Towards Efficient Analysis of Markov Automata

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften (Dr.-Ing.) der Fakultät für Mathematik und Informatik der Universität des Saarlandes

vorgelegt von:
Yuliya Butkova

Saarbrücken, 2020

Dean of Faculty:
Univ.-Prof. Dr. Thomas Schuster

Date of Colloquium:
04.06.2020

Chair of the Committee:
Prof. Bernd Finkbeiner, Ph.D.

Reviewers:
Prof. Dr. Holger Hermanns
Prof. Dr. Lijun Zhang
Prof. Dr. Jan Křetínský

Academic Assistant:
Dr. Roland Leißa

Acknowledgements

This thesis would not be possible without the tremendous support from a number of remarkable people. First of all, my gratitude goes to my supervisor Holger Hermanns for seeing my potential and giving me the opportunity to devote several years of my life to research. I would like to thank him for always finding the time and patience to discuss my (mostly completely broken) ideas, for tolerating and fixing my incompetence in academic correspondence, for guiding and helping me in the academic and the real world.

I am grateful to Jan Křetínský and Lijun Zhang for agreeing to review my thesis. I had a pleasure to work with Jan for several months. His positive and cheerful attitude was contagious and kept me and Pranav motivated to reach our objectives.

I am grateful to Arnd Hartmanns for hosting my visit to the University of Twente, for our fruitful discussions and a great barbecue. During this visit, I have also had a chance to chat with Mariëlle Stoelinga and Carlos E. Budde. Despite the combined effort of Mariëlle, Holger and Carlos to bring me to the beautiful world of fault trees, my heart was already given to Markov automata. Nevertheless, thank you all for your time and effort.

I would like to thank all the members of the Dependable Systems and Software chair for maintaining such a fun, positive and lively work environment. Each time I was sitting in front of the whiteboard frustrated with yet another one of my ideas being broken, the honking wooden bike passing by my office reminded me that there is life outside of research. I am grateful to Sabine Nermerich and Christa Schäfer, who helped to keep all my bureaucratic matters in check, something that I am so incapable to do on my own.

Throughout my PhD time, I also had a pleasure to collaborate with Pranav Ashok, Pedro R. D'Argenio, Gereon Fox, Hassan Hatefi, Jan and Pavel Krčál, Daniel Stan and Ralf Wimmer on a handful of amazing projects. Thank you for all the inspiring discussions we had, I enjoyed working with you.

Special thanks go to the Saarbrücken Graduate School of Computer Science. I spent a great deal of time in the common areas and many of my fellows became my true friends in non-academic life. Aurela, John, Janis, Susanne, Charalampos, and Pedro, thank you all for being there with me.

I would like to thank my mum, dad, brother, grandma and grandpa for all the love, care and understanding they gave me throughout all these years.

And most importantly, I want to thank from the bottom of my heart my husband Giulio. Thank you for pushing my limits, for being there with me through joy and pain, and for reminding me that even the greatest minds doubt and question their competence.

Zusammenfassung

Markov-Automaten bilden einen der ausdrucksstärksten Formalismen um Nebenläufige Systeme zu modellieren. Sie werden benutzt um die Semantik vieler höherer Formalismen wie stochastischer Petri-Netze [Mar95, EHZ10] und Dynamic Fault Trees [DBB90] zu beschreiben.

Die zwei herausfordernder Probleme im Bereich der Analyse großer Markov-Automaten sind (i) *die zeitbeschränkten Erreichbarkeitwahrscheinlichkeit* und (ii) *optimale langfristige durchschnittliche Rewards*. Diese Arbeit zielt auf das Design effizienter und korrekter Techniken um sie zu untersuchen.

Das Problem der zeitbeschränkten Erreichbarkeitswahrscheinlichkeit gehen wir aus zwei verschiedenen Richtungen an: Zum einen studieren wir die Eigenschaften optimaler Lösungen und nutzen dieses Wissen um einen effizienten Approximationsalgorithmus zu bilden, der optimale Werte bis auf eine garantierte Fehlertoleranz berechnet. Dieser Algorithmus basiert darauf, Werte für jeden Zustand des Markov-Automaten zu berechnen. Dies kann die Anwendbarkeit für große oder gar unendliche Automaten einschränken. Um diese Problem zu lösen präsentieren wir einen zweiten Algorithmus, der die optimale Lösung approximiert, und dabei ausschließlich einen Teil des Zustandsraumes betrachtet.

Für das Problem der optimalen langfristigen durchschnittlichen Rewards gibt es einen polynomiellen Algorithmus auf Basis linearer Programmierung. Anstelle eine bessere theoretische Komplexität anzustreben, konzentrieren wir uns darauf, eine praktische Lösung auf Basis eines iterativen Ansatzes zu finden. Wie entwickeln einen Werte-iterierenden Algorithmus der in unserer empirischen Evaluation um mehrere Größenordnungen besser als der auf linearer Programmierung basierende Ansatz skaliert.

Abstract

One of the most expressive formalisms to model concurrent systems is Markov automata. They serve as a semantics for many higher-level formalisms, such as generalised stochastic Petri nets [Mar95, EHZ10] and dynamic fault trees [DBB90].

Two of the most challenging problems for Markov automata to date are (i) *the optimal time-bounded reachability probability* and (ii) *the optimal long-run average rewards.* In this thesis, we aim at designing efficient sound techniques to analyse them.

We approach the problem of time-bounded reachability from two different angles. First, we study the properties of the optimal solution and exploit this knowledge to construct an efficient algorithm that approximates the optimal values up to a guaranteed error bound. This algorithm is *exhaustive*, i. e. it computes values for each state of the Markov automaton. This may be a limitation for very large or even infinite Markov automata. To address this issue we design a second algorithm that approximates the optimal solution by only working with part of the total state-space.

For the problem of long-run average rewards there exists a polynomial algorithm based on linear programming. Instead of chasing a better theoretical complexity bound we search for a practical solution based on an iterative approach. We design a value iteration algorithm that in our empirical evaluation turns out to scale several orders of magnitude better than the linear programming based approach.

Contents

1 Introduction **1**
1.1 Modelling Concurrent Systems 2
1.2 System Properties 4
1.3 Analysis 6
1.4 Contribution 10
1.5 Chapter Origins 11
1.6 Overview 11

2 Foundations **13**
2.1 Mathematical Notation and Definitions 13
2.2 Markov Automata 14
2.2.1 Modelling with Markov Automata 16
2.2.2 Operational Behaviour of Closed Markov Automata 18
2.2.3 Probability Measure 20
2.2.4 Zeno Behaviour 25
2.2.5 Important Subclasses and Related Models 26

3 Analysis of Markov Automata **29**
3.1 Continuous Stochastic Logic for Markov Automata 30
3.2 Analysis of CSL Properties 32
3.3 Reward Properties 34
3.4 Analysis of Reward Properties 40
3.5 Discussion 40

4 Time-Bounded Reachability **43**
4.1 Preliminaries 44
4.1.1 Known Solutions and Related Work 44
4.1.2 Reduction to Two-Step Solution 48
4.2 Optimal Time-Bounded Reachability Problem 63
4.2.1 Optimal Schedulers 64
4.2.2 Hop-Unbounded Reachability Sub-problem 65
4.3 A New Solution for Time-Bounded Reachability 67
4.3.1 Time-Bounded Reachability for a Scheduler 69
4.3.2 Optimal Strategy 77
4.3.3 Switching Points 86
4.3.4 Approximation of Switching Points 88

4.3.5 Algorithm . . . 106
4.4 Time-Bounded Reachability on Partial State-Space . . . 114
4.4.1 Relevant Subset . . . 116
4.4.2 Partial Markov Automaton . . . 117
4.4.3 Under- and Over-Approximations . . . 118
4.4.4 The Choice of Simulating Scheduler . . . 122
4.4.5 The Algorithm . . . 124
4.5 Experimental Evaluation . . . 130
4.5.1 Comparison of Exhaustive Approaches . . . 130
4.5.2 Comparison of Partial Approaches . . . 132
4.5.3 Conclusions . . . 135
4.6 Conclusions and Discussion . . . 136

5 Optimal Long-Run Rewards 139
5.1 Preliminaries and Problem Statement . . . 140
5.1.1 Cumulative Unbounded Reward for MDPs . . . 140
5.2 General Case Solution . . . 142
5.3 Iterative Approach . . . 145
5.3.1 Long-run Rewards for CTMDPs . . . 146
5.3.2 Value Preserving CTMDP . . . 147
5.3.3 Dealing with Exponentiality . . . 152
5.3.4 Algorithmic Solution . . . 158
5.3.5 Experimental Evaluation . . . 162
5.4 Conclusions and Discussion . . . 164

6 Conclusions 167

Bibliography 169

A Proofs 177
A.1 Auxiliary Lemmas for Section 4.1.2 . . . 177
A.2 Auxiliary Lemmas For Lemma 4.3.2 . . . 179
A.3 Auxiliary Lemmas for Lemma 4.3.6 . . . 181
A.4 Auxiliary Lemma for Lemma 4.3.15 . . . 186
A.5 Auxiliary Lemma for Lemma 4.3.16 . . . 187

B Experiments 189
B.1 Command Line Arguments . . . 189
B.1.1 Time-Bounded Reachability . . . 189
B.1.2 Long-Run Average Rewards . . . 229
B.2 Case Studies . . . 244
B.2.1 Time-Bounded Reachability (exhaustive) and Long-Run Average Rewards . . . 244
B.2.2 Time-Bounded Reachability (partial) . . . 253

Introduction 1

In our every day lives we often interact with highly complex systems: electricity is generated in nuclear power plants, trains and planes are nowadays the usual means of transportation, many cities are being equipped with a self-driving subway, and in a few areas one is already allowed to use the autopilot of self-driving cars.

The convenience of modern life comes at a price though. Malfunctioning of one of those systems has the potential to endanger the lives of hundreds of people at once. When we trust our lives to a complicated piece of hardware controlled by an even more complex piece of software we need to understand how these systems work. One of the ways to achieve this is by answering basic queries about the system functionality: Does the temperature of a nuclear reactor ever reach dangerous levels? Is it possible that two trains moving towards each other will occupy the same rail track? Would a self-driving car take better decisions faster than the person in that car? And so on.

Problems of this kind are studied in the scientific area of formal methods. In this thesis, our interest lies in a specific sub-field of formal methods. Namely, we will consider the formal analysis of scenarios in which multiple parties act concurrently and interact with each other. The behaviour of the parties may be to a certain degree unknown. For example, we may not know exactly what a participant will do next, except for the set of possible next steps, or we may have access to a probability distribution over all possible next steps. Our goal is to establish whether certain properties of the system as a whole hold. As an example, consider the following:

Example 1. *Consider the situation in which multiple self-driving cars are present on the roads. The cars behave mostly independently, however from time to time they need to exchange information, such as notifying neighbouring cars when a change of lane is needed, when the speed is increased or decreased, when a car has to exit the highway, and so on. From the receiving car's perspective, messages from another car can arrive at random at any point in time. The probability of message arrival may be estimated based on, for example, the location of the car (e. g. more messages close to highway exits). Car hardware is subject to failures, which means that cars may behave differently from what they are supposed to do or from what the car* *announced it will do. If the probability of some hardware component to malfunction is available from statistical data, then the behaviour of a car may be considered probabilistic. We want to make sure that the software controlling the cars is safe under normal circumstances and responds properly to abnormal situations on the road. This requires estimating the worst-case probability of an undesirable situation to occur under any possible conditions, i. e. for any number of cars on the road, state of the car hardware, any sequence of exchanged messages, and so on.*

Formally speaking, we will consider probabilistic systems running in continuous time, in parallel to other systems that they can interact with. In the following, we will refer to such systems as *concurrent systems*. We are interested in estimating *dependability* and *performance* metrics of concurrent systems, such as the probability of catastrophic consequences to occur, reliable performance within a time period, the average energy consumption, etc. Large concurrent systems may contain so many interacting sub-components, that no human can possibly predict and evaluate all the consequences of this interaction. We, therefore, need ways to (i) formally describe these systems as well as (ii) the requirements we want them to fulfil and develop computer-assisted techniques to (iii) check whether the systems satisfy these requirements.

1.1 Modelling Concurrent Systems

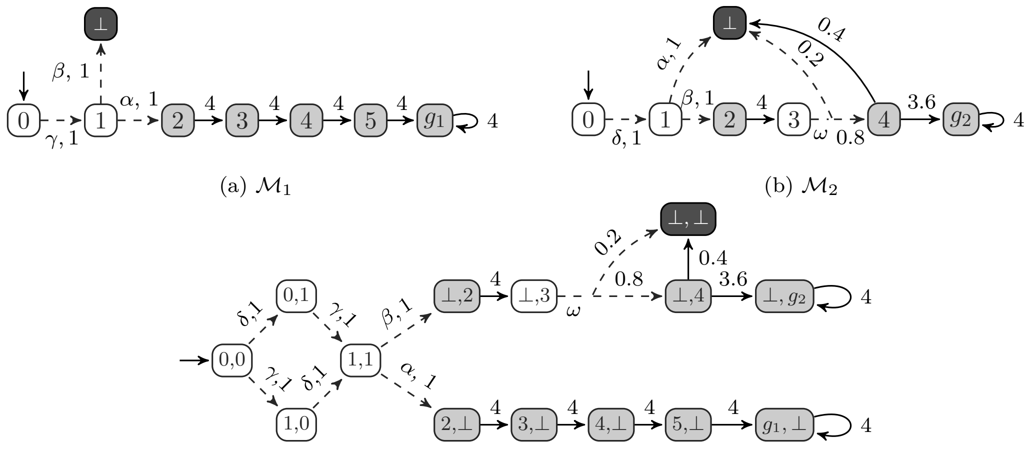
In order to formally analyse systems with respect to a dependability or a performance query, one needs a formal model that represents those aspects of system behaviour that are relevant for these questions. Throughout the years, concurrent systems have been studied from various angles:

Interaction of independent agents is the topic of research of process algebras, such as *communicating sequential processes* (CSP) [Hoa78] or *calculus of communicating systems* (CCS) [Mil80]. They provide means to describe how multiple independent concurrent processes evolve and communicate with each other in discrete time, i. e. abstracting away from the actual timing of events. The underlying mathematical model of such process algebras is often a *labelled transition system* (LTS). An LTS is a pair, consisting of a finite set of states and a finite set of labelled transitions. The latter model possible evolution of the system from one state to another. We will refer to labels of transitions as *actions*. A situation when multiple outgoing transitions are available for a state is commonly referred to as *non-determinism*. Process algebras model with an LTS each of the interacting agents, as well as the concurrent system as a whole, comprising of multiple interacting processes. The latter is enabled by *compositionality* of LTS, i. e. the possibility to obtain one LTS model, encompassing the behaviour of multiple other LTS models.

Probabilistic uncertainty. If transitioning from one state to another in a process can only be ensured with a certain probability, then such a process can be modelled as a *probabilistic automaton* (PA) [SL94]. Probabilistic automata generalise labelled transition systems by assigning a probability distribution to state-action pairs. Just like LTS, probabilistic automata are compositional. A similar formalism is that of *discrete time Markov decision process* (MDP) [Put94]. As opposed to PAs, MDPs are not compositional and thus cannot be used to model concurrent systems.

Timed behaviour. For many dependability properties, the actual timing of transitions is important. One of the most widely used models that incorporates this information is the *continuous-time Markov chain* (CTMC) [Ros06]. Similarly to LTS, CTMCs can be described by a pair of a finite set of states and a finite set of labelled transitions. In CTMCs, however, transitioning from state to state occurs with a timed delay. The value of the delay depends on labels of the transitions, which can only be strictly positive real numbers and are called *rates*. The meaning of these labels is the following: An expected timed delay when taking a transition labelled with λ is $1/\lambda$ time units. Formally speaking, the value of the delay is a random variable distributed *exponentially*, i. e. the probability that the transition will happen within t time units is $1 - e^{-\lambda \cdot t}$, where e is Euler's number and λ is the average transition rate. If multiple outgoing transitions are available in a state, then the CTMC chooses a transition that selected the smallest delay value. Thus CTMCs model time somewhat imprecisely. We will refer to this timing model as *stochastic timing*.

Markov Automata (MA) [EHZ10] can model all aspects of concurrent systems listed above and is one of the most general formalisms to model concurrent systems. Notable sub-classes of Markov automata are LTS, PAs, MDPs, CTMCs and *interactive Markov chains* [Her02]. Many higher level formalisms, such as *generalised stochastic Petri nets* [Mar95, EHZ10] and *dynamic fault trees* [DBB90], have Markov automata as their underlying semantics. We will informally introduce the modelling power of Markov automata with the help of a few examples, shown in Figure 1.1. Consider the Markov automaton $\mathcal{M}_2$. Different colours of states refer to the type of outgoing transitions available in a state. States in white colour have only LTS or PA style transitions (dashed). For example, state (0) has only one transition, labelled with action δ, that leads with probability 1 to state (1). State (3) has a PA style transition labelled with ω. After making this transition the next state will be either $(\perp)$ with probability 0.2, or (4) with probability 0.8. Grey coloured states have CTMC-style transitions with timed delays, which are denoted by solid edges. Finally, states in dark grey colour have no outgoing transitions at all. The Markov automaton shown in Figure (1.1c) represents the behaviour of $\mathcal{M}_1$ and $\mathcal{M}_2$ running concurrently. A state of this MA is a pair, composed of a state of $\mathcal{M}_1$ and a state of $\mathcal{M}_2$. These two MA can perform transitions either independently (interleaving), or simultaneously (synchronously). The latter serve as a basic communication mechanism for Markov automata. For example, the γ-labelled transition in $\mathcal{M}_1$ and the δ-labelled transition in $\mathcal{M}_2$ are performed independently. Therefore in state $(0, 0)$ of $\mathcal{M}$ there are two transitions labelled γ and δ leading to $(1, 0)$ and $(0, 1)$ respectively, depending on which MA performs the transition. Transitions α and β are performed synchronously. Whenever $\mathcal{M}_1$ is in state (1) and $\mathcal{M}_2$ is in state (1), actions α and β are available for both of them. If, for example, α is selected, then $\mathcal{M}_1$ transitions to state (2) and $\mathcal{M}_2$ transitions to $(\perp)$ simultaneously. This



(c) $\mathcal{M} = \mathcal{M}_1 || \mathcal{M}_2$

Figure 1.1

is reflected in $\mathcal{M}$ with the α-labelled transition leading from state $(1, 1)$ to state $(2, \perp)$.

Describing large Markov automata by writing down all state-transition pairs, just like in Figure 1.1, would be space-consuming and error-prone. An additional advantage of Markov automata is the availability of a high-level modelling language `Modest` [HHHK13] that provides a convenient way to describe MA models. The `Modest` language provides process algebraic operators, such as parallel composition and sequential composition, it supports typed variables including `bool`, `int`, `real`, arrays, and others, and its syntax for expressions is similar to those of `C` programming languages. All in all, `Modest` is more verbose than classic process algebras, but also more readable and beginner-friendly. The `Modest` language is a supported input language of the `Modest Toolset` [HH14] and `Storm` [DJKV17] (via translation to `Jani` input language), which makes it easy to debug and analyse Markov automata.

There is another model that is very similar to Markov automata, called a *continuous-time Markov decision process* (CTMDP) [Put94]. In CTMDPs, timed transitions (CTMC-style) and non-deterministic transitions (PA-style) are merged together, forming only one type of transitions. One of the differences between CTMDPs and Markov automata is that CTMDPs are not compositional. This means that CTMDPs cannot be used to model a concurrent system by creating a CTMDP model of each of its sub-components.

In this thesis, we will be using Markov automata to model concurrent systems, as it is one of the most general and expressive formalisms that achieves this goal.

1.2 System Properties

Having a complex model at hand, the first task that model designers usually face is how to make sure that the model is bug-free, i. e. reflects correctly the system that one has attempted to model. And when this question is solved, how can we use the model to prove that the system behaviour is all as intended?

