Formal Methods for Probabilistic Energy Models

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Marcus Daum
geboren am 25. Mai 1983 in Leipzig

Gutachter:
Prof. Dr. rer. nat. Christel Baier
Technische Universität Dresden

Dr. David Parker
University of Birmingham, United Kingdom

Tag der Verteidigung: 27. Juni 2018

Abstract

The energy consumption that arises from the utilisation of information processing systems adds a significant contribution to environmental pollution and has a big share of operation costs. This entails that we need to find ways to reduce the energy consumption of such systems. When trying to save energy it is important to ensure that the utility (e.g., user experience) of a system is not unnecessarily degraded, requiring a careful trade-off analysis between the consumed energy and the resulting utility. Therefore, research on energy efficiency has become a very active and important research topic that concerns many different scientific areas, and is as well of interest for industrial companies.

The concept of quantiles is already well-known in mathematical statistics, but its benefits for the formal quantitative analysis of probabilistic systems have been noticed only recently. For instance, with the help of quantiles it is possible to reason about the minimal energy that is required to obtain a desired system behaviour in a satisfactory manner, e.g., a required user experience will be achieved with a sufficient probability. Quantiles also allow the determination of the maximal utility that can be achieved with a reasonable probability while staying within a given energy budget. As those examples illustrate important measures that are of interest when analysing energy-aware systems, it is clear that it is beneficial to extend formal analysis-methods with possibilities for the calculation of quantiles.

In this monograph, we will see how we can take advantage of those quantiles as an instrument for analysing the trade-off between energy and utility in the field of probabilistic model checking. Therefore, we present algorithms for their computation over Markovian models. We will further investigate different techniques in order to improve the computational performance of implementations of those algorithms. The main feature that enables those improvements takes advantage of the specific characteristics of the linear programs that need to be solved for the computation of quantiles. Those improved algorithms have been implemented and integrated into the well-known probabilistic model checker Prism. The performance of this implementation is then demonstrated by means of different protocols with an emphasis on the trade-off between the consumed energy and the resulting utility. Since the introduced methods are not restricted to the case of an energy-utility analysis only, the proposed framework can be used for analysing the interplay of cost and its resulting benefit in general.

Acknowledgements

First of all I want to express my gratitude to Christel Baier for her professional guidance and support. She always managed to find some time and patience when I reached a point where I did not know how to continue. Without her support this interesting journey would have been impossible for me.

I also thank the Collaborative Research Center HAEC[1] for financial support and all the people from the different participating projects for the fruitful collaborations that gave feedback in order to extend the formal methods to support different fields of application. This way the collaborations helped to guide the formation of the framework which constitutes the core of the monograph you are looking at. Especially the collaboration with project B04 supervised by Hermann Härtig was very pleasant and influenced the herewith presented work to a huge extent.

I am really thankful for the support of Karina, Kerstin and Sandy who helped to get along with the bureaucracy and all the other things no one thinks about until one has to deal with it. All three thought of those things long before myself and so they made life a lot easier. I thank all the other PhD-students (Clemens, Daniel, David, Linda, Lisa, Philipp, Sascha and Steffen) and post-docs (Frank, Joachim, Michael and Sascha) who shared their time and expertise with me. The majority of our discussions pushed me a bit further to manage this interesting exercise. Most of the time was a lot of fun and I really enjoyed our lunch times (although something tells me that you are in a way glad that you are not bothered with my speed at lunch any longer). Joachim and Sascha really helped me to keep focussed on my tasks and to finally cross the finishing line. And of course Steffen needs to be mentioned because of his special awesomeness ;-). I am really glad that I had the chance to work with you and am so thankful for your support.

I also appreciate my new colleagues (especially André, Christoph, Dirk, Hendrik, Ina, Michael, Sebastian, Stefan and Torsten) at the Fraunhofer IVI[2] who integrated me into the very nice team and who also supported me during the final phase of this thesis. I want to give special thanks to Dirk for some very interesting discussions.

The year 2017 was a very special one where so much happened which I would not have managed without the loving support of my family (especially my wife, my children, my parents, my brother and his family, my favourite aunt and uncle, …). I am really thankful for all of your support and I hope that you all feel a bit relieved now. I thank

---

[1] `http://tu-dresden.de/sfb912`, retrieved 28th March 2018

[2] `https://www.ivi.fraunhofer.de/de/forschungsfelder/intelligente-verkehrssysteme/ticketing-und-tarife.html`, retrieved 28th March 2018

the most important persons in my life for being simply there and living their lives by my side. I am so glad that my sons successfully managed all the critical moments they had to face in their young lives, and am entirely happy that they are simply there. I am so proud of my daughter that she is such a sunshine and that she has a feeling for knowing when it is the exact right moment for being quiet and peaceful. Under all the other circumstances she is just a normal fun loving child ;-). I love my wife and am thankful for her support, especially for taking all the burdens on her in order to manage that we finally have the ability to raise happy children. Of course, I am grateful for the professionality of all the physicians and nurses who did the best they could in order to guide my small family through the toughest time we were confronted with so far. Nils, Anne and Tobi deserve thanks for giving me some moments to catch a breath.

Right now my thoughts are with my brother who needs to become healthy as soon as possible. Hurry up!

Last but not least I thank the fire department of Dresden who entered the stage when the *Amt für Stadtgrün und Abfallwirtschaft* of the city of Dresden failed to do their duty.

Contents

1 Introduction 1

1.1 Related work . . . 2

1.2 Contribution and outline . . . 14

2 Preliminaries 17

3 Reward-bounded reachability properties and quantiles 23

3.1 Essentials . . . 23

3.2 Dualities . . . 25

3.3 Upper-reward bounded quantiles . . . 27

3.3.1 Precomputation . . . 28

3.3.2 Computation scheme . . . 28

3.3.3 Qualitative quantiles . . . 30

3.4 Lower-reward bounded quantiles . . . 34

3.4.1 Precomputation . . . 34

3.4.2 Computation scheme . . . 37

3.5 Energy-utility quantiles . . . 38

3.6 Quantiles under side conditions . . . 40

3.6.1 Upper reward bounds . . . 41

3.6.2 Lower reward bounds . . . 45

3.6.2.1 Maximal reachability probabilities . . . 45

3.6.2.2 Minimal reachability probabilities . . . 49

3.7 Reachability quantiles and continuous time . . . 52

3.7.1 Dualities . . . 53

4 Expectation Quantiles 55

4.1 Computation scheme . . . 56

4.2 Arbitrary models . . . 57

4.2.1 Existential expectation quantiles . . . 59

4.2.2 Universal expectation quantiles . . . 64

5 Implementation 67

5.1 Computation optimisations . . . 67

5.1.1 Back propagation . . . 68

5.1.2 Reward window . . . 69

5.1.3 Topological sorting of zero-reward sub-**MDP**s . . . 71

5.1.4 Parallel computations . . . 71

5.1.5 Multi-thresholds . . . 73
5.1.6 Multi-state solution methods . . . 73
5.1.7 Storage for integer sets . . . 77
5.1.8 Elimination of zero-reward self-loops . . . 78
5.2 Integration in Prism . . . 79
5.2.1 Computation of reward-bounded reachability probabilities . . . 81
5.2.2 Computation of quantiles in **CTMC**s . . . 82

6 Analysed Protocols 85
6.1 Prism Benchmark Suite . . . 87
6.1.1 Self-Stabilising Protocol . . . 87
6.1.2 Leader-Election Protocol . . . 96
6.1.3 Randomised Consensus Shared Coin Protocol . . . 107
6.2 Energy-Aware Protocols . . . 115
6.2.1 Energy-Aware Job-Scheduling Protocol . . . 115
6.2.1.1 Energy-Aware Job-Scheduling Protocol with side conditions . . . 125
6.2.1.2 Energy-Aware Job-Scheduling Protocol and expectation quantiles . . . 131
6.2.1.3 Multiple shared resources . . . 132
6.2.2 Energy-Aware Bonding Network Device (eBond) . . . 137
6.2.3 HAECubie Demonstrator . . . 142
6.2.3.1 Operational behaviour of the protocol . . . 144
6.2.3.2 Formal analysis . . . 150

7 Conclusion 155
7.1 Classification . . . 156
7.2 Future prospects . . . 157

Bibliography 159

List of Figures 169

List of Tables 171

1 Introduction

The energy consumption that arises from the utilisation of information processing systems adds a significant contribution to environmental pollution and has a big share of operation costs [Cla11]. Since the technological progress of our society is backed by the usage of computing devices and their possibilities of data processing, the caused negative effects will even further increase in the near future. An immediate effect of the growing demand for services that are based on computing infrastructure is an increase in the consumption of energy. Unfortunately, the generation of this energy can not completely rely on (environmentally) sustainable methods these days, and is therefore directly linked with extra burdens on the environment. As a consequence the environmental pollution will pose a challenge even for future generations. Among other duties it is important to find ways to improve the energy efficiency of the information processing systems in order to counteract these negative effects. Therefore, the research on energy efficiency has become a very active and important research topic that concerns many different scientific areas, and is as well of interest for industrial companies.

The Collaborative Research Center 912[1] "Highly Adaptive Energy-Efficient Computing" (HAEC) is a research project at the *Technische Universität Dresden* and focusses on the energy efficiency of future server architectures. One of the problems that needs to be addressed by HAEC is the utilisation of computing-power in a nearly optimal way, meaning that it is of importance to ensure that the utility (e.g., user experience) of the computing devices is not unnecessarily degraded when trying to reduce the energy consumption of the system. Therefore, it is required to do a careful trade-off analysis between the consumed energy and the produced utility.

Formal methods like (probabilistic) model checking can support the analysis of this trade-off by using well-established procedures based on mathematical formalisms. Those methods have shown to be helpful for analysing a large number of systems for varying areas of application [KNP05; Gar14]. In order to do a trade-off analysis tailored to the needs of energy efficiency as described above, we want to minimise the energy consumption and at the same time maximise the gained utility. Therefore, we want to exploit the concept of quantiles which is already well-known in mathematical statistics. Its benefits, however, for the formal quantitative analysis of probabilistic systems have been noticed only recently. For instance, with the help of quantiles it is possible to reason about the minimal energy that is required to obtain a desired system behaviour in a satisfactory manner, e.g., a required user experience will be achieved with a sufficient probability. Quantiles also allow the determination of the maximal utility that can be achieved with a reasonable probability while staying within

---

[1] `http://tu-dresden.de/sfb912`, retrieved 28th March 2018

a given energy budget. In this monograph we will present how we can take advantage of quantiles as an instrument for the trade-off analysis between energy and utility in the field of probabilistic model checking, and present algorithms for the computation of quantiles in Markovian models. We will further show results of implementations of those algorithms by means of different case studies that are driven by the cooperations in the Collaborative Research Center 912 HAEC.

1.1 Related work

Model checking is a formal framework consisting of techniques that allow to provide guarantees for the behaviour of a given system. The utilised methods offer support for a huge variety of systems used in many different application domains. Model checking has its origin in the works of Clarke and Emerson [CE82], and Queille and Sifakis [QS82]. It has since been developed rather quickly, and it has been extended in order to cope with probabilistic behaviour. This extension was necessary since the investigated systems revealed that stochastic phenomena occur quite naturally, and it has led to the development of *probabilistic model checking* (or PMC for short). As an example the usage of probabilistic behaviour is essential for the adequate modelling of erroneous system behaviour in order to provide a feasible analysis for situations where with some probability a normal operation of the system cannot be ensured any longer. Randomisation also helps to increase the anonymity of data transmission, and therefore improves the security of communication protocols. Other examples are distributed coordination protocols where coin tossing serves as a symmetry breaker, e.g., in order to elect a leader out of a number of possible candidates.

For the verification process the (probabilistic) model checker needs two types of input. The first one is a formal model of the system under consideration, and the second one is a formal specification of characteristics that should or should not hold for the system.

Usually the model is given as some kind of transition system describing the operational behaviour of the system. Probabilistic model checking is tailored such that the model can be equipped with probabilities in order to express likelihoods for a specific behaviour of the system. Due to the application of probabilities the utilisation of Markovian models like *discrete-time Markov chains* (**DTMC**s) or *Markov decision processes* (**MDP**s) are well established in the field of PMC. A **DTMC** is a model where time elapses using discrete steps, and it can be therefore seen as a formalism which is time abstract. Its behaviour is given probabilistically, and this entails that the successor of a state will be specified in accordance with a given probability distribution. **MDP**s are an extension of **DTMC**s that introduce nondeterminism, and therefore allow us to formalise behaviour which is influenced by the environment of the system and thus not manageable by the system itself. So, the usage of nondeterminism enables the modelling of uncontrollable or unpredictable behaviour. Nondeterminism adds a certain degree of freedom to the reaction of the system by not forcing a specific operation in particular situations. This allows the system to be not restricted in its behaviour, and as a consequence it is possible to determine the optimal behaviour of the system in those situations. There also exist formalisms that support the continuous elapse of time, the so-called *continuous-time Markov chains* (**CTMC**s). This allows us to express timings in a more natural way than is possible when using discrete time domains. Those models are also invariant against changes of the time scale.

The formal specification of the system behaviour that has to be analysed is usually given as a formula of some kind of temporal logic, like LTL [Pnu77] or PCTL [HJ94]. LTL (*Linear Temporal Logic*) is an extension of the classical propositional or predicate logic, which allows us to refer to the infinite behaviour of a system by providing temporal operators like $\Diamond$ (finally at some time in the future) and $\Box$ (from now on and forever), and combinations thereof like $\Diamond\Box$ (eventually forever) and $\Box\Diamond$ (infinitely often). Therefore, LTL constitutes an intuitive way of reasoning about executions of a system. PCTL (*Probabilistic Computation Tree Logic*) is a probabilistic extension of CTL, which formulates statements concerning the states of a model. Unlike CTL it does not quantify about universal or existential path characteristics. Instead, it is equipped with a probability operator that allows us to reason about the likelihood for characteristics of the paths starting in some state.

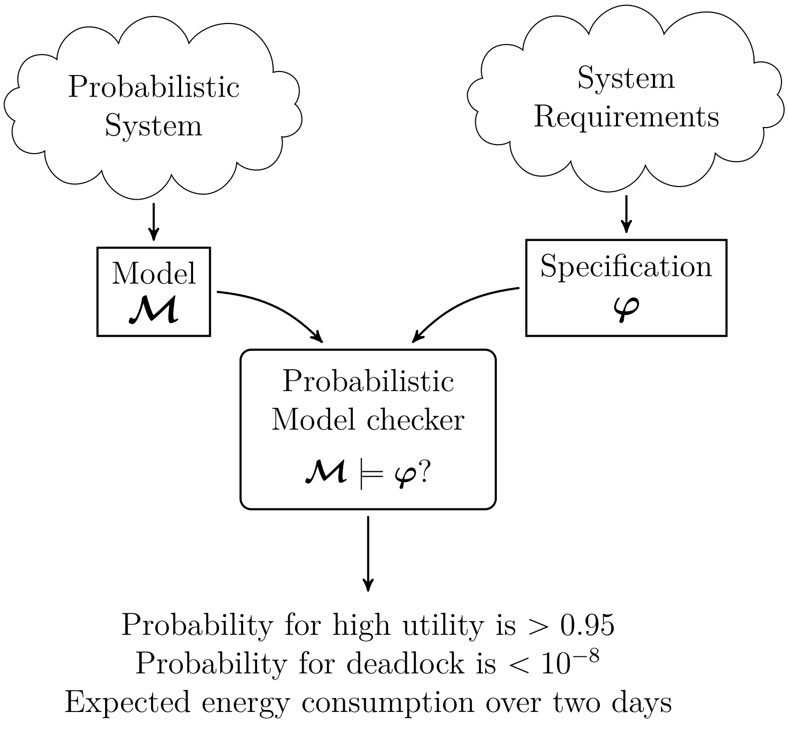
Probabilistic System
System Requirements
Model $\mathcal{M}$
Specification φ
Probabilistic Model checker
$\mathcal{M} \models \varphi$?
Probability for high utility is > 0.95
Probability for deadlock is $< 10^{-8}$
Expected energy consumption over two days

Figure 1.1: Schematic overview of probabilistic model checking

Given both the model and the specification as input, a probabilistic model checker then does the desired analysis by calculating the probability for the specified behaviour. This resulting probability expresses the likelihood of the behaviour that is of interest for the model under consideration. The verification procedure is sketched in Figure 1.1, and works fully automatically by exploring all states of the system, and can be used to, e.g., calculate the probability that some “bad” system behaviour occurs, or to compute the expected energy consumption of the system over a specific time span. The different types of models and specifications that are used for the verification procedure require adapted methods in order to provide the desired results. For instance, the usage of LTL formulas will be handled by transferring the formula under consideration into an appropriate ω-automaton[2] and building the product of the given system and this automaton. The verification then concentrates on the analysis of the resulting product. This approach is called the *automa-theoretic approach* [Var85; VW86; CY95] and constitutes the standard methodology for the analysis of specifications given as LTL[3]. It is known that model checking of LTL is **PSPACE**-complete for the analysis of **DTMC**s and **2EXPTIME**-complete for **MDP**s [Var85; CY95]. On the other hand, when considering formulas specified in PCTL one can at first determine all the contained subformulas of the given formula, and then compute the satisfiable states for each subformula in a recursive manner. This gives an algorithm for the analysis of a PCTL-formula with a running time that is polynomial in the size of the model and the size of the formula [HJ94; BA95; BK98]. [HJ94] introduces PCTL and presents the treatment of PCTL over **DTMC**s, whereas [BA95] extends the algorithmic treatment for models augmented with nondeterminism, i.e., **MDP**s. [BK98] in particular considers the usage of fairness for model checking of PCTL. Fairness allows one to exclude behaviour that does not reflect a realistic operation of the system. This unrealistic behaviour may be a result of the resolution of the nondeterminism in the model, and it is thus not in the scope of the desired analysis. An illustrative example is the case of a process scheduler that prefers a specific process, and therefore access to a shared resource may rarely be granted to the other involved processes. Fairness allows one to abstract from this behaviour for the analysis, e.g., when one wants to make sure that the shared resource continues its operation in a valid system state after every involved process has finished its specific task using the resource. Since fairness constraints (like, e.g., *strong fairness* or *weak fairness*) can be expressed using ordinary LTL, the analysis of fairness in the case of LTL boils down to usual checking of some specific LTL formula. This is not the case for PCTL, and therefore the subformulas that constitute the fairness condition will be replaced by the corresponding satisfying states in a specific preprocessing step. Those state sets are then used by the analysis in order to consider the fair executions of the system.

When doing the analysis for a model enriched with nondeterminism it is required to resolve the nondeterminism by a *scheduler* (sometimes also called strategy). This then allows us to reason about the probability for the specified behaviour under the given scheduler, and schedulers also facilitate the possibility of computing the minimal or maximal probabilities for the behaviour. When the specification represents a desired “good” system behaviour the maximal probability serves as a measure allowing to see how the system behaves in the optimal case, and the minimal probability gives an

---

[2] The size of this automaton can be exponential in the size of the given formula.

[3] Section 3.6 utilises this approach in order to analyse quantiles with side conditions.

estimation of the behaviour in the worst case. If, on the other hand, an undesired "bad" behaviour is analysed then the maximal probability shows to what extent the undesired behaviour can occur in the worst case. Here, the minimal probability is used in order to see how likely it is that the undesired behaviour occurs at any time. The usage of nondeterminism fits the targeted trade-off between consumed energy and gained utility, since this allows us to reason about the energy budget one has to pay under any circumstances and to which extent it is possible to reduce this budget. On the other hand, we can reason about the gained utility if the system operates in its most efficient way and we can also reason about the outcome of the system in the worst case.

An *end component* is a specific part of a given **MDP** whose nodes are strongly connected and where the corresponding actions cannot reach a state that is outside of the component. A *maximal end component* is an end component which is not contained in another end component. The concept of end components is essential when considering the analysis of **MDP**s[4], since they play an important role for the analysis of a multitude of specifications. Since we are analysing infinite behaviour (or paths) over finite models, there must exist some point when the models are forced to enter those components. In order to receive the necessary information one has to investigate those end components. A fundamental work which covers very important aspects for the analysis of end components can be found in [Alf98]. Alfaro presents graph algorithms in order to compute the maximal end components of a model (the presented algorithm has a running time that is quadratic in the size of the given **MDP**), and some fundamental statements that are linked to end components and have an impact on many model-checking algorithms.

Model-checking techniques are tailored such that they support the annotation of the states and transitions of the Markovian models with non-negative values. Those values are called *rewards* and can be for example seen as the utility that is achievable when reaching a specific state or using a particular transition. It is also possible to interpret these rewards as a cost which needs to be paid in order to use a state or transition. An example for such a cost is the energy which will be consumed. Therefore, rewards allow us to formalise the energy consumption of the system and also the utility that can be achieved by the execution of the system. This enables us to investigate the trade-off between the system's energy consumption and its gained utility. [AHK03] introduces the logic PRCTL, which is an extension of PCTL in order to take the treatment of rewards into account. This extension allows one to reason about the expected reward rate per time unit until the model performed a desired number of steps. It is as well possible to analyse the instantaneous reward at a specific time instant, or the expected accumulated reward. The running time of the presented model-checking algorithm is polynomial in the size of the analysed system and the number of desired steps, when only rationals or integers are considered as rewards (which is sufficient for the purpose

---

[4] Bottom strongly connected components (BSCCs) correspond to maximal end components when the analysis concentrates on **DTMC**s, i.e., when there exists no nondeterminism. Those are strongly connected components which cannot be left once the system has entered them.

of this thesis).

[BK08] delivers a comprehensive introduction to the field of (probabilistic) model checking, which covers most of the concepts that are used in modern system verification. Some of these concepts are also reflected in the deliberations about the computation of quantiles, and therefore help to build the formal basis for the created toolset that will be presented throughout this thesis.

Multi-objective PMC We want to provide possibilities to relate the consumed energy of a system with its produced utility. In particular, we are interested in minimising the energy consumption by simultaneously keeping the achieved utility at a high level. So, we need to find solutions for multi-objective criteria, and in the field of probabilistic model checking there has already been some work concerning methods to provide a formal analysis that aims at satisfying multiple criteria at the same time. For instance, [CMH06] considers the computation of multiple discounted reward objectives for **MDP**s, and it is shown that randomised memoryless strategies are adequate for obtaining any possible value vector for the given objectives. The authors also show that it is decidable in polynomial time if a given value vector can be achieved by some strategy, and reduce the multi-objective optimisation and achievability problems for **MDP**s with discounted rewards to multi-objective linear programming. [Ete+07] demonstrates how a scheduler can be computed that satisfies multiple linear-time properties simultaneously for a given **MDP**. The authors present a polynomial-time algorithm that uses linear programming-methods for its computation. This allows us to, e.g., maximise the sum of several different reachability probabilities. The paper also solves qualitative multi-objective model checking problems by applying methods that solely make use of graph-theoretic methods. [For+11b] presents a combination of the multi-objective reachability optimisation with reward-expectations by again utilising techniques based on linear programming. The paper also presents computation results and statistics of an implementation of the presented theory. This extension to reward structures comes in very useful since reward structures are a natural way for annotating the model under consideration with its energy consumption and the utility-gain. [Bas+15] aims at computing strategies such that the long-run averages of multiple reward values are almost surely above a given multi-dimensional threshold vector. The authors as well show the applicability of their approach by presenting an analysis of the electric power distribution of an aircraft using the proposed methods. [Bai+14b] presents among others the computation of the quotient of two accumulated values. This allows to reason about the ratio between the consumed energy and the provided utility of a system. [Bai+14d] aims at finding a scheduler that maximises the probability for some temporal objective when a specific ω-regular condition is specified. The paper presents a transformation-based approach for LTL conditions that allows one to handle the computation of conditional probabilities like ordinary probabilities in the transformed model. The paper as well presents results for an implementation of the considered approach. [Mär+17] refines the implementation presented in [Bai+14d] for the computation of conditional probabilities in **DTMC**s and **MDP**s, when the objective and the condition are LTL formulas. The performance of the refined implementation is compared to the performance of the probabilistic model checker Storm [Deh+17] which is equipped with implementations for the computation of conditional probabilities as well. [Bai+17a] discusses the computation of maximal conditional expectations in finite-state **MDP**s when a reachability constraint is given as condition. This extends the results of [Bai+14d] since the approach presented there is not applicable for the calculation of maximal conditional expectations.

Quantiles and PMC In order to minimise the consumed energy and maximise the gained utility as well, we need to be able to optimise the accumulated reward over the considered model. Therefore, the concept of quantiles is an interesting formalism which becomes the focus of attention in order to help answering the emerging questions. Quantiles[5] were already well-known in the field of mathematical statistics (see [Ser80] for a comprehensive overview on quantiles in the field of mathematical statistics). For a real-valued random variable X a quantile corresponds to the minimal x such that the probability of the event $X > x$ exceeds a specified probability threshold $p \in [0, 1)$. Therefore, quantiles allow us to express, e.g., the minimal energy consumption necessary to guarantee the successful termination of multiple computation tasks with an acceptable high probability. This enables us to optimise the accumulation of the energy of an energy-aware system and therefore offers useful insights into the interplay of the consumed energy and the provided utility of the system, hence supporting the realisation of the envisioned goal for HAEC. Since quantiles are applicable as general-purpose methods a framework based on quantiles (as presented in this monograph) is not just restricted to the demands of energy and utility. Instead, quantiles can be used for various areas of application, e.g., for reasoning about minimal timings or the minimal number of communication operations.

Quantiles recently became a very active research topic in the field of probabilistic model checking. So, we want to outline important contributions in this field. In [Bai+12b] quantiles were considered in the field of probabilistic model checking for the very first time in a direct manner using a self-contained formalism[6]. We used PMC-techniques in order to do a formal analysis of a test-and-test-and-set spinlock protocol and combined this analysis with a measure-based simulation. One of the investigated properties asks for a quantile stating the minimal time that some process has to wait in order to access its critical computation section with a probability of at least 95%. We used long-run quantiles for discrete Markovian models in order to provide analysis results, and were able to align the results of our formal analysis with the measure-based simulation. Based on the experiences made in [Bai+12b] and the resultant need for a formal analysis of quantiles the group of Christel Baier laid the theoretical foundations

---

[5] Quantiles even build the basic blocks for a sorting algorithm named *Flashsort* [Neu97].

[6] The PRISM benchmark suite [KNP12] considers series of reachability probabilities for several protocols which can be analysed using, for instance, PRISM's experimental functionality (see `http://www.prismmodelchecker.org/manual/RunningPRISM/Experiments`, retrieved 28th March 2018) in order to analyse multiple model-checking runs. One is therefore able to extract the information on quantiles in an indirect manner by studying the plotted results.

for the computation of quantiles for Markovian models in [UB13] and [Bai+14a]. [UB13] presents algorithms for the computation of qualitative and quantitative quantile queries for upper-reward bounded until properties over discrete-time Markovian models. A graph-based polynomial-time algorithm for the computation of qualitative quantiles[7] is presented in this work. For the computation of quantitative quantiles[8] [UB13] presents an exponential-time algorithm which relies on solving a linear program. The theory was then extended by [Bai+14a] to lower-reward bounded reachability-quantiles and upper-reward bounded expectation-quantiles. Here, the presented algorithms rely on similar principles as already presented in [UB13]. [Bai+14a] also presents an implementation for the computation of quantiles in Markovian models and which is based on an approach called *back-propagation* (or *BP* for short). [Cie+08] presents several approaches in order to improve the performance of the quantitative analysis of Markov decision processes. One of the presented approaches is the blockwise *value iteration*, which performs a topological sorting of the strongly connected components in the model under consideration. The *BP*-approach presented in [Bai+14a] (and as well in this monograph) uses similar ideas as for the blockwise *value iteration* in order to improve its computational performance.

In [LS05], the authors do complexity-theoretic investigations which have a direct influence on the computation of quantiles. It is shown that the computation of (reward-)bounded probabilistic reachability formulae is **NP**-hard even for **DTMC**s. The authors reduce the K-th largest subset problem (see [GJ79, page 225]) to the problem of checking a bounded probabilistic reachability formula. This result is directly applicable to the computation of quantiles, since we need to check multiple bounded probabilistic reachability formulae in order to properly compute quantiles. It can be therefore stated that the computation of quantiles is **NP**-hard as well. In [HK15] the computation of accumulated costs over cost chains and cost processes is investigated. It is shown that it is possible to decide for an arbitrary cost process within **ExpTime** whether a specific (multi-dimensional) cost can be accumulated with a desired probability when a designated goal t should be reached (a so-called *cost problem*). It is as well demonstrated that a cost problem for an *atomic cost formula*[9] considered over acyclic cost processes is already **PSPACE**-complete, and therefore **PSPACE**-hardness for atomic cost problems over arbitrary cost processes can be derived in a direct manner. Since the computation of quantiles is directly related to cost problems for atomic cost formulas[10] the lower bound for the computation of quantiles received by the considerations of [LS05] becomes even tighter. [HK15] shows as well that a

---

[7] Qualitative quantiles are quantiles where the specified probability threshold is either 0 or 1.

[8] The probability threshold of quantitative quantiles is strictly greater 0 and smaller 1.

[9] An atomic cost formula constitutes an upper bound for the accumulated value of a cost in only one dimension, i.e., the related cost problem considers the accumulation of the cost with respect to this dimension only.

[10] Given cost process C with inital state s and target t, the existential quantile $\mathrm{qu}_s\big(\exists\mathbb{P}_{\geqslant p}(\Diamond^{\leqslant ?}\{t\})\big)$ (see Section 3.1) corresponds to the value $B \in \mathbb{N}$ if and only if there exists a scheduler $\mathfrak{S}$ over C such that the cost problems $P_{\mathfrak{S}}(K_C \leqslant B) \geqslant p$ and $P_{\mathfrak{S}}(K_C \leqslant B-1) < p$ hold. The monotonicity of the accumulated (non-negative) costs along the paths (see Figure 3.1 in Section 3.1) immediately implies $P_{\mathfrak{S}}(K_C \leqslant b) < p$ for each $b \in \{0, \ldots, B-2\}$ in this case.

cost problem can be decided in **PSPACE** for a cost chain. The authors furthermore examine which circumstances allow to relax those computational complexities. For the application of cost chains [HKL17] shows that the cost problems of [HK15] can be computed within the counting hierarchy (see [AW90]), and therefore it is believed that the computation can be done better than in **PSPACE**. The authors present an implementation that decides finitary cost problems based on Parikh-Images, and do a performance-comparison between their implemented methods and the probabilistic model checker PRISM (see [KNP11]). Inspired by the work of [UB13] and [Bai+14a] the authors of [RRS15] present theoretical deliberations on percentiles, which extend quantiles to multiple dimensions and combine it with several payoff functions.

Since so far we only considered models with discrete time steps, [Bai+14b] presents an algorithm for the computation of quantiles in continuous-time Markovian models. This algorithm is based on an exponential search with a subsequent binary search[11], and it was envisioned by the maintainers of the model checker MARCIE (see [HRS13]) to integrate an adapted form of this algorithm into their tool in order to support the computation of quantiles for stochastic Petri nets. As part of this monograph, the algorithm for **CTMC**s has been also implemented in the probabilistic model checker PRISM (see Section 3.7 and Section 5.2.2). [BDK14] presents a computation scheme for the analysis of quantiles under conjunctive constraints. When considering constraints that should hold for all schedulers (universal constraints), the idea is to compute each conjunct separately and afterwards do an intersection of the resulting sets. The minimal value in this intersection is then the result for the conjunctive constraint. The computation of conjunctive constraints for the best scheduler (existential constraints) uses techniques for solving standard multi-objective queries suggested in [Ete+07] and [For+11b].

Keep in mind that the usage of positive rational reward functions can be addressed by the same quantile algorithms as for reward functions involving natural numbers only. The idea is to scale each value of the given reward function properly such that the resulting denominators will correspond to the least common multiple of all involved denominators. These denominators will be then ignored and only the resulting numerators build the reward values for the upcoming quantile computations. As a last point the obtained computation result then needs to be divided by the previously ignored least common denominator in order to receive the demanded rational quantile value. Another generalisation to the concept of quantiles, as investigated here, is the usage of weight functions[12] instead of reward functions. The usage of weight functions makes the computation of quantiles more difficult. An important indication is the fact that the usage of weights entails that the accumulated costs along the paths of the considered model are no longer monotonic, which builds an essential feature that

---

[11] Of course, the computation-scheme of an exponential search followed by a binary search is also applicable for the naïve computation of quantiles in discrete-time models, like **DTMC**s or **MDP**s.

[12] Weight functions permit negative annotations to the states or transitions of a Markovian model, and it is therefore possible to formalise, e.g., the life-span of a battery-powered system which undergoes multiple cycles where the power supply will be discharged and recharged numerous times.

is used throughout the computations as presented in this thesis. On the other hand there is a related result in [Brá+10] which has a direct impact on the computation of quantiles over weight functions. As [Brá+10] states in its Introduction and as well in Section 4 the termination (i.e., reachability) probabilities for One-Counter Markov decision processes[13] can represent an irrational number in general. This fact makes a computational analysis for quantiles over weight functions very challenging. The mentioned aspects give an indication that it is not expected that it is possible to provide an efficient analysis of quantiles over models equipped with weight functions. The paper [Krä+15] addresses this issue in its Conclusions by citing [EY09; Brá+10; Bai+14b] and stating that the corresponding $\Box$-weight decision problem in unit-weight Markov chains is known to be **PosSLP**-hard. It is hence reasonable to restrict the discussed quantile framework to reward functions over natural numbers only, as it is done here. Nevertheless, the authors of [Krä+15] investigate this generalisation to weight functions and restrict their studies to the computation of ratio- and weight-quantiles for the qualitative case only, i.e., where the specified probability threshold is either 0 or 1. They present polynomial-time algorithms for the computation of those quantiles in Markov chains. The related problem concerning the computation of minimal expected weights until reaching a specific target in an integer-weighted **MDP** has been analysed in [Bai+18]. The authors present polynomial-time algorithms for computing the minimal expected weight until reaching the target almost surely (i.e., with probability 1) when considering all schedulers, and with positive probability (i.e., probability greater 0) in the best case. The problem of computing the expected weight for almost sure reachability in the best case, and the reachabiliy with positive probability over all schedulers are shown to be in $\mathbf{NP} \cap \mathbf{coNP}$. The algorithms presented in [Bai+18] take advantage of a classification of the end components of the analysed model. This classification will be done with respect to the weights that occur within the respective end components.

A very serious problem in the field of (probabilistic) model checking is the well-known state-explosion problem (see [Bur+90; McM93; BK08]). The problem arises from the fact that the number of reachable states grows in an exponential fashion with the number of variables that occur in the formal description of the system. One possibility to tackle this difficulty is to symbolically represent the state space of the model under consideration by means of binary decision diagrams (BDDs) [Lee59]. A comprehensive scientific investigation on how probabilistic model checking can be performed efficiently when relying on BDD-based techniques can be found in [Par02]. There, the important concepts are described which are needed in order to successfully provide a usable analysis that is based on symbolic representations of the model under consideration. The practical feasibility of such symbolic methods is demonstrated by a variety of analysed protocols. [Kle+16] and [Kle+17] sketch how to effectively perform symbolic computations to analyse quantiles with reachability constraints. In those papers we

---

[13] One-Counter Markov decision processes can be regarded as **MDP**s (like we use them here) which are equipped with a weight function that assigns either the weight $-1, 0$ or $+1$ (in other words unit weights) to each transition of the model.

present results of the computations and at the same time report on improvements that could be achieved by using reordering techniques to reduce the size of the utilised BDDs. Since smaller BDD-sizes normally have a positive impact on the computational effort caused by dealing with those BDDs the reordering allows to directly influence the performance of symbolic quantile-calculations.

Another method to avoid the state-explosion problem using a different approach is the utilisation of *statistical model checking* [YS02; LP02], or SMC for short. Instead of using numerical computations respecting the whole state space of the model, SMC is based on observing multiple executions of the model in a simulation-based fashion. Its results will be then obtained from statistical evidence and show if the desired behaviour will be satisfied or violated by the system. This has the advantage that the whole verification procedure does need far less memory since it needs to store only a small fragment of the information provided by the model. This also improves the time that is needed in order to perform the analysis. Hence, the scaling to large system instances is not such an issue as it is for classical (probabilistic) model checking. Therefore, SMC can also be applied to a larger class of systems. Of course, there also exist significant disadvantages when using SMC. One major disadvantage is that the outcome of the procedure does not give a sharp guarantee as it is the case for classical (probabilistic) model checking. The computed result instead corresponds to a statistical claim. In order to provide a suitable correctness of the result the sample sizes used by the analysis tend to grow very large, thus also increasing the computation time of the analysis. As the considered simulation runs require to be finite, SMC seems to only handle problems that can be decided on finite executions of the system, like e.g., bounded reachability properties. But, there has been some work in order to overcome this limitation, see for example [SVA05] or [RP09]. And this in fact allows to analyse unbounded until-properties with the methods of statistical model checking. Also, there are issues when using SMC for models comprising nondeterminism, like **MDP**s. The problem here is that it is not clear how to resolve the nondeterminism for the sampling procedure in a proper way. Nevertheless, there has been some work starting to address this issue. For example, [Hen+12] introduces methods for analysing bounded LTL over **MDP**s by using statistical model checking.

The algorithms for the computation of quantiles presented in [UB13] and [Bai+14a] (and as well used in this monograph) do rely on solving numerous linear programs. So, one can construct and solve a linear programming problem for calculating a solution, but it turns out that this approach does not scale to large model sizes [For+11a] that normally occur when applying probabilistic model checking. Therefore, another neat and relatively fast approach to calculate the solution of a linear program, which is proposed in the literature (see, e.g., [BK08, page 854]), is the so-called *value iteration*. In [HM14] the authors investigated the correctness of the convergence of the *value iteration* for reachability properties and show that there might occur correctness issues when utilising *value iteration* for the case of **MDP**s due to insufficient termination criteria. The authors propose a method called *interval iteration* in order to address these issues, where the correct result is narrowed from below and from above at the same time. This method is also considered for the implementations presented in this monograph. Since [HM14] considers the problems of the *value iteration* for reachability properties, and since we will also treat quantiles over expectation constraints, the same problems can arise for the computation of expected values. Therefore, [Bai+17b] reformulates the *interval iteration*-approach for the computation of expected accumulated weights in **MDP**s, and the results can be carried over in order to support *interval iteration* for the calculation of expectation-quantiles. The paper also presents a topological *interval iteration* which treats the strongly connected components of the model under consideration separately[14] in order to improve the performance of the computations. The paper [HH16] is related since it presents three different techniques for the computation of time- and reward-bounded properties in **MDP**s. The first approach is a reformulation of the *back-propagation* approach originally published in [Bai+14a]. The other two presented methods are *scheduler enumeration* and *state elimination*. Both approaches are based on the fact that the reward structures for the model under consideration do only supply rewards of either 0 or 1. For reward structures providing higher reward values $r > 1$, the authors propose to emulate the rewards by chains of r additional states augmented with reward 1 each.

Model-checking tools Since the aim of this monograph is to provide machinery for doing an analysis of the energy efficiency of a system, we want to list some of the well-established tools available in the field of probabilistic model checking: A very prominent probabilistic model checker supporting a variety of probabilistic models (**CTMC**s, **DTMC**s, **MDP**s, probabilistic timed automata) and property specifications is PRISM[15] [KNP11]. PRISM uses techniques based on (multi-terminal) binary decision diagrams in order to symbolically represent the state space of the model under consideration, as well as providing an engine using an explicit representation of the state space. PRISM also provides a simulation engine which allows one to do statistical model checking for the models that were created using PRISM's guarded command language. In this way PRISM is able to provide classical (probabilistic) model checking and statistical model checking for the same input. Many state-of-the-art model-checking techniques are integrated into PRISM for the purpose of delivering tool-support for a wide range of application areas. In order to take advantage of the infrastructure supported by PRISM, the quantile-framework presented in this monograph has been integrated into PRISM. Another relatively young probabilistic model checker is Storm[16] [Deh+17], which has its focus on the analysis of discrete-time Markov chains and Markov decision processes, and the continuous-time variants thereof. Storm focusses on the analysis of PCTL [HJ94] and CSL [Bai+03], and it also delivers support for the computation of conditional probabilities and expectations as presented in [Bai+14d]. It was build with an emphasis on a modular concept such that the functionality can be extended very easily, and this also allows one to exchange different components with

---

[14] A similar approach was presented in [Bai+14a] for improving the computation of the zero-reward fragments of the model. This approach has been implemented for the framework presented in this monograph as well (see Section 5.1.3).

[15] `http://www.prismmodelchecker.org/`, retrieved 28th March 2018

[16] `http://www.stormchecker.org/`, retrieved 28th March 2018

other alternatives in order to adapt the solution algorithms to varying requirements. Storm even supports PRISM's (guarded command) input language in order to provide its functionality for models already developed in the context of PRISM. Up to now, Storm does not support the analysis of LTL formulas and statistical model checking. MRMC[17] [Kat+09] is a command-line tool supporting the verification of PCTL or CSL for (discrete-time and continuous-time) Markov chains equipped with reward structures, and continuous-time Markov decision processes as well. MRMC additionally supports statistical model checking for **CTMC**s. The Modest Toolset[18] [HH14] is an agglomeration of different tools supporting a variety of widely used model types, ranging from Markov decision processes, over (discrete- or continuous-time) Markov chains to (stochastic) timed and hybrid automata. Modest is built in order to support the verification of stochastic hybrid automata [Hah+13], and this enables Modest to actually cover a variety of formalisms. Therefore, Modest delivers tool-support for many practically relevant concepts, and it also provides a powerful high-level compositional modelling language. IscasMC[19] [Hah+14] is a model checker which follows a different approach as the previously presented tools. It is a web-based model checker which uses a client-server architecture where the client provides the input and delegates the computation tasks to the server. This approach allows to provide the abilities of model checking even for not so powerful computing devices like mobile phones, the only requirement is a connection to a server doing the compute- and memory-intense working tasks. IscasMC analyses Markov chains and Markov decision processes against PCTL and PCTL*, and also provides support for models written in PRISM's input language. ProbDiVinE-MC[20] [Bar+08] is a probabilistic model checker integrated into the verification tool DiVinE [Bar+06]. Its focus is on checking qualitative and quantitative LTL specifications over Markov decision processes, and as a distinctive feature this model checker supplies parallel and distributed verification algorithms. The parallel computations rely on a decomposition of the model under consideration into its strongly connected components. Those components will be then solved independently of each other whenever it is possible. A similar idea builds the basis for the parallel quantile computations of the zero-reward fragment of a given model (see Section 5.1.4 for further details), and therefore provide a possibility to improve the performance of the analysis methods presented in this monograph.

The following selection shows tools that have their emphasis on "classical" model checking, so they do not consider the handling of probabilistic behaviour: SMV[21] [Bur+90] is a tool for checking finite state systems against specifications formalised in CTL. It is a classical model checker without support for probabilistic behaviour, and was the very first model checker to use symbolic methods based on BDDs. Its main focus is on providing an application of symbolic model checking for the field of

---

[17] `http://www.mrmc-tool.org/trac/wiki/WikiStart`, retrieved 28th March 2018

[18] `http://www.modestchecker.net/`, retrieved 28th March 2018

[19] `http://iscasmc.ios.ac.cn/IscasMC`, retrieved 28th March 2018

[20] `http://divine.fi.muni.cz/darcs/branch-3.0/gui/help/divine/probdivine.html`, retrieved 28th September 2018

[21] `http://www.cs.cmu.edu/~modelcheck/smv.html`, retrieved 28th March 2018

hardware verification. NuSMV[22] [Cim+02] is a reimplementation of SMV and also extends the technology by integrating techniques based on propositional satisfiability (SAT) [Bie+99] in order to allow a tool-set for certain scenarios which were not realisable by BDD-based techniques. This allows for tool-support for the verification of a broader range of complex scenarios. Vereofy[23] [BKK; Bai+09] is a model checker which was developed within the group of Christel Baier, and is tailored for the verification of component-based systems. The tool uses the Reo coordination language [Arb04] in order to formalise the communication of the participating components, and it supports branching-time, alternating-time and linear-time model checking. It also delivers a framework for equivalence checking based on bisimulation. UPPAAL[24] [LPY97] is a tool which supports the verification of real-time systems. The system under consideration is hereby described using a graphical representation of the system's dynamic behaviour. For doing the analysis UPPAAL relies on timed automata augmented with real-valued clocks, allowing to provide a feasible analysis for systems where timing aspects are critical. There exist a number of extensions to the UPPAAL-platform that extend the capabilities of the model checker and therefore allow to apply the methods of UPPAAL to a broad variety of application areas. For example, the extension SMC[25] applies statistical model checking for supporting the analysis of stochastic timed systems.

1.2 Contribution and outline

The intention of this monograph is to provide a formal framework for doing a multi-objective analysis of the energy efficiency for a variety of energy-aware systems. Therefore, several interesting measures were identified that rely on quantiles and allow to carry out practically relevant results. The presented quantile-based approach is not restricted to just the needs of a trade-off analysis between the consumed energy and the provided utility of a system. Instead, it is designed to allow a general-purpose trade-off analysis for a variety of relevant characteristics. In order to make the framework available for practical applications we are as well interested in an implementation allowing to perform the desired analysis efficiently.

The main contributions worked out in this monograph in order to contribute to the given task are as follows:

Quantiles for (constrained) reachability properties: We present the computation of quantiles over Markovian models, which allow us to analyse the minimal accumulated energy that needs to be consumed in order to reach a specific goal. Therefore, we will consider the computation of upper-reward bounded reachability quantiles. We also want to analyse the maximal accumulated utility that can be provided using a specific energy budget. The described approaches allow to

---

[22] `http://nusmv.fbk.eu/`, retrieved 28th March 2018

[23] `http://www.vereofy.de/`, retrieved 28th March 2018

[24] `http://uppaal.org/`, retrieved 28th March 2018

[25] `http://people.cs.aau.dk/~adavid/smc/`, retrieved 28th September 2018

optimise the accumulation of a specific reward function while at the same time a (constrained) reachability analysis will be performed. Its core concept is based on an iterated computation of multiple (reward-)bounded reachability probabilities.

Based on the investigations done in [UB13] (which considers reachability quantiles where the accumulation of the reward is bounded from above) the computation of reachability quantiles as presented in this thesis has been published previously in [Bai+14a].

Quantiles for (constrained) reachability properties under side conditions: We are as well interested in the analysis of reachability quantiles when some ω-regular side condition should be respected. This allows for the computation of quantiles when at the same time certain important objectives should be preserved for the system under consideration, e.g., whenever there occurs some critical failure there should happen some recovery mechanisms. The utilisation of quantiles under side conditions allows to reason about such objectives.

Expectation quantiles: In order to relate the accumulation of the consumed energy of the analysed system with the accumulation of its gained utility, we present methods that allow the computation of expectation quantiles over Markov decision processes (and discrete-time Markov chains as well). Those quantiles allow to reason about the minimal energy budget that needs to be invested in order to gain a desired utility-expectation. In contrast to the previously introduced reachability quantiles we do not need to fix the accumulation of one of the two involved values. Instead, we can directly relate the accumulation of both values within one evaluation.

The computation of quantiles over expectation objectives has already been presented in [Bai+14a].

Implementation of quantile-algorithms and integration into PRISM: Since one aim of this thesis is to provide a practical toolbox that allows one to realise a quantile-based analysis, there exists an implementation of the presented quantile computations that is integrated into the probabilistic model checker PRISM. The implementation considers several possibilities for improving the performance of the utilised computations. Those optimisations make use of the specific structural characteristics that are inherent for the linear programs that need to be solved when computing quantile queries.

Parts of the implementation presented and used in this monograph have already been introduced in [Bai+14a], [Kle+16] and [Kle+17].

Quantile-based analysis of energy-aware protocols: The presented implementations are used for analysing the energy efficiency of several protocols. Therefore, we report on the performance of the presented implementations by first analysing already existing case studies from the PRISM benchmark suite (see [KNP12]) in order to demonstrate the efficiency of the presented quantile-based framework.

> We will also see how the framework can be utilised for analysing protocols that have an emphasis on their energy efficiency.
>
> In doing so we present an energy-aware job scheduling protocol already introduced in [Bai+14a], and an analysis of this protocol has been shown in the mentioned paper, and in [Kle+16] and [Kle+17] as well. Other protocols we consider are the eBond protocol (originally proposed in [Häh+13]) and the HAECubie demonstrator. Some of the results presented in this monograph for both protocols already passed a reviewing process as those results have been integrated into demonstrators shown to the reviewers of the *Deutsche Forschungsgemeinschaft*[26] as a small part of the (successful) defence of the first phase of HAEC.

The document is structured in the following way. Chapter 2 provides a short introduction into the necessary theoretical basis needed for the computation of quantiles. The computation of reachability quantiles over Markovian models is presented in Chapter 3. In this connection Section 3.3 describes the computation of upper-reward bounded reachability quantiles, and the computation of lower-reward bounded reachability quantiles will be handled in Section 3.4. Reachability quantiles under side conditions are treated in Section 3.6, and the computation of quantiles over **CTMC**s is considered in Section 3.7 as well. The mentioned expectation quantiles over **DTMC**s or **MDP**s are then subject of the analysis presented in Chapter 4. The integration of all the provided quantile approaches into the probabilistic model checker PRISM is described in Chapter 5, which starts by presenting methods for improving the computational performance of the quantile calculations in Section 5.1. We will also give some information on the computation of quantiles using symbolic (MT)BDD-based methods. The described implementation of the quantile algorithms will be then used in Chapter 6 in order to provide an analysis of several energy-aware protocols. The chapter starts by presenting results for protocols already known from PRISM's benchmark suite [KNP12] for demonstrating the potential of the provided implementation. Afterwards it is shown in Section 6.2 how the presented implementation can be used for the analysis of energy-aware systems. It is as well illustrated how the different implemented approaches of Chapter 5 perform for a variety of different situations, and how one can take advantage of the different optimisations presented in Section 5.1 in order to improve the performance of the analysis. Chapter 7 then concludes this thesis by giving a brief summary of the presented results and a classification of the analysis results obtained in Chapter 6.

---

[26] `http://www.dfg.de`, retrieved 28th March 2018

2 Preliminaries

Here, the relevant concepts will be provided which serve as the formal basis for the analysis of reward-bounded reachability properties and quantiles in Markovian models. As well, the specifications given as formulas in probabilistic computation tree logic with reward-bounded modalities (PRCTL) will be introduced briefly. Further details can be found in, e.g., [Put94; Alf98; BK08].

For all the upcoming considerations we make use of the natural numbers including zero, i.e., $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. The cardinality of a (countable) set A is denoted by $\mathsf{card}(A)$ and it characterises the number of the elements contained in A. The (reachable) states of a given model are denoted as S. The cartesian product of the states of a model and all natural numbers up to bound $r \in \mathbb{N}$ is denoted by $\mathsf{S}[r]$, i.e., $\mathsf{S}[r] = \mathsf{S} \times \{0, 1, \ldots, r\}$.

Distributions If X is a countable, nonempty set then a distribution on X is a function $\mu : X \to [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$. We write $\mathrm{Dist}(X)$ for the set of distributions on X. $\mathrm{supp}(\mu)$ denotes the support of μ, i.e., the set of elements $x \in X$ where $\mu(x)$ is positive.

Markov decision processes (MDPs) An **MDP** is a tuple $\mathcal{M} = (\mathsf{S}, \mathsf{Act}, P)$, where S is a finite set of states, Act a finite set of actions, $P : \mathsf{S} \times \mathsf{Act} \times \mathsf{S} \to [0, 1]$ such that $\sum_{s' \in \mathsf{S}} P(s, \alpha, s') \in \{0, 1\}$ for all states $s \in \mathsf{S}$ and actions $\alpha \in \mathsf{Act}$. The tuples $(s, \alpha, s') \in \mathsf{S} \times \mathsf{Act} \times \mathsf{S}$ with $P(s, \alpha, s') > 0$ are called *steps* and we then say that state s' is an α-successor of s. We write $\mathsf{Act}(s)$ for the set of enabled actions α, i.e., those that have an α-successor from state $s \in \mathsf{S}$ and require that $\mathsf{Act}(s) \neq \varnothing$ for all states s.

Intuitively, if the current state of $\mathcal{M}$ is s, then first there is a nondeterministic choice to select one of the enabled actions α. Then, $\mathcal{M}$ behaves probabilistically and moves with probability $P(s, \alpha, s')$ to some state s'.

The *size* of an **MDP** $\mathcal{M}$ is defined as the sum of the number of its reachable states (cardinality of S) and the number of its transitions, i.e., the number of the elements $(s, \alpha, s') \in \mathsf{S} \times \mathsf{Act} \times \mathsf{S}$ such that $P(s, \alpha, s') > 0$.

Paths in MDPs Paths in an **MDP** can be seen as sample runs where the nondeterminism has been resolved. Formally, paths are finite or infinite sequences

$$\pi = s_0 \, \alpha_0 \, s_1 \, \alpha_1 \, s_2 \, \alpha_2 \, \ldots \in (\mathsf{S} \times \mathsf{Act})^* \mathsf{S} \cup (\mathsf{S} \times \mathsf{Act})^\omega$$

that are built by consecutive steps, i.e., $\alpha_i \in \mathsf{Act}(s_i)$ and $P(s_i, \alpha_i, s_{i+1}) > 0$ for all i.

The very first state s_0 of the path π is denoted by $\mathit{first}(\pi)$. $\pi[k]$ denotes the $(k{+}1)$-th state in π and $\mathit{pref}(\pi, k)$ the prefix of π consisting of the first k steps, ending in state $\pi[k] = s_k$. If $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\dots\,\alpha_{n-1}\,s_n$ is finite, then $\mathit{last}(\pi)$ denotes the last state that is reached by π, i.e., $\mathit{last}(\pi) = s_n$. The probability of a finite path $\rho = s_0\,\alpha_0\,s_1\,\alpha_1\,\dots\,\alpha_{n-1}\,s_n$ is the product of the probabilities of its steps: $\mathrm{prob}(\rho) = \prod_{0 \leqslant i<n} P(s_i, \alpha_i, s_{i+1})$.

The set $\mathit{FinPaths}(s)$ denotes the set of finite paths and $\mathit{InfPaths}(s)$ denotes the set of infinite paths starting in state s.

Sub-MDPs, end components A *sub-**MDP*** of an **MDP** $\mathcal{M}$ is denoted by a pair $(T, \mathcal{A})$ where $T \subseteq \mathsf{S}$ and $\mathcal{A} : T \to 2^{\mathsf{Act}}$ such that for all $t \in T$:

1. $\mathcal{A}(t) \subseteq \mathsf{Act}(t)$ and

2. if $\alpha \in \mathcal{A}(t)$ and $P(t, \alpha, t') > 0$ then $t' \in T$.

An *end component* of $\mathcal{M}$ is a sub-**MDP** $(T, \mathcal{A})$ of $\mathcal{M}$ where $\mathcal{A}(t)$ is nonempty for all $t \in T$ and the underlying directed graph with node set T and the edge relation $t \to t'$ iff $P(t, \alpha, t') > 0$ for some $\alpha \in \mathcal{A}(t)$ is strongly connected. An end component is said to be *maximal* if it is not contained in any other end component.

Discrete-time Markov chains (DTMCs) A **DTMC** is a purely probabilistic instance of an **MDP**, i.e., where the action set is a singleton for each state. As a consequence the only enabled action that can be chosen in state s is defined by s unambiguously, and therefore there exists no nondeterminism in the model. The definition of P can be therefore simplified to $P : \mathsf{S} \times \mathsf{S} \to [0, 1]$ as well, and we require that $\sum_{s' \in \mathsf{S}} P(s, s') = 1$ for all states $s \in \mathsf{S}$.

Continuous-time Markov chains (CTMCs) A **CTMC** is a pair $(\mathcal{M}, E)$ where $\mathcal{M} = (\mathsf{S}, \mathsf{Act}, P)$ is a **DTMC** as before (it is called the *embedded **DTMC***), and $E : \mathsf{S} \to \mathbb{R}_{\geqslant 0}$ specifies exit-rates for the states of the model. $E(s)$ is the rate of an exponential distribution specifying the frequency of taking a transition from s. The probability to take some transitions from s within t time units is given by $1 - e^{-E(s) \cdot t}$ with e being the Eulerian number. The probability to take a transition from s to s' within t time units is:

$$P(s, [0, t], s') = P(s, s') \cdot \left(1 - e^{-E(s) \cdot t}\right)$$

As a consequence, $1/E(s)$ is the average sojourn time in state s. If in state s there exist multiple states s' with $P(s, s') > 0$, we say that there occurs a race condition. A *trajectory* is an alternating sequence $s_0\,t_0\,s_1\,t_1\,s_2\,t_2\dots$ of states s_i and nonnegative real numbers t_i (the corresponding sojourn times in the states) such that $P(s_i, s_{i+1}) > 0$.

Reward structure A reward structure R for the Markovian model $\mathcal{M}$ consists of finitely many reward functions $\mathsf{rew} : \mathsf{S} \cup (\mathsf{S} \times \mathsf{Act}) \to \mathbb{N}$. We call rew a *transition-reward function* (or *state-action reward function*) if $\mathsf{rew}(s) = 0$ for all $s \in \mathsf{S}$. rew is called a *state-reward function* if $\mathsf{rew}(s, \alpha) = 0$ for each pair $(s, \alpha) \in \mathsf{S} \times \mathsf{Act}$. For simplicity we will consider transition-reward functions from now on[1]. If we handle state-reward functions it is stated explicitly[2]. Throughout this document we will use the reward function $\mathsf{rew}_e : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ to reason about the energy consumption of the analysed model, and $\mathsf{rew}_u : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ formalises its utility.

If $\rho = s_0\, \alpha_0\, s_1\, \alpha_1\, \ldots\, \alpha_{n-1}\, s_n$ is a finite path, then we can declare the accumulated reward $\mathsf{rew}(\rho)$ of the path as the sum of the rewards for the state-action pairs, i.e., $\mathsf{rew}(\rho) = \sum_{0 \leqslant i < n} \mathsf{rew}(s_i, \alpha_i)$.

When reasoning about **CTMC**s, state rewards are understood as rewards per time spent in the corresponding state, and therefore need to be scaled with the corresponding sojourn times in trajectories during their accumulation.

Schedulers and induced probability space Reasoning about probabilities for path properties in **MDP**s requires the selection of an initial state and the resolution of the nondeterministic choices between the possible transitions. The latter is formalised via *schedulers*, often also called policies or adversaries, which take as input a finite path ρ and select an action to be executed. A (deterministic) scheduler is a function $\mathfrak{S} : \mathit{FinPaths} \to \mathsf{Act}$ such that $\mathfrak{S}(\rho) \in \mathsf{Act}\big(s_n\big)$ for all finite paths $\rho = s_0\, \alpha_0\, \ldots\, \alpha_{n-1}\, s_n$. An $\mathfrak{S}$-*path* is any path that arises when the nondeterministic choices in $\mathcal{M}$ are resolved using $\mathfrak{S}$, i.e., $\mathfrak{S}\big(\mathit{pref}(\rho, k)\big) = \alpha_k$ for all $0 \leqslant k < n$. Infinite $\mathfrak{S}$-paths are defined accordingly. Given some scheduler $\mathfrak{S}$ and state s (viewed as the initial state), the behaviour of $\mathcal{M}$ under $\mathfrak{S}$ is purely probabilistic and can be formalised by a tree-like (infinite-state) Markov chain $\mathcal{M}_s^{\mathfrak{S}}$. One can think of the states in $\mathcal{M}_s^{\mathfrak{S}}$ as finite $\mathfrak{S}$-paths $\rho = s_0\, \alpha_0\, \ldots\, \alpha_{n-1}\, s_n$ starting in state $s_0 = s$, where the probability to move from ρ to $\rho\, \alpha\, s'$ is simply $P(s_n, \alpha, s')$. Using standard concepts of measure and probability theory, a sigma-algebra and a probability measure $\mathsf{Pr}_s^{\mathfrak{S}}$ for measurable sets of the infinite paths in the Markov chain $\mathcal{M}_s^{\mathfrak{S}}$, also called *(path) events* or *path properties*, is defined and can be transferred to maximal $\mathfrak{S}$-paths in $\mathcal{M}$ starting in s.

For Markov chains, the concept of schedulers is irrelevant (as there exists no nondeterminism) and we simply write Pr_s for the probability measure induced by $\mathcal{M}$ when s is viewed as the initial state. For further details, it is recommended to study standard text books such as [Hav98; Kul95; Put94].

For a worst-case analysis of a system modelled by an **MDP** $\mathcal{M}$, one ranges over all initial states and all schedulers (i.e., all possible resolutions of the nondeterminism) and considers the minimal or maximal probabilities for φ. If φ represents a desired path property, then $\mathsf{Pr}_s^{\min}(\varphi) = \inf_{\mathfrak{S}} \mathsf{Pr}_s^{\mathfrak{S}}(\varphi)$ is the probability for $\mathcal{M}$ satisfying φ that

---

[1] An arbitrary state-reward function can be realised as a transition-reward function by setting $\mathsf{rew}(s, \alpha) = \mathsf{rew}(s)$ for each action $\alpha \in \mathsf{Act}(s)$.

[2] The later presented implementation is tailored such that it is able to handle arbitrary reward functions.

can be guaranteed even for worst-case scenarios, i.e., when ranging over all schedulers. Similarly, if φ stands for a bad (undesired) path property, then $\mathsf{Pr}^{\max}_s(\varphi) = \sup_{\mathfrak{S}} \mathsf{Pr}^{\mathfrak{S}}_s(\varphi)$ is the least upper bound that can be guaranteed for the bad behaviours. Vice versa, the value $\mathsf{Pr}^{\max}_s(\varphi)$ for a desired path property φ is relevant when addressing the task to synthesise a (control) mechanism for scheduling actions with the objective φ. In this context, it is of interest to compute a scheduler $\mathfrak{S}$ such that $\mathsf{Pr}^{\mathfrak{S}}_s(\varphi) = \mathsf{Pr}^{\max}_s(\varphi)$.

A scheduler $\mathfrak{S}$ is called *finite-memory* scheduler if there exists a finite set $\mathfrak{M}$ of modes, a decision function $\mathfrak{dec} : \mathsf{S} \times \mathfrak{M} \to \mathsf{Act}$, an initial mode function $\mathfrak{init} : \mathsf{S} \to \mathfrak{M}$, and a next-mode function $\mathfrak{next} : \mathfrak{M} \times \mathsf{S} \to \mathfrak{M}$ such that for each $\mathfrak{S}$-path ρ we have $\mathfrak{S}(\rho) = \mathfrak{dec}\big(\mathit{last}(\rho), \mathfrak{m}(\rho)\big)$. Here, $\mathfrak{m}(\rho)$ is defined inductively on the length of finite paths by $\mathfrak{m}(s_0) = \mathfrak{init}(s_0)$ and $\mathfrak{m}(\rho\,\alpha\,s) = \mathfrak{next}\big(\mathfrak{m}(\rho), s\big)$. The size of a finite-memory scheduler is the number of its modes. A finite-memory scheduler where $\mathfrak{M}$ is a singleton is called a *memoryless* scheduler.

Limits of paths It is well-known [Alf98] that for any scheduler $\mathfrak{S}$, the *limit* of almost all $\mathfrak{S}$-paths constitutes an end component. Here, the limit of an infinite path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots$ is given by the set $\inf(\pi)$ of states which appear infinitely often in π and the function that assigns to each state $t \in \inf(\pi)$ the set of actions α with $(s_i, \alpha_i) = (t, \alpha)$ for infinitely many i.

State and path properties Let s be a state, $\bowtie \in \{<, \leqslant, \geqslant, >\}$ a relation operator, $p \in [0,1]$ a probability bound and φ a path property. We write $s \models \exists\mathbb{P}_{\bowtie p}(\varphi)$ if there exists a scheduler $\mathfrak{S}$ with $\mathsf{Pr}^{\mathfrak{S}}_s(\varphi) \bowtie p$. Similarly, $s \models \forall\mathbb{P}_{\bowtie p}(\varphi)$ if $\mathsf{Pr}^{\mathfrak{S}}_s(\varphi) \bowtie p$ for all schedulers $\mathfrak{S}$. For the extreme cases where $p = 1$ is used, $\exists\mathbb{P}_{=1}(\varphi)$, $\exists\mathbb{P}_{<1}(\varphi)$, $\forall\mathbb{P}_{=1}(\varphi)$ and $\forall\mathbb{P}_{<1}(\varphi)$ are called *almost-sure* properties (where the probability bound "$\geqslant 1$" has been replaced with "$= 1$"). These, as well as the properties with dual probability bounds "$= 0$" or "> 0", are called *qualitative.*

Given a model and a reward structure R over this model consisting of reward function rew, sets A, $B \subseteq \mathsf{S}$, and $r \in \mathbb{N}$, then $A\,\mathcal{U}\,\big((\mathsf{rew} \bowtie r) \wedge B\big)$ stands for the set of infinite paths π such that there is some $k \in \mathbb{N}$ with $\mathsf{rew}(\mathit{pref}(\pi, k)) \bowtie r$ and $\pi[k] \in B$, $\pi[i] \in A$ for all $0 \leqslant i < k$. If rew is clear from the context (e.g., if the reward structure R is a singleton set), we use $A\,\mathcal{U}^{\bowtie r}\,B$ to abbreviate $A\,\mathcal{U}\,\big((\mathsf{rew} \bowtie r) \wedge B\big)$. Intuitively, $A\,\mathcal{U}^{\bowtie r}\,B$ denotes all paths such that some state from B will be reached eventually, and along the way only states specified by A are allowed. The reward that is accumulated before finally reaching B is bounded by r with respect to $\bowtie$. We often use the notation $\pi \models A\,\mathcal{U}^{\bowtie r}\,B$ instead of $\pi \in A\,\mathcal{U}^{\bowtie r}\,B$. As usual, we derive the release operator $\mathcal{R}$ by $A\,\mathcal{R}^{\bowtie r}\,B = \neg(\neg A\,\mathcal{U}^{\bowtie r}\,\neg B)$, where $\neg B$ denotes the complement of B ($= \mathsf{S} \backslash B$). The temporal modalities $\Diamond$ (eventually) and $\Box$ (always) with or without reward-bounds are derived as usual, e.g., $\Diamond^{\bowtie r} B = \mathsf{true}\,\mathcal{U}^{\bowtie r}\,B$ and $\Box^{\bowtie r} B = \neg\Diamond^{\bowtie r}\neg B$, where true stands for the full state space S.

