details on the models and quantile queries we refer to appendix C. [4]

As can be seen in Table 2, there are model instances were the quantile implementation in the EXPLICIT engine easily outperforms our symbolic approach, e.g., for the "self-stabilizing" case study, despite a very compact MTBDD representation of the model. To put computation times such as 2153.2 s (for N=18) into context, the number of iterations in the quantile computation has to be kept in mind: Here, 392 iterations were required, with an average time per iteration of around 5 s. The large number of iterations thus amplifies the time spent in each iteration. For the "asynchronous leader-election" case study, our symbolic implementation becomes competitive for N=8 because of the time spent for model construction in the EXPLICIT engine due to the large state space. The symbolic computations still yield results for N=9, where the explicit approach times out. A similar picture is seen for query Q7 of the "energy-aware job scheduling" case study. For the instances of this case study and query Q8 shown in Table 2, the symbolic implementation vastly outperforms the explicit implementation. This is mainly due to the large amount of time spent there for the precomputation step (1971 s for N=6), while the symbolic engines perform this step in around 2 s. This appears to be due to inefficiencies in some of the end component computations in the EXPLICIT engine, which we are currently investigating and working on a potential fix. For the "energy-aware job scheduling" case study, the computations were carried out in a reordered model, which lead to a significant decrease in MTBDD size and computation time. For instance, for (Q7) and N=6 we observed a reduction in the size of the transition matrix of 78.2 % and the quantile computation (SEMIHYBRID) took 43,832.9 s in the original model instead of 3808.4 s in the reordered model, with similar reductions for MTBDD and SEMISPARSE.

[4] The benchmarks for the quantile computations were carried out on a machine with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux.

Table 2. Quantile computations for selected case studies, with statistics for the model size (reachable state space, MTBDD size of symbolic transition matrix) and times spent for model building and computing the quantile query (in seconds). The "it." column depicts the number of overall iterations in the quantile computation.

N	States	MTBDD size	it.	Explicit		symbolic quantile computations: SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
Self-stabilizing algorithm (Israeli/Jalfon), N processes (query Q1)											
11	2047	433	144	0.5	0.2	<0.1	2.5	<0.1	2.3	<0.1	2.0
15	32767	729	271	1.6	3.9	<0.1	119.1	<0.1	114.7	<0.1	159.3
18	262143	993	392	9.8	53.8	<0.1	2135.2	<0.1	2865.1	<0.1	2240.3
Asynchronous leader election, N processes (query Q3)											
7	2095783	180383	206	63.1	83.2	5.9	358.8	7.0	369.8	7.6	402.7
8	18674484	392093	238	1633.2	891.8	20.6	1228.4	24.6	1307.4	21.9	1448.8
9	167748115	868257	279	–	–	106.1	7751.9	92.3	5728.1	92.9	7545.6
Energy-aware job scheduling, N processes (query Q7)											
5	6079533	187458	302	334.1	285.9	7.8	1196.2	7.2	1128.0	7.9	1142.9
6	44072357	507805	416	–	–	21.7	3808.4	22.8	4045.9	25.0	3962.8
Energy-aware job scheduling, N processes (query Q8)											
5	3049471	25363	13	62.5	398.6	0.8	86.0	0.9	74.3	0.9	104.6
6	7901694	38911	15	210.0	2375.5	1.8	390.6	1.8	378.1	1.3	317.1

Quantiles in Feature-Oriented Systems. In product lines, collections of systems are described through the combination of features. Thus, the systems in a product line usually share a lot of common behaviors, which makes symbolic approaches appealing. Using a family-based approach, i.e., modeling the product line in a single model, in previous work [13] we performed experiments on an energy-aware server product line eServer. There, we illustrated the benefits of symbolic representations in product-line verification and showed that variable orderings have a crucial impact on the analysis performance. However, due to the lack of a symbolic quantile implementation, an energy-utility analysis of eServer had to be postponed as future work.

In Table 3, we summarize statistics for the computation of quantiles on two instances of eServer, becoming possible due to our symbolic implementation. We computed the minimal amount of energy required to guarantee in 95% of the cases a certain percentage of the time without any package drop. The table shows the impact of our four reorder mechanisms on the model size and the quantile computation time. We only included the results for the Mtbdd engine, as the other engines struggled with the size of the model and reached a timeout after one day. Within all computations, 1476 quantile iterations were required. Interestingly, although the model presented in [13] already used heuristics to find good initial variable orderings, the fully automatic reorder mechanisms presented here allow for a further significant reduction of the model size and a speedup of the analyses.

Table 3. Quantile computations for eServer, with statistics for the reachable state space and MTBDD size of the transition matrix, reduction and time for reordering, and time building the model and computing the quantile query (in seconds).

		States	MTBDD nodes	Reduction in %	t_{reorder}	Mtbdd t_{build}	Mtbdd t_{query}
Instance 1	original	145 984 112	64 030	-	-	10.2	1 018.6
	reordered	"	40 096	37.4	3.5	7.8	766.4
	with explode	"	34 902	45.5	8.3	9.1	726.2
	with globalize	"	36 149	43.5	2.9	6.3	704.3
	with exp.+glob.	"	30 325	52.6	3.1	7.9	638.6
Instance 2	original	441 704 832	140 556	-	-	27.2	3 664.6
	reordered	"	72 565	48.4	68.5	16.6	3 510.5
	with explode	"	66 874	52.4	32.3	17.7	3 204.7
	with globalize	"	43 249	69.2	6.5	11.2	3 099.0
	with exp.+glob.	"	37 674	73.3	8.5	12.1	2 833.0

4 Additional enhancements

We report here on additional enhancements we have implemented in PRISM both for the symbolic and explicit engines, related to the support of ω-automata.

Accepting end component computations for Streett conditions. Traditionally, PRISM has relied on an internal implementation of Safra's determinization construction for generating the deterministic Rabin automata used for LTL model checking, e.g., for computing $\Pr^{\max}(\varphi)$ or $\Pr^{\min}(\varphi)$ for an LTL formula φ. Recently, support was added for automata with generalized Rabin acceptance [9,2] to benefit from advances in the construction of small deterministic automata [14,22]. This includes support for calling external tools for the transformation of LTL formulas into deterministic Muller-automata, relying on the recent Hanoi Omega Automata (HOA) format [2], which supports the concise representation of common acceptance conditions in a generic normal form.

We have extended PRISM's MDP model checking with support for Streett conditions, relying on the recursive algorithm for end-component analysis of [6]. Streett conditions are dual to Rabin conditions and are well suited for the specification of fairness constraints and for conjunctions of properties. They appear as well in the computation of conditional probabilities in MDPs [7]. It is well known, both in theory [29] and in practice [20], that for some languages, deterministic Streett automata can be significantly smaller than Rabin automata.

Extremal probabilities for automata specifications. We have furthermore extended the property syntax of the probability operators in PRISM to allow the use of a HOA-automaton file instead of an LTL formula, providing the full power of ω-regular languages. For DTMC models, the full range of acceptance conditions in the normal form of the HOA format [2] is supported. For MDPs, Rabin, generalized Rabin and Streett conditions are supported. For the computation of $\Pr^{\min}$, which requires the complementation of the language of the automaton, we support Rabin and Streett conditions, exploiting their duality.

5 Conclusion

In this paper, we have demonstrated the potential for automatic variable reordering for symbolic model checking in PRISM, including the benefits of now having fine-grained control over the variable order using our syntactic transformations. We have also shown that our symbolic implementation for quantiles is useful in practice, particularly where explicit representations of the model are infeasible.
Future work: In the area of automatic variable reordering, it would be interesting to support more structured reordering: Often, models are obtained from templates with parametrization, e.g., specifying the number of copies of certain modules in the model. By swapping the variables of all copies simultaneously, it might be possible to discover good initial variable orders from instances with few copies and apply these to instances with more copies. This approach would also be interesting when the aim is to apply symmetry reduction [25,12], as all copies would remain symmetrical. While our syntactic transformations provide very fine-grained reordering for the state variables, it would be interesting to have the option of adding back some restrictions or hints for the reordering by annotating the variable declarations in the PRISM model. This would allow to state preferences which variable should be kept together, etc. In this context it would also make sense to revisit previous work on heuristics for good initial variable orderings in PRISM [30], making use of the finer-grained control that is now possible. In addition, our benchmark results serve as an indication that it would be worthwhile to attempt a refactoring of PRISM to remove the variable order assumptions and add support for asynchronous reordering.