Both problems can be addressed by introducing a formal language to specify various properties of the system. Going back to our self-driving cars scenario, we can consider the protocol controlling cars safe, if, for example, *no two cars are ever located dangerously close to each other at a high speed*, or *under any circumstances all the cars will reach a safe stable state within a reasonable time with high probability.* In the context of concurrent systems *Continuous Stochastic Logic* (CSL) is expressive enough to describe basic dependability properties like the ones mentioned above. Its building blocks are the following properties:

– *Unbounded reachability*, that takes the value of the probability to eventually reach a certain subset of states in an MA.

– *Time-bounded reachability* describes the probability to reach a certain subset of states in an MA within a given interval of time.

Yet, CSL alone does not provide sufficient means to reason about the performance of the system, such as the costs of running the system, its energy consumption or the revenue generated by it. In the scenario with self-driving cars, one could be interested in, for example, *how much time on average a car spends in a safe state*, or *what is the expected total fuel consumption*, etc. To be able to model such properties formally, states and transitions of Markov automata are enriched with *rewards* or *costs.* A run of the MA, consisting of visited states and transitions, accumulates their rewards in such a way, that state rewards are collected proportional to the residence time in that state and transition rewards are accumulated once the transition is taken. A lot of effort has been invested [GHH$^+$14, GTH$^+$14, BWH18] into analysing this kind of properties for Markov automata. Below are some examples of the properties that have been considered:

– *Cumulative time-bounded reward*, that describes the total worst- or best-case reward expected to be accumulated over the runs of MA, until the running time reaches a certain time bound.

– *Cumulative unbounded reward* is very similar to the time-bounded reward, however, the rewards over paths are being accumulated forever. Usually, this value is only considered in models in which all paths reach with probability 1 a state that has no reward assigned to it, to avoid this value to be infinity.

– *Long-run average reward* denotes the reward that is accumulated on average at every time unit in the worst- or best-case.

– *Discounted reward* is the worst- or best-case total reward accumulated in such a way that future rewards contribute less to the total value, i. e. are *discounted.*

In this thesis, we will be looking into CSL properties as well as rewards properties for Markov automata.

1.3 Analysis

Now that we have a basic understanding of how to (i) model concurrent systems and (ii) describe their desired properties, our attention moves to ways to evaluate these properties.

The viewpoint that we take in this work is that the analysis of Markov automata should be *practical*, i. e. it needs to be efficient and lightweight in terms of memory on *typical* applications. We value this more than theoretical worst-case complexity, that forces the algorithms to be optimal also for those scenarios that are highly unlikely to appear in real life. This is the viewpoint that guides the research presented in this thesis.

As we have discussed before, Markov automata can describe each component of a concurrent system and their interaction separately. Thus the syntactic size of a concurrent system can be thought of as the sum of the description sizes of each component. We, therefore, consider the analysis to be efficient and lightweight if its runtime and memory consumption are polynomial in this number.

A candidate solution is compositional analysis [Pnu85] that evaluates each of the components of the system separately and combines the results to obtain the solution for the whole system. In probability theory, such techniques are usually referred to as the *product-form solution*. For the MA shown in Figure 1.1, this translates to analysing separately $\mathcal{M}_1$ and $\mathcal{M}_2$ and combining the results to obtain the solution for $\mathcal{M}$. This approach has polynomial complexity if each of the steps is polynomial. So far, however, such techniques were only developed for very restricted sub-classes of Markov automata and properties, see for example [BH99, HMN13].

When it is not possible to analyse a model compositionally with respect to a certain property, then the analysis is performed on the Markov automaton model of the whole system (MA $\mathcal{M}$ in Figure 1.1c). The size of this Markov automaton is in the worst case exponential in the size of the model description that we defined above. Thus, before an algorithm even starts, it gets an exponential disadvantage. Namely, if it is exhaustive and computes values for all states, then even if it runs computations in constant time on each of the states, it has to at least visit each state, which takes an exponential amount of time.

Our main motivation for this work is to advance the analysis techniques for Markov automata. Many MA properties can be analysed by solving similar problems on MDPs. Such properties, therefore, do not require dedicated MA algorithms. Model-checking MDPs has been extensively studied for many years (see, e. g. [Put94]) and constitutes a separate field of research. In this work, we will not concentrate on these problems, but rather consider only those that are specific to Markov automata.

The following is the list of MA properties that require dedicated solution techniques: Time-bounded reachability, cumulative time-bounded reward, long-run average reward, and discounted reward. While there are a few algorithms to analyse each of these properties, we believe that they are not efficient enough to be considered practical (with the exception of the discounted reward property). In the following, we discuss in detail time-bounded and long-run properties and the algorithms available for their analysis.

Time-Bounded Reachability and Reward are by far the most challenging problems in terms of Markov automata analysis. Formally, time-bounded reachability is the optimal (worst- or best-case) probability of the given MA to reach a subset of states, within a given time interval. The time-bounded reward is the optimal expected reward that can be collected in the given MA within a given time bound. The problem of time-bounded reward is closely related to time-bounded reachability, however, has not received as much attention so far. Most of the discussion below can be applied to both, although we will, for simplicity, concentrate only on the time-bounded reachability.

The time-bounded reachability problem can be subdivided into two sub-problems: (i) the optimisation problem that computes optimal choices (strategy) for the non-deterministic transitions, and (ii) the analysis of the stochastic process induced on the MA by the selected strategy. Each of these problems in isolation admits efficient solutions, it is the combination that makes the problem hard.

Problem (i) is the unbounded reachability that needs to be solved on a subclass of Markov automata with only instantaneous transitions (MDPs, or PAs). The value can be computed exactly (linear programming, policy iteration [Put94]) or approximated via value iteration [HM14, QK18, BKL$^+$17]. Problem (ii) is the computation of the *transient distribution* in Markov automata that have no non-determinism (CTMCs), and can be solved via, e. g. *uniformisation* [Jen53].

When both non-determinism and stochastic delays are present, the optimal non-deterministic option can vary depending on time. Thus, an optimal decision has to be computed *for every time point*, which belongs to the continuous domain. As an example, consider Figure 1.2: The plot shows two functions, one for each of the non-deterministic options available in state $(1, 1)$ of the MA in Figure (1.1c). Each of the functions takes the value of the probability to reach one of the states in $\{(g_1, \perp), (\perp, g_2)\}$ starting from state $(1, 1)$ within t time units, where t is the value of the x-axis. If more than roughly 1.16 seconds remain, option α has a higher probability of reaching some goal state, while option β is preferable as long as less than 1.16 seconds are left.

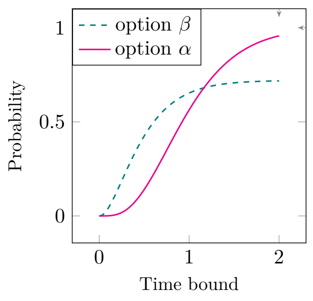


FIGURE 1.2: Reachability probability for different decisions.

Existing solutions. Classically, one deals with continuity by *discretising* the time horizon, as it is the case for most algorithms for CTMDPs and MA ([GHH$^+$14, FRSZ11, HH13, BS11, Neu10]): The time horizon is partitioned into finitely many intervals, and the value within each interval is approximated by e. g. polynomial or exponential functions. There are however also techniques that are not based on discretisation, such as the ones presented in [BHHK15, Gro18].

The Markov automaton shown in Figure 1.1c is an example of the following scenario: Optimal strategy changes based on how much time is available for the system. We conjecture that this scenario is very common and, as such, similar situations should appear often in real-life case studies. As we can see from Figure

1.2, it is enough to discretise the time horizon with roughly 2 intervals: $[0, 1.16]$ and $(1.16, 2]$. The discretisation-based algorithms for CTMDPs and MA that are known to date use from 85 to $2 \cdot 10^7$ intervals to analyse this model[1] with precision 10^{-6}. This number is important because it reflects the amount of work that the algorithm performs for a certain problem. While it may not be the only indicator of how efficient an algorithm is, it certainly affects its running time. Usually the number of intervals that an algorithm uses to discretise the time horizon depends on various parameters of the problem and one of the major contributors is the approximation error bound. Notice, however, that the number of intervals that is actually needed for our example is two, irrespectively of the given error bound.

Another technique [BHHK15, Gro18] that is not based on discretisation, may take from a few seconds up to half an hour to compute the reachability probability for various time bounds within interval $[0, 2]$ for this Markov automaton[2]. Notice that $\mathcal{M}$ has only 14 states and is nowhere close to the sizes of models that one would encounter in real life. The Quantitative Verification Benchmark Set [HKP+19] collects all the published Markov automata models and we use it as a reference for *realistic* case studies. One can see that the state-space of many models in this set goes up to millions. We conjecture that an algorithm that takes half an hour to model-check a system of 14 states is likely to take a lot of time to model-check a system six orders of magnitude larger than that.

Switching points. The reason why the discussed algorithms perform so many iterations/take so much time for this example, is that it is not known to date how to compute efficiently the time point (located somewhere close to 1.16) at which one decision becomes better than another. We will call such time points *switching points.* In order to provide formal guarantees on the computed value, the algorithms usually over-approximate the amount of switching points by considering the worst-case scenario.

The interest in switching points is both practical and theoretical. On the one hand, if we can efficiently compute this point somewhere close to 1.16, then we will know that, for the MA in Figure 1.1c, roughly two discretisation intervals will suffice to approximate the solution. This way, instead of solving the worst-case problem (which is usually the hardest), the algorithms will adapt to a given problem and may become more efficient. On the other hand, the notion of switching points has been in the air since at least the work of Miller on CTMDPs from 1968 [Mil68]. So far, however, there has been no adequate characterisation of switching points for CTMDPs or Markov automata that would enable one to somehow "see" them. Available characterisations mostly state: *If there are switching points on this time interval, then there are at least N of them*, where N is usually a very coarse over-approximation. Other characterisations require one to check whether a certain condition holds for each time point of a time interval ([Mil68, BS11]), which does not seem to be implementable.

[1]Here we used the CTMDP that has the same non-deterministic choices, exit rates and probability distribution over successor states as the MA $\mathcal{M}$.

[2]For example, the running time for time bound 2 is 0.1 second, while for time bound 1.16 it is more than half an hour.

In this thesis, we improve time-bounded reachability analysis by studying the switching points of optimal schedulers and developing an adaptive algorithm that would use the switching points of a given problem, rather than the ones of a worst-case scenario.

Huge Markov automata. So far we have been considering exhaustive algorithms for time-bounded reachability, i. e. the algorithms that perform computations on the whole state-space of a given Markov automaton. No matter how efficient an exhaustive algorithm is, there is a lower bound on its running time, which is the amount of time it takes to visit each state at least once. What if our models are so large, that they cannot even fit into memory? What if the models are infinite? Visiting each state, not to mention performing non-trivial computations for each of those states, may be extremely difficult, if possible at all. For MDPs even in these cases there are techniques to obtain approximations of various properties with guarantees on the introduced error bounds [BCC$^+$14, ACD$^+$17]. These techniques are based on ideas originally proposed in the field of probabilistic planning, such as bounded real-time dynamic programming (BRTDP) [MLG05]. The main idea is to estimate which states are actually relevant for the property under consideration, and which can be discarded so that the introduced error is not very large. These techniques proved to be efficient for many practical case studies, however, none is available so far for Markov automata.

Apart from exploring the limits of exhaustive approaches for time-bounded reachability in Markov automata, in this thesis we also consider BRTDP-style solutions. The BRTDP algorithm first samples a path in the model and then updates the values for only those states that are visited along the path. When repeating this multiple times, the algorithm discovers which states are relevant to the considered property (which may be a small subset of the total state-space) and the values that it computes eventually converge to the actual ones. This works well for discrete-time unbounded reachability, where each time a state is visited, its value is updated, or for step-bounded reachability in discrete time, when the step bound is not very large. In this case, reachability values are updated for pairs of a state and a time step. A naïve approach to develop BRTDP for time-bounded reachability is the following: Use one of the discretisation algorithms discussed above to reduce the problem from continuous to discrete time, then apply off-the-shelf BRTDP solvers for the obtained discrete-time problem. However, this approach is not practical. In order to provide bounds on the error, discretisation-based algorithms usually partition the time interval very finely. This means that the step-bound in the obtained discrete-time problem will be very large. After sampling a path, the values along the path are updated for state-time pairs. Since the discretisation is very fine, relevant state-time pairs are not visited often and therefore this naïve approach converges very slowly. In addition, this approach requires a lot of memory, since the values of many state-time pairs have to be stored.

Long-Run Average Reward takes the value of the worst- or best-case reward that can be collected on average every time unit, considering that the system will keep running forever. In Markov automata, this problem can thus far only be solved via a reduction to a linear program (LP) [GTH$^+$14]. Linear programming has polynomial worst-case complexity and there exist a lot of techniques improving the efficiency of LP solvers. Despite this fact, however, it has been observed that the running time of LP-based solutions on other Markovian models tends to be worse than that of iterative approaches, such as value iteration [Put94]. Our target for this thesis is to develop iterative techniques for analysing long-run properties for Markov automata and evaluate their performance relative to LP-based approaches.

For many continuous-time Markovian models, *infinite horizon properties*, i. e. properties that do not depend on any time bound, can be solved by a simple reduction to an instance of the same problem in discrete time. Since the actual timing of transitions does not matter for a property, then there is no need to have timed transitions. And indeed the same applies to many properties for Markov automata, for example, unbounded reachability or cumulative unbounded reward, and others. For long-run average rewards in Markov automata, however, it is not straightforward how to make this approach work. The long-run average reward problem in continuous time averages total accumulated reward with total time passed from the beginning. Long-run average reward in discrete time averages the total accumulated reward with the number of transitions performed from the beginning. Since Markov automata have two types of transitions, timed and untimed, then when averaging the total accumulated reward in the discrete case, one needs to filter out somehow untimed transitions that have no effect in the continuous case. This is the main challenge to design iterative algorithms for long-run average reward property.

1.4 Contribution

The contribution of this thesis are three novel efficient algorithms that address challenging problems for Markov automata:

– *Time-bounded reachability on full state-space.* We develop a characterisation of switching points in terms of the intersection of finitely many functions. These functions are represented by a system of differential equations. Each of these functions corresponds to a specific state s and a decision in that state α and we will denote it with $f_{s,\alpha}$. If for the current time point the best decision for a state s is action α, then the next switching point is the closest point in time when the function $f_{s,\alpha}$ intersects some other function $f_{s,\beta}$. This characterisation allows us to design an algorithm for quantifying the time-bounded reachability, which is based on approximating these intersection points.

– *Time-bounded reachability on partial state-space.* We lift the BRTDP technique from reachability in discrete-time MDPs to time-bounded reachability in continuous-time Markov automata. Recall that the classical BRTDP approach only updates values for state and time pairs that were visited along the last sampled path. Above we argued that the naïve approach to do this in continuous time is impractical due to a large number of state-time pairs that have to be stored. Instead of storing these values, our solution is to compute them from scratch at each iteration. The update step computes values for *all* states visited at some point from the beginning and for *all* the time points in the interval from 0 to the time bound. Essentially, at each iteration our algorithm solves a time-bounded reachability sub-problem for a smaller Markov automaton. Thus in our algorithm the iterations are more expensive than the iterations for the discrete-time case. However, we compensate by performing only a few of them.

– *Long-run average reward.* We develop a new iterative algorithm for approximating long-run average rewards. Recall that the main issue when solving long-run average rewards for Markov automata is that the reduction to a discrete-time problem needs to somehow ignore the untimed transitions when averaging total accumulated reward. The main idea of our approach is to avoid counting untimed transitions by essentially considering them to be a part of the preceding timed transitions. This way a Markov automaton can be seen as a CTMDP. Long-run average rewards for CTMDPs can be solved efficiently via a reduction to a discrete-time long-run average reward problem [Put94]. However, simply applying the CTMDP algorithms to the transformed MA does not work right away, since this transformed model can be exponentially larger than the original Markov automaton. We manage to avoid this issue by a dedicated treatment of exponentiality via dynamic programming.

1.5 Chapter Origins

The results of this thesis are partially based on the following published results:

– Yuliya Butkova and Gereon Fox. Optimal time-bounded reachability analysis for concurrent systems. In TACAS (2), volume 11428 of Lecture Notes in Computer Science, pages 191–208. Springer, 2019.

– Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Kretínský. Continuous time Markov decisions based on partial exploration. In ATVA, volume 11138 of Lecture Notes in Computer Science, pages 317–334. Springer, 2018.

– Yuliya Butkova, Ralf Wimmer, and Holger Hermanns. Long-run rewards for Markov automata. In TACAS (2), volume 10206 of Lecture Notes in Computer Science, pages 188–203, 2017.

– Yuliya Butkova, Holger Hermanns, and Arnd Hartmanns. A Modest approach to modelling and checking Markov automata. In QEST, volume 11785 of Lecture Notes in Computer Science, pages 52-69. Springer, 2019.

1.6 Overview

In **Chapter 2** we settle the mathematical notation used throughout the thesis, introduce Markov automata and discuss how to use them to model concurrent systems. We formally define the probability measure for Markov automata, required to analyse various properties. The chapter is concluded with the discussion of the Zeno behaviour and a brief overview of notable sub-classes of MA as well as other relevant models.

In **Chapter 3** we formally introduce various dependability and performance properties for Markov automata. We reiterate over the CSL logic, that defines important probabilistic properties and later consider properties involving rewards. In this chapter, we also give an overview of various algorithms available for the analysis of these properties.

In **Chapter 4** we study the problem of analysing the time-bounded reachability property. The chapter gives an overview of existing solutions and introduces two new approximation algorithms, both provide bounds on the error induced by the approximations. The first algorithm is based on exhaustive state-space exploration and exploits the results we have obtained on the analysis of switching points. The second algorithm performs computations only on a part of the total state-space and uses simulations together with heuristics to identify the relevant part of the state-space.

In **Chapter 5** we consider the problem of computing long-run average reward values. We reiterate over a reduction of the solution for an arbitrary MA to a solution for a subclass of MA. We develop an iterative approach to approximate long-run average reward values for this subclass of MA, up to an arbitrary given error bound.

In **Chapter 6** we conclude with an overview of the achieved results.

Foundations 2

In this chapter, we will lay the foundations for the results of this thesis. We start in Section 2.1 with a brief recap of standard mathematical concepts. The notation used throughout this work is established here. In Section 2.2 we introduce the Markov automaton model [EHZ10, DH11] that we study in this thesis. With Markov automata, one can model systems running concurrently in continuous time and that exhibit such characteristics as non-deterministic uncertainty, probabilistic uncertainty and stochastic timing. We will discuss how Markov automata can be used to model such systems (Sections 2.2.1 and 2.2.2), the semantics of Markov automata in terms of the probability measure (Section 2.2.3), rule out a subclass of models that describe unrealistic behaviour (Section 2.2.4) and conclude with a brief discussion of a few more standard models that are closely related to the analysis of Markov automata (Section 2.2.5).

2.1 Mathematical Notation and Definitions

Before diving into the interesting part of this thesis we first need to agree on the notation. For most of the mathematical concepts, we use the standard notation. Below we explicitly define those terms that are often represented in various ways or those that have multiple definitions.

Numbers and Sets. In this thesis we use the classical notation for the sets of natural numbers $\mathbb{N}$, integers $\mathbb{Z}$, rational numbers $\mathbb{Q}$ and reals $\mathbb{R}$. For $X \in \{\mathbb{Q}, \mathbb{R}, \mathbb{Z}\}$, $\trianglerighteq \in \{>, \geqslant\}$ we define $X_{\trianglerighteq 0} := \{x \in X \mid x \trianglerighteq 0\}$. For a set A we define $A^\infty := A \cup \{\infty\}$. The disjoint union of two sets A and B is denoted with $A \uplus B$ and the power set of A is denoted with 2^A. Let $I \subseteq \mathbb{R}_{\geqslant 0}$ and $z \in \mathbb{R}_{\geqslant 0}$. We define $I \ominus z := \{w - z \mid w \in I \wedge w \geqslant z\}$.