Reward-bounded path properties such as $\varphi[r] = A\,\mathcal{U}^{\leqslant r}\,B$ are called *increasing* as $\pi \models \varphi[r]$ implies $\pi \models \varphi[r{+}1]$. The dual path properties $\psi[r] = \neg\varphi[r]$ are called *decreasing* as $\pi \models \psi[r{+}1]$ implies $\pi \models \psi[r]$. Analogously, a state property $\Phi[r]$ is called increasing if $s \models \Phi[r]$ implies $s \models \Phi[r{+}1]$. Examples for increasing state properties are $\exists\mathbb{P}_{>p}(\varphi[r])$, $\forall\mathbb{P}_{>p}(\varphi[r])$, $\exists\mathbb{P}_{<p}(\psi[r])$ and $\forall\mathbb{P}_{<p}(\psi[r])$. Decreasing state properties are defined accordingly.

The following simple observation will be used in the upcoming considerations, and is therefore stated here:

Proposition 2.0.1. *For a path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots$, arbitrary sets $A, B \subseteq \mathsf{S}$, and $r \in \mathbb{N}$ the following holds:*

$$\pi \models A\,\mathcal{U}^{\leqslant r}\,B \qquad \textit{iff} \qquad \pi \models (A \cup B)\,\mathcal{U}^{\leqslant r}\,B$$

Proof. The implication from left to right is a simple consequence from the fact that $A \subseteq A \cup B$ for arbitrary sets A and B.

So, we focus on the implication from right to left, and therefore assume that $\pi \models (A \cup B)\,\mathcal{U}^{\leqslant r}\,B$. This means that there exists an index k such that $s_k \in B$, for all $m < k$ holds $s_m \in A \cup B$, and $\mathsf{rew}(s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\,\alpha_{k-1}\,s_k) \leqslant r$. Given k is the smallest index with $s_k \in B$, then for each index $m < k$ it must hold $s_m \in A$, and this would imply $\pi \models A\,\mathcal{U}^{\leqslant r}\,B$ immediately. If on the other hand k is not the smallest index, we can find the smallest index $l < k$ satisfying the needs. As a consequence we have $s_l \in B$, $s_m \in A$ holds for all $m < l$, and $\mathsf{rew}(s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\,\alpha_{l-1}\,s_l) \leqslant \mathsf{rew}(s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\,\alpha_{k-1}\,s_k) \leqslant r$. Here, we use the fact that we do not consider negative rewards. $\square$

3 Reward-bounded reachability properties and quantiles

Since the ultimate goal of the framework presented here is to reduce the expenses of the system under consideration to a minimum while simultaneously keeping the systems performance at a high level, we need to investigate the interplay of both measures. Therefore, it is necessary to relate the accumulated costs to the outcome that can be achieved by the successful operation of the system. Transferred to the setting of HAEC this means that we want to link the energy consumption of the system to the performance that can be achieved by the operation of the analysed system.

In order to provide a formal energy-aware analysis for the system and its components it is therefore crucial to pay attention to the described relation between the consumed energy and the gained utility. So, it is required to do a multi-objective analysis in order to get insights into the energy efficiency of the system. Therefore, we will transfer the well-known concept of quantiles into the field of probabilistic model checking in order to provide a tailored framework for the formal analysis of energy-aware systems. The main idea behind quantiles is the optimisation of the accumulation of a specific reward along the paths in the given model, while at the same time a reachability analysis will be performed. So, when considering an increasing path property one aims at minimising the accumulated reward, whereas a maximisation of the reward is considered for a decreasing path property.

The algorithms that will be demonstrated in the following build the basis for an implementation presented in Chapter 5, and which will be used in order to realise the examination of the protocols shown in Chapter 6.

3.1 Essentials

We start by introducing the essential concepts for the computation of quantiles over discrete-time Markovian models (like **DTMC**s or **MDP**s), and will later provide algorithms for their computation. We will further see that an iterated computation of bounded (constrained) reachability properties builds the basis for the presented algorithms.

We will now specify the questioned quantiles formally. Let $\mathcal{M} = (\mathsf{S}, \mathsf{Act}, P)$ be an **MDP** and $\mathsf{rew} : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ a distinguished reward function in its reward structure. Given an increasing path property $\varphi[r]$ where parameter $r \in \mathbb{N}$ stands for some bound on the accumulation of the reward function rew, we define the following types of *existential quantiles* for state $s \in \mathsf{S}$, where $\psi[r] = \neg\varphi[r]$, $\trianglerighteq \in \{\geqslant, >\}$ and $p \in [0,1] \cap \mathbb{Q}$:

$$\begin{aligned}
\mathrm{qu}_s\big(\exists\mathbb{P}_{\trianglerighteq p}(\varphi[?])\big) &= \min\big\{r \in \mathbb{N} \,:\, s \models \exists\mathbb{P}_{\trianglerighteq p}\big(\varphi[r]\big)\big\} \\
&= \min\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\max}_s\big(\varphi[r]\big) \trianglerighteq p\big\} \\
\mathrm{qu}_s\big(\exists\mathbb{P}_{\trianglerighteq p}(\psi[?])\big) &= \max\big\{r \in \mathbb{N} \,:\, s \models \exists\mathbb{P}_{\trianglerighteq p}\big(\psi[r]\big)\big\} \\
&= \max\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\max}_s\big(\psi[r]\big) \trianglerighteq p\big\}
\end{aligned}$$

Similarly, we can define the corresponding types of *universal quantiles* which are considering $\mathsf{Pr}^{\min}$ instead of $\mathsf{Pr}^{\max}$:

$$\begin{aligned}
\mathrm{qu}_s\big(\forall\mathbb{P}_{\trianglerighteq p}(\varphi[?])\big) &= \min\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\min}_s\big(\varphi[r]\big) \trianglerighteq p\big\} \\
\mathrm{qu}_s\big(\forall\mathbb{P}_{\trianglerighteq p}(\psi[?])\big) &= \max\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\min}_s\big(\psi[r]\big) \trianglerighteq p\big\}
\end{aligned}$$

When considering **DTMC**s there is no nondeterminism involved. Therefore, an existential quantile coincides with an universal quantile and we can thus simply drop this distinction:

$$\begin{aligned}
\mathrm{qu}_s\big(\mathbb{P}_{\trianglerighteq p}(\varphi[?])\big) &= \min\big\{r \in \mathbb{N} \,:\, s \models \mathbb{P}_{\trianglerighteq p}\big(\varphi[r]\big)\big\} \\
&= \min\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s\big(\varphi[r]\big) \trianglerighteq p\big\} \\
\mathrm{qu}_s\big(\mathbb{P}_{\trianglerighteq p}(\psi[?])\big) &= \max\big\{r \in \mathbb{N} \,:\, s \models \mathbb{P}_{\trianglerighteq p}\big(\psi[r]\big)\big\} \\
&= \max\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s\big(\psi[r]\big) \trianglerighteq p\big\}
\end{aligned}$$

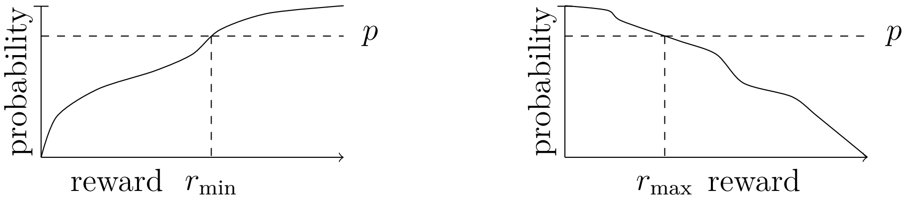

Figure 3.1: Quantiles for increasing and decreasing path properties

Figure 3.1 depicts the typical shape for quantiles. On the left-hand side of the figure an increasing path property $\varphi[r]$ is shown. Here, a quantile asks for the minimum reward that needs to be accumulated in order to guarantee the desired probability threshold p ($r_{\min} \overset{\text{def}}{=} \mathrm{qu}_s\big(Q\mathbb{P}_{\trianglerighteq p}(\varphi[?])\big)$ with $Q \in \{\exists, \forall\}$). A typical application for such a quantile is the reasoning about the minimal energy consumption, since normally the utilisation of additional energy increases the reliability of achieving a desired way of behaviour. On the right-hand side of Figure 3.1 a decreasing path property $\psi[r]$ is used, and here a quantile asks for the maximal reward that can be accumulated before the probability drops below the specified threshold p ($r_{\max} \overset{\text{def}}{=} \mathrm{qu}_s\big(Q\mathbb{P}_{\trianglerighteq p}(\psi[?])\big)$ with $Q \in \{\exists, \forall\}$). We will benefit from those quantiles for reasoning about the maximal obtained utility that can be achieved with some fixed energy budget.

The calculation of quantiles where the accumulated reward is bounded from above was already investigated in [UB13]. So, the quantiles studied there are obtained by considering $\varphi[r] = A\,\mathcal{U}^{\leqslant r}\,B$ and $\psi[r] = (\neg A)\ \mathcal{R}^{\leqslant r}\ (\neg B)$. In the current chapter we pay attention to the calculation of those quantiles and also address until-properties with lower reward bounds, i.e., $\varphi[r] = A\,\mathcal{U}^{\geqslant r}\,B$ and $\psi[r] = (\neg A)\ \mathcal{R}^{\geqslant r}\ (\neg B)$. This work was previously published in [Bai+14a] and builds the theoretical foundation of this monograph.

Optimal and adversarial schedulers Considering an existential quantile, say $r = \mathrm{qu}_s(\exists\mathbb{P}_{>p}(\varphi[?]))$, scheduler $\mathfrak{S}$ is said to be *optimal* if $\mathsf{Pr}_s^{\mathfrak{S}}(\varphi[r]) > p$. Scheduler $\mathfrak{S}$ is called *adversarial* for the universal quantile $r = \mathrm{qu}_s(\forall\mathbb{P}_{>p}(\varphi[?]))$ if $\mathsf{Pr}_s^{\mathfrak{S}}(\varphi[r{-}1]) \leqslant p$. The definition of optimal and adversarial schedulers for other quantiles is analogous.

3.2 Dualities

Each of the presented quantiles allows to derive three additional quantiles by the application of duality arguments, e.g., $\mathsf{Pr}_s^{\max}(\varphi[r]) = 1 - \mathsf{Pr}_s^{\min}(\psi[r])$, and the fact that $\min\{r \in \mathbb{N} : s \models \Phi[r]\}$ equals $\max\{r \in \mathbb{N} : s \not\models \Phi[r{-}1]\}$ for an increasing state property $\Phi[r]$. Therefore, the following can be deduced:

$$\begin{aligned}\min\bigl\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s^{\max}\bigl(\varphi[r]\bigr) > p\bigr\} &= \min\bigl\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s^{\min}\bigl(\psi[r]\bigr) < 1{-}p\bigr\}\\ &= \max\bigl\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s^{\min}\bigl(\psi[r{-}1]\bigr) \geqslant 1{-}p\bigr\}\\ &= \max\bigl\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s^{\max}\bigl(\varphi[r{-}1]\bigr) \leqslant p\bigr\}\end{aligned}$$

And this observation directly yields groups of four quantiles that are derivable from each other. So, it can be concluded that:

$$\begin{aligned}\mathrm{qu}_s\bigl(\exists\mathbb{P}_{>p}(\varphi[?])\bigr) &= \mathrm{qu}_s\bigl(\exists\mathbb{P}_{<1-p}(\psi[?])\bigr)\\ &= \mathrm{qu}_s\bigl(\forall\mathbb{P}_{\geqslant 1-p}(\psi[?])\bigr) + 1\\ &= \mathrm{qu}_s\bigl(\forall\mathbb{P}_{\leqslant p}(\varphi[?])\bigr) + 1\end{aligned}$$

Using arguments like these, it is possible to present an overview on the dualities of quantiles.

Concerning increasing state properties:

$$\begin{aligned}\mathrm{qu}_s\bigl(\exists\mathbb{P}_{>p}(A\,\mathcal{U}^{\leqslant ?}\,B)\bigr) &= \mathrm{qu}_s\bigl(\exists\mathbb{P}_{<1-p}((\neg A)\ \mathcal{R}^{\leqslant ?}\ (\neg B))\bigr)\\ &= \mathrm{qu}_s\bigl(\forall\mathbb{P}_{\geqslant 1-p}((\neg A)\ \mathcal{R}^{\leqslant ?}\ (\neg B))\bigr) + 1\\ &= \mathrm{qu}_s\bigl(\forall\mathbb{P}_{\leqslant p}(A\,\mathcal{U}^{\leqslant ?}\,B)\bigr) + 1\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\bigl(\exists\mathbb{P}_{>p}(A\,\mathcal{R}^{\geqslant ?}\,B)\bigr) &= \mathrm{qu}_s\bigl(\exists\mathbb{P}_{<1-p}((\neg A)\ \mathcal{U}^{\geqslant ?}\ (\neg B))\bigr)\\ &= \mathrm{qu}_s\bigl(\forall\mathbb{P}_{\geqslant 1-p}((\neg A)\ \mathcal{U}^{\geqslant ?}\ (\neg B))\bigr) + 1\\ &= \mathrm{qu}_s\bigl(\forall\mathbb{P}_{\leqslant p}(A\,\mathcal{R}^{\geqslant ?}\,B)\bigr) + 1\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\big(\forall\mathbb{P}_{>p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big) &= \mathrm{qu}_s\big(\forall\mathbb{P}_{<1-p}((\neg A)\,\mathcal{R}^{\leqslant ?}\,(\neg B))\big)\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\geqslant 1-p}((\neg A)\,\mathcal{R}^{\leqslant ?}\,(\neg B))\big)+1\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\leqslant p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)+1\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\big(\forall\mathbb{P}_{>p}(A\,\mathcal{R}^{\geqslant ?}\,B)\big) &= \mathrm{qu}_s\big(\forall\mathbb{P}_{<1-p}((\neg A)\,\mathcal{U}^{\geqslant ?}\,(\neg B))\big)\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\geqslant 1-p}((\neg A)\,\mathcal{U}^{\geqslant ?}\,(\neg B))\big)+1\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\leqslant p}(A\,\mathcal{R}^{\geqslant ?}\,B)\big)+1\end{aligned}$$

Concerning decreasing state properties:

$$\begin{aligned}\mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(A\,\mathcal{U}^{\geqslant ?}\,B)\big) &= \mathrm{qu}_s\big(\exists\mathbb{P}_{<1-p}((\neg A)\,\mathcal{R}^{\geqslant ?}\,(\neg B))\big)\\ &= \mathrm{qu}_s\big(\forall\mathbb{P}_{\geqslant 1-p}((\neg A)\,\mathcal{R}^{\geqslant ?}\,(\neg B))\big)-1\\ &= \mathrm{qu}_s\big(\forall\mathbb{P}_{\leqslant p}(A\,\mathcal{U}^{\geqslant ?}\,B)\big)-1\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(A\,\mathcal{R}^{\leqslant ?}\,B)\big) &= \mathrm{qu}_s\big(\exists\mathbb{P}_{<1-p}((\neg A)\,\mathcal{U}^{\leqslant ?}\,(\neg B))\big)\\ &= \mathrm{qu}_s\big(\forall\mathbb{P}_{\geqslant 1-p}((\neg A)\,\mathcal{U}^{\leqslant ?}\,(\neg B))\big)-1\\ &= \mathrm{qu}_s\big(\forall\mathbb{P}_{\leqslant p}(A\,\mathcal{R}^{\leqslant ?}\,B)\big)-1\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\big(\forall\mathbb{P}_{>p}(A\,\mathcal{U}^{\geqslant ?}\,B)\big) &= \mathrm{qu}_s\big(\forall\mathbb{P}_{<1-p}((\neg A)\,\mathcal{R}^{\geqslant ?}\,(\neg B))\big)\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\geqslant 1-p}((\neg A)\,\mathcal{R}^{\geqslant ?}\,(\neg B))\big)-1\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\leqslant p}(A\,\mathcal{U}^{\geqslant ?}\,B)\big)-1\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\big(\forall\mathbb{P}_{>p}(A\,\mathcal{R}^{\leqslant ?}\,B)\big) &= \mathrm{qu}_s\big(\forall\mathbb{P}_{<1-p}((\neg A)\,\mathcal{U}^{\leqslant ?}\,(\neg B))\big)\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\geqslant 1-p}((\neg A)\,\mathcal{U}^{\leqslant ?}\,(\neg B))\big)-1\\ &= \mathrm{qu}_s\big(\exists\mathbb{P}_{\leqslant p}(A\,\mathcal{R}^{\leqslant ?}\,B)\big)-1\end{aligned}$$

Those dualities are helpful, since they allow to restrict the computation to several basic quantiles and the results for other queries can be derived based on the presented dualities.

3.3 Upper-reward bounded quantiles

Now, we investigate computation-methods for quantiles over (constrained) reachability properties where the accumulation of the reward is bounded from above. So, we study quantiles of the form

$$\mathrm{qu}_s\big(Q\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)$$

for arbitrary sets $A, B \subseteq \mathsf{S}$, $\trianglerighteq \in \{>, \geqslant\}$, and $Q \in \{\exists, \forall\}$. Quantiles of the presented form ask for the minimum reward-accumulation until a certain event occurs (i.e., a specific state of the model has been reached) with probability of at least p, and therefore a typical application for upper-reward bounded quantiles is the analysis of the minimal energy that a system consumes in order to supply a desired amount of utility. Since this allows to analyse the minimal energy consumption of a given system within a variety of different scenarios, this nicely integrates into HAEC (see Chapter 1).

The computation of those quantiles restricted to the case of state rewards has been previously presented in [UB13]. Here, we recall the algorithms that are necessary to calculate the quantiles, and simultaneously the algorithms will be extended for supporting transition rewards as well. Those algorithms serve then as the basis for the implementation of the quantile framework used for the computations shown in Chapter 6.

The first thing one needs to do before starting to compute the quantile values is to ensure that the demanded quantile does actually exist. As Figure 3.2 suggests, it can be the case that the (constrained) reachability property can not be fulfilled for probability threshold p no matter the accumulated reward. Therefore, the accumulation of the reward in this case is absolutely meaningless, as the fulfilment of the property will not be influenced by its accumulation. So, in order to make sure that a quantile exists, a preceding precomputation is needed. Using such a precomputation it can be ensured that the reachability property can be fulfilled and therefore the demanded quantile is finite. Or, if it is not possible to fulfill the reachability property, the computation simply returns ∞ and stops.

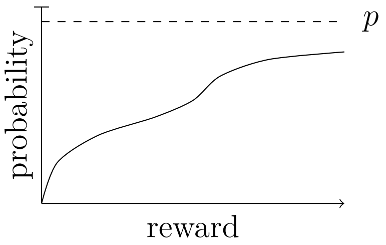

Figure 3.2: Quantile for increasing path property

3.3.1 Precomputation

The precomputation for queries of the form $\mathrm{qu}_s(Q\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B))$ (with $Q \in \{\exists, \forall\}$) can be done by removing the reward bound and calculating the unbounded (constrained) maximum (or minimum) reachability probability $p_s \overset{\text{def}}{=} \mathsf{Pr}_s^{\max}(A\,\mathcal{U}\,B)$ (or $\mathsf{Pr}_s^{\min}(A\,\mathcal{U}\,B)$, respectively) for $s \in \mathsf{S}$. Using standard model-checking techniques, this can be done in time polynomial in the size of the given model (see, e.g., [BK08, Chapter 10]). As [UB13] shows in Lemma 10 and in Lemma 13, the resulting probability p_s will be then compared with the specified probability threshold p according to $\trianglerighteq$. If $\trianglerighteq\, = \,>$ and $p \geqslant p_s$, or if $\trianglerighteq\, = \,\geqslant$ and $p > p_s$, it is impossible to guarantee the desired threshold, and the computation will be stopped and ∞ will be returned as the result. Otherwise, the demanded quantile value is finite and we can use an iterative algorithm (as sketched in the next section) for finding the least i such that $\mathsf{Pr}_s^{\max}(A\,\mathcal{U}^{\leqslant i}\,B) \trianglerighteq p$ (or $\mathsf{Pr}_s^{\min}(A\,\mathcal{U}^{\leqslant i}\,B) \trianglerighteq p$, respectively).

So, the intuition behind the precomputation is to ensure that the succeeding iterative computation scheme will be only used when it can be guaranteed that this procedure will terminate after a finite number of iterations.

3.3.2 Computation scheme

[UB13] presents a linear-programming approach for the computation of quantiles over (constrained) reachability properties with upper reward bounds (briefly called $\mathcal{U}^{\leqslant ?}$-quantiles) over **MDP**s annotated with state rewards. The paper provides related approaches for both, existential and universal quantiles. We recall the approach for existential $\mathcal{U}^{\leqslant ?}$-quantiles, and as [UB13] considers state rewards, the approach will be simultaneously extended to be applicable for state-action rewards as well.

The main ingredient for the computation of quantiles is the calculation of (constrained) reachability probabilities for all states of the model when the accumulation of the reward has been restricted from above to a certain bound. So, we want to solve a linear program that incorporates multiple bounded reachability probabilities, and Figure 3.3 depicts the linear program of [UB13] that needs to be solved for the computation of existential upper-reward bounded quantiles, adapted for the case of state-action rewards (rather than state rewards)[1]. So, we compute reachability probabilities for all existing states for a sequence of bounds. Lemma 10 and Lemma 13 in [UB13] show that it is completely sufficient to analyse the reward-bounded reachability up to a specific (exponential) upper bound r_{max}. If the requested quantile is finite it can be guaranteed that its value can not be greater than this bound. This LP-based computation scheme can be therefore solved in exponential time, using this exponential bound for the smallest (finite) quantile. Therefore, the computation of upper-reward bounded $\mathcal{U}^{\leqslant ?}$-quantiles in **MDP**s can be evaluated in exponential time (as also stated in Theorem 11 and Theorem 14 of [UB13]).

---

[1] The intuitive meaning of $x_{s,i}$ is the probability of reaching B through A-states from the state s such that a reward budget of i will not be overspent.

$$\text{minimise} \sum_{(s,i)\in \mathsf{S}[r]} x_{s,i} \text{ subject to}$$

$$\begin{aligned}
x_{s,i} &= 0 && \text{if } s \not\models \exists(A\,\mathcal{U}\,B) \text{ and } 0 \leqslant i \leqslant r\\
x_{s,i} &= 1 && \text{if } s \in B \text{ and } 0 \leqslant i \leqslant r\\
x_{s,i} &\geqslant \sum_{t\in\mathsf{S}} P(s,\alpha,t)\cdot x_{t,i-rew(s,\alpha)} && \text{if } s \notin B, s \models \exists(A\,\mathcal{U}\,B) \text{ and } \alpha \in \mathsf{Act}(s)\\
& && \text{such that } \mathsf{rew}(s,\alpha) \leqslant i \leqslant r
\end{aligned}$$

Figure 3.3: Linear program $\mathbb{LP}_r$ with the unique solution $p_{s,i} = \mathsf{Pr}_s^{\max}\big(A\,\mathcal{U}^{\leqslant i}\,B\big)$

A naïve approach thus could first compute r_{max}, generate the linear program with variables $x_{s,i}$ for $(s,i) \in \mathsf{S}[r_{\mathsf{max}}]$ and then use general-purpose linear- or dynamic-programming techniques to solve the constructed linear program (e.g., the Simplex algorithm [Nas00], ellipsoid methods [GLS93] or value [Bel57] or policy iteration [How90]). However, since the upper bound r_{max} is exponential in the size of $\mathcal{M}$ and depends on the transition probabilities and rewards in $\mathcal{M}$ and the specified probability bound p, this approach turns out to be intractable when $\mathcal{M}$ or the reward values are large. It turns out that in practice the demanded quantile values are normally much smaller than the theoretically established bound r_{max} (see the different analysis-results presented in Chapter 6). Even for those cases where the calculations do not consider all theoretically possible iterations an approach that tries to solve the linear program in one single self-contained step by using general-purpose methods like an LP-solver is not feasible (see Table 6.3 in Section 6.1.1 (Self-Stabilising Protocol) or Table 6.8 in Section 6.1.2 (Asynchronous Leader-Election Protocol) for more details on this incident). Instead, it is recommended to compute the reward-bounded reachability probabilities using the iterative *back-propagation* procedure described in Section 5.1.1.

Anyways, since a precomputation procedure has already been passed, we can assume that the user-specified probability threshold p can be met. We use this information and therefore compute the maximal probabilities $p_{s,r} = \mathsf{Pr}_s^{\max}\big(A\,\mathcal{U}^{\leqslant r}\,B\big)$ for increasing reward bound r (starting with 0), until $p_{s,r} \trianglerighteq p$ holds for the first time. Then, r will be returned as the result of the computation and corresponds to the demanded quantile value.

The computation of universal $\mathcal{U}^{\leqslant ?}$-quantiles can be done using an analogous approach where some details need to be adapted slightly. As already stated in Section 3.2, quantiles that refer to reward-bounded release formulas are dual and can be computed using the same techniques.

A simple consequence of the representation of $\mathsf{Pr}_s^{\max}\big(A\,\mathcal{U}^{\leqslant r}\,B\big)$ as the unique solution of the linear program in Figure 3.3 is the existence of a finite-memory scheduler $\mathfrak{S}$ with the modes $0, 1, \ldots, r$ that maximises the probability for $A\,\mathcal{U}^{\leqslant r}\,B$. In particular, we obtain:

Lemma 3.3.1 (Optimal schedulers for until-quantiles). *For each of the existential* *until-quantiles there exists an optimal scheduler* $\mathfrak{S}$ *such that for all finite paths* ρ_1 *and* ρ_2*:*

$$\textit{if } \textit{last}(\rho_1) = \textit{last}(\rho_2) \textit{ and } \mathsf{rew}(\rho_1) = \mathsf{rew}(\rho_2) \textit{ then } \mathfrak{S}(\rho_1) = \mathfrak{S}(\rho_2) \tag{3.1}$$

For existential quantiles $r = \mathrm{qu}_s(\exists \ldots)$ *there are optimal finite-memory schedulers whose size is bounded by* $r + 1$*.*

Similarly, the universal until-quantiles have adversarial schedulers $\mathfrak{S}$ satisfying statement 3.1. Thus, an analogous statement holds for universal until-quantiles and adversarial schedulers.

[HK15] presents an **ExpTime**-completeness result for the computation of cost problems for general cost processes. The authors show the **ExpTime**-hardness by reducing the problem of determining the winner in a countdown game [JLS07] to a cost problem over cost processes by constructing a cost process such that a player of the countdown game can only win if and only if there exists a scheduler solving the considered cost problem. Since cost problems are directly related to the computation of quantiles over **MDP**s, there is no hope to improve the computation of quantiles substantially by utilising another class of algorithms. Nevertheless, Section 5.1 presents several methods to enhance the computation of quantiles / reward-bounded reachability probabilities that improve the practical performance of the computations significantly. It is even the case that the calculations carried out practically are impossible in some places without the utilisation of those optimisations.

3.3.3 Qualitative quantiles

Now, we investigate a special case for the computation of upper-reward bounded quantile queries, i.e., the computation of *qualitative* quantiles where the probability threshold is either 0 or 1. So, we consider queries of the form

$$\mathrm{qu}_s\big(Q\mathbb{P}_\theta(A \,\mathcal{U}^{\leqslant ?}\, B)\big)$$

with $Q \in \{\forall, \exists\}$, $\theta \in \{> 0, = 1\}$. The computation of qualitative quantiles allows the usage of less expensive methods and was studied in [UB13, Section 4] restricted for the case of state rewards. For the computation a polynomial time-algorithm was presented in [UB13, Algorithm 1], which relies on iterated reachability computations analysing the structure of the graph representing the model under consideration. Figure 3.4 shows this algorithm for the computation of qualitative quantiles where the considered reward function is restricted to state rewards. As can be seen, no methods for solving linear programs are needed in this case. So, the implementation of this algorithm could be done straight forward by utilising the graph-based methods already available for doing standard calculations of probabilistic model checkers.

Since the application of quantiles for the analysis of energy-critical systems revealed that a tool support that is restricted to state rewards only does not meet the requirements in many cases, the demand arose that the framework should provide support for

```
Input: MDP $\mathcal{M} = (\mathsf{S}, \mathsf{Act}, P)$, state-reward function $\mathsf{rew} : \mathsf{S} \to \mathbb{N}$,
       quantile query $\mathrm{qu}(Q(A \, \mathcal{U}^{\leqslant ?} \, B))$ with $Q \in \{\forall\mathbb{P}_{>0}, \exists\mathbb{P}_{>0}, \forall\mathbb{P}_{=1}, \exists\mathbb{P}_{=1}\}$
for each $s \in \mathsf{S}$ do
  if $s \in B$ then $v(s) \leftarrow 0$ else $v(s) \leftarrow \infty$
done
$X \leftarrow \{s \in \mathsf{S} : v(s) = 0\}$
$R \leftarrow \{0\}$
$Z \leftarrow \{s \in \mathsf{S} : s \in A \backslash B$ and $\mathsf{rew}(s) = 0\}$
while $R \neq \varnothing$ do
  $r \leftarrow \min R$
  $Y \leftarrow \{s \in X : v(s) \leqslant r\} \backslash Z$
  for each $s \in \mathsf{S} \backslash X$ with $s \in A$ and $s \models Q \bigcirc (Z \, \mathcal{U} \, Y)$ do
    $v(s) \leftarrow r + \mathsf{rew}(s)$
    $X \leftarrow X \cup \{s\}$
    $R \leftarrow R \cup \{v(s)\}$
  done
  $R \leftarrow R \backslash \{r\}$
done
return $v$
```

Figure 3.4: Algorithm 1 from [UB13]

transition rewards as well. In order to allow transition rewards, a transformation is used to encode the transition-reward function $\mathsf{rew} : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ over system $\mathcal{M}$ into a new reward function $\widetilde{\mathsf{rew}} : \widetilde{\mathsf{S}} \to \mathbb{N}$ (consisting of state rewards only) over the transformed system $\widetilde{\mathcal{M}}$ by introducing intermediate states. Those additional states simply serve as an emulation for the transition rewards of the original model. This encoding then allows the application of [UB13, Algorithm 1] (as shown in Figure 3.4) for the computation of qualitative quantiles when the model under consideration is equipped with transition rewards. Figure 3.5 depicts the idea of the encoding transformation when $\mathsf{rew}(s, \alpha) = 0$, $\mathsf{rew}(s, \beta) > 0$, and $\mathsf{rew}(t, \gamma) > 0$. The figure shows that only rewards greater zero are taken into account and the zero-reward transitions stay without any modification.

Formally, for a given **MDP** $\mathcal{M} = (\mathsf{S}, \mathsf{Act}, P)$ and a specific transition-reward function $\mathsf{rew} : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$, the transformation defines a new **MDP** $\widetilde{\mathcal{M}} = (\widetilde{\mathsf{S}}, \widetilde{\mathsf{Act}}, \widetilde{P})$ with

- $\widetilde{\mathsf{S}} = \mathsf{S} \cup \{s_\alpha : s \in \mathsf{S}, \alpha \in \mathsf{Act}(s), \mathsf{rew}(s, \alpha) > 0\}$

- $\widetilde{\mathsf{Act}} = \mathsf{Act} \cup \{\hat{\alpha} : \exists s \in \mathsf{S} \text{ s.t. } \alpha \in \mathsf{Act}(s), \mathsf{rew}(s, \alpha) > 0\}$

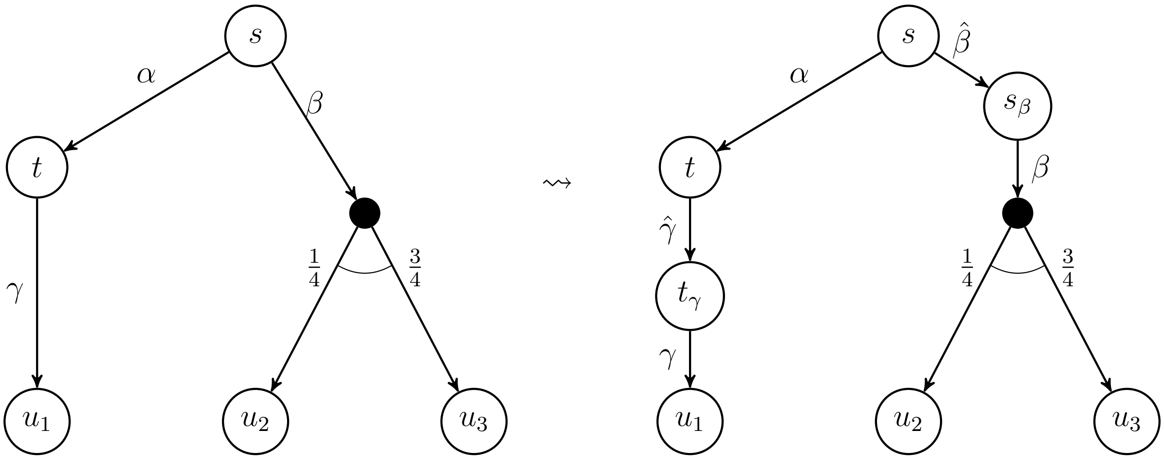

Figure 3.5: Encoding of transition rewards into state rewards

- $\widetilde{P}(s,\alpha,t) = \begin{cases} P(s,\alpha,t) & \text{if } \mathsf{rew}(s,\alpha) = 0 \\ 0 & \text{if } \mathsf{rew}(s,\alpha) > 0 \text{ and } t \neq s_\alpha \\ 1 & \text{if } \mathsf{rew}(s,\alpha) > 0 \text{ and } t = s_\alpha \\ P(u,\beta,t) & \text{if } s = u_\beta \text{ for } u \in \mathsf{S}, \beta \in \mathsf{Act}(u), \mathsf{rew}(u,\beta) > 0 \end{cases}$

and a new state-reward function $\widetilde{\mathsf{rew}} : \widetilde{\mathsf{S}} \to \mathbb{N}$ with

$$\widetilde{\mathsf{rew}}(t) = \begin{cases} 0 & \text{if } t \in \mathsf{S} \\ \mathsf{rew}(s,\alpha) & \text{if } t = s_\alpha \text{ for } s \in \mathsf{S}, \alpha \in \mathsf{Act}(s) \end{cases}$$

From a theoretical point of view it is possible to introduce a newly created intermediate state for each state-action pair (even if its reward is zero), but since the calculation of qualitative quantiles is currently only supported for an explicit representation of the model's state space (see Section 5.2), this transformation would increase the size of the transformed model dramatically. So, for practical purposes it is much better to consider only the states that are essential for the encoding, and therefore utilise the presented transformation.

The following shows that the transformation indeed preserves the reachability probabilities, and therefore this transformation is sound.

Lemma 3.3.2. *Let $s \in \mathsf{S}$, $A, B \subseteq \mathsf{S}$ and $r \in \mathbb{N}$. For each scheduler $\mathfrak{S}$ in $\mathcal{M}$ there exists a scheduler $\widetilde{\mathfrak{S}}$ in $\widetilde{\mathcal{M}}$ such that*

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}(A \,\mathcal{U}^{\leqslant r}\, B) = \mathsf{Pr}^{\widetilde{\mathfrak{S}}}_{\widetilde{\mathcal{M}},s}(A' \,\mathcal{U}^{\leqslant r}\, B),$$

where $A' = A \cup \{s_\alpha : s \in A, \alpha \in \mathsf{Act}(s)\}$. For each scheduler $\widetilde{\mathfrak{S}}$ in $\widetilde{\mathcal{M}}$ there exists a corresponding scheduler $\mathfrak{S}$ in $\mathcal{M}$ fulfilling the same equation.

Proof. Let $\rho = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\,\alpha_n\,s_n$ with $s = s_0$ and $t = s_n$ denote a finite path from state s to state t in $\mathcal{M}$. Using the transformation $\mathcal{M} \rightsquigarrow \widetilde{\mathcal{M}}$ there exists a unique finite path $\widetilde{\rho}$ in $\widetilde{\mathcal{M}}$ starting in s and ending in t. So, it is possible to assign each finite path in $\mathcal{M}$ to exactly one finite path in $\widetilde{\mathcal{M}}$. Vice versa, when a path in $\widetilde{\mathcal{M}}$ starts in an arbitrary state of S and also ends in a state of S it is possible to assign this path unambiguously to a path in $\mathcal{M}$. In this way one obtains a bijective function between paths in $\mathcal{M}$ and paths in $\widetilde{\mathcal{M}}$ starting in a state of S and also terminating in a state of S.

The probabilities along the paths ρ and $\widetilde{\rho}$ are the same, and also the accumulated reward along both paths is the same since the transformation introduces exactly the same reward to a newly created state s_α when $\mathsf{rew}(s, \alpha) > 0$.

It is possible to find a mimicking scheduler $\widetilde{\mathfrak{S}}$ over $\widetilde{\mathcal{M}}$ for an arbitrary scheduler $\mathfrak{S}$ over $\mathcal{M}$ using the following observations:

- For $s \in \mathsf{S}$ and $\alpha \in \mathsf{Act}(s)$ with $\mathsf{rew}(s, \alpha) = 0$, $\widetilde{\mathfrak{S}}$ simply picks $\alpha \in \widetilde{\mathsf{Act}}(s)$ in $\widetilde{\mathcal{M}}$ as this action was preserved by the transformation.

- If $s \in \mathsf{S}$ and $\alpha \in \mathsf{Act}(s)$ with $\mathsf{rew}(s, \alpha) > 0$, the scheduler $\widetilde{\mathfrak{S}}$ will pick action $\hat{\alpha} \in \widetilde{\mathsf{Act}}(s)$ which was introduced by the transformation. Since $\hat{\alpha}$ only has the newly created intermediate state s_α as the sole successor the positive transition-reward rew(s,α) will be added to the accumulated reward in state s_α in any case. In s_α the only available action is α and therefore will be taken and all successors of α in $\mathcal{M}$ can be reached with their respective probabilities.

Therefore, it is possible to mimic any scheduler over $\mathcal{M}$ in $\widetilde{\mathcal{M}}$ and the accumulated reward is the same for both schedulers.

Since the transformation $\mathcal{M} \rightsquigarrow \widetilde{\mathcal{M}}$ does not introduce any nondeterminism to the transformed model $\widetilde{\mathcal{M}}$, and since the previous observations can be inverted, each scheduler $\widetilde{\mathfrak{S}}$ over $\widetilde{\mathcal{M}}$ can also be mimicked by a scheduler over $\mathcal{M}$, again accumulating the same reward.

So, the following holds for arbitrary r and $s \in \mathsf{S}$:

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}(A\,\mathcal{U}^{\leqslant r}\,B) = \mathsf{Pr}^{\widetilde{\mathfrak{S}}}_{\widetilde{\mathcal{M}},s}((A \cup \{s_\alpha \,:\, s \in A, \alpha \in \mathsf{Act}(s)\})\,\mathcal{U}^{\leqslant r}\,B)$$

This shows the claim. □

Lemma 3.3.2 is of help when considering the computation of quantiles since direct consequences of this lemma are as follows:

Corollary 3.3.3. *Let $s \in \mathsf{S}$ and $A, B \subseteq \mathsf{S}$. For $Q \in \{\exists, \forall\}$, $\trianglerighteq \in \{\geqslant, >\}$, and $p \in [0,1] \cap \mathbb{Q}$ the following holds:*

$$\mathrm{qu}^{\mathcal{M}}_s(Q\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)) = \mathrm{qu}^{\widetilde{\mathcal{M}}}_s(Q\mathbb{P}_{\trianglerighteq p}(A'\,\mathcal{U}^{\leqslant ?}\,B)),$$

with $A' = A \cup \{s_\alpha \,:\, s \in A, \alpha \in \mathsf{Act}(s)\}$ as in Lemma 3.3.2.

Corollary 3.3.3 shows that the transformation is indeed quantile-preserving, and especially the computation of qualitative quantiles (where p is either 0 or 1) over models using a transition-reward function is therefore possible utilising the presented transformation and afterwards the qualitative algorithm introduced in [UB13, Algorithm 1].

3.4 Lower-reward bounded quantiles

Now, we want to present methods for computing quantiles where the accumulation of the reward is bounded from below. So, we analyse quantile queries of the form

$$\mathrm{qu}_s\big(Q\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\geqslant ?}\,B)\big) \tag{3.2}$$

with $A, B \subseteq \mathsf{S}$, $\trianglerighteq\, \in \{>, \geqslant\}$, and $Q \in \{\exists, \forall\}$. Here, we calculate the maximum accumulated reward before a certain event occurs with high probability p, and so it is possible to ask, e.g., for the maximal utility that can be provided by an energy-critical system when there is only a specific energy budget available. For example, one could ask for the maximal number of videos that can be decoded on a mobile device using only a specific portion of the power that will be provided by its battery.

But, instead of calculating the demanded values in a direct way, we propose methods for the calculation of

$$\mathrm{qu}_s\big(Q\mathbb{P}_{\trianglelefteq p}(A\,\mathcal{U}^{\geqslant ?}\,B)\big) \tag{3.3}$$

with $\trianglelefteq\, \in \{<, \leqslant\}$. Using the dualities presented in Section 3.2, we can derive the desired quantile value 3.2 from the value 3.3 by suitable transformations. So, instead of computing the maximum reward possible with a probability of at least p, the idea is to compute the minimal accumulated reward such that the probability becomes smaller than p for the first time. This enables the computation to rely on similar iteration-based methods as already presented for the calculation of upper-reward bounded quantiles. And, like it was already the case there we do need a mechanism guaranteeing that the demanded quantile really exists. Therefore, the computation also starts with a precomputation followed by the actual calculation of the demanded quantile value.

For simplicity, only the treatment of reachability ($\lozenge^{\geqslant ?}B$) with a lower reward bound will be sketched in the upcoming considerations.

3.4.1 Precomputation

In the following the set C denotes all states $t \in \mathsf{S}$ that are contained in some (maximal) end component $(T, \mathcal{A})$ with $\mathsf{rew}(t', \alpha) > 0$ for some state $t' \in T$ and some action $\alpha \in \mathcal{A}(t')$. So, C is the union of all end components possessing a positive reward.

At first, we want to learn under which circumstances it is possible to obtain finite quantile values for reachability quantiles with lower reward-bounds. Therefore the following lemma is very helpful when interested in the computation of a universal quantile.

Lemma 3.4.1. *For all states s in $\mathcal{M}$, we have:*

$$\mathrm{qu}_s\big(\forall\mathbb{P}_{<p}(\Diamond^{\geqslant ?}B)\big) = \infty \quad \textit{iff} \quad \mathsf{Pr}_s^{\max}\big(\Diamond(C \wedge \Diamond B)\big) \geqslant p$$

Proof. A first observation is as follows:

$$\begin{array}{ll} & \min\big\{r \in \mathbb{N} : \mathsf{Pr}_s^{\max}\big(\Diamond^{\geqslant r}B\big) < p\big\} = \infty \\ \text{iff} & \text{there is no } r \in \mathbb{N} \text{ such that } \mathsf{Pr}_s^{\max}\big(\Diamond^{\geqslant r}B\big) < p \\ \text{iff} & \text{for all } r \in \mathbb{N} \text{ there exists a scheduler } \mathfrak{S}_r \text{ with } \mathsf{Pr}_s^{\mathfrak{S}_r}\big(\Diamond^{\geqslant r}B\big) \geqslant p \end{array}$$

"$\Longleftarrow$": Suppose $\mathsf{Pr}_s^{\max}\big(\Diamond(C \wedge \Diamond B)\big) \geqslant p$. Let $\mathfrak{S}_{opt}$ be a (finite-memory) scheduler that maximises the probability for $\Diamond(C \wedge \Diamond B)$ for all states. Now, pick some (finite-memory) scheduler $\mathfrak{S}_C$ such that from each state $t \in C$ with probability 1 all states of the maximal end component $(T, \mathcal{A})$ with $t \in T$ will be visited infinitely often and each of its actions will be taken infinitely often. Then, the accumulated reward for almost all infinite $\mathfrak{S}_C$-paths starting in a C-state is ∞. Furthermore, let $\mathfrak{S}_{\Diamond B}$ be a (memoryless) scheduler that maximises the probability to reach B for all states. Given $r \in \mathbb{N}$, we now regard the scheduler $\mathfrak{S}_r$ that operates in three phases:

- Phase 1: As long as C has not been reached, $\mathfrak{S}_r$ behaves as $\mathfrak{S}_{opt}$. As soon as C has been reached, $\mathfrak{S}_r$ switches from phase 1 to phase 2.
- Phase 2: $\mathfrak{S}_r$ mimics $\mathfrak{S}_C$, provided that the total accumulated reward is less than r. If the current state belongs to C and the total accumulated reward is larger or equal r then $\mathfrak{S}_r$ switches from phase 2 to phase 3.
- Phase 3: $\mathfrak{S}_r$ behaves as $\mathfrak{S}_{\Diamond B}$.

When entering a C-state in the first phase and the total accumulated reward is $\geqslant r$ then $\mathfrak{S}_r$ can move directly from phase 1 to phase 3.

Now, the fact is used that all states that belong to the same maximal end component have the same maximal reachability probabilities [Cie+08]. This yields that for each $\rho \in \{0, 1, \ldots, r-1\}$ and all states t of a maximal end component $(T, \mathcal{A})$ of $\mathcal{M}$ that contains at least one state-action pair with positive reward:

$$\begin{aligned} & \mathsf{Pr}^{\mathfrak{S}_r}_{t|\mathsf{rew}=\rho}\big(\Diamond^{\geqslant r}B\big) \\ =& \sum_{t' \in T} \mathsf{Pr}^{\mathfrak{S}_C}_{t|\mathsf{rew}=\rho}\big(C\,\mathcal{U}^{\geqslant r}\,t'\big) \cdot \mathsf{Pr}^{\mathfrak{S}_{\Diamond B}}_{t'}\big(\Diamond B\big) \\ =& \sum_{t' \in T} \mathsf{Pr}^{\mathfrak{S}_C}_{t|\mathsf{rew}=\rho}\big(C\,\mathcal{U}^{\geqslant r}\,t'\big) \cdot \mathsf{Pr}^{\max}_{t}\big(\Diamond B\big) \\ =& \mathsf{Pr}^{\max}_{t}\big(\Diamond B\big) \end{aligned}$$

Here, the notation $\mathsf{Pr}^{\mathfrak{S}_r}_{t|\mathsf{rew}=\rho}$ indicates the probability under $\mathfrak{S}_r$ under the condition that state t has been just entered by switching from phase 1 to phase 2, while the accumulated reward is ρ. We obtain:

$$\begin{aligned}
&\mathsf{Pr}_s^{\mathfrak{S}_r}\big(\Diamond^{\geqslant r}B\big)\\
=&\sum_{0\leqslant\rho<r}\sum_{(T,\mathcal{A})}\sum_{t\in T}\mathsf{Pr}_s^{\mathfrak{S}_r}\big(\neg C\,\mathcal{U}\,((\mathsf{rew}=\rho)\wedge t)\big)\cdot\mathsf{Pr}_{t|\mathsf{rew}=\rho}^{\mathfrak{S}_r}\big(\Diamond^{\geqslant r}B\big)\\
&+\sum_{t\in C}\mathsf{Pr}_s^{\mathfrak{S}_r}\big(\neg C\,\mathcal{U}\,((\mathsf{rew}\geqslant r)\wedge t)\big)\cdot\mathsf{Pr}_t^{\mathfrak{S}_{\Diamond B}}\big(\Diamond B\big)\\
=&\sum_{t\in C}\mathsf{Pr}_s^{\mathfrak{S}_{opt}}\big(\neg C\,\mathcal{U}\,t\big)\cdot\mathsf{Pr}_t^{\max}\big(\Diamond B\big)\\
=&\mathsf{Pr}_s^{\max}\big(\Diamond(C\wedge\Diamond B)\big)
\end{aligned}$$

where $(T,\mathcal{A})$ ranges over all maximal end components that contain a state-action pair with positive reward.

"$\Longrightarrow$": We now suppose that the quantile for the state s and the objective $\forall\mathbb{P}_{<p}(\Diamond^{\geqslant ?}B)$ is ∞. Let $\big(\mathfrak{S}_r\big)_{r\in\mathbb{N}}$ be a family of schedulers such that:

$$\mathsf{Pr}_s^{\mathfrak{S}_r}\big(\Diamond^{\geqslant r}B\big)\geqslant p$$

The task is to show that $\mathsf{Pr}_s^{\max}(\Diamond(C\wedge\Diamond B))\geqslant p$. For this it is shown that for each $\varepsilon>0$ there exists a scheduler $\mathfrak{S}$ such that $\mathsf{Pr}_s^{\mathfrak{S}}(\Diamond(C\wedge\Diamond B))\geqslant p-\varepsilon$.

In what follows, some positive ε is fixed. There exists some $r\in\mathbb{N}$ such that for each scheduler $\mathfrak{S}$:

$$\mathsf{Pr}_s^{\mathfrak{S}}\big((\neg C)\,\mathcal{U}^{\geqslant r}\,B\big)<\varepsilon$$

This is due to the fact that the limit of almost all $\mathfrak{S}$-paths constitutes an end component and that the reward earned in end components not contained in C is zero. For scheduler $\mathfrak{S}=\mathfrak{S}_r$ we obtain:

$$\begin{aligned}
p&\leqslant\mathsf{Pr}_s^{\mathfrak{S}}\big(\Diamond^{\geqslant r}B\big)\\
&=\mathsf{Pr}_s^{\mathfrak{S}}\big((\neg C)\,\mathcal{U}^{\geqslant r}\,B\big)+\mathsf{Pr}_s^{\mathfrak{S}}\big(\Diamond(C\wedge\Diamond((\mathsf{rew}\geqslant r)\wedge B))\big)\\
&\leqslant\varepsilon+\mathsf{Pr}_s^{\mathfrak{S}}\big(\Diamond(C\wedge\Diamond B)\big)
\end{aligned}$$

Hence, $\mathsf{Pr}_s^{\mathfrak{S}}\big(\Diamond(C\wedge\Diamond B)\big)$ is at least $p-\varepsilon$. □

Let $\widetilde{\mathcal{M}}$ be the MDP that results from $\mathcal{M}$ by adding two new states *goal* and *fail* and a fresh action symbol τ with transition probabilities:

$$\begin{aligned}
P(t,\tau,goal)&=\mathsf{Pr}_{\mathcal{M},t}^{\max}\big(\Diamond B\big)\\
P(t,\tau,fail)&=1-\mathsf{Pr}_{\mathcal{M},t}^{\max}\big(\Diamond B\big)
\end{aligned}$$

if $t\in C$ and $P(s,\tau,s')=0$ for all states $s\in\mathsf{S}\backslash C$, $s'\in\mathsf{S}$. The outgoing transitions of the new states *goal* and *fail* are irrelevant for our purposes. We then have:

$$\mathsf{Pr}_{\mathcal{M},s}^{\max}\big(\Diamond(C\wedge\Diamond B)\big)=\mathsf{Pr}_{\widetilde{\mathcal{M}},s}^{\max}\big(\Diamond goal\big)$$

The generation of $\widetilde{\mathcal{M}}$ mainly requires the computation of the values $\mathsf{Pr}^{\max}_{\mathcal{M},s}(\Diamond B)$ and the computation of the maximal end components of $\mathcal{M}$. The former can be done using graph algorithms and linear-programming techniques in time polynomial in the size of $\mathcal{M}$, while the latter is possible using standard algorithms in time quadratic in the size of the underlying graph of $\mathcal{M}$.

Now, that we have seen the criteria that need to be fulfilled in order to guarantee a finite value for a universal lower-reward bounded quantile, we want to investigate the same in the case of an existential quantile. We therefore start our investigation with the following logical consequences:

$$\begin{aligned}
&\min\big\{r \in \mathbb{N} : \mathsf{Pr}^{\min}_s\big(\Diamond^{\geqslant r} B\big) < p\big\} = \infty \\
\text{iff}\quad &\text{there is no } r \in \mathbb{N} \text{ such that } \mathsf{Pr}^{\min}_s\big(\Diamond^{\geqslant r} B\big) < p \\
\text{iff}\quad &\mathsf{Pr}^{\min}_s\big(\Diamond^{\geqslant r} B\big) \geqslant p \text{ for all } r \in \mathbb{N}
\end{aligned}$$

Obviously, this is the case if under each scheduler, with probability at least p, the set B will be visited infinitely often and the accumulated reward tends to infinity. A direct consequence of those considerations is stated in the following lemma. Therefore, let $\mathit{posRew} \subseteq \mathsf{S} \times \mathsf{Act}$ be the set of state-action pairs (s, α) with $\mathsf{rew}(s, \alpha) > 0$.

Lemma 3.4.2. *For all states s in $\mathcal{M}$, we have:*

$$\mathrm{qu}_s\big(\exists \mathbb{P}_{<p}(\Diamond^{\geqslant ?} B)\big) = \infty \quad \textit{iff} \quad \mathsf{Pr}^{\min}_s\big(\Box\Diamond B \wedge \Box\Diamond \mathit{posRew}\big) \geqslant p$$

In order to compute the minimal probability for the generalised Büchi condition $\Box\Diamond B \wedge \Box\Diamond \mathit{posRew}$ we can rely on standard techniques. We compute the set D consisting of states that are contained in some end component $(T, \mathcal{A})$ with $T \cap B = \varnothing$ or with $\mathsf{rew}(t', \alpha) = 0$ for all actions $\alpha \in \mathcal{A}(t')$ and states $t' \in T$. We then have:

$$\mathsf{Pr}^{\min}_s\big(\Box\Diamond B \wedge \Box\Diamond \mathit{posRew}\big) = 1 - \mathsf{Pr}^{\max}_s\big(\Diamond D\big)$$

Using the previous statements, it is possible to formulate the following corollary enabling to check if a quantile exists or if it is infinite.

Corollary 3.4.3. *The following two problems are in **P**:*

(1) decide whether $\mathrm{qu}_s\big(\forall \mathbb{P}_{<p}(\Diamond^{\geqslant ?} B)\big) = \infty$

(2) decide whether $\mathrm{qu}_s\big(\exists \mathbb{P}_{<p}(\Diamond^{\geqslant ?} B)\big) = \infty$

3.4.2 Computation scheme

The approach for computing upper-reward bounded quantiles as in Section 3.3.2 can be adapted to the computation of quantiles for reachability formulas with lower reward bounds, i.e., $\Diamond^{\geqslant ?} B$. We start with the universal quantile:

$$\mathrm{qu}_s\big(\forall \mathbb{P}_{<p}(\Diamond^{\geqslant ?} B)\big) = \min\big\{r \in \mathbb{N} : \mathsf{Pr}^{\max}_s\big(\Diamond^{\geqslant r} B\big) < p\big\}$$

Clearly, if $\mathsf{Pr}_s^{\max}(\lozenge B) < p$ then the quantile for state s is 0. Furthermore (see Lemma 3.4.1):

$$\mathrm{qu}_s\big(\forall\mathbb{P}_{<p}(\lozenge^{\geqslant ?}B)\big) = \infty \quad \text{iff} \quad \mathsf{Pr}_s^{\max}\big(\lozenge(C \wedge \lozenge B)\big) \geqslant p,$$

where C consists of all states t that are contained in a maximal end component $(T, \mathcal{A})$ with $\mathsf{rew}(t', \alpha) > 0$ for some state $t' \in T$ and an action $\alpha \in \mathcal{A}(t')$. Intuitively, when entering C one can stay in C until the accumulated reward is greater or equal than r, before entering B. Otherwise, we apply the same idea as before and compute the values $p_{s,r} = \mathsf{Pr}_s^{\max}(\lozenge^{\geqslant r}B)$ for increasing r until $p_{s,r} < p$. The values $p_{s,r}$ are obtained as the unique solution of the following LP with variables $x_{s,i}$[2] for $(s,i) \in \mathsf{S}[r]$ and the following constraints for $s \in \mathsf{S}$ and $1 \leqslant i \leqslant r$:

$$\begin{aligned}
x_{s,0} &= \mathsf{Pr}_s^{\max}\big(\lozenge B\big) \\
x_{s,i} &\geqslant 0 \\
x_{s,i} &\geqslant \sum_{t \in S} P(s,\alpha,t) \cdot x_{t,\ell} \qquad && \text{if } \alpha \in \mathsf{Act}(s) \text{ and } \ell = \max\{0, i - \mathsf{rew}(s,\alpha)\}
\end{aligned}$$

The objective is to minimise $\sum_{(s,i)\in\mathsf{S}[r]} x_{s,i}$. To speed up the computation, one can add the following constraints: $x_{s,i} = 1$ if $\mathsf{Pr}_s^{\max}\big(\lozenge(C \wedge \lozenge B)\big) = 1$ for $s \in S$.

The existential quantile

$$\mathrm{qu}_s\big(\exists\mathbb{P}_{<p}(\lozenge^{\geqslant ?}B)\big) = \min\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}_s^{\min}\big(\lozenge^{\geqslant r}B\big) < p\big\}$$

can then be computed by an analogous approach, using the fact that the values $p_{s,r} = \mathsf{Pr}_s^{\min}\big(\lozenge^{\geqslant r}B\big)$ are the greatest solutions in $[0,1]$ of the linear constraints

$$\begin{aligned}
x_{s,0} &= \mathsf{Pr}_s^{\min}\big(\lozenge B\big) \\
x_{s,i} &= 0 && \text{if } i \geqslant 1, \mathsf{Pr}_s^{\min}(\lozenge B) = 0 \text{ or } \mathsf{Pr}_s^{\min}(\lozenge \mathit{posRew}) = 0 \\
x_{s,i} &\leqslant \sum_{t \in S} P(s,\alpha,t) \cdot x_{t,\ell} && \text{if } i \geqslant 1, \mathsf{Pr}_s^{\min}(\lozenge B) > 0 \text{ and } \mathsf{Pr}_s^{\min}(\lozenge \mathit{posRew}) > 0, \\
& && \alpha \in \mathsf{Act}(s) \text{ and } \ell = \max\{0, i - \mathsf{rew}(s,\alpha)\}
\end{aligned}$$

where $\mathit{posRew} \subseteq \mathsf{S} \times \mathsf{Act}$ is the set of state-action pairs (s,α) with $\mathsf{rew}(s,\alpha) > 0$. Then, $\mathrm{qu}_s\big(\exists\mathbb{P}_{<p}(\lozenge^{\geqslant ?}B)\big) = \infty$ iff $\mathsf{Pr}_s^{\min}\big(\Box\lozenge B \wedge \Box\lozenge \mathit{posRew}\big) \geqslant p$ (see Lemma 3.4.2). Again, one could add the following constraints: $x_{s,i} = 1$ if $\mathsf{Pr}_s^{\min}(\Box\lozenge B \wedge \Box\lozenge \mathit{posRew}) = 1$ for $s \in S$.

3.5 Energy-utility quantiles

The aim of the previously presented methods is to provide a framework for the formal analysis of the trade-off between the consumed energy and the gained utility of an

---

[2] Intuitively, $x_{s,i}$ denotes the probability of reaching B from the state s such that a reward of i still needs to be accumulated. Therefore, the very first iteration $i = 0$ corresponds to the maximal unbounded reachability probability when there is no reward left that needs to be accumulated.

energy-critical system. Now, we want to see how those tools can be employed for this task.

Defining

$$\lambda_{e,u} = \Diamond\big((\text{energy} \leqslant e) \wedge (\text{utility} \geqslant u)\big),$$

for a fixed energy budget $e \in \mathbb{N}$ and a desired amount of utility $u \in \mathbb{N}$, allows to relate the energy consumption of the system with the utility that can be provided.

For an infinite path π, we have $\pi \models \lambda_{e,u}$ iff π has a finite prefix ρ with $\mathsf{rew}_e(\rho) \leqslant e$ and $\mathsf{rew}_u(\rho) \geqslant u$. Likewise, $\lambda_{e,u}$ can be interpreted as an instance of an until-property with an upper or a lower reward bound. For a fixed utility threshold u, the property $\lambda_{e,u} = \Diamond^{\leqslant e}(\text{utility} \geqslant u)$ corresponds to an increasing path property $\varphi[e]$, while $\psi[u] = \lambda_{e,u} = \Diamond^{\geqslant u}(\text{energy} \leqslant e)$ is decreasing for some fixed energy budget e. The task to compute the existential quantiles

$$\begin{aligned} \mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(\lambda_{?,u})\big) &= \min\big\{e \in \mathbb{N} : \mathsf{Pr}_s^{\max}(\lambda_{e,u}) > p\big\} \\ \mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(\lambda_{e,?})\big) &= \max\big\{u \in \mathbb{N} : \mathsf{Pr}_s^{\max}(\lambda_{e,u}) > p\big\} \end{aligned}$$

corresponds to the problem of constructing a scheduler that minimises the energy ensuring that the achieved utility is at least u with probability $> p$ or to maximise the achieved degree of utility for a given energy budget e. Analogously, universal quantiles can provide the corresponding information on the energy-utility characteristics in worst-case scenarios.

So, the energy-utility quantile $\mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(\lambda_{?,u})\big)$ allows to compute the minimal energy needed in order to obtain the desired utility u with probability p. This quantile can be computed using the same techniques as explained for reachability quantiles of the form $\mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(\Diamond^{\leqslant ?}B)\big)$ (see Section 3.3). For this purpose, the accumulated utility will be encoded into the state space of the model under consideration. Since the utility is bounded by u, one might use an automaton $\mathcal{U}_u$ with states $q_0, q_1, \ldots, q_{u-1}, q_u$ describing the accumulation of the utility value. The goal state q_u represents the fact that the system achieved an utility of at least u. The transitions of $\mathcal{U}_u$ are then given by $q_i \to q_j$ for $j \geqslant i$. Now, the representation $\mathcal{M}$ for the system and $\mathcal{U}_u$ can be put in parallel and a new **MDP** $\mathcal{M} \otimes \mathcal{U}_u$ with a single reward function for the energy and synchronous transitions that capture the meaning of $\mathcal{U}_u$'s states is obtained. Formally, $\mathcal{M} \otimes \mathcal{U}_u = (\mathsf{S} \times \{q_0, \ldots, q_u\}, \mathsf{Act}, P')$ where

$$P'(\langle s, q_i\rangle, \alpha, \langle t, q_j\rangle) = \begin{cases} P(s, \alpha, t) & \text{if } j = \min\{u, i + \mathsf{rew}_u(s, \alpha)\} \\ 0 & \text{else} \end{cases}$$

The reward structure of $\mathcal{M} \otimes \mathcal{U}_u$ consists of the energy reward function rew_e lifted to the product. That is, we deal with the reward function rew'_e for $\mathcal{M} \otimes \mathcal{U}_u$ given by $\mathsf{rew}'_e(\langle s, q_i\rangle, \alpha) = \mathsf{rew}_e(s, \alpha)$ for all $s \in \mathsf{S}$, $0 \leqslant i \leqslant u$ and $\alpha \in \mathsf{Act}$. By setting $B = \mathsf{S} \times \{q_u\}$, we then have

$$\mathsf{Pr}^{\max}_{\mathcal{M},s}\big(\Diamond((\text{energy} \leqslant e) \wedge (\text{utility} \geqslant u))\big) = \mathsf{Pr}^{\max}_{\mathcal{M}\otimes\mathcal{U}_u,\langle s,q_0\rangle}\big(\Diamond^{\leqslant e}B\big)$$

and therefore

$$\mathrm{qu}_s^{\mathcal{M}}(\exists\mathbb{P}_{>p}(\lambda_{?,u})) = \mathrm{qu}_{\langle s,q_0\rangle}^{\mathcal{M}\otimes\mathcal{U}_u}(\exists\mathbb{P}_{>p}(\Diamond^{\leqslant ?}B)).$$

The quantile $\mathrm{qu}_s(\exists\mathbb{P}_{>p}(\lambda_{e,?}))$ can be computed using an analogous automata-based approach. Here, the energy consumption of the system is encoded using an automaton $\mathcal{U}_e$, and the maximal accumulated utility can be obtained using the methods suggested for the computation of quantiles with lower reward bounds (see Section 3.4).