For our symbolic quantile computations, it appears worthwhile to consider an iterative implementation that fully exploits the approach of the Hybrid engine, with a symbolic transition matrix and explicit state value vector storage. This could allow the application of several of the techniques employed by the quantile computations of the Explicit engine to speed-up the computations.

The implementation of the end component computation for Streett conditions could serve as the base for supporting more complex types of fairness conditions via the approach of [6], such as fairness for the scheduling of the modules in a PRISM model. It would also be interesting to perform a detailed experimental evaluation on the use of Streett versus (generalized) Rabin automata for probabilistic model checking in practice.

References

1. S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 88–104. Springer, 2003.
2. T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi omega-automata format. In *Proc. Computer Aided Verification, Part I (CAV'15)*, volume 9206 of *LNCS*, pages 479–486. Springer, 2015.
3. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.

4. C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 430–440, 1997.
5. C. Baier, M. Daum, C. Dubslaff, J. Klein, and S. Klüppelholz. Energy-utility quantiles. In *Proc. NASA Formal Methods (NFM'14)*, volume 8430 of *LNCS*, pages 285–299. Springer, 2014.
6. C. Baier, M. Größer, and F. Ciesinski. Quantitative analysis under fairness constraints. In *Proc. Automated Technology for Verification and Analysis (ATVA'09)*, volume 5799 of *LNCS*, pages 135–150. Springer, 2009.
7. C. Baier, J. Klein, S. Klüppelholz, and S. Märcker. Computing conditional probabilities in Markovian models efficiently. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *LNCS*, pages 515–530. Springer, 2014.
8. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
9. K. Chatterjee, A. Gaiser, and J. Kretínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *Proc. Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 559–575. Springer, 2013.
10. G. Ciardo, A. S. Miner, and M. Wan. Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. *SIGMETRICS Performance Evaluation Review*, 36(4):58–63, 2009.
11. A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 359–364. Springer, 2002.
12. A. F. Donaldson, A. Miller, and D. Parker. Language-level symmetry reduction for probabilistic model checking. In *Proc. Quantitative Evaluation of Systems (QEST'09)*, pages 289–298. IEEE, 2009.
13. C. Dubslaff, C. Baier, and S. Klüppelholz. Probabilistic model checking for feature-oriented systems. *Transactions on Aspect-Oriented Software Development*, 12:180–220, 2015.
14. J. Esparza and J. Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In *Proc. Computer Aided Verification (CAV'14)*, volume 8559 of *LNCS*, pages 192–208. Springer, 2014.
15. M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.
16. C. Haase and S. Kiefer. The odds of staying on budget. In *Proc. Automata, Languages, and Programming (ICALP'15)*, volume 9135 of *LNCS*, pages 234–246. Springer, 2015.
17. G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(12):1479–1493, 1996.
18. V. Hartonas-Garmhausen, S. V. A. Campos, and E. M. Clarke. ProbVerus: probabilistic symbolic model checking. In *Proc. Formal Methods for Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *LNCS*, pages 96–110, 1999.
19. H. Hermanns, M. Z. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003.

20. J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363(2):182–195, 2006.
21. J. Klein, C. Baier, P. Chrszon, M. Daum, C. Dubslaff, S. Klüppelholz, S. Märcker, and D. Müller. Advances in symbolic probabilistic model checking with PRISM (extended version), 2016. `http://wwwtcs.inf.tu-dresden.de/ALGI/PUB/TACAS16/`.
22. Z. Komárková and J. Kretínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *Proc. Automated Technology for Verification and Analysis (ATVA'14)*, volume 8837 of *LNCS*, pages 235–241. Springer, 2014.
23. M. Kuntz and M. Siegle. CASPA: symbolic model checking of stochastic systems. In *Proc. Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB'06)*, pages 465–468. VDE Verlag, 2006.
24. M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *Software Tools for Technology Transfer*, 6(2):128–142, 2004.
25. M. Z. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proc. Computer Aided Verification (CAV'06)*, volume 4144 of *LNCS*, pages 234–248. Springer, 2006.
26. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
27. M. Z. Kwiatkowska, G. Norman, and D. Parker. The PRISM benchmark suite. In *Proc. Quantitative Evaluation of Systems (QEST'12)*, pages 203–204. IEEE, 2012. Website: `https://github.com/prismmodelchecker/prism-benchmarks/`.
28. K. Lampka. *A symbolic approach to the state graph based analysis of high-level Markov reward models.* PhD thesis, Universität Erlangen-Nürnberg, 2007.
29. C. Löding. Optimal bounds for transformations of omega-automata. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, volume 1738 of *LNCS*, pages 97–109. Springer, 1999.
30. V. Maisonneuve. Automatic heuristic-based generation of MTBDD variable orderings for PRISM models. Internship report, ENS Cachan & Oxford University, 2009. `http://www.prismmodelchecker.org/papers/vivien-bdds-report.pdf`.
31. K. L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.
32. A. S. Miner and D. Parker. Symbolic representations and analysis of large probabilistic systems. In *Validation of Stochastic Systems - A Guide to Current Research*, volume 2925 of *LNCS*, pages 296–338, 2004.
33. S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Proc. Computer-Aided Design (ICCAD'95)*, pages 74–77. IEEE, 1995.
34. D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems.* PhD thesis, University of Birmingham, 2002.
35. PRISM model checker. Website: `http://www.prismmodelchecker.org/`.
36. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. Computer-Aided Design (ICCAD'93)*, pages 42–47. IEEE, 1993.
37. F. Somenzi. CUDD: Colorado University decision diagram package. Website: `http://vlsi.colorado.edu/~fabio/CUDD/`.
38. M. Ummels and C. Baier. Computing quantiles in Markov reward models. In *Proc. Foundations of Software Science and Computation Structures (FOSSACS'13)*, volume 7794 of *LNCS*, pages 353–368. Springer, 2013.

Appendix

A Initial variable order variants in PRISM

The variable order of Fig. 1 in Sec. 2 corresponds to the "top" variable ordering discussed in Sec. 4.2.2 of [34] and is the standard variable ordering when using the HYBRID and SPARSE engines of PRISM. When using the MTBDD engine, a slightly different ordering is used by default: Here, the MTBDD variables for the nondeterministic choice of scheduling, i.e., determining which module is active, are not placed with the other nondeterministic choice variables but are instead interleaved with the state variables of the modules, i.e., each module starts with the scheduling MTBDD variable, followed by the MTBDD variables for the state variables. This corresponds to the so-called "middle" variable ordering discussed in Sec. 4.2.2 of [34]. In contrast to the HYBRID and SPARSE engines that require that all nondeterministic choice variables are placed before the first state variable for algorithmic reasons, the MTBDD engine has no such restrictions and thus allows the more natural placement of the scheduling variable next to the state variables.

The "middle" variable order and "globalizing state variables". When the "top" variable order is used, the syntactic transformation of "globalizing" the state variables 2.3 does not change the resulting variable order. However, when using the "middle" variable order, moving the state variables outside of the PRISM modules leads to a subtly changed variable order: As the scheduling choice variables remain with their modules, the effect is that all state variables are located before the scheduling variables, which form a block at the bottom of the MTBDD. This corresponds to the "bottom" ordering scheme considered in Sec. 4.2.2 of [34], which was found to be often suboptimal. However, a command-line switch allows to select the "top" ordering also for the MTBDD engine, if so desired.