For $a, b \in \mathbb{Z}$ we denote with $a..b$ the set of all integers i, such that $i \geqslant a$ and $i \leqslant b$, if $a \leqslant b$, and an empty set otherwise. We will write $x = a..b$ to denote $x \in a..b$.

For a non-empty set A and $n \in \mathbb{Z}_{>0}$ we define $A^n := \{a_1 a_2 \cdots a_n \mid \forall i = 1..n : a_i \in A\}$, $A^+ := \cup_{n \in \mathbb{Z}_{>0}} A^n$, $A^* := \{\theta\} \cup A^+$, where θ is a designated value representing an empty sequence, and $A^\omega := \{\theta\} \cup \{a_1 a_2 \cdots a_n \cdots \mid \forall i \in \mathbb{Z}_{>0} : a_i \in A\}$.

We denote with e the Euler's number.

Functions. Let f be a possibly partial function defined on a set X that maps values from X to values from Y. We will denote this with $f : X \rightharpoonup Y$. Let $X' \subseteq X$ be the set of all elements in X for which f is defined. We call X' to be the *domain* of f and denote it with *dom*(f). For a set $A \subseteq X$ we define $f|_A = u$, where $u : X \rightharpoonup [0,1]$ is a possibly partial function, such that for all $x \in A$, where x is in the domain of f, $u(x) = f(x)$ and otherwise $u(x)$ is undefined.

Let A be a set. We denote with $\mathbb{1}_A(x)$ the *indicator function*, such that $\mathbb{1}_A(x) = 1$ iff $x \in A$ and 0 otherwise.

Measures and Probabilities. We assume that the reader is familiar with the basic notions of probability theory. In this section, we briefly introduce some of those concepts, for a rigorous mathematical overview we refer the reader to any textbook on probability theory, e. g. [Ros06].

Given a finite or countable set S, a *probability distribution* μ over S is a function $\mu : S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We denote the set of all probability distributions over S by $\mathrm{Dist}(S)$. We set $\mu(S') := \sum_{s \in S'} \mu(s)$ for $S' \subseteq S$. We define a *Dirac distribution* $\Delta_S(s)$ to be a distribution over set S that assigns probability 1 to element $s \in S$ and probability 0 to other elements.

Let μ be a distribution over $S = \{s_0, \cdots, s_n\}$, such that $\mu(s_i) = p_i$, for $i = 0..n$. For convenience we often denote distribution μ with $[s_0 \to p_0, \ldots, s_n \to p_n]$, or with $[s_i \to p_i \mid s_i \in S]$.

We will denote a *probability space (or probability triple)* as a tuple $(\Omega, \mathcal{F}, \mathrm{Pr})$, where Ω is a sample space, $\mathcal{F} \subseteq 2^\Omega$ is a *sigma-algebra* over Ω, its elements are often referred to as *events*, and $\mathrm{Pr} : \mathcal{F} \to [0,1]$ is a *probability measure* over the events of $\mathcal{F}$.

A *random variable* X is a function defined over a probability space $(\Omega, \mathcal{F}, \mathrm{Pr})$, such that $X : \Omega \to \mathbb{R}$. For $E \in \mathcal{F}$ value $\mathrm{Pr}[E]$ is the probability that X takes one of the values from event E. Given a random variable X we denote with $\mathbb{E}[X]$ its expected value (or expectation).

For a probability space $(\Omega, \mathcal{F}, \mathrm{Pr})$, $E \in \mathcal{F}$ and a function $f : \Omega \to \mathbb{R}^\infty$ we will denote with $\int_E f(x) \cdot \mathrm{Pr}[\,\mathrm{d}x]$ its Lebesgue integral, if it exists.

2.2 Markov Automata

In this section we will discuss the Markov automaton model, that was first described in [EHZ10]. We start by introducing the formal definition of the model as well as some related concepts.

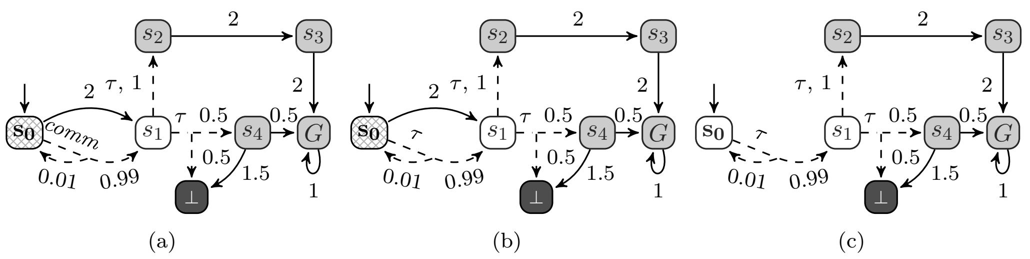


FIGURE 2.1: Example of a Markov automaton (2.1a), closed MA obtained from 2.1a (2.1b) and MA restricted to having no hybrid states (2.1c).

Definition 2.2.1. *A* Markov automaton (MA) $\mathcal{M}$ *is a tuple* $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$ *such that*

- *S is a finite non-empty set of states;*
- *$Act = Act_{\setminus\{\tau\}} \uplus \{\tau\}$ is a finite set of actions;*
- *$\dashrightarrow \subseteq S \times Act \times \mathrm{Dist}(S)$ is a finite* probabilistic transition relation*;*
- *$\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ is* Markovian transition matrix*;*

Example 2.2.1. *Figure (2.1a) shows an example MA. Here $\perp, G$ and $s_0, \ldots, s_4$ are states. Different styles used to depict states refer to edges emanating the states. This will be discussed in detail later in this section. Labels τ and comm are actions.*

We call elements of the set $\dashrightarrow$ *probabilistic transitions* or *hops*. They are shown as dashed edges marked with an action. The distribution emanating the edge is the distribution assigned to this transition. For example, transition $(s_0, comm, \mu)$ for state s_0, action *comm* and $\mu = [s_0 \to 0.01, s_1 \to 0.99]$ is in the probabilistic transition relation $\dashrightarrow$, or $(s_0, comm, \mu) \in \dashrightarrow$. In the following we often denote (s, α, μ) by $s \overset{\alpha}{\dashrightarrow} \mu$. The set $Act(s) := \{\alpha \in Act \mid \exists \mu \in \mathrm{Dist}(S) : (s, \alpha, \mu) \in \dashrightarrow\}$ is the set of *enabled actions* in state s. If for each state s and for each enabled action $\alpha \in Act(s)$ there exists at most one probabilistic transition $s \overset{\alpha}{\dashrightarrow} \mu$, then every probabilistic transition can be uniquely identified by a pair (s, α). In this case we can define a matrix $\mathbb{P}[s, \alpha, \cdot] \in \mathrm{Dist}(S)$ as follows: $\forall s' \in S : \mathbb{P}[s, \alpha, s'] := \mu(s')$.

Strictly positive entries in matrix $\mathbf{R}$ are called *Markovian transitions*. They are depicted with solid edges. For example, for state s_4 records $\mathbf{R}[s_4, \perp] = 1.5$ and $\mathbf{R}[s_4, G] = 0.5$ are Markovian transitions. The value $\mathbf{R}[s, s']$ is called the *(transition) rate* from s to s'. The *exit rate* of a state s is $E(s) := \sum_{s' \in S} \mathbf{R}[s, s']$ and $\mathbf{E}_{\max} := \max_{s \in S} E(s)$. For states that satisfy $E(s) > 0$, we define the discrete branching probability distribution $\mathbb{P}[s, \cdot] \in \mathrm{Dist}(S)$ by $\mathbb{P}[s, s'] := \mathbf{R}[s, s'] / E(s)$. The value $\mathbb{P}[s, \cdot]$ is undefined in case $E(s) = 0$.

For symmetry, we define a *Markovian transition relation* $\longrightarrow \subseteq S \times \mathbb{R}_{>0} \times \mathrm{Dist}(S)$ such that $(s, E(s), \mu) \in \longrightarrow$ iff $E(s) > 0$ and $\mu(s') = \mathbf{R}[s, s'] / E(s)$. Notice that relation $\longrightarrow$ is just a different way to represent matrix $\mathbf{R}$. We therefore call elements

$(s, \lambda, \mu) \in \longrightarrow$ also *Markovian transitions* and often denote them as $s \xrightarrow{\lambda} \mu$. Notice that each state s has at most one Markovian transition $s \xrightarrow{\lambda} \mu$.

We will use the word *transition* to refer to either a Markovian or a probabilistic transition of a Markov automaton. The set of all transitions is defined as $\rightsquigarrow := \dashrightarrow \uplus \longrightarrow$. Similarly $s \overset{\nu}{\rightsquigarrow} \mu$ is simply a more convenient way of representing (s, ν, μ).

We say that a transition $s \overset{\nu}{\rightsquigarrow} \mu$, where $\mu = [s_0 \to p_0, \ldots, s_n \to p_n]$, is an *outgoing transition of* s, and an *incoming transition of* s_i, for each $i = 0..n$. We will call states s_i the *successors of* s *via transition* (s, ν, μ) and denote this set with $post(s, (s, \nu, \mu)) = \{s_0, \ldots, s_n\}$. The set of all successors of s via all of its outgoing transitions is denoted with $post(s) := \cup_{(s,\nu,\mu)\in\rightsquigarrow} post(s, (s, \nu, \mu))$. For a subset of states $S' \subseteq S : post(S') = \cup_{s\in S'} post(s)$.

Classification of States. A state s is called *probabilistic*, if it has at least one outgoing transition and all its outgoing transitions are probabilistic, or formally $Act(s) \neq \emptyset$ and $E(s) = 0$. We denote the set of all probabilistic states with *PS*.

Similarly, a state s is called *Markovian*, if $E(s) > 0$ and $Act(s) = \emptyset$. The set of all Markovian states is denoted with *MS*.

A state s that has both probabilistic and Markovian outgoing transitions is called *hybrid*: $E(s) > 0$ and $Act(s) \neq \emptyset$. We refer to the set of all hybrid states with *HS*.

A state s that has no outgoing transitions is called a *deadlock state*: $E(s) = 0$ and $Act(s) = \emptyset$.

Finally, a Markovian state that has only self-loop transitions is called *absorbing*.

In this work, probabilistic states will always be depicted with white, Markovian with grey and deadlock states with dark grey colours. We depict hybrid states with cross-hatching. For example, in Figures (2.1a) and (2.1b) states s_2, s_3, s_4 and G are Markovian. State s_1 is probabilistic, s_0 is hybrid and state $\perp$ is a deadlock.

Remark 2.2.1. *All the notions defined above usually have a specific Markov automaton in mind, for example, the set PS is the set of probabilistic states of a given Markov automaton. Sometimes we will use sub-/superscripts to clarify which exactly Markov automaton is meant. For the set of probabilistic states, for example, we will write $PS_{\mathcal{M}}$ to denote that this is the set of probabilistic states of a Markov automaton $\mathcal{M}$. If no sub-/superscript is indicated, this usually means that the model under consideration is clear from the context. This is applied to all the definitions of this thesis, including those that have not been introduced yet.*

2.2.1 Modelling with Markov Automata

Markov automata can be cast into a compositional formalism that is expressive enough to describe in a modular way systems evolving in continuous time. This means that the model of a large system can be obtained by considering this system to be a collection of interacting sub-components. Each sub-system can be modelled as a separate Markov automaton, evolving in time independently of other sub-components. Interaction between different sub-systems is arranged by means of synchronisation with the help of probabilistic transitions.

In order to formally define the composition of Markov automata, we will work with a more convenient alternative definition of MA. When it comes to the analysis of various properties the two definitions are equivalent.

Definition 2.2.2. *A* Markov automaton (MA) $\mathcal{M}$ *is a tuple* $\mathcal{M} = (S, Act, \dashrightarrow, \longrightarrow)$ *such that*

- S *is a finite non-empty set of states;*
- $Act = Act_{\setminus\{\tau\}} \uplus \{\tau\}$ *is a finite set of actions;*
- $\dashrightarrow \subseteq S \times Act \times \mathrm{Dist}(S)$ *is a finite* probabilistic transition relation*;*
- $\longrightarrow \subseteq S \times \mathbb{R}_{\geqslant 0} \times S$ *is a finite* Markovian transition relation*;*

This definition differs from Definition 2.2.1 only in the way Markovian transitions are defined. Markovian transitions now form a relation, rather than a matrix, which means that between two states s and s' there may be multiple transitions (s, λ, s') and (s, ν, s'), where $\lambda \neq \nu$. We can now define $\mathbf{R}[s, s'] = \sum_{(s,\lambda,s')\in\longrightarrow} \lambda$. This way we can obtain a Markov automaton according to Definition 2.2.1, given a Markov automaton in this new form.

For $\mu_1 \in \mathrm{Dist}(S_1)$ and $\mu_2 \in \mathrm{Dist}(S_2)$ we define a distribution $\mu_1 \circ \mu_2$ as the joint distribution of μ_1 and μ_2, i. e. for $s_1 \in S_1, s_2 \in S_2 : \mu_1 \circ \mu_2(s_1, s_2) = \mu_1(s_1) \cdot \mu_2(s_2)$.

We are now in a position to define the composition of two Markov automata:

Definition 2.2.3 ([EHZ10])**.** *For MA* $\mathcal{M}_1 = (S_1, Act_1, \dashrightarrow_1, \longrightarrow_1)$ *and* $\mathcal{M}_2 = (S_2, Act_2, \dashrightarrow_2, \longrightarrow_2)$ *and a* synchronisation alphabet $A \subseteq (Act_1 \cap Act_2) \setminus \{\tau\}$*, the* parallel composition $\mathcal{M}_1 \|_A \mathcal{M}_2$ *is the MA* $\mathcal{M} = (S, Act, \dashrightarrow, \longrightarrow)$ *where* $S = S_1 \times S_2$*,* $Act = Act_1 \cup Act_2$ *and* $\longrightarrow, \dashrightarrow$ *satisfy the following conditions:*

- $((s_1, s_2), \lambda, (s_1', s_2')) \in \longrightarrow$ *iff either*
  - *for* $i \in \{1, 2\} : s_i = s_i', \mathbf{R}[s_i, s_i'] > 0$ *and* $\lambda = \mathbf{R}[s_1, s_1'] + \mathbf{R}[s_2, s_2']$*, otherwise*
  - $\lambda = \mathbf{R}[s_1, s_1']$ *and* $s_2 = s_2'$ *or* $\lambda = \mathbf{R}[s_2, s_2']$ *and* $s_1 = s_1'$*;*
- $((s_1, s_2), \alpha, \mu_1 \circ \mu_2) \in \dashrightarrow$ *iff either:*
  - $\alpha \in A$ *and for each* $i \in \{1, 2\} : (s_i, \alpha, \mu_i) \in \dashrightarrow_i$*, or*
  - $\alpha \in Act \setminus A$ *and* $(s_1, \alpha, \mu_1) \in \dashrightarrow_1$*,* $\mu_2 = \Delta_{S_2}(s_2)$ *or* $(s_2, \alpha, \mu_2) \in \dashrightarrow_2$*,* $\mu_1 = \Delta_{S_1}(s_1)$

As an example, consider Markov automata $\mathcal{M}_1$ and $\mathcal{M}_2$ depicted in Figures (1.1a) and (1.1b). Their parallel composition $\mathcal{M}_1 \|_{\{\alpha,\beta\}} \mathcal{M}_2$ is depicted in Figure (1.1c).

The parallel composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is a Markov automaton that encompasses the behaviour of both $\mathcal{M}_1$ and $\mathcal{M}_2$. They can either perform a step *synchronously* or one after another. Probabilistic transitions labelled with actions from the synchronization alphabet are taken synchronously, while all the other transitions interleave.

Notice that the synchronisation alphabet cannot include action τ. Transitions labelled with action τ have special meaning. They are considered to be only visible to the MA itself, cannot be synchronised with and can be executed without any constraints. We will refer to action τ and transitions labelled with τ as *internal* and to actions from the set $Act_{\setminus\{\tau\}}$ and transitions labelled with these actions as *external*. Since a Markov automaton does not interact with other models through its internal probabilistic transitions, they are assumed to be taken *instantaneously*, i.e. with no time delay. This assumption arises from the process algebraic notion of maximal progress, a useful concept for modelling concurrent systems [NS91, Yi91].

In order to prohibit further interaction of an MA with other MA, one can *hide* its external actions thus making them internal. This is achieved via the *hiding operator*:

Definition 2.2.4. *For an MA $\mathcal{M} = (S, Act, \dashrightarrow, \longrightarrow)$ and a set $A \subseteq Act_{\setminus\{\tau\}}$ the Markov automaton $\mathcal{M}\setminus A$ is defined as follows $\mathcal{M}\setminus A = (S, Act', \dashrightarrow', \longrightarrow)$ where $Act' = (Act \setminus A) \cup \{\tau\}$, and $\dashrightarrow'$ is the smallest relation satisfying the following: If $(s, \alpha, \mu) \in \dashrightarrow$ and $\alpha \in A$, then $(s, \tau, \mu) \in \dashrightarrow'$, if $(s, \alpha, \mu) \in \dashrightarrow$ and $\alpha \notin A$, then $(s, \alpha, \mu) \in \dashrightarrow'$.*

As an example, consider the MA depicted in Figure (2.1a). Hiding the only external transition of this Markov automaton (labelled with action *comm*) results in the Markov automaton depicted in Figure (2.1b).

We will call Markov automata that have no external actions *closed*. We will only perform a formal analysis of systems on their complete Markov automata models. Therefore from now on we will consider only closed Markov automata. This means that no external communication is possible and therefore all the transitions in the MA are either Markovian or internal.

2.2.2 Operational Behaviour of Closed Markov Automata

Markov automata are operating in continuous time. The process starts in an initial state s_0, sampled from some initial distribution that is defined by the modeller. Upon entering a state $s \in S$ the behaviour of the MA depends on which transitions are available in this state if any. Each of the outgoing transitions $\gamma \in \rightsquigarrow$ of s is associated with a variable $X_\gamma \in \mathbb{R}_{\geqslant 0}$. The value of this variable is the time delay incurred by taking transition γ. It is defined based on the type of transition:

– If γ is an internal probabilistic transition, then X_γ is a constant 0. This is due to the maximal progress assumption implying that internal probabilistic transitions happen instantaneously.

– For a Markovian transition $\gamma = s \xrightarrow{\lambda} \mu$, X_γ is a continuous random variable. This random variable has an exponential distribution with parameter λ, i. e. its probability density function is $f(t) = \lambda \cdot e^{-\lambda \cdot t}$ for $t \geqslant 0$ and $f(t) = 0$ for $t < 0$. This means that the variable takes values in the domain $\mathbb{R}$ and for $a, b \in \mathbb{R}_{\geqslant 0}$ the value of X_γ lies within $[a, b]$ with probability $e^{-\lambda a} - e^{-\lambda b}$.

Each time a Markov automaton enters a state that has an outgoing Markovian transition γ, the random variable X_γ samples a new value.

Recall that states of a closed Markov automaton can have at most one outgoing Markovian transition and any finite amount of internal probabilistic transitions. When entering a state that has multiple transitions $\gamma_1, \ldots, \gamma_n \in \rightsquigarrow$, Markov automaton resides in s as long as the residence time t in s reaches one of the values $X_{\gamma_i}, i = 1..n$, or $t = X_{\min}$, where $X_{\min} = \min\{X_{\gamma_1}, \ldots, X_{\gamma_n}\}$. There are several cases to consider:

Singleton transition. If there is only one transition $\gamma_i = s \overset{\nu_i}{\rightsquigarrow} \mu_i$, such that $X_{\gamma_i} = X_{\min}$, then the system resides in s for X_{γ_i} time units and then proceeds to a successor state s' selected at random with probability $\mu_i(s')$.

Multiple probabilistic transitions. If there are multiple transitions γ, such that $X_\gamma = X_{\min}$ and they all are probabilistic, then the decision has to be taken as to which of the transitions to choose. Usually, tie-breaking is performed by a *scheduler*, a function that observes the history of the system evolution from the beginning and chooses a single probabilistic transition out of all available. We describe schedulers formally later in this chapter. For now we assume that only one probabilistic transition $\gamma_i = s \overset{\nu_i}{\rightsquigarrow} \mu_i$ is selected out of multiple and similarly to the previous case the system proceeds to a successor state s' with probability $\mu_i(s')$.