Various other energy-utility quantiles can be computed with the presented framework using reductions to the case of reward-bounded until formulas or derived path properties, such as

$$\max\big\{u \in \mathbb{N} \,:\, \mathsf{Pr}_s^{\max}\big((\text{utility} \geqslant u)\ \mathcal{R}\ (\text{energy} \leqslant e)\big) > p\big\}.$$

It is obvious that an analogous automata-based approach is applicable for quantiles where the objective is a probability constraint on path properties of the form $\Diamond((\mathsf{rew} \bowtie r) \wedge \kappa)$, where κ is a Boolean combination of constraints of the form $\mathsf{rew}_i \bowtie_i r_i$ for multiple reward functions $\mathsf{rew}_1, \ldots, \mathsf{rew}_k$ (other than rew).

3.6 Quantiles under side conditions

So far, we presented methods for computing quantiles for plain reward-bounded until path properties. We now address quantiles for reward-bounded until properties with ω-regular side conditions. That is, we consider path formulas of the form

$$\begin{aligned} \varphi[r] &= (A\,\mathcal{U}^{\leqslant r}\ B) \wedge \varphi \qquad\qquad \text{and} \\ \psi[r] &= (A\,\mathcal{U}^{\geqslant r}\ B) \wedge \varphi \end{aligned}$$

where φ is an ω-regular path property. Those side conditions allow to compute optimal quantile values (or quantile values for the worst case) while at the same time certain objectives should be preserved by the protocol under consideration. This allows to compute quantiles when for example some failure-preserving mechanisms can be guaranteed, e.g., some recovery procedure will be initiated as soon as some storage device crashes during its operation.

We will see that the task to compute probability quantiles for $\varphi[r]$ or $\psi[r]$ is reducible to the problem of computing plain until-quantiles (as presented previously in Section 3.3 and Section 3.4). For this purpose, we provide transformations $(\mathcal{M}, s) \mapsto (\widetilde{\mathcal{M}}, \widetilde{s})$ such that for each $r \in \mathbb{N}$ the maximal resp. minimal probabilities for $\varphi[r]$ (resp. $\psi[r]$) in $\mathcal{M}$ with starting state s agrees with the maximal resp. minimal probability for a plain reward-bounded until-property in $\widetilde{\mathcal{M}}$ with starting state $\widetilde{s}$. Thus, quantiles for $\varphi[r]$ (resp. $\psi[r]$) in $\mathcal{M}$ are computable by applying the methods of Section 3.3 (resp. Section 3.4) to the transformed $\widetilde{\mathcal{M}}$, and also the optimisations that will be presented in Section 5.1 are applicable for the computation as well.

The presented reduction is performed in two steps. In a first step, we rely on the standard approach for handling ω-regular path properties by representing the side condition φ by a deterministic automaton $\mathcal{A}$ that runs in parallel to $\mathcal{M}$, yielding a product **MDP** $\mathcal{M} \otimes \mathcal{A}$ (see Notation 10.128 in [BK08, page 881] on how to construct the product of a given **MDP** $\mathcal{M}$ and an automaton $\mathcal{A}$). This permits to replace φ with $\mathcal{A}$'s acceptance condition, lifted to the product $\mathcal{M} \otimes \mathcal{A}$. As usual, when considering the paths starting in a state s in $\mathcal{M}$, we have to consider the corresponding paths starting in the corresponding product state in $\mathcal{M} \otimes \mathcal{A}$ derived from the initial state of the automaton $\mathcal{A}$. It thus remains to consider the computation of quantiles for path formulas of the form

$$\begin{aligned} \varphi[r] &= (A\, \mathcal{U}^{\leqslant r}\, B) \wedge Acc \qquad\qquad \text{and} \\ \psi[r] &= (A\, \mathcal{U}^{\geqslant r}\, B) \wedge Acc \end{aligned}$$

in an **MDP** $\mathcal{M}$, where Acc is some ω-regular acceptance condition for paths of $\mathcal{M}$, such as a Rabin or Streett condition, with the requirement that Acc is prefix independent, i.e., that an infinite path π satisfies Acc iff all suffixes of π likewise satisfy Acc. Additionally, we have to be able to compute $\mathsf{Pr}^{\max}_{\mathcal{M},s}(Acc)$ and $\mathsf{Pr}^{\min}_{\mathcal{M},s}(Acc)$, which can be done in polynomial time for the standard acceptance condition types such as Rabin, Streett or parity.

To deal with side conditions Acc, we provide a second set of transformations for the four cases of upper and lower reward bounds and existential and universal quantification. Speaking roughly, for schedulers maximising or minimising the probabilities for $\varphi[r]$ resp. $\psi[r]$, the prefix independency of Acc allows to "postpone" the treatment of Acc until the event $A\, \mathcal{U}^{\bowtie r}\, B$ has occurred.

3.6.1 Upper reward bounds

We first address the task to compute probability quantiles for the path formulas $\varphi[r] = (A\, \mathcal{U}^{\leqslant r}\, B) \wedge Acc$ and a lower probability bound p and the induced quantiles:

$$\begin{aligned} \mathrm{qu}_{\mathcal{M},s}\big(\forall \mathbb{P}_{\trianglerighteq p}(\varphi[?])\big) &= \min\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\min}_{\mathcal{M},s}\big(\varphi[r]\big) \trianglerighteq p\big\} \\ \mathrm{qu}_{\mathcal{M},s}\big(\exists \mathbb{P}_{\trianglerighteq p}(\varphi[?])\big) &= \min\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\max}_{\mathcal{M},s}\big(\varphi[r]\big) \trianglerighteq p\big\} \end{aligned}$$

Such quantiles for upper reward-bounded until properties under ω-regular side conditions can be computed by a reduction to quantiles for pure upper reward-bounded until properties. For this purpose, we define the transformation $\mathcal{M} \rightsquigarrow \widetilde{\mathcal{M}}$ mapping $\mathcal{M}$ to a new **MDP** $\widetilde{\mathcal{M}} = \mathcal{M}^{up}_{\min}$ or $\widetilde{\mathcal{M}} = \mathcal{M}^{up}_{\max}$ arising from $\mathcal{M}$ by inserting two fresh trap states goal and fail both equipped with reward zero. The behaviour of the target states $s \in B$ in $\mathcal{M}$ will be purely probabilistically in $\widetilde{\mathcal{M}}$. That is, their enabled actions in Act get discarded and new probabilistic transitions to goal or fail via fresh action τ will be added. The probability to move to goal is given by the minimal or maximal probability for satisfying Acc from s, depending on whether we compute the universal or existential quantile. Formally:

$$\begin{aligned} \mathcal{M}^{up}_{\min} &= (\widetilde{\mathsf{S}}, \mathsf{Act} \cup \{\tau\}, P^{up}_{\min}, \widetilde{\mathsf{rew}}) \\ \mathcal{M}^{up}_{\max} &= (\widetilde{\mathsf{S}}, \mathsf{Act} \cup \{\tau\}, P^{up}_{\max}, \widetilde{\mathsf{rew}}) \end{aligned}$$

where $\widetilde{\mathsf{S}} = \mathsf{S} \cup \{\mathsf{goal}, \mathsf{fail}\}$ and τ is a fresh action symbol not contained in Act. The reward function in the transformed **MDP** is given by $\widetilde{\mathsf{rew}}(s, \mathsf{act}) = \mathsf{rew}(s, \mathsf{act})$ if $s \in \mathsf{S}$, $\mathsf{act} \in \mathsf{Act}(s)$ and $\widetilde{\mathsf{rew}}(\mathsf{goal}, \tau) = \widetilde{\mathsf{rew}}(\mathsf{fail}, \tau) = 0$. For $s \in \mathsf{S}$,

$$\begin{aligned} \mathsf{Act}_{\mathcal{M}^{up}_{\min}}(s) &= \mathsf{Act}_{\mathcal{M}^{up}_{\max}}(s) = \mathsf{Act}_{\mathcal{M}}(s) && \text{if } s \notin B \\ \mathsf{Act}_{\mathcal{M}^{up}_{\min}}(s) &= \mathsf{Act}_{\mathcal{M}^{up}_{\max}}(s) = \{\tau\} && \text{if } s \in B \end{aligned}$$

If $s \notin B$ then $P^{up}_{\min}(s, \mathsf{act}, t) = P^{up}_{\max}(s, \mathsf{act}, t) = P(s, \mathsf{act}, t)$ for all $\mathsf{act} \in \mathsf{Act}$ and $t \in \mathsf{S}$. For the states $s \in B$:

$$\begin{aligned} P^{up}_{\min}(s, \tau, \mathsf{goal}) &\overset{\text{def}}{=} \mathsf{Pr}^{\min}_{\mathcal{M},s}(Acc) \\ P^{up}_{\max}(s, \tau, \mathsf{goal}) &\overset{\text{def}}{=} \mathsf{Pr}^{\max}_{\mathcal{M},s}(Acc) \end{aligned}$$

and for $* \in \{\min, \max\}$:

$$P^{up}_{*}(s, \tau, \mathsf{fail}) = 1 - P^{up}_{*}(s, \tau, \mathsf{goal})$$

The states goal and fail are traps, e.g., we may deal with

$$\mathsf{Act}_{\mathcal{M}^{up}_{*}}(\mathsf{goal}) = \mathsf{Act}_{\mathcal{M}^{up}_{*}}(\mathsf{fail}) = \{\tau\}$$

and $P^{up}_{*}(\mathsf{goal}, \tau, \mathsf{goal}) = P^{up}_{*}(\mathsf{fail}, \tau, \mathsf{fail}) = 1$.

Intuitively, the constructed **MDP** $\mathcal{M}^{up}_{*}$ can simulate $\mathcal{M}$-paths satisfying $\varphi[r] = (A\ \mathcal{U}^{\leqslant r}\ B) \wedge Acc$ by paths satisfying $(A \cup B)\ \mathcal{U}^{\leqslant r}\ \mathsf{goal}$ and $\mathcal{M}$-paths satisfying $(A\ \mathcal{U}^{\leqslant r}\ B) \wedge \neg Acc$ by paths satisfying $(A \cup B)\ \mathcal{U}^{\leqslant r}\ \mathsf{fail}$, and vice versa. We use $(A \cup B)$ on the left side of the until operator in $\mathcal{M}^{up}_{*}$ to allow goal to be reached even if there are B-states that are not A-states. This is safe due to the fact that $A\, \mathcal{U}^{\leqslant r}\ B \equiv (A \cup B)\, \mathcal{U}^{\leqslant r}\ B$ (see Proposition 2.0.1), as the until formula is satisfied as soon as the first B-state is reached and reaching the goal-state in $\mathcal{M}^{up}_{*}$ requires passing through a B-state. The transition probabilities in $\mathcal{M}^{up}_{*}$ to move from a B-state to goal or fail reflect the objective in $\mathcal{M}$ to maximise or minimise the probability for $\varphi[r]$.

Lemma 3.6.1. *For all states s of $\mathcal{M}$ and $r \in \mathbb{N}$:*

(a) $\mathsf{Pr}^{\min}_{\mathcal{M},s}\big((A\, \mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) = \mathsf{Pr}^{\min}_{\mathcal{M}^{up}_{\min},s}\big((A \cup B)\, \mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$

(b) $\mathsf{Pr}^{\max}_{\mathcal{M},s}\big((A\, \mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) = \mathsf{Pr}^{\max}_{\mathcal{M}^{up}_{\max},s}\big((A \cup B)\, \mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$

Proof. To prove statement (a) we first show that the minimal probability for $(A\, \mathcal{U}^{\leqslant r}\ B) \wedge Acc$ in $(\mathcal{M}, s)$ is bounded from above by the minimal probability for $(A \cup B)\, \mathcal{U}^{\leqslant r}$ goal in $(\mathcal{M}^{up}_{\min}, s)$. For this, we pick a scheduler $\mathfrak{T}$ for $\mathcal{M}^{up}_{\min}$. Furthermore, let $\mathfrak{S}_{\min}$ be a scheduler for $\mathcal{M}$ such that

$$\mathsf{Pr}^{\mathfrak{S}_{\min}}_{\mathcal{M},s}(Acc) = \mathsf{Pr}^{\min}_{\mathcal{M},s}(Acc)$$

for all states s of $\mathcal{M}$. We now combine $\mathfrak{T}$ and $\mathfrak{S}_{\min}$ to construct a scheduler $\mathfrak{S}$ for $\mathcal{M}$ as follows. In its initial mode, $\mathfrak{S}$ simulates $\mathfrak{T}$, unless the last state of the generated path is a B-state, in which case $\mathfrak{S}$ switches to its second mode where it behaves as $\mathfrak{S}_{\min}$. Formally, this means:

- $\mathfrak{S}(\rho) = \mathfrak{T}(\rho)$ for all finite paths ρ in $\mathcal{M}$ that do not contain a B-state,
- $\mathfrak{S}(\rho) = \mathfrak{S}_{\min}(\rho_B)$ for all finite paths ρ in $\mathcal{M}$ containing at least one B-state and where ρ_B is the suffix of ρ starting in the first B-state of ρ.

In the following calculation, let $FP[A\,\mathcal{U}^{\leqslant r}\ t]$ denote the set of all finite $\mathfrak{S}$-paths $\rho = s_0\,\mathsf{act}_0\,s_1\,\mathsf{act}_1\,\ldots\,\mathsf{act}_{n-1}\,s_n$ in $\mathcal{M}$ with $s_0 = s$ and $\{s_0, s_1, \ldots, s_{n-1}\} \subseteq A\backslash B$ such that $s_n = t$ and $\mathsf{rew}(\rho) \leqslant r$. We write $\mathsf{Pr}(\rho)$ for the probability of ρ given by the product of the transition probabilities. Note that each $\rho \in FP[A\,\mathcal{U}^{\leqslant r}\ t]$ is also a $\mathfrak{T}$-path with the same probability. Furthermore, we write $\mathfrak{S}[\rho]$ for the scheduler "$\mathfrak{S}$ after ρ" which means $\mathfrak{S}[\rho](\rho') = \mathfrak{S}(\rho \circ \rho')$ if $last(\rho) = first(\rho')$ and where $\circ$ denotes the concatenation of finite paths. We then have:

$$\begin{aligned}
\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) &= \sum_{t\in B} \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(((A \wedge \neg B)\,\mathcal{U}^{\leqslant r}\ t) \wedge Acc\big)\\
&= \sum_{t\in B}\ \sum_{\rho\in FP[A\mathcal{U}^{\leqslant r}t]} \mathsf{Pr}(\rho)\cdot \mathsf{Pr}^{\mathfrak{S}[\rho]}_{\mathcal{M},t}\big(Acc\big)\\
&\overset{(*)}{=} \sum_{t\in B}\ \sum_{\rho\in FP[A\mathcal{U}^{\leqslant r}t]} \mathsf{Pr}(\rho)\cdot \mathsf{Pr}^{\min}_{\mathcal{M},t}\big(Acc\big)\\
&= \sum_{t\in B}\ \sum_{\rho\in FP[A\mathcal{U}^{\leqslant r}t]} \mathsf{Pr}(\rho)\cdot P^{up}_{\min}(t, \tau, \mathsf{goal})\\
&= \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{up}_{\min},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)
\end{aligned}$$

where $(*)$ holds because $\mathfrak{S}[\rho] = \mathfrak{S}_{\min}$ for each $\rho \in \bigcup_{t\in B} FP[A\,\mathcal{U}^{\leqslant r}\ t]$. Recall that $\mathfrak{S}$ mimics $\mathfrak{S}_{\min}$ after having visited a B-state. In summary, for every scheduler $\mathfrak{T}$ for $\mathcal{M}^{up}_{\min}$ we can construct a scheduler $\mathfrak{S}$ for $\mathcal{M}$ such that

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) = \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{up}_{\min},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

This yields:

$$\mathsf{Pr}^{\min}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) \leqslant \mathsf{Pr}^{\min}_{\mathcal{M}^{up}_{\min},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

Vice versa, given a scheduler $\mathfrak{S}$ for $\mathcal{M}$, we construct a scheduler $\mathfrak{T}$ for $\mathcal{M}^{up}_{\min}$ such that the probability for $(A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}$ under $\mathfrak{T}$ is less or equal than the probability for $(A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc$ under $\mathfrak{S}$ as follows. As long as no state $s \in B$ has been reached, scheduler $\mathfrak{T}$ for $\mathcal{M}^{up}_{\min}$ behaves as $\mathfrak{S}$. If the input path for $\mathfrak{T}$ ends in a state of B then $\mathfrak{T}$ has no choice and must schedule τ from this moment on. That is, $\mathfrak{T}(\rho) = \tau$ for all finite paths ρ in $\mathcal{M}^{up}_{\min}$ that contain a B-state. (Note that these paths either end in a B-state or in one of the trap states goal or fail.)

We can now use the same calculation as before, but replace the equality symbol in $(*)$ with $\geqslant$ as:

$$\mathsf{Pr}^{\mathfrak{S}[\rho]}_{\mathcal{M},t}\big(Acc\big) \geqslant \mathsf{Pr}^{\min}_{\mathcal{M},t}\big(Acc\big)$$

This yields that for every scheduler $\mathfrak{S}$ for $\mathcal{M}$ there is a scheduler $\mathfrak{T}$ for $\mathcal{M}^{up}_{\min}$ with

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{up}_{\min},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

As a consequence we obtain:

$$\mathsf{Pr}^{\min}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\min}_{\mathcal{M}^{up}_{\min},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

The proof of statement (b) is analogous. To show that the maximal probability for $(A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc$ in the pointed **MDP** $(\mathcal{M}, s)$ is greater or equal than the maximal probability for the event $((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal})$ in $(\mathcal{M}^{up}_{\max}, s)$, we pick a scheduler $\mathfrak{T}$ for $\mathcal{M}^{up}_{\max}$ and a scheduler $\mathfrak{S}_{\max}$ for $\mathcal{M}$ such that

$$\mathsf{Pr}^{\mathfrak{S}_{\max}}_{\mathcal{M},s}\big(Acc\big) = \mathsf{Pr}^{\max}_{\mathcal{M},s}\big(Acc\big)$$

for all states s of $\mathcal{M}$. We design a scheduler $\mathfrak{S}$ for $\mathcal{M}$ that operates in two modes. In its initial mode, scheduler $\mathfrak{S}$ simulates $\mathfrak{T}$, unless a B-state has been reached, in which case $\mathfrak{S}$ switches to its second mode and behaves as $\mathfrak{S}_{\max}$ from then on. Formally, the behaviour of $\mathfrak{S}$ for a given finite input path ρ in $\mathcal{M}$ is as follows:

- $\mathfrak{S}(\rho) = \mathfrak{T}(\rho)$ if ρ does not contain a B-state
- $\mathfrak{S}(\rho) = \mathfrak{S}_{\max}(\rho_B)$ if ρ contains at least one B-state and ρ_B is the longest suffix of ρ starting in a B-state

Using analogous calculations as in the proof of statement (a), we get:

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{up}_{\max},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

This shows:

$$\mathsf{Pr}^{\max}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\max}_{\mathcal{M}^{up}_{\max},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

It remains to prove that the maximal probability for $(A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc$ in the pointed **MDP** $(\mathcal{M}, s)$ is less or equal than the maximal probability for $(A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}$ in $(\mathcal{M}^{up}_{\max}, s)$. To see this, we take an arbitrary scheduler $\mathfrak{S}$ for $\mathcal{M}$. Let $\mathfrak{T}$ be the scheduler that behaves as $\mathfrak{S}$ as long as no B-state has been reached. For all input paths for $\mathfrak{T}$ that contain at least one B-state, $\mathfrak{T}$ schedules action τ. With analogous arguments as in the proof of statement (a), we obtain:

$$\mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{up}_{\max},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big) \geqslant \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big)$$

As a consequence, we get:

$$\mathsf{Pr}^{\max}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\leqslant r}\ B) \wedge Acc\big) \leqslant \mathsf{Pr}^{\max}_{\mathcal{M}^{up}_{\max},s}\big((A \cup B)\,\mathcal{U}^{\leqslant r}\ \mathsf{goal}\big)$$

This yields the claim. □

As a consequence of Lemma 3.6.1, the transformations $\mathcal{M} \rightsquigarrow \mathcal{M}^{up}_{\min}$ and $\mathcal{M} \rightsquigarrow \mathcal{M}^{up}_{\max}$ can be used to compute quantiles for upper reward-bounded until properties under side conditions by means of the linear programming techniques discussed in Section 3.3:

$$\begin{aligned}\mathrm{qu}_{\mathcal{M},s}\big(\forall\mathbb{P}_{\trianglerighteq p}((A\,\mathcal{U}^{\leqslant ?}\ B) \wedge Acc)\big) &= \mathrm{qu}_{\mathcal{M}^{up}_{\min},s}\big(\forall\mathbb{P}_{\trianglerighteq p}((A \cup B)\,\mathcal{U}^{\leqslant ?}\ \mathsf{goal})\big)\\ \mathrm{qu}_{\mathcal{M},s}\big(\exists\mathbb{P}_{\trianglerighteq p}((A\,\mathcal{U}^{\leqslant ?}\ B) \wedge Acc)\big) &= \mathrm{qu}_{\mathcal{M}^{up}_{\max},s}\big(\exists\mathbb{P}_{\trianglerighteq p}((A \cup B)\,\mathcal{U}^{\leqslant ?}\ \mathsf{goal})\big)\end{aligned}$$

3.6.2 Lower reward bounds

Let $\psi[r] = (A\,\mathcal{U}^{\geqslant r}\,B) \wedge Acc$ where Acc is an ω-regular acceptance condition on the paths of $\mathcal{M}$, as before. We now address the task to compute the quantiles:

$$\begin{aligned}
\mathrm{qu}_{\mathcal{M},s}\big(\exists\mathbb{P}_{\trianglerighteq p}(\psi[?])\big) &= \max\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\max}_{\mathcal{M},s}\big(\psi[r]\big) \trianglerighteq p\big\}\\
\mathrm{qu}_{\mathcal{M},s}\big(\forall\mathbb{P}_{\trianglerighteq p}(\psi[?])\big) &= \max\big\{r \in \mathbb{N} \,:\, \mathsf{Pr}^{\min}_{\mathcal{M},s}\big(\psi[r]\big) \trianglerighteq p\big\}
\end{aligned}$$

Again, we provide transformations $\mathcal{M} \rightsquigarrow \mathcal{M}^{lo}_{\min}$ and $\mathcal{M} \rightsquigarrow \mathcal{M}^{lo}_{\max}$ such that for each $r \in \mathbb{N}$ the minimal (resp. maximal) probability for $\psi[r]$ in $\mathcal{M}$ agrees with the minimal (resp. maximal) probability in $\mathcal{M}^{lo}_{\min}$ (resp. $\mathcal{M}^{lo}_{\max}$).

3.6.2.1 Maximal reachability probabilities

Let us first consider the task to compute the maximal probabilities for $\psi[r]$. The **MDP** $\mathcal{M}^{lo}_{\max}$ is obtained from $\mathcal{M}$ as follows. In each state of B, there is the choice between two fresh actions τ and ι. Similar to $\mathcal{M}^{up}_{\max}$, the action τ represents the choice of maximising Acc, going to states goal and fail with the corresponding probabilities. The action ι from a state $b \in B$ represents the choice of continuing normally, going with probability one to a fresh copy b_c, where the actions for b in $\mathcal{M}$ are enabled. Effectively, each B-state is split into two parts, with the first part allowing to choose τ and the second part allowing the original choices.

Formally, $\mathcal{M}^{lo}_{\max} = (\widetilde{\mathsf{S}}, \mathsf{Act} \cup \{\tau,\iota\}, P^{lo}_{\max}, \widetilde{\mathsf{rew}})$ arises from $\mathcal{M}$ by adding trap states goal and fail and fresh and pairwise distinct copies b_c for each state $b \in B$, i.e.,

$$\widetilde{\mathsf{S}} = \mathsf{S} \cup B_c \cup \{\mathsf{goal}, \mathsf{fail}\}$$

with $B_c = \{b_c : b \in B\}$. The action set of $\mathcal{M}^{lo}_{\max}$ is $\mathsf{Act} \cup \{\tau,\iota\}$ where:

$$\begin{aligned}
\mathsf{Act}_{\mathcal{M}^{lo}_{\max}}(s) &= \mathsf{Act}_{\mathcal{M}}(s) && \text{if } s \in \mathsf{S}\backslash B\\
\mathsf{Act}_{\mathcal{M}^{lo}_{\max}}(s) &= \big\{\tau,\iota\big\} && \text{if } s \in B\\
\mathsf{Act}_{\mathcal{M}^{lo}_{\max}}(s_c) &= \mathsf{Act}_{\mathcal{M}}(s) && \text{if } s \in B
\end{aligned}$$

The probabilistic effect of the actions $\mathsf{act} \in \mathsf{Act}$ in $\mathcal{M}^{lo}_{\max}$ is the same as in $\mathcal{M}$. More precisely, for $t \in \mathsf{S}$ and $\mathsf{act} \in \mathsf{Act}$:

$$\begin{aligned}
P^{lo}_{\max}(s, \mathsf{act}, t) &= P(s, \mathsf{act}, t) && \text{if } s \in \mathsf{S}\backslash B\\
P^{lo}_{\max}(s_c, \mathsf{act}, t) &= P(s, \mathsf{act}, t) && \text{if } s \in B
\end{aligned}$$

Furthermore, for $s \in B$:

$$\begin{aligned}
P^{lo}_{\max}(s, \tau, \mathsf{goal}) &= \mathsf{Pr}^{\max}_{\mathcal{M},s}\big(\,Acc\,\big)\\
P^{lo}_{\max}(s, \tau, \mathsf{fail}) &= 1 - P^{lo}_{\max}(s, \tau, \mathsf{goal})\\
P^{lo}_{\max}(s, \iota, s_c) &= 1
\end{aligned}$$

Furthermore, $P^{lo}_{\max}(\mathsf{goal},\tau,\mathsf{goal}) = P^{lo}_{\max}(\mathsf{fail},\tau,\mathsf{fail}) = 1$ and no other action is enabled in states goal and fail. The reward structure $\widetilde{\mathsf{rew}}$ for $\mathcal{M}^{lo}_{\max}$ is given by:

$$\begin{aligned}
\widetilde{\mathsf{rew}}(s,\mathsf{act}) &= \mathsf{rew}(s,\mathsf{act}) && \text{for } s \in \mathsf{S}\backslash B\\
\widetilde{\mathsf{rew}}(s_c,\mathsf{act}) &= \mathsf{rew}(s,\mathsf{act}) && \text{for } s \in B\\
\widetilde{\mathsf{rew}}(s,\tau) &= 0 && \text{for } s \in B \cup \{\mathsf{goal},\mathsf{fail}\}\\
\widetilde{\mathsf{rew}}(s,\iota) &= 0 && \text{for } s \in B
\end{aligned}$$

Each infinite path $\widetilde{\pi}$ in $\mathcal{M}^{lo}_{\max}$ where action τ is never scheduled induces a path $\widetilde{\pi}|_{\mathcal{M}}$ in $\mathcal{M}$ by dropping all occurrences of action ι and the B_c-states. Vice versa, whenever π is an infinite path in $\mathcal{M}$ then a corresponding path $\widetilde{\pi}$ in $\mathcal{M}^{lo}_{\max}$ is obtained by replacing each B-state s in π with $s\,\iota\,s_c$. Analogous transformations can be provided for finite paths in $\mathcal{M}^{lo}_{\max}$ and $\mathcal{M}$. For the transformation of a finite path ρ in $\mathcal{M}$ that ends in a B-state, we skip the copy at the end and suppose that $last(\widetilde{\rho}) = last(\rho) \in B$.

Intuitively, $\mathcal{M}^{lo}_{\max}$ can simulate $\mathcal{M}$-paths satisfying $\psi[r] = (A\ \mathcal{U}^{\geqslant r}\ B) \wedge Acc$ by paths satisfying $(A'\ \mathcal{U}^{\geqslant r}\ \mathsf{goal})$, and vice versa, where A' consists of the A-states, the B-states and those B_c-states b_c where $b \in A$. For an infinite path π in $\mathcal{M}$ that satisfies $(A\ \mathcal{U}^{\geqslant r}\ B) \wedge Acc$, the corresponding path $\widetilde{\pi}$ is obtained by choosing action τ and going to goal the first time B is visited with an accumulated reward $\geqslant r$. As all B-states visited on π strictly before that point have to be included in A (otherwise, $A\,\mathcal{U}^{\geqslant r}\ B$ would not hold), the corresponding B_c-states visited along $\widetilde{\pi}$ in $\mathcal{M}^{lo}_{\max}$ are included in A', as are all the B-states. Thus, $\widetilde{\pi}$ satisfies $(A'\ \mathcal{U}^{\geqslant r}\ \mathsf{goal})$. Vice versa, paths in $\mathcal{M}^{lo}_{\max}$ satisfying $(A'\,\mathcal{U}^{\geqslant r}\ \mathsf{goal})$ induce paths satisfying $(A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc$, with path fragments $\widetilde{\rho}$ in $\mathcal{M}^{lo}_{\max}$ satisfying $(A'\,\mathcal{U}^{\geqslant r}\ B)$ inducing path fragments $\widetilde{\rho}|_{\mathcal{M}}$ in $\mathcal{M}$ satisfying $(A\,\mathcal{U}^{\geqslant r}\ B)$.

Lemma 3.6.2. *For all states s of $\mathcal{M}$ and all $r \in \mathbb{N}$:*

$$\mathsf{Pr}^{\max}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) = \mathsf{Pr}^{\max}_{\mathcal{M}^{lo}_{\max},s}\big(A'\,\mathcal{U}^{\geqslant r}\ \mathsf{goal}\big)$$

where $A' = A \cup B \cup \{b_c \in B_c \,:\, b \in A\}$.

Proof. As for Lemma 3.6.1, the proof relies on a scheduler transformation.

Part 1. To prove that the maximal probability for $\psi[r] = (A\ \mathcal{U}^{\geqslant r}\ B) \wedge Acc$ is bounded from above by the maximal probability for $A'\,\mathcal{U}^{\geqslant r}\ \mathsf{goal}$ in $\mathcal{M}^{lo}_{\max}$, we pick an arbitrary scheduler $\mathfrak{S}$ for $\mathcal{M}$ and define a scheduler $\mathfrak{T}$ for $\mathcal{M}^{lo}_{\max}$ as follows. In its first mode, scheduler $\mathfrak{T}$ simulates $\mathfrak{S}$ by scheduling ι in the B-states, i.e., $\mathfrak{T}(\widetilde{\rho}) = \mathfrak{S}(\widetilde{\rho}|_{\mathcal{M}})$ if $last(\widetilde{\rho}) \in \widetilde{\mathsf{S}}\backslash B$ and $\mathfrak{T}(\widetilde{\rho}) = \iota$ if $last(\widetilde{\rho}) \in B$. As soon as a finite path $\widetilde{\rho}$ with

$$\widetilde{\rho} = \widetilde{s}_0\ \mathsf{act}_0\ \widetilde{s}_1\ \mathsf{act}_1\ \ldots\ \widetilde{s}_n \qquad \text{where } \widetilde{s}_0,\ldots,\widetilde{s}_{n-1} \in A', \widetilde{s}_n \in B \text{ and } \widetilde{\mathsf{rew}}(\widetilde{\rho}) \geqslant r$$

has been generated, $\mathfrak{T}$ switches mode and schedules τ from now on.

Obviously, whenever $\widetilde{\rho}$ is a finite $\mathfrak{T}$-path not ending in goal or fail, then by dropping the ι-actions and the states $s_c \in B_c$, we obtain an $\mathfrak{S}$-path in $\mathcal{M}$.

Let $FP[A\,\mathcal{U}^{\geqslant r}\,B]$ denote the set consisting of all finite $\mathfrak{S}$-paths ρ in $\mathcal{M}$ that have the following form:

$$\rho = s_0\,\mathsf{act}_0\,s_1\,\mathsf{act}_1\,\ldots\,s_n \qquad \text{where } s_n \in B, s_0,\ldots,s_{n-1} \in A \text{ and } \mathsf{rew}(\rho) \geqslant r$$

such that no proper prefix of ρ belongs to $FP[A\,\mathcal{U}^{\geqslant r}\,B]$, i.e., if $m < n$ and $s_m \in B$ then $\mathsf{rew}(s_0\,\mathsf{act}_0\,s_1\,\mathsf{act}_1\ldots\mathsf{act}_{m-1}\,s_m) < r$. Similarly, let $\widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,B]$ denote the set consisting of all finite $\mathfrak{T}$-paths $\widetilde{\rho}$ in $\mathcal{M}^{lo}_{\max}$ that have the following form:

$$\widetilde{\rho} = \widetilde{s}_0\,\mathsf{act}_0\,\widetilde{s}_1\,\mathsf{act}_1\,\ldots\,\widetilde{s}_n \qquad \text{where } \widetilde{s}_n \in B, \widetilde{s}_0,\ldots,\widetilde{s}_{n-1} \in A' \text{ and } \widetilde{\mathsf{rew}}(\widetilde{\rho}) \geqslant r$$

such that no proper prefix of $\widetilde{\rho}$ belongs to $\widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,B]$.

Note that each finite path $\rho \in FP[A\,\mathcal{U}^{\geqslant r}\,B]$ in $\mathcal{M}$ has a corresponding path $\widetilde{\rho} \in \widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,B]$ in $\mathcal{M}^{lo}_{\max}$ with $\widetilde{\rho}|_{\mathcal{M}} = \rho$ and vice versa. For $t \in B$, we define $FP[A\,\mathcal{U}^{\geqslant r}\,t]$ as the set of paths $\rho \in FP[A\,\mathcal{U}^{\geqslant r}\,B]$ with $last(\rho) = t$ and $\widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,t]$ as the set of paths $\widetilde{\rho} \in \widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,B]$ with $last(\widetilde{\rho}) = t$.

As in the proof of Lemma 3.6.1, we write $\mathsf{Pr}(\rho)$ for the probability of ρ given by the product of the transition probabilities. Note that $\mathsf{Pr}(\widetilde{\rho}) = \mathsf{Pr}(\widetilde{\rho}|_{\mathcal{M}})$ for $\widetilde{\rho} \in \widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,B]$, i.e., that the transformation of adding or dropping the ι actions and B_c-states does not change the probability. We then have:

$$\begin{aligned}
\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\,B) \wedge Acc\big) &= \sum_{t\in B}\ \sum_{\rho\in FP[A\mathcal{U}^{\geqslant r}t]} \mathsf{Pr}(\rho)\cdot \mathsf{Pr}^{\mathfrak{S}[\rho]}_{\mathcal{M},t}\big(Acc\big)\\
&\leqslant \sum_{t\in B}\ \sum_{\rho\in FP[A\mathcal{U}^{\geqslant r}t]} \mathsf{Pr}(\rho)\cdot \mathsf{Pr}^{\max}_{\mathcal{M},t}\big(Acc\big)\\
&= \sum_{t\in B}\ \sum_{\widetilde{\rho}\in\widetilde{FP}[A'\mathcal{U}^{\geqslant r}t]} \mathsf{Pr}(\widetilde{\rho})\cdot P^{lo}_{\max}(t,\tau,\mathsf{goal})\\
&\overset{(\dagger)}{=} \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\max},s}\big(A'\,\mathcal{U}^{\geqslant r}\,\mathsf{goal}\big)
\end{aligned}$$

Equation (†) in the above calculation holds because the set of infinite $\mathfrak{T}$-paths satisfying $A'\,\mathcal{U}^{\geqslant r}\,\mathsf{goal}$ consists of all infinite paths in $\mathcal{M}^{lo}_{\max}$ that have a prefix $\widetilde{\rho}$ in $\widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\,t]$ for some $t \in B$ and move from $last(\widetilde{\rho}) = t$ to goal (rather than fail) via action τ. It can be concluded:

$$\mathsf{Pr}^{\max}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\,B) \wedge Acc\big) \leqslant \mathsf{Pr}^{\max}_{\mathcal{M}^{lo}_{\max},s}\big(A'\,\mathcal{U}^{\geqslant r}\,\mathsf{goal}\big)$$

Part 2. For proving that the maximal probability for $(A\,\mathcal{U}^{\geqslant r}\,B) \wedge Acc$ in the pointed **MDP** $(\mathcal{M}, s)$ is greater or equal than the maximal probability for $A'\,\mathcal{U}^{\geqslant r}\,\mathsf{goal}$ in $(\mathcal{M}^{lo}_{\max}, s)$, we consider an arbitrary scheduler $\mathfrak{T}$ for $\mathcal{M}^{lo}_{\max}$. Let $\mathfrak{S}_{\max}$ be a scheduler for $\mathcal{M}$ maximising the probability for Acc from all states $s \in \mathsf{S}$. We now construct a scheduler $\mathfrak{S}$ for $\mathcal{M}$ as follows. In its initial mode, $\mathfrak{S}$ mimics $\mathfrak{T}$ by using $\mathfrak{T}$'s choice in the b_c copy for states $b \in B$, provided that $\mathfrak{T}$ does not select action τ in b. As soon as $\mathfrak{T}$ schedules action τ, scheduler $\mathfrak{S}$ switches its mode and simulates $\mathfrak{S}_{\max}$ from then on.

Let $\widetilde{FP}_\tau[A' \,\mathcal{U}^{\geqslant r}\, B]$ be the set of $\mathfrak{T}$-paths $\widetilde{\rho}$ in $\mathcal{M}^{lo}_{\max}$ of the form

$$\widetilde{\rho} = \widetilde{s}_0 \,\mathsf{act}_0\, \widetilde{s}_1 \,\mathsf{act}_1\, \ldots \,\mathsf{act}_{n-1}\, \widetilde{s}_n \qquad \text{with } \widetilde{s}_0, \ldots, \widetilde{s}_{n-1} \in A', \widetilde{s}_n \in B,$$

with $\mathsf{rew}(\widetilde{\rho}) \geqslant r$ and $\mathfrak{T}(\widetilde{\rho}) = \tau$, i.e., up to the B-state where $\mathfrak{T}$ schedules τ for the first time. Obviously, no path $\widetilde{\rho} \in \widetilde{FP}_\tau[A' \,\mathcal{U}^{\geqslant r}\, B]$ is a proper prefix of some other path in $\widetilde{FP}_\tau[A' \,\mathcal{U}^{\geqslant r}\, B]$. Similarily, let $FP_\tau[A \,\mathcal{U}^{\geqslant r}\, B]$ be the set of $\mathfrak{S}$-paths ρ in $\mathcal{M}$ with

$$\rho = s_0 \,\mathsf{act}_0\, s_1 \,\mathsf{act}_1\, \ldots \,\mathsf{act}_{n-1}\, s_n \qquad \text{with } s_0, \ldots, s_{n-1} \in A, s_n \in B,$$

and $\mathsf{rew}(\rho) \geqslant r$ and such that $\mathfrak{T}(\widetilde{\rho}) = \tau$, where $\widetilde{\rho}$ is the path in $\mathcal{M}^{lo}_{\max}$ corresponding to ρ (by adding the ι actions and B_c copies) and such that no proper prefix of ρ belongs to $FP_\tau[A \,\mathcal{U}^{\geqslant r}\, B]$.

Then:

$$\mathfrak{S}[\rho] = \mathfrak{S}_{\max} \text{ for all } \rho \in FP_\tau[A \,\mathcal{U}^{\geqslant r}\, B]$$

Additionally, we have a one-to-one correspondence between the paths $\rho \in FP_\tau[A \,\mathcal{U}^{\geqslant r}\, B]$ and $\widetilde{\rho} \in \widetilde{FP}_\tau[A' \,\mathcal{U}^{\geqslant r}\, B]$, satisfying $\widetilde{\rho}|_{\mathcal{M}} = \rho$, $\widetilde{\mathsf{rew}}(\widetilde{\rho}) = \mathsf{rew}(\rho)$ and $\mathsf{Pr}(\widetilde{\rho}) = \mathsf{Pr}(\rho)$. For $t \in B$, let $FP_\tau[A \,\mathcal{U}^{\geqslant r}\, t]$ be the set of finite paths $\rho \in FP_\tau[A \,\mathcal{U}^{\geqslant r}\, B]$ with $last(\rho) = t$. Likewise, let $\widetilde{FP}_\tau[A' \,\mathcal{U}^{\geqslant r}\, t]$ be the set of finite paths $\widetilde{\rho} \in \widetilde{FP}_\tau[A' \,\mathcal{U}^{\geqslant r}\, B]$ with $last(\widetilde{\rho}) = t$. We get:

$$\begin{aligned}
\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A \,\mathcal{U}^{\geqslant r}\, B) \wedge Acc\big) &= \sum_{t \in B} \sum_{\rho \in FP_\tau[A\mathcal{U}^{\geqslant r} t]} \mathsf{Pr}(\rho) \cdot \mathsf{Pr}^{\mathfrak{S}[\rho]}_{\mathcal{M},t}\big(Acc\big) \\
&= \sum_{t \in B} \sum_{\rho \in FP_\tau[A\mathcal{U}^{\geqslant r} t]} \mathsf{Pr}(\rho) \cdot \mathsf{Pr}^{\mathfrak{S}_{\max}}_{\mathcal{M},t}\big(Acc\big) \\
&= \sum_{t \in B} \sum_{\rho \in FP_\tau[A\mathcal{U}^{\geqslant r} t]} \mathsf{Pr}(\rho) \cdot \mathsf{Pr}^{\max}_{\mathcal{M},t}\big(Acc\big) \\
&= \sum_{t \in B} \sum_{\widetilde{\rho} \in \widetilde{FP}_\tau[A'\mathcal{U}^{\geqslant r} t]} \mathsf{Pr}(\widetilde{\rho}) \cdot P^{lo}_{\max}(t, \tau, \mathsf{goal}) \\
&= \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\max},s}\big(A' \,\mathcal{U}^{\geqslant r}\, \mathsf{goal}\big)
\end{aligned}$$

In summary, this yields for every scheduler $\mathfrak{T}$ for $\mathcal{M}^{lo}_{\max}$ a scheduler $\mathfrak{S}$ for $\mathcal{M}$ with

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A \,\mathcal{U}^{\geqslant r}\, B) \wedge Acc\big) = \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\max},s}\big(A' \,\mathcal{U}^{\geqslant r}\, \mathsf{goal}\big)$$

Hence:

$$\mathsf{Pr}^{\max}_{\mathcal{M},s}\big((A \,\mathcal{U}^{\geqslant r}\, B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\max}_{\mathcal{M}^{lo}_{\max},s}\big(A' \,\mathcal{U}^{\geqslant r}\, \mathsf{goal}\big)$$

This completes the proof of Lemma 3.6.2. □

3.6.2.2 Minimal reachability probabilities

Finally, we address the task of computing the minimal probabilities for $\psi[r]$. The **MDP** $\mathcal{M}^{lo}_{\min}$ is fairly the same as $\mathcal{M}^{lo}_{\max}$ except that we assign reward 1 to the self-loop in the goal state and that we deal with the minimal probability for Acc in $\mathcal{M}$ in the definition of the transition probabilities for the τ-transitions from the B-states to the goal state.

Formally, $\mathcal{M}^{lo}_{\min} = (\widetilde{\mathsf{S}}, \mathsf{Act} \cup \{\tau, \iota\}, P^{lo}_{\min}, \widetilde{\mathsf{rew}})$, where $\widetilde{\mathsf{S}}$ as well as the action set and the enabled actions are as for $\mathcal{M}^{lo}_{\max}$. Likewise, $P^{lo}_{\min}$ is as for $P^{lo}_{\max}$, except for the probabilities in the B-states, i.e., for $s \in B$,

$$\begin{aligned} P^{lo}_{\min}(s, \tau, \mathsf{goal}) &= \mathsf{Pr}^{\min}_{\mathcal{M},s}(Acc) \\ P^{lo}_{\min}(s, \tau, \mathsf{fail}) &= 1 - P^{lo}_{\min}(s, \tau, \mathsf{goal}) \\ P^{lo}_{\min}(s, \iota, s_c) &= 1 \end{aligned}$$

The reward structure for $\mathcal{M}^{lo}_{\min}$ is given by:

$$\begin{aligned} \widetilde{\mathsf{rew}}(s, \mathsf{act}) &= \mathsf{rew}(s, \mathsf{act}) && \text{for } s \in \mathsf{S} \backslash B \\ \widetilde{\mathsf{rew}}(s_c, \mathsf{act}) &= \mathsf{rew}(s, \mathsf{act}) && \text{for } s \in B \\ \widetilde{\mathsf{rew}}(s, \tau) &= 0 && \text{for } s \in B \cup \{\mathsf{fail}\} \\ \widetilde{\mathsf{rew}}(s, \iota) &= 0 && \text{for } s \in B \\ \widetilde{\mathsf{rew}}(s, \tau) &= 1 && \text{for } s = \mathsf{goal} \end{aligned}$$

As for $\mathcal{M}^{lo}_{\max}$ in the previous transformation, we can transform between paths in $\mathcal{M}$ and $\mathcal{M}^{lo}_{\min}$.

Intuitively, $\mathcal{M}^{lo}_{\min}$ can simulate $\mathcal{M}$-paths satisfying $\psi[r] = (A\, \mathcal{U}^{\geqslant r}\, B) \wedge Acc$ by paths satisfying $(A'\, \mathcal{U}^{\geqslant r}\, C)$, and vice versa, where A' consists of the A-states, the B-states, those B_c-states b_c where $b \in A$ and the goal state. The right hand side C of the until consists of all B_c-states and the goal state. In contrast to $\mathcal{M}^{lo}_{\max}$, the goal state is included on the left side of the until to ensure (in conjunction with the self-loop on goal with reward 1) that every finite path $\widetilde{\rho}$ that satisfies $(A'\, \mathcal{U}\, \mathsf{goal})$ will be extended to an infinite path that satisfies $(A'\, \mathcal{U}^{\geqslant r}\, \mathsf{goal})$, regardless of the concrete value of r. Without the positive reward loop for the goal state, a minimising scheduler could ensure by scheduling τ that every path fragment $\widetilde{\rho}$ consisting of A'-states, ending in a B-state b and having reward $\widetilde{\mathsf{rew}}(\widetilde{\rho}) < r$ is extended in such a way that $(A'\, \mathcal{U}^{\geqslant r}\, \mathsf{goal})$ (and thus also $(A'\, \mathcal{U}^{\geqslant r}\, C)$) is never satisfied. With the presented construction of $\mathcal{M}^{lo}_{\min}$, a minimising scheduler has the following choice for continuing these paths: If it chooses τ, $(A'\, \mathcal{U}^{\geqslant r}\, C)$ will be satisfied with $\mathsf{Pr}^{\min}_{\mathcal{M},b}(Acc)$. Essentially, choosing τ focuses on minimising the probability of $\psi[r] = (A\, \mathcal{U}^{\geqslant r}\, B) \wedge Acc$ by minimising the probability only of Acc, ignoring the possibility of minimising the probability for $(A\, \mathcal{U}^{\geqslant r}\, B)$. The choice ι in this situation postpones the minimisation of the probability of Acc to a later moment. For a path fragment consisting of A'-states, ending in a B-state b and having reward $\widetilde{\mathsf{rew}}(\widetilde{\rho}) \geqslant r$ that has not yet satisfied $(A'\, \mathcal{U}^{\geqslant r}\, C)$, the choice of τ becomes attractive in all cases, as choosing ι will surely lead to satisfaction of $(A'\, \mathcal{U}^{\geqslant r}\, C)$, as a B_c-state is reached in the next step.

Lemma 3.6.3. *For all states s of $\mathcal{M}$ and all $r \in \mathbb{N}$:*

$$\mathsf{Pr}^{\min}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) = \mathsf{Pr}^{\min}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

where $A' = A \cup B \cup \{b_c \in B_c\ :\ b \in A\} \cup \{\mathsf{goal}\}$ *and* $C = B_c \cup \{\mathsf{goal}\}$.

Proof. As before, we will provide scheduler transformations in both directions.

Part 1. We first show that the minimal probability for $(A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc$ in $\mathcal{M}$ is greater or equal than the minimal probability for $A'\ \mathcal{U}^{\geqslant r}\ C$ in $\mathcal{M}^{lo}_{\min}$. For this, we consider an arbitrary scheduler $\mathfrak{S}$ for $\mathcal{M}$. We define a scheduler $\mathfrak{T}$ for $\mathcal{M}^{lo}_{\min}$ as in Part 1 of the proof of Lemma 3.6.2, i.e., which first simulates $\mathfrak{S}$ and switches to schedule τ continuously as soon as a finite path

$$\widetilde{\rho} = \widetilde{s}_0\,\mathsf{act}_0\,\widetilde{s}_1\,\mathsf{act}_1\,\ldots\,\widetilde{s}_n \qquad \text{where } \widetilde{s}_0,\ldots,\widetilde{s}_{n-1} \in A', \widetilde{s}_n \in B \text{ and } \widetilde{\mathsf{rew}}(\widetilde{\rho}) \geqslant r,$$

has been generated.

With the above definition of $\mathfrak{T}$, there is no finite $\mathfrak{T}$-path of the form:

$$\widetilde{s}_0\,\mathsf{act}_0\,\widetilde{s}_1\,\mathsf{act}_1\,\ldots\,\widetilde{s}_n \qquad \text{where } \widetilde{s}_0,\ldots,\widetilde{s}_{n-1} \in A', \widetilde{s}_n \in B_c \text{ and } \widetilde{\mathsf{rew}}(\widetilde{\rho}) \geqslant r$$

Note that for such paths we would have $\widetilde{s}_{n-1} \in B$ and $\mathsf{act}_{n-1} = \iota$, which conflicts with $\mathfrak{T}(\widetilde{s}_0\,\mathsf{act}_0\,\widetilde{s}_1\,\mathsf{act}_1\,\ldots\,\mathsf{act}_{n-2}\,\widetilde{s}_{n-1}) = \tau$. This yields:

$$\mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ \mathsf{goal}\big) = \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ (\mathsf{goal} \vee B_c)\big) = \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

With a calculation as in the proof of Lemma 3.6.2 we get:

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ \mathsf{goal}\big)$$

Hence, we get:

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

As a consequence we obtain:

$$\mathsf{Pr}^{\min}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) \geqslant \mathsf{Pr}^{\min}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

Part 2. To prove that the minimal probability for $A'\ \mathcal{U}^{\geqslant r}\ C$ in $\mathcal{M}^{lo}_{\min}$ is greater or equal than the minimal probability for $(A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc$ in $\mathcal{M}$, we pick an arbitrary scheduler $\mathfrak{T}$ for $\mathcal{M}^{lo}_{\min}$. Let $\mathfrak{S}_{\min}$ be a scheduler for $\mathcal{M}$ that minimises the probabilities for Acc from all states. We now construct a scheduler $\mathfrak{S}$ for $\mathcal{M}$ as follows. In its initial mode, $\mathfrak{S}$ mimics $\mathfrak{T}$, provided that $\mathfrak{T}$ does not select action τ. As soon as $\mathfrak{T}$ schedules action τ, scheduler $\mathfrak{S}$ switches its mode and simulates $\mathfrak{S}_{\min}$ from then on. The goal is now to show that:

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) \leqslant \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

The set of $\mathfrak{T}$-paths in $\mathcal{M}^{lo}_{\min}$ that satisfy $A' \,\mathcal{U}^{\geqslant r}\, C$ can be partitioned into two sets, those that satisfy $A' \,\mathcal{U}^{\geqslant r}\, B_c$ and those that satisfy $A' \,\mathcal{U}^{\geqslant r}\, \mathsf{goal}$. As it can be the case that a path satisfies both $A' \,\mathcal{U}^{\geqslant r}\, B_c$ and $A' \,\mathcal{U}^{\geqslant r}\, \mathsf{goal}$, i.e., with a prefix that satisfies $A' \,\mathcal{U}^{\geqslant r}\, B_c$ before the τ-action is scheduled at a later point, we partition according to which of the two path formulas is satisfied first. Formally, let $\widetilde{FP}[A' \,\mathcal{U}^{\geqslant r}\, B_c]$ denote the set of finite $\mathfrak{T}$-paths of the form

$$\widetilde{\rho} = \widetilde{s}_0 \,\mathsf{act}_0\, \widetilde{s}_1 \,\mathsf{act}_1\, \ldots\, \widetilde{s}_n \qquad \text{where } \widetilde{s}_n \in B_c, \widetilde{s}_0, \ldots, \widetilde{s}_{n-1} \in A' \text{ and } \mathsf{rew}(\rho) \geqslant r$$

such that no proper prefix of $\widetilde{\rho}$ belongs to $\widetilde{FP}[A' \,\mathcal{U}^{\geqslant r}\, B_c]$, i.e., if $m < n$ and $s_m \in B_c$ then $\mathsf{rew}(\widetilde{s}_0 \,\mathsf{act}_0\, \widetilde{s}_1 \,\mathsf{act}_1\, \ldots\, \mathsf{act}_{m-1}\, \widetilde{s}_m) < r$. Let $\widetilde{FP}_\tau[A' \,\mathcal{U}\, B]$ be the set of $\mathfrak{T}$-paths $\widetilde{\rho}$ in $\mathcal{M}^{lo}_{\min}$ of the form

$$\widetilde{\rho} = \widetilde{s}_0 \,\mathsf{act}_0\, \widetilde{s}_1 \,\mathsf{act}_1\, \ldots\, \mathsf{act}_{n-1}\, \widetilde{s}_n \qquad \text{with } \widetilde{s}_0, \ldots, \widetilde{s}_{n-1} \in A', \widetilde{s}_n \in B,$$

with $\mathfrak{T}(\widetilde{\rho}) = \tau$, i.e., up to the B-state where $\mathfrak{T}$ schedules τ for the first time and which do not have a prefix in $\widetilde{FP}[A' \,\mathcal{U}^{\geqslant r}\, B_c]$. Obviously, no path $\widetilde{\rho} \in \widetilde{FP}_\tau[A' \,\mathcal{U}\, B]$ is a proper prefix of some other path in $\widetilde{FP}_\tau[A' \,\mathcal{U}\, B]$ and all infinite $\mathfrak{T}$-paths that satisfy $A' \,\mathcal{U}^{\geqslant r}\, \mathsf{goal}$ have a prefix in $\widetilde{FP}_\tau[A' \,\mathcal{U}\, B]$. We write $\widetilde{FP}_\tau[A' \,\mathcal{U}\, t]$ for $t \in B$ to denote the set of paths in $\widetilde{FP}_\tau[A' \,\mathcal{U}\, t]$ ending in t, with an equivalent notation for $\widetilde{FP}[A' \,\mathcal{U}^{\geqslant r}\, t]$. Then:

$$\begin{aligned}\mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big((A' \,\mathcal{U}^{\geqslant r}\, C)\big) = {} & \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(\widetilde{FP}[A' \,\mathcal{U}^{\geqslant r}\, B_c]\big) \\ & + \sum_{t \in B} \sum_{\widetilde{\rho} \in \widetilde{FP}_\tau[A' \mathcal{U} t]} \mathsf{Pr}(\widetilde{\rho}) \cdot P^{lo}_{\min}(t, \tau, \mathsf{goal})\end{aligned}$$

Let $FP[A \,\mathcal{U}^{\geqslant r}\, B]$ be the set of finite $\mathcal{M}$-paths ρ with $\widetilde{\rho}|_{\mathcal{M}} = \rho$ and $\widetilde{\rho} \in \widetilde{FP}[A' \,\mathcal{U}^{\geqslant r}\, B_c]$ and let $FP_\tau[A \,\mathcal{U}\, B]$ be the set of finite $\mathcal{M}$-paths ρ with $\widetilde{\rho}|_{\mathcal{M}} = \rho$ and $\widetilde{\rho} \in \widetilde{FP}_\tau[A' \,\mathcal{U}\, B]$. No path in $FP[A \,\mathcal{U}^{\geqslant r}\, B]$ has a prefix in $FP_\tau[A \,\mathcal{U}\, B]$ and vice-versa. Additionally, all paths in $FP[A \,\mathcal{U}^{\geqslant r}\, B]$ or $FP_\tau[A \,\mathcal{U}\, B]$ are $\mathfrak{S}$-paths.

In particular, all $\mathfrak{S}$-paths that satisfy $(A \,\mathcal{U}^{\geqslant r}\, B) \wedge Acc$ have a prefix in either $FP[A \,\mathcal{U}^{\geqslant r}\, B]$ or $FP_\tau[A \,\mathcal{U}\, B]$. However, not all infinite $\mathfrak{S}$-paths that have a prefix in $FP[A \,\mathcal{U}^{\geqslant r}\, B]$ satisfy $(A \,\mathcal{U}^{\geqslant r}\, B) \wedge Acc$, as Acc is not guaranteed to be satisfied. Likewise, not all infinite $\mathfrak{S}$-paths that satisfy Acc with a prefix in $FP_\tau[A \,\mathcal{U}\, B]$ satisfy

$(A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc$, as it is not guaranteed that $(A\,\mathcal{U}^{\geqslant r}\ B)$ holds. Thus, we have

$$\begin{aligned}
\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) &= \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(FP[A\,\mathcal{U}^{\geqslant r}\ B] \wedge Acc\big)\\
&\quad + \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(FP_{\tau}[A\,\mathcal{U}\ B] \wedge (A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big)\\
&\leqslant \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(FP[A\,\mathcal{U}^{\geqslant r}\ B]\big)\\
&\quad + \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(FP_{\tau}[A\,\mathcal{U}\ B] \wedge Acc\big)\\
&= \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(FP[A\,\mathcal{U}^{\geqslant r}\ B]\big)\\
&\quad + \sum_{t \in B}\ \sum_{\rho \in FP_{\tau}[A\mathcal{U}t]} \mathsf{Pr}(\rho) \cdot \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},t}(Acc)\\
&= \mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big(FP[A\,\mathcal{U}^{\geqslant r}\ B]\big)\\
&\quad + \sum_{t \in B}\ \sum_{\rho \in FP_{\tau}[A\mathcal{U}t]} \mathsf{Pr}(\rho) \cdot \mathsf{Pr}^{\mathfrak{S}_{\min}}_{\mathcal{M},t}(Acc)\\
&= \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(\widetilde{FP}[A'\,\mathcal{U}^{\geqslant r}\ B_c]\big)\\
&\quad + \sum_{t \in B}\ \sum_{\tilde{\rho} \in \widetilde{FP}_{\tau}[A'\mathcal{U}t]} \mathsf{Pr}(\tilde{\rho}) \cdot P^{lo}_{\min}(t, \tau, \mathsf{goal})\\
&= \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big((A'\,\mathcal{U}^{\geqslant r}\ C)\big)
\end{aligned}$$

Thus, for every scheduler $\mathfrak{T}$ in $\mathcal{M}^{lo}_{\min}$ we can construct a scheduler $\mathfrak{S}$ in $\mathcal{M}$ with

$$\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) \leqslant \mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

Hence, we get:

$$\mathsf{Pr}^{\min}_{\mathcal{M},s}\big((A\,\mathcal{U}^{\geqslant r}\ B) \wedge Acc\big) \leqslant \mathsf{Pr}^{\min}_{\mathcal{M}^{lo}_{\min},s}\big(A'\,\mathcal{U}^{\geqslant r}\ C\big)$$

This completes the proof of Lemma 3.6.3. ☐

As a consequence of Lemma 3.6.2 and Lemma 3.6.3 the transformations $\mathcal{M} \rightsquigarrow \mathcal{M}^{lo}_{\min}$ and $\mathcal{M} \rightsquigarrow \mathcal{M}^{lo}_{\max}$ permit to apply the methods presented in Section 3.4 for the computation of quantiles for lower reward-bounded until properties under side conditions:

$$\begin{aligned}
\mathrm{qu}_{\mathcal{M},s}\big(\exists\mathbb{P}_{\trianglerighteq p}((A\,\mathcal{U}^{\geqslant ?}\ B) \wedge Acc)\big) &= \mathrm{qu}_{\mathcal{M}^{lo}_{\max},s}\big(\exists\mathbb{P}_{\trianglerighteq p}(A'\,\mathcal{U}^{\geqslant ?}\ \mathsf{goal})\big)\\
\mathrm{qu}_{\mathcal{M},s}\big(\forall\mathbb{P}_{\trianglerighteq p}((A\,\mathcal{U}^{\geqslant ?}\ B) \wedge Acc)\big) &= \mathrm{qu}_{\mathcal{M}^{lo}_{\min},s}\big(\forall\mathbb{P}_{\trianglerighteq p}(A'\,\mathcal{U}^{\geqslant ?}\ C\big)
\end{aligned}$$

where A' and C are defined as in Lemma 3.6.2 and Lemma 3.6.3, respectively. Please note that the definition of A' slightly differs in both cases.

3.7 Reachability quantiles and continuous time

Quantiles for continuous-time Markov chains can be defined in a similar way as for discrete-time Markov chains (see Section 3.1). However, in the case of **CTMC**s we have to consider trajectories rather than paths, in which case the quantile can be a real number (rather than an integer) and min and max in the definitions need to be replaced with inf and sup, respectively. As an example, the quantile

$$\inf\big\{t \in \mathbb{R} : \mathsf{Pr}_s\big(\Diamond^{\leqslant t}\Phi\big) \geqslant p\big\}$$

asks for the "smallest" time-bound t such that the probability of reaching states statisfying Φ within the given time is at least p starting from some fixed state s.

The theoretical basis for the computation of reachability quantiles in **CTMC**s relies on an approximation scheme that was already proposed in [Bai+14b][3]. The presented scheme consists of a combination of an *exponential search* with a succeeding *binary search* in order to find the demanded quantile value. As a very first step, it is checked that the quantile is finite using a precomputation similar to the calculations required for reachability quantiles in **DTMC**s or **MDP**s (see Section 3.3.1 or Section 3.4.1). Then, the mentioned *exponential search* is used to determine the smallest $i \in \mathbb{N}$ such that $\mathsf{Pr}_s\big(\Diamond^{\leqslant 2^i}\Phi\big) \geqslant p$ holds for the first time. The existence of such an i can be guaranteed due to the already completed precomputation. If $i \geqslant 1$, then a *binary search* is performed in the interval $[2^{i-1}, 2^i]$ to find a $t \in [2^{i-1}, 2^i]$ such that $\mathsf{Pr}_s\big(\Diamond^{\leqslant t-\frac{\varepsilon}{2}}\Phi\big) < p$ and $\mathsf{Pr}_s\big(\Diamond^{\leqslant t+\frac{\varepsilon}{2}}\Phi\big) \geqslant p$ holds for a user-defined accuracy of $\varepsilon > 0$. Due to the preceding precomputation such a t exists, and it corresponds to an ε-approximation of the demanded quantile $\inf\big\{t \in \mathbb{R} : \mathsf{Pr}_s\big(\Diamond^{\leqslant t}\Phi\big) \geqslant p\big\}$. If it is the case that the exponential search aborts immediately for $i = 0$, the binary search will be performed in the interval $[0, 1]$.

3.7.1 Dualities

When handling continuous time the only considered models are **CTMC**s. Since a **CTMC** does not involve any nondeterminism, there is no need to distinguish between existential or universal quantiles.

So, the following dualities can be established when considering increasing state properties:

$$\begin{aligned}
\mathrm{qu}_s\big(\mathbb{P}_{>p}(A\ \mathcal{U}^{\leqslant ?}\ B)\big) &= \mathrm{qu}_s\big(\mathbb{P}_{<1-p}(\,(\neg A)\ \mathcal{R}^{\leqslant ?}\ (\neg B\,)\,)\big)\\
&= \mathrm{qu}_s\big(\mathbb{P}_{\geqslant 1-p}(\,(\neg A)\ \mathcal{R}^{\leqslant ?}\ (\neg B\,)\,)\big)\\
&= \mathrm{qu}_s\big(\mathbb{P}_{\leqslant p}(A\ \mathcal{U}^{\leqslant ?}\ B)\big)
\end{aligned}$$

$$\begin{aligned}
\mathrm{qu}_s\big(\mathbb{P}_{>p}(A\ \mathcal{R}^{\geqslant ?}\ B)\big) &= \mathrm{qu}_s\big(\mathbb{P}_{<1-p}(\,(\neg A)\ \mathcal{U}^{\geqslant ?}\ (\neg B\,)\,)\big)\\
&= \mathrm{qu}_s\big(\mathbb{P}_{\geqslant 1-p}(\,(\neg A)\ \mathcal{U}^{\geqslant ?}\ (\neg B\,)\,)\big)\\
&= \mathrm{qu}_s\big(\mathbb{P}_{\leqslant p}(A\ \mathcal{R}^{\geqslant ?}\ B)\big)
\end{aligned}$$

---

[3] The group of Monika Heiner planned to integrate an adapted form of this scheme into a development version of the tool MARCIE [HRS13] in order to support the computation of quantiles in the context of stochastic Petri nets.