B Further benchmark details: variable reordering

We provide here further statistics for our benchmarking of the automatic variable reordering using the PRISM benchmark suite. The parameters are the same as in the main part of the paper.

Fig. 5 shows statistics for the case where the "explode bits" syntactic transformation is applied before reordering, i.e., allowing the state variables inside a module to be reordered at the bit level.

Fig. 6 shows statistics for the case where the "globalize variables" syntactic transformation is applied before reordering, i.e., allowing the state variables to be reordered without being restricted by module boundaries.

Fig. 7 then shows statistics for the case where both syntactic transformations are applied in combination before reordering. This provides the most flexibility in reordering, as the individual bits of state variables may be interleaved, without regard for the module boundaries.

As discussed before (Fig. fig:overview-reorder), the "csma4_6" instance had a timeout during reordering.

To allow a detailed comparison of the reductions with the four considered variants, we provide Tables 4–8 with the statistics for the individual model instances.

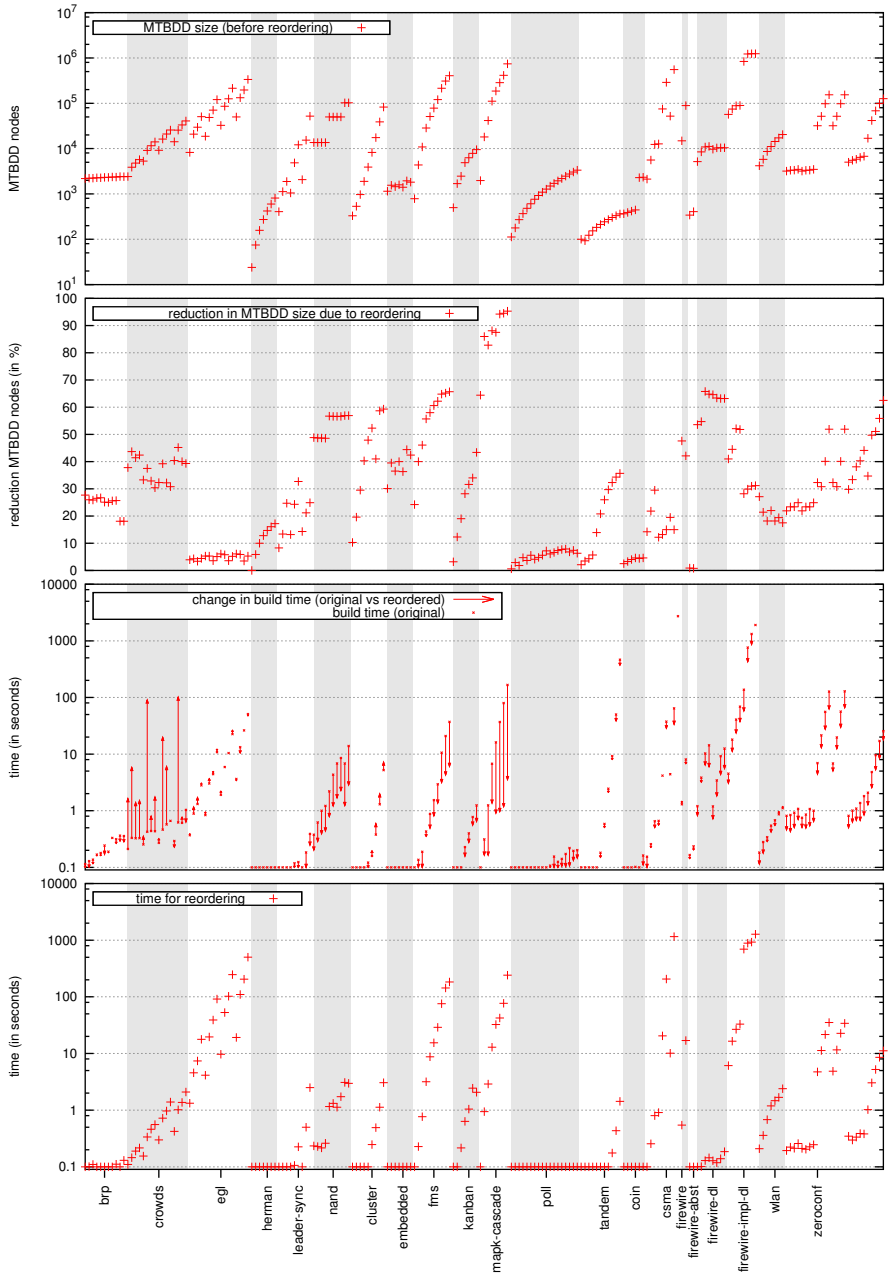

Fig. 5. Statistics for reordering after the "explode bits" syntactic transformation: the number of MTBDD nodes before reordering, reduction in the number of MTBDD nodes, the change in time for building the model (before/after reordering) and the time spent reordering. Times below 0.1 seconds are clipped to 0.1 for visualization purposes.

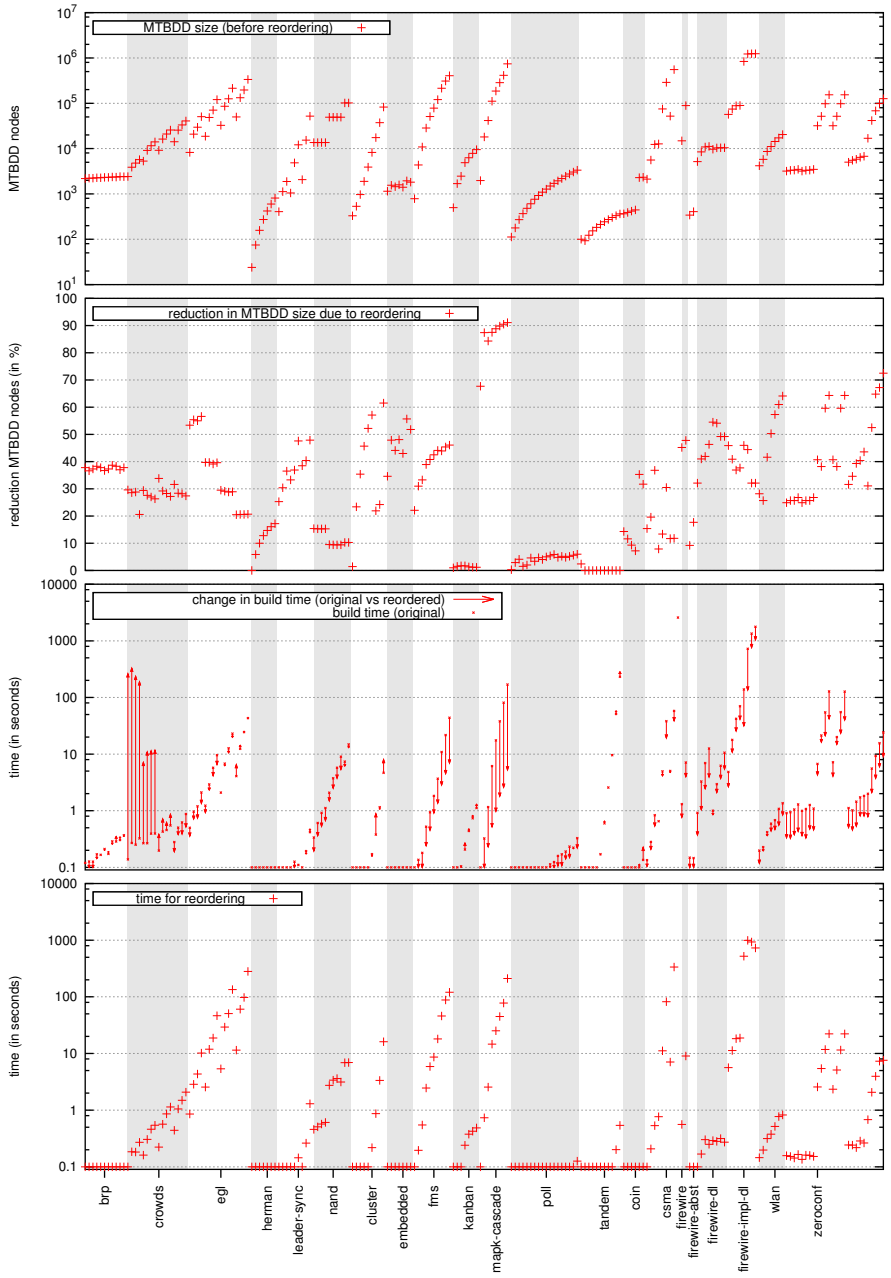

Fig. 6. Statistics for reordering after the "globalize variables" syntactic transformation: the number of MTBDD nodes before reordering, reduction in the number of MTBDD nodes, the change in time for building the model (before/after reordering) and the time spent reordering. Times below 0.1 seconds are clipped to 0.1 for visualization purposes.