Multiple probabilistic and Markovian transitions. Next we consider the case when there exists at least one probabilistic transition γ_p and at least one Markovian transition γ_m, such that $X_{\gamma_p} = X_{\gamma_m} = X_{\min}$. Since X_{γ_p} is always 0, this means that $X_{\gamma_m} = 0$. For the general analysis of Markov automata, it is important to define precisely how the system behaves in such a situation. In this work, however, we are only interested in probabilistic properties. The probability that random variable X_{γ_m} takes value 0 is $e^{-E(s)\cdot 0} - e^{-E(s)\cdot 0} = 0$. Therefore for this work, it does not matter what exactly the Markov automaton does if there is a tie between a Markovian and a probabilistic transition. The precise semantics will not have any impact on the properties under consideration. We deliberately leave this case undefined.

No outgoing transitions. So far we have defined residence time in a state depending on the outgoing transitions available in this state. Deadlock states, however, have no outgoing transitions and therefore it is unclear how to define residence time in them. Moreover, deadlock states prevent Markov automata from performing infinitely many transitions, making it possible for different runs of an MA to result in either finitely many transitions that end in a deadlock state or infinitely many transitions. This makes it unnecessarily tedious and cumbersome to define a probability measure for Markov automata. Therefore in this work, we only consider Markov automata that have no deadlock states.

We have thus defined the behaviour of Markov automaton upon entering a state s. When the system has resided in s for some time and has selected a successor (as described above), it enters the successor state and the whole process proceeds in the same way ad infinitum.

2.2.3 Probability Measure

In this section, we introduce a probability space for closed Markov automata without deadlock states. All the notions and results presented here originate from [Hat17].

Notice that in closed Markov automata all the probabilistic transitions are labelled with the same action τ. This means that one cannot uniquely identify probabilistic transitions emanating the same state s by using only actions of the transitions. In order to resolve this issue one can use in place of actions pairs (τ, μ), where $\mu \in \mathrm{Dist}(S)$. Namely, instead of the original set of actions $Act = \{\tau\}$ one can use set $Act' = \{\tau\} \times \mathrm{Dist}(S)$. An action $\alpha = (\tau, \mu) \in Act'$ is enabled in a state s if transition $s \overset{\tau}{\dashrightarrow} \mu$ was present in the original Markov automaton. This is only a syntactic change, the semantics of the transitions and their actions remains intact in a sense that the transitions and the actions remain internal. We will denote this newly obtained set of actions as before with Act.

The probability space for Markov automata will be constructed iteratively by first defining the probability space over single transitions, then extending it to the probability space over finitely many transitions and finally obtaining the space for infinite runs.

Steps, Paths and Fragments

We start with defining the sample spaces of the probability triples.

A *step* σ is a pair $\sigma = (\tau, t) \in (\dashrightarrow \times \{0\}) \uplus (\longrightarrow \times \mathbb{R}_{\geqslant 0})$, where τ is a probabilistic or Markovian transition, and t is a non-negative real number representing the delay required to execute the transition. The set of all steps in an MA $\mathcal{M}$ is denoted with $\mathit{Steps}(\mathcal{M}) := (\dashrightarrow \times \{0\}) \uplus (\longrightarrow \times \mathbb{R}_{\geqslant 0})$.

An *infinite path* $\rho = \sigma_0 \cdots \sigma_n \cdots$ is an infinite sequence of steps $\sigma_i \in \mathit{Steps}(\mathcal{M})$ for $i \in \mathbb{Z}_{\geqslant 0}$. The set of all infinite paths is denoted with $\mathit{Paths}^{\omega}(\mathcal{M})$.

A *(path) fragment* $\phi = \sigma_0 \cdots \sigma_{n-1}$ is finite sequence of steps $\sigma_i \in \mathit{Steps}(\mathcal{M})$ for $i = 0..n-1$. Here $|\phi| := n$ is the *length* of the path fragment. We define a special *empty* fragment θ to be the path fragment of length 0. The set of all fragments of length $n \geqslant 0$ is denoted with $\mathit{Frags}^n(\mathcal{M})$. The set of all fragments is defined as $\mathit{Frags}^*(\mathcal{M}) := \cup_{n \in \mathbb{Z}_{\geqslant 0}} \mathit{Frags}^n(\mathcal{M})$.

A *finite path* $\rho = \phi \cdot s$, where $\phi \in \mathit{Frags}^*(\mathcal{M})$ and $s \in S$ is a finite sequence of steps with a selected final state. Here the length $|\rho|$ of a path $\rho = \phi \cdot s$ is defined as the length of the respective path fragment, i. e. $|\rho| := |\phi|$. We will denote paths of the form $\theta \cdot s$ simply with s, i. e. we omit θ. The set of all finite paths of length $n \geqslant 0$ is denoted with $\mathit{Paths}^n(\mathcal{M})$ and the set of all finite paths is defined as $\mathit{Paths}^*(\mathcal{M}) := \cup_{n \in \mathbb{Z}_{\geqslant 0}} \mathit{Paths}^n(\mathcal{M})$. For a finite path $\rho = \sigma_0 \sigma_1 \cdots \sigma_{n-1} \cdot s$ we define $\rho{\downarrow} := s$ to be the last state on the path.

Whenever the MA under consideration is clear from the context we will denote the sets $\mathit{Steps}(\mathcal{M})$, $\mathit{Paths}^{\omega}(\mathcal{M})$, $\mathit{Frags}^n(\mathcal{M})$, $\mathit{Frags}^*(\mathcal{M})$, $\mathit{Paths}^n(\mathcal{M})$ and $\mathit{Paths}^*(\mathcal{M})$ with Steps, Paths^{ω}, Frags^n, Frags^*, Paths^n and Paths^* respectively.

Let $\rho = (s_0 \overset{\nu_0}{\rightsquigarrow} \mu_0, t_0) \cdots (s_k \overset{\nu_k}{\rightsquigarrow} \mu_k, t_k) \cdots$ be an infinite path or a path fragment. For convenience we will represent ρ as

$$\rho = s_0 \xrightarrow{\nu_0, t_0} s_1 \xrightarrow{\nu_1, t_1} \cdots \xrightarrow{\nu_{k-1}, t_{k-1}} s_k \xrightarrow{\nu_k, t_k} \cdots$$

And, analogously, a finite path $\rho = (s_0 \overset{\nu_0}{\rightsquigarrow} \mu_0, t_0) \cdots (s_k \overset{\nu_k}{\rightsquigarrow} \mu_k, t_k) \cdot s_{k+1} \cdots$ can be represented as

$$\rho = s_0 \xrightarrow{\nu_0, t_0} s_1 \xrightarrow{\nu_1, t_1} \cdots \xrightarrow{\nu_{k-1}, t_{k-1}} s_k \xrightarrow{\nu_k, t_k} s_{k+1}$$

We define the concatenation operation $\circ$ for paths and fragments in an obvious way. If a path fragment $\phi = \sigma_0 \cdots \sigma_n$ is concatenated with another path fragment $\phi' = \sigma'_0 \cdots \sigma'_k$, finite path $\rho = \sigma'_0 \cdots \sigma'_k \cdot s$ or infinite path $\rho_\infty = \sigma'_0 \cdots \sigma'_k \cdots$, the result is another path fragment $\phi \circ \phi' = \sigma_0 \cdots \sigma_n \sigma'_0 \cdots \sigma'_k$, finite path $\phi \circ \rho = \sigma_0 \cdots \sigma_n \sigma'_0 \cdots \sigma'_k \cdot s$ or infinite path $\phi \circ \rho_\infty = \sigma_0 \cdots \sigma_n \sigma'_0 \cdots \sigma'_k \cdots$. The result of $\theta \circ f$ will be simply denoted with f, for $f \in \mathit{Frags}^* \cup \mathit{Paths}^* \cup \mathit{Paths}^\omega$.

Next, we define total time over a path. For an infinite path $\rho = s_0 \xrightarrow{\nu_0, t_0} \cdots \xrightarrow{\nu_{k-1}, t_{k-1}} s_k \xrightarrow{\nu_k, t_k} \cdots$ we define total time spent after performing n transitions as follows: $\tau_{\text{total}}(\rho, 0) := 0$ and for $n \in \mathbb{Z}_{>0} : \tau_{\text{total}}(\rho, n) := \sum_{i=0}^{n-1} t_i$.

For a finite path $\rho = s_0 \xrightarrow{\nu_0, t_0} \cdots \xrightarrow{\nu_{k-1}, t_{k-1}} s_k$, and $n \leqslant k, \tau_{\text{total}}(\rho, n)$ is defined analogously. In addition we define $\tau_{\text{total}}(\rho) := \sum_{i=0}^{k-1} t_i$.

Let $\rho = (s_0 \overset{\nu_0}{\rightsquigarrow} \mu_0, t_0) \cdots (s_k \overset{\nu_k}{\rightsquigarrow} \mu_k, t_k) \cdots$ be a finite or infinite path. For a quick access to states visited over a path we define $\rho[i] := s_i$ if ρ is infinite and $i \in \mathbb{Z}_{\geqslant 0}$, or if ρ is finite, then we require that $i \leqslant |\rho|$. Similarly the action taken at i-th transition will be denoted with $\rho_{Act}[i] := \nu_i$ and time with $t[\rho, i] := t_i$, with the same restrictions on the value of i as before.

The set of states that a finite or infinite path ρ visits at time t is denoted with $\rho@t$ and defined as follows:

$$\rho@t := \bigcup_{i \in \mathbb{Z}_{\geqslant 0}} \Big(S_{=t}(i) \cup S_{MS}(i, t) \Big),$$

where

$$S_{=t}(i) = \begin{cases} \{\rho[i]\} & \text{if } \tau_{\text{total}}(\rho, i) = t \\ \emptyset & \text{otherwise} \end{cases}$$

$$S_{MS}(i, t) = \begin{cases} \{\rho[i]\} & \text{if } \rho[i] \in MS \text{ and } \tau_{\text{total}}(\rho, i) < t, \tau_{\text{total}}(\rho, i+1) > t \\ \emptyset & \text{otherwise} \end{cases}$$

Example 2.2.2. *Consider the following infinite path of the MA depicted in Figure 2.1c:*

$$\rho = s_0 \xrightarrow{\tau, 0} s_1 \xrightarrow{\tau, 0} s_4 \xrightarrow{\perp, 1.1} G \xrightarrow{\perp, 0.7} G \xrightarrow{\perp, 1.3} \cdots$$

At time 0 *the Markov automaton passes through states* s_0, s_1 *and enters state* s_4*, therefore* $\tau_{\text{total}}(\rho, 0) = \tau_{\text{total}}(\rho, 1) = \tau_{\text{total}}(\rho, 2) = 0$ *and* $S_{=0}(0) = \{s_0\}, S_{=0}(1) = \{s_1\}, S_{=0}(2) = \{s_4\}$ *and from now on* $\forall i \in \mathbb{Z}_{\geqslant 0}, i > 2 : S_{=0}(i) = \emptyset$*. Since no Markovian state satisfies the conditions of* $S_{MS}(i, 0)$*, then* $\forall i \in \mathbb{Z}_{\geqslant 0} : S_{MS}(i, 0) = \emptyset$*. Thus* $\rho@0 = \{s_0, s_1, s_4\}$*.*

At time point $t = 1.1$ *the system enters state* G *and therefore* $\rho@1.1 = \{G\}$*, which is reflected in only the set* $S_{=1.1}(3) = \{G\}$ *being non-empty. When* $t = 1.2$*, the MA is residing in state* G *and only set* $S_{MS}(3, 1.2) = \{G\}$ *is non-empty. Therefore* $\rho@1.2 = \{G\}$*.*

Measurable Spaces

We can now proceed to define measurable spaces over the set of steps *Steps*, the set Frags^n of fragments of length n, the set Paths^n of paths of length n and the set of infinite paths Paths^ω.

Steps. Recall that the set of steps is a disjoint union of sets $\longrightarrow \times \mathbb{R}_{\geqslant 0}$ and $\dashrightarrow \times \{0\}$. Since $\dashrightarrow$ is finite, the σ-algebra over $\dashrightarrow \times \{0\}$ is defined as $2^{\dashrightarrow \times \{0\}}$. Similarly, for Markovian transition relation the σ-algebra is $2^{\longrightarrow}$. For the set $\mathbb{R}_{\geqslant 0}$ the Borel σ-algebra is used, the smallest σ-algebra generated by open intervals over non-negative reals, that is denoted with $\mathcal{B}(\mathbb{R}_{\geqslant 0})$. Given two measurable spaces $(\longrightarrow, 2^{\longrightarrow})$ and $(\mathbb{R}_{\geqslant 0}, \mathcal{B}(\mathbb{R}_{\geqslant 0}))$, the σ-algebra for the set $\longrightarrow \times \mathbb{R}_{\geqslant 0}$ is defined as the product σ-algebra of $(\longrightarrow, 2^{\longrightarrow})$ and $(\mathbb{R}_{\geqslant 0}, \mathcal{B}(\mathbb{R}_{\geqslant 0}))$, i. e. the smallest σ-algebra that contains all sets $A \times B$, where $A \in 2^{\longrightarrow}$, $B \in \mathcal{B}(\mathbb{R}_{\geqslant 0})$. We denote this σ-algebra with $2^{\longrightarrow} \otimes \mathcal{B}(\mathbb{R}_{\geqslant 0})$. We are now in position to define the σ-algebra over the set of steps as $\mathcal{S} = \{A \cup B \mid A \in 2^{\longrightarrow} \otimes \mathcal{B}(\mathbb{R}_{\geqslant 0}), B \in 2^{\dashrightarrow \times \{0\}}\}$.

Fragments. The σ-algebra over the set of fragments and finite paths of length n is defined once again via the product σ-algebra. First, for $n = 0$ we define $\mathcal{F}^0 := \{\{\theta\}, \emptyset\}$, and for $n \geqslant 1 : \mathcal{F}^n := \otimes_{i=1}^n \mathcal{S}$.

Finite paths. The σ-algebra over the set of paths of length n, for $n \geqslant 1$ is defined as $\mathcal{P}^n := \mathcal{F}^n \otimes 2^S$.

Infinite paths. The σ-algebra over the set of infinite paths is constructed over the set of cylinders. Let P_n be a set of finite paths of length n. For $n \geqslant 0$ the *cylinder with base* P_n is defined as

$$\begin{aligned} \mathit{Cyl}_{\mathcal{M}}(P_n) := \{\sigma_0 \cdots \sigma_{n-1} \cdots \in \mathit{Paths}^\omega(\mathcal{M}) \mid & \forall i \in \mathbb{Z}_{\geqslant 0} : \sigma_i \in \mathit{Steps} \text{ and} \\ & \exists \rho \in P_n : \rho = \sigma_0 \cdots \sigma_{n-1} s_n\} \end{aligned}$$

Whenever $\mathcal{M}$ is clear from the context, we will just write $\mathit{Cyl}(P_n)$. A cylinder $\mathit{Cyl}(P_n)$ is called *measurable* if its base is measurable, i. e. $P_n \in \mathcal{P}^n$. The σ-algebra $\mathcal{P}^\omega$ over the set of infinite paths is defined as the smallest σ-algebra generated by the set of measurable cylinders $\bigcup_{i \in \mathbb{Z}_{\geqslant 0}} \{\mathit{Cyl}(P_i) \mid P_i \in \mathcal{P}^i\}$.

We have thus constructed measurable spaces over the set of steps $(\mathit{Steps}, \mathcal{S})$, fragments of length n $(\mathit{Frags}^n, \mathcal{F}^n)$, finite paths of length n $(\mathit{Paths}^n, \mathcal{P}^n)$ and infinite paths $(\mathit{Paths}^\omega, \mathcal{P}^\omega)$.

Measurable Schedulers

In the presence of multiple probabilistic transitions emanating the same state, the behaviour of a Markov automaton is not a stochastic process and therefore no probability measure can be defined. In order to resolve this issue, we need to be able to choose only one probabilistic transition out of all those that are available in the same state. This can be achieved with a concept of a scheduler.

Definition 2.2.5. *A* scheduler *(or* strategy, policy*) is a function* $\pi : \mathit{Paths}^* \to \mathrm{Dist}(\mathit{Act})$*, where for* $\rho \in \mathit{Paths}^*$ *if* $\pi(\rho)(\alpha) > 0$ *then* $\alpha \in \mathit{Act}(\rho\downarrow)$*.*

Schedulers observe the history of the system evolution from the beginning in a form of a finite path and provide a distribution over actions enabled in the final state of the path. In order to be able to define the probability measure we need to restrict ourselves to a subset of schedulers that are *measurable*:

Definition 2.2.6. *A scheduler* $\pi : \mathit{Paths}^* \to \mathrm{Dist}(\mathit{Act})$ *is called* generic measurable *iff for every* $\alpha \in \mathit{Act}, B \in \mathcal{B}([0,1]), n \in \mathbb{Z}_{\geqslant 0} : \{\rho \in \mathit{Paths}^n | \pi(\rho)(\alpha) \in B\} \in \mathcal{P}^n$*. The set of all generic measurable schedulers for an MA* $\mathcal{M}$ *is denoted with* $\Pi_\mu^{\mathcal{M}}$*.*

We will distinguish a certain subclass of generic measurable schedulers, called *stationary schedulers:*

Definition 2.2.7. *A scheduler* π *is called* stationary*, if* $\forall \rho_1, \rho_2 \in \mathit{Paths}^*$*, s. t.* $\rho_1\downarrow = \rho_2\downarrow$ *it holds that* $\pi(\rho_1) = \pi(\rho_2) = \Delta_{\mathit{Act}}(\alpha)$*, for some* $\alpha \in \mathit{Act}(\rho_1\downarrow)$*. The set of all stationary schedulers is denoted by* $\Pi_{\mathtt{stat}}^{\mathcal{M}}$*.*

Since stationary schedulers can be equivalently defined as $\pi : \mathit{PS} \to \mathit{Act}$, we will often use this simplified definition.

Probability Measure

We will define a probability measure on the measurable space $(\mathit{Paths}^\omega, \mathcal{P}^\omega)$ iteratively. We start with probability measure over steps and then extend it to probability measure over fragments, finite paths and infinite paths.

Steps. Let $\pi \in \Pi_\mu$ be a generic measurable scheduler and $\phi = (\tau_0, t_0) \cdots (\tau_n, t_n)$ is a path fragment, such that $\tau_n = s_n \overset{\nu_n}{\rightsquigarrow} \mu_n$. We define the probability measure on measurable space $(\mathit{Steps}, \mathcal{S})$ denoted with $\boldsymbol{\mu}_{\pi,\phi} : \mathcal{S} \to [0,1]$, as follows: $\boldsymbol{\mu}_{\pi,\phi}(\emptyset) := 0, \boldsymbol{\mu}_{\pi,\phi}(\mathit{Steps}) := 1$ and otherwise:

$$\boldsymbol{\mu}_{\pi,\phi}(A) := \sum_{\{\tau = s \overset{\nu}{\rightsquigarrow} s' | (\tau,t) \in A\}} \begin{cases} f_\pi(\theta, \tau, A) & \text{if } \phi = \theta \\ \mu_n(s) \cdot f_\pi(\phi, \tau, A) & \text{otherwise} \end{cases}$$

$$f_\pi(\phi, \tau, A) := \begin{cases} \pi(\phi \circ s)(\alpha) & \text{if } \tau = s \overset{\alpha}{\dashrightarrow} \mu \\ \displaystyle\int_{\mathbb{R}_{\geqslant 0}} \mathbb{1}_A((\tau, x)) \cdot \nu \cdot e^{-\nu \cdot x} \,\mathrm{d}x & \text{if } \tau = s \xrightarrow{\nu} \mu, \end{cases}$$

Here for a probabilistic transition $\tau = s \overset{\nu}{\dashrightarrow} \mu$ function $f_\pi(\phi, \tau, A)$ evaluates to the probability that scheduler π selects transition τ when it observes finite path $\phi \circ s$. Let $T = \{x \mid \mathbb{1}_A((\tau, x)) = 1\}$. Then for a Markovian transition $\tau = s \xrightarrow{\nu} \mu$ the value of $f_\pi(\phi, \tau, A)$ is the probability that Markov automaton leaves state s within time $x \in T$. Or in other words, $f_\pi(\phi, \tau, A)$ equals the probability that exponentially distributed random variable with parameter ν takes one of the values from set T.