For the case of decreasing state properties the following dualities can be introduced:

$$\begin{aligned}\mathrm{qu}_s\big(\mathbb{P}_{>p}(A\ \mathcal{U}^{\geqslant ?}\ B)\big) &= \mathrm{qu}_s\big(\mathbb{P}_{<1-p}(\,(\neg A)\ \mathcal{R}^{\geqslant ?}\ (\neg B\,)\,)\big)\\ &= \mathrm{qu}_s\big(\mathbb{P}_{\geqslant 1-p}(\,(\neg A)\ \mathcal{R}^{\geqslant ?}\ (\neg B\,)\,)\big)\\ &= \mathrm{qu}_s\big(\mathbb{P}_{\leqslant p}(A\ \mathcal{U}^{\geqslant ?}\ B)\big)\end{aligned}$$

$$\begin{aligned}\mathrm{qu}_s\big(\mathbb{P}_{>p}(A\ \mathcal{R}^{\leqslant ?}\ B)\big) &= \mathrm{qu}_s\big(\mathbb{P}_{<1-p}(\,(\neg A)\ \mathcal{U}^{\leqslant ?}\ (\neg B\,)\,)\big)\\ &= \mathrm{qu}_s\big(\mathbb{P}_{\geqslant 1-p}(\,(\neg A)\ \mathcal{U}^{\leqslant ?}\ (\neg B\,)\,)\big)\\ &= \mathrm{qu}_s\big(\mathbb{P}_{\leqslant p}(A\ \mathcal{R}^{\leqslant ?}\ B)\big)\end{aligned}$$

4 Expectation Quantiles

To investigate the interplay of two reward functions (such as one for the consumed energy and one for the achieved utility) we address here path formulas where instead of sets $A, B \subseteq \mathsf{S}$ (as done for reachability quantiles, see Chapter 3), constraints for some other reward function are imposed. For instance, given two reward functions $\mathsf{rew}_e : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ (for the energy) and $\mathsf{rew}_u : \mathsf{S} \times \mathsf{Act} \to \mathbb{N}$ (for the utility), the energy-utility trade-off (as introduced in Section 3.5)

$$\lambda_{e,u} = \Diamond\big((\text{energy} \leqslant e) \wedge (\text{utility} \geqslant u)\big)$$

directly relates the accumulation of the reward function for the energy with the accumulation of the reward function for the gained utility.

So, the objective is the minimal or maximal expected value of a random variable $f[r] : \mathit{InfPaths} \to \mathbb{N} \cup \{\infty\}$. For instance, if $f[r]$ is increasing in r and θ stands for some rational threshold, then an expectation quantile can be defined as the least $r \in \mathbb{N}$ such that the expected value of $f[r]$ is larger than θ for all or some scheduler(s). As an example for quantiles with expectation objectives, we consider a Boolean condition *cond* for finite paths and the random variable $f[e] = \text{utility}|_{\mathit{cond}} : \mathit{InfPaths} \to \mathbb{N} \cup \{\infty\}$ that returns the utility value that is earned along finite paths where *cond* holds. Formally:

$$\text{utility}|_{\mathit{cond}}(\pi) = \sup\big\{\mathsf{rew}_u\big(\mathit{pref}(\pi, k)\big) \;:\; k \in \mathbb{N}, \mathit{pref}(\pi, k) \models \mathit{cond}\big\}$$

That is, if π is an infinite path with $\pi \models \Diamond \mathit{cond}$ (i.e., $\mathit{pref}(\pi, k) \models \mathit{cond}$ for some $k \in \mathbb{N}$) then $\text{utility}|_{\mathit{cond}}(\pi) = \mathsf{rew}_u(\rho)$, where ρ is the longest prefix of π with $\rho \models \mathit{cond}$. If $\pi \models \Box \mathit{cond}$ (i.e., $\mathit{pref}(\pi, k) \models \mathit{cond}$ for all $k \in \mathbb{N}$) then $\text{utility}|_{\mathit{cond}}(\pi)$ can be finite or infinite, depending on whether there are infinitely many positions i with $\mathsf{rew}_u(s_i, \alpha_i) > 0$. Given a scheduler $\mathfrak{S}$ and a state s in $\mathcal{M}$, the *expected utility* for condition *cond* is the expected value of the random variable $\text{utility}|_{\mathit{cond}}$ under the probability measure induced by $\mathfrak{S}$ and s:

$$\text{ExpUtil}_s^{\mathfrak{S}}\big(\mathit{cond}\big) = \sum_{r \in \mathbb{N}} r \cdot \mathsf{Pr}_s^{\mathfrak{S}}\big\{\pi \in \mathit{InfPaths} \;:\; \text{utility}|_{\mathit{cond}}(\pi) = r\big\}$$

Note that $\text{ExpUtil}_s^{\mathfrak{S}}\big(\mathit{cond}\big) = \infty$ is possible if $\mathsf{Pr}_s^{\mathfrak{S}}\big(\Diamond\Box(\mathit{cond})\big) > 0$. We define

$$\begin{aligned} \text{ExpUtil}_s^{\max}\big(\mathit{cond}\big) &= \sup_{\mathfrak{S}} \text{ExpUtil}_s^{\mathfrak{S}}\big(\mathit{cond}\big) \text{ and} \\ \text{ExpUtil}_s^{\min}\big(\mathit{cond}\big) &= \inf_{\mathfrak{S}} \text{ExpUtil}_s^{\mathfrak{S}}\big(\mathit{cond}\big). \end{aligned}$$

Expectation energy-utility quantiles can be formalised by dealing with conditions $\mathit{cond}[e]$ that are parameterised by some energy value $e \in \mathbb{N}$. Examples are the following quantiles that fix a lower bound u for the extremal expected degree of utility and ask to minimise the required energy:

$$\begin{aligned}\mathrm{qu}_s\big(\exists\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?)\big) &= \min\big\{e \in \mathbb{N} : \mathrm{ExpUtil}_s^{\max}\big(\mathrm{energy} \leqslant e\big) > u\big\}\\ \mathrm{qu}_s\big(\forall\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?)\big) &= \min\big\{e \in \mathbb{N} : \mathrm{ExpUtil}_s^{\min}\big(\mathrm{energy} \leqslant e\big) > u\big\}\end{aligned}$$

where a path $\rho \models (\mathrm{energy} \leqslant e)$ if and only if $\mathsf{rew}_e(\rho) \leqslant e$. We now want to discuss how to compute expectation quantiles in **MDP**s with the two reward functions rew_e and rew_u. The approach will be shown computing

$$\begin{aligned}E_s^{\exists} &= \mathrm{qu}_s\big(\exists\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?)\big) \text{ and}\\ E_s^{\forall} &= \mathrm{qu}_s\big(\forall\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?)\big)\end{aligned}$$

when the utility-bound u has been fixed. Using known results for standard **MDPs**, it can be obtained that $\mathrm{ExpUtil}_s^{\max}(\mathrm{energy} \leqslant e)$ is finite, provided that $\mathsf{Pr}_s^{\min}(\Diamond(\mathrm{energy} > e)) = 1$. If, however, $\mathcal{M}$ contains end components where all the state-action pairs have zero energy reward then $\mathsf{Pr}_s^{\min}(\Diamond(\mathrm{energy} > e)) < 1$ and $\mathrm{ExpUtil}_s^{\max}(\mathrm{energy} \leqslant e) = \infty$ is possible. Therefore, a precomputation is necessary as already used for the computation of reachability quantiles (see Chapter 3).

4.1 Computation scheme

In order to introduce the computation of expectation quantiles we start by restricting ourselves to models where the demanded expectation quantile turns out to be finite and therefore the presented computation will terminate. This renders a precomputation unnecessary for those specific models, and allows to concentrate on the required steps for computing the quantile values.

So, let us make the simplifying assumption that all end components are both energy- and utility-divergent, i.e., whenever $(T, \mathcal{A})$ is an end component of $\mathcal{M}$ then there exist state-action pairs (t, α) and (v, β) with $t, v \in T$ and $\alpha \in \mathcal{A}(t)$, $\beta \in \mathcal{A}(v)$ such that $\mathsf{rew}_e(t, \alpha)$ and $\mathsf{rew}_u(v, \beta)$ are positive. This assumption yields that $\mathsf{Pr}_s^{\min}(\Diamond(\mathrm{energy} > e)) = 1$ and hence, $\mathrm{ExpUtil}_s^{\max}(\mathrm{energy} \leqslant e)$ and $\mathrm{ExpUtil}_s^{\min}(\mathrm{energy} \leqslant e)$ are finite for all states $s \in \mathsf{S}$ and arbitrary energy bounds $e \in \mathbb{N}$. Moreover, for each scheduler $\mathfrak{S}$ we have $\lim_{e\to\infty} \mathrm{ExpUtil}_s^{\mathfrak{S}}(\mathrm{energy} \leqslant e) = \infty$. This yields the finiteness of the expectation quantiles $E_s^{\exists}$ and $E_s^{\forall}$. The computation of $E_s^{\exists}$ and $E_s^{\forall}$ can be carried out using an iterative approach as it was done for reachability quantiles.

For $E_s^{\forall}$, we compute iteratively the values $u_{s,e} = \mathrm{ExpUtil}_s^{\min}(\mathrm{energy} \leqslant e)$ until $u_{s,e} > u$ for the first time, in which case $E_s^{\forall} = e$. It remains to explain how to compute $u_{s,e}$. We can use an LP-based approach and characterise the vector $(u_{s,i})_{(s,i)\in\mathsf{S}[e]}$ as the unique solution of the LP with variables $x_{s,i}$ for $(s, i) \in \mathsf{S}[e]$ and the objective to maximise the sum of the $x_{s,i}$'s (for $i \leqslant e$) with subject to:

$$\begin{aligned}x_{s,i} &\leqslant \mathsf{rew}_u(s,\alpha) + \sum_{t\in\mathsf{S}} P(s,\alpha,t)\cdot x_{t,i-\mathsf{rew}_e(s,\alpha)} && \text{if } \mathsf{rew}_e(s,\alpha) \leqslant i \text{ for } \alpha \in \mathsf{Act}(s)\\ x_{s,i} &\leqslant 0 && \text{if } \mathsf{rew}_e(s,\alpha) > i \text{ for } \alpha \in \mathsf{Act}(s)\end{aligned}$$

For computing $E^{\exists}_s$, the values $v_{s,e} = \mathrm{ExpUtil}^{\max}_s(\text{energy} \leqslant e)$ can be computed by a similar schema. Here, the objective is to minimise the sum of the variables $x_{s,i}$ with subject to:

$$x_{s,i} \geqslant \mathsf{rew}_u(s,\alpha) + \sum_{t \in \mathsf{S}} P(s,\alpha,t) \cdot x_{t,i-\mathsf{rew}_e(s,\alpha)}$$

if $\alpha \in \mathsf{Act}(s)$ and $\mathsf{rew}_e(s,\alpha) \leqslant i \leqslant e$.

Obviously, the linear programs which need to be solved are quite related to the linear programs we already solved for the computation of reachability quantiles (see Section 3.3.2). Therefore, most of the techniques for computing reachability quantiles are applicable for the computation as well. We will take advantage of this observation in order to improve the performance of the implemented algorithms (see Chapter 5 and especially Section 5.1 for more details).

4.2 Arbitrary models

Now, we turn to the computation of expectation quantiles for the general case, where no assumptions on the end components are imposed. That is, there might exist end components that contain no state-action pair with positive energy or utility reward. Basically, this computation relies on an analogous LP approach, but requires a preprocessing step to identify the states where $\mathrm{ExpUtil}^{\max}_s(\text{energy} \leqslant e) = \infty$, respectively $\mathrm{ExpUtil}^{\min}_s(\text{energy} \leqslant e) = \infty$ and computing those states where the quantile is infinite. Essentially, we can use the same linear program as sketched in Section 4.1, restricted to those variables $x_{s,e}$ where $\mathrm{ExpUtil}^{\max}_s(\text{energy} \leqslant e) < \infty$ resp. $\mathrm{ExpUtil}^{\min}_s(\text{energy} \leqslant e) < \infty$. Furthermore, we have to identify those states s where the quantile is infinite. The main feature for this preprocessing is an analysis of end components, similar as in [Alf99; For+11a].

Zero-reward and reward-divergent end components An end component $(T, \mathcal{A})$ of $\mathcal{M}$ with $\mathsf{rew}_e(t,\alpha) = 0$ for all states $t \in T$ and actions $\alpha \in \mathcal{A}(t)$ is called a *zero-energy end component.* $(T, \mathcal{A})$ is called *energy-divergent* if there is some state-action pair (t,α) with $t \in T$, $\alpha \in \mathcal{A}(t)$ and $\mathsf{rew}_e(t,\alpha) > 0$. *ZE* denotes the set of all states $t \in \mathsf{S}$ that are contained in some zero-energy end component, and *ED* stands for the set of all states t that belong to some energy-divergent end component.

Similar notations are used for the reward function rew_u characterising the utility. End components $(T, \mathcal{A})$ with $\mathsf{rew}_u(t,\alpha) > 0$ for some state-action pair (t,α) where $t \in T$ and $\alpha \in \mathcal{A}(t)$ are said to be *utility-divergent.* End components that are not utility-divergent are called *zero-utility end components.* *UD* denotes the set of states that are contained in some utility-divergent end component, while *ZU* stands for the set of states that are contained in some zero-utility end component.

The sets *UD* and *ED* are obtained using standard algorithms for computing maximal end components, see, e.g., [CY95] and [Alf98; BK08]. Note that $t \in \mathit{UD}$ if and only if there exists a maximal end component of $\mathcal{M}$ that contains t and that is utility-divergent.

The analogous statement holds for *ED*. This, however, does not hold for *ZE* and *ZU* since zero-energy end components might be contained in energy-divergent maximal end components. Nevertheless, the computation of *ZE* and *ZU* can also be carried out using algorithms to determine maximal end components of the sub-**MDP**s $\mathcal{M}|_{\mathsf{rew}_e=0}$ and $\mathcal{M}|_{\mathsf{rew}_u=0}$, respectively. For instance, $\mathcal{M}|_{\mathsf{rew}_e=0}$ results from $\mathcal{M}$ by successively removing actions and states. We start dropping all actions α with $\mathsf{rew}_e(s, \alpha) > 0$ from $\mathsf{Act}(s)$. Afterwards, all states s with $\mathsf{Act}(s) = \varnothing$ are removed, and all actions β with $P(s', \beta, s) > 0$ get removed from $\mathsf{Act}(s')$ as well (i.e., we remove the actions where some successors have been removed), and so on. This process carries on until all relevant states and actions have been finally removed, and it needs to terminate since there are only finitely many states and actions available. The previous considerations directly lead to the following statement:

Lemma 4.2.1. *The sets ZE, ED, ZU and UD can be computed in time quadratic in the size of the underlying graph of* $\mathcal{M}$*.*

Expected accumulated utility for fixed energy budget For each infinite path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\,\ldots$ and fixed energy budget $e \in \mathbb{N}$, $\text{utility}|_{\text{energy} \leqslant e}(\pi) = \infty$ if and only if for each $u \in \mathbb{N}$ there exists a finite prefix ρ of π such that $\text{energy}(\rho) \leqslant e$ and $\text{utility}(\rho) > u$. As the energy budget e is fixed, this is only possible if there exists some $k \in \mathbb{N}$ such that $\mathsf{rew}_e(s_i, \alpha_i) = 0$ for all $i \geqslant k$ and $\mathsf{rew}_u(s_i, \alpha_i) > 0$ for infinitely many indices i. Hence, if $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\,\ldots$ then $\text{utility}|_{\text{energy} \leqslant e}(\pi) = \infty$ if and only if there exists $k \in \mathbb{N}$ such that

(1) $\text{energy}(\mathit{pref}(\pi, k)) \leqslant e$

(2) $\forall j \geqslant k.\mathsf{rew}_e(s_j, \alpha_j) = 0$

(3) $\overset{\infty}{\exists}\, j \in \mathbb{N}.\mathsf{rew}_u(s_j, \alpha_j) > 0$

Let *ZEUD* denote the set of all states $t \in \mathsf{S}$ that are contained in some zero-energy utility-divergent end component.

Lemma 4.2.2. *For all states* s *in* $\mathcal{M}$ *and all* $e \in \mathbb{N}$*:*

(a) $\text{ExpUtil}^{\max}_s(\text{energy} \leqslant e) < \infty$ *iff* $s \not\models \exists\Diamond((\text{energy} \leqslant e) \wedge \mathit{ZEUD}))$

(b) $\text{ExpUtil}^{\min}_s(\text{energy} \leqslant e) < \infty$ *iff* $\mathsf{Pr}^{\min}_s(\varphi) = 0$

where φ *denotes the path property given by* $\pi \models \varphi$ *iff there exists some* $k \in \mathbb{N}$ *such that conditions (1), (2) and (3) hold.*

Proof. To prove part (a), we first suppose $s \models \exists\Diamond((\text{energy} \leqslant e) \wedge \mathit{ZEUD}))$ and show that there is a scheduler $\mathfrak{S}$ with $\text{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\text{energy} \leqslant e) = \infty$. Let $\mathfrak{T}$ be any finite-memory scheduler with $\mathsf{Pr}^{\mathfrak{T}}_t(\Box \mathit{ZEUD}) = 1$ for all states $t \in \mathit{ZEUD}$ and such that the limit of almost all $\mathfrak{T}$-paths that start in some state $t \in \mathit{ZEUD}$ is a zero-energy utility-divergent end component. We may pick any scheduler $\mathfrak{S}$ that generates a finite path ρ with $\mathrm{energy}(\rho) \leqslant e$ and $last(\rho) \in ZEUD$ and that behaves as $\mathfrak{T}$ as soon as a state in $ZEUD$ has been reached. Then, $\mathrm{utility}|_{\mathrm{energy} \leqslant e}(\pi) = \infty$ for almost all infinite $\mathfrak{S}$-paths π in the cylinder set of ρ. This yields:

$$\mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\mathrm{energy} \leqslant e) = \infty$$

To prove the second part of statement (a), we suppose $s \not\models \exists\Diamond((\mathrm{energy} \leqslant e) \wedge ZEUD))$. The task is to show that $\mathrm{ExpUtil}^{\max}_{\mathcal{M},s}(\mathrm{energy} \leqslant e)$ is finite. We regard the **MDP** $\mathcal{M}_e$ that arises from $\mathcal{M}$ by mimicking the energy function rew_e of $\mathcal{M}$ by a counter with values in $\{0, 1, \ldots, e\}$. The enabled actions of state $\langle t, f\rangle \in \mathsf{S}[e]$ in $\mathcal{M}_e$ are the actions $\alpha \in \mathsf{Act}_{\mathcal{M}}(t)$ where $\mathsf{rew}_e(t, \alpha) \leqslant e - f$. Furthermore, $\mathcal{M}_e$ collapses all zero-energy zero-utility end components of $\mathcal{M}$ into a single state and discards $\mathcal{M}$'s actions inside zero-energy zero-utility end components[1]. Then, $\mathsf{Pr}^{\min}_{\mathcal{M}_e,\langle s,0\rangle}(\Diamond\mathsf{final}) = 1$ where final is an atomic proposition characterising all states of $\mathcal{M}_e$ that do not have any enabled action. Thus, $\mathrm{ExpUtil}^{\max}_{\mathcal{M}_e,\langle s,0\rangle}(\Diamond\mathsf{final})$ is finite, and we have:

$$\mathrm{ExpUtil}^{\max}_{\mathcal{M},s}(\mathrm{energy} \leqslant e) = \mathrm{ExpUtil}^{\max}_{\mathcal{M}_e,\langle s,0\rangle}(\Diamond\mathsf{final})$$

We now turn to the proof of part (b). Let us first consider the case where $\mathrm{ExpUtil}^{\min}_{s}(\mathrm{energy} \leqslant e)$ is finite, i.e., $\mathrm{ExpUtil}^{\mathfrak{S}}_{s}(\mathrm{energy} \leqslant e) < \infty$ for some scheduler $\mathfrak{S}$. Then:

$$\mathsf{Pr}^{\mathfrak{S}}_{s}\big\{\pi \in \mathit{InfPaths} : \mathrm{utility}|_{\mathrm{energy} \leqslant e}(\pi) = \infty\big\} = 0$$

But then, $\mathsf{Pr}^{\mathfrak{S}}_{s}(\varphi) = 0$.

Vice versa, suppose $\mathsf{Pr}^{\min}_{s}(\varphi) = 0$. We pick some finite-memory scheduler $\mathfrak{S}$ with $\mathsf{Pr}^{\mathfrak{S}}_{s}(\varphi) = 0$. Then, the limit of almost all $\mathfrak{S}$-paths from s is either an energy-divergent end component or a zero-energy zero-utility end component. Let $\mathcal{M}_e$ be the **MDP** as above. When regarding $\mathfrak{S}$ as a scheduler for $\mathcal{M}_e$ (as done in the proof of Lemma 4.2.3), $\mathsf{Pr}^{\mathfrak{S}}_{\mathcal{M}_e,\langle s,0\rangle}(\Diamond\mathsf{final}) = 1$ as $\mathcal{M}_e$ has no zero-energy zero-utility end components (by construction). Moreover:

$$\mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\mathrm{energy} \leqslant e) = \mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M}_e,\langle s,0\rangle}(\Diamond\mathsf{final}) < \infty$$

Hence, $\mathrm{ExpUtil}^{\min}_{\mathcal{M},s}(\mathrm{energy} \leqslant e)$ is finite. □

4.2.1 Existential expectation quantiles

The procedure to compute the expectation quantiles $\mathrm{qu}_s(\exists\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?))$ and $\mathrm{qu}_s(\forall\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?))$ by an iterative search $e = 0, 1, 2, \ldots$ as explained in Section 4.1 requires a preprocessing that identifies all states where the quantile has value ∞.

---

[1] We drop here the precise definition of $\mathcal{M}_e$. A similar construction of the **MDP** $\widetilde{\mathcal{M}}$ that arises from $\mathcal{M}$ by identifying all states that belong to some zero-utility end component is presented in the proof of Lemma 4.2.3.

We start with explanations for the case of existential expectation quantiles. Obviously, for each utility bound $u \in \mathbb{R}$:

$$\begin{aligned} \mathrm{qu}_s\big(\exists \mathrm{ExpU}_{>u}(\text{energy} \leqslant ?)\big) = \infty \quad & \text{iff} \quad \big\{e \in \mathbb{N} \,:\, \mathrm{ExpUtil}_s^{\max}(\text{energy} \leqslant e) > u\big\} = \varnothing \\ & \text{iff} \quad \sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\max}(\text{energy} \leqslant e) \leqslant u \\ & \text{iff} \quad \mathrm{ExpUtil}_s^{\mathfrak{S}}(\text{energy} \leqslant e) \leqslant u \end{aligned}$$

for all $e \in \mathbb{N}$ and all schedulers $\mathfrak{S}$.

Let $\mathit{Lim}(UD)$ denote the event consisting of all infinite paths π such that the limit of π is a utility-divergent end component. Likewise, we write $\mathit{Lim}(ZU)$ to denote the event given by $\pi \models \mathit{Lim}(ZU)$ iff the limit of π is a zero-utility end component. Hence, $\mathsf{Pr}_s^{\min}(\mathit{Lim}(UD) \vee \mathit{Lim}(ZU)) = 1$.

Lemma 4.2.3. *For each state s:*

$$s \models \exists \Diamond \mathit{UD} \quad \textit{iff} \quad \sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\max}(\text{energy} \leqslant e) = \infty$$

Proof. To prove "$\Longrightarrow$", we suppose $s \models \exists \Diamond \mathit{UD}$ and pick some utility bound $u \in \mathbb{Q}$. The task is to show that there exists an energy bound $e \in \mathbb{N}$ with $\mathrm{ExpUtil}_s^{\max}(\text{energy} \leqslant e) > u$. The latter amounts proving the existence of some scheduler $\mathfrak{U}$ with $\mathrm{ExpUtil}_s^{\mathfrak{U}}(\text{energy} \leqslant e) > u$.

As $s \models \exists \Diamond \mathit{UD}$, we may choose a shortest finite path ρ starting in s and ending in some state in UD. Let $\mathfrak{S}$ be a (memoryless) scheduler such that ρ is a $\mathfrak{S}$-path. With $e_1 = \text{energy}(\rho)$ and p the probability of ρ we get

$$\mathsf{Pr}_s^{\mathfrak{S}}\big(\Diamond((\text{energy} \leqslant e_1) \wedge \mathit{UD})\big) \geqslant p > 0.$$

Let $k \in \mathbb{N}$ be a positive natural number with $p \geqslant 1/k$. We now consider any finite-memory scheduler $\mathfrak{T}$ that "realises" the utility-divergent end components, i.e., the limit of almost all infinite $\mathfrak{T}$-paths starting in some state $t \in \mathit{UD}$ is a utility-divergent end component. Clearly, along all these infinite $\mathfrak{T}$-paths, the accumulated utility tends to ∞. Hence, there exists some $e_2 \in \mathbb{N}$ such that for all states $t \in \mathit{UD}$:

$$\mathsf{Pr}_t^{\mathfrak{T}}\big(\Diamond((\text{energy} \leqslant e_2) \wedge (\text{utility} > 2k \cdot u))\big) > \frac{1}{2}$$

Let $\mathfrak{U} = \mathfrak{S} \circ \mathfrak{T}$ be the scheduler that behaves like $\mathfrak{S}$ as long as no state in UD has been visited. As soon as a state in UD is visited, then $\mathfrak{U}$ mimics $\mathfrak{T}$. We get:

$$\mathsf{Pr}_s^{\mathfrak{U}}\{\pi \;:\; \pi \models \varphi\} > \frac{1}{2k}$$

where φ is the following LTL-like formula:

$$\varphi = \Diamond\big((\text{energy} \leqslant e_1) \wedge \mathit{UD}\big) \;\wedge\; \Diamond\big((\text{energy} \leqslant e) \wedge (\text{utility} > 2k \cdot u)\big)$$

and $e = e_1 + e_2$. Hence:

$$\mathrm{ExpUtil}_t^{\mathfrak{U}}\big(\text{energy} \leqslant e\big) > \frac{1}{2k} \cdot 2k \cdot u = u$$

This yields the claim.

We now turn to the proof of the implication "$\Longleftarrow$". The task is to show that $s \not\models \exists\Diamond UD$ implies the existence of some $u \in \mathbb{Q}$ such that

$$\mathrm{ExpUtil}_s^{\max}(\text{energy} \leqslant e) \leqslant u$$

for all energy bounds $e \in \mathbb{N}$. The assumption that no utility-divergent end component is reachable from s yields that all end components that are reachable from s enjoy the zero-utility property. Since under each scheduler the limit of almost all infinite paths is an end component we get:

$$\mathsf{Pr}_s^{\min}(Lim(ZU)) = 1$$

In what follows, we define a new **MDP** $\widetilde{\mathcal{M}}$ with a reward function rew and a goal state that will be reached almost surely under each scheduler for $\widetilde{\mathcal{M}}$ such that, for each $e \in \mathbb{N}$, the value $\mathrm{ExpUtil}_{\mathcal{M},s}^{\max}(\text{energy} \leqslant e)$ is bounded by the maximal expected reward until reaching the goal state in $\widetilde{\mathcal{M}}$.

The idea for the definition of $\widetilde{\mathcal{M}}$ is to collapse the effect of all transitions inside zero-utility end components. Intuitively, this is justified as for the accumulated utility in $\mathcal{M}$, the behaviour inside zero-utility end components is irrelevant. The accumulated energy might increase inside zero-utility end components. However, we are interested only in an upper bound for the expected total accumulated utility in the new **MDP** $\widetilde{\mathcal{M}}$.

We first define an equivalence relation $\sim_{ZU}$ on ZU. For $t_1, t_2 \in ZU$, $t_1 \sim_{ZU} t_2$ holds iff there exists some zero-utility end component that contains t_1 and t_2. For $t \in ZU$, let $[t]_{ZU}$ denote the $\sim_{ZU}$-equivalence class of t and let

$$\mathsf{Act}_{ZU}(t) = \big\{\beta \in \mathsf{Act}(t) \ : \ \mathsf{rew}_u(s,\beta) = 0, \{t' \in \mathsf{S} \ : \ P(t,\beta,t') > 0\} \subseteq [t]_{ZU}\big\}$$

denote the set of actions of t inside some zero-utility end component. We write $\sim$ to denote the following equivalence relation on the state space S of $\mathcal{M}$:

$$s_1 \sim s_2 \quad \text{iff} \quad (s_1 = s_2) \vee (s_1, s_2 \in ZU \wedge s_1 \sim_{ZU} s_2)$$

Let $[s]$ denote the equivalence class of s with respect to $\sim$. Thus, $[s] = \{s\}$ if $s \in \mathsf{S}\backslash ZU$, while $[t] = [t]_{ZU}$ for $t \in ZU$. The state space of the new **MDP** $\widetilde{\mathcal{M}}$ is:

$$\widetilde{S} = \big\{[s] \ : \ s \in \mathsf{S}\big\} \cup \big\{\mathsf{goal}\big\}$$

For simplicity, let us suppose that the actions in $\mathcal{M}$ have been renamed such that $\mathsf{Act}(s_1) \cap \mathsf{Act}(s_2) = \varnothing$ if $s_1 \neq s_2$. Then, the action set of $\widetilde{\mathcal{M}}$ is

$$\widetilde{\mathsf{Act}} = \mathsf{Act} \cup \{\tau\}$$

The transition probability function $\widetilde{P}$ of $\widetilde{\mathcal{M}}$ is defined as follows.

- If $s \in \mathsf{S} \backslash ZU$ and $\alpha \in \mathsf{Act}(s)$, then $\widetilde{P}([s], \alpha, [s']) = P(s, \alpha, [s'])$ for $s' \in \mathsf{S}$, $\widetilde{P}([s], \alpha, \mathsf{goal}) = 0$ and $\mathsf{rew}(s, \alpha) = \mathsf{rew}_u(s, \alpha)$[2]. Action τ is not enabled in the states $[s]$ where $s \in \mathsf{S} \backslash ZU$. Thus, the set $\widetilde{\mathsf{Act}}([s])$ of actions that are enabled in $[s]$ equals $\mathsf{Act}(s)$.
- Let $Z \subseteq ZU$ be a $\sim_{ZU}$-equivalence class. The set of actions that are enabled in Z is:

  $$\widetilde{\mathsf{Act}}(Z) = \{\tau\} \cup \bigcup_{t \in Z} \mathsf{Act}(t) \backslash \mathsf{Act}_{ZU}(t)$$

  If $t \in Z$ and $\alpha \in \mathsf{Act}(t) \backslash \mathsf{Act}_{ZU}(t)$ then $\widetilde{P}(Z, \alpha, [s']) = P(t, \alpha, [s'])$ for $s' \in \mathsf{S}$, $\widetilde{P}(Z, \alpha, \mathsf{goal}) = 0$ and $\mathsf{rew}(Z, \alpha) = \mathsf{rew}_u(t, \alpha)$. Furthermore, $\widetilde{P}(Z, \tau, \mathsf{goal}) = 1$ and $\mathsf{rew}(Z, \tau) = 0$.
- The fresh state goal is a trap with a single τ-labeled zero-reward self-loop, i.e., $\widetilde{P}(\mathsf{goal}, \tau, \mathsf{goal}) = 1$ and $\mathsf{rew}(\mathsf{goal}, \tau) = 0$.

As $\mathsf{Pr}^{\min}_{\mathcal{M},s}(Lim(ZU)) = 1$, the only end component that is accessible from state s in $\widetilde{\mathcal{M}}$ consists of the goal state. In particular:

$$\mathsf{Pr}^{\min}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal}) = 1$$

Moreover, for each scheduler $\mathfrak{S}$ for $\mathcal{M}$ there exists a corresponding (randomised) scheduler $\widetilde{\mathfrak{S}}$ for $\widetilde{\mathcal{M}}$ that mimics $\mathfrak{S}$'s behaviour inside zero-utility end components by a probabilistic choice. Speaking roughly, if the current state is neither the goal state nor a $\sim_{ZU}$-equivalence class, then $\widetilde{\mathfrak{S}}$ behaves as $\mathfrak{S}$. When entering a $\sim_{ZU}$-equivalence class Z, then $\widetilde{\mathfrak{S}}$ selects an action in $\widetilde{\mathsf{Act}}(Z)$ probabilistically according to the probabilities for $\mathfrak{S}$ to generate a finite path $\rho = t_0\, \beta_0\, t_1\, \beta_1 \, \ldots \, \beta_{n-1}\, t_n\, \alpha\, v$ where $t_0, \ldots, t_n$ and their actions $\beta_0, \ldots, \beta_{n-1}$ belong to Z and $\alpha \notin \mathsf{Act}_{ZU}(t_n)$ or to stay forever in ZU (in which case $\widetilde{\mathfrak{S}}$ moves to the goal state by scheduling τ). Then, the expected total accumulated utility for $\mathfrak{S}$ from state s in $\mathcal{M}$ equals the expected total utility for $\widetilde{\mathfrak{S}}$ from state $[s]$ in $\widetilde{\mathcal{M}}$, which again agrees with the expected accumulated utility until reaching goal from $[s]$ under scheduler $\widetilde{\mathfrak{S}}$. That is:

$$\mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\mathsf{true}) = \mathsf{ExpRew}^{\widetilde{\mathfrak{S}}}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal})$$

Recall that the limit of almost all $\mathfrak{S}$-paths is a zero-utility end component in $\mathcal{M}$. Thus, the expected total accumulated utility exists in $\mathcal{M}$. Hence, for all $e \in \mathbb{N}$ we have:

$$\mathrm{ExpUtil}^{\max}_{\mathcal{M},s}(\mathrm{energy} \leqslant e) \leqslant \mathsf{ExpRew}^{\max}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal})$$

and therefore:

$$\sup_{e \in \mathbb{N}} \mathrm{ExpUtil}^{\max}_{\mathcal{M},s}(\mathrm{energy} \leqslant e) \leqslant \mathsf{ExpRew}^{\max}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal})$$

The latter is finite as $\Diamond$goal holds almost surely under each scheduler for $\widetilde{\mathcal{M}}$. $\square$

---

[2] Here and in what follows, for $s \in \mathsf{S}$, $\alpha \in \mathsf{Act}$ and $R \subseteq \mathsf{S}$, we write $P(s, \alpha, R)$ for $\sum_{s' \in R} P(s, \alpha, s')$.

Corollary 4.2.4 (Infinite existential expectation quantiles). *For each $u \in \mathbb{Q}$, the following statements are equivalent:*

(a) $\mathrm{qu}_s\big(\,\exists\mathrm{ExpU}_{>u}(\text{energy} \leqslant ?)\,\big) = \infty$

(b) $\sup\limits_{e \in \mathbb{N}} \ \mathrm{ExpUtil}_s^{\max}(\text{energy} \leqslant e) \ \leqslant \ u$

(c) $s \not\models \exists\Diamond\, UD$ *and* $\mathsf{ExpRew}^{\max}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal}) \leqslant u$

where $\widetilde{\mathcal{M}}$ is defined as in the proof of Lemma 4.2.3.

Proof. The equivalence of (a) and (b) is obvious. The implication "(c) $\Longrightarrow$ (b)" has been shown in the proof of Lemma 4.2.3.

It remains to prove the implication "(b) $\Longrightarrow$ (c)". The fact that (b) implies $s \not\models \exists\Diamond\, UD$ is a consequence of Lemma 4.2.3. To see why (b) implies $\mathsf{ExpRew}^{\max}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal}) \leqslant u$ we pick some scheduler $\widetilde{\mathfrak{S}}$ for $\widetilde{\mathcal{M}}$. It suffices to show that there exists a scheduler $\mathfrak{S}$ for $\mathcal{M}$ such that the expected total accumulated utility under $\mathfrak{S}$ agrees with the expected accumulated reward until reaching the goal state in $\widetilde{\mathcal{M}}$ under the scheduler $\widetilde{\mathfrak{S}}$. That is:

$$\mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\mathsf{true}) = \mathsf{ExpRew}^{\widetilde{\mathfrak{S}}}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal})$$

To prove the existence of such a scheduler $\mathfrak{S}$, we pick a finite-memory scheduler $\mathfrak{T}$ for $\mathcal{M}$ realising the zero-utility end components in the sense that $\mathsf{Pr}^{\mathfrak{T}}_{\mathcal{M},t'}(ZU \,\mathcal{U}\, t) = 1$ for all states $t, t' \in ZU$ with $t \sim_{ZU} t'$, where $ZU \,\mathcal{U}\, t$ is used as a short form notation for all infinite paths that have a finite prefix $s_0\, \alpha_0\, s_1\, \alpha_1 \ldots \alpha_{k-1}\, s_k$ with $s_k = t$ and $\mathsf{rew}_u(s_i, \alpha_i) = 0$ for $0 \leqslant i < k$.

Let $\mathfrak{S}$ be the scheduler that behaves as $\widetilde{\mathfrak{S}}$ for the states $s \in \mathsf{S}\backslash ZU$ (where we identify s and the singleton $[s] = \{s\}$) and mimics $\widetilde{\mathfrak{S}}$'s decisions for the $\sim_{ZU}$-equivalence classes by simulating the actions $\alpha \in \mathsf{Act}(t)\backslash\mathsf{Act}_{ZU}(t)$ for $t \in ZU$ by first following $\mathfrak{T}$'s decisions until state t has been reached, and then selecting action α. If $\widetilde{\mathfrak{S}}$ moves to the goal state via the action τ from some $\sim_{ZU}$-equivalence class $Z \in \widetilde{S}$, then $\mathfrak{S}$ changes mode and behaves as $\mathfrak{T}$ from then on. We obtain:

$$\begin{aligned}\mathsf{ExpRew}^{\widetilde{\mathfrak{S}}}_{\widetilde{\mathcal{M}},[s]}(\Diamond \mathsf{goal}) &= \mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\mathsf{true}) \\ &= \sup_{e \in \mathbb{N}} \mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\text{energy} \leqslant e)\end{aligned}$$

As $\sup\limits_{e \in \mathbb{N}} \mathrm{ExpUtil}^{\mathfrak{S}}_{\mathcal{M},s}(\text{energy} \leqslant e) \leqslant u$ by assumption (b), this yields the claim. $\square$

4.2.2 Universal expectation quantiles

Now, universal expectation quantiles are considered. Obviously, for each utility bound $u \in \mathbb{Q}$ we have the following:

$$\begin{aligned}\mathrm{qu}_s\big(\forall\mathrm{ExpU}_{>u}(\mathrm{energy} \leqslant ?)\big) = \infty \quad &\text{iff} \quad \big\{e \in \mathbb{N} : \mathrm{ExpUtil}_s^{\min}(\mathrm{energy} \leqslant e) > u\big\} = \varnothing \\ &\text{iff} \quad \sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\min}(\mathrm{energy} \leqslant e) \leqslant u \\ &\text{iff} \quad \text{for each } e \in \mathbb{N} \text{ there is some scheduler } \mathfrak{S}_e \\ &\qquad \text{such that } \mathrm{ExpUtil}_s^{\mathfrak{S}_e}(\mathrm{energy} \leqslant e) \leqslant u\end{aligned}$$

We first observe that for reasoning about $\mathrm{ExpUtil}_s^{\min}(\mathrm{energy} \leqslant e)$ it suffices to consider schedulers that stay forever in ZU as soon as they have reached some state in ZU, see Lemma 4.2.5 below.

With abuse of notations, we interpret ZU as a set of states or as a set of states and actions. Thus, if $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,s_2\,\alpha_2\,\ldots$ is an infinite path then:

$$\begin{aligned}\pi \models \Diamond ZU \quad &\text{iff} \quad s_k \in ZU \text{ for some } k \in \mathbb{N} \\ \pi \models \Box ZU \quad &\text{iff} \quad s_k \in ZU \text{ and } \mathsf{rew}_u(s_k, \alpha_k) = 0 \text{ for all } k \in \mathbb{N}.\end{aligned}$$

Moreover, $\pi \models \Box(ZU \to \Box ZU)$ if and only if either $ZU \cap \{s_j : j \in \mathbb{N}\} = \varnothing$ or there is some $k \in \mathbb{N}$ with $ZU \cap \{s_0, \ldots, s_{k-1}\} = \varnothing$ and $\mathit{suff}(\pi, k) \models \Box ZU$, where $\mathit{suff}(\pi, k) = s_k\,\alpha_k\,s_{k+1}\,\alpha_{k+1}\,s_{k+2}\,\alpha_{k+2}\,\ldots$ denotes the suffix of π starting at position k.

Lemma 4.2.5. *Let $\mathfrak{S}$ be a scheduler for $\mathcal{M}$. Then, there exists a scheduler $\mathfrak{S}'$ for $\mathcal{M}$ such that for each state s and each energy bound $e \in \mathbb{N}$:*

$$\mathrm{ExpUtil}_s^{\mathfrak{S}'}(\mathrm{energy} \leqslant e) \leqslant \mathrm{ExpUtil}_s^{\mathfrak{S}}(\mathrm{energy} \leqslant e)$$

and

$$\mathsf{Pr}_s^{\mathfrak{S}'}\big(\Box(ZU \to \Box ZU)\big) = 1$$

Proof. Again, let $\mathfrak{T}$ be a finite-memory scheduler realising the zero-utility end components, i.e., $\mathsf{Pr}_t^{\mathfrak{T}}(\Box ZU) = 1$ for each state $t \in ZU$. Given an arbitrary scheduler $\mathfrak{S}$, we define $\mathfrak{S}'$ to be a scheduler that behaves as $\mathfrak{S}$ until some ZU-state has been reached, in which case $\mathfrak{S}'$ switches mode and behaves as $\mathfrak{T}$ from then one. $\square$

Lemma 4.2.6. *If $\mathfrak{S}$ is a scheduler for $\mathcal{M}$ such that $\mathsf{Pr}_s^{\mathfrak{S}}(\mathit{Lim}(UD)) > 0$, then*

$$\sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\mathfrak{S}}(\mathrm{energy} \leqslant e) = \infty$$

Proof. We first observe that

$$\mathrm{ExpUtil}_s^{\mathfrak{S}}(\mathsf{true}) = \sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\mathfrak{S}}(\mathrm{energy} \leqslant e)$$

is the expected total utility under scheduler $\mathfrak{S}$ from state s. The claim then follows from the fact that the accumulated utility value of the prefixes along paths whose limit is a utility-divergent end component converges to ∞. The assumption $\mathsf{Pr}_s^{\mathfrak{S}}(\mathit{Lim}(UD)) > 0$ yields that these paths have positive measure. $\square$

An immediate consequence of Lemma 4.2.6 is as follows:

Corollary 4.2.7. *If* $\mathsf{Pr}_s^{\min}(\mathit{Lim}(UD)) > 0$ *then* $\sup_{e\in\mathbb{N}} \mathrm{ExpUtil}_s^{\min}(\text{energy} \leqslant e) = \infty$.

Now, we address the case $\mathsf{Pr}_s^{\min}(\mathit{Lim}(UD)) = 0$, in which case $\mathsf{Pr}_s^{\max}(\mathit{Lim}(ZU)) = 1$. We will rely on the following assumptions that can be ensured by some adequate preprocessing. Lemma 4.2.5 allows to suppose that $\mathcal{M}$ satisfies the following property:

(A1) Whenever $t \in ZU$ and $P(t,\alpha,t') > 0$ then $t' \in ZU$ and $\mathsf{rew}_u(t,\alpha) = 0$.

Furthermore, we suppose that $t \models \exists\Diamond ZU$ for all states t in $\mathcal{M}$. Hence, there is a scheduler $\mathfrak{S}$ such that from all states t, the limit of almost all $\mathfrak{S}$-paths is a zero-utility end component. In particular:

(A2) $\mathsf{Pr}_{\mathcal{M},t}^{\max}(\Diamond ZU) = 1$ for all states t

Let $\mathfrak{V}$ be a deterministic memoryless scheduler for $\mathcal{M}$ such that $\mathsf{Pr}_{\mathcal{M},t}^{\mathfrak{V}}(\Diamond ZU) = 1$ for all states t and

(A3) $\mathrm{ExpUtil}_{\mathcal{M},t}^{\mathfrak{V}}(\Diamond ZU) = \mathrm{ExpUtil}_{\mathcal{M},t}^{\min}(\Diamond ZU)$ for all states t

The existence of such a (deterministic and memoryless) scheduler $\mathfrak{V}$ has been shown by de Alfaro [Alf99]. Note that $\mathrm{ExpUtil}_{\mathcal{M},t}^{\mathfrak{V}}(\Diamond ZU)$ is the expected total utility from t under scheduler $\mathfrak{V}$.

Lemma 4.2.8. *Suppose assumptions (A1) and (A2) hold. Then, for all* $u \in \mathbb{Q}$ *and all states* s *of* $\mathcal{M}$*:*

$$\sup_{e\in\mathbb{N}} \mathrm{ExpUtil}_{\mathcal{M},s}^{\min}(\text{energy} \leqslant e) = \mathrm{ExpUtil}_{\mathcal{M},s}^{\min}(\Diamond ZU)$$

Proof. To prove "$\geqslant$" we consider a deterministic memoryless scheduler $\mathfrak{V}$ minimising the expected total utility (see (A3)). Then:

$$\begin{aligned}
\mathrm{ExpUtil}_{\mathcal{M},s}^{\min}(\Diamond ZU) &= \mathrm{ExpUtil}_{\mathcal{M},s}^{\mathfrak{V}}(\Diamond ZU)\\
&= \mathrm{ExpUtil}_{\mathcal{M},s}^{\mathfrak{V}}(\mathsf{true})\\
&= \sup_{e\in\mathbb{N}} \mathrm{ExpUtil}_{\mathcal{M},s}^{\mathfrak{V}}(\text{energy} \leqslant e)\\
&\geqslant \sup_{e\in\mathbb{N}} \mathrm{ExpUtil}_{\mathcal{M},s}^{\min}(\text{energy} \leqslant e)
\end{aligned}$$

The remaining task is to prove "$\leqslant$". Let

$$u^{\min} = \sup_{e\in\mathbb{N}} \mathrm{ExpUtil}_{\mathcal{M},s}^{\min}(\text{energy} \leqslant e)$$

For each $e \in \mathbb{N}$ there is a (deterministic) scheduler $\mathfrak{S}_e$ such that

$$\mathrm{ExpUtil}_s^{\mathfrak{S}_e}(\text{energy} \leqslant e) \leqslant u^{\min}$$

The sequence $(\mathfrak{S}_e)_{e \in \mathbb{N}}$ can be used to generate a scheduler $\mathfrak{S}$ such that for each $k \in \mathbb{N}$ there are infinitely many $e \in \mathbb{N}$ with $\mathfrak{S}(\rho) = \mathfrak{S}_e(\rho)$ for all finite paths ρ of length at most k. For this scheduler $\mathfrak{S}$ and each $e \in \mathbb{N}$, we have:

$$\begin{aligned}\mathrm{ExpUtil}_s^{\mathfrak{S}}(\text{energy} \leqslant e) &= \sup_{k \in \mathbb{N}} \mathrm{ExpUtil}_s^{\mathfrak{S}}\big((\text{energy} \leqslant e) \wedge (\mathit{steps} \leqslant k)\big) \\ &\leqslant u^{\min}\end{aligned}$$

where *steps* serves as a step counter. Thus:

$$\sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\mathfrak{S}}(\text{energy} \leqslant e) \leqslant u^{\min}$$

In particular, $\mathsf{Pr}_s^{\mathfrak{S}}(\mathit{Lim}(UD)) = 0$ by Lemma 4.2.6. Therefore, $\mathsf{Pr}_s^{\mathfrak{S}}(\mathit{Lim}(ZU)) = 1$ and:

$$\begin{aligned}\mathrm{ExpUtil}_s^{\mathfrak{S}}(\Diamond ZU) &= \mathrm{ExpUtil}_s^{\mathfrak{S}}(\mathsf{true}) \\ &= \sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\mathfrak{S}}(\text{energy} \leqslant e)\end{aligned}$$

Putting things together, we obtain:

$$\mathrm{ExpUtil}_s^{\min}(\Diamond ZU) \quad \leqslant \quad \mathrm{ExpUtil}_s^{\mathfrak{S}}(\Diamond ZU) \quad \leqslant \quad u^{\min}$$

This yields the claim. □

Corollary 4.2.9 (Infinite universal expectation quantiles). *Under assumptions (A1) and (A2), for each $u \in \mathbb{Q}$, the following statements are equivalent:*

(a) $\mathrm{qu}_s\big(\forall \mathrm{ExpU}_{>u}(\text{energy} \leqslant ?)\big) = \infty$

(b) $\sup_{e \in \mathbb{N}} \mathrm{ExpUtil}_s^{\min}(\text{energy} \leqslant e) \leqslant u$

(c) $\mathsf{Pr}_s^{\max}(\Diamond ZU) = 1$ *and* $\mathrm{ExpUtil}_s^{\min}(\Diamond ZU) \leqslant u$

Statements (c) of both, Corollary 4.2.4 and Corollary 4.2.9, provide criteria to check whether a given expectation quantile has a finite value. Those checks can be done in polynomial time. Therefore, an immediate consequence is as follows:

Corollary 4.2.10. *The following two problems are in* ***P****:*

- *decide whether* $\mathrm{qu}_s\big(\exists \mathrm{ExpU}_{>u}(\text{energy} \leqslant ?)\big) = \infty$
- *decide whether* $\mathrm{qu}_s\big(\forall \mathrm{ExpU}_{>u}(\text{energy} \leqslant ?)\big) = \infty$

So, the statements (c) of the referred corollaries serve as the basis for the precomputations of the expectation quantiles (see Section 5.2 for details on the importance of precomputations for the quantile calculations).

5 Implementation

Since the presented computation methods were designed in order to analyse and optimise the energy efficiency of adaptive systems, it is of high interest to have reliable tools for supporting an energy-efficient analysis based on the so far presented methods. Therefore, the current chapter introduces a realisation of the framework for providing quantiles as an instrument for the analysis. The probabilistic model checker Prism [KNP11] hereby serves as the basis for the implementation, since an integration of the methods into Prism allows one to have support for some of the necessary fundamental methods that are required for the computation of quantiles in Markovian models. For example, Prism delivers some of the necessary infrastructure for the handling of the Markovian models themselves or the related reward structures, and one can therefore utilise the shipped model-support and focus on the implementation of the computation methods for the quantiles. Another important fact is that Prism is a well-established tool within the formal methods community. So, the integration of the quantile computations into Prism might help to make the presented framework publicly available to a large audience. And moreover there exists a variety of already created models for a large number of protocols. Those protocols cover various areas of application and can be analysed using the presented framework immediately.

Before starting to describe the integration into Prism, we first take a look at computational optimisations that can be done in order to improve the calculations of quantiles and that help to decrease the memory consumption of the computations as well as improving the timings needed for doing the desired analysis. All the described optimisations have been integrated into the implementation in order to improve the performance of the calculations.

5.1 Computation optimisations

Since model checking is a data- and compute-intensive task it is very important to provide different optimisation techniques to reduce the memory- and the time-requirements of the performed analysis. Since a reduction in the consumed memory can cause an increase in computation-times, and as reduced computation-times may need an increased amount of memory one needs to find a suitable trade-off between the utilised resources for the desired analysis. Therefore, different mechanisms to adapt the performed calculations to varying needs are proposed in the following with respect to the computation of quantiles. The provided methods mainly rely on structural information inherently given by the type of linear programs that need to be solved when computing quantile queries. Therefore, there exist a number of technological possibilities that have a thorough impact on the performance of the quantile computations. In the following we want to outline the possibilities and go into some necessary details.

5.1.1 Back propagation

The main bottleneck of the *LP*-approach for computing quantitative quantiles (see, e.g., Figure 3.3 on page 29) is the possibly exponential size of the linear program, and that the complete linear program is solved in one huge chunk. Therefore, an iterative approach is proposed that computes the values $p_{s,i} = \mathsf{Pr}^{\max}_s(A\,\mathcal{U}^{\leqslant i}\,B)$ successively for $i = 0, 1, 2, \dots$ by decomposing the complete linear program that needs to be solved into smaller ones, i.e., one linear program for $i = 0$ (called $\mathbb{LP}_0$), one for $i = 1$ ($\mathbb{LP}_1$), and so on. This allows to solve the linear programs separately and reusing already computed values as much as possible by propagating those values. Therefore, due to the reuse and propagation of already computed values, this approach is called *back-propagation approach* or simply *BP-approach.*

Given that the solution $(p_{s,j})_{0\leqslant j<i}$ for $\mathbb{LP}_{i-1}$ is known when considering $\mathbb{LP}_i$, the constraints for variable $x_{s,i}$ in the third case of the linear program in Figure 3.3 on page 29 (i.e., if $s \notin B$, $s \models \exists(A\,\mathcal{U}\,B)$ and $\alpha \in \mathsf{Act}(s)$) can be rewritten as follows:

$$\begin{aligned} x_{s,i} &\geqslant \textstyle\sum_{t\in \mathsf{S}} P(s,\alpha,t)\cdot x_{t,i} && \text{if } \mathsf{rew}(s,\alpha) = 0\\ x_{s,i} &\geqslant c_{s,i} && \text{if } \mathsf{rew}(s,\alpha) > 0 \end{aligned}$$

where

$$c_{s,i} \overset{\text{def}}{=} \max\left\{ \sum_{t\in\mathsf{S}} P(s,\alpha,t)\cdot p_{t,i-\mathsf{rew}(s,\alpha)} \;:\; \alpha \in \mathsf{Act}(s), \mathsf{rew}(s,\alpha) > 0 \right\}.$$

The linear programs obtained in this manner will be denoted by $\mathbb{LP}'_i$. We can now use standard methods to solve $\mathbb{LP}'_i$ with variables $(x_{s,i})_{s\in\mathsf{S}}$ consisting of the above linear constraints together with the terminal cases $x_{s,i} = 0$ if $s \not\models \exists(A\,\mathcal{U}\,B)$ and $x_{s,i} = 1$ if $s \in B$, where the objective is to "minimise $\sum_{s\in\mathsf{S}} x_{s,i}$". $\mathbb{LP}'_i$ has indeed a unique solution which agrees with the (unique) solution $(p_{s,i})_{s\in\mathsf{S}}$ of $\mathbb{LP}_i$ for the variables $x_{s,i}$. An immediate consequence of the back-propagation approach is the fact that the computation of the positive-reward fragment of the model becomes quite simple within every single iteration of the *BP*-approach. Only the treatment of the zero-reward fragment remains a task that is a bit more involved.

Suppose the task is to compute $q_s = \mathrm{qu}_s\big(\exists\mathbb{P}_{>p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)$ for all states s. Let $n = \mathsf{card}(\mathsf{S})$, $m = \sum_{s\in\mathsf{S}} \mathsf{card}(\mathsf{Act}(s))$ and z be the number of state-action pairs (s,α) for which $s \in \mathsf{S}$, $\alpha \in \mathsf{Act}(s)$ and $\mathsf{rew}(s,\alpha) = 0$. Then, with the proposed back-propagation approach, $(q_s)_{s\in\mathsf{S}}$ is obtained by first computing $\mathsf{Pr}^{\max}_s(A\,\mathcal{U}\,B)$ for all states s (which can be done in time polynomial in the size of $\mathcal{M}$ [BA95; BK08] and serves to identify the states $s \in \mathsf{S}$ where $q_s = \infty$) and then solving the LPs $\mathbb{LP}'_0, \mathbb{LP}'_1, \dots, \mathbb{LP}'_r$ consecutively (where $r \in \max\{q_s : \mathsf{Pr}^{\max}_s(A\,\mathcal{U}\,B) > p\}$) with n variables and $z + n$ linear constraints each.

5.1.2 Reward window

The usage of the *BP*-approach takes advantage of reusing already calculated values from previous iterations. This enables optimisation techniques for reducing the memory-consumption needed to store the previously calculated values. In order to reduce the memory requirements, one can use the observation that the constants $c_{s,i}$ in $\mathbb{LP}'_i$ are obtained from the values $p_{t,i-\mathsf{rew}(s,\alpha)}$ where $\alpha \in \mathsf{Act}(s)$ and $\mathsf{rew}(s,\alpha) > 0$. As a consequence, the solution $\big(p_{t,i}\big)_{t \in \mathsf{S}}$ for $\mathbb{LP}'_i$ can be discarded as soon as $\mathbb{LP}'_{i+w}$ has been solved for the maximal reward value $w = \max\big\{\mathsf{rew}(s,\alpha) \,:\, s \in \mathsf{S}, \alpha \in \mathsf{Act}(s)\big\}$ in $\mathcal{M}$. A further improvement considers the maximum reward of all incoming transitions per state. That is, the value of $p_{t,i}$ is not needed any more as soon as $\mathbb{LP}'_{i+w}$ has been solved where w equals the maximal reward of the state-action pairs (s,α) with $P(s,\alpha,t) > 0$.

Since there is only a reference to a previously calculated value when there occurs a positive reward (see the third case of the linear program in Figure 3.3 on page 29), only the values of direct successors of positive-reward states occur at the right-hand side of the inequation. Whereas, when there is no reward involved the references inside $\mathbb{LP}'_i$ do belong to the very same linear program and therefore the referenced value was not calculated previously. Therefore, only the values of direct successors of positive-reward states will be propagated by the *BP*-approach, and as a consequence it is only necessary to store the computed values of such states.

Using those observations we can utilise a reward window for storing the already computed values from previous iterations. This reward window has a fixed capacity determining the number of iterations within the *BP*-approach that the stored values should be available for supply. New values will be than added to the storage by simply replacing values that are no longer needed. The implementation of the quantile calculations supports three different approaches for storing the previously calculated values using a reward window:

All states

Using this approach the values for each state of the model are stored, regardless whether the values are needed in the necessary computations or not. Therefore, each state can be addressed directly without any need to do an index-recalculation whenever a reference is needed. In a precomputation-step the maximal reward value $w = \max\big\{\mathsf{rew}(s,\alpha) \,:\, s \in \mathsf{S}, \alpha \in \mathsf{Act}(s)\big\}$ of the whole model will be determined and every state has a reward window of size w. Therefore, it can be the case that many values will be stored despite the fact that the values may not be needed by the computations of the *BP*-approach.

Depending on the investigated model this approach can waste a huge amount of memory for values that will never be used during the whole calculations. But, on the other hand there are no additional index-recalculations needed like it is the case for the two approaches described next. So, when there is a model where almost all states are positive-reward successors this approach might be a good decision.

Uniform positive-reward successors

As stated earlier there is only a need to store values for the successors of positive-reward states (either positive state-rewards or positive transition-rewards or both). Therefore, this approach stores only the computed values for those successors. Depending on the model and the corresponding reward function this allows one to save a lot of memory compared to the previous approach. But, in order to be able to assign the stored values to the corresponding states of the model, it is required to use an additional data structure that takes care of the correct mapping. So, as a downside one has to do a mapping-computation each time an access to a stored value is needed. That is why one has to see if the memory savings will make up for the overhead of the additional index-recalculations. Again, a uniform reward window of size $w = \max\big\{\mathsf{rew}(s,\alpha) \,:\, s \in \mathsf{S}, \alpha \in \mathsf{Act}(s)\big\}$ will be used for each stored state, like it is done for the previous approach.

Depending on the model and the reward structure this approach might be a good compromise between the used memory for the storage and the time spent on the computations.

Individual positive-reward successors

As before, this approach only stores the calculated values for the successors of positive-reward states. But here, each state has an individual reward window that is exactly tailored for the specific state, meaning that each state $s \in \mathsf{S}$ has its own reward window of size $w_s = \max\big\{\mathsf{rew}(s,\alpha) \,:\, \alpha \in \mathsf{Act}(s)\big\}$. So, the consumed memory is minimised using this approach, and this approach becomes useful when it is the case that there are only a few states that need a very large reward window and most of the states come along with just a small window. Because using the two previous approaches in this case would allocate huge reward windows even for states that do not need those huge windows, so a significant amount of memory would be consumed without any use for the calculations.

As a drawback this approach needs more operations for resolving the references to the corresponding stored values from the previous iterations. Therefore, the time for the computation might be effected in a negative way in comparison to the other two approaches.

See the eBond-protocol on page 140 for performance statistics on the usage of the different reward window-approaches. There the number of elements that is stored by each approach (see Figure 6.20) can be related with the computation time that is needed in order to compute a complete query (see Figure 6.21). It can be seen that the approaches where we only store values for the positive-reward successors can reduce the memory consumption in a substantial way, but this reduction has to be paid in terms of increased computation times due to the additional operations needed for resolving the correct references within the stored values.

5.1.3 Topological sorting of zero-reward sub-MDPs

The back-propagation approach can yield a major speed-up compared to the naïve approach solving one huge single linear program. However, if the number of state-action pairs with zero reward is large compared to the full set of actions in S, the numerous linear programs $\mathbb{LP}'_i$ need still to be solved for several i. In order to improve the computation performance for solving a linear program $\mathbb{LP}'_i$ the idea is to decompose $\mathbb{LP}'_i$ and treat the sub-LPs in a specific order. Let $\mathcal{G}$ be the directed graph with node set S and the edge relation $\rightarrow \subseteq \mathsf{S} \times \mathsf{S}$ given by $s \rightarrow t$ iff $P(s, \alpha, t) > 0$ for some action $\alpha \in \mathsf{Act}(s)$ with $\mathsf{rew}(s, \alpha) = 0$. Applying standard graph algorithms[1], the strongly connected components in $\mathcal{G}$ will be computed and also a topological sorting $C_1, \ldots, C_k$ for those components. This will be done once initially and the derived topological order will be stored to use it throughout each required iteration. The SCCs $C_1, \ldots, C_k$ are then the finest partition of S such that:

$$\text{if } s \in C_h,\ t \in C_j,\ P(s, \alpha, t) > 0 \text{ and } \mathsf{rew}(s, \alpha) = 0, \text{ then } h \leqslant j.$$

Thus, it is possible to decompose $\mathbb{LP}'_i$ into multiple linear programs $\mathbb{LP}'_{i,1}, \ldots, \mathbb{LP}'_{i,k}$, where $\mathbb{LP}'_{i,h}$ consists of the linear constraints

$$\begin{aligned} x_{s,i} &\geqslant c_{s,i} && \text{and} \\ x_{s,i} &\geqslant \sum_{t \in C_h} P(s, \alpha, t) \cdot x_{t,i} + \sum_{u \in C_{>h}} P(s, \alpha, u) \cdot p_{u,i} && \end{aligned}$$

for $s \in C_h$, $\alpha \in \mathsf{Act}(s)$, $\mathsf{rew}(s, \alpha) = 0$. Here, $C_{>h} = C_{h+1} \cup \ldots \cup C_k$ and $(p_{u,i})_{u \in C_j}$ denotes the solutions of $\mathbb{LP}'_{i,j}$. The objective of $\mathbb{LP}'_{i,h}$ is to minimise the sum $\sum_{s \in C_h} x_{s,i}$.

Assuming that the sub-MDP $\mathcal{M}|_{\mathsf{rew}=0}$ of $\mathcal{M}$ resulting by removing all actions α from $\mathsf{Act}(s)$ with $\mathsf{rew}(s, \alpha) > 0$ is acyclic, no linear program needs to be solved using this approach. In this case, the sets $C_1, \ldots, C_k$ are singletons, say $C_h = \{s_h\}$, and the solution $\big(p_{s,i}\big)_{s \in \mathsf{S}}$ is obtained directly when processing the states in reversed topological order $s_k, s_{k-1}, \ldots, s_1$.

5.1.4 Parallel computations

Another direct advantage of the *BP*-approach (as described in Section 5.1.1) is that it allows the usage of parallel computation-techniques. Here, we consider the possibilities to parallelise the calculations within the respective iterations of the *BP*-approach. In principal, there are two opportunities where it is possible to exploit parallelism within such an iteration:

---

[1] The topological sorting can be realised by utilising, e.g., the algorithm of Tarjan (see [Tar72]) over the zero-reward fragment of the given model, and its running-time is therefore linear in the size of this zero-reward fragment.

Positive-reward states

Since in each linear program $\mathbb{LP}'_i$ a positive-reward state has only references to values that were calculated in linear programs $\mathbb{LP}'_j$ with $j < i$, the values for positive-reward states can be computed directly by propagating the known values from those previous iterations. So, there are no cyclic dependencies between two different positive-reward states s_1 and s_2, and it is therefore completely irrelevant if $x_{s_1,i}$ will be computed before $x_{s_2,i}$ in iteration i, or the other way round. Therefore, within $\mathbb{LP}'_i$ all positive-reward states can be calculated completely independently from each other and this allows one to compute all those states in parallel. There is no additional overhead needed, except the overhead that usually arises when employing parallel computation techniques (additional overhead emerges due to the coordination needs of the parallel tasks [Gra+03]).

Zero-reward states

The topological sorting of the zero-reward sub-**MDP**s (see Section 5.1.3) gives the opportunity to calculate the zero-reward sub-**MDP**s (and therefore the zero-reward states) in parallel. For the parallel computation of the zero-reward components it is essential to know which sub-**MDP** refers to values from other sub-**MDP**s. So, the DAG[2] of all sub-**MDP**s will be computed, and whenever there is an edge from component C_i to component C_j inside the DAG, the values of component C_i can not be calculated before the values of C_j are known. But, at the time when C_i refers only to components where all values can be provided, it is then possible to resolve the values for the states of C_i. Therefore, it is possible to calculate all components $C_{i_1}, \ldots, C_{i_k}$ in parallel whenever the values of all referenced components have been already computed previously.

Of course, the determination of the DAG leads to an additional overhead compared to the sequential computation of the zero-reward states. But this needs to be done only once, and the information will be used for each iteration of the *BP*-approach. So, every single iteration will benefit from the DAG-generation, and usually this overhead can be neglected.

As parallel computations usually involve some overhead, the current implementation of the quantile algorithms supports the independent activation of the two different approaches. So, whenever it is the case that the determination of the DAG for the zero-reward states is too expensive, one can only utilise the parallelism for the positive-reward states. Or, if it is the case that there are only a few positive-reward states, but many independent zero-reward sub-**MDP**s, it is supported to only calculate the zero-reward states in parallel. However, the computation statistics for the parallel Explicit-engine presented in Chapter 6 do make use of both parallel approaches.