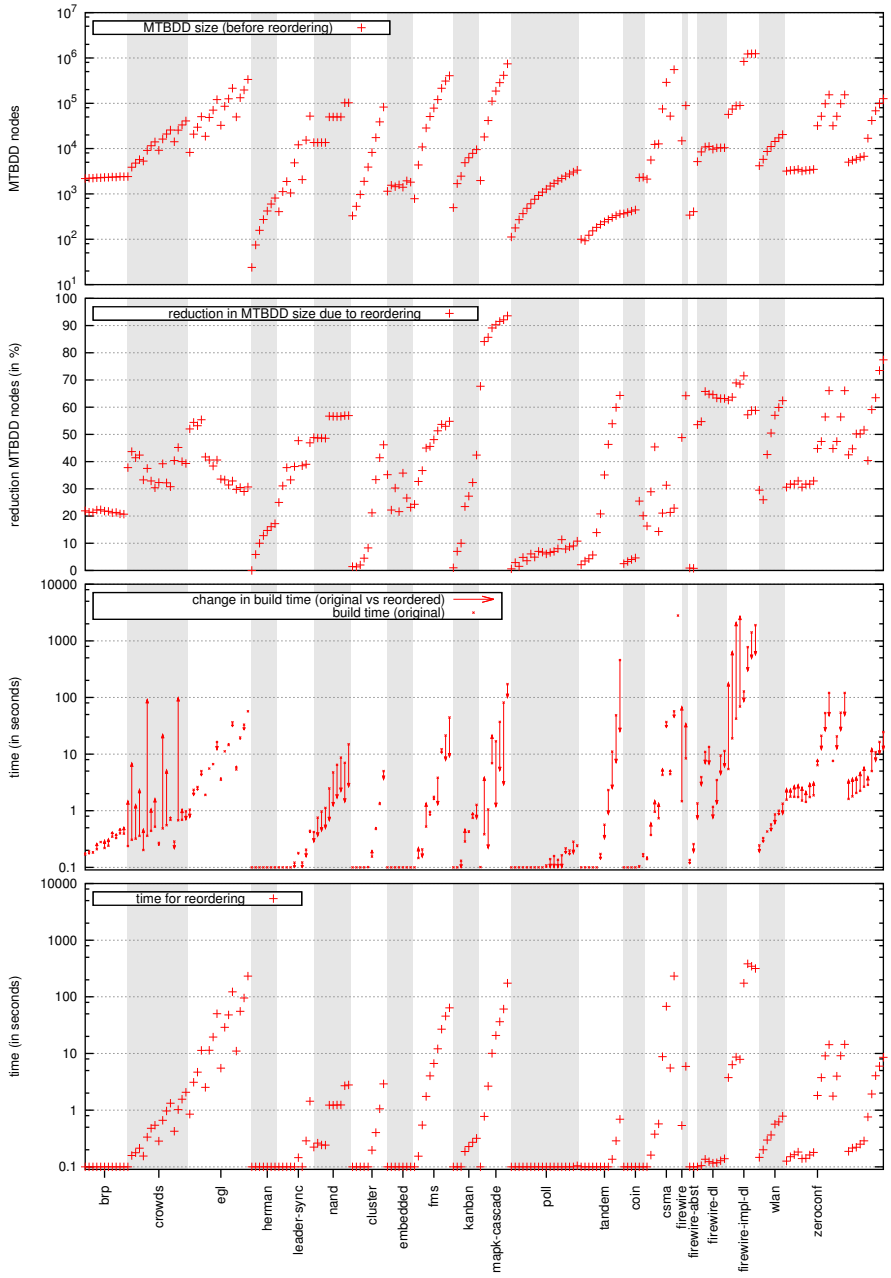

Fig. 7. Statistics for reordering after the "globalize variables" and "explode bits" syntactic transformations: the number of MTBDD nodes before reordering, reduction in the number of MTBDD nodes, the change in time for building the model (before/after reordering) and the time spent reordering. Times below 0.1 seconds are clipped to 0.1 for visualization purposes.

Table 4. (Part 1 of 5). Statistics for the reduction achieved using reordering on the standard model instance and where the "explode bits" and "globalize variables" transformations were applied. In the last column, both transformations are applied. For reference, the MTBDD size before reordering is included as well.

	MTBDD	reduction in %			
Model instance	before	standard	explode	globalize	exp.+glob.
brp,N=16,MAX=2	4299	22.0	27.5	38.3	21.8
brp,N=16,MAX=3	4286	22.5	27.7	37.8	21.9
brp,N=16,MAX=4	4414	23.2	26.0	36.6	21.4
brp,N=16,MAX=5	4395	23.8	25.9	37.3	21.3
brp,N=32,MAX=2	4493	21.9	26.5	38.3	22.2
brp,N=32,MAX=3	4480	22.3	26.7	37.8	22.3
brp,N=32,MAX=4	4608	22.4	25.1	36.7	21.8
brp,N=32,MAX=5	4589	23.0	25.0	37.3	21.7
brp,N=64,MAX=2	4687	21.7	25.6	38.7	21.2
brp,N=64,MAX=3	4674	22.2	25.7	38.3	21.3
brp,N=64,MAX=4	4802	21.7	18.1	37.0	20.8
brp,N=64,MAX=5	4783	22.2	18.1	37.8	20.7
crowds,TR=3,CS=5	3465	29.6	37.8	29.6	37.8
crowds,TR=4,CS=5	5388	28.6	43.7	28.6	43.7
crowds,TR=5,CS=5	6612	28.8	41.4	28.8	41.4
crowds,TR=6,CS=5	7819	20.6	42.4	20.6	42.4
crowds,TR=3,CS=10	7187	29.4	33.3	29.4	33.3
crowds,TR=4,CS=10	12078	27.6	37.5	27.6	37.5
crowds,TR=5,CS=10	15242	27.0	32.9	27.0	32.9
crowds,TR=6,CS=10	18389	26.3	30.4	26.3	30.4
crowds,TR=3,CS=15	12215	33.8	32.3	33.8	32.3
crowds,TR=4,CS=15	21332	29.2	39.2	29.2	39.2
crowds,TR=5,CS=15	27227	28.3	32.2	28.3	32.2
crowds,TR=6,CS=15	33105	27.2	30.8	27.2	30.8
crowds,TR=3,CS=20	18660	31.6	40.4	31.6	40.4
crowds,TR=4,CS=20	33351	28.4	45.2	28.4	45.2
crowds,TR=5,CS=20	42800	28.2	40.0	28.2	40.0
crowds,TR=6,CS=20	52232	27.4	39.3	27.4	39.3
egl,N=5,L=2	15529	2.6	3.9	53.4	52.0
egl,N=5,L=4	37660	2.4	4.3	55.4	54.4
egl,N=5,L=6	52976	2.7	3.4	55.0	53.2
egl,N=5,L=8	88979	2.2	4.4	56.6	55.4
egl,N=10,L=2	35863	2.6	5.3	39.7	41.7
egl,N=10,L=4	89564	2.2	5.3	39.7	40.6
egl,N=10,L=6	127850	2.3	3.6	39.1	38.4
egl,N=10,L=8	216018	2.0	5.1	39.6	40.6
egl,N=15,L=2	63056	2.5	6.1	29.5	33.6
egl,N=15,L=4	160350	1.9	5.8	29.1	33.3
egl,N=15,L=6	230779	1.9	3.6	28.8	31.4
egl,N=15,L=8	391510	1.6	5.2	28.9	32.9
egl,N=20,L=2	97327	2.2	6.1	20.5	29.8
egl,N=20,L=4	250469	1.6	5.9	20.6	30.5
egl,N=20,L=6	362446	1.6	3.5	20.6	29.0
egl,N=20,L=8	616370	1.4	5.3	20.7	30.7

Table 5. (Part 2 of 5). Statistics for the reduction achieved using reordering on the standard model instance and where the "explode bits" and "globalize variables" transformations were applied. In the last column, both transformations are applied. For reference, the MTBDD size before reordering is included as well.