Function $\boldsymbol{\mu}_{\pi,\phi}(A)$ computes the probability that steps in A are taken after path fragment ϕ has been observed. Positive probability is only assigned to those combinations of ϕ and $\sigma \in A$ that are in fact possible in the Markov automaton under consideration. This is taken care of by the multiplication of $\mu_n(s)$ with $f_\pi(\phi, \tau, A)$.

Fragments. Let $\pi \in \Pi_\mu$, $s \in S$ and $n \geqslant 0$. The probability measure $\boldsymbol{\mu}^{\mathcal{F}^n}_{\pi,s}(\cdot) : \mathcal{F}^n \to [0,1]$ over the measurable space $(\mathit{Frags}^n, \mathcal{F}^n)$ is defined as $\boldsymbol{\mu}^{\mathcal{F}^n}_{\pi,s}(\emptyset) := 0, \boldsymbol{\mu}^{\mathcal{F}^n}_{\pi,s}(\mathit{Frags}^n) := 1$ and otherwise for $n \geqslant 1$:

$$\boldsymbol{\mu}^{\mathcal{F}^n}_{\pi,s}(A) := \begin{cases} \boldsymbol{\mu}_{\pi,\theta}\Big(\{(\tau,t) \in A \mid \tau = s \overset{\nu}{\rightsquigarrow} \mu, \text{ for some } \nu, \mu\}\Big) & \text{if } n = 1 \\ \displaystyle\int_{\phi \in \mathit{Frags}^{n-1}} \boldsymbol{\mu}^{\mathcal{F}^{n-1}}_{\pi,s}(\mathrm{d}\phi) \int_{\sigma \in \mathit{Steps}} \mathbb{1}_A(\phi \circ \sigma) \cdot \boldsymbol{\mu}_{\pi,\phi}(\mathrm{d}\sigma) & \text{if } n > 1 \end{cases}$$

For $n \geqslant 1$ the measure $\boldsymbol{\mu}^{\mathcal{F}^n}_{\pi,s}(A)$ assigns non-zero values only to those sets A that contain fragments originating from state s. For $n > 1$ the measure is defined as multiplication of two measures: $\boldsymbol{\mu}^{\mathcal{F}^{n-1}}_{\pi,s}(\cdot)$ for fragments of length $n-1$ and $\boldsymbol{\mu}_{\pi,\phi}(\cdot)$ for steps that originate in a fragment of length $n-1$. Once again the probability is computed via integration over all fragments of length n that are in A.

Finite paths. Similarly, the probability measure for paths of length n originating in some state s is defined via the probability measure for fragments of length n. Let $\pi \in \Pi_\mu$, $s \in S$ and $n \geqslant 0$, then $\boldsymbol{\mu}^{\mathcal{P}^n}_{\pi,s}(\emptyset) := 0$, $\boldsymbol{\mu}^{\mathcal{P}^n}_{\pi,s}(\mathit{Paths}^n) := 1$, for $n = 0 : \boldsymbol{\mu}^{\mathcal{P}^n}_{\pi,s}(A) := 1$ if $s \in A$ and 0 otherwise. In all other cases, i. e. $n \geqslant 1, A \neq \emptyset, A \neq \mathit{Paths}^n$:

$$\boldsymbol{\mu}^{\mathcal{P}^n}_{\pi,s}(A) := \int_{\substack{\phi \in \mathit{Frags}^n, \mathit{and} \\ \phi = (\tau_0, t_0) \cdots (\tau_{n-1}, t_{n-1}), \\ \tau_{n-1} = s_{n-1} \overset{\nu_{n-1}}{\rightsquigarrow} \mu_{n-1}}} \boldsymbol{\mu}^{n}_{\pi,s}(\mathrm{d}\phi) \sum_{\{s' \mid \phi \circ s' \in A\}} \mu_{n-1}(s')$$

Infinite paths. The following lemma shows that the measures defined above lead to a unique probability measure over the set of infinite paths:

Lemma 2.2.1 ([Hat17]). *Let $\pi \in \Pi_\mu$ be a generic measurable strategy, $s \in S$. There exists a unique probability measure $\boldsymbol{\mu}^{\mathcal{P}^\omega}_{\pi,s}(\cdot) : \mathcal{P}^\omega \to [0,1]$ (also denoted with $\mathrm{Pr}^{\mathcal{M}}_{\pi,s}[\cdot]$), such that for every $n \in \mathbb{Z}_{\geqslant 0}, B_n \in \mathcal{P}^n$: $\boldsymbol{\mu}^{\mathcal{P}^\omega}_{\pi,s}(\mathit{Cyl}(B_n)) = \boldsymbol{\mu}^{\mathcal{P}^n}_{\pi,s}(B_n)$.*

Remark 2.2.2. *Consider Markov automata $\mathcal{M}_1, \mathcal{M}_2$ depicted in Figures (2.1b) and (2.1c) respectively. Notice that the measure $\boldsymbol{\mu}^{\mathcal{P}^\omega}_{\pi,s}(\cdot)$ assigns value 0 to those events in which Markovian transitions occur at a specific moment of time, or in other words within a time interval of length 0. This implies that the set of measurable events in $\mathcal{M}_1$ and $\mathcal{M}_2$ that have non-zero probability value are equal. When dealing with probabilistic properties, we can, therefore, assume that all states in Markov automata have either only probabilistic outgoing transitions or only Markovian ones. In case a closed Markov automaton has hybrid states, Markovian transitions of such states can be safely discarded thus transforming the hybrid state into probabilistic. Markov automaton depicted in Figure (2.1c) is the result of such transformation applied to state s_0 of the Markov automaton shown in Figure (2.1b).*

2.2.4 Zeno Behaviour

In general, Markov automata may act in such a way that infinitely many transitions are performed within a finite amount of time, a behaviour commonly referred to as *Zeno*.

We will call an infinite path ρ a *Zeno-path* if $\exists t \in \mathbb{R}_{\geqslant 0} : \forall i \in \mathbb{Z}_{\geqslant 0} : \tau_{\text{total}}(\rho, i) \leqslant t$. For a Zeno path the limit $\lim_{i\to\infty} \tau_{\text{total}}(\rho, i)$ exists and we can therefore define $\tau_{\text{total}}(\rho) = \lim_{i\to\infty} \tau_{\text{total}}(\rho, i)$.

Zeno-paths can be of two kinds: those induced by Markovian transitions and those induced by probabilistic ones. As an example, consider the following infinite path of a Markov automaton depicted in Figure (2.1c):

$$\rho_{MS} = s_0 \xrightarrow{\tau,0} s_1 \xrightarrow{\tau,0} s_4 \xrightarrow{\perp,1} G \xrightarrow{\perp,1/2} G \xrightarrow{\perp,1/4} \cdots G \xrightarrow{\perp,1/2^n} \cdots$$

The total time over this path converges to 2 and therefore infinitely many Markovian transitions occur within a finite amount of time. Consider the same MA in which state G is probabilistic with a self-loop transition $(G, \tau, \Delta_S(G))$. The following path is an example of a Zeno path in this MA induced by probabilistic transitions:

$$\rho_{PS} = s_0 \xrightarrow{\tau,0} s_1 \xrightarrow{\tau,0} s_4 \xrightarrow{\perp,1} G \xrightarrow{\tau,0} G \xrightarrow{\tau,0} \cdots G \xrightarrow{\tau,0} \cdots$$

Paths like ρ_{MS}, induced by Markovian transitions, are possible, however “unrealistic”, since it can be shown, analogously to the same results for continuous-time Markov chains [BHHK03], that the measure of the set of such paths is 0. Zeno paths induced by probabilistic transitions are different. For example, the probability of the set of paths

$$\{s_0 \xrightarrow{\tau,0} s_1 \xrightarrow{\tau,0} s_4 \xrightarrow{\perp,t} G \xrightarrow{\tau,0} G \xrightarrow{\tau,0} \cdots G \xrightarrow{\tau,0} \cdots \mid t \in [0,1]\}$$

is $p = 0.99 \cdot 1/8 \cdot (1 - e^{-2})$, which is non-zero and therefore with probability p the MA can perform infinitely many transitions within a finite amount of time. Since MA are used to model real-life systems, it is widely accepted [HH12, GHH+14, GTH+14, QJK17] that such behaviour should not be permitted and Markov automata that enable such paths are considered to have a modelling error.

We say that a Markov automaton $\mathcal{M}$ is *Zeno* if:

$\exists \pi \in \Pi^{\mathcal{M}}_{\mu}, s \in S :$

$$\Pr^{\mathcal{M}}_{\pi,s}\left[\{\rho \in Paths^{\omega}(\mathcal{M}) \mid \exists t \in \mathbb{R}_{\geqslant 0} : \forall i \in \mathbb{Z}_{\geqslant 0} : \tau_{\text{total}}(\rho, i) \leqslant t\}\right] > 0$$

Informally, $\mathcal{M}$ is Zeno if there exists a scheduler, such that the probability of the set of all Zeno paths under this scheduler is non-zero. A Markov automaton is called *non-Zeno* if it is not Zeno.

Remark 2.2.3. *In the following chapters, we will often work with absorbing states of a non-Zeno Markov automaton. Recall that absorbing states are those states that only have self-loop transitions. It follows from the definition of non-Zeno Markov automata that their absorbing states can only be Markovian.*

2.2.5 Important Subclasses and Related Models

We conclude this chapter with a brief overview of notable subclasses of Markov automata as well as related models. The models considered here are often encountered in the process of analysing Markov automata.

A *(homogeneous) continuous-time Markov chain (CTMC)* [Ros06] is a Markov automaton $\mathcal{C} = (S, \emptyset, \emptyset, \mathbf{R})$ without probabilistic transitions, and can be equivalently represented as a pair $(S, \mathbf{R})$. A *non-homogeneous CTMC* is a pair $(S, \mathbf{R}(t))$ in which the Markovian transition matrix $\mathbf{R}(t)$ is dependent on time t at which the system arrives in a state. Formally $\mathbf{R} : \mathbb{R}_{\geqslant 0} \times S \times S \to \mathbb{R}_{\geqslant 0}$. For $s, s' \in S$ and $\tau \in \mathbb{R}_{\geqslant 0}$ we will denote values of matrix $\mathbf{R}(t)$ with $\mathbf{R}(\tau)[s, s']$.

(Discrete-time) Markov decision processes (MDP) [Put94] is a Markov automaton without Markovian transitions, i. e. $\forall s \in S : E(s) = 0$. In this work we will mostly use a more traditional way of representing an MDP as a tuple $\mathcal{D} = (S, Act, \mathbf{P})$, where S is a finite set of states, Act is a finite set of actions and $\mathbf{P} : S \times Act \times S \to [0, 1]$ is a probabilistic transition matrix, s. t. $\forall \alpha \in Act : \sum_{s' \in S} \mathbf{P}[s, \alpha, s'] \in \{0, 1\}$.

We apply to MDPs the same no-deadlock assumption as to Markov automata in general. For the definition of MDPs that we use here this assumption is translated into the following. Let $\mathcal{D} = (S, Act, \mathbf{P})$ be an MDP, then

$$\forall s \in S : \exists \alpha \in Act, s' \in S, \text{ such that } \mathbf{P}[s, \alpha, s'] > 0$$

Notice that since MDPs do not have any Markovian transitions, they could be regarded as Zeno Markov automata, in light of the discussion in the previous section. This, however, should not be the case, since the semantics of MDP transitions is different from semantics attached to probabilistic transitions of Markov automata. Namely, MDP transitions are not considered to be executed instantly, but rather their execution shows that some unspecified amount of time has passed. Thus the notion of Zeno behaviour should not be applied to MDPs.

The probability measure for MDPs, stated in, for example, [Put94], coincides with the one presented in the previous section, and it is the latter one that we will refer to when working with MDPs.

PS-acyclic Markov automata. Most of the published Markov automata benchmarks (see e. g. [HKP+19]) satisfy certain restrictions on the shape of probabilistic transition relation. Namely, the relation is often acyclic, i. e. there exists no path

$$ps_0 \overset{\alpha_0,0}{\dashrightarrow} ps_1 \overset{\alpha_1,0}{\dashrightarrow} \cdots \overset{\alpha_{n-1},0}{\dashrightarrow} ps_n \overset{\alpha_n,0}{\dashrightarrow} ps_{n+1},$$

such that $ps_{n+1} = ps_0, \forall i = 0..n : ps_i \in PS, \alpha_i \in Act(ps_i), \mathbb{P}[ps_i, \alpha_i, ps_{i+1}] > 0$. We will call such Markov automata *PS*-acyclic. Acyclic probabilistic transition relation enables particularly efficient analysis of Markov automata.

For a *PS*-acyclic Markov automaton the *depth* $d(s)$ of a state s is the length of the longest path from this state to a Markovian state (including s itself). Or formally, let

$$\begin{aligned} A(s) = \Big\{ \rho = s_0 \overset{\alpha_0,0}{\dashrightarrow} s_1 \overset{\alpha_1,0}{\dashrightarrow} \cdots \overset{\alpha_{n-1},0}{\dashrightarrow} s_n \overset{\alpha_n,0}{\dashrightarrow} s_{n+1} \\ \mid s_0 = s, s_{n+1} \in MS, \forall i = 0..n : \alpha_i \in Act(s_i), \mathbb{P}[s_i, \alpha_i, s_{i+1}] > 0 \Big\} \end{aligned} \tag{2.1}$$

Then $d(s) := \max_{\rho \in A(s)} |\rho|$. We define $d_{\max} := \max_{s \in S} d(s)$.

Continuous-time Markov decision process (CTMDP) [Put94] is another well known probabilistic model closely related to Markov automata. CTMDPs, just like MA, can model non-determinism and exponential delays. However, in contrast to Markov automata, both of these features in CTMDPs are merged into one state and not split into probabilistic and Markovian states. Markov automata can be seen as a compositional extension of CTMDPs, because, as opposed to MA, CTMDPs cannot be used to model a system as a collection of concurrently running and interacting components.

Definition 2.2.8. *A* continuous-time Markov decision process (CTMDP) *is a tuple $\mathcal{C} = (S_\mathcal{C}, Act_\mathcal{C}, \mathbf{R}_\mathcal{C})$, where $S_\mathcal{C}$ is a finite set of* states*, $Act_\mathcal{C}$ is a finite set of* actions *and $\mathbf{R}_\mathcal{C} : S_\mathcal{C} \times Act_\mathcal{C} \times S_\mathcal{C} \to \mathbb{R}_{\geqslant 0}$ is a* rate matrix.

We will call the set $Act_\mathcal{C}(s) = \{\alpha \in Act_\mathcal{C} \mid \exists s' \in S : \mathbf{R}_\mathcal{C}[s, \alpha, s'] > 0\}$ *enabled actions* in state s. A *path* in a CTMDP is a finite or infinite sequence $\rho = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} s_2 \cdots$, where $s_i \in S_\mathcal{C}$, $\alpha_i \in Act(s_i)$ and $t_i \in \mathbb{R}_{\geqslant 0}$ denote the residence time of the system in state s_i. Analogously, a *path fragment* is defined as follows $\phi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots s_n \xrightarrow{\alpha_n,t_n}$. $E(s,\alpha) := \sum_{s' \in S} \mathbf{R}_\mathcal{C}[s, \alpha, s']$ and $\mathbb{P}_\mathcal{C}[s, \alpha, s'] := \mathbf{R}_\mathcal{C}[s, \alpha, s']/E(s, \alpha)$.

Most of the notions that we will need for CTMDPs are defined analogously to respective notions for Markov automata. Those include: The set of finite paths $Paths^*(\mathcal{C})$, infinite paths $Paths^\omega(\mathcal{C})$, path fragments $Frags^*(\mathcal{C})$, path or fragment length $|\rho|$ and last element of a finite path $\rho\downarrow$. Residence time in a state of a CTMDP is governed by an exponential distribution, just like in Markovian states of Markov automata. The rate of this distribution depends on the selected action, e. g. for action $\alpha \in Act(s)$ it equals $E(s, \alpha)$.

A *(late) scheduler* in a CTMDP is a measurable function $\pi : Paths^*(\mathcal{C}) \times \mathbb{R}_{\geqslant 0} \to \text{Dist}(Act)$, where for $\rho \in Paths^*(\mathcal{C}), t \in \mathbb{R}_{\geqslant 0}$ if $\pi(\rho, t)(\alpha) > 0$ then $\alpha \in Act(\rho\downarrow)$.

Notice that in contrast to Markov automata, schedulers in CTMDPs depend not only on the finite path leading to the current state but also on the parameter $t \in \mathbb{R}_{\geqslant 0}$, which is the amount of time that the system has spent in the current state. In Markov automata, this parameter is redundant since probabilistic states are left instantaneously.

A scheduler π is called *early* if $\forall \rho \in \mathit{Paths}^*(\mathcal{C}), \forall t, t' \in \mathbb{R}_{\geqslant 0} : \pi(\rho, t) = \pi(\rho, t')$, i. e. the scheduler does not depend on the amount of time that has passed in the current state.

Analysis of Markov Automata 3

The goal of this thesis is to advance analysis techniques for systems modelled as Markov automata. One could be interested in, for example, *qualitative* properties of the system, that have a simple boolean *yes* or *no* answer, e. g.: Can the system reach a certain state starting from a given one? Sometimes the answer to such a question is *maybe*, for example, when half of the times the system reaches the target state, while another half it does not. We call properties that capture such behaviour *quantitative* since they describe various aspects of system performance as a real number, rather than a boolean value. This number can represent probabilities, for example, the probability to reach certain states, alternatively, it can represent costs or revenue, in which case it is a positive or a negative real value. We will address the former as *probabilistic properties* and the latter as *reward properties.* In this chapter, we introduce such properties formally and touch upon techniques used for their analysis.

Many interesting probabilistic properties are described by *Continuous Stochastic Logic (or CSL).* CSL is one of the most popular temporal logics for many continuous-time Markovian models, including CTMCs [BHHK03] and CTMDPs [BHHZ11]. An example of a property that can be described by CSL is: *The probability to reach a subset of states within a certain time bound is above a certain threshold.* CSL was introduced for Markov automata in [HH12] and we reiterate over its syntax and semantics in Section 3.1, followed by a brief overview of algorithms designed to analyse CSL properties in Section 3.2.

When it comes to reward properties there is no reward-based logic for Markov automata that would unify their syntax and semantics. Instead, reward properties are usually introduced as stand-alone problems [GHH$^+$14, GTH$^+$14, BWH18]. We introduce them formally in Section 3.3 and describe existing analysis methods in Section 3.4.

We conclude this chapter with Section 3.5 that gives a brief overview of those properties for which no satisfactory analysis techniques are available to date.

Preliminaries. In this and the following chapters, we only consider Markov automata that satisfy the assumptions listed below.

– MA are closed (see Chapter 2.2.1).

– For each probabilistic state s and any two of its outgoing transitions $\gamma_1 = s \overset{\alpha_1}{\dashrightarrow} \mu_1, \gamma_2 = s \overset{\alpha_2}{\dashrightarrow} \mu_2, \gamma_1 \neq \gamma_2 : \alpha_1 \neq \alpha_2$. Due to this assumption we can define matrix $\mathbb{P}[\cdot,\cdot,\cdot]$ (see Chapter 2 Section 2.2).

– MA have no deadlock states (see discussion in Chapter 2 Section 2.2.2)

– MA are non-Zeno (see discussion in Chapter 2 Section 2.2.4)

– MA have no hybrid states (see Remark 2.2.2)

Throughout the chapter we will work with a Markov automaton $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$.