---

[2] Directed Acyclic Graph, i.e., a directed graph that has no cycles. Therefore, a specific topological order is inherently defined by the DAG.

Therefore, all presented timings for parallel calculations involve the generation of the DAG for the zero-reward sub-**MDP**s.

5.1.5 Multi-thresholds

Assume we fix a specific model and also some quantile query that is of interest. Normally, the aim is then to compute the resulting quantile values when the provided threshold varies. One observation that can be done in this case is that each computation requires exactly the same initialisation phase. And as well the iterative computation steps are the same regardless of the given threshold, since the threshold comes into play for determining the termination criterion of the computation. This observation demonstrates that there is potential to reuse already calculated results for the computation of quantiles with different probability thresholds. This has the benefits of doing the required precomputation steps just once and simply reuse the results for other requested thresholds. Also, the computed (and stored) values can be used for different thresholds and the whole computation will terminate as soon as the results for all demanded thresholds were determined. This allows us to calculate multiple quantile values for a fixed query in just one run. So, instead of calculating quantiles for a number of different thresholds like

$$\begin{aligned}&\mathrm{qu}_s\big(\exists\mathbb{P}_{>0.2}(A\,\mathcal{U}^{\leqslant ?}\,B)\big),\\&\mathrm{qu}_s\big(\exists\mathbb{P}_{>0.5}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)\text{ and}\\&\mathrm{qu}_s\big(\exists\mathbb{P}_{>0.7}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)\end{aligned}$$

separately, it is possible to use a multi-threshold query like

$$\mathrm{qu}_s\big(\exists\mathbb{P}_{>\{0.2,0.5,0.7\}}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)$$

and calculate the demanded quantile values in just one run. This characteristic is of course not restricted to the usage of reachability quantiles and is also applicable for the computation of expectation quantiles.

5.1.6 Multi-state solution methods

Let C denote a zero-reward sub-**MDP** (see Section 5.1.3) that needs to be solved for the linear program $\mathbb{LP}'_i$. When $\mathsf{card}(C) = 1$ then there is only one state that needs to be calculated and all the referenced values of this state have already been calculated earlier (this is ensured by the topological sorting of the zero-reward components). This means that there are no cyclic references involved when solving C, and therefore the computation of C can be carried out in a direct manner.

When on the other hand it is the case that C consists of multiple states, the values for all states of C need to be solved interdependently, since the states of C have cyclic references amongst themselves. Therefore, it is a good choice to utilise multi-state solution methods already known from the literature. The implementation of the presented quantile framework supports the following three different computation methods:

LP-solver

This approach uses the Mixed Integer Linear Programming (MILP) solver lp_solve[3] in order to compute the values for all the states of C. The usage of an *LP-solver* provides the computation of exact solutions, but as a drawback the approach does not really scale (see [For+11a]). So, the usage of this approach may be inappropriate for the analysis of very large models and it is recommended to use one of the other two approaches in this case.

value iteration

Here, *value iteration* is used for the computation of values for the states of C. This approach works by successively iterating the necessary operations on a specific initial vector until the vector has reached a fixpoint. This fixpoint then serves as the demanded result of the computation. *Value iteration* allows fast computations and the memory-requirements are also very moderate, making it a very prominent approach which is rather often used in practice. Further details on *value iteration* can be found in [BK08, Theorem 10.100].

interval iteration

It is stated in [HM14] that some problems may arise when the *value iteration*-approach is utilised in practice. Since there needs to be a specific stopping criterion in order to ensure the termination of the fixpoint computations in practice, the literature proposes to stop the calculations as soon as the difference between the results of the current iteration and the results of the previous iteration is smaller than some user-defined $\varepsilon > 0$ for all states. This leads to the fact that there is no guarantee that the computed values indeed correspond to ε-approximations of the exact values, and therefore influences the accuracy of the computation results in a way that could have problematic consequences. So, in [HM14] the authors came up with an alternative approach called *interval iteration* which allows to tackle this problem. In order to do so *interval iteration* approximates the correct result by approaching from above and from below at the same time, narrowing the correct result within each iteration by some kind of hallway that is getting closer and closer. As soon as both the lower and the upper values meet within an ε-environment, *interval iteration* stops and returns the computed values. This way it is possible to ensure that the computed results are in fact ε-approximations of the requested exact values.

Interval iteration does need several preconditions to work properly (see [HM14] for the details). Therefore, the conditions need to be ensured inside the quantile framework to allow the usage of *interval iteration* as a method for the computation of zero-reward sub-**MDP**s.

So far, *interval iteration* is only supported for the computation of reachability quantiles (see Chapter 3) and thus there is no support for the computation of ex-

---

[3] `http://lpsolve.sourceforge.net/5.5/`, retrieved 28th March 2018

pectation quantiles (see Chapter 4) using *interval iteration*. Therefore, the following considerations refer to the computation of reachability quantiles only.

Universal quantiles In order to compute universal quantiles one needs to apply the calculation of minimal reachability probabilities, and for this case the authors propose to use the min-reduction (see [HM14, Definition 3]). The main idea of this reduction is to avoid reaching the target-states when possible. So, whenever there is an end component which does not contain any target-state the adversary will simply stay in this respective end component. This means that all end components that do not contain any target-states (or, as in the case of lower-reward bounded quantiles the end component does not contain any positive reward) should be merged into a fresh trap state, meaning that its minimal reachability probability is zero. Since those states were already recognised previously by the necessary precomputation of the quantile framework, and their value is declared to be zero for all iterations of the *BP*-approach, there is no need to manipulate the model in such a way. Instead, it is possible to directly apply the *interval iteration*-algorithm [HM14, Algorithm 1] on the zero-reward sub-**MDP**s obtained by the topological sorting.

Existential quantiles When facing the computation of existential quantiles (and therefore maximal reachability probabilities are calculated) a bit more effort is needed, because some kind of model-transformation is required. The basis is built by the max-reduction of [HM14, Definition 7], and the transformation works as follows. All zero-reward end components are collapsed into a designated representative and the necessary computations will be done using the collapsed model. The goal-states of the original model do also need an adaptation, since with probability one each state inside an end component will be reached from any other state as long as the scheduler decides to stay in the respective end component (see [BK08, Lemma 10.119]). Since we collapse the zero-reward end components, each state inside such an end component can be reached without consuming any reward. So, if it is the case that a goal-state is contained in an end component each state of the end component will transition to a goal-state with probability one. Therefore, the representative of such a collapsed end component will be added to the set of goal-states. If on the other hand there is an end component which cannot be left and which has no goal-state, there is no chance of reaching a goal-state from this end component. So, the reachability probability of the representative of this end component will be viewed as zero for the upcoming quantile computations. This transformation allows the usage of the usual quantile calculations on the collapsed model, and whenever there is a zero-reward sub-**MDP**, the adapted form of [HM14, Algorithm 1] for maximum reachability probabilities as described on [HM14, page 11] can be utilised. Of course, in the end we are forced to properly remap the computed results from the collapsed model to the original model in order to assign the results to the appropriate states.

Topological sorting The topological sorting of the zero-reward sub-**MDP**s (see Section 5.1.3) and hence the separate computation of the individual zero-reward sub-models (by applying *interval iteration*) can lead to the fact that the desired accuracy of the computations can not be ensured any longer (see [Bai+17b, Section 4]). The reason is that the ε-approximation of the value for a state s is given by an upper and lower bound of the actual value for the state. For calculating the ε-approximation of a state t that depends on s the approximation of s is needed. When s and t do belong to different zero-reward sub-**MDP**s, and hence the states are examined within separate *interval iteration*-computations, the state t may refer to any value between the upper and lower bound of the approximation for state s. This entails a numerical error of at most ε for the value of t. This error may propagate and become more pronounced in the following. Therefore, it is important to consider the numerous different *interval iteration*-computations in an aggregated way to prevent an accumulation of those numerical errors, since the aim of the whole *interval iteration*-procedure is to guarantee suitably precise approximations of the correct values. [Bai+17b, Section 4] considers this fact for the computation of expected accumulated weights and presents the so-called *topological interval iteration*[4]. The authors propose the construction of a new **MDP** integrating the computed lower-estimates of the demanded values, and one **MDP** that incorporates estimates for the upper bounds from previously performed computations. So, for an upcoming *interval iteration* the previously computed estimates are taken into account and the computations will be performed on the newly constructed models instead of the original model, and as the authors show this ensures that the final result indeed corresponds to an ε-approximation of the exact results after all *interval iteration*-computations have finished, and therefore the user-defined precision will be acquired.

A similar idea is reflected in the quantile framework as well for performing multiple interdependent *interval iteration*s over the topologically sorted zero-reward components of the model (Section 5.1.3). Here, we simply keep the already computed lower-estimates and the upper-estimates for the states obtained by previously performed *interval iteration*s. For this purpose, two value-vectors for the states of the model will be stored, one for the lower-estimates and one for the upper-estimates. Those stored values will be then simply reused whenever the corresponding states are referenced by a component that will be treated currently (the topological order determines the computational sequence of the components). So, when approaching the correct result for the states of a zero-reward component from below, and there occur references to states from other components that have been computed already, the previously calculated lower-estimates will be used for those references (instead of an ε-approximation of these lower-estimates). When on the other hand approaching the results from above,

---

[4] [HH16] also considers the problem for multiple consecutive *value iteration*-computations. There, the authors propose to counteract the computational imprecision by modifying the user-defined ε-precision depending on the given structure of the model under consideration. So, in the end this may result in longer runtimes before the *value iteration* will terminate due to the fact that an increased precision entails a higher number of iteration-steps. For models with huge zero-reward sub-**MDP**s the impact on the computational performance might be significant.

the already known upper-estimates are applied for the corresponding references. The effect of this procedure is that the *interval iteration* references the values exactly the way it would do if no topological order would be assumed, and all states would be computed in one huge *interval iteration*-procedure at once. The difference to an iteration without the topological order is that not all the values will be computed at the same time. Instead, all components can be computed one after another and the number of state-updates is therefore greatly reduced.

5.1.7 Storage for integer sets

By doing the topological sorting of the zero-reward sub-**MDP**s (see Section 5.1.3) and storing the results of the sorting-procedure a serious problem emerges very soon, since one has to struggle with memory constraints very fast when utilising the reference-implementation of the BitSet-class found in the standard-library of the programming language Java[5]. It is essential to store the zero-reward sub-**MDP**s and their topological order in a way that allows efficient access and also grants the ability to be economical with the available memory. So, the utilisation of the quantile framework revealed that there are issues (see Section 6.1.3 or Section 6.2.1), which need to be tackled in order to allow the usage of the presented framework in practice.

The problem with the usage of the standard implementation is that a lot of memory is consumed when the integer set which needs to be represented by a BitSet is sparsely populated and the represented values correspond to huge numbers. This is in fact what happens when storing state indices of the zero-reward sub-**MDP**s used in the context of quantiles. Therefore, the implementation runs into memory problems very soon by storing the topological order. So, a way to fix this problem is to provide a data structure that typically deals with sets that only contain a small number of indices, but as soon as it is required there should be the possibility to increase the storage capabilities. Therefore, the idea is to use a dynamically adapting data structure which uses an ordered array of integers for storing small amounts of elements, and as soon as more and more elements need to be stored, the data structure automatically switches to the BitSet-implementation for storing the increased number of elements.

Since it turned out that normally a zero-reward sub-**MDP** consists of only a few states, the integer-array is usually completely sufficient and therefore the switching to the BitSet-data structure is not needed frequently. As well the different integer sets used for the quantile framework are manipulated only once (during their initialisation), and after that they stay fixed and the sole operation that will be performed with this data structure is an iteration over the set values until the required quantile computation has ended. Therefore, there is no real need for sophisticated manipulation operations for the used sets, and the proposed approach is completely fine for the presented computations. This allows us to use the proposed implementation even for huge models and being efficient in terms of memory and computation times. Another positive effect

---

[5] Since the main parts of Prism were written in Java, it is quite natural to begin with the BitSet-class as a starting point.

from the fact that the stored sets are manipulated only once and stay stable for the rest of the calculations is that the overhead that emerges from keeping the elements in a sorted manner pays off during the computations. One has to ensure the order only once during the initialisation, and there normally exist no operations like adding or deleting elements that may destroy the order after this initialisation. Therefore, the sorting leads to faster searching and iteration methods as soon as the array has been constructed.

5.1.8 Elimination of zero-reward self-loops

Another heuristics that can be integrated to speed up the computation time or to decrease the memory requirements, is for instance, that zero-reward self-loops can be removed by a quantile-preserving transformation $\mathcal{M} \rightsquigarrow \mathcal{M}'$. The **MDP** $\mathcal{M}'$ has the same state space S as $\mathcal{M}$ and the same rewards for all state-action pairs. For $s, t \in \mathsf{S}$ the transition probability function P' of $\mathcal{M}'$ is given by

$$P'(s,\alpha,t) = \begin{cases} \dfrac{P(s,\alpha,t)}{1-P(s,\alpha,s)} & \text{if } \mathsf{rew}(s,\alpha) = 0, t \neq s \text{ and } 0 < P(s,\alpha,s) < 1 \\ P(s,\alpha,t) & \text{else} \end{cases}$$

The following lemma shows that the transformation $\mathcal{M} \rightsquigarrow \mathcal{M}'$ is indeed quantile-preserving:

Lemma 5.1.1 (Soundness of the transformation)**.** *Let $A, B \subseteq \mathsf{S}$ denote subsets of the state space. For all states $s \in \mathsf{S}$ we then have:*

(a) $\mathrm{qu}_s^{\mathcal{M}}(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)) = \mathrm{qu}_s^{\mathcal{M}'}(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B))$

(b) $\mathrm{qu}_s^{\mathcal{M}}(\forall\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)) = \mathrm{qu}_s^{\mathcal{M}'}(\forall\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B))$

Proof. In order to prove statement (a), suppose that $\mathfrak{S}$ is a scheduler for the original **MDP** $\mathcal{M}$ that is optimal for the existential quantile $\mathrm{qu}_s^{\mathcal{M}}(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B))$ and enjoys condition 3.1 in Lemma 3.3.1. As all paths in $\mathcal{M}'$ are paths in $\mathcal{M}$ and the enabled actions in $\mathcal{M}$ and $\mathcal{M}'$ agree, $\mathsf{Act}_{\mathcal{M}}(s) = \mathsf{Act}_{\mathcal{M}'}(s)$ for all $s \in \mathsf{S}$, the scheduler $\mathfrak{S}$ can also be viewed as a scheduler for $\mathcal{M}'$. We consider the case where $\mathcal{M}$ and $\mathcal{M}'$ differ for a single state-action pair (t, α), the general case with multiple state-action pairs then follows from repeated application of the same arguments.

Let (t, α) be the state-action pair where $\mathcal{M}$ and $\mathcal{M}'$ differ, i.e, with $0 < P(t, \alpha, t) < 1$ and $\mathsf{rew}(t, \alpha) = 0$. If $\mathfrak{S}$ never schedules action α for paths ending in t then the probabilities for $A\,\mathcal{U}^{\leqslant r}\,B$ under $\mathfrak{S}$ viewed as scheduler in $\mathcal{M}$ and in $\mathcal{M}'$ are the same for all states s. Suppose now that $\mathfrak{S}(\rho) = \alpha$ for some finite $\mathfrak{S}$-path ρ with $\mathit{last}(\rho) = t$. Then, all finite paths of the form $\rho\,\alpha\,t\,\alpha\,t\,\alpha\,\ldots\,\alpha\,t$ have the same accumulated reward as ρ. By condition 3.1, they are $\mathfrak{S}$-paths too. Almost all infinite $\mathfrak{S}$-paths in $\mathcal{M}$ that start with ρ will eventually take a step $t \xrightarrow{\alpha} t'$ with $t \neq t'$, after having taken the self-loop at t finitely often. In $\mathcal{M}$, the probability of the set consisting of all infinite paths of the form

$$\rho\,\alpha\,t\,\alpha\,t\,\alpha\,\ldots\,\alpha\,t\,\alpha\,t'$$

under $\mathfrak{S}$ is $\mathrm{prob}(\rho)\cdot P'(t,\alpha,t')$, which is the probability of the finite path $\rho\,\alpha\,t'$ in $\mathcal{M}'$. Since no other paths are affected by the switch from $\mathcal{M}$ to $\mathcal{M}'$, this yields that for each state s, the value $\mathsf{Pr}_s^{\mathfrak{S}}(A\,\mathcal{U}^{\leqslant r}\,B)$ does not depend on whether we consider $\mathfrak{S}$ as a scheduler for $\mathcal{M}$ or $\mathcal{M}'$. This yields that the existential quantile in $\mathcal{M}$ is less or equal than in $\mathcal{M}'$:

$$\mathrm{qu}_s^{\mathcal{M}}\big(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)\leqslant\mathrm{qu}_s^{\mathcal{M}'}\big(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)$$

Vice versa, let us suppose that $\mathfrak{S}'$ is a scheduler for $\mathcal{M}'$. We then consider the scheduler $\mathfrak{S}$ for $\mathcal{M}$ where $\mathfrak{S}(\rho)$ is $\mathfrak{S}'(\rho')$ when ρ arises from ρ by erasing all steps $t\xrightarrow{\alpha}t$. Since $\mathsf{rew}(t,\alpha)=0$, the probability $\mathsf{Pr}_s^{\mathfrak{S}}(A\,\mathcal{U}^{\leqslant r}\,B)$ in $\mathcal{M}$ agrees with $\mathsf{Pr}_s^{\mathfrak{S}'}(A\,\mathcal{U}^{\leqslant r}\,B)$ in $\mathcal{M}'$ for all states s. Hence:

$$\mathrm{qu}_s^{\mathcal{M}}\big(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)\geqslant\mathrm{qu}_s^{\mathcal{M}'}\big(\exists\mathbb{P}_{\trianglerighteq p}(A\,\mathcal{U}^{\leqslant ?}\,B)\big)$$

The argument for universal quantiles (statement (b)) is analogous, except that adversarial schedulers are considered. □

The transformation $\mathcal{M}\rightsquigarrow\mathcal{M}'$ indeed can simplify the computation of quantiles by the presented algorithm. This is in particular of interest in combination with the SCC-based decomposition techniques of the LPs $\mathbb{LP}_r'$. If $\mathsf{Pr}^{\max}(A\,\mathcal{U}^{\leqslant r}\,B)$ for all α-successors t' with $t'\neq t$ has already been computed the inequality for state t and α in the linear program reduces to $x_t\leqslant c$ for some constant c. There is even no need to consider state t in $\mathbb{LP}_r'$ when the quantiles for all successors different from s are already known.

5.2 Integration in PRISM

The results stated in the sections of Chapter 3 and Chapter 4 can be transferred into a specific pattern for handling the computation of quantiles. So, in principal we need the following two essential stages:

precomputation: The starting point for the computation of quantiles is a stage determining if the considered quantile has either a finite or an infinite value. This stage is called *precomputation*. The necessary steps needed for the precomputation are determined by the type of the quantile under consideration. The objective of this precomputation-step is to ensure that the subsequent computation-stage will not end in a non-terminating loop. Therefore, the necessary algorithms will be processed to make sure that the quantile exists. If on the other hand the quantile is infinite the precomputation will prevent the execution of the following stage.

For the precomputation-stage, the implementation can rely on the machinery delivered by PRISM for the computation of unbounded minimal / maximal reachability probabilities (see Section 3.3.1), or adapted forms of Prism's methods for finding and analysing the (maximal) end components of the model (see Section 3.4.1 and Section 4.2).

iterative computations: In this stage, the intended computation of the quantile values will be performed by an iterative algorithm using the *back-propagation approach* (see Section 5.1.1). So, the values $x_{s,r}$ will be calculated for all the states $s \in \mathsf{S}$ and for increasing $r = 0, 1, 2, 3, \ldots$ until the specified probability or expectation threshold q has been reached. Since the precomputation has already been completed it is guaranteed that q will be reached and therefore the computation-stage will terminate at some point. Here, all the necessary computations determined by the special type of the quantile will be performed (see the different sections for the details on the respective computations), and after the termination of this stage the demanded quantile value will be returned.

This stage supports all the different computation options presented in Section 5.1. So, it is possible to adapt the computation of quantiles to specific characteristics of the model under consideration by activating or deactivating the desired options.

Each quantile query passes through the two described stages for calculating and providing the demanded results.

Prism is equipped with four different computation-engines in order to perform model-checking calculations, which are MTBDD, Sparse, Hybrid, and Explicit. The first three engines use symbolic techniques based on multi-terminal binary decision diagrams (MTBDDs[6]) either partly or entirely for representing the model under consideration efficiently in terms of consumed memory. The MTBDD-engine relies completely on MTBDDs to perform all the necessary computations, whereas the Sparse-engine utilises sparse matrices built from the MTBDD-representation of the system. The Hybrid-engine is a combination of the first two approaches by representing the system's transition-matrix using MTBDDs and the solution vector for the states of the model is represented explicitly. The Explicit-engine does not use any symbolic techniques for the representation. Instead, the model and the solution-vector for the states will be represented in an explicit manner. So, this engine suffers more from the serious state-explosion problem (see [BK08, Section 2.3]) than the symbolic ones, since the memory needed for the explicit representation of the model might grow exponentially with the model itself. Therefore, it is crucial to optimise the memory consumption of the quantile calculations to such a degree that its usage is practically feasible for the protocols under investigation. Using the techniques presented in Section 5.1 allows us to tune the implementation such that the computations are applicable even for huge models.

The main focus of the implementation of the quantile framework presented here is on the Explicit-engine. But, in order to also support symbolic quantile calculations there exist implementations for the different symbolic engines done by Joachim Klein

---

[6] For a comprehensive introduction into binary decision diagrams and how they can be utilised for performing model-checking techniques it is recommended to see [Par02].

that borrow ideas originally conceived to improve the performance of the EXPLICIT-engine (like, e.g., the *back-propagation approach* presented in Section 5.1.1, or the multi-thresholds from Section 5.1.5). Due to the use of the *BP*-approach the states having a positive reward will be treated by reusing previously calculated values, and the states with no reward at all will be computed using an adapted version of the symbolic *value iteration* implemented for the respective engines of PRISM. The implementation for the MTBDD-engine stores the computed values from previous iterations of the *back-propagation* using one MTBDD per reward bound i. So, all state-action pairs having the same reward are stored using the same MTBDD and will be therefore handled at the same time. Therefore, this engine might be a good choice when there are many state-action pairs, but at the same time only a few different reward values occur in the model. Using the HYBRID-engine the previously calculated values will be stored explicitly, while the transition matrix is stored symbolically. This results in a higher memory usage but at the same time the access to the stored values might be faster compared to the MTBDD-engine. The computations for the SPARSE-engine make use of similar techniques like the HYBRID-engine using sparse matrices for the positive-reward and zero-reward fragments of the model. More details on the computation of the symbolic quantiles can be found in [Kle+16] or [Kle+17].

Due to the fact that the focus of this monograph is on the EXPLICIT-engine the symbolic engines do only provide support for the reachability quantiles as they were discussed in Chapter 3 (currently without support for quantiles under side conditions as proposed in Section 3.6). At the moment there is no tool-support for qualitative reachability quantiles (see Section 3.3.3), or for expectation quantiles (see Chapter 4) using symbolic techniques. For now, only the EXPLICIT-engine of PRISM is equipped with full support for all the various methods of the discussed quantile framework.

5.2.1 Computation of reward-bounded reachability probabilities

Formulas of the form $\mathsf{Pr}^{\max}(\lozenge^{\leqslant r} A)$ are examples for reward-bounded reachability path-formulas over **MDP**s. Here, the standard reachability operator $\lozenge A$ ("eventually a state of A will be reached") is augmented with reward bound r, i.e., of the form $\lozenge^{\leqslant r} A$ ("eventually some state of A is reached while at most a reward of r was accumulated along the way"). The accumulated reward for a given path fragment in the model corresponds to the sum of the rewards that are assigned to the states and actions that comprise the path fragment.

Since the algorithm for the computation of quantiles relies on an iterated computation of reward-bounded reachability probabilities, the employed computation methods are indeed capable of computing reward-bounded reachability probabilities. Only a few adaptations needed to be done in order to correctly support the necessary computations. In contrast to a quantile query a reward-bounded reachability query does not need any precomputation to check if there exist computational impossibilities which may result in a non-terminating loop. Instead, the reward-bounded reachability query is already equipped with a certain bound determining the exact number of iterations that need to be performed in order to calculate the demanded reachability probabilities.

Since there was no tool-support for the calculation of reward-bounded reachability probabilities integrated into Prism, using the quantile backend an efficient support was added. The utilisation of the quantile backend does of course take advantage of all the improvements described in Section 5.1 that were implemented for the computation of quantiles, and this also improves the performance of the computation of reward-bounded reachability probabilities.

Lower reward bounds, i.e., of the form $\Diamond^{\geqslant r} A$, are supported as well.

5.2.2 Computation of quantiles in CTMCs

Since Prism currently does not support the computation of reward-bounded reachability probabilities in **CTMC**s, the implementation of the described approach of Section 3.7 in Prism only supports quantiles with upper or lower time bounds on simple path formulas for now. But once support for reward-bounded reachability probabilities is added to Prism (e.g., using the duality between reward- and time-bounds [Bai+00]), the implementation for quantile computations can easily be adapted to also handle arbitrary reward-bounded quantiles in **CTMC**s. Both, the Explicit and the symbolic engines, are equipped with an implementation of the scheme presented in Section 3.7.

To give a brief example of the performance of this computation scheme, an instance of the "tandem" case study[7] from the Prism benchmark suite is considered here, with parameter c set to 10. To obtain an ε-approximation of the quantile value for a precision of $\varepsilon = 10^{-6}$ and the quantile

$$\inf\big\{t \in \mathbb{R} \,:\, \mathsf{Pr}_s\big(\Diamond^{\leqslant t} \text{“network becomes full”}\big) \geqslant 0.1\big\},$$

the exponential search requires 14 probability computations to find the upper bound 4 096, which is then refined by the binary search using 31 additional probability computations to obtain the result $t = 2\,954.281344$. The overall computation time was 106.41 s (Explicit), 5 224.89 s (MTBDD), 42.75 s (Hybrid) and 20.44 s (Sparse). In order to relate the timings, the computation of the probability for the result of the quantile computation, i.e.,

$$\mathsf{Pr}_s\big(\Diamond^{\leqslant t} \text{“network becomes full”}\big) \text{ for } t = 2\,954.281344,$$

takes 3.15 s (Explicit), 153.9 s (MTBDD), 1.27 s (Hybrid) and 0.61 s (Sparse).

The computation of time-bounded probabilities for **CTMC**s in Prism relies on the computation of a finite sum using the uniformised **DTMC** (see, e.g., [Bai+03; Par02]), where the number of summands is chosen depending on a user-supplied value for the desired precision. In general, it can be expected that computations with a coarser precision require fewer iterations in the computation of the sum. Therefore, experiments with an approach that gradually refines the precision for the probability computations were done: Start with a coarse precision, e.g., allowing imprecision of up to 0.1. As long as there are definitive results for the probability-threshold computations when

---

[7] `http://www.prismmodelchecker.org/casestudies/tandem.php`, retrieved 28th March 2018

taking the imprecision of the result value into account, just keep the same precision for the upcoming computations. If at some point there occurs an inconclusive result, i.e., the threshold p lies inside the possible values when taking the imprecision into account, the precision can be refined, e.g., by a division by 10. Unfortunately, the realised experiments indicate that the potential savings in runtime due to the coarser precision tend to be negated by the required additional probability computations during the refinement step when encountering an inconclusive result. The reason is that there currently exists no possibility for reusing the already carried out computations done by the coarser precision when refining the precision for the same time bound. This entails that the very same computations need to be done multiple times. For example, when considering an initial precision of 0.1 and a desired precision of $\varepsilon = 10^{-6}$ for the quantile value, the computation time for the EXPLICIT-engine has increased to 118.83 s. Of course, the number of iterations slightly increased as well: The exponential search now required 15 computations since the precision needs to be set to 0.01 at one point, and the binary search used 35 additional computations. So, in total 5 additional iterations are necessary to perform the calculations (this is clear since there need to be 5 refinements in order to improve the precision from 0.1 to 10^{-6}).

Each of those additional iterations could be improved by reusing the information already computed previously for a coarser precision. Therefore, as a possible improvement, it is suggested to integrate the threshold check into the computation of the time-bounded reachability probabilities in PRISM using an adaptive precision as future work: During the computation of the sum, a periodic check against the threshold could be carried out, taking the achieved bound on the precision into account and returning early if it can be conclusively determined that the threshold is satisfied or can never be satisfied.

6 Analysed Protocols

Here, the applicability of the shown quantile-computation schemes is presented by means of different analysed protocols. We start in Section 6.1 by presenting a selection of already well-known protocols from Prism's benchmark suite [KNP12], demonstrating the feasibility of a quantile-based multi-objective analysis using the framework developed throughout this monograph. Afterwards, in Section 6.2 we will analyse protocols where the primary goal is on the optimisation of their energy consumption in such a way that the waste of energy will be minimised and that the utility generated by the protocol does not suffer from this optimisation. The characteristics of those protocols are of interest and also serve to show the applicability of the developed framework for different situations that arise in complex scenarios, and it is demonstrated how the analysis results of the quantile framework help to organise systems more efficiently in terms of consumed energy.

All the following presented calculations were carried out on a server system equipped with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384 GB of RAM in total. For each respective calculation a specific memory limit needed to be specified in order to guarantee that all the calculations could be done without any swapping effects. For reducing variations in the computation times and in order to deliver reproducible statistics, techniques like Hyper-Threading [Mar+02] or Turbo Boost[1] were disabled for all the computations.

Since the calculations were carried out for both the Explicit-engine of Prism and the different symbolic engines (see Section 5.2), the automated variable reordering described in [Kle+16] and [Kle+17] was executed beforehand. This technique aims at reducing the size of the MTBDD representation of the model under consideration by rearranging its state variables. Therefore, this enables the symbolic engines to reduce their memory consumption and it is very likely that the performed computations happen much faster. This does not influence the behaviour of the Explicit-engine. Due to the fact that the models of Section 6.1 were taken from Prism's benchmark suite [KNP12] the size-reduction gained by the variable reordering for the models of this section turns out to be marginal. The reason might be that the authors of the models already tried to minimise their symbolic representations by using different heuristics[2], since the symbolic engines were mainly used for the calculations presented in the respective cases. But, since the models introduced in Section 6.2 were crafted inside the formal-methods group headed by Christel Baier, the utilisation of the tools

---

[1] see `https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html`, retrieved 28th March 2018

[2] Section 4.1.2 and Section 4.2.2 of [Par02] present more details on heuristics that help to find good orders of the variables.

described in [Kle+16] and [Kle+17] is recommended before executing the computations using the symbolic engines. This improves the analysis of the presented queries using the symbolic calculations.

All graphical representations of the computation results depicted in the current chapter were generated with the help of the Python-based framework Matplotlib[3] originally created by John D. Hunter [Hun07].

Note on the presentation of the computation statistics Due to space constraints the presented tables depicting the statistics of the analyses are divided into two parts. One depicts the performance of the Explicit-engine (Table 6.1), and the other contains the statistics for the three symbolic engines Hybrid, Sparse and MTBDD (Table 6.2).

Table 6.1: Columns for the statistics of explicit quantiles

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}

Table 6.2: Columns for the statistics of symbolic quantiles

	Model				symbolic computations					
					Hybrid		Sparse		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}

The column Instance depicts the analysed instance of the protocol, the second column (States) gives the size of the reachable state space for the Markovian model. Since the symbolic engines do represent their model by means of binary decision diagrams there exists the additional column Mtbdd, which denotes the size of the MTBDD-nodes. The column t_{build} stands for the time that was needed in order to construct the Markovian model out of the formal description using Prism's guarded-command input language. Iters gives the number of iterations that were needed by the back-propagation approach (see Section 5.1.1) in order to compute the desired quantile values. Since there needs to be a precomputation the column t_{pre} shows the time in seconds that was needed for this stage of the computation. Finally, t_{query} depicts the overall time that the complete quantile computation needed in order to provide the desired results. The time for the precomputation (t_{pre}) is included in t_{query}. Since all three symbolic engines at first build a symbolic representation of the model using the same MTBDD-techniques, Table 6.2 shows only one column for t_{build}. The different analysed protocols showed that all realised computations delivered nearly the same building-time for all three symbolic engines (the highest deviation for building-times over 1 second was below 1.2%), and therefore the differences between the symbolic building-times are negligible. Therefore,

---

[3] `https://matplotlib.org/`, retrieved 28th March 2018

the column t_{build} in tables for symbolic computations depicts the building-time for the MTBDD-engine. Table 6.1 for the EXPLICIT computations shows only one column for the precomputation. The reason is simply that the sequential and the parallel computations do exactly the same up to the point when they have finished their precomputation phase, and thereafter the methods differ in their behaviour. If not stated otherwise the column for the parallel computation always depicts the timings when using a parallelity of six, meaning that the algorithm splits its tasks among six processors in order to perform the independent parts of the necessary calculations simultaneously.

6.1 PRISM Benchmark Suite

This section presents a small selection of protocols obtained from PRISM's benchmark suite [KNP12]. It is shortly described how the respective protocols work and subsequently the results of the multi-objective quantile-based analysis are shown. Since the models of the selected protocols have different structural characteristics, they do present challenges for various aspects of the implementation and therefore serve as a presentation for the applicability of the developed quantile framework. This also shows the performance of the implementation when situations vary and therefore the demands on the implementation change. A comparison between the different implemented approaches (see Section 5.2) will be presented as well.

6.1.1 Self-Stabilising Protocol

The aim of the Self-Stabilising Protocol by Israeli and Jalfon [IJ90] is to establish a stable configuration within a finite number of steps autonomously without instrumenting or influencing this procedure from the outside. This protocol is modelled[4] as an **MDP** for N equal processes organised in a ring structure. Initially, the N processes all are active, i.e., have a token assigned to each of them. A stable configuration in this setting is reached when there is only one process left in the entire ring structure controlling the last remaining token. In each step an active process randomly sends tokens to its left or right neighbour and receives tokens from its neighbours. If, on this occasion, one process ends up with multiple tokens all the tokens from the process will be merged into one single token. This procedure will go on until there is only one token left and the ring is therefore in a stable state.

The quantile framework is used here for computing the minimal number of steps required for reaching a stable state with probability of at least p for some schedulers (existential quantile) or all schedulers (universal quantile). The latter problem has as well been answered in the referred PRISM case study by iteratively increasing the step bound until the demanded probability bound p was met. Each adjustment of the step bound hereby involves a separate query that needs to be computed. The approach

---

[4] `http://www.prismmodelchecker.org/casestudies/self-stabilisation.php#ij`, retrieved 28th March 2018

using quantiles is more elegant by implicitly computing the probability values and answering only one (quantile) query.

Formally, the desired quantiles can be stated as

$$\begin{aligned} &\mathrm{qu}_s\big(\,\exists\mathbb{P}_{\geqslant p}(\Diamond^{\leqslant ?}\mathrm{Stable})\,\big) && \text{and} \\ &\mathrm{qu}_s\big(\,\forall\mathbb{P}_{\geqslant p}(\Diamond^{\leqslant ?}\mathrm{Stable})\,\big), \end{aligned}$$

where s is the initial state in which each process holds its own token and where Stable is the set of all configurations where there exists only one process owning the last remaining token. The results of the computations for the existential and universal quantiles for $N \in \{3, 4, 5, 6, 7, 8, 9, 10, 12, 16, 20, 24\}$ with probability bounds $p \in \{0, 0.01, 0.05, 0.1, \ldots, 0.95, 0.99\}$ are depicted in Figure 6.1, and the corresponding performance metrics can be found in Table 6.4 (explicit computations) on page 92, and in Table 6.5 (symbolic computations) on page 93 for the existential quantile. In Table 6.6 (explicit computations) on page 94 and Table 6.7 (symbolic computations) on page 95 the statistics for the universal quantile are depicted. Those tables do

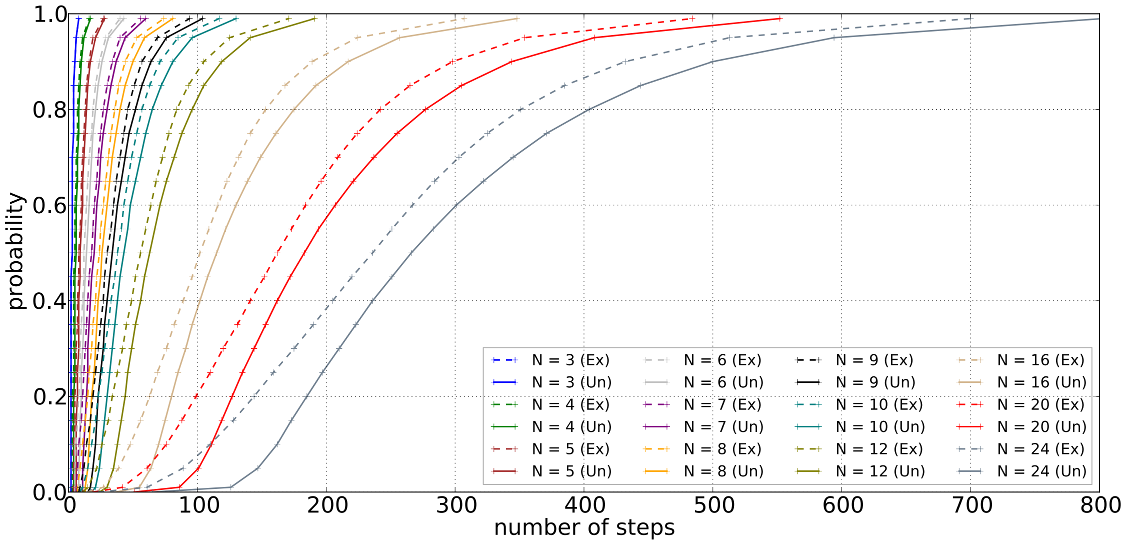

Figure 6.1: Results for Self-Stabilising Protocol

show the performance when utilising the back-propagation approach described in Section 5.1.1 for the different computation engines available in Prism. As can be seen the performance of the Hybrid-engine is nearly the same as it is for the Sparse-engine. Another eyecatcher is that the Explicit-engine suffers most from the time needed in order to construct the model. While the different symbolic engines do only need a few milliseconds in order to build the model, the time for constructing the explicit model corresponds to half an hour for $N = 24$ processes. The time for the necessary precomputation which relies on the methods provided by Prism to compute minimal / maximal reachability probabilities is also noticeable for the Explicit-engine, whereas the calculations for the symbolic engines do not even need one second for the computation of maximal reachability probabilities. But, the effective calculation of the quantiles using the EXPLICIT-engine happens very fast when utilising the parallel computation techniques (see Section 5.1.4). In a way this behaviour is surprising since the symbolic representation of the model is extremely compact (1 629 MTBDD-nodes represent 16 777 215 reachable states in the model for $N = 24$), and therefore one is willing to expect that the symbolic engines do outperform the explicit computations easily. Using the MTBDD-engine for $N = 24$ even led to a timeout, since the computations needed more than 24 hours. Another interesting observation is that the computation times for the universal quantile using the EXPLICIT-engine are also much higher compared to the existential quantile. The reason for this is that the time for the precomputation is much higher than it is for the existential quantile (for $N = 24$ the precomputation for the universal quantile needs 4 629.97 s, whereas the existential quantile needs 227.11 s). For the symbolic engines the times for the precomputation are also higher than for the existential quantile, but not to such an extent. Therefore, as expected the overall computation times for the symbolic engines are slightly increased for the universal quantile when compared to the existential quantile, but mainly due to the fact that the universal quantile needs a few more iterations than the existential one.

Please note that the part of the plot in Figure 6.1 that considers the results for the universal quantiles, presents almost the same information as the plot for the minimal probability of reaching a stable configuration presented in PRISM's model description[5]. But there the parameterisation was done over the number of steps, in contrast to the parameterisation over the probability bound p typically done using quantiles.

Note on the utilisation of the *LP*- and the *BP*-approach A comparison between the back-propagation approach (*BP*) and an approach where the whole linear program is solved by using an external LP-solver (*LP*) can be found in Table 6.3. For the *BP*-approach the sequential EXPLICIT-engine was utilised. The time for *BP* covers the entire computation of the respective quantile values, including the time needed for the necessary precomputation. For an appropriate comparison the underlying linear program used for *LP* is arranged in such a way that it only consists of the iterations that need to be considered in order to obtain the requested quantile value. So, the depicted time for *LP* corresponds to the time needed for solving the linear program $\mathbb{LP}_r$ (see Section 5.1.1) with r being the result for the requested quantile. In [UB13], a theoretical upper bound r_{max} for the reward was established that is exponential in the size of the model. For the given bound it is guaranteed that solving $\mathbb{LP}_{r_{\mathsf{max}}}$ yields the desired quantile. However, for the protocol under consideration it can be observed that this theoretical upper bound r_{max} is hardly reached and usually much greater than the actual calculated quantile value r. For the evaluation of the naïve *LP*-approach $\mathbb{LP}_r$ rather than $\mathbb{LP}_{r_{\mathsf{max}}}$ is considered, as it is known that solving $\mathbb{LP}_r$

---

[5] see `http://www.prismmodelchecker.org/casestudies/self-stabilisation.php#ij`, retrieved 28th March 2018

Table 6.3: Comparison between *LP*- and *BP*-approach (Self-Stabilising Protocol)

	Model			Existential Quantile		
Instance	States	t_{build}	p	Result	*BP*	*LP*
N = 5	31	0.02 s	0.1	4	0.07 s	0.09 s
			0.5	8	0.07 s	0.09 s
			0.99	27	0.08 s	0.22 s
N = 10	1 023	0.09 s	0.1	18	0.14 s	95.18 s
			0.5	38	0.15 s	712.28 s
			0.99	117	0.17 s	8 144.49 s
N = 15	32 767	1.08 s	0.1	42	1.03 s	> 6 h
			0.5	89	1.61 s	> 6 h
			0.99	270	3.72 s	> 6 h
N = 20	1 048 575	53.57 s	0.1	76	40.37 s	> 6 h
			0.5	162	69.81 s	> 6 h
			0.99	484	173.11 s	> 6 h

is sufficient for computing the desired quantile. The computational effort for solving $\mathbb{LP}_{r-1}$, $\mathbb{LP}_{r-2}$, … that would normally occur in the iterative scheme is also ignored, since the correct quantile value r is known by a prior analysis using the *BP*-approach. Therefore, the LP-solver only computes one linear program that is as huge as necessary, and iterations that do not contribute to the result are not considered. As can be seen the *BP*-approach clearly outperforms the *LP*-approach, and that the *LP*-approach turns out to be infeasible already for very small instances of the model and queries where several iterations are needed. Already for $N = 15$ processes, the *LP*-approach exceeded the timeout bound of 6 hours, whereas the *BP*-approach is still applicable with computation times of only a few seconds. Table 6.3 also reveals that especially within the *LP*-approach the time spent for evaluating the quantile increases significantly when the probability bound p is high (and hence, also the corresponding quantile value is high). So, for a relatively small model consisting of only 1 023 states ($N = 10$) the computation-times increase from 95 seconds for $p = 0.1$ to more than 2 hours for $p = 0.99$. In summary, it can be stated that it is recommended to not use the *LP*-approach for huge models that normally occur when utilising a formal analysis based on probabilistic model checking.

Note on the implementation of step-bounded until For quantiles using a step-bounded until, the implementation of the *BP*-approach for reward-bounded until is utilised by assigning a reward of 1 to each state of the model. As a consequence there is no further need of solving any linear program for zero-reward cycles, since every state holds a positive reward. This procedure allows the direct application of the *BP*-approach for all the states inside the model without using such a method as *value iteration*, *interval iteration* or an external *LP-solver*. Another advantage is that each state needs to reference only those values in iteration $i > 0$ that were computed previously during iteration $i - 1$, and therefore a reward window of size 1 (see Section 5.1.2) is sufficient for each state.

Note on parallel computations using the Explicit-engine Since it is possible to enable parallel computations for the calculation of quantiles in the EXPLICIT-engine, we do consider the performance of this approach by looking at the resulting computation times. Figure 6.2 shows the running times for computing the explicit existential quantile for $N = 24$. The plot depicts the running times together with the time necessary for the precomputation used by the quantile query. Since the parallel approach does only influence the iterative computations determining the quantiles the time for the precomputation is not influenced by any parallelism at all. So, the precomputation time serves as some kind of lower bound for the time necessary for doing the quantile-based analysis. Keep in mind that the considered quantile uses a step-bounded until. Therefore, each state of the model holds a positive reward (see previous paragraph), and because of this fact there is no necessity of generating a DAG for the zero-reward components of the model. So, the parallel computations can be done without providing any further requirements (see Section 5.1.4).

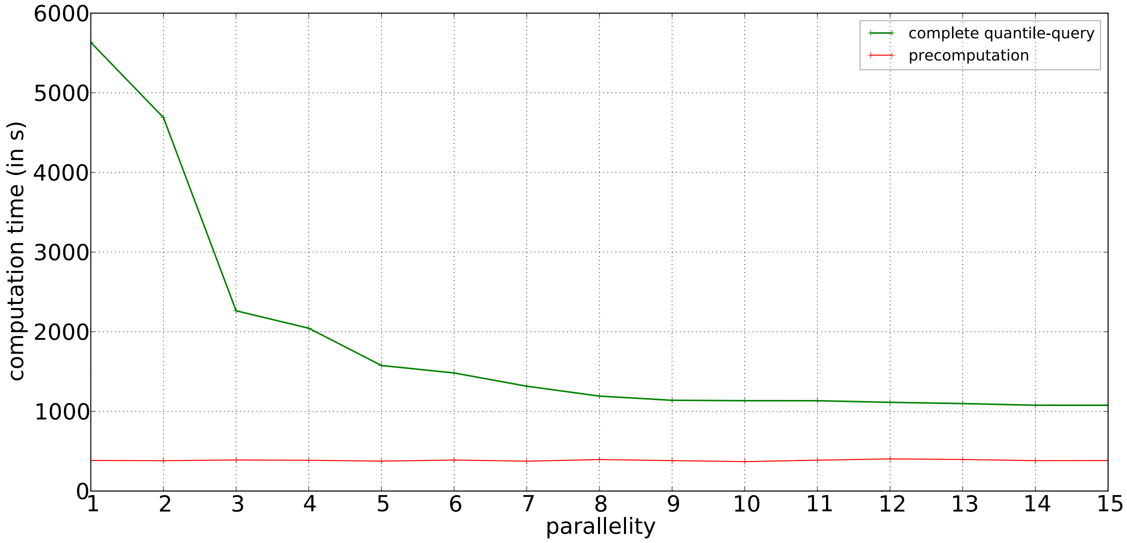

Figure 6.2: Parallel computations for Self-Stabilising Protocol

Due to the fact that the utilised computing device has 16 physical cores the degree of parallelism was restricted to at most 15 in order to keep the device responsive for important operating-system tasks. As can be seen the usage of parallelism scales really well for this model and the normally occuring overhead of parallel computations is negligible due to the gained performance out of the parallel utilisation. Another message of Figure 6.2 is that an increase of the parallelity to more than 8 does not really boost the computations that much, but the involved tasks do not interfere with each other in a way that has a negative effect on the computation such that it gets noticeably slower.

Table 6.4: Statistics for explicit existential quantile (Self-Stabilising Protocol)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 3	7	0.02 s	9	0.01 s	0.01 s	0.01 s
N = 4	15	0.02 s	17	0.01 s	0.01 s	0.01 s
N = 5	31	0.02 s	28	0.01 s	0.01 s	0.02 s
N = 6	63	0.03 s	41	0.01 s	0.01 s	0.02 s
N = 7	127	0.04 s	57	0.01 s	0.02 s	0.03 s
N = 8	255	0.04 s	75	0.01 s	0.03 s	0.04 s
N = 9	511	0.06 s	95	0.01 s	0.04 s	0.06 s
N = 10	1 023	0.09 s	118	0.03 s	0.09 s	0.08 s
N = 12	4 095	0.22 s	172	0.08 s	0.34 s	0.24 s
N = 16	65 535	2.63 s	308	0.55 s	9.39 s	4.16 s
N = 20	1 048 575	53.13 s	485	10.81 s	185.79 s	62.12 s
N = 24	16 777 215	2 210.48 s	701	227.11 s	5 633.58 s	1 796.87 s

Table 6.5: Statistics for symbolic existential quantile (Self-Stabilising Protocol)

	Model				symbolic computations					
					Hybrid		Sparse		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 3	7	36	0.02 s	9	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.01 s
N = 4	15	56	0.02 s	17	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.01 s
N = 5	31	103	0.02 s	28	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.01 s
N = 6	63	141	0.02 s	41	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.02 s
N = 7	127	199	0.02 s	57	< 0.01 s	0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.05 s
N = 8	255	262	0.02 s	75	< 0.01 s	0.01 s	< 0.01 s	0.01 s	< 0.01 s	0.1 s
N = 9	511	309	0.02 s	95	< 0.01 s	0.01 s	< 0.01 s	0.01 s	< 0.01 s	0.24 s
N = 10	1 023	369	0.02 s	118	< 0.01 s	0.03 s	< 0.01 s	0.02 s	< 0.01 s	0.6 s
N = 12	4 095	501	0.03 s	172	0.01 s	0.14 s	0.01 s	0.12 s	0.01 s	4.56 s
N = 16	65 535	813	0.03 s	308	0.03 s	4.84 s	0.02 s	4.11 s	0.03 s	282.18 s
N = 20	1 048 575	1 189	0.04 s	485	0.07 s	161.08 s	0.09 s	131.24 s	0.09 s	12 492.45 s
N = 24	16 777 215	1 629	0.04 s	701	0.19 s	4 678.47 s	0.27 s	4 144.24 s	0.28 s	> 24 h

Table 6.6: Statistics for explicit universal quantile (Self-Stabilising Protocol)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 3	7	0.02 s	9	0.05 s	0.07 s	0.08 s
N = 4	15	0.02 s	18	0.05 s	0.08 s	0.09 s
N = 5	31	0.02 s	29	0.06 s	0.08 s	0.1 s
N = 6	63	0.03 s	44	0.06 s	0.09 s	0.12 s
N = 7	127	0.04 s	61	0.06 s	0.1 s	0.14 s
N = 8	255	0.04 s	82	0.07 s	0.12 s	0.16 s
N = 9	511	0.06 s	105	0.08 s	0.14 s	0.19 s
N = 10	1 023	0.09 s	131	0.1 s	0.19 s	0.21 s
N = 12	4 095	0.22 s	192	0.16 s	0.46 s	0.36 s
N = 16	65 535	2.63 s	349	0.83 s	9.63 s	3.91 s
N = 20	1 048 575	53.13 s	553	27.77 s	215.83 s	91.31 s
N = 24	16 777 215	2 210.48 s	804	4 629.97 s	9 711.2 s	6 142.87 s

Table 6.7: Statistics for symbolic universal quantile (Self-Stabilising Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 3	7	36	0.02 s	9	< 0.01 s	0.02 s	< 0.01 s	0.01 s	< 0.01 s	0.02 s
N = 4	15	56	0.02 s	18	< 0.01 s	0.02 s	< 0.01 s	0.02 s	< 0.01 s	0.03 s
N = 5	31	103	0.02 s	29	< 0.01 s	0.02 s	< 0.01 s	0.02 s	< 0.01 s	0.03 s
N = 6	63	141	0.02 s	44	< 0.01 s	0.02 s	< 0.01 s	0.02 s	< 0.01 s	0.05 s
N = 7	127	199	0.02 s	61	< 0.01 s	0.02 s	< 0.01 s	0.02 s	< 0.01 s	0.08 s
N = 8	255	262	0.02 s	82	< 0.01 s	0.02 s	< 0.01 s	0.02 s	< 0.01 s	0.16 s
N = 9	511	309	0.02 s	105	0.01 s	0.03 s	< 0.01 s	0.03 s	0.01 s	0.35 s
N = 10	1 023	369	0.02 s	131	0.01 s	0.05 s	0.01 s	0.04 s	0.01 s	0.87 s
N = 12	4 095	501	0.03 s	192	0.02 s	0.19 s	0.02 s	0.17 s	0.03 s	5.79 s
N = 16	65 535	813	0.03 s	349	0.21 s	5.61 s	0.21 s	4.92 s	0.22 s	372.45 s
N = 20	1 048 575	1 189	0.04 s	553	3.04 s	188.0 s	3.02 s	152.09 s	3.05 s	16 890.45 s
N = 24	16 777 215	1 629	0.04 s	804	37.81 s	5 507.57 s	37.93 s	4 632.54 s	38.03 s	> 24 h

6.1.2 Leader-Election Protocol

Leader-election protocols aim to elect a leader, i.e., a uniquely designated process out of N equal processes organised in a ring structure. Each process can hereby send messages to the other processes. A synchronous and an asynchronous variant of such a protocol, both developed by Itai and Rodeh [IR90] and also described in the context of probabilistic model checking within the benchmark suite of Prism [KNP12], is considered here. The emphasis of the upcoming analysis is on the minimal number of rounds or steps that is necessary in order to successfully elect a leader with a desirable probability. For this purpose quantiles with reward-bounded until properties are used for the realisation of the analysis.

Asynchronous leader election In the asynchronous setting (see Section 3 in [IR90]) the processes are located in a ring, and initially the processes are all active. Inactive processes are able to still pass along messages to their neighbours. 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 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. Otherwise the protocol proceeds with a further round.

The **MDP** model of the Asynchronous Leader-Election Protocol obtained from Prism's benchmark suite[6] will be analysed using the presented quantile framework.

Synchronous leader election The synchronous variant of the protocol (see Section 2 in [IR90]) fixes a number K of probabilistic choices, where in each round every process selects an ID from the set of allowed IDs $\{1, \dots, K\}$ uniformly. This ID will be then synchronously passed over the ring. If a process has chosen an ID which is unique, the process with the maximal unique ID will be the elected leader. As long as there is no unique ID, a new round starts where the processes choose their ID randomly from $\{1, \dots, K\}$ again.

The analysis will be carried out on the Markov chain model of the Synchronous Leader-Election Protocol from Prism's benchmark suite[7].

Analysis results A very important quantitative measure for both, the synchronous and the asynchronous variant of the protocol, is the minimal number of rounds r required to elect a leader with a certain probability p for some/all schedulers. Note that in a Markov chain, the probabilities to elect a leader agree for all schedulers, i.e., the minimal number of rounds r is the same for all schedulers in the synchronous

---

[6] `http://www.prismmodelchecker.org/casestudies/asynchronous_leader.php`, retrieved 28th March 2018

[7] `http://www.prismmodelchecker.org/casestudies/synchronous_leader.php`, retrieved 28th March 2018

protocol variant. As noticed in the Prism case study, this is also the case for the asynchronous variant, although the considered model is an **MDP**. Hence, only the following existential quantile is of interest, whose value agrees with the value of the universal quantile:

$$\mathrm{qu}_s\big(\,\exists\mathbb{P}_{\geqslant p}(\Diamond^{\leqslant ?}\mathrm{LeaderElected})\,\big),$$

interpreted over step-bounded reachability and round-bounded reachability. Besides reasoning over quantiles with step-bounded until properties, as it was done in the previously presented Self-Stabilising Protocol (see Section 6.1.1), the utilised quantile framework also allows for quantiles with reward-bounded until properties over arbitrary reward functions. Therefore, it is possible to reason about the minimal number of rounds by using such a tailored reward function. Figure 6.3 shows the experimental results for $N \in \{3, 4, 5, 6\}$ processes, $K \in \{2, 4, 6, 8\}$ and probability bounds $p \in \{0, 0.1, 0.2, \ldots, 0.9, 0.95, 0.99\}$ for the synchronous protocol-variant. It can be seen that with increasing K the necessary rounds respectively steps become smaller for successfully electing a leader. So, an increase in the parameter K enables the protocol to terminate earlier. The number of involved processes does not play such an important role on the outcome of the analysis, instead the highest impact on the number of needed operational steps is determined by an appropriate choice of K. Figure 6.3 delivers the exact same message as the plots presented in `http://www.prismmodelchecker.org/casestudies/synchronous_leader.php#mc` (retrieved 28th March 2018) for computing the bounded reachability query

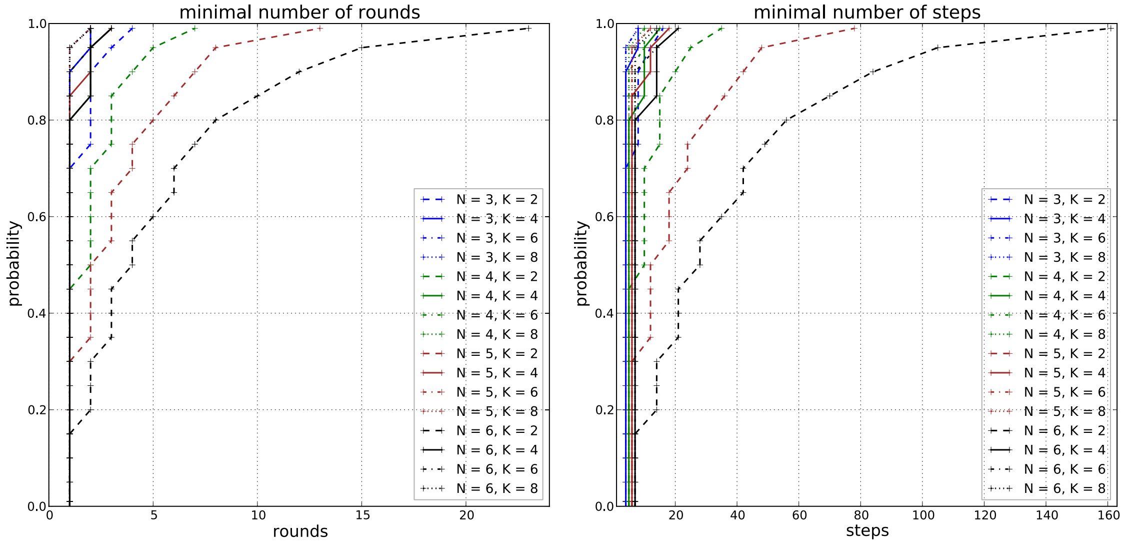

Figure 6.3: Results for Synchronous Leader-Election Protocol

$$\mathsf{Pr}(\Diamond^{\leqslant L\cdot(N+1)}(s_1 = 3 \wedge \ldots \wedge s_N = 3)) = ?$$

with the upper bound $L \cdot (N + 1)$ being altered by using multiple values for L (for the model under consideration N is already fixed as the number of involved processes).

The product in the upper bound was used in order to emulate the number of rounds, since in the protocol a round consists of $N + 1$ steps. So, instead of needing some smart encoding like this product, the quantile framework allows to calculate the results by just specifying the desired reward function. And instead of solving multiple queries for altered L the calculation of just one query is completely sufficient thanks to the usage of multiple thresholds (see Section 5.1.5).

The statistics for the calculation of the minimal number of rounds for the synchronous variant of the protocol can be found in Table 6.9 on page 101 for the operation of Prisms Explicit-engine, whereas Table 6.10 on page 102 shows the corresponding statistics for the different symbolic engines. The statistics for the minimal number of steps for the Synchronous Leader-Election Protocol is depicted in Table 6.11 (Explicit-engine) on page 103 and in Table 6.12 (symbolic engines) on page 104. As can be seen the Hybrid-engine has some issues with this protocol, whereas the Explicit-engine performs really well here. A factor that surely contributes to the higher run times for the symbolic engines is the size of the binary decision diagram encoding the state space. The number of the MTBDD-nodes is much higher than the actual number of reachable states. For this protocol it is even the case that the parallel computation turns out to be slower than the sequential computation of the Explicit-engine for most of the considered models. An explanation is that the state space is relatively small and there are just a few quantile iterations necessary in order to compute the desired results. So, the additional overhead introduced by instrumenting the parallel computations does not really pay off in this case. There are simply not enough iterations required (and along with this computation-tasks) to compensate for the overhead.

Figure 6.4 depicts the results for the asynchronous variant of the protocol for $N \in \{2, 3, \ldots, 9\}$ processes. As can be seen the plot for the minimal number of

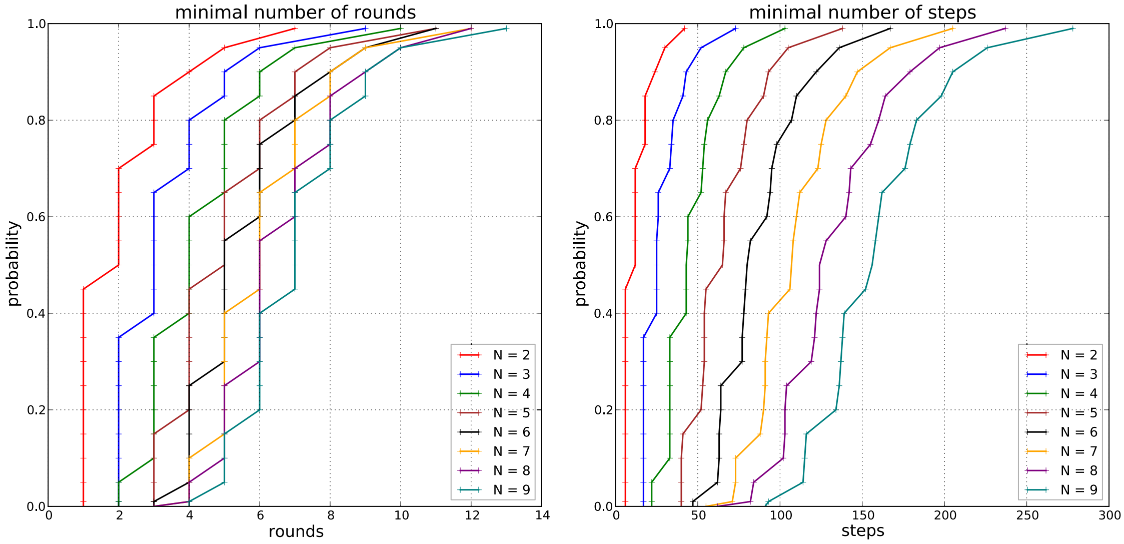

Figure 6.4: Results for Asynchronous Leader-Election Protocol

steps corresponds to the plot presented in `http://www.prismmodelchecker.org/casestudies/asynchronous_leader.php#mc` (retrieved 28th March 2018) depicting the minimum/maximum probability of electing a leader within k steps. There, the plot was generated by computing the bounded reachability query

$$\mathsf{Pr}^{\min}(\lozenge^{\leqslant k}(s_1 = 4 \vee \ldots \vee s_N = 4)) =?$$

such that the parameter k was changed during multiple model-checking runs. Using the discussed quantile framework, the result could be generated by calculating just one single quantile query specifying multiple thresholds (see Section 5.1.5).

Table 6.13 on page 105 depicts the computational performance for the calculation of the minimal number of rounds for the asynchronous protocol-variant using the Explicit-engine, whereas Table 6.14 depicts the statistics when applying the different symbolic engines. The corresponding performance for the minimal number of steps can be found in Table 6.15 (Explicit-engine) on page 106, and in Table 6.16 (symbolic engines) on the same page. It is obvious that the building process of the model using the Explicit-engine is a bottleneck again. Whereas the models for the symbolic engines were constructed within a few seconds, the Explicit-engine needs up to 13 minutes. Due to the number of states (167 748 115) it was not possible to build the model for $N = 9$ using the Explicit-engine. Therefore, the calculations were only possible up to $N = 8$, and results for $N = 9$ could only be generated when utilising one of Prisms symbolic engines. One interesting fact is that, when considering the minimal number of rounds, the parallel Explicit-engine is slower than the sequential engine. This observation will be discussed in more detail in the upcoming paragraph. The explanation can be found in the necessity of generating the DAG (see Section 5.1.4) of the zero-reward components in combination with the small number of required iterations. When instead considering the number of steps each state of the model has a positive reward, and therefore there is no need for computing a DAG for the zero-reward states. As well the number of required iterations is much higher (238 for $N = 8$) and therefore the parallelism is used more frequently. So, the parallel Explicit-engine is really fast in this case.