	MTBDD	reduction in %			
Model instance	before	standard	explode	globalize	exp.+glob.
herman3	90	0.0	0.0	0.0	0.0
herman5	169	5.9	5.9	5.9	5.9
herman7	280	10.0	10.0	10.0	10.0
herman9	423	12.8	12.8	12.8	12.8
herman11	598	14.7	14.7	14.7	14.7
herman13	805	16.1	16.1	16.1	16.1
herman15	1044	17.2	17.2	17.2	17.2
leader_sync3_2	664	7.7	8.3	25.3	25.0
leader_sync3_3	1596	14.6	13.4	30.4	31.1
leader_sync3_4	2521	22.2	24.7	36.5	37.8
leader_sync4_2	1504	12.7	13.2	33.3	33.3
leader_sync4_3	6201	24.1	24.3	36.9	38.2
leader_sync4_4	14852	32.2	32.7	47.6	47.7
leader_sync5_2	2799	14.0	14.3	38.5	38.5
leader_sync5_3	18775	21.4	21.2	40.4	39.0
leader_sync5_4	61211	24.6	24.9	47.9	46.9
nand,N=20,K=1	21684	15.4	48.8	15.4	48.8
nand,N=20,K=2	21707	15.3	48.7	15.3	48.7
nand,N=20,K=3	21709	15.3	48.7	15.3	48.7
nand,N=20,K=4	21730	15.3	48.6	15.3	48.6
nand,N=40,K=1	75215	9.5	56.7	9.5	56.7
nand,N=40,K=2	75238	9.4	56.6	9.4	56.6
nand,N=40,K=3	75240	9.4	56.6	9.4	56.6
nand,N=40,K=4	75261	9.4	56.6	9.4	56.6
nand,N=60,K=1	147017	10.3	56.9	10.3	56.9
nand,N=60,K=2	147040	10.3	56.9	10.3	56.9
cluster,N=2	983	10.3	10.3	1.4	1.4
cluster,N=4	1480	20.0	19.6	23.4	1.3
cluster,N=8	2333	29.7	29.5	35.4	2.0
cluster,N=16	3992	38.7	40.3	45.7	4.5
cluster,N=32	7391	45.5	47.9	52.2	8.3
cluster,N=64	14526	49.9	52.3	57.1	21.2
cluster,N=128	29638	52.5	41.0	21.9	33.4
cluster,N=256	61749	53.6	58.7	24.2	41.4
cluster,N=512	132908	55.1	59.3	61.5	46.2
embedded,MAX_COUNT=2	2026	29.5	30.1	34.6	35.2
embedded,MAX_COUNT=3	2663	39.6	39.5	47.9	22.2
embedded,MAX_COUNT=4	2515	36.1	36.5	44.1	30.3
embedded,MAX_COUNT=5	2721	39.5	40.0	48.1	21.6
embedded,MAX_COUNT=6	2433	35.8	36.3	43.0	35.8
embedded,MAX_COUNT=7	3261	46.0	44.4	55.7	26.6
embedded,MAX_COUNT=8	3089	41.8	42.4	51.8	23.2

Table 6. (Part 3 of 5). Statistics for the reduction achieved using reordering on the standard model instance and where the "explode bits" and "globalize variables" transformations were applied. In the last column, both transformations are applied. For reference, the MTBDD size before reordering is included as well.

	MTBDD	reduction in %			
Model instance	before	standard	explode	globalize	exp.+glob.
fms,n=1	2289	22.0	24.2	22.1	24.3
fms,n=2	10155	38.1	40.0	31.0	32.7
fms,n=3	23594	42.7	46.1	33.3	36.7
fms,n=4	57522	51.6	55.7	38.9	45.0
fms,n=5	99794	55.9	58.0	40.8	45.6
fms,n=6	139245	59.1	60.6	42.5	48.1
fms,n=7	230617	59.9	62.2	44.1	51.3
fms,n=8	376485	60.9	64.8	43.9	53.7
fms,n=9	539121	63.2	65.2	45.4	53.0
fms,n=10	658124	64.4	65.7	46.1	54.8
kanban,t=1	808	3.2	3.2	1.0	1.0
kanban,t=2	2721	8.7	12.3	1.5	7.0
kanban,t=3	4077	14.0	19.0	1.7	10.0
kanban,t=4	8307	19.1	28.2	1.7	23.5
kanban,t=5	10964	23.6	31.6	1.3	27.3
kanban,t=6	14001	27.5	34.0	1.2	32.3
kanban,t=7	17308	31.2	43.4	1.2	42.4
mapk_cascade,N=1	4807	64.4	64.4	67.7	67.7
mapk_cascade,N=2	47134	85.5	86.0	87.4	84.1
mapk_cascade,N=3	103992	80.4	82.8	84.3	85.7
mapk_cascade,N=4	260910	85.9	88.1	87.5	89.1
mapk_cascade,N=5	414699	92.2	87.5	88.8	90.3
mapk_cascade,N=6	610340	92.8	94.2	89.8	91.5
mapk_cascade,N=7	856986	93.1	94.5	90.5	92.1
mapk_cascade,N=8	1478511	93.5	95.3	91.1	93.6
poll3	313	0.3	0.6	0.3	0.6
poll4	442	2.9	2.9	2.9	2.9
poll5	664	3.9	1.8	4.1	1.5
poll6	847	1.5	4.7	1.5	4.8
poll7	1066	2.9	3.7	2.0	3.6
poll8	1297	4.6	5.6	4.6	6.1
poll9	1657	1.9	4.1	3.4	4.9
poll10	1942	2.4	4.8	4.7	7.0
poll11	2263	3.6	5.6	4.0	6.7
poll12	2596	4.7	7.2	5.0	6.1
poll13	2977	6.2	6.1	5.4	6.6
poll14	3358	5.5	6.6	5.9	7.0
poll15	3775	6.2	7.3	4.8	8.0
poll16	4204	7.0	7.7	5.2	11.3
poll17	4822	5.5	7.9	4.8	7.9
poll18	5305	3.9	6.8	5.2	8.7
poll19	5824	3.4	7.4	5.6	9.0
poll20	6355	5.3	6.3	6.0	10.8

Table 7. (Part 4 of 5). Statistics for the reduction achieved using reordering on the standard model instance and where the "explode bits" and "globalize variables" transformations were applied. In the last column, both transformations are applied. For reference, the MTBDD size before reordering is included as well.