3.1 Continuous Stochastic Logic for Markov Automata

In this section we discuss *Continuous Stochastic Logic (CSL)* for Markov automata, that was originally presented in [HH12].

First of all, to define the semantics of CSL formulas we need to extend the definition of Markov automata to include a non-empty set of *atomic propositions* and a *(state) labelling function.* Atomic propositions are basic truths about the states. The labelling function assigns to each state of the Markov automaton a subset of atomic propositions, those that hold in that state.

Definition 3.1.1. *A* labelled Markov automaton $\mathcal{M}_\ell$ *is a tuple* $\mathcal{M}_\ell = (\mathcal{M}, AP, lab)$*, where* $\mathcal{M}$ *is a Markov automaton,* AP *is a non-empty set of atomic propositions and* $lab : S \to 2^{AP}$ *is a total* labelling function.

We assume, w. l. o. g. that for each state s there always exists an atomic proposition $ap_s \in AP$ that uniquely identifies s, i. e. $\forall s' \neq s : ap_s \notin lab(s')$.

Let $\mathcal{I}(\mathbb{R}_{\geqslant 0})$ be the set of all non-empty intervals of positive real numbers, i. e. intervals with open, closed or mixed bounds. For convenience we will denote an interval $I \in \mathcal{I}(\mathbb{R}_{\geqslant 0})$ with $I = [\![a, b]\!]$, where $[\![\in \{[, (\}$, $]\!] \in \{],)\}$, $a = \inf I$ and $b = \sup I$.

Definition 3.1.2 (CSL syntax)**.**

The following are valid CSL state *formulas:*

$$\begin{array}{rl} a & \textit{if } a \in AP \\ \neg\Phi & \textit{if } \Phi \textit{ is a state formula} \\ \Phi_1 \wedge \Phi_2 & \textit{if } \Phi_1 \textit{ and } \Phi_2 \textit{ are state formulas} \\ \mathcal{P}^{\mathrm{opt}}_{\trianglelefteq p}(\psi) & \textit{if } p \in [0,1],\ \mathrm{opt} \in \{\sup, \inf\},\ \psi \textit{ is a path formula and} \\ & \trianglelefteq \in \{<, \leqslant, >, \geqslant\} \end{array}$$

and path *formulas:*

$$\begin{array}{rl} \mathrm{X}^{[\![a,b]\!]}\Phi & \textit{if } [\![a,b]\!] \in \mathcal{I}(\mathbb{R}_{\geqslant 0}), a, b \in \mathbb{Q}_{\geqslant 0} \textit{ is an interval and} \\ & \Phi \textit{ is a state formula} \\ \Phi_1 \,\mathrm{U}\, \Phi_2 & \textit{if } \Phi_1, \Phi_2 \textit{ are state formulas} \\ \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2 & \textit{if } [\![a,b]\!] \in \mathcal{I}(\mathbb{R}_{\geqslant 0}) \textit{ is an interval, } a, b \in \mathbb{Q}_{\geqslant 0} \textit{ and} \\ & \Phi_1, \Phi_2 \textit{ are state formulas} \end{array}$$

Using this basic set of formulas we can express $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi 2)$ and $\mathtt{tt} = a \vee \neg a$, where $a \in AP$.

Definition 3.1.3 (CSL Semantics)**.** *Let $\mathcal{M}_\ell = (\mathcal{M}, AP, lab)$ be a labelled Markov automaton and $s \in S$. Below we define conditions under which state s* satisfies *a CSL state formula Φ, written as $s \models_{\mathcal{M}_\ell} \Phi$:*

$$\begin{array}{ll} s \models_{\mathcal{M}_\ell} a & \textit{iff } a \in lab(s) \\ s \models_{\mathcal{M}_\ell} \neg\Phi & \textit{iff } s \not\models_{\mathcal{M}_\ell} \Phi \\ s \models_{\mathcal{M}_\ell} \Phi_1 \wedge \Phi_2 & \textit{iff } s \models_{\mathcal{M}_\ell} \Phi_1 \textit{ and } s \models_{\mathcal{M}_\ell} \Phi_2 \\ s \models_{\mathcal{M}_\ell} \mathcal{P}^{\mathrm{opt}}_{\trianglelefteq p}(\psi) & \textit{iff } \operatorname*{opt}_{\pi \in \Pi_\mu} \Pr_{\pi,s}\left[\{\rho \in Paths^\omega \mid \rho \models_{\mathcal{M}_\ell} \psi\}\right] \trianglelefteq p \end{array}$$

Let $\rho = s_0 \xrightarrow{\nu_0, t_0} s_1 \xrightarrow{\nu_1, t_1} \cdots s_n \xrightarrow{\nu_n, t_n} \cdots \in Paths^\omega$ be an infinite path in $\mathcal{M}_\ell$. In the following we define conditions under which path ρ satisfies *a CSL path formula ψ, written as $\rho \models_{\mathcal{M}_\ell} \psi$:*

$$\begin{array}{lll} \rho \models_{\mathcal{M}_\ell} \mathrm{X}^{[\![a,b]\!]}\Phi & \textit{iff} & s_1 \models_{\mathcal{M}_\ell} \Phi \wedge t_0 \in [\![a,b]\!] \\ \rho \models_{\mathcal{M}_\ell} \Phi_1 \,\mathrm{U}\, \Phi_2 & \textit{iff} & \exists n \in \mathbb{Z}_{\geqslant 0} : s_n \models_{\mathcal{M}_\ell} \Phi_2, \forall k = 0..n-1 : s_k \models_{\mathcal{M}_\ell} \Phi_1 \\ \rho \models_{\mathcal{M}_\ell} \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2 & \textit{iff} & \exists t \in [\![a,b]\!], n \in \mathbb{Z}_{\geqslant 0} : s = \rho[n], s \in \rho@t, s \models_{\mathcal{M}_\ell} \Phi_2, \\ & & \forall \tau \in [0,t), s' \in \rho@\tau : s' \models_{\mathcal{M}_\ell} \Phi_1 \textit{ and} \\ & & \forall k = 0..n-1 : \rho[k] \models \Phi_1 \end{array}$$

The semantics of most operators is straightforward. We will just clarify that a path satisfies formula $\mathrm{X}^{[\![a,b]\!]}\Phi$ if the second state visited on the path satisfies formula Φ and transition from the initial state of the path to the next state happened within $[\![a, b]\!]$ time interval. A path satisfies formula $\Phi_1 \,\mathrm{U}\, \Phi_2$ if the path eventually reaches a state that satisfies Φ_2 and all the states visited along the path before this moment satisfy Φ_1. Formula $\Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2$ adds another limitation that all this must happen within time interval $[\![a, b]\!]$. We will call property $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\Phi_1 \,\mathrm{U}\, \Phi_2)$ *(optimal) unbounded reachability (probability)* and property $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2)$ will be referred to as *(optimal) time-bounded reachability (probability)*

Example properties that can be specified with this set of formulas include: *The probability that the system eventually reaches a bad state is at most* 0.999*, The probability that the system recovers from a critical situation within 24 hours is at least* 0.999, etc.

Whenever the labelled MA under consideration is clear from the context, we omit the subscript and simply write $\models$ instead of $\models_{\mathcal{M}_\ell}$. For a set of infinite paths A and a path formula ψ we will write $A \models \psi$ iff $\forall \rho \in A : \rho \models \psi$. And similarly, for a subset of states S' and a state formula Φ we write $S' \models \Phi$ iff $\forall s \in S' : s \models \Phi$. Additionally, for convenience, we will abuse the notation and write $\Pr_{\pi,s}[\psi]$ instead of $\Pr_{\pi,s}[\{\rho \in \mathit{Paths}^{\omega} \mid \rho \models \psi\}]$.

3.2 Analysis of CSL Properties

The process of exhaustive automated checking whether a given Markov automaton $\mathcal{M}_\ell$ satisfies a given CSL state formula Φ is called *model-checking.* Formally this means finding the set of states $\mathrm{Sat}_{\mathcal{M}_\ell}(\Phi)$, such that $\forall s \in \mathrm{Sat}_{\mathcal{M}_\ell}(\Phi) : s \models \Phi$. This can be performed recursively over the structure of the formula in a bottom-up manner. Model-checking of state formulas, except for operator $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\psi)$, is straightforward:

– $\Phi = a : \mathrm{Sat}(\Phi) = \{s \in S \mid a \in \mathit{lab}(s)\}$

– $\Phi = \neg\Phi' : \mathrm{Sat}(\Phi) = S \setminus \mathrm{Sat}(\Phi')$

– $\Phi = \Phi_1 \wedge \Phi_2 : \mathrm{Sat}(\Phi) = \mathrm{Sat}(\Phi_1) \cap \mathrm{Sat}(\Phi_2)$

Model-checking operator $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\psi)$ is more involved and the algorithms depend on the path formula ψ. All the algorithms essentially perform the following two steps: (i) for each state $s \in S$ compute the values $p_s = \mathrm{opt}_{\pi \in \Pi_\mu} \Pr_{\pi,s}[\psi]$; (ii) find the set of states S', such that $\forall s \in S' : p_s \unlhd p$. Thus the main challenge of model-checking CSL formulas is to compute the values p_s. When we say *analysis of a labelled MA against formula* $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\psi)$ we mean computation of these values. In the following, we will consider the analysis of formula $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\psi)$ separately for each possible ψ.

Formula $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\mathrm{X}^{[\![a,b]\!]}\Phi)$. Operator $\mathrm{X}^{[\![a,b]\!]}\Phi$ imposes restrictions on the second state appearing over a path and time that it takes to reach that state. Depending on the type of state, different procedures are used to determine whether it satisfies formula $\mathcal{P}^{\mathrm{opt}}_{\unlhd p}(\mathrm{X}^{[\![a,b]\!]}\Phi)$:

Lemma 3.2.1.
If $s \in PS$, then

$$\operatorname*{opt}_{\pi \in \Pi_\mu} \mathrm{Pr}_{\pi,s}\left[\mathrm{X}^{[\![a,b]\!]}\Phi\right] = \begin{cases} 0 & \textit{if } 0 \notin [\![a,b]\!] \\ \operatorname*{opt}_{\alpha \in Act(s)} \sum\limits_{s' \in \mathrm{Sat}(\Phi)} \mathbb{P}[s,\alpha,s'] & \textit{if } 0 \in [\![a,b]\!] \end{cases}$$

If $s \in MS$, then

$$\operatorname*{opt}_{\pi \in \Pi_\mu} \mathrm{Pr}_{\pi,s}\left[\mathrm{X}^{[\![a,b]\!]}\Phi\right] = \left(e^{-E(s)\cdot a} - e^{-E(s)\cdot b}\right) \sum_{s' \in \mathrm{Sat}(\Phi)} \mathbb{P}[s,s']$$

Proof. Consider $A = \{\rho \in Paths^\omega \mid \rho \text{ starts from } s \text{ and } \rho \models \mathrm{X}^{[\![a,b]\!]}\Phi\}$. With this operator only the very first transition affects the probability of event A, and what happens afterwards does not matter. We differentiate between the type of state:

– $s \in PS$. Residence time in s for the very first transition over any path can only be 0. If $0 \notin [\![a,b]\!]$ then $A = \emptyset$ and $\forall \pi \in \Pi_\mu : \mathrm{Pr}_{\pi,s}[A] = 0$. Consider the case $0 \in [\![a,b]\!]$. Let $\mu_\pi(\alpha)$ be the probability with which scheduler π chooses action α for path $\rho = s$, then

$$\begin{aligned} \forall \pi \in \Pi_\mu : \mathrm{Pr}_{\pi,s}[A] &= \sum_{s' \in \mathrm{Sat}(\Phi)} \sum_{\alpha \in Act(s)} \mu_\pi(\alpha) \cdot \mathbb{P}[s,\alpha,s'] \\ &\leqslant_{\mathrm{opt}} \operatorname*{opt}_{\alpha \in Act(s)} \sum_{s' \in \mathrm{Sat}(\Phi)} \mathbb{P}[s,\alpha,s'] \\ &= \operatorname*{opt}_{\pi \in \Pi_\mu} \mathrm{Pr}_{\pi,s}[A], \end{aligned}$$

where $\leqslant_{\mathrm{opt}} = \leqslant$ if opt = sup and $\leqslant_{\mathrm{opt}} = \geqslant$ if opt = inf. Thus the statement of the Lemma follows.

– Consider $s \in MS$. Event A can be split into two independent events A_1 and A_2. Event A_1 is the event of a Markovian transition originating from s to occur within time interval $[\![a,b]\!]$. A_2 describes discrete probability of a state satisfying Φ to be the selected successor of s. Then $\mathrm{Pr}_{\pi,s}[A] = \mathrm{Pr}[A_1] \cdot \mathrm{Pr}[A_2] = \left(e^{-E(s)\cdot a} - e^{-E(s)\cdot b}\right) \cdot \sum\limits_{s' \in \mathrm{Sat}(\Phi)} \mathbb{P}[s,s']$. The statement of the lemma follows.

$\square$

Formula $\mathcal{P}^{\mathrm{opt}}_{\trianglelefteq p}(\Phi_1 \,\mathrm{U}\, \Phi_2)$. Operator $\Phi_1 \,\mathrm{U}\, \Phi_2$ does not impose any restrictions on the time that it takes to reach a state satisfying Φ_2. Therefore the analysis of formula $\mathcal{P}^{\mathrm{opt}}_{\trianglelefteq p}(\Phi_1 \,\mathrm{U}\, \Phi_2)$ can be reduced to the analysis of the same formula over MDPs [QJK17]. This problem has been extensively studied and a multitude of algorithms has been developed to address it, including policy iteration [Put94], linear programming [Put94] and interval value iteration [HM14, QK18, BKL$^+$17].

Formula $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$ is the most challenging formula among the ones defined above. Decidability of model-checking formula $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$ is still an open problem and existing algorithms that approximate the probability values do not scale well enough. Analysis of this formula is a vast topic that we will discuss in greater detail in Chapter 4.

3.3 Reward Properties

When designing a system, costs associated with its maintenance, repairs as well as the generated revenue are of great importance. Example quantities of interest are the following: The expected average maintenance and repair costs, the expected total revenue, the expected system uptime, and so on. We will call quantities of this kind *reward properties*. The CSL logic for Markov automata described in the previous sections does not provide any means to express properties over quantities other than probabilities. In fact, to date there is no logic that can describe reward properties for Markov automata, as opposed to CTMCs [BHHK00] and CTMDPs [BHHZ11]. Nevertheless, over the years many reward properties have been introduced for Markov automata and in this section we will formally define them.

We start by extending the definition of Markov automata to include *rewards* for residing in a state or of taking a transition:

Definition 3.3.1. *A* reward structure *for an MA $\mathcal{M}$ is a tuple $\varrho := (\varrho_{\text{st}}, \varrho_{\text{trn}})$, where $\varrho_{\text{st}} : S \to \mathbb{R}_{\geqslant 0}$ is a* state reward function *and $\varrho_{\text{trn}} : \rightsquigarrow \to \mathbb{R}_{\geqslant 0}$ is a* transition reward function*.*

Definition 3.3.2. *Given a Markov automaton $\mathcal{M}$ and a reward structure ϱ over $\mathcal{M}$ we define a* Markov reward automaton (MRA) *to be a tuple $\mathcal{M}_\varrho := (\mathcal{M}, \varrho)$.*

Example 3.3.1. *Figure 3.1 shows an example of an MRA. Here rewards assigned to states and transitions are shown next to the respective state or transition in a green frame. For example, state* high *is assigned state reward 0.4, state* low *has state reward 0.1 and transition $(\mathit{has\ a\ task}) \overset{high}{\dashrightarrow} \mu$ for $\mu = [\mathit{high} \to 0.99, \mathit{lost} \to 0.01]$, has transition reward 10.*

The MRA models a task processing system. Tasks arrive at rate 11*; this is modelled by a Markovian transition with rate* 11 *emanating state* no tasks*. Whenever there is a task to process, the system decides whether to handle it with high or low reliability. In the former case, the system receives an immediate reward of 10. A low-reliability task produces a reward of 2 only.*

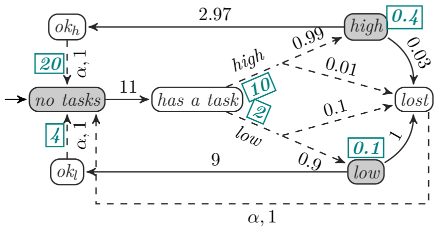


Figure 3.1: An example of an MRA

The tasks are sent for processing to a remote server over lossy channels. The high-reliability channel loses tasks with probability 0.01, while low-reliability tasks are *lost ten times more often. Whenever a task is lost, no further reward for it is paid. Processing high-reliability tasks takes more time, which is modelled with exit rate* 3 *of state* high*, however, it generates high reward. Low-reliability processing is fast (exit rate 10 of state* low*), however, produces lower reward 0.1. Upon successful completion of the processing, the system is awarded a reward of 20 or 4 for high- and low-reliability modes respectively.*

We will now define how the rewards are accumulated in an MRA. First of all, for an infinite path $\rho = s_0 \xrightarrow{\nu_0,t_0} \cdots s_k \xrightarrow{\nu_k,t_k} \cdots$ and $n \in \mathbb{Z}_{>0}$ we define the *prefix of ρ of length n* to be the following path fragment

$$\mathrm{prefix}_n^{\phi}(\rho) := s_0 \xrightarrow{\nu_0,t_0} \cdots s_{n-1} \xrightarrow{\nu_{n-1},t_{n-1}} \in \mathit{Frags}^n$$

For $t \in \mathbb{R}_{\geqslant 0}$ the *prefix* of ρ until time t is defined as

$$\mathrm{prefix}_t^{\phi}(\rho) := \begin{cases} \theta & \text{if } t_0 > t \\ \mathrm{prefix}_{n(t)}^{\phi}(\rho) & \text{if } n(t) \in \mathbb{Z}_{>0}, \tau_{\mathrm{total}}(\rho, n(t)) \leqslant t, \tau_{\mathrm{total}}(\rho, n(t)+1) > t \\ \rho & \text{otherwise} \end{cases}$$

Here $n(t)$ is such an integer value, that total time taken by $n(t)$ transitions is not greater than t, while within $n(t)+1$ transitions the total time exceeds t. For non-Zeno paths it is always possible to find such a value. However, for Zeno paths this is not always the case. Let ρ be a Zeno path. Recall that by definition total time $\tau_{\mathrm{total}}(\rho, k)$ for any number of transitions $k \in \mathbb{Z}_{\geqslant 0}$ is bounded: $\exists T : \forall k : \tau_{\mathrm{total}}(\rho, k) < T$. If one wants to compute prefix until time $t > T$, then there are infinitely many transitions that happen until time t, namely, all the transitions of ρ. For such cases we define the prefix until time t to be the path itself.