Note on parallel computations using the Explicit-engine As already stated earlier, the parallel computation (using a parallelism of six) of the minimal number of rounds for the asynchronous variant of the protocol is slower than the sequential quantile computation. Therefore, it is of interest how the computations for the asynchronous variant perform for different degrees of parallelism. Figure 6.5 depicts the computation performance for the analysis of the minimal number of rounds for the asynchronous variant in the case $N = 8$. As it was also done for the Self-Stabilising Protocol (see Section 6.1.1), the degree of parallelism was restricted to at most 15 in order to preserve the reactivity of the computing device. It can be seen in the figure that there is no benefit of utilising the parallel computations. Instead, all the parallel executions are slower than the sequential one. The reason is that the generation of the DAG (see Section 5.1.4) for the zero-reward components needs a considerable amount of time

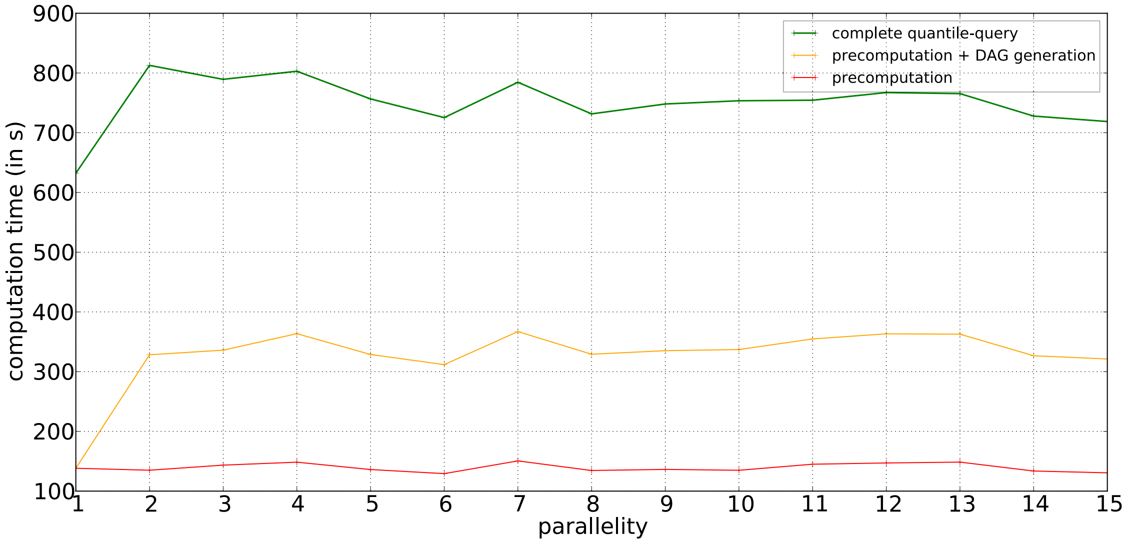

Figure 6.5: Parallel computations for Asynchronous Leader-Election Protocol

in order to be constructed (the average generation time for all parallel computations is 200.83 seconds in the case $N = 8$). Since there are only 13 iterations needed for determining the quantile in the case $N = 8$, the time for building the DAG does not pay off during the complete computation. Instead, each parallel computation starts its calculations with a delay of roughly 200 seconds compared to the sequential one. And this delay is huge enough to keep a distance between the times for the sequential computation and the parallel computations.

Note on the utilisation of the *LP*- and the *BP*-approach For a performance comparison between the LP-approach and the BP-approach a round-bounded reachability analysis is given in Table 6.8 for the asynchronous variant of the protocol. It can be seen that the BP-approach outperforms the LP-approach even for very small model-instances. For models with a slightly larger state space LP could not even calculate the demanded result in a reasonable time, whereas the BP-approach was able to do the analysis in a few seconds, which was the case for $N = 6$ processes. Due to the need of considering an increased number of reward iterations the computation time for $N = 5$ processes increased dramatically when the probability threshold p increased while utilising the LP-approach. Instead, the computation times remained stable for the BP-approach even when the number of the required iterations almost doubled.

Table 6.8: Comparison between *LP*- and *BP*-approach (minimal number of rounds, Asynchronous Leader-Election Protocol)

	Model			Existential Quantile		
Instance	States	t_{build}	p	Result	*BP*	*LP*
N = 3	364	0.05 s	0.1	2	0.12 s	0.16 s
			0.5	3	0.11 s	0.22 s
			0.99	9	0.12 s	1.09 s
N = 4	3 172	0.15 s	0.1	3	0.19 s	7.67 s
			0.5	4	0.2 s	14.96 s
			0.99	10	0.22 s	108.86 s
N = 5	27 299	0.75 s	0.1	3	0.58 s	597.94 s
			0.5	5	0.59 s	2 398.42 s
			0.99	11	0.64 s	14 502.94 s
N = 6	237 656	6.26 s	0.1	4	2.65 s	> 6 h
			0.5	5	2.73 s	> 6 h
			0.99	11	3.19 s	> 6 h
N = 7	2 095 783	58.13 s	0.1	4	22.55 s	> 6 h
			0.5	6	24.54 s	> 6 h
			0.99	12	30.26 s	> 6 h

Table 6.9: Statistics for explicit quantile (minimal number of rounds, Synchronous Leader-Election Protocol)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 3, K = 2	22	0.02 s	5	0.01 s	0.01 s	0.01 s
N = 3, K = 4	135	0.03 s	3	0.01 s	0.02 s	0.02 s
N = 3, K = 6	439	0.05 s	3	0.01 s	0.02 s	0.04 s
N = 3, K = 8	1 031	0.08 s	3	0.02 s	0.03 s	0.05 s
N = 4, K = 2	55	0.03 s	8	0.01 s	0.01 s	0.02 s
N = 4, K = 4	782	0.07 s	4	0.01 s	0.03 s	0.04 s
N = 4, K = 6	3 902	0.16 s	3	0.03 s	0.07 s	0.1 s
N = 4, K = 8	12 302	0.34 s	3	0.09 s	0.15 s	0.23 s
N = 5, K = 2	136	0.04 s	14	0.01 s	0.02 s	0.03 s
N = 5, K = 4	4 124	0.18 s	4	0.04 s	0.08 s	0.1 s
N = 5, K = 6	31 133	0.72 s	3	0.19 s	0.33 s	0.52 s
N = 5, K = 8	131 101	2.56 s	3	0.65 s	1.15 s	1.74 s
N = 6, K = 2	329	0.05 s	24	0.01 s	0.03 s	0.05 s
N = 6, K = 4	20 524	0.57 s	4	0.14 s	0.24 s	0.36 s
N = 6, K = 6	233 340	4.68 s	3	1.06 s	1.92 s	3.03 s
N = 6, K = 8	1 310 780	26.57 s	3	6.77 s	12.75 s	20.83 s

Table 6.10: Statistics for symbolic quantile (minimal number of rounds, Synchronous Leader-Election Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 3, K = 2	22	394	0.03 s	5	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.01 s
N = 3, K = 4	135	1 808	0.03 s	3	< 0.01 s	0.01 s	< 0.01 s	0.01 s	< 0.01 s	0.01 s
N = 3, K = 6	439	5 659	0.05 s	3	0.01 s	0.03 s	0.01 s	0.02 s	0.01 s	0.03 s
N = 3, K = 8	1 031	10 622	0.07 s	3	0.02 s	0.08 s	0.02 s	0.05 s	0.02 s	0.05 s
N = 4, K = 2	55	941	0.03 s	8	< 0.01 s	0.01 s	< 0.01 s	0.01 s	< 0.01 s	0.01 s
N = 4, K = 4	782	10 834	0.06 s	4	0.02 s	0.09 s	0.02 s	0.05 s	0.03 s	0.05 s
N = 4, K = 6	3 902	58 357	0.25 s	3	0.12 s	1.45 s	0.12 s	0.31 s	0.14 s	0.35 s
N = 4, K = 8	12 302	165 658	1.35 s	3	0.73 s	13.76 s	0.48 s	2.57 s	0.55 s	2.42 s
N = 5, K = 2	136	1 764	0.04 s	14	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	0.03 s
N = 5, K = 4	4 124	41 561	0.17 s	4	0.12 s	1.2 s	0.12 s	0.27 s	0.14 s	0.31 s
N = 5, K = 6	31 133	337 141	5.19 s	3	2.72 s	103.21 s	2.42 s	10.77 s	1.86 s	7.86 s
N = 5, K = 8	131 101	1 274 346	53.13 s	3	7.36 s	3 822.38 s	10.28 s	76.47 s	9.72 s	69.96 s
N = 6, K = 2	329	3 202	0.04 s	24	0.01 s	0.03 s	0.01 s	0.03 s	0.01 s	0.07 s
N = 6, K = 4	20 524	140 774	1.2 s	4	0.99 s	19.78 s	1.04 s	3.44 s	1.24 s	4.16 s
N = 6, K = 6	233 340	1 732 135	129.42 s	3	30.95 s	9 891.37 s	26.19 s	184.49 s	20.47 s	173.87 s
N = 6, K = 8	1 310 780	8 899 587	8 814.66 s	3	316.57 s	> 24 h	315.25 s	9 288.16 s	294.03 s	9 098.35 s

Table 6.11: Statistics for explicit quantile (minimal number of steps, Synchronous Leader-Election Protocol)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 3, K = 2	22	0.02 s	17	< 0.01 s	0.01 s	0.01 s
N = 3, K = 4	135	0.03 s	9	< 0.01 s	0.01 s	0.01 s
N = 3, K = 6	439	0.05 s	9	0.01 s	0.01 s	0.02 s
N = 3, K = 8	1 031	0.08 s	9	0.01 s	0.01 s	0.02 s
N = 4, K = 2	55	0.03 s	36	< 0.01 s	0.01 s	0.02 s
N = 4, K = 4	782	0.07 s	16	0.01 s	0.02 s	0.02 s
N = 4, K = 6	3 902	0.16 s	11	0.03 s	0.04 s	0.06 s
N = 4, K = 8	12 302	0.34 s	11	0.06 s	0.09 s	0.1 s
N = 5, K = 2	136	0.04 s	79	< 0.01 s	0.01 s	0.02 s
N = 5, K = 4	4 124	0.18 s	19	0.03 s	0.05 s	0.07 s
N = 5, K = 6	31 133	0.72 s	13	0.13 s	0.18 s	0.21 s
N = 5, K = 8	131 101	2.56 s	13	0.53 s	0.69 s	0.62 s
N = 6, K = 2	329	0.05 s	162	0.01 s	0.02 s	0.04 s
N = 6, K = 4	20 524	0.57 s	22	0.09 s	0.14 s	0.18 s
N = 6, K = 6	233 340	4.68 s	15	0.96 s	1.22 s	1.1 s
N = 6, K = 8	1 310 780	26.57 s	15	6.38 s	7.09 s	7.21 s

Table 6.12: Statistics for symbolic quantile (minimal number of steps, Synchronous Leader-Election Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 3, K = 2	22	394	0.03 s	17	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.01 s
N = 3, K = 4	135	1 808	0.03 s	9	< 0.01 s	0.01 s	< 0.01 s	0.01 s	< 0.01 s	0.01 s
N = 3, K = 6	439	5 659	0.05 s	9	0.01 s	0.02 s	0.01 s	0.01 s	0.01 s	0.02 s
N = 3, K = 8	1 031	10 622	0.07 s	9	0.02 s	0.06 s	0.02 s	0.03 s	0.02 s	0.04 s
N = 4, K = 2	55	941	0.03 s	36	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.02 s
N = 4, K = 4	782	10 834	0.06 s	16	0.02 s	0.07 s	0.02 s	0.03 s	0.03 s	0.06 s
N = 4, K = 6	3 902	58 357	0.25 s	11	0.12 s	1.3 s	0.12 s	0.15 s	0.12 s	0.23 s
N = 4, K = 8	12 302	165 658	1.35 s	11	0.79 s	12.04 s	0.67 s	1.02 s	0.59 s	2.6 s
N = 5, K = 2	136	1 764	0.04 s	79	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	0.04 s
N = 5, K = 4	4 124	41 561	0.17 s	19	0.12 s	1.08 s	0.12 s	0.15 s	0.13 s	0.26 s
N = 5, K = 6	31 133	337 141	5.19 s	13	2.92 s	96.79 s	2.67 s	4.16 s	1.95 s	7.59 s
N = 5, K = 8	131 101	1 274 346	53.13 s	13	7.92 s	3 750.07 s	10.62 s	16.98 s	10.99 s	50.19 s
N = 6, K = 2	329	3 202	0.04 s	162	0.01 s	0.02 s	0.01 s	0.02 s	0.01 s	0.12 s
N = 6, K = 4	20 524	140 774	1.2 s	22	1.1 s	18.21 s	1.14 s	1.61 s	1.47 s	5.71 s
N = 6, K = 6	233 340	1 732 135	129.42 s	15	31.65 s	9 696.46 s	28.71 s	41.55 s	23.49 s	148.01 s
N = 6, K = 8	1 310 780	8 899 587	8 814.66 s	15	316.57 s	> 24 h	319.96 s	381.44 s	82.64 s	238.13 s

Table 6.13: Statistics for explicit existential quantile (minimal number of rounds, Asynchronous Leader-Election Protocol)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2	36	0.02 s	8	0.01 s	0.01 s	0.02 s
N = 3	364	0.05 s	10	0.01 s	0.02 s	0.04 s
N = 4	3 172	0.15 s	11	0.03 s	0.07 s	0.13 s
N = 5	27 299	0.83 s	12	0.19 s	0.4 s	0.58 s
N = 6	237 656	7.02 s	12	1.24 s	3.21 s	4.38 s
N = 7	2 095 783	79.3 s	13	9.41 s	35.27 s	44.41 s
N = 8	18 674 484	773.39 s	13	102.69 s	631.58 s	729.35 s

Table 6.14: Statistics for symbolic existential quantile (minimal number of rounds, Asynchronous Leader-Election Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 2	36	498	0.03 s	8	< 0.01 s	0.01 s	< 0.01 s	0.01 s	< 0.01 s	0.01 s
N = 3	364	2 876	0.05 s	10	0.02 s	0.03 s	0.02 s	0.03 s	0.02 s	0.08 s
N = 4	3 172	9 472	0.1 s	11	0.13 s	0.21 s	0.13 s	0.18 s	0.13 s	0.77 s
N = 5	27 299	27 716	0.3 s	12	1.19 s	1.84 s	1.01 s	1.38 s	1.38 s	7.64 s
N = 6	237 656	69 951	1.09 s	12	5.76 s	12.47 s	5.68 s	9.07 s	4.85 s	35.07 s
N = 7	2 095 783	163 758	4.56 s	13	23.55 s	113.93 s	33.94 s	70.46 s	23.15 s	206.13 s
N = 8	18 674 484	354 418	18.47 s	13	85.08 s	1 152.6 s	125.1 s	498.84 s	61.16 s	749.1 s
N = 9	167 748 115	789 166	65.56 s	14	451.49 s	13 782.0 s	282.29 s	4 768.49 s	243.63 s	4 629.83 s

Table 6.15: Statistics for explicit existential quantile (minimal number of steps, Asynchronous Leader-Election Protocol)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2	36	0.02 s	43	< 0.01 s	0.01 s	0.02 s
N = 3	364	0.05 s	74	0.01 s	0.02 s	0.02 s
N = 4	3 172	0.15 s	104	0.03 s	0.08 s	0.05 s
N = 5	27 299	0.83 s	139	0.12 s	0.7 s	0.31 s
N = 6	237 656	7.02 s	168	1.26 s	8.51 s	3.83 s
N = 7	2 095 783	79.3 s	206	8.5 s	62.43 s	27.51 s
N = 8	18 674 484	773.39 s	238	100.5 s	747.07 s	288.96 s

Table 6.16: Statistics for symbolic existential quantile (minimal number of steps, Asynchronous Leader-Election Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 2	36	498	0.03 s	43	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	< 0.01 s	0.01 s
N = 3	364	2 876	0.05 s	74	0.01 s	0.02 s	0.01 s	0.02 s	0.02 s	0.11 s
N = 4	3 172	9 472	0.1 s	104	0.07 s	0.11 s	0.07 s	0.1 s	0.11 s	1.32 s
N = 5	27 299	27 716	0.3 s	139	1.3 s	1.83 s	1.02 s	1.39 s	0.84 s	15.29 s
N = 6	237 656	69 951	1.09 s	168	5.68 s	11.54 s	5.93 s	9.96 s	4.96 s	70.94 s
N = 7	2 095 783	163 758	4.56 s	206	23.29 s	95.41 s	33.71 s	78.34 s	16.04 s	373.23 s
N = 8	18 674 484	354 418	18.47 s	238	81.76 s	879.78 s	105.87 s	575.51 s	66.01 s	1 426.9 s
N = 9	167 748 115	789 166	65.56 s	279	404.6 s	9 400.38 s	284.91 s	5 666.62 s	219.88 s	7 439.64 s

6.1.3 Randomised Consensus Shared Coin Protocol

A consensus protocol aims at achieving an agreement on a selected value out of a variety of different values when there are multiple participants involved. The consensus protocol of Aspnes and Herlihy [AH90] was analysed using formal methods in [KNS01] by separating the protocol into two different parts, one without any probabilistic behaviour and the other equipped with probabilism, as it was proposed in [PSL00]. Each part was then analysed separately using different methods and tools. For the analysis of the non-probabilistic part the authors of [KNS01] chose Cadence SMV[8] (an extension of SMV [McM93]) and the second part equipped with probabilistic behaviour was analysed using the probabilistic model checker PRISM. Since the aim of this monograph is to demonstrate the practical viability of the developed quantile framework and since this puts the emphasis on the treatment of probabilistic behaviour, the focus here is on the probabilistic part of the protocol, the so-called *Randomised Consensus Shared Coin Protocol*. It is publicly available[9] and will be analysed using the quantile framework discussed so far.

The protocol consists of N processes (participants) and provides a shared counter, which is accessible by each of the processes. It works in a way such that it performs multiple rounds, where in each round, a process flips a coin and, depending on the outcome of the coin toss, either the respective process increments or decrements the counter. The protocol fixes a constant $K > 1$ to establish barriers for specifying the desired value of a process. The barriers help to prevent processes to move the value of the counter into a specific direction, and therefore minimise the influence of an adversary on the outcome of the protocol. So, with increasing K the influence of an adversary decreases.

Since the operational behaviour of the described protocol depends on coin tosses, the minimal number of flips necessary for a successful termination is of peculiar interest. Therefore, there are several questions which are of interest for the protocol:

1. What is the minimal number of coin-flips needed in order to guarantee a termination of the protocol with a given probability?
2. What is the minimal number of coin-flips needed to guarantee a successful protocol termination with a specific probability, i.e., all involved processes agree on the same value?

Also, the required number of steps can be seen as a significant measure for the performance of the protocol:

3. How many steps does the protocol need in order to guarantee its termination with a given probability?
4. How many steps are at least necessary to guarantee a successful protocol termination with a specific probability?

---

[8] `http://www.kenmcmil.com/smv.html`, retrieved 28th March 2018

[9] `http://www.prismmodelchecker.org/casestudies/consensus_prism.php`, retrieved 28th March 2018

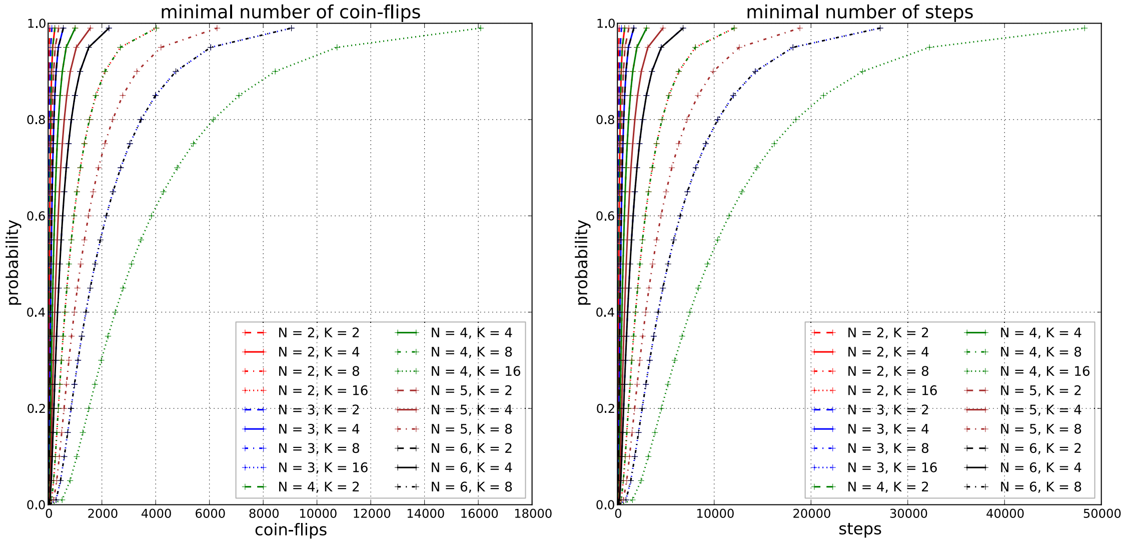

Figure 6.6: Results for Randomised Consensus Shared Coin Protocol

Formal analysis Since a successful protocol termination is highly important the formal analysis will answer the questions 2 and 4 formalised using the following existential quantile:

$$\min\{r \in \mathbb{N} \;:\; \mathsf{Pr}^{\max}(\Diamond^{\leq r}(\text{Finished} \wedge \text{Agreement})) > p\},$$

with r denoting either the number of coin-flips or the number of steps done in the protocol, and $p \in \{0, 0.01, 0.05, 0.1, \ldots, 0.9, 0.95, 0.99\}$ is the respective probability threshold. The labelling Finished hereby means that each process has finished the decision for itself, i.e., every involved participant selected its preferred value, and the label Agreement denotes situations where the selected value for all participants is the same. The considered queries can be computed using the proposed quantile framework, and the results of those computations are depicted in Figure 6.6, whereas the corresponding statistics for the computation of the minimal number of coin-flips can be found in Table 6.17 (EXPLICIT-engine) on page 111, and Table 6.18 (symbolic engines) on page 112. The corresponding statistics for the minimal number of steps are denoted in Table 6.19 (EXPLICIT-engine) on page 113, and in Table 6.20 (symbolic engines) on page 114.

As can be seen by the number of the required iterations in the corresponding tables, and as well in the plots of Figure 6.6 the resulting quantile values for this protocol are very high compared to the previously investigated protocols. The higher the value for K the more iterations are needed in order to find an agreement for the involved processes. Therefore, the overall time to calculate all necessary iterations is also very high compared to the previous ones. For this reason it is impossible to have statistics for the LP-approach for this protocol, since the linear program that needs to be solved by this approach would consist of too many variables simply because of the number of the iterations that need to be considered. But, this high number of required iterations comes in handy for the parallel computations of the Explicit-engine. The overhead that needs to be paid in order to prepare the parallel computations is negligible because of the huge number of required iterations, meaning that the investment of the overhead pays off in each iteration. Therefore, it is not that surprising that the timings for the parallel computations are really fast when compared to the other engines. Another thing is that the precomputation of the Explicit-engine is the fastest out of all four available engines. For the previously investigated protocols the opposite was true. This is a bit surprising since the model has an extremely compact symbolic representation in terms of MTBDD-nodes. The performance of the Sparse-engine and the sequential Explicit-engine is nearly the same, whereas the MTBDD-engine and the Hybrid-engine seem to have their issues with this protocol.

Another interesting observation is that reasoning over the minimal number of coin-flips and over the minimal number of steps is done in nearly the same time for all the considered engines although the required iterations for the number of steps are roughly three times higher than for the number of coin-flips. Since the computation of the minimal number of steps does not need to consider any zero-reward states and therefore there is no need of topologically sorting any zero-reward SCCs, the *BP*-approach can be applied for each state directly in each considered iteration. And this improves the computational performance to a high degree.

Note on the storage of zero-reward sub-MDPs As the analysis of the minimal number of coin-flips reveals the model consists of many zero-reward sub-**MDP**s which have a positive probability of reaching the desired goal-states and therefore need to be treated adequately by the framework (see Section 5.1.3 on page 71). Since for larger instances of the model there are quite a lot of those zero-reward sub-**MDP**s, for the Explicit-engine it was not even possible to store them in memory by using the standard BitSet-implementation shipped with version 1.8.0_72-b15 of the Java programming language. So, without the utilisation of the self-built storage for integer sets (see Section 5.1.7 on page 77) the computation would not have been possible for the Explicit-engine due to problems with the memory.

Note on parallel computations using the Explicit-engine This protocol needs a huge amount of iterations in order to compute the desired quantile values, and this fact is beneficial for the application of the parallel computations of the Explicit-engine. Therefore, we give an overview over the performance of the parallel computations as it was also done for the Self-Stabilising Protocol (see Section 6.1.1) or the asynchronous variant of the Leader-Election Protocol (see Section 6.1.2). In contrast to the Self-Stabilising Protocol the protocol here involves many zero-reward sub-**MDP**s when analysing the minimal number of coin-flips. It is therefore necessary to do a topological sorting of those sub-**MDP**s by constructing the corresponding DAG as explained in Section 5.1.4. Figure 6.7 shows the running times for the computation of the explicit existential quantile for computing the minimal number of coin-flips in the case

$N = 6, K = 8$. As already seen in Figure 6.5 the plot depicts the running times for the

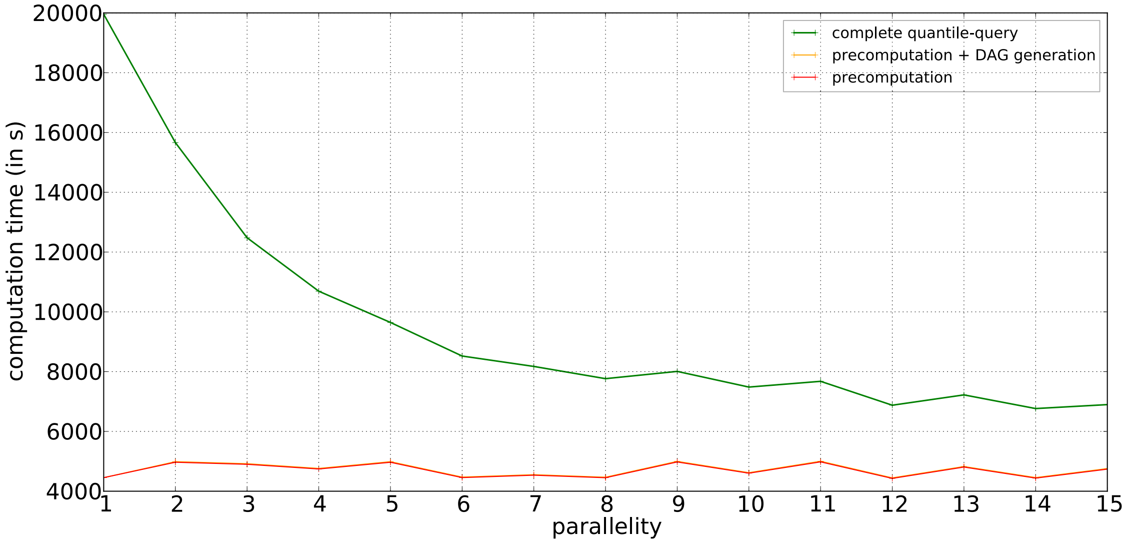

Figure 6.7: Parallel computations for Randomised Consensus Shared Coin Protocol

whole quantile calculation together with the time necessary for the precomputation and the time for the generation of the DAG. As before the maximal degree of parallelism was restricted to 15 in order to keep the computing system responsive. The generation of the DAG is negligible in relation to the amount of time that is consumed by the precomputation. So, the overall performance is not effected in a negative way by establishing the requirements for the parallel computations. And since the needed iterations are relatively high (9 051) there is a lot of potential for the parallel Explicit-engine to reduce the computation times. What this all amounts to is that the parallel computations do perform really well in this case, reducing the overall computation time by roughly 65% for a parallelism of 15.

Table 6.17: Statistics for explicit quantile (minimal number of coin-flips, Randomised Consensus Shared Coin Protocol)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2, K = 2	272	0.04 s	63	0.01 s	0.04 s	0.07 s
N = 2, K = 4	528	0.05 s	251	0.02 s	0.06 s	0.16 s
N = 2, K = 8	1 040	0.06 s	1 005	0.06 s	0.24 s	0.4 s
N = 2, K = 16	2 064	0.09 s	4 023	0.32 s	1.6 s	1.59 s
N = 3, K = 2	2 720	0.13 s	141	0.1 s	0.19 s	0.26 s
N = 3, K = 4	5 216	0.2 s	565	0.3 s	0.85 s	0.8 s
N = 3, K = 8	10 208	0.27 s	2 263	1.57 s	5.8 s	3.59 s
N = 3, K = 16	20 192	0.4 s	9 051	9.76 s	34.15 s	20.83 s
N = 4, K = 2	22 656	0.5 s	251	0.88 s	2.15 s	1.58 s
N = 4, K = 4	43 136	0.78 s	1 005	4.44 s	18.99 s	8.34 s
N = 4, K = 8	84 096	1.33 s	4 023	27.21 s	149.9 s	56.6 s
N = 4, K = 16	166 016	2.43 s	16 091	175.26 s	843.68 s	394.92 s
N = 5, K = 2	173 056	3.3 s	393	12.31 s	36.9 s	24.11 s
N = 5, K = 4	327 936	5.89 s	1 571	71.43 s	230.31 s	125.25 s
N = 5, K = 8	637 696	11.1 s	6 285	451.0 s	1 611.13 s	823.92 s
N = 6, K = 2	1 258 240	27.74 s	565	133.67 s	423.46 s	216.51 s
N = 6, K = 4	2 376 448	46.6 s	2 263	750.82 s	2 725.51 s	1 249.19 s
N = 6, K = 8	4 612 864	106.25 s	9 051	4 134.19 s	19 499.24 s	8 027.4 s

Table 6.18: Statistics for symbolic quantile (minimal number of coin-flips, Randomised Consensus Shared Coin Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 2, K = 2	272	424	0.03 s	63	0.03 s	0.04 s	0.03 s	0.04 s	0.13 s	0.27 s
N = 2, K = 4	528	464	0.03 s	251	0.11 s	0.14 s	0.1 s	0.12 s	0.86 s	1.66 s
N = 2, K = 8	1 040	504	0.03 s	1 005	0.5 s	0.7 s	0.4 s	0.54 s	5.95 s	11.57 s
N = 2, K = 16	2 064	544	0.04 s	4 023	2.47 s	3.87 s	1.81 s	2.76 s	42.2 s	85.35 s
N = 3, K = 2	2 720	1 279	0.04 s	141	0.35 s	0.46 s	0.28 s	0.35 s	3.14 s	4.91 s
N = 3, K = 4	5 216	1 271	0.04 s	565	1.45 s	2.2 s	0.99 s	1.47 s	21.88 s	32.99 s
N = 3, K = 8	10 208	1 329	0.06 s	2 263	7.17 s	12.95 s	4.37 s	7.89 s	149.59 s	234.61 s
N = 3, K = 16	20 192	1 387	0.1 s	9 051	40.76 s	86.31 s	21.84 s	49.34 s	1 086.39 s	1 822.62 s
N = 4, K = 2	22 656	2 685	0.06 s	251	3.7 s	5.8 s	2.31 s	3.56 s	43.55 s	62.63 s
N = 4, K = 4	43 136	2 761	0.08 s	1 005	18.09 s	33.97 s	9.27 s	18.19 s	220.84 s	321.64 s
N = 4, K = 8	84 096	2 837	0.12 s	4 023	99.0 s	222.4 s	44.48 s	113.6 s	1 353.75 s	2 038.94 s
N = 4, K = 16	166 016	2 913	0.2 s	16 091	593.28 s	1 567.7 s	247.29 s	798.72 s	9 888.01 s	15 547.06 s
N = 5, K = 2	173 056	4 996	0.09 s	393	39.22 s	73.41 s	19.07 s	37.75 s	230.56 s	313.77 s
N = 5, K = 4	327 936	5 164	0.13 s	1 571	253.32 s	600.64 s	86.42 s	234.89 s	1 259.44 s	1 779.99 s
N = 5, K = 8	637 696	5 216	0.2 s	6 285	1 657.43 s	4 553.45 s	471.3 s	1 630.34 s	6 514.38 s	9 603.54 s
N = 6, K = 2	1 258 240	8 506	0.15 s	565	572.01 s	1 279.57 s	153.86 s	419.69 s	967.2 s	1 316.03 s
N = 6, K = 4	2 376 448	8 492	0.21 s	2 263	3 368.72 s	8 913.29 s	788.37 s	2 819.84 s	4 061.78 s	5 802.06 s
N = 6, K = 8	4 612 864	8 604	0.34 s	9 051	20 321.39 s	63 502.53 s	4 615.25 s	20 138.23 s	26 010.39 s	37 059.0 s

Table 6.19: Statistics for explicit quantile (minimal number of steps, Randomised Consensus Shared Coin Protocol)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2, K = 2	272	0.04 s	187	0.01 s	0.02 s	0.04 s
N = 2, K = 4	528	0.05 s	751	0.01 s	0.07 s	0.1 s
N = 2, K = 8	1 040	0.06 s	3 013	0.06 s	0.36 s	0.39 s
N = 2, K = 16	2 064	0.09 s	12 067	0.31 s	2.75 s	1.81 s
N = 3, K = 2	2 720	0.13 s	421	0.06 s	0.19 s	0.15 s
N = 3, K = 4	5 216	0.2 s	1 693	0.25 s	1.25 s	0.69 s
N = 3, K = 8	10 208	0.27 s	6 787	1.51 s	9.47 s	6.28 s
N = 3, K = 16	20 192	0.4 s	27 151	9.64 s	53.87 s	28.33 s
N = 4, K = 2	22 656	0.5 s	751	0.77 s	2.92 s	1.49 s
N = 4, K = 4	43 136	0.78 s	3 013	4.37 s	22.33 s	8.44 s
N = 4, K = 8	84 096	1.33 s	12 067	28.63 s	121.89 s	62.62 s
N = 4, K = 16	166 016	2.43 s	48 271	173.1 s	884.62 s	384.89 s
N = 5, K = 2	173 056	3.3 s	1 177	10.99 s	31.88 s	18.88 s
N = 5, K = 4	327 936	5.89 s	4 711	67.19 s	211.2 s	108.57 s
N = 5, K = 8	637 696	11.1 s	18 853	435.82 s	1 542.89 s	765.15 s
N = 6, K = 2	1 258 240	27.74 s	1 693	132.66 s	369.97 s	181.92 s
N = 6, K = 4	2 376 448	46.6 s	6 787	746.37 s	2 360.51 s	1 163.84 s
N = 6, K = 8	4 612 864	106.25 s	27 151	4 131.67 s	19 360.68 s	8 015.83 s

Table 6.20: Statistics for symbolic quantile (minimal number of steps, Randomised Consensus Shared Coin Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 2, K = 2	272	424	0.03 s	187	0.02 s	0.03 s	0.02 s	0.02 s	0.13 s	0.27 s
N = 2, K = 4	528	464	0.03 s	751	0.09 s	0.12 s	0.07 s	0.1 s	0.87 s	1.73 s
N = 2, K = 8	1 040	504	0.03 s	3 013	0.49 s	0.7 s	0.39 s	0.55 s	5.95 s	12.14 s
N = 2, K = 16	2 064	544	0.04 s	12 067	2.45 s	3.92 s	1.78 s	2.83 s	40.24 s	87.01 s
N = 3, K = 2	2 720	1 279	0.04 s	421	0.36 s	0.45 s	0.28 s	0.34 s	3.52 s	5.12 s
N = 3, K = 4	5 216	1 271	0.04 s	1 693	1.42 s	2.07 s	1.0 s	1.43 s	24.2 s	37.11 s
N = 3, K = 8	10 208	1 329	0.06 s	6 787	7.25 s	12.13 s	4.39 s	7.54 s	193.05 s	318.86 s
N = 3, K = 16	20 192	1 387	0.1 s	27 151	41.28 s	79.19 s	21.92 s	46.17 s	1 073.18 s	1 908.82 s
N = 4, K = 2	22 656	2 685	0.06 s	751	3.67 s	5.15 s	2.25 s	3.23 s	43.18 s	60.0 s
N = 4, K = 4	43 136	2 761	0.08 s	3 013	17.56 s	28.8 s	8.74 s	15.94 s	212.6 s	316.99 s
N = 4, K = 8	84 096	2 837	0.12 s	12 067	97.25 s	184.32 s	42.87 s	98.44 s	1 401.78 s	2 228.5 s
N = 4, K = 16	166 016	2 913	0.2 s	48 271	588.08 s	1 273.48 s	239.94 s	693.58 s	9 492.08 s	16 435.96 s
N = 5, K = 2	173 056	4 996	0.09 s	1 177	38.7 s	59.78 s	18.02 s	32.22 s	223.21 s	303.15 s
N = 5, K = 4	327 936	5 164	0.13 s	4 711	249.56 s	459.96 s	83.14 s	189.87 s	1 187.34 s	1 725.41 s
N = 5, K = 8	637 696	5 216	0.2 s	18 853	1 627.69 s	3 364.09 s	456.5 s	1 295.75 s	6 626.94 s	10 490.12 s
N = 6, K = 2	1 258 240	8 506	0.15 s	1 693	570.3 s	945.04 s	153.48 s	331.16 s	926.26 s	1 255.29 s
N = 6, K = 4	2 376 448	8 492	0.21 s	6 787	3 364.65 s	6 311.22 s	784.23 s	2 127.69 s	4 105.01 s	6 059.45 s
N = 6, K = 8	4 612 864	8 604	0.34 s	27 151	20 490.71 s	43 574.52 s	4 570.97 s	14 978.98 s	23 821.4 s	40 227.42 s

6.2 Energy-Aware Protocols

The following section addresses protocols where the emphasis is on their respective energy efficiency. It is shown how the presented quantile framework can support the analysis of systems that adapt their configurations and utilisations to varying requirements in order to save energy (of course the application of a quantile-based multi-objective analysis is not restricted to just the demands of this case). So, we are interested in minimising the energy and simultaneously maximising the provided utility.

6.2.1 Energy-Aware Job-Scheduling Protocol

Here, we turn to the Energy-Aware Job-Scheduling Protocol which has been introduced previously in [Bai+14a]. For this protocol we want to do an energy-utility analysis using the presented quantiles. The system consists of a fixed number of processes which want to successfully perform tasks within a given deadline. During the completion of such a task a process needs to enter a critical section where it utilises a shared resource. The access to this resource will be granted exclusively by a scheduler selecting the process out of the pool of processes which requested access to the resource. 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 the computation of 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 also tripling energy consumption as a drawback.

Operational behaviour of the protocol The operational behaviour of the protocol is provided by means of control-flow graphs representing **MDP**s: one for the processes (see Figure 6.8) and one for the shared resource (see Figure 6.9). The number N is fixed and will denote the number of involved processes. Similar to the input language of Prism, such control-flow graphs define **MDP** modules which are denoted by $\mathcal{P}_i$ for each process $i = 1, ..., N$ and $\mathcal{R}$ for the resource. Using standard parallel composition [Seg95], these **MDP** modules yield the **MDP** semantics $\mathcal{M} = \mathcal{P}_1 \| \cdots \| \mathcal{P}_N \| \mathcal{R}$ of the whole protocol, which is subject of the analysis.

Each control-flow graph $\mathcal{P}_i$ of the i-th process is described by the four different locations $\textsc{start}_i$, $\textsc{ncrit}_i$, $\textsc{wait}_i$ and $\textsc{crit}_i$ (see Figure 6.8). Starting in the initial location $\textsc{start}_i$, the process moves to its noncritical location $\textsc{ncrit}_i$, indicating that process i has not requested access to the resource yet. At this transition, the timer t_i is randomly set according to a distribution τ, representing the time where the process stays in $\textsc{ncrit}_i$. Afterwards, it requests access to the shared resource by entering its waiting location $\textsc{wait}_i$ and randomly choosing a deadline counter d_i and a computation

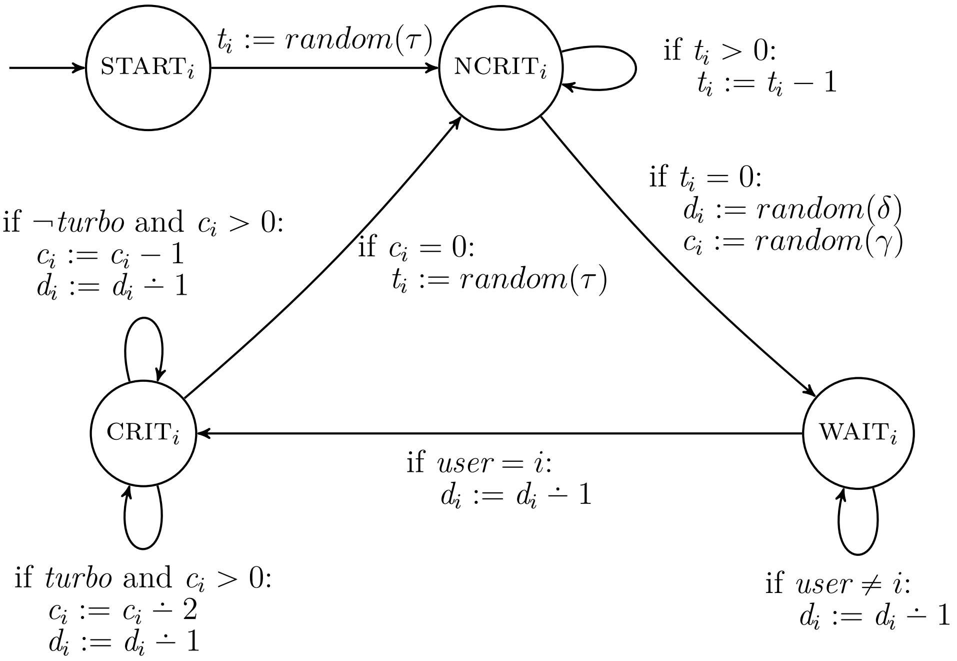

Figure 6.8: Control flow for process $\mathcal{P}_i$ (Energy-Aware Job-Scheduling Protocol)

timer c_i according to distributions δ and γ, respectively. Intuitively, the timer c_i represents the time required to finish the desired task while process i uses the shared resource, and the deadline counter d_i formalises the time until the task needs to be finished. Clearly, c_i is required to be smaller than d_i but greater than 0. Waiting for the access to the shared resource, d_i is decreased if it is greater than zero (this is expressed by the operation $d_i := d_i \dot{-} 1$ in Figure 6.8, where $\dot{-}$ stands for the positive difference, i.e., $m \dot{-} n = \max(0, m - n)$ for natural numbers $m, n \in \mathbb{N}$). Since the access to the shared resource is exclusive, the decision which process can access the resource is performed by a scheduler, which will be detailed later on. The scheduler decides to grant access to process i by setting $user = i$, allowing the process to move to its critical location CRIT$_i$. The process then performs the task which requires access to the shared resource. In order to meet the deadline, the scheduler may decide to activate a turbo mode ($turbo = \mathsf{true}$), doubling the computation speed of the task. This is modelled by subtracting 2 instead of the standard 1 from the computation timer c_i at every time step in the critical section. After the computation of the task has finished, the process leaves the critical location, enters its noncritical location and the procedure starts over again. All N processes are synchronously composed, i.e., at each step of the protocol, all processes perform exactly one transition in their control-flow graph.

Now, we turn to the resource module $\mathcal{R}$, which settles the rules for the access to the shared resource. The control-flow graph of $\mathcal{R}$ is depicted in Figure 6.9, where k and i range over all processes, i.e., one transition in the graph involving k or i stands for multiple transitions replacing k and i by $1, \ldots, N$. Synchronously composed with the processes' behaviour $\mathcal{P}_1 \| \cdots \| \mathcal{P}_N$, the resource behaviour depends on the processes (guards $c_i = 0$ indicating that process i has finished its computation and therefore leaves the critical section, and $\textsc{wait}_k$ standing for process k's request to enter the critical section) and the processes depend on the resource (the guards $user = i$ if process i has the right to be in the critical section and *turbo* standing for the case whether the turbo mode is activated (formally, true) or deactivated (false)). Whereas the processes' behaviour is deterministic, the model of the resource involves nondeterminism, e.g., if multiple processes are in their respective waiting location, or for the decision if the turbo mode will be utilised. At the beginning, the resource is not in use, meaning that *user* is undefined ($user = \bot$) and the turbo mode is deactivated (*turbo* set to false). Whenever there is a process k waiting (i.e., k is in location $wait_k$) and no process has access to the resource ($user = \bot$), the resource can be assigned to the waiting process by setting $user := k$. In this turn, it is also decided whether to activate the turbo mode or not. As long as the computation counter c_i of the process i in the critical section is set, i.e., the task of process i is still computed, the resource does not change its parameters. Only if the computation is finished ($c_i = 0$ and $user = i$), then either the resource is directly handed to another waiting process k, also activating or deactivating the turbo mode, or the user of the resource is set undefined if no process is in its waiting location.

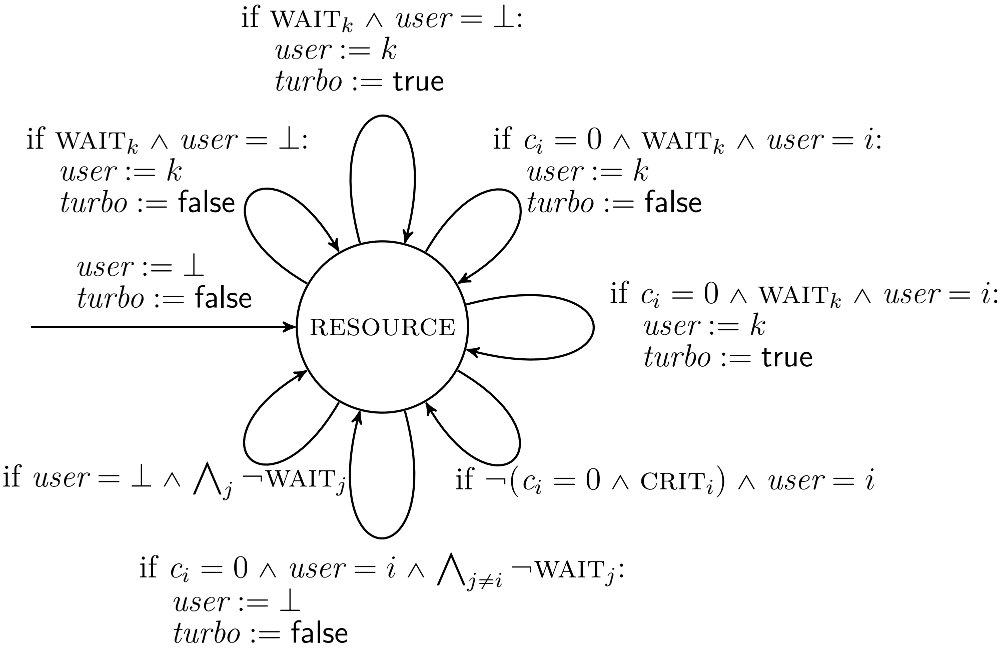
if $\textsc{wait}_k \wedge user = \bot$: $user := k$ $turbo := \mathsf{true}$
if $\textsc{wait}_k \wedge user = \bot$: $user := k$ $turbo := \mathsf{false}$
if $c_i = 0 \wedge \textsc{wait}_k \wedge user = i$: $user := k$ $turbo := \mathsf{false}$
$user := \bot$ $turbo := \mathsf{false}$
RESOURCE
if $c_i = 0 \wedge \textsc{wait}_k \wedge user = i$: $user := k$ $turbo := \mathsf{true}$
if $user = \bot \wedge \bigwedge_j \neg\textsc{wait}_j$
if $\neg(c_i = 0 \wedge \textsc{crit}_i) \wedge user = i$
if $c_i = 0 \wedge user = i \wedge \bigwedge_{j \neq i} \neg\textsc{wait}_j$: $user := \bot$ $turbo := \mathsf{false}$

Figure 6.9: Control flow for resource $\mathcal{R}$ (Energy-Aware Job-Scheduling Protocol)

Energy, utility and model parameters For the analysis certain parameters were fixed while the number of involved processes varies. The probability distributions for each process is provided in the form $(p{:}n, 1{-}p{:}m)$, where $p \in [0, 1]$ is the probability for value n and $1-p$ the probability that value m occurs. The deadline distribution is fixed to $\delta = \left(\frac{1}{3}{:}7, \frac{2}{3}{:}9\right)$, the computation distribution for the critical section is set to $\gamma = \left(\frac{1}{3}{:}2, \frac{2}{3}{:}3\right)$, and the timer distribution for the non-critical section is fixed to $\tau = \left(\frac{2}{3}{:}4, \frac{1}{3}{:}5\right)$.

By a synchronous parallel composition of the models for processes (see Figure 6.8) and the resource model (see Figure 6.9) the **MDP** $\mathcal{M}$ arises, which models the behaviour of the complete protocol. The energy consumption and the achieved degree of utility of the model are represented using distinct reward functions over $\mathcal{M}$. Utility is specified as the number of successfully finished tasks, i.e., a transition reward of 1 is granted if some process i takes a transition from $\textsc{crit}_i$ to $\textsc{ncrit}_i$ and the value of the deadline counter d_i is greater than 0. The energy consumption of the system is modelled by state rewards, where the global energy consumption in a state of $\mathcal{M}$ is the sum of the energy consumption of all processes in their respective local states. Depending on the local state of a process, the consumed energy differs as it is detailed in Table 6.21.

Table 6.21: Rewards modelling the energy consumption of a process (Energy-Aware Job-Scheduling Protocol)

location of process i	assigned energy consumption			
	$i \mod 3 \neq 0$		$i \mod 3 = 0$	
	$turbo = \mathsf{false}$	$turbo = \mathsf{true}$	$turbo = \mathsf{false}$	$turbo = \mathsf{true}$
$\textsc{ncrit}_i$	2	2	4	4
$\textsc{wait}_i$	1	1	2	2
$\textsc{crit}_i$	3	9	6	18

In order to model a heterogeneous architecture, the energy consumption of every third process is doubled. The scheduler has to take this heterogeneity into account, e.g., it needs to decide whether to schedule a job immediately to a process having a higher energy consumption and to process the job as soon as possible, or to postpone the processing of the job in order to utilise a process with a lower energy consumption. But the last alternative may result in a deadline miss for the job to be scheduled.

There is the possibility of activating a turbo mode whenever a process performs a computation in its critical section. The turbo mode results in a computation twice as fast as normal, but it comes at the cost of tripling the energy consumption (see Table 6.21). Since the doubled computation speed allows to leave the critical section twice as fast, the overall accumulated energy consumption is 50% higher in the critical section with activated turbo mode than it is with deactivated turbo.

Formal analysis The discussed quantile framework is used in order to analyse the trade-off between energy and utility. The analysis concentrates mainly on the following two values:

- The minimal amount of energy $\mathrm{e_{min}}$ that is required to exceed a desired utility bound u with a probability higher than p.

- The maximal utility $\mathrm{u}_{\max}$ that can be obtained with a probability greater than p such that the consumed energy is kept within a fixed energy budget e.

Formally, these values can be expressed using the following existential energy-utility bound quantiles:

$$\begin{aligned}\mathrm{e}_{\min} &= \mathrm{qu}_s\big(\,\exists\mathbb{P}_{>p}(\Diamond^{\leqslant ?}(\mathrm{utility} > \mathrm{u}))\,\big)\\ &= \min\big\{\hat{\mathrm{e}} \in \mathbb{N} : \mathsf{Pr}_s^{\max}(\ \Diamond(\mathrm{energy} \leqslant \hat{\mathrm{e}}\ \wedge\ \mathrm{utility} > \mathrm{u})\ ) > p\big\}\end{aligned}$$

$$\begin{aligned}\mathrm{u}_{\max} &= \mathrm{qu}_s\big(\,\exists\mathbb{P}_{>p}(\Diamond^{\geqslant ?}(\mathrm{energy} < \mathrm{e}))\,\big)\\ &= \max\big\{\hat{\mathrm{u}} \in \mathbb{N} : \mathsf{Pr}_s^{\max}(\ \Diamond(\mathrm{energy} < \mathrm{e}\ \wedge\ \mathrm{utility} \geqslant \hat{\mathrm{u}})\ ) > p\big\}\end{aligned}$$

It needs to be taken into account that the respective fixed energy- or utility-bound will be encoded into the state space of the model (see Section 3.5), and as a consequence this encoding results in an increased number of states for the model under consideration. The influence of the encoding for the produced utility is not of such a big deal and can be therefore addressed without any problems. But, the required energy budget to produce some utility is orders of magnitude larger than the utility resulting from this investment. Therefore, the encoding of the fixed energy bound results in a model where the state space is dramatically larger than without the inevitable encoding. This is the reason why the definition of $\mathrm{u}_{\max}$ has been changed slightly:

$$\begin{aligned}\mathrm{u}_{\max} &= \mathrm{qu}_s\big(\,\exists\mathbb{P}_{>p}(\Diamond^{\geqslant ?}(\mathrm{energy}_1 < \mathrm{e}))\,\big)\\ &= \max\big\{\hat{\mathrm{u}} \in \mathbb{N} : \mathsf{Pr}_s^{\max}(\ \Diamond(\mathrm{energy}_1 < \mathrm{e}\ \wedge\ \mathrm{utility}_1 \geqslant \hat{\mathrm{u}})\ ) > p\big\}\end{aligned}$$

In all the previous formulas, utility and energy correspond to the accumulated utility and energy rewards of all N processes, whereas $\mathrm{utility}_1$ and energy_1 stand for the accumulated utility and energy of process 1 only. Hence, $\mathrm{e}_{\min}$ describes a "global" quantile, whereas the final definition of $\mathrm{u}_{\max}$ stands for a "local" quantile. Considering the local accumulated rewards only with respect to process 1 and not with respect to another process is without loss of generality, since process 1 always consumes minimal energy compared to the processes i with $i \mod 3 = 0$ (see Table 6.21).

Analysis results The Energy-Aware Job-Scheduling Protocol was modelled in PRISM, encoding the program graphs detailed in Figure 6.8 and Figure 6.9 in the guarded-command input language of PRISM. Since both energy and utility are provided as reward functions in $\mathcal{M}$, the automata-based approach detailed in Section 3.5 was employed, e.g., the utility threshold u was encoded into the states of the model. This yields a model $\mathcal{M}_\mathrm{u}$ on which the quantile $\mathrm{e}_{\min}$ can be computed using the implementation for upper reward-bounded quantiles. The same pattern is used for computing the quantile $\mathrm{u}_{\max}$ in a model $\mathcal{M}_\mathrm{e}$, which arises from encoding the energy budget e into the states of the considered model $\mathcal{M}$, and afterwards the implementation for lower reward-bounded quantiles is used.

The Figures 6.10 and 6.11 document the results of the upper- and lower-bound quantiles detailed above and computed for the Energy-Aware Job-Scheduling Protocol, parameterising over the number N of processes. In Figure 6.10, $\mathrm{e}_{\min}$ for $\mathcal{M}_{\mathrm{u}}$ with $N = 2, \ldots, 6$ processes is shown with a fixed utility threshold of $\mathrm{u} = N$. It can be seen that the gap between the curves for $N = 2$ and $N = 3$ has a larger distance than the gap between the curves for $N = 3$ and $N = 4$. The same phenomenon is recognised when comparing the gap between the curves for $N = 4$ and $N = 5$ with the gap between the curves for $N = 5$ and $N = 6$. This can be explained by the fact that the energy consumption of every third process is doubled (see Table 6.21). The model for $N = 2$ does not contain any process with a doubled energy consumption, the models for process numbers between 3 and 5 contain exactly one process with an increased energy consumption, and the model for 6 processes contains two such processes. So, the doubled energy consumption of every third process groups the plots depending on the specific number of processes with an increased energy consumption.

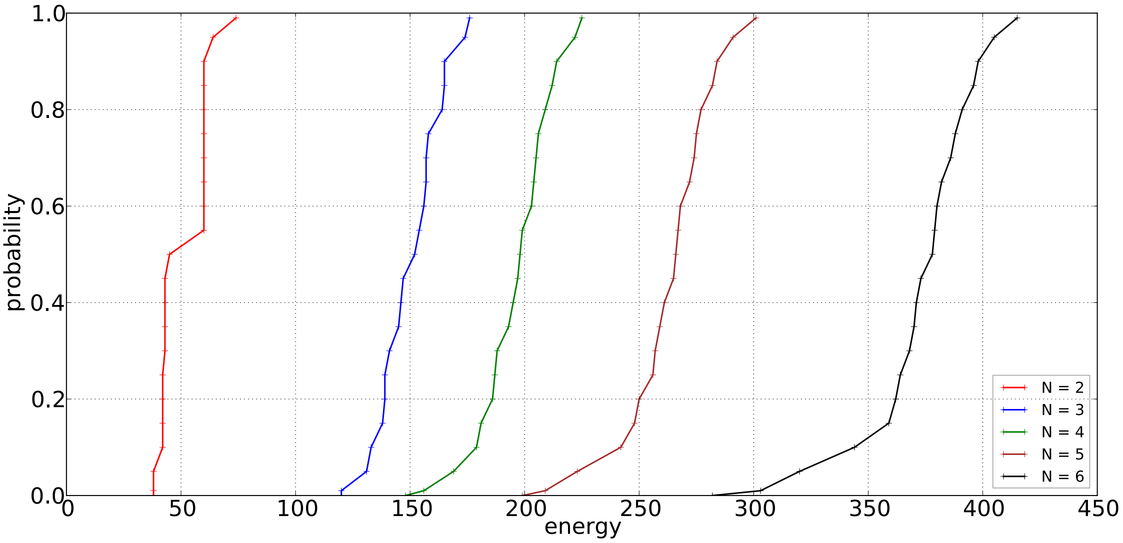

Figure 6.10: Results for quantile $\mathrm{e}_{\min}$ (Energy-Aware Job-Scheduling Protocol)

Figure 6.11 depicts $\mathrm{u}_{\max}$ for $N = 2, \ldots, 9$ processes and a fixed energy budget of $\mathrm{e} = 50 \cdot N$. It can easily be seen that the plots in Figures 6.10 and 6.11 have the typical shape for quantiles as depicted in Figure 3.1: Figure 6.10 clearly shows an increasing quantile for the minimal required energy budget, whereas Figure 6.11 represents a decreasing quantile for the maximal gained utility.

On page 123 the statistical data for the computation of quantile $\mathrm{e}_{\min}$ using the Explicit-engine can be seen in Table 6.23, and as well the corresponding statistics for the symbolic engines are depicted in Table 6.24. The statistics for quantile $\mathrm{u}_{\max}$ are listed in Table 6.25 (Explicit-engine) and in Table 6.26 (symbolic engines) on page 124. All tables illustrate that even for large model sizes with millions of states, the implementation of the *BP*-approach is feasible, whereas none of the quantile computations for $\mathrm{e}_{\min}$ and $\mathrm{u}_{\max}$ finished within 24 hours when trying to utilise the *LP*-approach instead of *BP*.

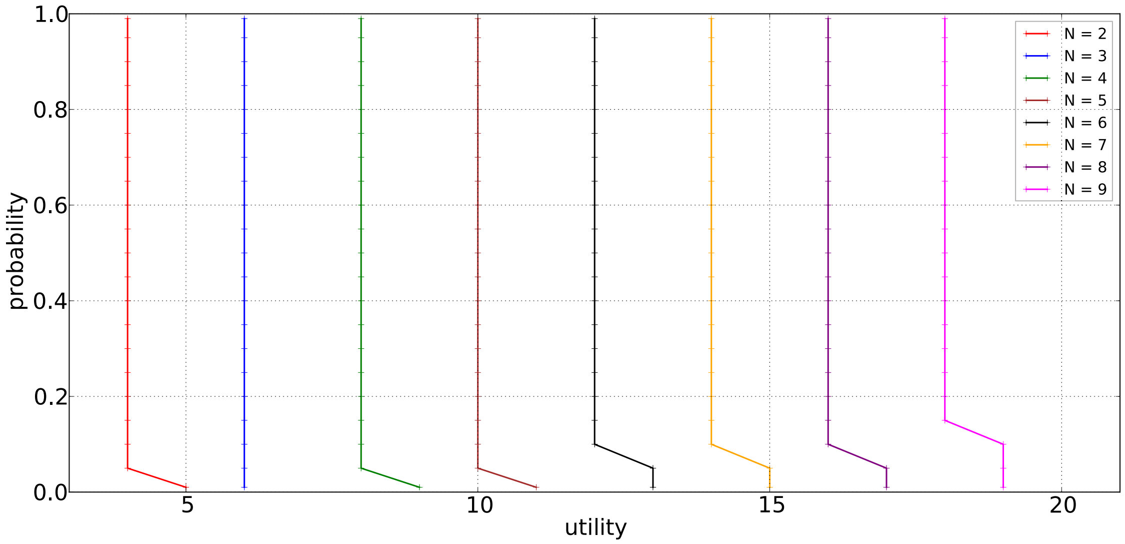

Figure 6.11: Results for quantile u_{max} (Energy-Aware Job-Scheduling Protocol)

As already recognised earlier the construction of the model using the EXPLICIT-engine is very time-consuming for both e_{min} and u_{max}. For instance, for e_{min} and $N = 6$ the EXPLICIT-engine needed nearly one hour in order to construct the model, whereas the symbolic engines did this construction process in only half a minute. But, the HYBRID-engine needed more than 13 hours for the calculation of the quantile itself, whereas the EXPLICIT-engine needed only half an hour in total using parallel computations. This is interesting since the symbolic representation of the state space is compact (1 394 808 nodes represent 44 072 357 states for $N = 6$), and since even the precomputation is much faster using the symbolic engines (EXPLICIT needs more than twice the time as the symbolic engines).

For the computation of u_{max} the things have changed completely, due to the fact that the precomputation for the EXPLICIT-engine is very slow compared to the symbolic engines. Since the calculation of u_{max} requires the computation of a lower-bounded quantile, the precomputation relies on an analysis of the maximal end components of the model (see Section 3.4.1). And the procedure for finding the end components seems to be implemented more efficiently for the symbolic engines. For example, the time for the precomputation for the case $N = 8$ needs nearly 7 hours, whereas the same procedure finishes in less than 5 seconds using any of the three symbolic engines. The solving-step of the linear program itself on the other hand is efficient when using the EXPLICIT-engine. The required 19 iterations could be entirely computed in less than 30 minutes, which corresponds more or less to the time the HYBRID-engine needs for this portion of the quantile computation. So, the overall computation time for the EXPLICIT-engine is completely dominated by the precomputation that relies on the methods shipped with PRISM in order to properly calculate the end components of the model. For $N = 9$ the precomputation needed even more than 20 hours, and that caused the time for the overall computation to be a little higher than the allowed 24 hours. So, Prisms current implementation for doing the end component analysis using the Explicit-engine seems to be a candidate for some improvement. It might be beneficial to further investigate this implementation, since the computation of end components is an important basis for many model-checking algorithms.

It can be seen in Tables 6.25 and 6.26 that the size of the model scales well due to the fact that the energy consumption and utility gain of just one selected process is considered. Therefore, the analysis can be carried out for a higher number of processes than it is the case for the computation of $\mathrm{e}_{\min}$, where the global energy consumption of all involved processes was considered.

Note on the storage of integer-sets for the Explicit-engine Please keep in mind that the timings given in Table 6.25 do utilise the data structures presented in Section 5.1.7. The statistics for the same computations using the standard BitSet-implementation shipped with Java in version 1.8.0_72-b15 is shown in Table 6.22. It can be seen that without the usage of the tailored data structures the performance of the computations is worse. For $N = 4$ the precomputation lasted roughly 20% longer, and the overall sequential computation was even 80% slower. For $N = 8$ the precomputation needed 30% more time, whereas the initialisation-phase of the *BP*-approach (see Section 5.1.1) even led to a memory-error. As part of this initialisation there happens the topological sorting of the zero-reward sub-**MDP**s (see Section 5.1.3). The resulting topological order needs to be stored somehow to be accessible during the particular iterations of the quantile calculations. Even for the case $N = 5$ a heap-space of 128 GB was not sufficient for this storage when using the standard BitSet-implementation for saving the topological order of the components. Whereas for the usage of the tailored data structures presented in Section 5.1.7 128 GB were completely fine for doing the complete quantile computation up to the case $N = 8$[10].

Table 6.22: Statistics for explicit quantile $\mathrm{u}_{\max}$ (without tailored storage, Energy-Aware Job-Scheduling Protocol)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2	12 828	0.3 s	7	0.36 s	0.53 s	0.66 s
N = 3	143 155	2.33 s	8	3.31 s	8.53 s	8.21 s
N = 4	872 410	15.67 s	11	48.11 s	211.28 s	140.67 s
N = 5	3 049 471	66.59 s	–	336.58 s	memory-error	
N = 6	7 901 694	181.10 s	–	1 996.73 s	memory-error	
N = 7	17 037 097	460.14 s	–	10 704.06 s	memory-error	
N = 8	32 442 898	994.94 s	–	34 657.52 s	memory-error	

---

[10] Since the model for the case $N = 9$ has a huge number of reachable states the heap-space for the Java-process needed to be increased, and was set to 250 GB.