	MTBDD	reduction in %			
Model instance	before	standard	explode	globalize	exp.+glob.
tandem,c=5	290	2.4	2.1	2.4	2.1
tandem,c=7	286	0.0	3.5	0.0	3.5
tandem,c=15	441	0.0	4.3	0.0	4.3
tandem,c=31	716	0.0	5.7	0.0	5.7
tandem,c=63	1263	0.0	13.9	0.0	13.9
tandem,c=127	2418	0.0	20.8	0.0	20.8
tandem,c=255	4917	0.0	26.0	0.0	35.1
tandem,c=511	10360	0.0	29.7	0.0	46.3
tandem,c=1023	22203	0.0	32.3	0.0	53.9
tandem,c=2047	47870	0.0	34.3	0.0	59.9
tandem,c=4095	103233	0.0	35.7	0.0	64.3
coin2,K=2	671	0.0	2.5	14.3	2.5
coin2,K=4	724	0.0	3.3	11.6	3.3
coin2,K=8	777	0.0	4.0	9.3	4.0
coin2,K=16	830	0.0	4.6	7.2	4.6
coin4,K=2	3616	0.0	4.4	35.3	25.5
coin4,K=4	3709	0.0	4.6	31.7	20.1
csma2_2	4394	10.3	14.2	15.4	16.3
csma2_4	11141	14.5	21.8	19.6	28.9
csma2_6	24643	23.1	29.5	36.8	45.4
csma3_2	26363	9.6	12.2	7.9	14.3
csma3_4	152892	8.3	13.2	13.4	21.1
csma3_6	604313	13.5	15.0	30.5	31.3
csma4_2	110915	14.7	19.5	11.7	21.3
csma4_4	1139902	11.8	15.0	11.8	22.9
csma4_6	-	-	-	-	-
firewire,d=3	28627	39.3	47.6	45.2	48.8
firewire,d=36	164283	32.5	42.1	47.8	64.2
firewire_abst,d=3	671	9.2	0.9	9.2	0.9
firewire_abst,d=36	786	17.7	0.8	17.7	0.8
firewire_dl,dl=200,d=3	8445	32.1	53.6	32.1	53.6
firewire_dl,dl=400,d=3	14987	41.0	54.7	41.0	54.7
firewire_dl,dl=600,d=3	20920	41.9	65.8	41.9	65.8
firewire_dl,dl=800,d=3	21635	46.3	64.8	46.3	64.8

Table 8. (Part 5 of 5). Statistics for the reduction achieved using reordering on the standard model instance and where the "explode bits" and "globalize variables" transformations were applied. In the last column, both transformations are applied. For reference, the MTBDD size before reordering is included as well.

	MTBDD	reduction in %			
Model instance	before	standard	explode	globalize	exp.+glob.
firewire_dl,dl=200,d=36	17890	54.5	64.6	54.5	64.6
firewire_dl,dl=400,d=36	19250	54.1	63.4	54.1	63.4
firewire_dl,dl=600,d=36	19307	49.2	63.2	49.2	63.2
firewire_dl,dl=800,d=36	19313	49.2	63.2	49.2	63.2
firewire_impl_dl,dl=200,d=3	106794	34.3	41.0	45.9	62.6
firewire_impl_dl,dl=400,d=3	146178	29.4	44.5	40.9	63.7
firewire_impl_dl,dl=600,d=3	176223	26.9	52.1	36.9	68.9
firewire_impl_dl,dl=800,d=3	178648	27.6	51.8	37.7	68.5
firewire_impl_dl,dl=200,d=36	1573881	24.7	28.2	46.0	71.5
firewire_impl_dl,dl=400,d=36	2348306	22.8	29.9	44.4	57.2
firewire_impl_dl,dl=600,d=36	2396942	21.3	31.0	32.1	58.8
firewire_impl_dl,dl=800,d=36	2396950	21.3	31.2	32.1	58.8
wlan0,COL=0	10289	24.7	27.1	28.2	29.5
wlan1,COL=0	14084	15.1	21.4	25.7	26.0
wlan2,COL=0	20646	12.8	18.2	41.6	42.6
wlan3,COL=0	25973	16.5	22.0	50.3	50.5
wlan4,COL=0	33506	11.8	18.2	57.3	57.0
wlan5,COL=0	39604	12.9	19.4	61.0	59.9
wlan6,COL=0	46304	14.7	17.5	64.1	62.4
zeroconf,r=t,N=20,K=2	5856	18.2	21.9	24.9	30.6
zeroconf,r=t,N=20,K=4	6085	19.2	23.4	25.7	31.7
zeroconf,r=t,N=20,K=6	6101	19.3	23.4	25.7	31.7
zeroconf,r=t,N=20,K=8	6314	20.6	24.9	26.8	32.9
zeroconf,r=t,N=1000,K=2	5856	18.2	21.9	24.9	30.6
zeroconf,r=t,N=1000,K=4	6085	19.2	23.4	25.7	31.7
zeroconf,r=t,N=1000,K=6	6101	19.3	23.4	25.7	31.7
zeroconf,r=t,N=1000,K=8	6314	20.6	24.9	26.8	32.9
zeroconf,r=f,N=20,K=2	59919	30.6	32.3	40.7	44.8
zeroconf,r=f,N=20,K=4	97931	26.0	30.8	38.2	47.4
zeroconf,r=f,N=20,K=6	184183	42.5	40.1	59.6	56.4
zeroconf,r=f,N=20,K=8	285854	48.1	51.9	64.3	66.1
zeroconf,r=f,N=1000,K=2	59919	30.6	32.3	40.7	44.8
zeroconf,r=f,N=1000,K=4	97931	26.0	30.8	38.2	47.4
zeroconf,r=f,N=1000,K=6	184183	42.5	40.1	59.6	56.4
zeroconf,r=f,N=1000,K=8	285854	48.1	51.9	64.3	66.1
zeroconf_dl,r=t,dl=10,N=1000,K=1	9015	22.9	29.8	31.6	42.5
zeroconf_dl,r=t,dl=20,N=1000,K=1	10101	26.6	33.4	34.6	44.7
zeroconf_dl,r=t,dl=30,N=1000,K=1	10909	31.5	38.1	39.3	50.2
zeroconf_dl,r=t,dl=40,N=1000,K=1	12118	34.6	40.3	40.4	50.2
zeroconf_dl,r=t,dl=50,N=1000,K=1	12992	38.0	44.1	43.6	51.6
zeroconf_dl,r=f,dl=10,N=1000,K=1	27810	22.0	34.7	31.1	40.4
zeroconf_dl,r=f,dl=20,N=1000,K=1	67440	35.2	49.7	52.5	59.1
zeroconf_dl,r=f,dl=30,N=1000,K=1	110040	54.4	51.1	64.8	63.5
zeroconf_dl,r=f,dl=40,N=1000,K=1	158224	57.6	55.9	67.2	73.5
zeroconf_dl,r=f,dl=50,N=1000,K=1	203142	64.7	62.5	72.5	77.4

Impact of reordering on the model checking time (PRISM benchmark suite)

We present here statistics for two examples from the PRISM benchmark suite, detailing the impact of the automatic reordering on the model checking time for these instances.

Table 9. Statistics for the "egl" case study with "unfairA" query (L=8), MTBDD engine and "top" initial variable ordering. Reordering with "globalize" and "explode bits".

Instance	States	reordered MTBDD	reduction in %	t_{reorder}	t_{query} original	t_{query} reordered
N=5	156 670	21 797	56.9	11.2 s	3.9 s	1.9 s
N=10	317 718 526	72 001	40.1	47.7 s	20.6 s	12.1 s
N=15	486 405 046 270	146 352	31.9	115.5 s	53.0 s	39.3 s
N=20	663 005 511 548 926	238 028	28.8	218.8 s	104.2 s	80.3 s

Table 10. Statistics for the "fms" case study with "productivity" query, HYBRID engine. Reordering with "explode bits".

Instance	States	reordered MTBDD	reduction in %	t_{reorder}	t_{query} original	t_{query} reordered
N=5	152 712	20 681	59.4	5.0 s	15.6 s	9.9 s
N=6	537 768	30 228	61.3	8.824	68.334	52.337
N=7	1 639 440	40 610	66.4	16.2 s	227.9 s	179.2 s
N=8	4 459 455	72 908	66.1	28.6 s	808.8 s	596.7 s
N=9	11 058 190	102 514	66.8	66.6 s	3458.0 s	1575.7 s

The reordering method was chosen to be the one that provided the best reduction. For each instance, the tables present statistics for the number of states in the reachable part of the state space, the size of the MTBDD for the transition/rate matrix after reordering, the reduction in size of the MTBDD representation of the matrix due to reordering and the time spent for reordering. It has to be kept in mind that the reordering time can be amortized over multiple runs of the model checker and multiple queries by reusing the reordered model. Additionally, the tables depict the time for computing the respective query, once for the original, unreordered model and once for the reordered model.