We are now in a position to define total reward accumulated over an infinite path or a path fragment:

Definition 3.3.3. The total reward over an infinite path (or a path fragment) *of an MRA $\mathcal{M}_\varrho$ is a function* $\mathrm{rew}_{\mathcal{M}_\varrho}(\cdot) : \mathit{Frags}^{n+1} \uplus \mathit{Paths}^{\omega} \to \mathbb{R}^{\infty}_{\geqslant 0}$*, such that*

$$\mathrm{rew}_{\mathcal{M}_\varrho}(\upsilon) := \begin{cases} 0 & \textit{if } \upsilon = \theta \\ \sum\limits_{i=0}^{n} \left(\varrho_{\mathrm{st}}(s_i) \cdot t_i + \varrho_{\mathrm{trn}}\left(s_i \overset{\nu_i}{\rightsquigarrow} \mu_i\right)\right) & \textit{if } \upsilon = s_0 \xrightarrow{\nu_0,t_0} \cdots s_n \xrightarrow{\nu_n,t_n} \in \mathit{Frags}^{n+1} \\ \lim\limits_{k\to\infty} \mathrm{rew}_{\mathcal{M}_\varrho}(\mathrm{prefix}_k^{\phi}(\rho)) & \textit{if } \upsilon = s_0 \xrightarrow{\nu_0,t_0} \cdots s_n \xrightarrow{\nu_n,t_n} \cdots \in \mathit{Paths}^{\omega} \end{cases}$$

Two sources contribute to the total reward over a path, or a fragment: states and transitions. Each visited state contributes the reward that is proportional to the amount of time that the system resides in the state. Given that the system resided for t time units in a state s, the reward accumulated in that state equals $\varrho_{\mathrm{st}}(s) \cdot t$. When leaving state s via transition $s \overset{\nu}{\rightsquigarrow} \mu$ the system collects a one-shot reward $\varrho_{\mathrm{trn}}\left(s \overset{\nu}{\rightsquigarrow} \mu\right)$. The total reward over a path fragment (infinite path) is the sum of all the rewards from states and transitions along the path fragment (infinite path). Naturally, it is possible that in an infinite path there exist infinitely many states or transitions with non-zero reward. In this case the limit $\lim\limits_{k\to\infty} \mathrm{rew}_{\mathcal{M}_\varrho}(\mathrm{prefix}_k^{\phi}(\rho)) = \infty$, which is the reason the value ∞ is in the domain of function $\mathrm{rew}_{\mathcal{M}_\varrho}$.

Example 3.3.2. *Consider the MRA from Fig. 3.1. In the following we will denote state* no tasks *with* nt *and* has a task *with* ht*. Consider path fragment*

$$\phi = nt \xrightarrow{\perp,t_0} ht \xrightarrow{high,0} high \xrightarrow{\perp,t_1} ok_h \xrightarrow{\alpha,0}$$

Then total reward accumulated over this fragment is $\text{rew}_{\mathcal{M}_\varrho}(\phi) = 10 + 0.4 \cdot t_1 + 20$.

We proceed to define various reward-based measures over infinite paths. For the following we fix an MRA $\mathcal{M}_\varrho$, one of its states s and a scheduler $\pi \in \Pi_\mu$. Formally we will define random variables on the probability space $(Paths^\omega, \mathcal{P}^\omega, \Pr_{\pi,s}[\cdot])$ and use their expected values to reason over all possible paths of an MRA.

Cumulative Time-Bounded Reward was introduced in [GTH+14]. It takes the value of the total reward accumulated over a given infinite path until time bound b.

Definition 3.3.4. *Let $b \in \mathbb{R}_{\geqslant 0}$ be a time-bound. The* cumulative time-bounded reward *is a random variable $C^b_{\mathcal{M}_\varrho,s,\pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ defined as follows:*

$$C^b_{\mathcal{M}_\varrho,s,\pi}(\rho) := \text{rew}_{\mathcal{M}_\varrho}(\text{prefix}^\phi_b(\rho)),$$

Example 3.3.3. *Consider the MRA from Fig. 3.1 and infinite path*

$$\rho = nt \xrightarrow{\perp,1.1} ht \xrightarrow{high,0} high \xrightarrow{\perp,2.2} ok_h \xrightarrow{\alpha,0} nt \xrightarrow{\perp,3.3} \cdots$$

For the time bound $b = 3.4$ the prefix $\text{prefix}^\phi_b(\rho) = nt \xrightarrow{\perp,1.1} ht \xrightarrow{high,0} high \xrightarrow{\perp,2.2} ok_h \xrightarrow{\alpha,0}$ *and* $C^b_{\mathcal{M}_\varrho,s,\pi} = \text{rew}_{\mathcal{M}_\varrho}(\text{prefix}^\phi_b(\rho)) = 10 + 0.4 \cdot 2.2 + 20$.

The random variable takes the value infinity on, for example, a Zeno path ρ, such that total time over the path $\tau_{\text{total}}(\rho) < b$ and the path takes infinitely often a probabilistic transition with non-zero transition reward.

Cumulative Goal-Bounded Reward was defined for Markov automata in [GTH+14].

Definition 3.3.5. *Let $G \subseteq S$ be a subset of states and $\rho = s_0 \xrightarrow{\nu_0,t_0} s_1 \cdots s_k \xrightarrow{\nu_k,t_k} s_{k+1} \cdots$ is an infinite path.* Cumulative goal-bounded reward *is a random variable $C^G_{\mathcal{M}_\varrho,s,\pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ defined as follows:*

$$C^G_{\mathcal{M}_\varrho,s,\pi}(\rho) := \begin{cases} 0 & \textit{if } s_0 \in G \\ \text{rew}_{\mathcal{M}_\varrho}(\text{prefix}^\phi_{n-1}(\rho)) & \textit{else if } s_n \in G \textit{ and } \forall i = 0..n-1 : s_i \notin G \\ \text{rew}_{\mathcal{M}_\varrho}(\rho) & \textit{if } \forall i \in \mathbb{Z}_{\geqslant 0} : s_i \notin G \end{cases}$$

This random variable takes the value of the total reward accumulated until the system visits some state $s_g \in G$ for the first time, or total reward over the whole path in case this never happens. If such a state s_g is encountered for the first time at position $n \in \mathbb{Z}_{\geqslant 0}$ (i. e. $n = 0, \rho[0] = s_g$ or $n > 0, \rho[n] = s_g, \forall i = 0..n-1 : \rho[i] \notin G$), then the rewards of states and transitions at indices 0 to $n-1$ are accumulated and the rewards of all the states and transitions starting from and including index n (i. e. starting from and including state s_g) are ignored.

Example 3.3.4. *Consider the MRA from Fig. 3.1 and infinite path*

$$\rho = nt \xrightarrow{\perp,1.1} ht \xrightarrow{high,0} high \xrightarrow{\perp,2.2} ok_h \xrightarrow{\alpha,0} nt \xrightarrow{\perp,3.3} ht \xrightarrow{low,0} low \xrightarrow{\perp,0.05} lost \xrightarrow{\alpha,0} \cdots$$

For the set $G = \{lost\}$ the prefix of the path until reaching G is

$$\phi = nt \xrightarrow{\perp,1.1} ht \xrightarrow{high,0} high \xrightarrow{\perp,2.2} ok_h \xrightarrow{\alpha,0} nt \xrightarrow{\perp,3.3} ht \xrightarrow{low,0} low \xrightarrow{\perp,0.05}$$

and $C^G_{\mathcal{M}_\varrho,s,\pi}(\rho) = 10 + 0.4 \cdot 2.2 + 20 + 2 + 0.1 \cdot 0.05$.

Cumulative Unbounded Reward is a special case of cumulative goal-bounded reward for $G = \emptyset$ and is widely used in various Markovian models, e. g. Markov decision processes. The value of such a random variable is the total reward accumulated over the infinite path.

Definition 3.3.6. Cumulative unbounded reward *is a random variable $C^{unb}_{\mathcal{M}_\varrho,s,\pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ defined as:*

$$C^{unb}_{\mathcal{M}_\varrho,s,\pi}(\rho) := C^G_{\mathcal{M}_\varrho,s,\pi}(\rho), \text{ where } G = \emptyset$$

Time Until Reaching the Goal is also a special case of $C^G_{\mathcal{M}_\varrho,s,\pi}$ and has been introduced in [GHH+14]. Let $\varrho(\tau)$ be a reward structure, such that all the transition rewards are set to 0 and all the state rewards equal 1. Consider the random variable $C^G_{\mathcal{M}_\varrho,s,\pi}$ for an MRA with reward structure $\varrho(\tau)$. The value of this random variable over a path ρ equals the time it takes to reach some state in the set G, or ∞ if no such state appears in ρ.

Definition 3.3.7. *Let $G \subseteq S$ be a subset of states. For a Markov automaton $\mathcal{M}$ a random variable $rT^G_{\mathcal{M},s,\pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ is defined as*

$$rT^G_{\mathcal{M},s,\pi}(\rho) := C^G_{\mathcal{M}_{\varrho(\tau)},s,\pi}(\rho)$$

Notice that the meaning of $rT^G_{\mathcal{M},s,\pi}(\rho)$ as the time to reach the goal set is not very intuitive in presence of Zeno paths. It is then possible that there exists an infinite path that never reaches any state from G and the value $rT^G_{\mathcal{M}_\varrho,s,\pi}$ over this path is finite. An example of such a path is a path ρ that never reaches G and from some point on only cycles within probabilistic states. The total time spent on such a path - $\lim_{n\to\infty} \tau_{\text{total}}(\rho, n)$ - is finite, which means that the value of the variable $rT^G_{\mathcal{M},s,\pi}$ is also finite. In such cases the value of the variable $rT^G_{\mathcal{M},s,\pi}$ does not represent the intuitive notion of time to reach G, since intuitively in this case it should be ∞.

Long-Run Average Reward takes the value of the average reward accumulated in path ρ per time unit. It was first defined for MRA in [GTH+14].

Definition 3.3.8. *The* long-run average reward *is a random variable* $L_{\mathcal{M}_\varrho,s,\pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ *defined as follows:*

$$L_{\mathcal{M}_\varrho,s,\pi}(\rho) := \lim_{t\to\infty} \frac{1}{t}\text{rew}_{\mathcal{M}_\varrho}(\text{prefix}^\phi_t(\rho))$$

Here *long-run* refers to the fact that the value of $L_{\mathcal{M}_\varrho,s,\pi}(\rho)$ depends on the whole path, not only on its finite prefix, what is the case for $C^b_{\mathcal{M}_\varrho,s,\pi}(\rho)$, unless the total time over the path does not exceed bound b, and for $C^G_{\mathcal{M}_\varrho,s,\pi}(\rho)$, unless no state in the set G is visited over the path.

Example 3.3.5. *Consider the MRA from Fig. 3.1 and infinite path*

$$\rho = nt \xrightarrow{\perp,\tau} ht \xrightarrow{high,0} high \xrightarrow{\perp,\tau} ok_h \xrightarrow{\alpha,0} nt \xrightarrow{\perp,\tau} \cdots$$

that always selects action high in state ht, never reaches state lost and stays in all Markovian states for exactly τ *time units. Then for* $k \in \mathbb{Z}_{\geqslant 0}$ *the total reward within* $2\cdot k\cdot\tau$ *time units is* $(10 + 0.4\cdot\tau + 20)\cdot k$ *and the average reward after* $2\cdot k\cdot\tau$ *time units is* $(10 + 0.4\cdot\tau + 20)\cdot k/(2\cdot k\cdot\tau) = 15/\tau + 0.2$. *Thus* $L_{\mathcal{M}_\varrho,s,\pi}(\rho) = 15/\tau + 0.2$.

Long-Run Average Time is a special case of long-run average reward defined for a subset of states $G \subseteq S$ and a reward structure $\varrho(\tau)$, such that all the transition rewards are set to 0 and state rewards equal 1 only for states from G, otherwise they are set to 0. The value was introduced in [GHH+14].

Definition 3.3.9. *Let* $G \subseteq S$. *The* long-run average time *is the random variable* $aT^G_{\mathcal{M},s,\pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ *defined as follows:*

$$aT^G_{\mathcal{M},s,\pi}(\rho) := L_{\mathcal{M}_{\varrho(\tau)},s,\pi}(\rho)$$

This random variable takes as a value the average amount of time that the system spends in states from G.

Discounted Reward considers rewards that are decreasing over time and was first introduced for Markov automata in [BWH18].

All the random variables defined above are based on the total reward $\text{rew}_{\mathcal{M}_\varrho}(\cdot)$ over a path or fragment. Here irrespectively of the time point at which a state s is entered or a transition $s \overset{\nu}{\rightsquigarrow} \mu$ is taken, this state or transition contributes the same reward to the total reward value, namely $\varrho_{\text{st}}(s)$ or $\varrho_{\text{trn}}\left(s \overset{\nu}{\rightsquigarrow} \mu\right)$. In many cases however it does make a difference when the reward is accumulated. For example, if the rewards are of monetary value, then they are subject to inflation. The more time passes by the less the same amount of money is worth. In the following we define the *discounted* total reward that takes into account time passage when rewards are accumulated.

Definition 3.3.10. *Let* $\phi = (s_0 \overset{\nu_0}{\rightsquigarrow} \mu_0, t_0) \cdots (s_n \overset{\nu_n}{\rightsquigarrow} \mu_n, t_n) \in \mathit{Frags}^{n+1}$ *be a path fragment in* $\mathcal{M}_\varrho$*. For* $\beta \in (0, 1]$ *we define* discounted reward with rate β over ϕ*, denoted as* $\mathrm{drew}^{\beta}_{\mathcal{M}_\varrho}(\phi)$*, as follows:*

$\mathrm{drew}^{\beta}_{\mathcal{M}_\varrho}(\theta) := 0$ *and for* $\phi \neq \theta$

$$\mathrm{drew}^{\beta}_{\mathcal{M}_\varrho}(\phi) := \sum_{i=1}^{n+1} \left[\int\limits_{\Delta(i-1)}^{\Delta(i-1)+t_{i-1}} e^{-\beta \cdot t} \cdot \varrho_{\mathrm{st}}\left(s_{i-1}\right) \mathrm{d}t + e^{-\beta \cdot \Delta(i)} \cdot \varrho_{\mathrm{trn}}\left(s_{i-1} \overset{\nu_{i-1}}{\rightsquigarrow} \mu_{i-1}\right) \right]$$

where $\Delta(i) := \tau_{\mathrm{total}}(\rho, i)$*.*

The rewards that are to be collected at time point t are multiplied with value $e^{-\beta \cdot t}$. The higher the value t is the smaller is $e^{-\beta \cdot t}$ and therefore for two time points t_1 and t_2, such that $t_2 > t_1$ the reward that is being contributed at time t_2 is smaller than that of t_1. The rate with which the rewards are discounted is controlled via parameter β. The closer this value is to 0, the slower is the discounting. The value $\beta = 0$ is excluded due to the fact that $\mathrm{drew}^{0}_{\mathcal{M}_\varrho}(\phi) = \mathrm{rew}_{\mathcal{M}_\varrho}(\phi)$.

Example 3.3.6. *Consider the MRA from Fig. 3.1 and its path fragment*

$$\phi = nt \xrightarrow{\perp, t_0} ht \xrightarrow{high, 0} high \xrightarrow{\perp, t_1} ok_h \xrightarrow{\alpha, 0}$$

Then the discounted reward collected over this path fragment is:

$$\mathrm{drew}^{\beta}_{\mathcal{M}_\varrho}(\phi) =$$

$$= e^{-\beta \cdot t_0} \cdot \varrho_{\mathrm{trn}}\left(ht \overset{high}{\dashrightarrow} \mu\right) + \int\limits_{t_0}^{t_0+t_1} \varrho_{\mathrm{st}}\left(high\right) \cdot e^{-\beta \cdot \tau} \,\mathrm{d}\tau + e^{-\beta \cdot (t_0+t_1)} \cdot \varrho_{\mathrm{trn}}\left(ok_h \overset{\alpha}{\dashrightarrow} \mu'\right)$$

$$= e^{-\beta \cdot t_0} \cdot 10 + \int\limits_{t_0}^{t_0+t_1} 0.4 \cdot e^{-\beta \cdot \tau} \,\mathrm{d}\tau + e^{-\beta \cdot (t_0+t_1)} \cdot 20$$

Definition 3.3.11. *We define random variable* $D^{\beta}_{\mathcal{M}_\varrho, s, \pi} : \mathit{Paths}^{\omega} \to \mathbb{R}^{\infty}_{\geqslant 0}$ *as follows:*

$$D^{\beta}_{\mathcal{M}_\varrho, s, \pi}(\rho) := \lim_{n \to \infty} \mathrm{drew}^{\beta}_{\mathcal{M}_\varrho}(\mathrm{prefix}^{\phi}_{n}(\rho))$$

Remark 3.3.1. *Notice that the value of reward functions* $\mathrm{rew}_{\mathcal{M}_\varrho}$ *and* $\mathrm{drew}^{\beta}_{\mathcal{M}_\varrho}$ *does not depend on state rewards of probabilistic states. This is due to the fact that residence time in probabilistic states is always* 0*. One can therefore assume w. l. o. g. that those values are always 0, i. e.* $\forall ps \in PS : \varrho_{\mathrm{st}}\left(ps\right) = 0$*.*

Optimal Expected Reward and Reward Properties. What we are interested in is the expected value of the above defined random variables for an optimal scheduler. Recall that the expected value of a random variable $X_{\pi,s}$ on the probability space $(Paths^{\omega}, \mathcal{P}^{\omega}, \Pr_{\pi,s}[\cdot])$ is defined as follows:

$$\mathbb{E}[X_{\pi,s}] = \int\limits_{Paths^{\omega}} X_{\pi,s}(\rho) \cdot \Pr_{\pi,s}[\mathrm{d}\rho]$$

For $X^{u}_{\mathcal{M}_y,s,\pi} \in \{C^{G}_{\mathcal{M}_\varrho,s,\pi}, C^{b}_{\mathcal{M}_\varrho,s,\pi}, C^{unb}_{\mathcal{M}_\varrho,s,\pi}, rT^{G}_{\mathcal{M},s,\pi}, L_{\mathcal{M}_\varrho,s,\pi}, aT^{G}_{\mathcal{M},s,\pi}, D^{\beta}_{\mathcal{M}_\varrho,s,\pi}\}$ we will use the following notation for the optimal expected reward:

$$\mathbb{E}^{\mathrm{opt}}[X^{u}_{\mathcal{M}_y,s}] := \operatorname*{opt}_{\pi\in\Pi_\mu} \mathbb{E}\left[X^{u}_{\mathcal{M}_y,s,\pi}\right]$$

We will refer to values $\mathbb{E}^{\mathrm{opt}}[X^{u}_{\mathcal{M}_y,s}]$ as *reward properties.* For example, the value $\mathbb{E}^{\mathrm{opt}}[C^{G}_{\mathcal{M}_\varrho,s}]$ is *cumulative goal-bounded reward property.*

3.4 Analysis of Reward Properties

In this section, we will give a brief overview of algorithms for the analysis of reward properties. By *analysis* here we mean computation of the respective optimal expected values.

Computation of the cumulative goal-bounded reward property $\mathbb{E}^{\mathrm{opt}}[C^{G}_{\mathcal{M}_\varrho,s}]$ can be reduced to the same problem on MDPs, as it has been shown in [GTH+14]. One can therefore use standard algorithms such as linear programming, policy iteration [Put94] and interval value iteration [BKL+17, QK18]. Computation of $\mathbb{E}^{\mathrm{opt}}[rT^{G}_{\mathcal{M}_\varrho,s}]$ can be reduced to the computation of $\mathbb{E}^{\mathrm{opt}}[C^{G}_{\mathcal{M}'_\varrho,s}]$ on an adapted MRA $\mathcal{M}'_\varrho$ (different reward structure). Computation of $\mathbb{E}^{\mathrm{opt}}[C^{unb}_{\mathcal{M}_\varrho,s}]$ can be reduced to computing $\mathbb{E}^{\mathrm{opt}}[C^{G}_{\mathcal{M}_\varrho,s}]$ for $G = \emptyset$ and can therefore be solved with the same techniques.

Analysis of properties $\mathbb{E}^{\mathrm{opt}}[L_{\mathcal{M}_\varrho,s}]$, $\mathbb{E}^{\mathrm{opt}}[aT^{G}_{\mathcal{M}_\varrho,s}]$ and $\mathbb{E}^{\mathrm{opt}}[D^{\beta}_{\mathcal{M}_\varrho,s}]$ requires specialised algorithms. Value $\mathbb{E}^{\mathrm{opt}}[L_{\mathcal{M}_\varrho,s}]$ can be computed with a reduction to a linear programming problem, as shown in [GTH+14]. Analysis of this property will be discussed in detail in Chapter 5. Value $\mathbb{E}^{\mathrm{opt}}[aT^{G}_{\mathcal{M}_\varrho,s}]$ is a special instance of $\mathbb{E}^{\mathrm{opt}}[L_{\mathcal{M}_\varrho,s}]$ and can thus be solved with the same techniques. Computation of value $\mathbb{E}^{\mathrm{opt}}[D^{\beta}_{\mathcal{M}_\varrho,s}]$ can be performed via policy- and value-iteration algorithms that have been presented in [BWH18].