Table 6.23: Statistics for explicit quantile $e_{\min}$ (Energy-Aware Job-Scheduling Protocol)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2	917	0.09 s	75	0.09 s	0.16 s	0.19 s
N = 3	16 341	0.62 s	177	0.24 s	0.69 s	0.6 s
N = 4	368 521	17.18 s	226	2.19 s	17.78 s	9.37 s
N = 5	6 079 533	341.22 s	302	41.35 s	281.58 s	149.05 s
N = 6	44 072 357	3 217.4 s	416	465.2 s	3 423.97 s	1 754.88 s

Table 6.24: Statistics for symbolic quantile $e_{\min}$ (Energy-Aware Job-Scheduling Protocol)

	Model				symbolic computations					
					Hybrid		Sparse		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 2	917	2 169	0.04 s	75	0.02 s	0.06 s	0.02 s	0.05 s	0.02 s	0.24 s
N = 3	16 341	15 664	0.16 s	177	0.16 s	1.73 s	0.16 s	0.6 s	0.16 s	9.32 s
N = 4	368 521	103 052	1.34 s	226	3.3 s	90.42 s	3.19 s	21.34 s	3.21 s	177.82 s
N = 5	6 079 533	189 841	7.34 s	302	16.61 s	1 958.47 s	16.47 s	529.55 s	17.16 s	1 408.65 s
N = 6	44 072 357	1 394 808	37.32 s	416	177.71 s	47 750.14 s	139.77 s	6 646.87 s	138.65 s	13 232.38 s

Table 6.25: Statistics for explicit quantile $\mathrm{u}_{\max}$ (Energy-Aware Job-Scheduling Protocol)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2	12 828	0.32 s	7	0.42 s	0.6 s	0.78 s
N = 3	143 155	2.45 s	8	3.26 s	4.28 s	4.94 s
N = 4	872 410	16.08 s	11	38.56 s	42.65 s	50.64 s
N = 5	3 049 471	71.27 s	13	273.96 s	332.75 s	361.17 s
N = 6	7 901 694	188.22 s	15	1 558.49 s	1 858.31 s	1 976.72 s
N = 7	17 037 097	470.75 s	17	6 852.92 s	7 462.75 s	8 465.79 s
N = 8	32 442 898	1 026.66 s	19	24 669.21 s	26 325.71 s	29 921.42 s
N = 9	56 485 515	2 100.13 s	21	74 737.12 s	88 387.07 s	86 970.27 s

Table 6.26: Statistics for symbolic quantile $\mathrm{u}_{\max}$ (Energy-Aware Job-Scheduling Protocol)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 2	12 828	6 310	0.12 s	7	0.23 s	0.41 s	0.23 s	0.36 s	0.23 s	0.48 s
N = 3	143 155	11 247	0.22 s	8	0.49 s	1.66 s	0.52 s	1.28 s	0.45 s	4.33 s
N = 4	872 410	14 007	0.35 s	11	0.7 s	12.12 s	0.68 s	6.25 s	0.66 s	32.69 s
N = 5	3 049 471	25 363	0.62 s	13	1.11 s	82.81 s	1.09 s	27.54 s	1.09 s	155.24 s
N = 6	7 901 694	38 911	1.06 s	15	2.18 s	246.9 s	2.05 s	87.89 s	2.18 s	462.43 s
N = 7	17 037 097	55 545	1.7 s	17	2.87 s	630.5 s	2.87 s	228.73 s	2.91 s	1 448.54 s
N = 8	32 442 898	80 782	2.64 s	19	4.75 s	1 519.12 s	4.88 s	512.91 s	4.67 s	2 647.77 s
N = 9	56 485 515	104 246	4.14 s	21	7.55 s	3 879.8 s	8.42 s	1 039.58 s	7.58 s	5 240.73 s

6.2.1.1 Energy-Aware Job-Scheduling Protocol with side conditions

Now, we want to analyse the minimal energy consumption as before, but we want to take specific side conditions into account. Therefore, quantiles with side conditions (see Section 3.6) are used to carry out the upcoming analysis. Formally, we are interested in the computation of

$$\mathrm{e}^{\varphi}_{\mathrm{min}} = \min\{\hat{\mathrm{e}} \in \mathbb{N} : \mathsf{Pr}^{\max}_{s}(\Diamond(\mathrm{energy} \leqslant \hat{\mathrm{e}} \ \wedge \ \mathrm{utility} > \mathrm{u}) \wedge \varphi) > p\}$$

where φ is defined according to the desired side condition.

Prohibit turbo mode: We want to get a feeling for the impact of the deactivation of the turbo mode on the energy consumption of the protocol. Therefore, we like to disable the turbo mode for our analysis, and use the following side condition:

$$\varphi = \Box(\mathit{turbo} = \mathsf{false})$$

The side condition that is induced by φ is an invariance, and therefore it needs to hold for all reachable states of the investigated model. The results of the analysis are depicted in Figure 6.12 and the computational statistics can be found in Table 6.27.

The reference-values depicted in Figure 6.10 also appear as transparent shapes in Figure 6.12 in order to allow a better visualisation of the impact of the deactivation of the turbo mode. As can be seen this deactivation dramatically increases the energy consumption of the protocol in order to achieve the same utility as before. For $N = 2$ there is not a huge difference between the case without side conditions and

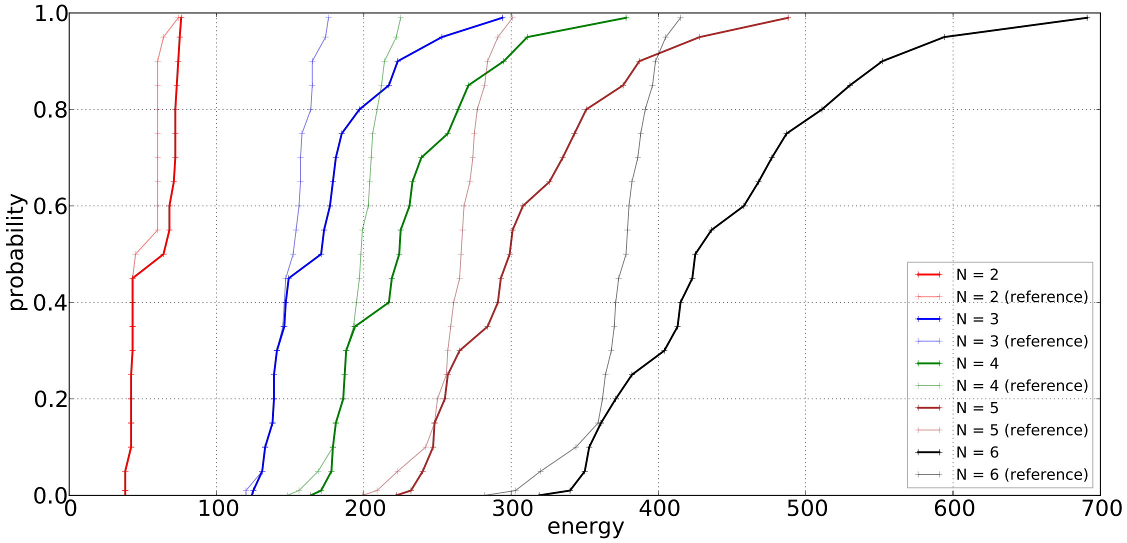

Figure 6.12: Results for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, no turbo mode)

Table 6.27: Statistics for quantile e_{min} with side conditions (Energy-Aware Job-Scheduling Protocol, no turbo mode)

Instance	$\mathcal{A}$ States	Model Product States	$t_{\text{transform}}$	Iters	EXPLICIT t_{pre}	EXPLICIT sequential t_{query}	EXPLICIT parallel t_{query}
N = 2	3	1 329	0.18 s	77	0.07 s	0.31 s	0.37 s
N = 3	3	20 999	0.64 s	295	0.27 s	1.41 s	1.3 s
N = 4	3	465 603	9.42 s	379	5.14 s	41.92 s	22.32 s
N = 5	3	6 798 695	107.89 s	489	230.3 s	689.03 s	424.1 s
N = 6	3	47 227 099	1 096.43 s	692	3 798.33 s	7 744.49 s	5 930.83 s

the case where the turbo mode has been prohibited. But for an increased number of processes there exists a specific probability where the results of both cases start to differ enormously. For $N = 3$ this difference starts at a probability of around 0.45, and for $N = 4$ the boundary is at a probability of roughly 0.35. With an increase in the number of involved processes this probability is lowered even more, resulting in the fact that the energy consumption is getting even worse when the number of involved processes increases. In the end, the invested energy needs to be increased by nearly 40% in order to achieve the desired utility with a probability of 99% (except for the case $N = 2$). In conclusion, it can be said that it pays off to invest three times the energy (when turbo mode is active) for a computation twice as fast.

Since currently only the EXPLICIT-engine supports the computation of quantiles with side conditions there are only statistics for this engine available. As explained in Section 3.6 the algorithm works with a deterministic automaton $\mathcal{A}$ encoding φ. Here, $\mathcal{A}$ consists of 3 states together with one Rabin-pair (see [GTW02]), and the parallel composition of $\mathcal{A}$ to the model presented previously results in the number of states depicted in Table 6.27 (compare Table 6.23 for statistics of the original model). Therefore, the model under consideration is a bit larger than the original model. The column $t_{\text{transform}}$ in Table 6.27 depicts the time that was needed to transform the original model to the model used for the analysis. This transformation includes the building of the deterministic Rabin automaton $\mathcal{A}$ (consisting of 3 states for this case), and the subsequent construction of the product-model out of the original model and automaton $\mathcal{A}$, followed by the necessary model-transformations described in Section 3.6 for computing quantiles with side conditions. As it is clear that there exists the necessity to do a transformation soonest after the complete quantile query has been parsed, the time for doing the transformation ($t_{\text{transform}}$) is included in the time for computing the complete quantile (t_{query}). That explains the increased quantile-calculation times in comparison to, e.g., Table 6.23 where no side conditions were demanded.

Force turbo mode after first deadline-miss: When there exists a task which could not be handled in time (its deadline has not been kept) by a specific process we want to give any further job the opportunity to finish in time by the means that are supplied by the protocol. Therefore, we do activate the turbo mode for all incoming calculations as soon as a deadline has been missed for the first time. The corresponding side condition can be specified as follows:

$$\varphi = \Box \left(\left(\bigvee_{p=1}^{N} \textsc{crit}_p \wedge c_p = 0 \wedge d_p = 0 \right) \implies \Box(\mathit{user} \neq \bot \implies \mathit{turbo} = \mathsf{true}) \right)$$

Here, the boolean formula $\textsc{crit}_p \wedge c_p = 0 \wedge d_p = 0$ describes a situation where the deadline of a job assigned to process p has not been kept. This event triggers the activation of the turbo mode for all incoming jobs, and from now on there is only the possibility of doing all the critical calculations twice as fast. But, as a drawback the energy consumption inside the critical section is three times higher (see Table 6.21). Since the turbo mode only influences computations done in their respective critical section there is no need to make any demand on the usage of the turbo when there currently exists no process located in its critical section (defined by $\mathit{user} = \bot$). So, we only specify the usage of the turbo when the resource is occupied by some process ($\mathit{user} \neq \bot$).

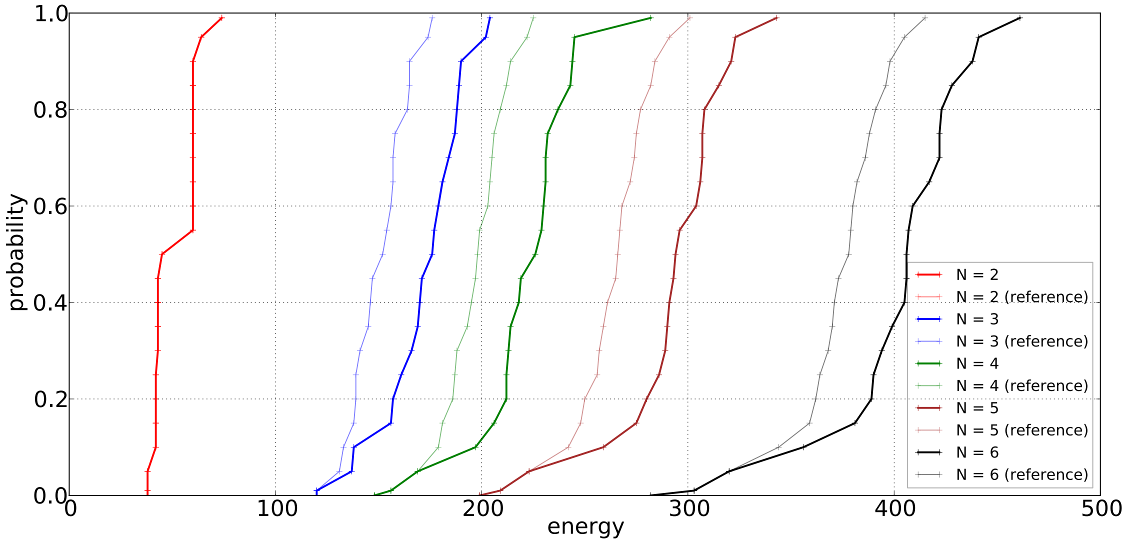

Figure 6.13: Results for quantile e_{min} with side conditions (Energy-Aware Job-Scheduling Protocol, enable turbo after first miss)

The results of the analysis are depicted in Figure 6.13 and the corresponding statistics can be found in Table 6.28. As before, the results from Figure 6.10 are shown as well to allow a direct comparison to the case without any side conditions. As can be seen the consumed energy is increased in comparison to having the freedom of activating or deactivating the turbo mode at will. This is in a way clear, since the desired side condition eliminates a possibility for adapting the system to the current workload dynamically. But, the consumed energy does not increase to such an extent as it is the

Table 6.28: Statistics for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, enable turbo after first miss)

Instance	$\mathcal{A}$ States	Model Product States	$t_{\mathrm{transform}}$	Iters	EXPLICIT t_{pre}	EXPLICIT sequential t_{query}	EXPLICIT parallel t_{query}
N = 2	4	2 107	0.23 s	75	0.07 s	0.41 s	0.47 s
N = 3	4	34 115	1.08 s	205	0.33 s	2.31 s	1.94 s
N = 4	4	606 831	14.38 s	283	5.35 s	59.17 s	29.34 s
N = 5	4	9 180 455	245.75 s	344	267.9 s	1 460.07 s	700.21 s
N = 6	4	65 361 499	3 177.46 s	462	6 709.81 s	21 755.49 s	13 345.27 s

case when deactivating the turbo mode completely (compare to the results shown in Figure 6.12). So, the bottom line is that it is a better choice to keep the turbo mode active instead of ignoring it completely.

The computation statistics reveal that the parallelisation of this analysis is beneficial since the calculation times were almost halved (only for $N = 2$ the parallelism slightly slowed the overall performance due to the necessary overhead).

Round-robin scheduling: Quantiles with side conditions turn out to be useful for the analysis of specific scheduling mechanisms. Therefore, we want to further investigate round-robin scheduling in this paragraph. Round-robin scheduling is a method in order to allow a fair scheduling, such that all involved processes are able to perform their computation tasks. A formal specification tailored to the needs of the Energy-Aware Job-Scheduling Protocol and using temporal logics can be done in the following way:

$$\varphi = \Box\Bigg(\bigwedge_{p=1}^{N}\Big(\mathit{user} = p \implies \\ (\mathit{user} = p \vee \mathit{user} = \bot)\,\mathcal{U}\,(\mathit{user} = p + 1 \mod N)\Big)\Bigg)$$

Here, φ adapts to the number of used processes (N), and therefore φ becomes more complex as N increases. This entails that the automaton used for the encoding of φ grows with N. So, the parallel composition of the automaton and the model of the protocol becomes more expensive for larger N.

Figure 6.14 depicts the results when a round-robin scheduler will be chosen in order to manage access to the shared resource. Since the results for $N = 5$ are really large, Figure 6.15 shows a more detailed view to depict the results for smaller N. As can be seen for $N = 2$ the usage of a round-robin scheduler delivers the same performance as without a specific scheduling strategy (compare Figure 6.10). This can be explained by the fact that there are only two processes involved. Since the protocol needs to keep the deadlines for the tasks by simultaneously minimising the energy consumption the protocol has no other choice than alternating the access to the shared resource for

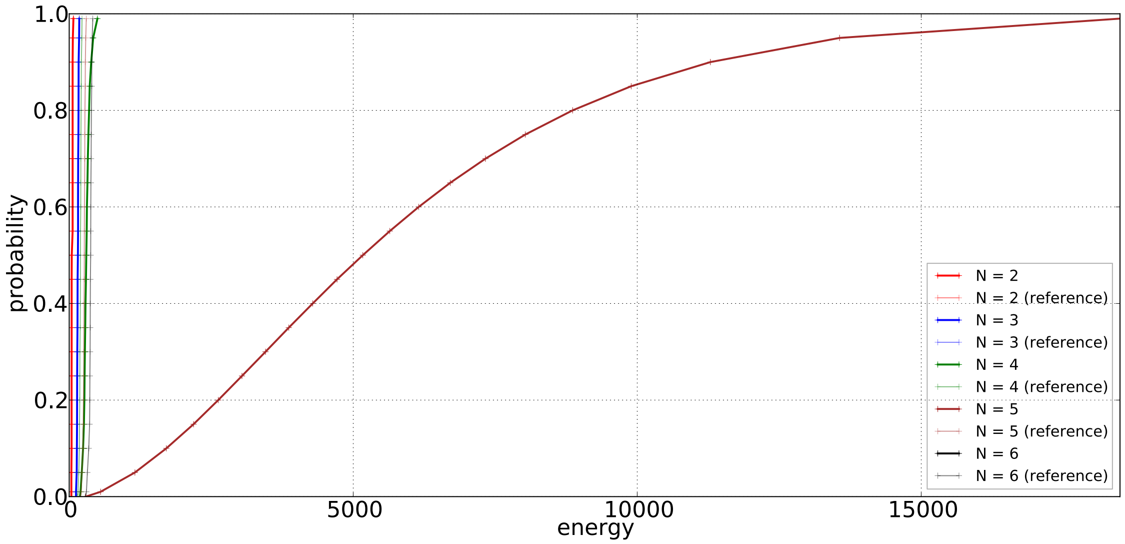

Figure 6.14: Results for quantile $e_{\min}$ with side conditions (Energy-Aware Job-Scheduling Protocol, round-robin scheduling)

Table 6.29: Statistics for quantile $e_{\min}$ with side conditions (Energy-Aware Job-Scheduling Protocol, round-robin scheduling)

	Model				Explicit		
	$\mathcal{A}$	Product				sequential	parallel
Instance	States	States	$t_{\text{transform}}$	Iters	t_{pre}	t_{query}	t_{query}
N = 2	11	942	0.19 s	75	0.06 s	0.33 s	0.4 s
N = 3	89	22 945	1.06 s	178	0.25 s	1.77 s	1.7 s
N = 4	459	414 314	16.61 s	498	4.05 s	42.74 s	26.08 s
N = 5	2 109	6 249 946	471.56 s	18 501	853.71 s	4 981.67 s	2 986.43 s
N = 6	9 131	44 212 114	58 257.76 s	0	2 894.49 s	61 158.43 s	

the two processes. Therefore the protocol behaves implicitly as if it is scheduled by a round-robin scheduler in this case. This is also true to some extent for $N = 3$. But, for more than three processes the energy consumption of the round-robin scheduling increases dramatically. The point here is that the energy consumption of every third process is increased (see Table 6.21) and the scheduler forces the usage of those more expensive processes whenever they are next in line. Therefore, a more expensive process will be scheduled regardless if there is a cheaper process that could do a task. Another factor is that the considered scheduling forces to do the calculation in a fixed order. Therefore, no dynamic reactions to environmental demands are possible, and the waiting times of the participating processes (and their energy consumption) increase as well. This also causes more deadline-misses to the protocol due to the fact that processes that wait for access to the shared resource are forced to drop their working package since the access to the shared resource cannot be granted in time. So, all

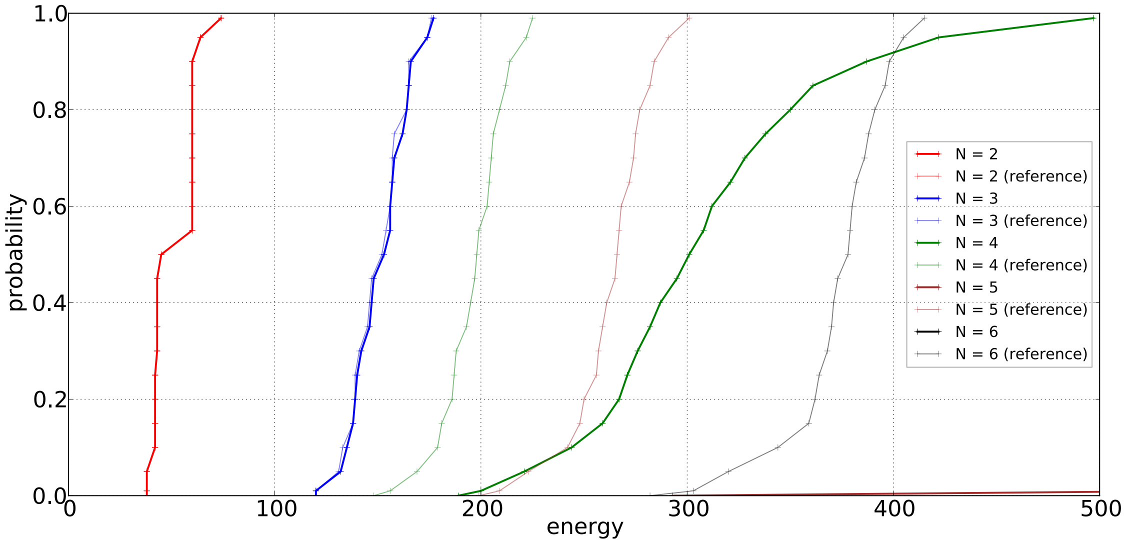

Figure 6.15: Results for quantile $e_{\min}$ with side conditions (Energy-Aware Job-Scheduling Protocol, round-robin scheduling, zoomed)

the energy that was spent in the uncritical section is gone without any outcome and another energy-portion needs to be spent for the completion of another job. For $N = 6$ it is even the case that the desired amount of utility cannot be obtained using the proposed scheduling-technique.

As can be seen in Table 6.29 the calculation times for the computation of the quantiles grow fast when considering more than four processes. The reason is that the states of the model grow and the most important fact is that the number of required iterations is really high for the case $N = 5$. For $N = 6$ no iterations were needed since according to the precomputation there is no chance of achieving the desired utility in this case. But the transformation process for the model, that contributes to the time for the quantile query when considering quantiles with side conditions, is expensive here. The most time-consuming part was the translation of φ to a representation using a deterministic automaton $\mathcal{A}$, which needed around 15 hours. Since φ depends on the number of involved processes the automaton representing φ has different characteristics for the different instances of the protocol. Therefore the number of states for $\mathcal{A}$ increases with the value of N. After building the product model the quantile computation performed in the usual manner.

In conclusion it can be seen that the usage of quantiles with side conditions is an elegant way of analysing management strategies without the need of manually encoding the strategies into the state space of the model, and this allows to avoid errors that can occur in the modelling phase.

6.2.1.2 Energy-Aware Job-Scheduling Protocol and expectation quantiles

For the previously considered energy-utility analysis we had to fix one of the two quantities (either the energy or the utility) and encode those fixed values into the state space of the model (see Section 3.5), and this directly contributes to an increase in the number of states of the model. Instead, when interested in the minimal energy consumption for an arbitrary expected amount of utility we can help ourselves by using expectation quantiles as introduced in Chapter 4, and calculate the minimal energy that needs to be spent in order to gain this utility using the methods presented there. We therefore compute the quantile query

$$E_s^{\exists} = \mathrm{qu}_s\big(\exists \mathrm{ExpU}_{>\mathrm{u}}(\mathrm{energy} \leqslant ?)\big)$$

for doing the analysis, where u can be specified arbitrarily.

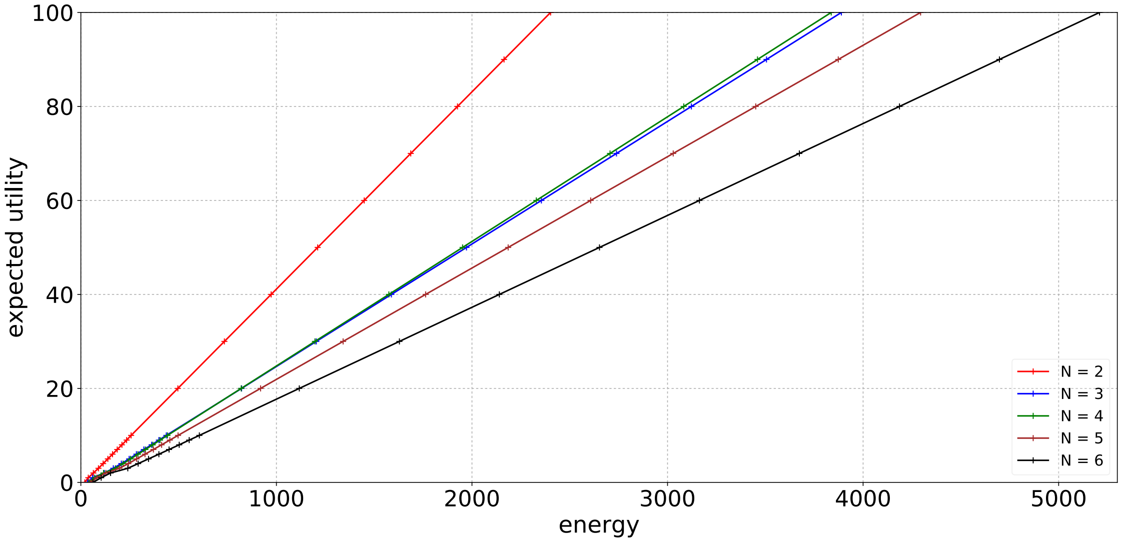

Figure 6.16: Results for quantile $E_s^{\exists}$ (Energy-Aware Job-Scheduling Protocol)

Table 6.30: Statistics for quantile $E_s^{\exists}$ (Energy-Aware Job-Scheduling Protocol)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 2	349	0.05 s	2 403	0.11 s	0.27 s	0.4 s
N = 3	4 584	0.22 s	3 890	0.24 s	2.34 s	1.3 s
N = 4	78 289	1.86 s	3 838	1.66 s	70.93 s	18.39 s
N = 5	1 032 733	28.06 s	4 295	26.72 s	688.21 s	239.7 s
N = 6	6 367 337	180.23 s	5 210	169.27 s	4 974.5 s	1 698.98 s

The results of the analysis for $\mathrm{u} \in \{0, 1, 2, \ldots, 10, 20, 30, \ldots, 100\}$ are depicted in Figure 6.16 and the corresponding statistics can be found in Table 6.30. As can be seen the state space of the analysed models is much smaller when compared to the number of states for the reachability quantile $\mathrm{e}_{\mathrm{min}}$ (see Table 6.23 on page 123). As already stated, the reason is that the desired goal for the computation does not need to be encoded into the state space of the model, and instead an additional reward function for the gained utility is used. This would also allow to analyse the protocol for even higher numbers of N, but since the previously investigated quantiles $\mathrm{e}_{\mathrm{min}}$ used $N = 6$ as its highest number of processes we do not exceed six here. The parallel computation is really beneficial here, since due to the number of required iterations there exist so many iterations that they can easily pay off for the additional parallel overhead that needs to be invested. So, the whole computation profits from the parallel approach, and we were able to reduce the overall computation time by roughly 65% in the case $N = 6$.

Figure 6.16 shows that the energy consumption of $N = 3$ and $N = 4$ is nearly the same, and the curve for $N = 5$ appears also rather close to both curves. The reason is that more processes mean a higher energy consumption, but at the same time more processes can produce the desired utility in a shorter period of time. Therefore, the desired utility-gain is reached earlier and this helps to keep the energy consumption at a comparable level. For $N = 5$ there occurs a noticeable shortage of the shared resource more frequently (in comparison to $N = 3$ or $N = 4$), and therefore the processes need to wait for the resource for longer times (increasing their needed energy due to waiting). With an increased number of involved processes the impact of this shortage-effect will increase as well. But, the major part of the increased energy consumption can be illustrated with the help of Table 6.21. As already explained for the analysis of $\mathrm{e}_{\mathrm{min}}$ (Figure 6.10) there exist groups depending on the specific number of processes with an increased energy consumption (every third process consumes twice as much energy). This is also the case here, and therefore the energy consumption of $N = 3$, $N = 4$ and $N = 5$ is quite similar, whereas $N = 2$ consumes much less energy, and the consumption of $N = 6$ is much higher.

6.2.1.3 Multiple shared resources

For the Energy-Aware Job-Scheduling Protocol there exists also a variant where there are two shared resources instead of just one. In order to have a second resource, a duplication of the model for the resource (see Figure 6.9) was composed to $\mathcal{M}$. As a consequence, two processes can enter their respective critical section simultaneously without the risk of blocking each other. Since mutual exclusion effects do not appear when analysing the case $N = 2$, it is not very interesting to analyse the protocol in this case. Therefore, the analysis will be started for at least 3 processes, and we will concentrate on the computation of $\mathrm{e}_{\mathrm{min}}$ and $\mathrm{u}_{\mathrm{max}}$ as before. The results are depicted in Figure 6.17 and in Figure 6.18. Since there is an additional resource it is more likely for a process to access one of the shared resources, and therefore the waiting times for starting the critical section are lower. The benefits of this fact can be seen

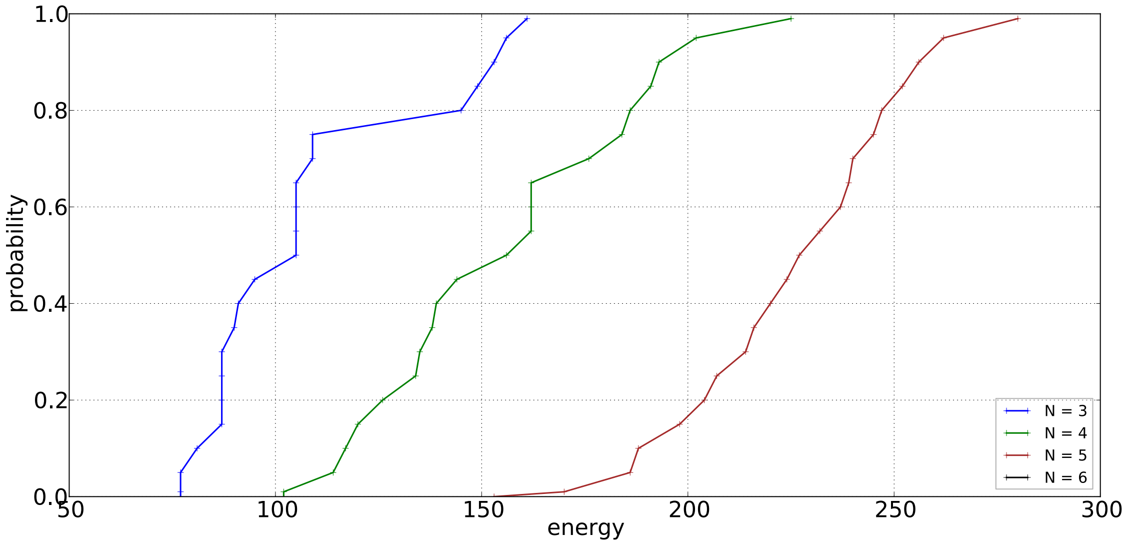

Figure 6.17: Results for quantile $e_{\min}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources)

in Figure 6.17 in terms of reduced energy-costs for accumulating the same amount of utility as for the case of one shared resource (compare Figure 6.10). In Figure 6.18 the second resource ensures that the same amount of utility can be accumulated with much higher probability (compare to Figure 6.11).

The corresponding computation statistics for quantile $e_{\min}$ can be found in Table 6.31 (Explicit-engine) and Table 6.32 (symbolic engines) on page 135. In contrast to just one shared resource, the scaling of the model becomes worse, and therefore it was only possible to receive results for up to five processes. Due to the number of reachable states (1 071 272 795) the Explicit-engine was not able to build the model for N = 6, and even the Sparse-engine failed to do the calculations because of memory-issues. Only the MTBDD- and Hybrid-engine would have been able to perform the calculations. But, since a timeout of 24 hours was set, the result could not be computed in time. So, there are no results for six processes with two shared resources. Up to five processes the Explicit-engine performs well. As already mentioned earlier it suffers from the model-building procedure, but the computation-times for the quantiles are short. Thanks to the size of the model and also the number of required iterations the parallel computation is really an option here.

Table 6.33 (Explicit-engine) and Table 6.34 (symbolic engines) on page 136 show the statistics for the computation of quantile $u_{\max}$. As expected, the used engines perform in the way as they already did for the case of one shared resource. For example, the precomputation for the Explicit-engine again constitutes the bigger part of the calculation time. So, there are not that many new facts for the protocol-variant that have not already been discussed for the case of a single resource. But, one interesting observation is that the scaling behaviour of the model changes dramatically when an

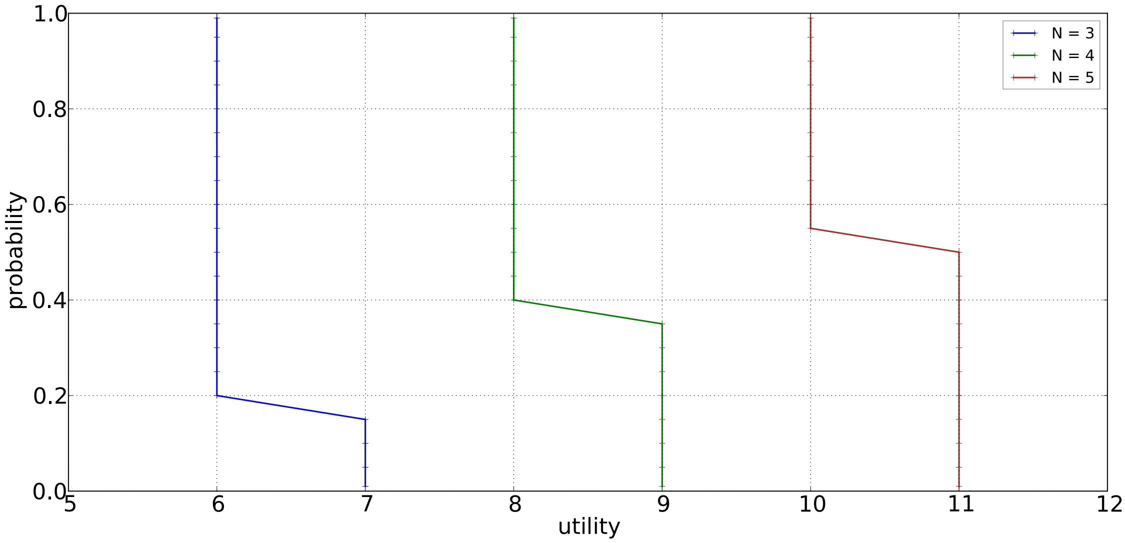

Figure 6.18: Results for quantile u_{max} (Energy-Aware Job-Scheduling Protocol, 2 shared resources)

additional resource is being attached to the protocol. Even the circumstance that the energy and utility of just one single process is considered does not provide a good scaling of the model. This leads to the fact that PRISM's EXPLICIT-engine is not able to handle the states of the model, even when N is relatively small. Therefore, the symbolic engines are handy in this case, since they allow to handle models with more involved processes. One approach that might be beneficial here for tackling the bad scalability of the model, is the symmetry-reduction as proposed in [Bai+12a] or [Bai+15]. Especially for u_{max} this could lead to an improved scaling since the behaviour of only one process is investigated, whereas the energy and utility of the remaining processes is not of interest. So, these other processes could be identified using counters, and in the end it would be possible to obtain a more condensed model hopefully allowing to investigate a higher number of processes.

Table 6.31: Statistics for explicit quantile $e_{\min}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 3	57 507	4.4 s	162	0.56 s	2.44 s	1.64 s
N = 4	1 346 711	230.95 s	226	7.62 s	164.41 s	35.44 s
N = 5	36 858 673	12 229.19 s	281	308.59 s	3 175.97 s	1 963.91 s

Table 6.32: Statistics for symbolic quantile $e_{\min}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources)

	Model				symbolic computations					
					HYBRID		SPARSE		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 3	57 507	15 925	0.33 s	162	0.23 s	10.35 s	0.23 s	2.24 s	0.23 s	16.61 s
N = 4	1 346 711	310 280	7.8 s	226	12.82 s	1 589.31 s	11.2 s	115.97 s	8.6 s	1 075.13 s
N = 5	36 858 673	3 710 514	181.16 s	281	212.15 s	> 24 h	296.48 s	5 841.36 s	304.96 s	49 016.64 s
N = 6	1 071 272 795	50 516 243	14 368.66 s	–	8 040.65 s	> 24 h	8 241.34 s	error	8 052.54 s	> 24 h

Table 6.33: Statistics for explicit quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
N = 3	983 661	50.15 s	9	40.0 s	45.54 s	49.27 s
N = 4	16 461 405	1 915.44 s	11	4 835.13 s	5 449.01 s	5 032.64 s

Table 6.34: Statistics for symbolic quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources)

	Model				symbolic computations					
					Hybrid		Sparse		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
N = 3	983 661	56 789	0.86 s	9	2.46 s	29.9 s	2.45 s	7.43 s	2.91 s	14.16 s
N = 4	16 461 405	288 368	9.04 s	11	28.76 s	2 473.99 s	28.46 s	132.63 s	30.66 s	389.12 s
N = 5	192 687 895	2 218 215	294.42 s	13	853.62 s	> 24 h	707.54 s	2 447.46 s	722.21 s	14 629.2 s

6.2.2 Energy-Aware Bonding Network Device (eBond)

The eBond protocol tries to reduce the energy consumption of network devices in server systems by adapting to dynamically changing bandwidth demands. The key feature of this reduction relies on the fact that typical server systems are equipped with redundant infrastructures for network communication in order to compensate the breakdown of such a device and therefore ensure the availability of the system. If those different network devices are now chosen such that they have different specifications in terms of consumed energy and provided bandwidth there arises the opportunity of adapting the energy consumption of the overall network communication to dynamically changing bandwidth demands. So, in a specific workload situation the network device will be utilised that is able to provide the required bandwidth and has the least energy consumption. The protocol was published in [Häh+13] and the authors evaluated the protocol using a tailored simulator. We did a formal PMC-based analysis of the protocol in collaboration with the developers of the protocol, and the analysis utilised the quantile framework discussed here.

The protocol consists of several network interface cards (NICs) offering different performance in terms of maximal provided bandwidth. The energy characteristics of the respective cards are hereby different for themselves. The main idea of the protocol is that the system uses the card with the lowest power consumption as long as the provided bandwidth is enough to serve the bandwidth needed to handle the requests to the server. This will save energy as long as cards with higher power consumption can be turned off in favour of using cards with a lower power consumption such that the provided bandwidth still meets the requirements for the needed bandwidth. If on the other hand the required bandwidth exceeds the offered bandwidth of the system, then a card supplying higher bandwidths must be used as this will avoid package delays. Of course, the energy consumption of the system will increase for such situations.

In [Häh+13] the authors picked two exemplary NICs at varying bandwidths, and chose on the one hand the card EXPI9301CTBLK[11], providing a maximum bandwidth of 1 GBit per second at the cost of an average power consumption of around 1.35 W to 1.92 W (see [Häh+13, Table 1]). The second chosen NIC is the X520-T2[12] card, which provides a maximum bandwidth of 10 GBit per second and has a power consumption of around 7.88 W to 8.10 W (see [Häh+13, Table 1] as well).

In [Häh+13] the authors stated that this approach allows to save up to 75% of the consumed energy for the network connection of the server. This was done using a tailored simulation based on precise measurements for the different energy profiles. The authors also came up with three different energy-saving heuristics used for controlling the activation/deactivation of the involved NICs and adapting the system to changing requirements (compare [Häh+13, Table 2]):

---

[11] `https://www-ssl.intel.com/content/www/us/en/ethernet-products/gigabit-server-adapters/gigabit-ct-desktop-adapter-brief.html`, retrieved 28th March 2018

[12] `http://www.intel.de/content/www/de/de/ethernet-products/converged-network-adapters/ethernet-x520-t2.html`, retrieved 28th March 2018

***aggressive*:** Whenever there is a situation where the requests did not need more than 1 GBit/s within the last time step the NIC EXPI9301CTBLK is utilised and X520-T2 will be deactivated.

This heuristics saves the highest amount of energy but it also provides the highest rate of package drops, as the 10 GBit-card will be turned off as soon as possible.

***high savings*:** The switching from the 10 GBit-card to the 1 GBit-card is done like for *aggressive*, but a predictor of 10% more bandwidth will be used. The predictor works as a kind of buffer that increases the required bandwidth for the last time step by the specified percentage value. So, the required bandwidth for the last time step is higher than it is for *aggressive* and therefore switching to a card with a lower energy consumption is only possible when the elevated required bandwidth can be served.

This heuristics produces less package drops than *aggressive*, but on average it also consumes more energy.

***balanced*:** This heuristics works as *high savings*, but an additional cooldown timer of 30 minutes is used. The cooldown timer serves to counteract situations where the bandwidth requirements drop unexpectedly but will reach the original level very soon again.

The *balanced* heuristics provides a high degree of service-availability but at the same time the energy consumption is also the highest out of the three presented heuristics.

Formal analysis The analysis described in the following concentrates on an investigation of the mentioned energy-saving heuristics, and it is based on real server logs equipped with information on the bandwidth required for the network communication. The traffic was observed over 43 days for a Debian/Ubuntu FTP mirror, and every 5 seconds the bandwidth requirements were recorded (see [Häh+13]). The presented formal analysis is based on those logs, but we will use a coarser grid for the time and for the precision of the bandwidth requirements. The grid for the time has a resolution of 4 minutes, meaning that 1 time step in the model corresponds to 4 minutes in the recorded data. The precision of the bandwidth on the other hand has been set to 50 MBit, describing that 1 unit for the bandwidth requirements in the model is associated with a bandwidth requirement of 50 MBit/s in the recorded data. This coarser grid is necessary since it allows to obtain model sizes that can be analysed by the means of PRISM using the presented quantile framework. All observed days were folded into one single day, allowing to have all bandwidth requirements probabilistically distributed for this folded day. Therefore, all the recorded information could be taken into account for the analysis. The folded bandwidth profile was then formalised using PRISM's guarded command language, and combined with a formalisation of the measured power profiles for the different NICs (see [Häh+13, Figure 2]). In the end the model was enriched by a formalisation of the three different energy-saving heuristics, and finally the formal analysis utilising the discussed quantile framework was done.

The analysis concentrates on minimising the energy consumption for ensuring that there is no package delay in u per cent of the time until the day has ended. This should be certificated with a high confidence of at least p. The following quantile query formalises this requirement for probability threshold p:

$$\mathrm{e}^{\mathrm{u}}_{\mathrm{min}} = \min\{\mathrm{e} : \mathsf{Pr}^{\max}_{s}(\ \Diamond^{\mathrm{energy} \leqslant \mathrm{e}} \mathrm{EndOfDay} \wedge \mathrm{utility} > \mathrm{u}\ ) > p\}$$

The results of the analysis for $\mathrm{u} = 95\%$ are depicted in Figure 6.19 and the corresponding runtime characteristics can be found in Table 6.35.

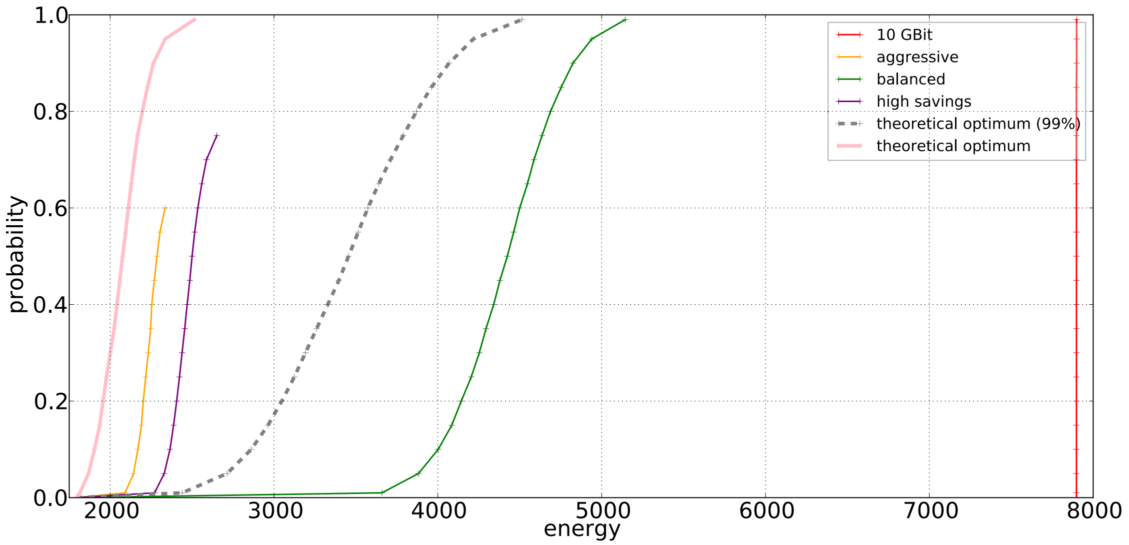

Figure 6.19: Results for quantile $\mathrm{e}^{95\%}_{\mathrm{min}}$ (eBond)

Table 6.35: Statistics for quantile $\mathrm{e}^{95\%}_{\mathrm{min}}$ (eBond)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
10 GBit	9 749	0.6 s	7 900	0.23 s	12.46 s	4.82 s
aggressive	516 309	21.75 s	2 336	13.37 s	182.1 s	130.32 s
balanced	1 634 188	74.16 s	5 145	33.52 s	912.61 s	498.6 s
high savings	577 137	24.39 s	2 652	14.87 s	227.79 s	147.64 s
theoretical optimum (99%)	28 234 049	1 116.14 s	4 514	97.11 s	2 712.14 s	2 507.43 s
theoretical optimum	30 462 467	1 182.41 s	2 514	122.69 s	3 080.04 s	2 596.8 s

It can be seen that choosing the *aggressive* heuristics has the least energy consumption, but at the same time its provided level of service-availability is poor compared to, e.g., the *balanced* heuristics. It is only possible to provide a guarantee that 95% of the packages will not drop with a certainty of roughly 60%. So, when there are cases where the service-availability is very important, it is not a good idea to stick to the *aggressive* strategy, since the risk of package drops due to insufficient provided bandwidth is too high. But, when the energy consumption of the system is more important than the availability of the services, the *aggressive* strategy might be a good choice. The *high savings* heuristics provides a better degree of service-availability at the cost of a bit more energy compared to *agressive*. But, in the end the trade-off between consumed energy and provided bandwidth of both strategies does not differ much where the energy consumption of both strategies is close to the minimum that can be achieved by the protocol with a strategy that always picks the optimal decision (*theoretical optimum*). The *balanced* heuristics on the other hand provides the highest level of service-availability, so a delay of the packages is very unlikely when one decides to employ this heuristics. But, this has the drawback that the energy consumption is the worst out of the three investigated strategies. Of course, its energy consumption is still much better than keeping the 10 GBit-card active the whole time (compare to *10 GBit*). So, *balanced* should be used when the availability of the service is essential. Another thing that can be recognised by the presented plot is the fact that the curve for *theoretical optimum (99%)* is situated right from the curves for the heuristics *aggressive* and *high savings*. This means that there is no possibility for both heuristics to provide the desired objectives for u = 99%, since their consumed energy is not sufficient. In conclusion, it can be seen that the presented results do deliver a theoretical justification for the observations stated in [Häh+13], which were derived by utilising a custom-made simulator. The presented results as well allow to rank the performance of the investigated heuristics in comparison to different theoretical optima.

The computation of the DAG for the zero-reward states is not that expensive for this protocol, and the required iterations are also a bit higher when compared to most of the previously presented protocols. Those facts are beneficial for the parallel quantile computation using the EXPLICIT-engine. In fact this can be observed in Table 6.35, illustrating that the parallel execution is a very good choice for this protocol. Unfortunately, the symbolic engines have their problems with this protocol, and therefore the time limit of 24 hours was exceeded throughout the calculations. Even the model-building process was not completed within the time frame. So, the quantile calculations could not even start for the symbolic engines, and that is the reason why there is no statistical data for the symbolic computations available.

Note on different reward windows As explained in Section 5.1.2 the framework integrated into the EXPLICIT-engine provides three different mechanisms for storing the computed values from previous iterations. Since the reward function modelling the consumed energy of the eBond-protocol is equipped with different values for different states, there exists potential for taking advantage of the several implemented mechanisms.

Figure 6.20 shows a comparison between the different mechanisms. In the figure

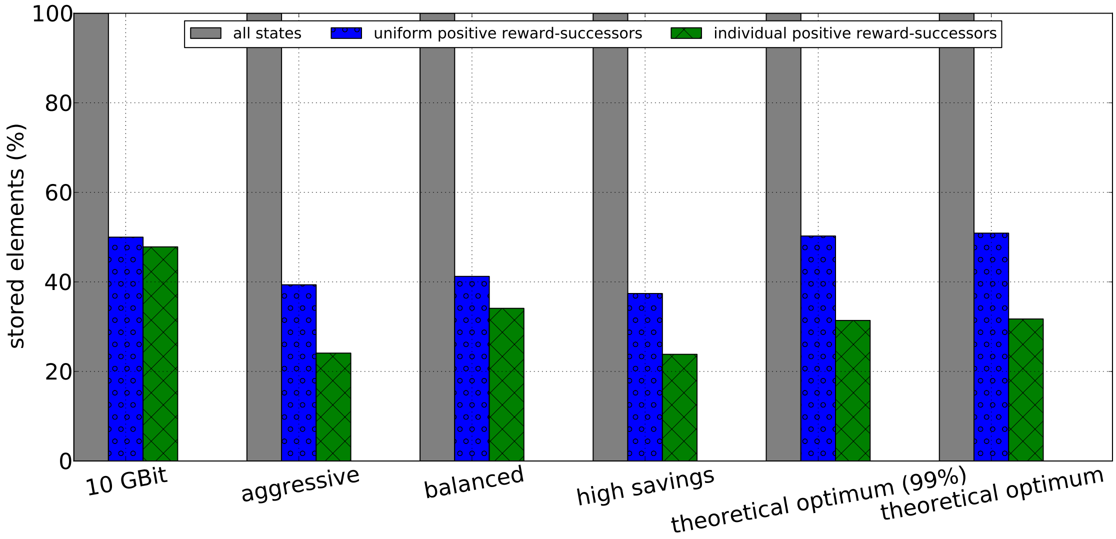

Figure 6.20: Storage for different reward windows (eBond)

the necessary sizes for the different reward window implementations are depicted by means of their stored elements. Instead of presenting the absolute numbers for the stored elements, Figure 6.20 depicts a percentual storage for those stored elements. This means that the number of stored elements is related to the number of elements when using the storage method *all states* (see Section 5.1.2). This supports the visual comparison of the storage needs for the depicted reward window approaches as the absolute numbers of the stored elements differ from each other by orders of magnitude. The storage mechanism *all states* stores values for every state of the model and each state uses a window corresponding to the largest reward that occurs in the whole model. *Uniform positive-reward successors* uses the same window for all the states that are stored, but it does not store information for each state of the model. Instead, only values for those states reachable when investing some positive reward will be saved. *Individual positive-reward successors* stores also the values for each positive-reward successor, but in comparison to *uniform positive-reward successors* there is no fixed window size for all the stored states. Every involved state has its own window which is justified by the maximal reward that needs to be invested in order to reach the respective state.

The mechanism of storing just the values for the positive-reward successors reduces the needed memory in a nice way. E.g., for the *aggressive*-heuristics the storage for the values can be reduced to 40% when using *uniform positive-reward successors*, and the usage of *individual positive-reward successors* even reduces the memory consumption for the storage to 24%. Therefore, much less memory is required in order to perform the calculations. But, this reduction in memory causes some overhead because there need to be additional computations in order to find the correct entry for the desired states. Therefore, Figure 6.21 shows the timings for the sequential quantile computation, and

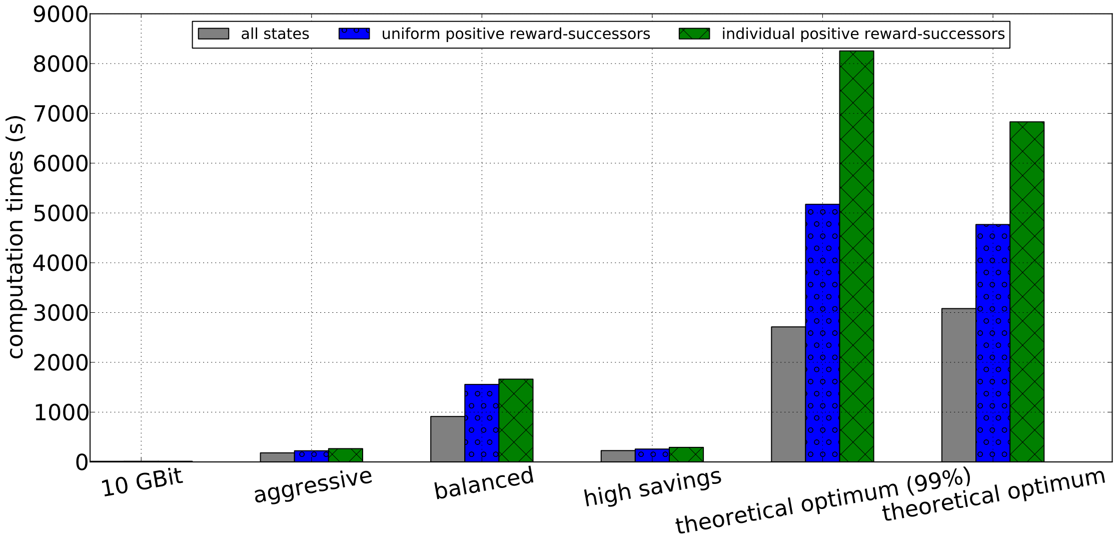

Figure 6.21: Timings for different reward windows (eBond)

it can be seen that the calculations become slower for the methods *uniform positive-reward successors* or *individual positive-reward successors*, where only values for the positive-reward successors are stored. For *aggressive*, *high savings* or *10 GBit* the difference is not really recognisable, since there are only a few states involved. But, for the both optimal cases (*theoretical optimum* and *theoretical optimum (99%)*) the impact is recognisable, since both models contain around 30 million states. Therefore, a lot of remapping is involved in order to ensure a proper mapping for the states in the used data structure, and this causes a computational overhead that can influence the computation times in a noticeable way. So, as can be seen, the reduction in the used memory has to be paid with an increased run time.

6.2.3 HAECubie Demonstrator

Now, we turn to the analysis of a complex service platform providing video downloading and video transcoding as a web service. This platform shows a closed energy-control loop embedded into a real server system, enabling the system to automatically adapt to dynamic environmental changes and to provide energy efficient services.

The HAECubie system consists of 30 Cubieboards[13] (abbreviated as CB) organised in six groups, where each group consists of 5 Cubieboards and a switch. All Cubieboards inside a group can communicate with each other using the corresponding switch of the group. A local storage is attached to each Cubieboard, which is a 32 GB SD-card, and serves to store the videos of the board. All six groups are connected directly to an additional switch bundling the connections of all the groups, and providing a connection to another distinguished Cubieboard, called *master*. This particular board

---

[13] `http://cubieboard.org/`, retrieved 28th March 2018

serves as a controller managing the operational behaviour, and controlling the energy management of the complete system. A schematic overview of the system is depicted in Figure 6.22.

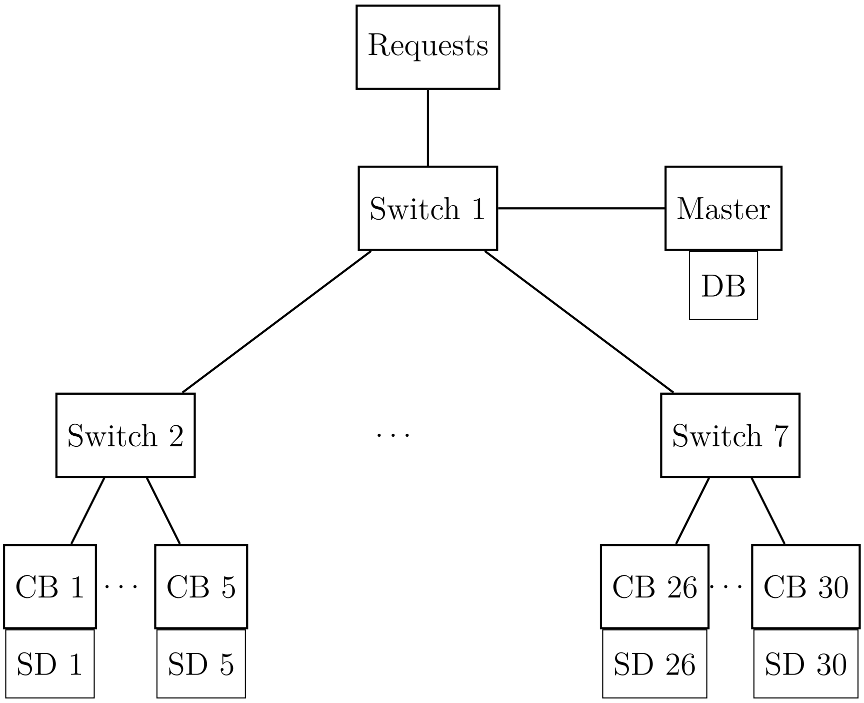

Figure 6.22: Topology of HAECubie demonstrator

The usage of PMC for this protocol has the objective to support the management system at run time by providing the results of an energy-utility trade-off analysis whenever the system needs to deal with complex situations that influence the system's energy efficiency to a large extent. Therefore an analysis of the behaviour of the system for specific situations was done at design time and the results were stored in a database to make them available at run time. This supports the decision making of the running system by providing the precomputed results of a specific system behaviour for a specific situation, and therefore allowing the management of the system to base its configuration adaptations in complex situations on the outcome of a formal analysis. In such situations it is difficult to make an appropriate system adaptation that is efficient in terms of consumed energy.

Since the structure and the operational behaviour of this protocol is rather complex, the PMC-based analysis of the HAECubie demonstrator can not be carried out on a model that describes all the details of the protocol. Instead, the analysis needs to be realised on an abstract model where all the important operational behaviour is taken into consideration and at the same time all the details that are not crucial are left out for the analysis. Otherwise an analysis is not possible due to the well-known state-space explosion problem (see [BK08, Section 2.3]). In order to avoid this problem, the behaviour of multiple boards will be condensed into clusters and those clusters are then used as operational units. The different clusters mainly differ in terms of the stored videos that can be provided. So, the modelled behaviour is therefore coarse-grained, but in the end it is sufficient to get useful insights into the behaviour of the real-world protocol. The abstracted model for the analysis is depicted in Figure 6.23. The abstract operational behaviour of one such cluster is then represented by the operational behaviour depicted in Figure 6.25.

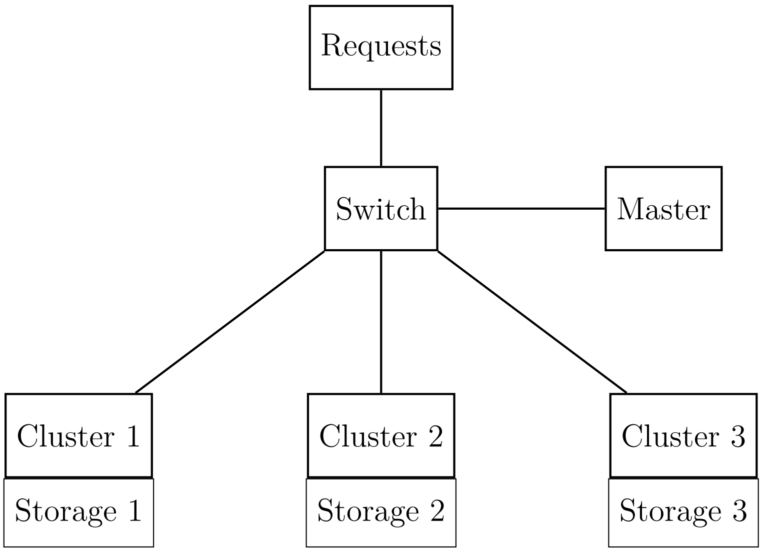

Figure 6.23: Topology of abstracted HAECubie demonstrator

6.2.3.1 Operational behaviour of the protocol

We use an **MDP** for modelling the abstract HAECubie demonstrator. Its behaviour is defined by the parallel composition of the several components depicted in Figure 6.23, that are detailed in the following using control-flow graphs.

Switch The main task of a switch is to enable the communication of every component that it is connected to. Since the scenario tries to be energy efficient, the model reflects the fact of turning a switch completely off. But of course the boards attached to it are then separated from other parts of the system since there is no other connection to those parts. The operational behaviour of a switch is depicted in Figure 6.24. A switch uses 1 Watt if it is turned on but idling, i.e., no communication to the attached boards is active, and it needs 3 Watts when it is under load, i.e., at least one attached board communicates over the channels provided by the switch. Of course the switch consumes 0 Watts if it is turned off completely.

There exist communication signals that can be triggered by the master in order to influence a switch. Those commands are *onSwitch* and *offSwitch*. The first command signalises a switch that it should turn on, and therefore all functions provided by the switch will be available after the necessary initialisation was done, i.e., some time has

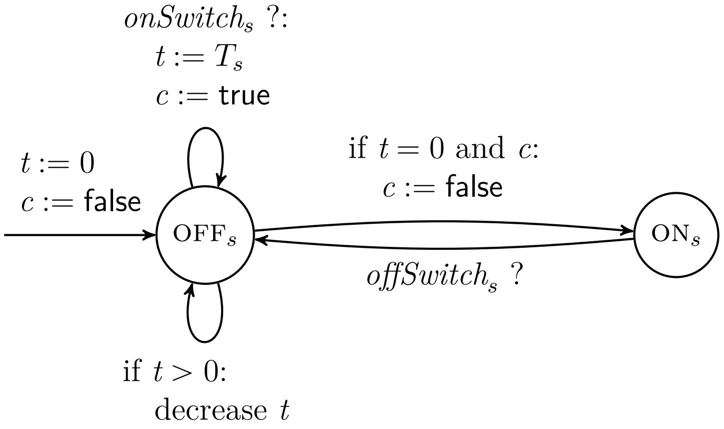

Figure 6.24: Control flow of a switch (HAECubie)

passed specified by variable t. After the command *onSwitch* has arrived, the timer is set to a specific constant T_s, and this amount of time needs to pass before the switch is able to operate. The second command will be sent to the respective switch whenever the master decides to turn the switch off in order to reduce the energy consumption of the system. Whenever a switch receives such a command from the master it reacts by changing its location appropriately and instantly.

Cluster A cluster is the most complex component of the protocol, since it is responsible for processing the work packages. It stores a number of videos in different formats which are then available for the streaming-service of the platform. There is the possibility of installing new videos or to delete already available videos if needed, and therefore the storage changes dynamically during run time. The cluster does also provide the possibility of transcoding a specific video from one format into another format if demanded. Figure 6.25 depicts the control-flow graph of a cluster. The operational behaviour is hereby inspired by the behaviour of a Cubieboard. The commands *onCB*, *offCB*, *trans* and *copy* are triggered by the master and have the aim to power up a cluster in order to prepare for upcoming tasks (*onCB*), to turn a cluster completely off to save energy (*offCB*), to transcode a specific video object into a specific target format (*trans*), and to copy a video object from one cluster to another cluster (*copy*). Each of the specific commands results in a particular reaction of the cluster. The variables x_v and x_f used in the control-flow graph are hereby needed for storing information on the video object the component is currently working with. The variable x_r is used in order to have knowledge about the request that will be served by the component. The cluster works by starting in location OFF, and after the signal *onCB* was received from the master, it transitions to IDLE (after some initialisation was done). There the component is able to perform multiple operations. It can transition to TRANS, signalising that an existing video will be transcoded from one format into another format. This transition sets a timer corresponding to the duration of the transcoding

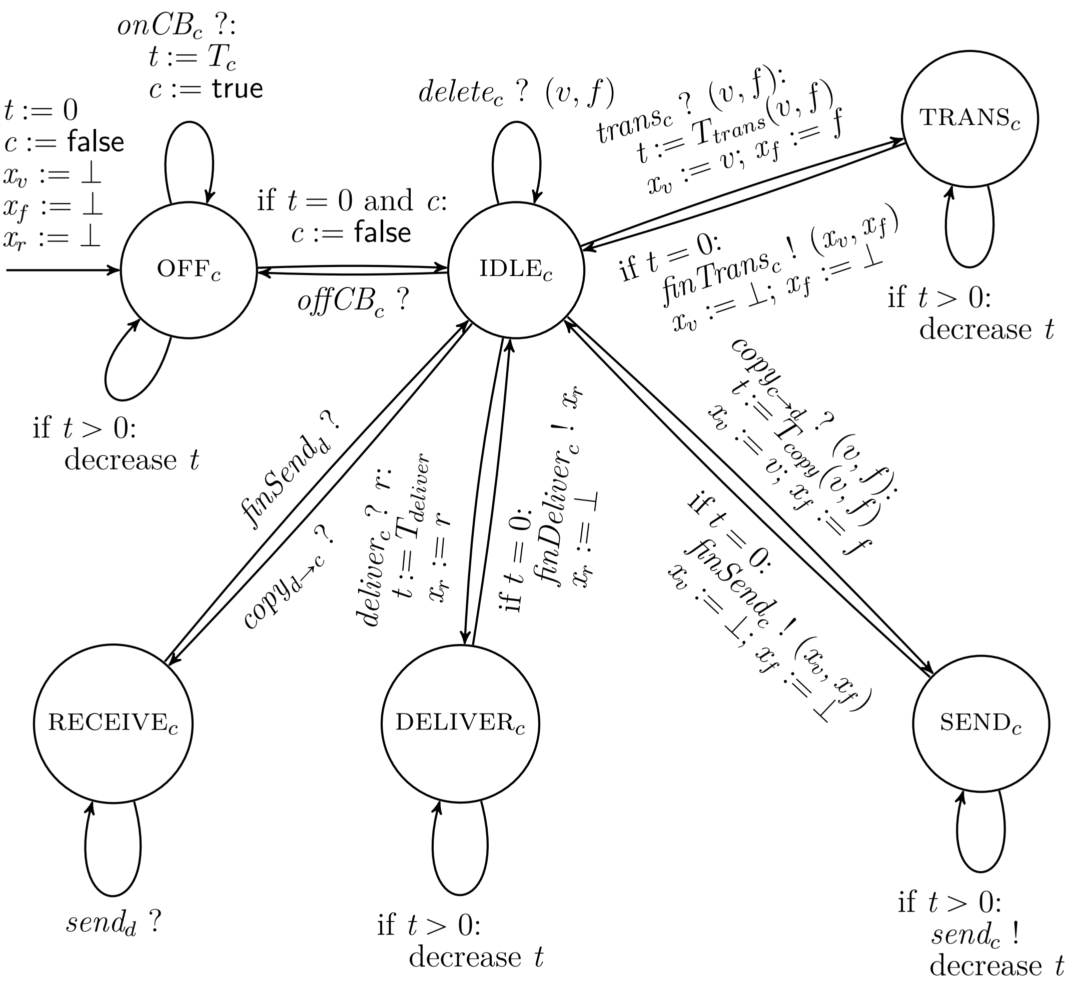

Figure 6.25: Control flow of a cluster (HAECubie)

process. In TRANS the timer will be decreased until it reaches zero, and the component will transition to IDLE again in order to accept new instructions. When a video in a specific format should be duplicated to another cluster, the transition $copy_{c \to d}$ lets the component transition into SEND. The *copy*-command hereby needs to be synchronised with another cluster-component. The synchronisation partner at the same time transitions to its location RECEIVE. In SEND the cluster continuously sends the video data to its partner (modelled by the *send*-command) and by the way decreases a timer specifying the duration of the sending-process. The partner hereby continuously receives the command *send* and can therefore not leave its location RECEIVE. As soon as this duplication-process has finished a *finSend*-signal will be sent, and both synchronisation partners will simultaneously transition back to IDLE. When the cluster receives a *deliver*-signal from the master it will transition to its corresponding location DELIVER. Here, the demanded video-file will be delivered to the user of the system in order to generate some utility for the protocol. This delivering-process works closely related to the environment (Figure 6.26).