It can be seen that the more compact MTBDD representation leads here to a reduction in the model checking time as well, with the largest relative improvement achieved for N=8 in the "fms" case study.

C Further benchmark details: quantile computations

All quantile experiments were carried out on a server with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384GB of RAM running Linux.

We provide details here for our experiments revisiting the benchmark case studies of [5]. We computed the quantile value of the initial state of the model. Our implementation supports the computation of quantiles for multiple probability thresholds simultaneously, as the intermediate values in the iterative approach can be used to determine the quantile value for thresholds with smaller quantile values. For the computations we set a memory limit of 30GB for the Java Runtime-Environment and allowed the (MT)BDD-library CUDD to use 20GB of the main memory for handling the involved binary decision diagrams.

We provide here a brief description for the columns in the tables in the following sections. "Instance" denotes the particular model instance, i.e., the number N of processes for the respective protocol. "States" is the number of states in the state space of the MDP and "MTBDD size" provides the number of nodes in the transition matrix. The number of quantile iterations in the computation is listed under "iter.", with the time for building and for the computation of the quantile given by t_{build} and t_{query}, respectively.

C.1 Self-stabilising protocol

The self-stabilising protocol by Israeli and Jalfon[5] is modeled as an MDP for N equal processes organized in a ring, each having a token at the beginning and aiming to randomly send and receive tokens until the ring is in a stable state, i.e., only one process has a token.

The self-stabilising protocol works as follows: At the beginning, the N processes all are active, i.e., have one token assigned to each of them. Then, the protocol is reassigning tokens step-wise to the processes, where at each step, one active process is selected through a scheduler to randomly pass its token either to its right or left neighbor. If a process has two tokens, the tokens are merged into one. The ring is in a stable state if exactly one process owns the last remaining token.

For a given N and probability threshold p, we compute the following quantiles:

Q1: "The minimal number of steps required for reaching a stable state with probability of at least p for the best scheduler."

Q2: "The minimal number of steps required for reaching a stable state with probability of at least p for all schedulers."

We computed the quantiles for $N \in \{3, \ldots, 18\}$ with probability thresholds $p \in \{0, 0.01, 0.05, 0.1, 0.2, \ldots, 0.9, 0.95, 0.99\}$, using the multi-threshold approach, i.e., computing the quantile for all thresholds in one run. The statistics of our computations are presented in Table 11 and Table 12.

[5] `http://www.prismmodelchecker.org/casestudies/self-stabilisation.php#ij`

Table 11. Statistics for quantile **Q1**, self-stabilising protocol

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput.					
						SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 3	7	36	9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 4	15	69	17	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 5	31	109	28	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 6	63	153	41	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 7	127	201	57	0.1	0.0	0.0	0.1	0.0	0.1	0.0	0.1
N = 8	255	253	75	0.2	0.0	0.0	0.1	0.0	0.1	0.0	0.1
N = 9	511	309	95	0.3	0.1	0.0	0.3	0.0	0.3	0.0	0.3
N = 10	1023	369	118	0.4	0.1	0.0	0.9	0.0	0.8	0.0	0.8
N = 11	2047	433	144	0.5	0.2	0.0	2.5	0.0	2.3	0.0	2.0
N = 12	4095	501	172	0.5	0.4	0.0	6.0	0.0	5.9	0.0	6.1
N = 13	8191	573	202	0.7	0.8	0.0	18.1	0.0	16.2	0.0	15.4
N = 14	16383	649	235	1.0	1.7	0.0	52.3	0.0	45.4	0.0	42.9
N = 15	32767	729	271	1.6	3.9	0.0	119.1	0.0	114.7	0.0	159.3
N = 16	65535	813	308	2.6	10.4	0.0	404.6	0.0	382.6	0.0	366.0
N = 17	131071	901	349	4.8	22.9	0.0	979.5	0.0	820.9	0.0	755.3
N = 18	262143	993	392	9.8	53.8	0.0	2135.2	0.0	2865.1	0.0	2240.3

Table 12. Statistics for quantile **Q2**, self-stabilising protocol

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput.					
						SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 3	7	36	9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 4	15	69	18	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0
N = 5	31	109	29	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0
N = 6	63	153	44	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0
N = 7	127	201	61	0.1	0.1	0.0	0.1	0.0	0.1	0.0	0.1
N = 8	255	253	82	0.2	0.1	0.0	0.2	0.0	0.2	0.0	0.2
N = 9	511	309	105	0.3	0.3	0.0	0.4	0.0	0.4	0.0	0.4
N = 10	1023	369	131	0.4	0.3	0.0	1.3	0.0	1.1	0.0	1.1
N = 11	2047	433	160	0.5	0.4	0.0	3.0	0.0	3.5	0.0	3.0
N = 12	4095	501	192	0.5	0.7	0.0	8.4	0.0	8.1	0.0	7.9
N = 13	8191	573	227	0.7	1.2	0.0	22.2	0.0	21.5	0.0	20.9
N = 14	16383	649	265	1.0	2.3	0.0	65.6	0.0	59.1	0.0	78.7
N = 15	32767	729	306	1.6	4.8	0.0	161.5	0.0	159.8	0.0	180.8
N = 16	65535	813	349	2.6	11.4	0.0	449.9	0.0	448.2	0.0	459.6
N = 17	131071	901	396	4.8	29.9	0.0	1458.5	0.0	1303.9	0.0	1113.4
N = 18	262143	993	445	9.8	65.3	0.0	3527.2	0.0	2851.3	0.0	2809.7

C.2 Asynchronous leader-election protocol

This protocol aims to elect a leader, i.e., a uniquely designated process out of N equal processes organised in a ring structure by sending messages to the other processes. We considered an asynchronous variant of such a protocol, developed by Itai and Rodeh and also described in the context of probabilistic model checking within the benchmark suite of PRISM[6].

All processes are located in a ring and are initially all active. Inactive processes still pass along messages to their neighbors. The protocol operates in rounds consisting of three phases. In the first phase each active process probabilistically selects its preference, i.e., whether to remain active or to become inactive. Then, a process communicates its preference to the next process along the ring. A process is then allowed to become inactive only if the active process preceding it in the ring prefers to remain active. In a third phase, the processes send a counter around the ring to determine if only one active process remains, which then becomes the leader. If no unique process remains active the protocol performs another round.

One question of interest for this protocol is the minimal number of steps or rounds r required to elect a leader with a certain probability p for some/all schedulers:

Q3: "The minimal number of rounds required for electing a leader with probability of at least p for the best scheduler."
Q4: "The minimal number of rounds required for electing a leader with probability of at least p for all schedulers."
Q5: "The minimal number of steps required for electing a leader with probability of at least p for the best scheduler."
Q6: "The minimal number of steps required for electing a leader with probability of at least p for all schedulers."

We computed the quantiles for $N \in \{2, \ldots, 9\}$ with probability thresholds $p \in \{0, 0.01, 0.05, 0.1, 0.2, \ldots, 0.9, 0.95, 0.99\}$. The statistics of our computations are presented in Tables 13 to 16.