Since the behaviour of the cluster is inspired by a Cubieboard its energy consumption was adapted to that of a Cubieboard as well. Therefore, it consumes 1 Watt if it is idling and it needs 3 Watts when it is under load, e.g., a video is streamed, copied or transcoded. This has the effect that the energy consumption of a switch and a cluster is quite similar.

Requests Here, the interplay of the system with the surrounding environment is specified. It formulates and sends the requests that arrive to the system. Each request consists of a video identifier in combination with several parameters, i.e., the demanded format and a deadline defining the maximal time span in which the request should be successfully answered by the system. Utility for the whole HAECubie-system is granted in terms of successfully delivered videos where the given deadline was kept. Figure 6.26 shows a control flow graph formalising the environment by specifying a timer t that regularly triggers a system-request and a queue with capacity q that stores the incoming requests to the system. The timer is continuously decreased, and when its value reaches zero a new request arrives to the system specifying the demanded characteristics, and the timer is set to a new value in order to prepare for the next arriving request. Hereby the new value for the timer is chosen randomly from the specific distribution $\mathbb{T}$, whereas the deadline for the new request is randomly chosen from the distribution $\mathbb{DL}$, and the data specifying the demanded video are taken randomly from V (set of available videos) and F (set of supported video-formats). Whenever the queue is filled to capacity and therefore there is no possibility of accepting a new request, the incoming request will be dropped. This will result in a missed deadline for this particular request, and is therefore not wanted by the user of the system and should be avoided. If on the other hand the capacity of the queue has not yet been reached, the incoming request will be stored in order to further process

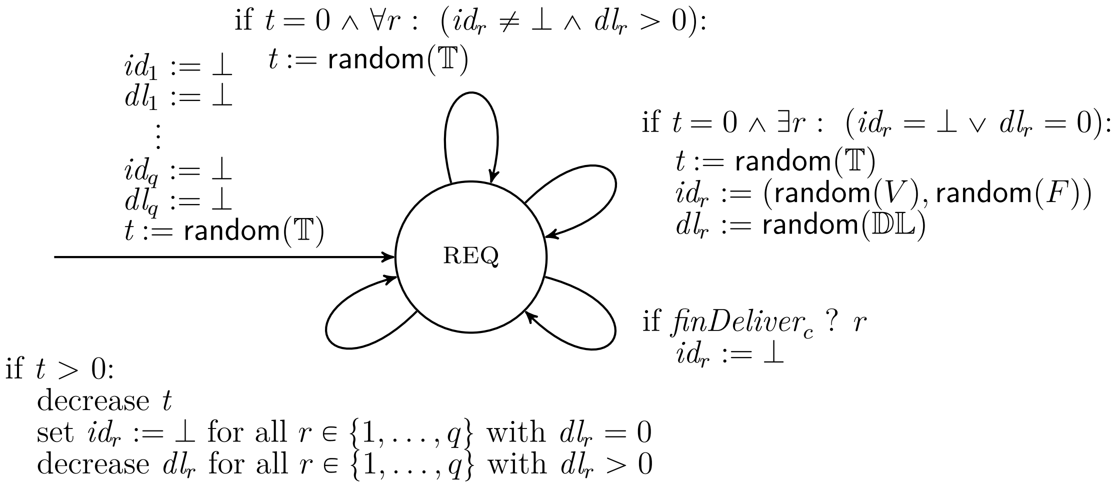
if $t = 0 \wedge \forall r : \ (id_r \neq \bot \wedge dl_r > 0)$:
$t := \mathsf{random}(\mathbb{T})$
$id_1 := \bot$
$dl_1 := \bot$
$\vdots$
$id_q := \bot$
$dl_q := \bot$
$t := \mathsf{random}(\mathbb{T})$
REQ
if $t = 0 \wedge \exists r : \ (id_r = \bot \vee dl_r = 0)$:
$t := \mathsf{random}(\mathbb{T})$
$id_r := (\mathsf{random}(V), \mathsf{random}(F))$
$dl_r := \mathsf{random}(\mathbb{DL})$
if $finDeliver_c \ ? \ r$
$id_r := \bot$
if $t > 0$:
decrease t
set $id_r := \bot$ for all $r \in \{1, \ldots, q\}$ with $dl_r = 0$
decrease dl_r for all $r \in \{1, \ldots, q\}$ with $dl_r > 0$

Figure 6.26: Control flow of environment (HAECubie)

it. When the *finDeliver*-signal was sent by cluster c the corresponding request r gets deleted from the queue and a slot for a new request is freed. A slot will be freed as well when the deadline timer of a specific request has reached zero, and the request has not been processed. This case corresponds to a negative outcome of the protocol, and should be avoided.

Video storage The service platform needs to know what videos are available on which cluster and which format is provided for the respective videos in order to successfully process the requests. If a video file is not available in the requested format the video needs to be transcoded in order to get the format right. For this mechanism the storage as depicted in Figure 6.27 is the foundation used to encode the mapping function $\mathsf{pos} : V \times F \longrightarrow 2^{CB}$, where V is the set of videos, F the set of formats, and CB the set of available clusters. Therefore, pos stores for each video $v \in V$ in format $f \in F$ the respective clusters where (v, f) is currently available, and therefore one of the respective clusters is a candidate for the streaming of v in format f.

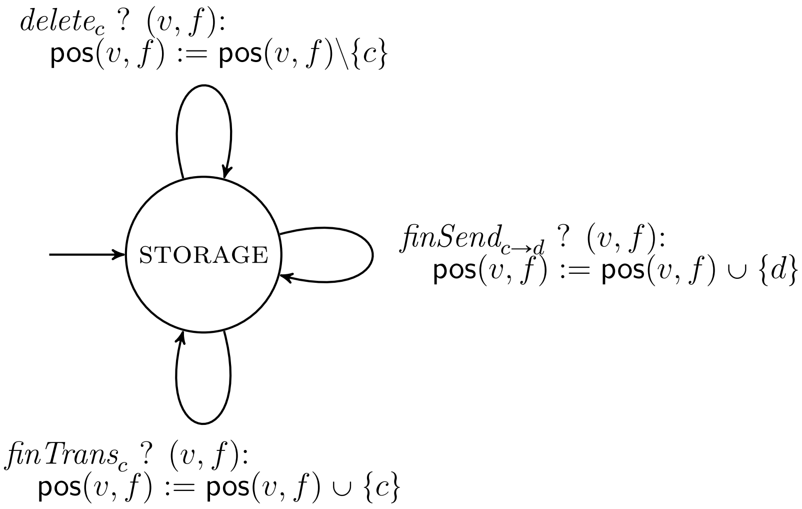
$delete_c$? (v, f): $\mathsf{pos}(v, f) := \mathsf{pos}(v, f)\backslash\{c\}$
STORAGE
$finSend_{c\rightarrow d}$? (v, f): $\mathsf{pos}(v, f) := \mathsf{pos}(v, f) \cup \{d\}$
$finTrans_c$? (v, f): $\mathsf{pos}(v, f) := \mathsf{pos}(v, f) \cup \{c\}$

Figure 6.27: Control flow of video-storage (HAECubie)

Master The main task of the master is the management of all involved components. It has the ability to directly influence the behaviour of the complete system. Therefore it uses commands to trigger a specific behaviour of all the other components. All the commands have the aim to ensure the usability of the system by simultaneously trying to be as energy efficient as possible. Therefore the operational behaviour of the master is split into two different fields of responsibility. The first one is controlling the dynamic handling of the video files among the different clusters (see Figure 6.28). The operations of the master allow to trigger a deletion of a video file or to start a transcoding process on a specific cluster. So, whenever a video is available on a cluster, the master can trigger the command *trans* to start a transcoding process on the specific

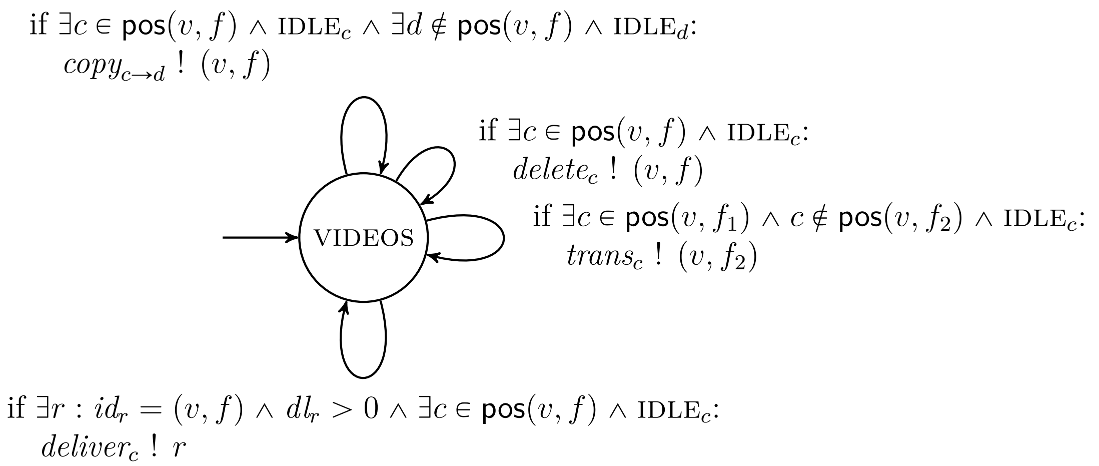
if $\exists c \in \mathsf{pos}(v,f) \wedge \textsc{idle}_c \wedge \exists d \notin \mathsf{pos}(v,f) \wedge \textsc{idle}_d$:
$copy_{c \to d}$! (v,f)
if $\exists c \in \mathsf{pos}(v,f) \wedge \textsc{idle}_c$:
$delete_c$! (v,f)
if $\exists c \in \mathsf{pos}(v,f_1) \wedge c \notin \mathsf{pos}(v,f_2) \wedge \textsc{idle}_c$:
$trans_c$! (v,f_2)
VIDEOS
if $\exists r : id_r = (v,f) \wedge dl_r > 0 \wedge \exists c \in \mathsf{pos}(v,f) \wedge \textsc{idle}_c$:
$deliver_c$! r

Figure 6.28: Control flow of the master (controlling video storage, HAECubie)

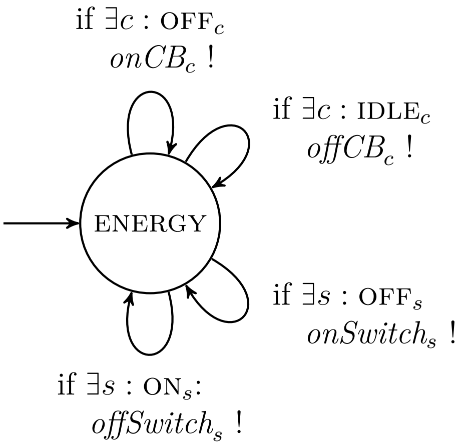
if $\exists c : \textsc{off}_c$
$onCB_c$!
if $\exists c : \textsc{idle}_c$
$offCB_c$!
ENERGY
if $\exists s : \textsc{off}_s$
$onSwitch_s$!
if $\exists s : \textsc{on}_s$:
$offSwitch_s$!

Figure 6.29: Control flow of the master (controlling energy consumption, HAECubie)

cluster component, in order to have the video available in another format. Whenever a video file is not available on some cluster it can be copied from another cluster holding the video by using the *copy*-command. Since this command requires two different clusters in order to work properly the corresponding clusters receive the command simultaneously, and afterwards start their working package. The master also links the request-queue with some cluster that is able to serve a request from this queue. Therefore, it checks whether there exists an idling cluster which stores the demanded video object. If this is the case a *deliver*-signal will be sent and the referenced cluster starts streaming the video and as soon as it finishes the corresponding request can be dropped from the queue, and some utility for the system has been generated.

The second component depicted in Figure 6.29 is responsible for turning the different involved switches and clusters on or off. This is used in order to instrument the system to save energy by keeping nonessential resources off.

So, all different adaptation strategies make use of the master-component and its provided commands. Arbitrary management strategies can be therefore implemented by interacting with the presented components of the master.

6.2.3.2 Formal analysis

Now, we want to analyse the trade-off between the consumed energy and the gained utility of the abstracted system (see Figure 6.23) in specific situations, and therefore it is necessary to describe the specific scenario where the demonstrator should operate as energy efficient as possible. The interesting situations of the scenario are characterised by a workload contradicting a usual system workload, and therefore it is hard to predict the best way to react (or anticipate) in order to satisfy the user's demands and on the other hand keeping the system as energy efficient as possible. Such a situation may arise whenever a paying user, a so-called *premium user*, logs to the system. Since this user is paying for the service, it is not beneficial to not satisfy the demands of the user. Otherwise, it is very likely that the user will not pay for the service in the future, and the video platform will lose its financial basis. Therefore, it is highly recommended to instruct the system in a way that the needs of the premium user can be satisfied and at the same time the energy efficiency of the whole system can be kept at an acceptable degree. So, for the analysis we assume that a premium user logs to the system and based on the experiences with this user in the past it is quite likely that this specific user will demand a video which is unpopular for the majority of the users. This fact can be represented using suitable probability distributions inside the control flow graph for the environment (see Figure 6.26). Therefore, the distribution of the requested video among all Cubieboards is very restricted, and only a small number of Cubieboards do have the requested video available on their local storage. Therefore, it is inevitable that one of the Cubieboards storing the video is involved into the process of streaming the video to the demanding user.

The analysis concentrates on determining the minimal energy that needs to be spent in order to stream a specific amount of video files while simultaneously keeping user-specified deadlines. Therefore, we analyse the (constrained) reachability quantile query

$$\mathrm{e}^{\mathrm{u}}_{\min} = \min\big\{\mathrm{e} : \mathsf{Pr}^{\max}_{s}\big((\neg\mathrm{SlaViolation})\,\mathcal{U}^{\mathrm{energy}\leqslant \mathrm{e}}\;(\mathit{streamedVideos} \geqslant \mathrm{u})\big) > p\big\}$$

with u being the desired amount of streamed videos, and p the probability threshold. SlaViolation hereby characterises states within the model where the protocol cannot guarantee a specific quality for its provided services. The protocol considers several situations where the availability of the services is insufficient. The first situation reflects the fact that a deadline of a premium user's request has not been kept before the streaming of the requested video has completed. The second situation that violates the service level of the protocol for premium users is by simply dropping the request out of the queue due to too many pending requests, since the queue has reached its capacity. So, the usage of the complement of SlaViolation characterises all states where the sketched situations do not occur. The result of the formal analysis using the previously mentioned reachability quantile is depicted in Figure 6.30 for a desired utility bound of u = 3. In order to satisfy the demands of premium users the system needs to operate preemptive. As can be seen in Figure 6.30 it is very likely that the system can not satisfy the demands of the paying user when there are no additional boards involved. So, it might be a good option to turn on additional boards storing barely demanded videos at the moment when the user logs to the system, in order to have them ready whenever an unusual request arrives. Another possibility for improving the performance of the system in this scenario is to increase the cooldown timer of the boards. Therefore, the boards are more responsive in this situation and it is more likely that the needs of the premium user can be satisfied in this case. A combination of both, increased cooldown timers and additional boards, stand a good chance of satisfying the requirements by simultaneously keeping the system as energy efficient as possible. The performance of the case where the cooldown timer was set to 6 and the additional boards were activated early enough, comes already quite close to the theoretical optimum. Therefore, the handling of the situation in this fashion would be most helpful.

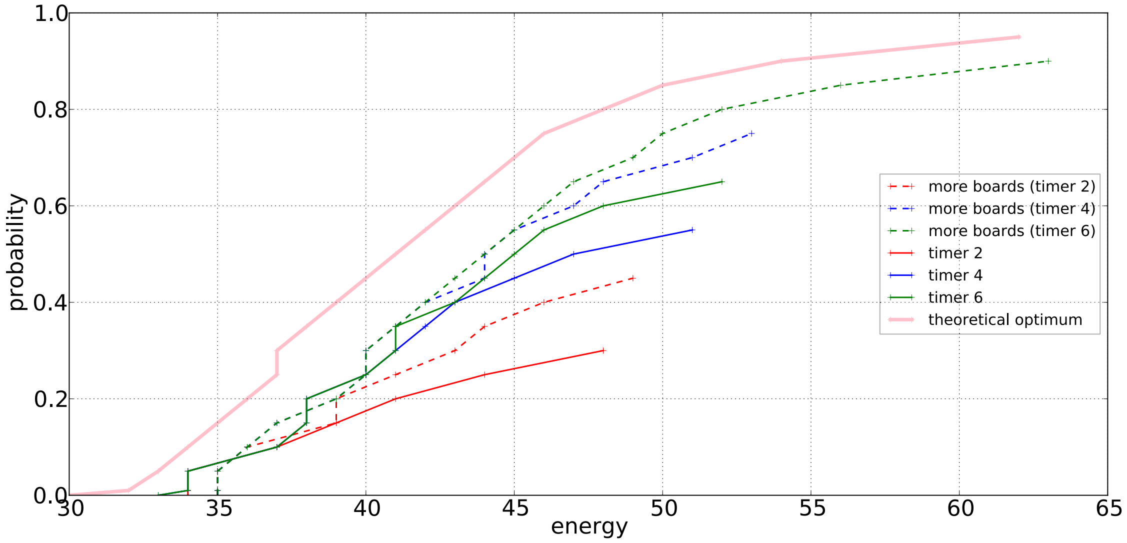

Figure 6.30: Results for HAECubie (minimal energy consumption until 3 videos were streamed successfully)

To compare the results depicted in Figure 6.30 to a normal daily workflow of the system, Figure 6.31 shows the results for the expectation quantile

$$E_s^{\exists} = \mathrm{qu}_s\big(\exists \mathrm{ExpU}_{>\mathrm{u}}(\mathrm{energy} \leqslant ?)\big)$$

for a variety of utility thresholds u. For the considered situation it is assumed that no user has logged to the system whose preferences contradict the normal daily workload. So, keeping any additional board active over a longer period of time is not feasible in terms of energy efficiency since the operation of those boards does not really support

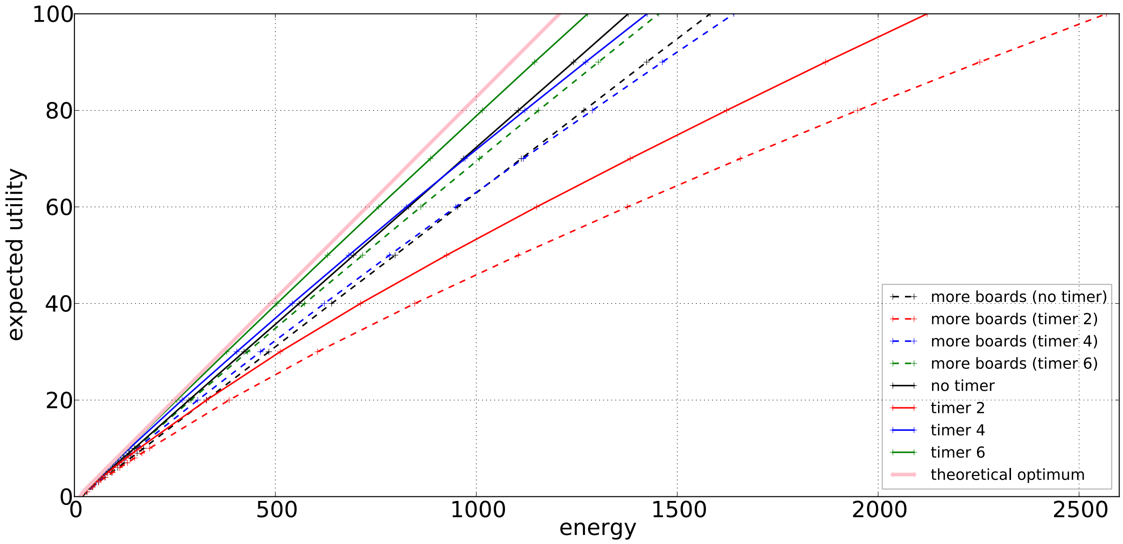

Figure 6.31: Results for HAECubie (minimal expected energy consumption for specific utility)

the system in order to serve usual requests (see Figure 6.31). Therefore, there is no noticeable utility-gain by operating the additional boards, but as a drawback the energy consumption is much higher. As a consequence, it is recommended to turn the additional boards off, and try to handle the requests without utilising those additional boards. The average overall performance of the system is not affected, but as an outcome the energy consumption of the system will be reduced significantly. A further observation that follows from Figure 6.31 is the fact that an increased cooldown timer for the Cubieboards has a positive effect on the energy efficiency of the system. But it needs to be taken into account that the timer cannot be increased arbitrarily since the energy efficiency for a system without any cooldown timer (specified as *no timer*) is worse than for the systems where the timer has been set to 6. The performance for the case where there are no additional boards involved and the timer was set to 6 entails the most energy efficient possibility out of the different investigated alternatives, and its performance is also very close to the optimum that is theoretically achievable.

The corresponding calculation statistics for the computation of the expectation quantiles can be found in Table 6.36. Due to the fact that the different symbolic engines do currently not support the computation of expectation quantiles, only statistics for the Explicit-engine are considered here. In Table 6.37 (Explicit-engine) and in Table 6.38 (symbolic engines) on page 154 the statistics for the computation of the reachability quantile $\mathrm{e}^{\mathrm{u}}_{\mathrm{min}}$ are depicted. Due to the number of reachable states of the corresponding models the Explicit-engine of Prism could not handle the building process for the cases when the timers are set to 4 or 6. It was therefore also not possible to utilise the quantile calculations for Prisms Explicit-engine in those cases. And as can be seen as well, the parallelisation of the computations for the Explicit-engine does not improve the calculation times that much. The reason is that the number of required quantile iterations is small (63 is the maximum out of all required iterations). So, the generation of the DAG for the zero-reward sub-**MDP**s (see Section 5.1.4) needs a noticeable amount of time for the considered models, e.g., the calculation of the DAG needed almost 40 seconds in order to finish successfully when considering the case *timer 2*. So, the parallel computation started its calculations with a delay of 40 seconds and there were simply not enough iterations in order to compensate this delay throughout the processing. The time required for the computation within the HYBRID-engine is fairly high, whereas SPARSE and MTBDD perform quite well on this protocol.

Table 6.36: Statistics for expectation quantile (minimal expected energy consumption for specific utility, HAECubie)

	Model			EXPLICIT		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
more boards (no timer)	474 865	29.48 s	1 583	15.94 s	198.68 s	71.27 s
more boards (timer 2)	3 377 698	208.49 s	2 569	207.52 s	2 188.03 s	679.28 s
more boards (timer 4)	8 092 061	500.2 s	1 642	945.04 s	4 310.2 s	1 879.66 s
more boards (timer 6)	14 760 739	929.76 s	1 455	3 053.11 s	9 979.0 s	5 329.94 s
no timer	123 463	7.84 s	1 379	4.02 s	57.48 s	23.24 s
timer 2	710 589	51.06 s	2 123	44.48 s	357.51 s	131.49 s
timer 4	1 560 165	112.7 s	1 426	100.81 s	631.78 s	236.09 s
timer 6	2 719 389	190.05 s	1 278	256.55 s	1 138.9 s	484.2 s
theoretical optimum	1 544 263	81.83 s	1 208	66.83 s	562.61 s	186.97 s

As seen in Table 6.36 the time for the precomputation constitutes a noticeable part of the overall computation time for the considered expectation quantile. Since the precomputation-step for existential expectation quantiles requires to build a new model where the zero-utility end components are collapsed into a single state (see Section 4.2), it is necessary to utilise the model-building process of PRISMs EXPLICIT-engine. As already seen in previous statistics this model-building process seems to be a part of the implementation where it might be beneficial to further improve it. Another fact that contributes to the computation times is simply the number of required calculation-iterations that need to be performed. But, this high amount of iterations suits the parallel computations, since this allows to compensate the additional overhead induced by parallel computations (like, e.g., the construction of the DAG).

Table 6.37: Statistics for explicit reachability quantile (minimal energy consumption until 3 videos were streamed successfully, HAECubie)

	Model			Explicit		
					sequential	parallel
Instance	States	t_{build}	Iters	t_{pre}	t_{query}	t_{query}
more boards (timer 2)	26 330 358	1 774.59 s	50	193.26 s	845.95 s	804.87 s
timer 2	35 500 849	2 501.28 s	49	269.44 s	1 237.65 s	1 227.71 s
theoretical optimum	6 177 052	287.33 s	63	49.65 s	149.58 s	129.98 s

Table 6.38: Statistics for symbolic reachability quantile (minimal energy consumption until 3 videos were streamed successfully, HAECubie)

	Model				symbolic computations					
					Hybrid		Sparse		MTBDD	
Instance	States	Mtbdd	t_{build}	Iters	t_{pre}	t_{query}	t_{pre}	t_{query}	t_{pre}	t_{query}
more boards (timer 2)	26 330 358	457 306	27.78 s	50	331.35 s	1 556.41 s	91.36 s	517.38 s	428.2 s	768.4 s
more boards (timer 4)	92 185 311	650 996	51.23 s	54	1 039.29 s	6 086.04 s	263.81 s	1 915.43 s	940.55 s	1 790.93 s
more boards (timer 6)	217 883 034	674 101	68.88 s	64	1 785.84 s	15 952.35 s	564.23 s	5 249.64 s	1 445.16 s	3 251.64 s
timer 2	35 500 849	310 859	24.66 s	49	287.08 s	1 854.02 s	96.42 s	661.2 s	267.74 s	490.18 s
timer 4	117 053 984	383 943	67.59 s	52	762.24 s	6 795.08 s	273.73 s	2 495.89 s	478.64 s	1 000.66 s
timer 6	271 683 232	405 790	93.06 s	53	1 583.06 s	17 892.82 s	627.45 s	5 965.02 s	772.71 s	1 366.86 s
theoretical optimum	6 177 052	62 329	1.11 s	63	24.04 s	282.1 s	11.55 s	103.84 s	19.71 s	73.32 s

7 Conclusion

We start to conclude this thesis by giving a short summary of the contributions that were developed throughout this monograph in order to analyse the trade-off between the consumed energy and the produced utility of an energy-aware system. For this purpose we presented a quantile-based framework that allows to analyse different aspects of a variety of energy-aware systems[1], and all the presented quantile metrics have their specific field of application where they can deliver valuable information about the system under consideration. This directly supports the refinement of the system specification by its maintainers in order to tailor the system in a more energy-efficient way.

Whereas Chapter 1 and Chapter 2 contain introductory material in order to support the understanding of this monograph, the presentation of the intended contributions starts in Chapter 3 with the introduction of the computation of upper- and lower-reward bounded reachability quantiles. Those quantiles allow to minimise the accumulated energy (respectively another important cost factor) or to maximise the accumulated utility (respectively another desired outcome that originates from the successful utilisation of the protocol under consideration). The desired effect is achieved by solving a linear program that minimises or maximises the accumulation of a reward structure by applying an iterative solution method. In order to deliver the same analysis for cases where it needs to be ensured that the protocol under consideration preserves specific objectives as well, we presented the utilisation of quantiles under side conditions on the other hand. This allows to perform the quantile-based analysis by doing a tailored transformation of the model and afterwards relying on the same mechanisms as already presented for the computation of quantiles without any side conditions.

For a minimisation of the specified cost when an arbitrary expected value should be guaranteed we demonstrated the computation of expectation quantiles in Chapter 4. The presented algorithms hereby rely on methods that are related to the already known routines from Chapter 3, and consider the simultaneous accumulation of two separate reward functions. This allows to handle the practical computation of expectation quantiles with similar approaches as already carried out for the computation of reachability quantiles.

Since our aim was also to provide an efficient tool-support for the introduced methods we presented an implementation and its integration into the well-known probabilistic model checker PRISM in Chapter 5. The chapter starts by showing several possibilities

---

[1] Keep in mind that the presented framework is not restricted to the needs of energy-aware systems only. Instead, it is possible to apply the presented methods for the analysis of other costs like, e.g., time.

that greatly improved the performance of the provided computation methods in terms of used memory and needed computation times. Those techniques do rely on the specific structure of the linear program that is typical for the computation of quantiles or reward-bounded reachability properties.

All the previously presented methods were utilised in Chapter 6 to present the successful analysis of a couple of protocols that are of practical interest. It was shown that the different implemented computation methods have their specific strengths and weaknesses depending on the protocol under consideration. And since there exist multiple settings that allow to adapt the implementation to the analysed protocol there is the possibility to adjust the computational performance of the analysis to the needs of the specific protocol. In the end this helps to deliver the desired results in an efficient manner.

7.1 Classification

The presented quantile framework is tailored such that the analysis can be adjusted to the various needs of the protocols under analysis. Chapter 6 shows that it would have been impossible to deliver an efficient support for the computation of quantiles without the utilisation of the different optimisations presented in Section 5.1. Table 6.3 (Self-Stabilising Protocol) and Table 6.8 (Asynchronous Leader-Election Protocol) reveal that an approach that relies on general-purpose methods for solving the linear programs reaches its limitations even for very small models and when there are only a few iterations required. Therefore, the introduction of the back-propagation approach was an essential step in order to provide a tool-support as desired.

Chapter 6 as well shows that the different implementations for the engines available in PRISM have their individual strengths and weaknesses. E.g., it can be said that the parallel execution of the EXPLICIT-engine improves the performance for the majority of the analysed cases. But there also exist situations where there is no benefits when utilising the parallel computation scheme. The analysis of the Asynchronous Leader-Election Protocol (see Section 6.1.2) corresponds to such a scenario, and it reveals that the generation of the DAG for the zero-reward components consumes such a huge amount of time in relation to the complete quantile computation that it would be the best choice to stick to the sequential computation in this case. The primary reason that negates the positive effects of parallel computations in this specific case is the circumstance that the demanded quantile can be computed within only a few required iterations. This fact entails that there are not enough possibilities to compensate for the characteristic overhead that occurs when using parallel computation methods. If on the other hand the number of required iterations is really high, then it is beneficial to invest the additional overhead in order to enable the parallel computations, and therefore benefit from the time saving. A good example is the Randomised Consensus Shared Coin Protocol in Section 6.1.3. There are so many iterations needed that the additional overhead can be easily neglected during the whole computation. It is also the case that parallel computations are a good choice when there are many reachable states for the model under consideration (see all analysed protocols in Section 6.2). The sheer size of the model supports the parallel computations due to the fact that there exists a huge work package which can be parallelised within each of the required iterations.

As long as the model does not contain too many reachable states the Explicit-engine performs really well. Due to the fact that model checking has to deal with the state-explosion problem at some point we are forced to rely on approaches that do not represent the reachable states in an explicit fashion. One such popular approach is the utilisation of symbolic (MT)BDD-based methods. It can be seen that the model-building process of the Explicit-engine reaches its limitations for some protocol instances presented in Section 6.2. In order to compute the desired quantile values we sometimes were dependent on the usage of Prisms symbolic engines. The HAECubie demonstrator presented in Section 6.2.3 constitutes such a case. There it was not possible to do the explicit computations for instances of the protocol that needed more than 35 million states. But, on the other hand the analysis of the eBond protocol in Section 6.2.2 shows a case where the symbolic engines were unable to perform the building process within an acceptable time frame of 24 hours although there were not more than 30 million reachable states for an instance of the model. The Explicit-engine was able to perform the analysis in an acceptable way without any problems for this protocol.

The utilisation of expectation quantiles relaxes the problem of the state explosion to some extent since we do directly relate the accumulation of two reward functions and this entails that we do not need to encode the accumulation of one of the two reward functions into the state space of the model under consideration. This directly results in a much smaller number of reachable states compared to the needed states for the analysis of reachability quantiles, and allows us to calculate expectation quantiles for cases where it was not possible to achieve the results in the case of reachability quantiles. See the reachable states for the analysis using expectation quantiles for the Energy-Aware Job-Scheduling Protocol (Table 6.30) or for the HAECubie demonstrator (Table 6.36).

In summary, it can be said that there is no implementation that embodies the best alternative over all the others and delivers the best performance for each and every given situation. Instead, each implementation has its advantages over other implementations depending on the model under consideration, and therefore each implementation has its area of application where it is best suited. The conglomeration of all the considered implementations as a whole forms an efficient framework that delivers support for a wide range of protocols and scenarios for doing a multi-objective analysis focussing on the trade-off between the cost and the provided utility for a given protocol.

7.2 Future prospects

Quantiles could be already considered previously in the literature in an indirect manner throughout various case studies performed by numerous researchers using different probabilistic model checkers. The diagrams illustrating the results of the performed experimental evaluation hereby help to infer information about various quantiles. Therefore, there seems to be an interest in the results that can be obtained by the quantile framework, and this entails some demand in a possibility for computing those results in a direct manner. It might be therefore beneficial to provide the presented framework to the formal methods community by integrating it into a future release of the probabilistic model checker PRISM. By the way, this does also extend PRISM in order to compute reward-bounded reachability probabilities.

Of course, the framework presented here does not cover each and every aspect relevant for a multi-objective energy-aware analysis. There exist several interesting aspects that were not considered in this monograph, but might be rewarding for further investigations. One such interesting open task is the combination of quantiles with the computation of conditional reachabilities and expectations (see [Bai+14d]). The combination of conditionals and quantiles over **DTMC**s can be done straightforward by doing the required conditional transformations of the model and afterwards computing the desired quantile over the model that results from this transformation. A prototypical implementation has already been carried out (in collaboration with Steffen Märcker) for the computation of conditional quantiles over **DTMC**s, but in order to support nondeterministic models it is not immediately clear how this could be achieved in an efficient way. This entails that there need to be done further investigations regarding this task beforehand.

As stated in this monograph there is no tool-support for some of the presented methods using PRISMs symbolic engines. Therefore, it is of interest to deliver support for the computation of expectation quantiles for symbolic computation methods, or for the computation of quantiles under side conditions. This would really help to provide a versatile framework that is applicable to a broad variety of protocols, since we have seen in Chapter 6 that there needs to be a tool-support for each of the engines supported by PRISM in order to provide the best performance.

Bibliography

[AH90] James Aspnes and Maurice Herlihy. 'Fast Randomized Consensus Using Shared Memory'. In: *Journal of Algorithms* 11.3 (1990), pp. 441–461.

[AHK03] Suzana Andova, Holger Hermanns and Joost-Pieter Katoen. 'Discrete-Time Rewards Model-Checked'. In: *Formal Modeling and Analysis of Timed Systems (FORMATS)*. Vol. 2791. Lecture Notes in Computer Science. Springer, 2003, pp. 88–104. ISBN: 978-3-540-40903-8.

[Alf98] Luca de Alfaro. 'Formal Verification of Probabilistic Systems'. PhD thesis. Stanford, CA, USA, 1998. ISBN: 0-591-90826-3.

[Alf99] Luca de Alfaro. 'Computing Minimum and Maximum Reachability Times in Probabilistic Systems'. In: *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR)*. Vol. 1664. Lecture Notes in Computer Science. Springer, 1999, pp. 66–81.

[Arb04] Farhad Arbab. 'Reo: A Channel-based Coordination Model for Component Composition'. In: *Mathematical Structures in Computer Science* 14.3 (2004), pp. 329–366. ISSN: 0960-1295.

[AW90] Eric W. Allender and Klaus W. Wagner. 'Counting Hierarchies: Polynomial Time and Constant Depth Circuits'. In: *Bulletin of the EATCS* 40 (1990), pp. 182–194.

[BA95] Andrea Bianco and Luca de Alfaro. 'Model Checking of Probabilistic and Nondeterministic Systems'. In: *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 1026. Lecture Notes in Computer Science. Springer, 1995, pp. 499–513.

[Bai+00] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns and Joost-Pieter Katoen. 'On the Logical Characterisation of Performability Properties'. In: *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 1853. Lecture Notes in Computer Science. Springer, 2000, pp. 780–792. ISBN: 978-3-540-45022-1.

[Bai+03] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns and Joost-Pieter Katoen. 'Model-Checking Algorithms for Continuous-Time Markov Chains'. In: *IEEE Transactions on Software Engineering (TSE)* 29.6 (2003), pp. 524–541.

[Bai+09] Christel Baier, Tobias Blechmann, Joachim Klein and Sascha Klüppelholz. ‘A uniform framework for modeling and verifying components and connectors’. In: *Proceedings of the 11th International Conference on Coordination Models and Languages (COORD)*. Vol. 5521. Lecture Notes in Computer Science. Springer, 2009, pp. 247–267.

[Bai+12a] Christel Baier, Marcus Daum, Benjamin Engel, Hermann Härtig, Joachim Klein, Sascha Klüppelholz, Steffen Märcker, Hendrik Tews and Marcus Völp. ‘Chiefly Symmetric: Results on the Scalability of Probabilistic Model Checking for Operating-System Code’. In: *Proceedings of the 7th Conference on Systems Software Verification (SSV)*. Vol. 102. Electronic Proceedings in Theoretical Computer Science. 2012, pp. 156–166.

[Bai+12b] Christel Baier, Marcus Daum, Benjamin Engel, Hermann Härtig, Joachim Klein, Sascha Klüppelholz, Steffen Märcker, Hendrik Tews and Marcus Völp. ‘Waiting for Locks: How Long Does It Usually Take?’ In: *Proceedings of the 17th International Workshop on Formal Methods for Industrial-Critical Systems (FMICS)*. Vol. 7437. Lecture Notes in Computer Science. Springer, 2012, pp. 47–62.

[Bai+14a] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein and Sascha Klüppelholz. ‘Energy-Utility Quantiles’. In: *Proceedings of the 6th NASA Formal Methods Symposium (NFM)*. Vol. 8430. Lecture Notes in Computer Science. Springer, 2014, pp. 285–299.

[Bai+14b] Christel Baier, Clemens Dubslaff, Joachim Klein, Sascha Klüppelholz and Sascha Wunderlich. ‘Probabilistic Model Checking for Energy-Utility Analysis’. In: *Horizons of the Mind. A Tribute to Prakash Panangaden*. Vol. 8464. Lecture Notes in Computer Science. Springer, 2014, pp. 96–123.

[Bai+14c] Christel Baier, Clemens Dubslaff, Sascha Klüppelholz, Marcus Daum, Joachim Klein, Steffen Märcker and Sascha Wunderlich. ‘Probabilistic Model Checking and Non-standard Multi-objective Reasoning’. In: *Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE)*. Vol. 8411. Lecture Notes in Computer Science. Springer, 2014, pp. 1–16.

[Bai+14d] Christel Baier, Joachim Klein, Sascha Klüppelholz and Steffen Märcker. ‘Computing Conditional Probabilities in Markovian Models Efficiently’. In: *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 515–530.

[Bai+15] Christel Baier, Marcus Daum, Benjamin Engel, Hermann Härtig, Joachim Klein, Sascha Klüppelholz, Steffen Märcker, Hendrik Tews and Marcus Völp. ‘Locks: Picking key methods for a scalable quantitative analysis’. In: *Journal of Computer and System Sciences (JCSS)* 81.1 (2015), pp. 258–287.

[Bai+17a] Christel Baier, Joachim Klein, Sascha Klüppelholz and Sascha Wunderlich. 'Maximizing the Conditional Expected Reward for Reaching the Goal'. In: *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part II.* Vol. 10206. Lecture Notes in Computer Science. Springer, 2017, pp. 269–285.

[Bai+17b] Christel Baier, Joachim Klein, Linda Leuschner, David Parker and Sascha Wunderlich. 'Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes'. In: *Proceedings of the 29th International Conference on Computer Aided Verification (CAV), Part I.* Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 160–180.

[Bai+18] Christel Baier, Nathalie Bertrand, Clemens Dubslaff, Daniel Gburek and Ocan Sankur. 'Stochastic Shortest Paths and Weight-Bounded Properties in Markov Decision Processes'. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).* ACM, 2018, pp. 86–94.

[Bar+06] Jiří Barnat, Luboš Brim, Ivana Černá, Pavel Moravec, Petr Ročkai and Pavel Šimeček. 'DiVinE – A Tool for Distributed Verification'. In: *Proceedings of the 18th International Conference on Computer Aided Verification (CAV).* Vol. 4144. Lecture Notes in Computer Science. Springer, 2006, pp. 278–281.

[Bar+08] Jiří Barnat, Luboš Brim, Ivana Černá, Milan Češka and Jana Tumová. 'ProbDiVinE-MC: Multi-core LTL Model Checker for Probabilistic Systems'. In: *Proceedings of the 5th International Conference on the Quantitative Evaluation of Systems (QEST).* IEEE Computer Society, 2008, pp. 77–78. ISBN: 978-0-7695-3360-5.

[Bas+15] Nicolas Basset, Marta Z. Kwiatkowska, Ufuk Topcu and Clemens Wiltsche. 'Strategy Synthesis for Stochastic Games with Multiple Long-Run Objectives'. In: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).* Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 256–271.

[BDK14] Christel Baier, Clemens Dubslaff and Sascha Klüppelholz. 'Trade-off Analysis Meets Probabilistic Model Checking'. In: *Proceedings of the 23rd Conference on Computer Science Logic and the 29th Symposium on Logic In Computer Science (CSL-LICS).* ACM, 2014, 1:1–1:10.

[Bel57] Richard E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[Bie+99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke and Yunshan Zhu. 'Symbolic Model Checking without BDDs'. In: *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction* *and Analysis of Systems (TACAS).* Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 193–207.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008.

[BK98] Christel Baier and Marta Z. Kwiatkowska. 'Model Checking for a Probabilistic Branching Time Logic with Fairness'. In: *Distributed Computing* 11.3 (1998), pp. 125–155.

[BKK] Tobias Blechmann, Joachim Klein and Sascha Klüppelholz. *Vereofy V1.1 - User Manual.* [retrieved: 28th March 2018]. URL: `http://www.vereofy.de/download/vereofy_manual.pdf`.

[Brá+10] Tomáš Brázdil, Václav Brožek, Kousha Etessami, Antonín Kučera and Dominik Wojtczak. 'One-counter Markov Decision Processes'. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).* Society for Industrial and Applied Mathematics, 2010, pp. 863–874. ISBN: 978-0-898716-98-6.

[Bur+90] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill and L. J. Hwang. 'Symbolic Model Checking: 10^20 States and Beyond'. In: *Proceedings of the 5th Annual Symposium on Logic in Computer Science (LICS).* 1990, pp. 428–439.

[CE82] Edmund M. Clarke and E. Allen Emerson. 'Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic'. In: *Logics of Programs.* Vol. 131. Lecture Notes in Computer Science. Springer, 1982, pp. 52–71. ISBN: 978-3-540-11212-9.

[Cie+08] Frank Ciesinski, Christel Baier, Marcus Größer and Joachim Klein. 'Reduction Techniques for Model Checking Markov Decision Processes'. In: *Proceedings of the 5th International Conference on Quantitative Evaluation of Systems (QEST).* IEEE Computer Society, 2008, pp. 45–54. ISBN: 978-0-7695-3360-5.

[Cim+02] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani and Armando Tacchella. 'NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking'. In: *Proceedings of the International Conference on Computer-Aided Verification (CAV).* Vol. 2404. Lecture Notes in Computer Science. Springer, July 2002.

[Cla11] Ulrich Clauß. *Wie das Internet zum Klimakiller wird.* [retrieved: 28th March 2018]. 2011. URL: `http://www.welt.de/webwelt/article13391627/`.

[CMH06] Krishnendu Chatterjee, Rupak Majumdar and Thomas A. Henzinger. 'Markov Decision Processes with Multiple Objectives'. In: *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS).* Springer, 2006, pp. 325–336. ISBN: 978-3-540-32288-7.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. 'The Complexity of Probabilistic Verification'. In: *Journal of the ACM (JACM)* 42.4 (1995), pp. 857–907. ISSN: 0004-5411.

[Deh+17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen and Matthias Volk. 'A storm is Coming: A Modern Probabilistic Model Checker'. In: *Proceedings of the 29th International Conference on Computer Aided Verification (CAV), Part II.* Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 592–600.

[Ete+07] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi and Mihalis Yannakakis. 'Multi-Objective Model Checking of Markov Decision Processes'. In: *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).* Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 50–65.

[EY09] Kousha Etessami and Mihalis Yannakakis. 'Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations'. In: *Journal of the ACM (JACM)* 56.1 (2009), 1:1–1:66. ISSN: 0004-5411.

[For+11a] Vojtěch Forejt, Marta Z. Kwiatkowska, Gethin Norman and David Parker. 'Automated Verification Techniques for Probabilistic Systems'. In: *Proceedings of the 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM).* Vol. 6659. Lecture Notes in Computer Science. Springer, 2011, pp. 53–113. ISBN: 978-3-642-21455-4.

[For+11b] Vojtěch Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker and Hongyang Qu. 'Quantitative Multi-Objective Verification for Probabilistic Systems'. In: *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).* Vol. 6605. Lecture Notes in Computer Science. Springer, 2011, pp. 112–127.

[Gar14] Hubert Garavel. *Three decades of success stories in formal methods.* [retrieved: 28th March 2018]. 2014. URL: `http://repmus.ircam.fr/_media/garavel-fm-14-v7-grenoble.pdf`.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Company, 1979. ISBN: 0-7167-1044-7.

[GLS93] Martin Grötschel, László Lovász and Alexander Schrijver. 'The Ellipsoid Method'. In: *Geometric Algorithms and Combinatorial Optimization.* Springer, 1993, pp. 64–101. ISBN: 978-3-642-78240-4.

[Gra+03] Ananth Grama, Anshul Gupta, George Karypis and Vipin Kumar. *Introduction to Parallel Computing.* 2nd Edition. Addison-Wesley, 2003. ISBN: 978-0-201-64865-2.

[GTW02] Erich Grädel, Wolfgang Thomas and Thomas Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research.* Vol. 2500. Lecture Notes in Computer Science. Springer, 2002.

[Hah+13] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns and Joost-Pieter Katoen. 'A compositional modelling and analysis framework for stochastic hybrid systems'. In: *Formal Methods in System Design* 43.2 (Oct. 2013), pp. 191–232.

[Häh+13] Marcus Hähnel, Björn Döbel, Marcus Völp and Hermann Härtig. 'eBond: Energy Saving in Heterogeneous R.A.I.N'. In: *Proceedings of the 4th International Conference on Future Energy Systems.* e-Energy '13. ACM, 2013, pp. 193–202. ISBN: 978-1-4503-2052-8.

[Hah+14] Ernst Moritz Hahn, Yi Li, Sven Schewe, Andrea Turrini and Lijun Zhang. 'iscasMc: A Web-Based Probabilistic Model Checker'. In: *Proceedings of the 19th International Symposium of the Formal Methods Europe association (FM).* Vol. 8442. Lecture Notes in Computer Science. Springer, 2014, pp. 312–317.

[Hav98] Boudewijn R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* Wiley, Jan. 1998. ISBN: 978-0-471-97228-0.

[Hen+12] David Henriques, Joao G. Martins, Paolo Zuliani, André Platzer and Edmund M. Clarke. 'Statistical Model Checking for Markov Decision Processes'. In: *Proceedings of the 9th International Conference on Quantitative Evaluation of Systems (QEST).* IEEE Computer Society, 2012, pp. 84–93. ISBN: 978-0-7695-4781-7.

[HH14] Arnd Hartmanns and Holger Hermanns. 'The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification'. In: *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).* 2014, pp. 593–598.

[HH16] Ernst Moritz Hahn and Arnd Hartmanns. 'A Comparison of Time- and Reward-Bounded Probabilistic Model Checking Techniques'. In: *Proceedings of the 2nd International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA).* Springer, 2016, pp. 85–100. ISBN: 978-3-319-47677-3.

[HJ94] Hans A. Hansson and Bengt Jonsson. 'A logic for reasoning about time and reliability'. In: *Formal Aspects of Computing* 6.5 (1994), pp. 512–535.

[HK15] Christoph Haase and Stefan Kiefer. 'The Odds of Staying on Budget'. In: *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part II.* 2015, pp. 234–246.

[HKL17] Christoph Haase, Stefan Kiefer and Markus Lohrey. 'Computing quantiles in Markov chains with multi-dimensional costs'. In: *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).* 2017, pp. 1–12.

[HM14] Serge Haddad and Benjamin Monmege. 'Reachability in MDPs: Refining Convergence of Value Iteration'. In: *Proceedings of the 8th International Workshop on Reachability Problems (RP)*. Cham: Springer, 2014, pp. 125–137. ISBN: 978-3-319-11439-2.

[How90] Ronald A. Howard. *Dynamic Programming and Markov Processes*. John Wiley and Sons, 1990.

[HRS13] Monika Heiner, Christian Rohr and Martin Schwarick. 'MARCIE – Model checking And Reachability analysis done effiCIEntly'. In: *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Vol. 7927. Lecture Notes in Computer Science. Springer, 2013, pp. 389–399. ISBN: 978-3-642-38697-8.

[Hun07] John D. Hunter. 'Matplotlib: A 2D graphics environment'. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[IJ90] Amos Israeli and Marc Jalfon. 'Token Management Schemes and Random Walks Yield Self-stabilizing Mutual Exclusion'. In: *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM New York, 1990, pp. 119–131.

[IR90] Alon Itai and Michael Rodeh. 'Symmetry Breaking in Distributed Networks'. In: *Information and Computation* 88.1 (1990), pp. 60–87.

[JLS07] Marcin Jurdziński, François Laroussinie and Jeremy Sproston. 'Model Checking Probabilistic Timed Automata with One or Two Clocks'. In: *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 170–184. ISBN: 978-3-540-71208-4.

[Kat+09] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns and David N. Jansen. 'The Ins and Outs of the Probabilistic Model Checker MRMC'. In: *Proceedings of the 6th International Conference on the Quantitative Evaluation of Systems (QEST)*. Sept. 2009, pp. 167–176. ISBN: 978-0-7695-3808-2.

[Kle+16] Joachim Klein, Christel Baier, Philipp Chrszon, Marcus Daum, Clemens Dubslaff, Sascha Klüppelholz, Steffen Märcker and David Müller. 'Advances in Symbolic Probabilistic Model Checking with PRISM'. In: *Proceedings of the 22th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 9636. Lecture Notes in Computer Science. Springer, 2016, pp. 349–366.

[Kle+17] Joachim Klein, Christel Baier, Philipp Chrszon, Marcus Daum, Clemens Dubslaff, Sascha Klüppelholz, Steffen Märcker and David Müller. 'Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata'. In: *International Journal on Software Tools for Technology Transfer (STTT)* (2017), pp. 1–16.

[KNP05] Marta Z. Kwiatkowska, Gethin Norman and David Parker. ‘Probabilistic model checking in practice: Case studies with PRISM’. In: *ACM SIGMETRICS Performance Evaluation Review* 32.4 (2005), pp. 16–21.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman and David Parker. ‘PRISM 4.0: Verification of Probabilistic Real-time Systems’. In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591.

[KNP12] Marta Z. Kwiatkowska, Gethin Norman and David Parker. ‘The PRISM Benchmark Suite’. In: *Proceedings of the 9th International Conference on Quantitative Evaluation of SysTems (QEST)*. IEEE CS Press, 2012, pp. 203–204.

[KNS01] Marta Z. Kwiatkowska, Gethin Norman and Roberto Segala. ‘Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM’. In: *Proceedings of the 13th International Conference on Computer Aided Verification (CAV)*. Vol. 2102. Lecture Notes in Computer Science. Springer, 2001, pp. 194–206.

[Krä+15] Daniel Krähmann, Jana Schubert, Christel Baier and Clemens Dubslaff. ‘Ratio and Weight Quantiles’. In: *Proceedings of the 40th Symposium on Mathematical Foundations of Computer Science (MFCS), Part I*. Vol. 9234. Lecture Notes in Computer Science. Springer, 2015, pp. 344–356.

[Kul95] Vidyadhar G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. London, UK, UK: Chapman & Hall, Ltd., 1995. ISBN: 0-412-04991-0.

[Lee59] C. Y. Lee. ‘Representation of Switching Circuits by Binary-Decision Programs’. In: *The Bell System Technical Journal* 38.4 (July 1959), pp. 985–999. ISSN: 0005-8580.

[LP02] Richard Lassaigne and Sylvain Peyronnet. ‘Approximate Verification of Probabilistic Systems’. In: *Proceedings of the 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV)*. Vol. 2399. Lecture Notes in Computer Science. Springer, 2002, pp. 213–214. ISBN: 978-3-540-45605-6.

[LPY97] Kim G. Larsen, Paul Pettersson and Wang Yi. ‘Uppaal in a Nutshell’. In: *International Journal on Software Tools for Technology Transfer (STTT)* 1.1–2 (1997), pp. 134–152.

[LS05] François Laroussinie and Jeremy Sproston. ‘Model Checking Durational Probabilistic Systems’. In: *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*. Springer, 2005, pp. 140–154.

[Mar+02] Deborah T. Marr, Frank Binns, David L. Hill, Glenn Hinton, David A. Koufty, J. Alan Miller and Michael Upton. ‘Hyper-Threading Technology Architecture and Microarchitecture’. In: *Intel Technology Journal* 6 (Jan. 2002).

[Mär+17] Steffen Märcker, Christel Baier, Joachim Klein and Sascha Klüppelholz. 'Computing Conditional Probabilities: Implementation and Evaluation'. In: *Proceedings of the 15th International Conference on Software Engineering and Formal Methods (SEFM)*. Vol. 10469. Lecture Notes in Computer Science. Springer, 2017, pp. 349–366.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. ISBN: 0-7923-9380-5.

[Nas00] John C. Nash. 'The (Dantzig) Simplex Method for Linear Programming'. In: *Computing in Science and Engineering* 2.1 (Jan. 2000), pp. 29–31.

[Neu97] Karl-Dietrich Neubert. 'The FlashSort Algorithm'. In: *Proceedings of the euroFORTH-Conference*. 1997.

[Par02] David Parker. 'Implementation of Symbolic Model Checking for Probabilistic Systems'. PhD thesis. University of Birmingham, 2002.

[Pnu77] Amir Pnueli. 'The temporal logic of programs'. In: *18th Annual Symposium on Foundations of Computer Science, Providence (FOCS)*. IEEE Computer Society, 1977, pp. 46–57.

[PSL00] Anna Pogosyants, Roberto Segala and Nancy Lynch. 'Verification of the randomized consensus algorithm of Aspnes and Herlihy: A case study'. In: *Distributed Computing* 13.3 (2000), pp. 155–186.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994. ISBN: 0471619779.

[QS82] Jean-Pierre Queille and Joseph Sifakis. 'Specification and verification of concurrent systems in CESAR'. In: *Proceedings of the 5th International Symposium on Programming*. Vol. 137. Lecture Notes in Computer Science. Springer, 1982, pp. 337–351. ISBN: 978-3-540-39184-5.

[RP09] Diana El Rabih and Nihal Pekergin. 'Statistical Model Checking Using Perfect Simulation'. In: *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Vol. 5799. Lecture Notes in Computer Science. Springer, 2009, pp. 120–134. ISBN: 978-3-642-04761-9.

[RRS15] Mickael Randour, Jean-François Raskin and Ocan Sankur. 'Percentile Queries in Multi-dimensional Markov Decision Processes'. In: *Proceedings of the 27th International Conference on Computer Aided Verification (CAV), Part I*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 123–139.

[Seg95] Roberto Segala. 'Modeling and Verification of Randomized Distributed Real-Time Systems'. PhD thesis. Massachusetts Institute of Technology, 1995.

[Ser80] Robert J. Serfling. *Approximation Theorems of Mathematical Statistics*. Wiley Series in Probability and Statistics - Applied Probability and Statistics Section Series. Wiley, 1980. ISBN: 978-0471024033.

[SVA05] Koushik Sen, Mahesh Viswanathan and Gul Agha. ‘On Statistical Model Checking of Stochastic Systems’. In: *Proceedings of the 17th International Conference on Computer Aided Verification (CAV)*. Vol. 3576. Lecture Notes in Computer Science. Springer, 2005, pp. 266–280. ISBN: 978-3-540-31686-2.

[Tar72] Robert E. Tarjan. ‘Depth-First Search and Linear Graph Algorithms’. In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160.

[UB13] Michael Ummels and Christel Baier. ‘Computing Quantiles in Markov Reward Models’. In: *Proceedings of the 16th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*. Vol. 7794. Lecture Notes in Computer Science. Springer, 2013, pp. 353–368.

[Var85] Moshe Y. Vardi. ‘Automatic Verification of Probabilistic Concurrent Finite-State Programs’. In: *26th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1985, pp. 327–338.

[VW86] Moshe Y. Vardi and Pierre Wolper. ‘An Automata-Theoretic Approach to Automatic Program Verification’. In: *Proceedings of the Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 1986, pp. 332–344.

[YS02] Håkan L. S. Younes and Reid G. Simmons. ‘Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling’. In: *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 223–235. ISBN: 978-3-540-45657-5.

List of Figures

1.1 Schematic overview of probabilistic model checking 3

3.1 Quantiles for increasing and decreasing path properties 24
3.2 Quantile for increasing path property 27
3.3 Linear program $\mathbb{LP}_r$ with the unique solution $p_{s,i} = \mathsf{Pr}^{\max}_s\big(A\,\mathcal{U}^{\leqslant i}\,B\big)$ . . 29
3.4 Algorithm 1 from [UB13] 31
3.5 Encoding of transition rewards into state rewards 32

6.1 Results for Self-Stabilising Protocol 88
6.2 Parallel computations for Self-Stabilising Protocol 91
6.3 Results for Synchronous Leader-Election Protocol 97
6.4 Results for Asynchronous Leader-Election Protocol 98
6.5 Parallel computations for Asynchronous Leader-Election Protocol . . . 100
6.6 Results for Randomised Consensus Shared Coin Protocol 108
6.7 Parallel computations for Randomised Consensus Shared Coin Protocol 110
6.8 Control flow for process $\mathcal{P}_i$ (Energy-Aware Job-Scheduling Protocol) . . 116
6.9 Control flow for resource $\mathcal{R}$ (Energy-Aware Job-Scheduling Protocol) . 117
6.10 Results for quantile $\mathrm{e}_{\mathrm{min}}$ (Energy-Aware Job-Scheduling Protocol) . . . 120
6.11 Results for quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol) . . . 121
6.12 Results for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, no turbo mode) 125
6.13 Results for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, enable turbo after first miss) 127
6.14 Results for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, round-robin scheduling) 129
6.15 Results for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, round-robin scheduling, zoomed) 130
6.16 Results for quantile $E^{\exists}_s$ (Energy-Aware Job-Scheduling Protocol) 131
6.17 Results for quantile $\mathrm{e}_{\mathrm{min}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources) 133
6.18 Results for quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources) 134
6.19 Results for quantile $\mathrm{e}^{95\%}_{\mathrm{min}}$ (eBond) 139
6.20 Storage for different reward windows (eBond) 141
6.21 Timings for different reward windows (eBond) 142
6.22 Topology of HAECubie demonstrator 143
6.23 Topology of abstracted HAECubie demonstrator 144

6.24 Control flow of a switch (HAECubie) 145
6.25 Control flow of a cluster (HAECubie) 146
6.26 Control flow of environment (HAECubie) 147
6.27 Control flow of video-storage (HAECubie) 148
6.28 Control flow of the master (controlling video storage, HAECubie) . . . 149
6.29 Control flow of the master (controlling energy consumption, HAECubie) 149
6.30 Results for HAECubie (minimal energy consumption until 3 videos were streamed successfully) 151
6.31 Results for HAECubie (minimal expected energy consumption for specific utility) 152

List of Tables

6.1 Columns for the statistics of explicit quantiles 86
6.2 Columns for the statistics of symbolic quantiles 86
6.3 Comparison between *LP*- and *BP*-approach (Self-Stabilising Protocol) . 90
6.4 Statistics for explicit existential quantile (Self-Stabilising Protocol) . . 92
6.5 Statistics for symbolic existential quantile (Self-Stabilising Protocol) . . 93
6.6 Statistics for explicit universal quantile (Self-Stabilising Protocol) . . . 94
6.7 Statistics for symbolic universal quantile (Self-Stabilising Protocol) . . 95
6.8 Comparison between *LP*- and *BP*-approach (minimal number of rounds, Asynchronous Leader-Election Protocol) 101
6.9 Statistics for explicit quantile (minimal number of rounds, Synchronous Leader-Election Protocol) 101
6.10 Statistics for symbolic quantile (minimal number of rounds, Synchronous Leader-Election Protocol) 102
6.11 Statistics for explicit quantile (minimal number of steps, Synchronous Leader-Election Protocol) 103
6.12 Statistics for symbolic quantile (minimal number of steps, Synchronous Leader-Election Protocol) 104
6.13 Statistics for explicit existential quantile (minimal number of rounds, Asynchronous Leader-Election Protocol) 105
6.14 Statistics for symbolic existential quantile (minimal number of rounds, Asynchronous Leader-Election Protocol) 105
6.15 Statistics for explicit existential quantile (minimal number of steps, Asynchronous Leader-Election Protocol) 106
6.16 Statistics for symbolic existential quantile (minimal number of steps, Asynchronous Leader-Election Protocol) 106
6.17 Statistics for explicit quantile (minimal number of coin-flips, Randomised Consensus Shared Coin Protocol) 111
6.18 Statistics for symbolic quantile (minimal number of coin-flips, Randomised Consensus Shared Coin Protocol) 112
6.19 Statistics for explicit quantile (minimal number of steps, Randomised Consensus Shared Coin Protocol) 113
6.20 Statistics for symbolic quantile (minimal number of steps, Randomised Consensus Shared Coin Protocol) 114
6.21 Rewards modelling the energy consumption of a process (Energy-Aware Job-Scheduling Protocol) 118
6.22 Statistics for explicit quantile u_{max} (without tailored storage, Energy-Aware Job-Scheduling Protocol) 122

6.23 Statistics for explicit quantile $\mathrm{e}_{\mathrm{min}}$ (Energy-Aware Job-Scheduling Protocol) 123
6.24 Statistics for symbolic quantile $\mathrm{e}_{\mathrm{min}}$ (Energy-Aware Job-Scheduling Protocol) . . . 123
6.25 Statistics for explicit quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol) . . . 124
6.26 Statistics for symbolic quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol) . . . 124
6.27 Statistics for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, no turbo mode) . . . 126
6.28 Statistics for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, enable turbo after first miss) . . . 128
6.29 Statistics for quantile $\mathrm{e}_{\mathrm{min}}$ with side conditions (Energy-Aware Job-Scheduling Protocol, round-robin scheduling) . . . 129
6.30 Statistics for quantile $E_s^{\exists}$ (Energy-Aware Job-Scheduling Protocol) . . . 131
6.31 Statistics for explicit quantile $\mathrm{e}_{\mathrm{min}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources) . . . 135
6.32 Statistics for symbolic quantile $\mathrm{e}_{\mathrm{min}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources) . . . 135
6.33 Statistics for explicit quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources) . . . 136
6.34 Statistics for symbolic quantile $\mathrm{u}_{\mathrm{max}}$ (Energy-Aware Job-Scheduling Protocol, 2 shared resources) . . . 136
6.35 Statistics for quantile $\mathrm{e}_{\mathrm{min}}^{95\%}$ (eBond) . . . 139
6.36 Statistics for expectation quantile (minimal expected energy consumption for specific utility, HAECubie) . . . 153
6.37 Statistics for explicit reachability quantile (minimal energy consumption until 3 videos were streamed successfully, HAECubie) . . . 154
6.38 Statistics for symbolic reachability quantile (minimal energy consumption until 3 videos were streamed successfully, HAECubie) . . . 154