[6] `http://www.prismmodelchecker.org/casestudies/asynchronous_leader.php`

Table 13. Statistics for quantile **Q3**, asynchronous leader election

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	36	635	8	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 3	364	3355	10	0.2	0.1	0.0	0.2	0.1	0.2	0.1	0.3
N = 4	3172	10760	11	0.4	0.1	0.1	1.1	0.1	1.4	0.1	1.7
N = 5	27299	31430	12	1.1	0.8	0.4	5.4	0.6	7.6	0.5	12.8
N = 6	237656	77999	12	5.3	8.2	1.5	39.1	1.8	34.2	1.8	52.6
N = 7	2095783	180383	13	63.1	109.0	5.9	253.8	7.0	197.9	7.6	247.3
N = 8	18674484	392093	13	1633.2	1098.2	20.6	2046.2	24.6	1443.1	21.9	916.6
N = 9	167748115	868257	14	–	–	106.1	20217.9	92.3	13412.5	92.9	4945.8

Table 14. Statistics for quantile **Q4**, asynchronous leader election

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	36	635	8	0.1	0.1	0.0	0.1	0.0	0.1	0.0	0.1
N = 3	364	3355	10	0.2	0.2	0.0	0.2	0.1	0.2	0.1	0.4
N = 4	3172	10760	11	0.4	0.4	0.1	0.9	0.1	1.2	0.1	2.4
N = 5	27299	31430	12	1.1	1.4	0.4	9.1	0.6	12.0	0.5	21.5
N = 6	237656	77999	12	5.3	9.3	1.5	47.6	1.8	50.8	1.8	113.0
N = 7	2095783	180383	13	63.1	100.8	5.9	360.4	7.0	272.7	7.6	675.2
N = 8	18674484	392093	13	1633.2	1155.9	20.6	3283.9	24.6	1873.9	21.9	3616.3
N = 9	167748115	868257	14	–	–	106.1	33645.4	92.3	17273.9	92.9	26244.7

Table 15. Statistics for quantile **Q5**, asynchronous leader election

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	36	635	43	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 3	364	3355	74	0.2	0.1	0.0	0.1	0.1	0.1	0.1	0.1
N = 4	3172	10760	104	0.4	0.1	0.1	1.4	0.1	1.9	0.1	1.5
N = 5	27299	31430	139	1.1	0.6	0.4	11.6	0.6	12.4	0.5	12.8
N = 6	237656	77999	168	5.3	7.9	1.5	78.9	1.8	67.8	1.8	67.4
N = 7	2095783	180383	206	63.1	83.2	5.9	358.8	7.0	369.8	7.6	402.7
N = 8	18674484	392093	238	1633.2	891.8	20.6	1228.4	24.6	1307.4	21.9	1448.8
N = 9	167748115	868257	279	–	–	106.1	7751.9	92.3	5728.1	92.9	7545.6

Table 16. Statistics for quantile **Q6**, asynchronous leader election

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	36	635	43	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N = 3	364	3355	74	0.2	0.1	0.0	0.2	0.1	0.2	0.1	0.2
N = 4	3172	10760	104	0.4	0.1	0.1	1.7	0.1	2.3	0.1	1.9
N = 5	27299	31430	139	1.1	0.6	0.4	14.2	0.6	17.1	0.5	15.6
N = 6	237656	77999	168	5.3	7.3	1.5	102.6	1.8	88.9	1.8	83.7
N = 7	2095783	180383	206	63.1	95.7	5.9	440.4	7.0	464.3	7.6	468.1
N = 8	18674484	392093	238	1633.2	887.9	20.6	1771.8	24.6	2158.3	21.9	1853.9
N = 9	167748115	868257	279	–	–	106.1	9906.6	92.3	8576.2	92.9	9404.0

C.3 Energy-aware job-scheduling protocol

This protocol [5] describes rules for the management of a shared resource when multiple processes want to access the mentioned resource concurrently.

It models a system of N processes which need to enter a critical section in order to perform tasks, each within a given deadline. Access to the critical section is exclusively granted by a scheduler, which selects the process which can access the resource out of a pool of processes which have requested to enter previously. When a process states such a request, a deadline counter is set and decreased over time even if the process did not enter the critical section yet. Since computing a task also requires a certain amount of time in the critical section, deadlines can be exceeded. Utility is hence provided in terms of tasks finished without exceeding their deadline. Each process consumes energy, especially if it is in the critical section, and the global energy consumption equals the sum of energy consumed by all processes. Additional dependencies between utility and energy arise as the scheduler can activate a turbo mode for the critical section, doubling the computation speed but as a drawback also tripling the energy consumption. For more in-depth details of the protocol we refer to [5].

For our calculations here, we fixed a deadline distribution $\delta = (\frac{1}{3}{:}7, \frac{2}{3}{:}9)$, a computation distribution $\gamma = (\frac{1}{3}{:}2, \frac{2}{3}{:}3)$ (time needed for computation in critical section) and a timer distribution $\tau = (\frac{1}{3}{:}5, \frac{2}{3}{:}4)$ (time between leaving the critical section and a subsequent request for the shared resource). N denotes the number of involved processes.

Minimal energy. Here, we are interested in computing the following quantile:

Q7: "The minimal energy that is required for gaining at least N utility with probability of at least p for the best scheduler."

We augmented our basic model with a module that is counting the finished tasks that did not violate their deadline.

We computed quantiles for $N \in \{2, \ldots, 6\}$ with probability thresholds $p \in \{0, 0.01, 0.05, 0.1, 0.2, \ldots, 0.9, 0.95, 0.99\}$. Table 17 shows the statistics of the respective quantile calculations.

Table 17. Statistics for quantile **Q7**

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	917	2169	75	0.3	0.3	0.1	0.2	0.0	0.2	0.0	0.2
N = 3	16341	11589	177	1.2	0.9	0.2	7.0	0.2	6.9	0.2	6.3
N = 4	368521	46756	226	11.9	12.1	1.3	90.6	1.1	81.5	1.3	87.8
N = 5	6079533	187458	302	334.1	285.9	7.8	1196.2	7.2	1128.0	7.9	1142.9
N = 6	44072357	507805	416	–	–	21.7	3808.4	22.8	4045.9	25.0	3962.8

Maximal utility. Here, we are interested in the following quantiles:

Q8: "The maximal utiliy that can be achieved consuming an energy-budget of $50 \cdot N$ with probability of at least p for the best scheduler."
Q9: "The maximal utiliy that can be achieved consuming an energy-budget of $50 \cdot N$ with probability of at least p for all schedulers."

Again, N is the numer of processes, ranging from 2 to 6 and the probability thresholds are $p \in \{0.01, 0.05, 0.1, 0.2, \ldots, 0.9, 0.95, 0.99\}$.

Note that we only consider here the utility-gain and the energy-consumption of a specific fixed process instead of all processes as before.

The respective statistics of our calculations can be seen in Table 18 and in Table 19.

Table 18. Statistics for quantile **Q8**

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	12828	6310	7	0.7	0.8	0.2	1.2	0.1	1.2	0.1	1.0
N = 3	143155	11247	8	2.5	3.1	0.3	6.0	0.3	4.7	0.3	7.3
N = 4	872410	14007	11	11.9	41.1	0.5	17.1	0.5	14.2	0.5	32.2
N = 5	3049471	25363	13	62.5	398.6	0.8	86.0	0.9	74.3	0.9	104.6
N = 6	7901694	38911	15	210.0	2375.5	1.8	390.6	1.8	378.1	1.3	317.1

Table 19. Statistics for quantile **Q9**

Instance	States	MTBDD size	iter.	Explicit		symbolic quantile comput. SemiHybrid		SemiSparse		Mtbdd	
				t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}	t_{build}	t_{query}
N = 2	12828	6310	5	0.7	0.2	0.2	1.2	0.1	1.2	0.1	1.0
N = 3	143155	11247	5	2.5	1.5	0.3	5.8	0.3	4.4	0.3	18.7
N = 4	872410	14007	2	11.9	11.2	0.5	2.3	0.5	2.4	0.5	1.5
N = 5	3049471	25363	2	62.5	47.2	0.8	13.3	0.9	15.5	0.9	2.5
N = 6	7901694	38911	2	210.0	157.8	1.8	64.9	1.8	66.4	1.3	4.6