The only algorithm for analysing property $\mathbb{E}^{\mathrm{opt}}[C^{b}_{\mathcal{M}_\varrho,s}]$ is presented in [GTH+14] and is based on *discretisation.* It is a natural extension of the algorithm for analysing formula $\mathcal{P}^{\mathrm{opt}}_{\trianglelefteq p}(\Phi_1\, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$ from Markov automata to Markov reward automata.

3.5 Discussion

In this chapter, we looked at various probabilistic and reward properties of Markov automata and briefly discussed techniques used for their analysis. Many of the considered properties can be analysed efficiently. Those include unbounded reachability probability, cumulative goal-bounded reward, expected time until reaching a goal, unbounded and discounted rewards. Some of the properties, however, remain challenging to date, and those are:

1. *Time-bounded reachability probability property* $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$. This property is a challenge not only for Markov automata, but also for CTMDPs. It involves solving an optimisation problem over the system dynamics governed by linear differential equations. Existing algorithms scale poorly either with the size of the model or with parameters of the problem, such as the time bound or approximation error. We discuss this problem in Chapter 4.

2. *Cumulative time-bounded reward property* $\mathbb{E}^{\text{opt}}[C^{b}_{\mathcal{M}_{\varrho},s}]$ is an extension of time-bounded reachability property with rewards. Everything discussed above applies here as well.

3. *Long-run average reward property* $\mathbb{E}^{\text{opt}}[L_{\mathcal{M}_{\varrho},s}]$. To date there is no iterative solution, such as value- or policy-iteration, to approximate this property. The only available solution is based on a reduction to linear programming. Despite the fact that linear programming has polynomial time complexity, for other Markovian models, such as MDPs and CTMDPs, iterative approaches tend to scale better on many practical applications. We will discuss in detail analysis of this property in Chapter 5.

4. *Long-run average time property* $\mathbb{E}^{\text{opt}}[aT^{G}_{\mathcal{M}_{\varrho},s}]$ is a special case of long-run average reward property and it suffers from the same issues.

Time-Bounded Reachability 4

Analysis of time-bounded properties, such as time-bounded reachability and cumulative time-bounded reward, is by far the most challenging problem for Markov automata. The time-bounded reachability $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$ is the simpler among the two and answers the question of whether the optimal probability to reach a certain subset of states within a given time interval while avoiding undesirable states satisfies $\trianglelefteq p$. In this chapter, we explore how to analyse this property. The main results of this chapter are two new algorithms for approximating optimal time-bounded reachability probabilities up to a given approximation error.

In Section 4.1 we set the basis. First in Section 4.1.1 we discuss existing algorithms that address the problem, their limitations and justify the need for a better solution. In Section 4.1.2 we show that by modifying the Markov automaton the problem can be reduced to solving at most two instances of a simpler one. Namely, computation of the optimal probabilities w. r. t. a path formula $\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ can be performed by computing optimal probabilities for at most two path formulas of the form $\texttt{tt} \, \mathrm{U}^{=c} \Phi_2$ and combining the results. Thus in order to obtain an algorithm for analysing formula $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$ one only needs an algorithm for analysing a slight generalisation of formula $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\texttt{tt} \, \mathrm{U}^{=c} \Phi_2)$.

In Section 4.2 we formalise the problem of quantifying $\mathcal{P}^{\text{opt}}_{\trianglelefteq p}(\texttt{tt} \, \mathrm{U}^{=c} \Phi_2)$ and discuss the structure of optimal schedulers. In Section 4.2.2 we introduce a sub-problem that is often encountered in the context of Markov automata analysis: computation of unbounded reachability probabilities over probabilistic states and we discuss existing algorithms to compute these values.

We present two new algorithms for approximating optimal time-bounded reachability probabilities up to a given error bound in Sections 4.3 and 4.4. The first one performs exhaustive state-space exploration. The main idea behind the algorithm is to exploit the structure of optimal schedulers, which is made possible through new insights on the switching points, developed in Section 4.3.3. The second algorithm performs computations on part of the total state-space. It can be thought of as a wrapper for any exhaustive solver, to speed up its computations. It uses the property and the structure of the Markov automaton to discard states that are not relevant for the problem and runs the exhaustive solver only on the remaining part of the total state-space. An extensive evaluation of both algorithm and comparison with existing approaches follows in Section 4.5.

4.1 Preliminaries

Throughout the chapter we work with a labelled Markov automaton $\mathcal{M}_\ell = (\mathcal{M}, AP, lab)$, where $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$. First of all, in order to present the results for both cases opt = sup and opt = inf, we define the following relation operators:

$$\preccurlyeq_{\text{opt}} := \begin{cases} \leqslant & \text{if opt} = \sup \\ \geqslant & \text{if opt} = \inf \end{cases} \qquad \succcurlyeq_{\text{opt}} := \begin{cases} \geqslant & \text{if opt} = \sup \\ \leqslant & \text{if opt} = \inf \end{cases} \tag{4.1}$$

$$\prec_{\text{opt}} := \begin{cases} < & \text{if opt} = \sup \\ > & \text{if opt} = \inf \end{cases} \qquad \succ_{\text{opt}} := \begin{cases} > & \text{if opt} = \sup \\ < & \text{if opt} = \inf \end{cases} \tag{4.2}$$

For the case of supremum, the semantics of these operators coincides with the intuitive meaning ($\preccurlyeq_{\text{opt}}$ is less-or-equal operator, $\succcurlyeq_{\text{opt}}$ is greater-or-equal, etc.), and is the opposite for infimum.

First, we spell out the problem of interest:

Problem 1. *Let $\mathcal{M}_\ell$ be a labelled Markov automaton and $\psi = \Phi_1 \, \mathrm{U}^{[a,b]} \Phi_2$ is a CSL path formula. Compute the values*

$$\forall s \in S : \mathrm{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi) := \underset{\pi \in \Pi_\mu}{\text{opt}} \; \Pr_{\pi,s}^{\mathcal{M}} \left[\{\rho \in Paths^\omega \mid \rho \models \psi\}\right]$$

*as well as scheduler π for which optimum is achieved in the equation above (*optimal *scheduler).*

For a scheduler π we define $\mathrm{val}_{\pi}^{\mathcal{M}_\ell}(s, \psi) := \Pr_{\pi,s}^{\mathcal{M}} \left[\{\rho \in Paths^\omega \mid \rho \models \psi\}\right]$. We will call a scheduler π *ϵ-optimal* if $\forall s \in S : \mathrm{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi) - \mathrm{val}_{\pi}^{\mathcal{M}_\ell}(s, \psi) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})}\epsilon$.

4.1.1 Known Solutions and Related Work

In this section, we will discuss various algorithms solving Problem 1. We start by looking at dedicated MA algorithms, then move to techniques that address the analogous problem on CTMDPs and conclude with a discussion of the relationship between methods designed for MA and those for CTMDPs. All the algorithms that are known to date use approximations to quantify the optimal reachability value. We will only consider those algorithms, that provide formal guarantees on the obtained results, i. e. an upper bound on the error between the value that they compute and the actual value. We will denote with ϵ the precision guaranteed by an approximation algorithm.

Dedicated MA Algorithms for Problem 1. Let $|S| = n$ be the amount of states, $a = |Act|$ and m the amount of *edges* in $\mathcal{M}$. A transition $s \overset{\nu}{\rightsquigarrow} \mu$ produces as many edges as there are elements in the support of μ. We will denote with $C_{i,j,a}$ the worst-case complexity of computing value $\text{opt}_{\pi\in\Pi_\mu} \Pr_\pi^{\mathcal{D}}[\texttt{tt}\,\mathrm{U}\,\Phi_2]$ in an MDP $\mathcal{D}$ with i states, j edges and a actions. If the problem is formulated as a linear program, then the linear program has i variables and $i \cdot a$ constraints. The following is the list of algorithms that approximate optimal reachability probabilities in Markov automata:

– The first algorithm to solve Problem 1 was presented in [GHH+14] and is based on a transformation to a discrete time model. The transformation is obtained by discretising the time horizon into intervals of equal length and approximating the value on each interval with exponential functions. The length of the interval is selected in such a way, that with high probability at most one Markovian transition will be performed within this interval. The complexity of the algorithm is $O((C_{m,n,a}+m+n)\cdot(\mathbf{E}_{\max}\cdot b)^2/(2\cdot\epsilon))$. Throughout the thesis we will refer to this algorithm as `FixStep`. In order to obtain guarantees on the error induced by the approximations `FixStep` essentially assumes that the optimal action for at least one state may change after each Markovian transition, and therefore computes optimal actions for all intervals and all states. In most case studies however optimal actions remain optimal over longer intervals than those used by `FixStep`. This means that for many practical applications `FixStep` performs too many unnecessary computations. This is the same problem as a similar algorithm for CTMDPs has, something that has been shown in [BHHK15].

– Another approach to solve Problem 1 is via a combination of uniformisation and time-abstract schedulers [Gro18], which is a Markov automata extension of the same technique designed for CTMDPs [BHHK15]. The approach is only shown for $\psi = \texttt{tt}\,\mathrm{U}^{[0,b]}\Phi_2$. The algorithm performs k iterations, each of complexity $O((\mathbf{E}_{\max}\cdot 2^{k-1}\cdot b\cdot e^2 - \ln(\epsilon))\cdot(C_{m,n,a}+m))$. There is no known upper bound on k. In this work, we will denote this algorithm with `Unif+`. The exponential dependency on the number of iterations k means that the algorithm performs well whenever only a few iterations are needed to achieve the desired accuracy. This is however not always the case and it has been observed that in the CTMDP case there are examples on which the algorithm has to perform many iterations to achieve the required accuracy and its running time falls below the running time of other algorithms.

Related CTMDP Algorithms. Many algorithms addressing Problem 1 for Markov automata are extensions of the respective algorithms for CTMDPs. The problem of optimal time-bounded reachability is defined for CTMDPs analogously to the way it is defined for MA and can be found in e. g. [RS11]. For CTMDPs it has been studied separately for *early* and *late* scheduler classes (see Chapter 2, Section 2.2.5) and in the following we will briefly visit all the algorithms:

– The discretisation-based algorithm presented in [NZ10, Neu10] is the CTMDP-predecessor of the `FixStep` algorithm discussed above. It is available for the early and late scheduler classes. The latter, however, is supported only if the CTMDP is *locally uniform.* Informally, it means that the exit rates are state-wise constant, or formally: $\forall s \in S, \forall \alpha, \beta \in Act_{\mathcal{C}}(s) : E(s,\alpha) = E(s,\beta)$.

– Another discretisation-based algorithm that approximates the value for late schedulers has been presented in [FRSZ11]. It is capable of discretising the time horizon with a coarser grid than the one of [Neu10] and within each interval approximates the value by polynomial functions.

– In most cases, the two algorithms mentioned above use a very fine discretisation grid, where each interval has the same length. It is assumed that the optimal behaviour within each of those intervals may be different from that in the neighbouring intervals. While this may happen in the worst-case scenario, in many cases the optimal behaviour is less regular and does not change that often. The discretisation-based approach presented in [BS11, BHHZ11] addresses this issue by splitting the time-horizon into intervals of *variable* length. The main idea is the following: If for a given problem the optimal behaviour does not change over an interval $[a, b]$, then this interval should not be discretised into finer ones. Thus the algorithm can adapt to each given problem instance by using a possibly different discretisation grid.

  The new algorithm that we will present later in Section 4.3 follows the same ideas. A more detailed discussion of the relationship between the two can be found in Section 4.3.

– Lastly, the `Unif+` approach for CTMDPs has been introduced in [BHHK15] for both early and late scheduler classes. It served as a prototype for the respective MA algorithm discussed above.

Scheduler Classes in MA and CTMDPs. We conclude this section with a discussion on whether CTMDP algorithms can be harvested for computing the optimal time-bounded reachability for MA and vice versa.

For simplicity, we will only consider the most classical formulation of the problem. Let $\Diamond^{\leqslant t}G$ be a set of paths that reach a state from set G within time interval $[0, t]$.

Case 1. Given a CTMDP $\mathcal{C}$, one of its states $s_{\mathcal{C}}$, a class of schedulers $\Pi^{\mathcal{C}}$ (early or late) and a subset of states $G_{\mathcal{C}}$ is there a Markov automaton $\mathcal{M}$, one of its states $s_{\mathcal{M}}$ and a subset of states $G_{\mathcal{M}}$, such that the following equation holds?

$$\operatorname*{opt}_{\pi\in\Pi^{\mathcal{C}}} \Pr^{\mathcal{C}}_{\pi,s_{\mathcal{C}}}\left[\Diamond^{\leqslant t}G_{\mathcal{C}}\right] = \operatorname*{opt}_{\pi\in\Pi^{\mathcal{M}}_{\mu}} \Pr^{\mathcal{M}}_{\pi,s_{\mathcal{M}}}\left[\Diamond^{\leqslant t}G_{\mathcal{M}}\right] \tag{4.3}$$

The answers to this question have been described in [RS11, RS10] for continuous time Markov games, which are more general than closed Markov automata and than CTMDPs. We will revisit these results below.

If $\Pi^{\mathcal{C}}$ is the class of early schedulers, then the required MA has been described in [RS11], Section 5.1. The main idea is to decouple every CTMDP state into one probabilistic state and several Markovian states, one per each enabled action in the CTMDP. The Markovian states have the same outgoing transitions as the CTMDP state via the respective action. The probabilistic state has as many outgoing probabilistic transitions as there are enabled actions in the CTMDP state. All of these transitions lead with probability 1 to the respective Markovian state.

A similar procedure can be applied when $\Pi^{\mathcal{C}}$ is the class of late schedulers. The main ideas have been described in Appendix D of [RS10]. First of all the CTMDP is uniformised, which does not affect the optimal reachability probability value. In the resulting uniform CTMDP, in contrast to the previous case, the probabilistic state is placed *after* the Markovian state.

In both of the cases described above the size of the Markov automaton that preserves the optimal reachability probabilities is linear in the size of the given CTMDP.

Case 2. Given an MA $\mathcal{M}$, one of its states $s_{\mathcal{M}}$ and a subset of states $G_{\mathcal{M}}$ is there a CTMDP $\mathcal{C}$, one of its states $s_{\mathcal{C}}$, a class of schedulers $\Pi^{\mathcal{C}}$ (early or late) and a subset of states $G_{\mathcal{C}}$, such that equation (4.3) holds?

For the class of late schedulers, the required CTMDP has been described in [RS11], Section 5. The main idea is the following. For Markovian states ms_1, ms_2 and an untimed path $ms_1 \xrightarrow{\nu_0} ps_1 \xrightarrow{\nu_1} \cdots ps_k \xrightarrow{\nu_k} ms_2$, where states ps_i are all probabilistic, the CTMDP has a transition $s_1 \xrightarrow{\nu} S_2$, where $S_2 = \underline{ps_1 \xrightarrow{\nu_1} \cdots ps_k \xrightarrow{\nu_k} ms_2}$ and $\nu = \underline{\nu_1 \to \cdots \to \nu_k}$. All states of the CTMDP that end in state ms_1, e. g. $\underline{ps_1 \xrightarrow{\nu_1} \cdots ps_k \xrightarrow{\nu_k} ms_1}$, get the same outgoing actions as s_1. The rate matrix is selected accordingly. If there are probabilistic states that have no incoming Markovian transitions, then such probabilistic states have not been added to the CTMDP by the transformation described above. However, they can only serve as initial states and therefore can be processed separately when computing the value.

For the class of early schedulers, a similar transformation can probably be used, however, we are not aware of any published results. We conjecture that the transformation described in [BWH18] may be applicable in this case. The main idea follows the same lines as the one described above. First, a subset of the state-space of the Markov automaton is selected as the basis for the state-space of the CTMDP. In contrast to the transformation described above, in this case, such states are selected among probabilistic ones. Next, the outgoing transitions for each of those states are selected in such a way, that the probabilistic and non-deterministic behaviour of the MA is preserved by the CTMDP. To this end, actions and transitions of states that are not part of the selected subspace are encoded into actions and transitions of states that are in it.

Notice that for both early and late scheduling problems the resulting CTMDP is in the worst case exponential in the size of the MA. This is due to the fact that all possible scheduler decisions over several probabilistic states of the MA have to be encoded into enabled actions of one state in the CTMDP. If there are n probabilistic states that have at least two enabled actions, then this can result in 2^n enabled actions in the CTMDP.

Conclusions. According to [RS11, RS10], in order to compute the optimal reachability probability for a CTMDP under late or early schedulers one can construct a Markov automaton (linear in the size of the CTMDP) that preserves the value and apply dedicated Markov automata algorithms to it. However, to compute the optimal reachability probability for an MA via CTMDP algorithms, one needs to construct a CTMDP that may in the worst case be exponentially larger than the given MA. We conjecture that it may be possible to avoid the expensive transformation by adapting the CTMDP algorithms to Markov automata, similarly to the way it is done for long-run average- and discounted rewards (Chapter 5 and [BWH18]).

4.1.2 Reduction to Two-Step Solution

In this section we will show that the solution of Problem 1 for an arbitrary path formula $\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ can be reduced to solving at most two instances of a simpler problem, in which the path formula is of the following shape: $\texttt{tt} \, \mathrm{U}^{=c} \, \Phi_2$. This result and parts of its proof echo a result first established for CTMCs [BHHK03].

As the first step, we will show that analysing formula $\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ can be reduced to analysing formula $\texttt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ on a slightly modified Markov automaton.

We will start with defining an operation of transforming states of a Markov automaton into absorbing states. A state $s \in S$ can be *made absorbing* by removing all of its outgoing transitions, if any, and substituting them with only one self-loop Markovian transition with the rate $\mathbf{E}_{\max}$. Notice that if this transformation is applied to a probabilistic state, then it will become Markovian.

We define $\mathcal{M}_\ell^{[\Phi]} := (\mathcal{M}^{[\Phi]}, AP, lab)$, where $\mathcal{M}^{[\Phi]} := (S^{[\Phi]}, Act, \dashrightarrow^{[\Phi]}, \mathbf{R}^{[\Phi]})$, to be a labelled Markov automaton obtained from $\mathcal{M}_\ell$ by making all the states in $\mathrm{Sat}(\Phi)$ absorbing.

Notice that any generic measurable scheduler π in $\mathcal{M}$ is a generic measurable scheduler in $\mathcal{M}^{[\Phi]}$. This is due to the fact that absorbing states are Markovian and thus $PS_{\mathcal{M}^{[\Phi]}} \subseteq PS_{\mathcal{M}}$. If no state satisfying Φ appears on a finite path ρ in $\mathcal{M}^{[\Phi]}$, then the same path is possible in $\mathcal{M}$ and the scheduler takes the same decisions in both models. Otherwise, after encountering a state satisfying Φ, the path never leaves this state, since it is absorbing, and there is no decision to make for the scheduler since the state is Markovian.

Lemma 4.1.1. *Let $\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ be a path formula and $\Phi_2 \Rightarrow \Phi_1$, then*

$$\forall s \in S, \pi \in \Pi_\mu : \mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}_\pi^{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}}(s, \texttt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$$

Proof. Consider a labelled MA $\mathcal{M}'_\ell$, a finite path ρ in $\mathcal{M}'_\ell$ and the following condition:

$$\begin{aligned} &\exists t \in [\![a,b]\!], 0 \leqslant n \leqslant |\rho|, s = \rho[n], s \in \rho@t, s \models_{\mathcal{M}'_\ell} \Phi_2, \\ &\forall \tau \in [0,t), s' \in \rho@\tau : s' \models_{\mathcal{M}'_\ell} \Phi_1 \text{ and} \\ &\forall k = 0..n-1 : \rho[k] \models_{\mathcal{M}'_\ell} \Phi_1 \end{aligned} \tag{4.4}$$

If ρ satisfies (4.4), we will say that $\rho \models_{\mathcal{M}'_\ell} \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$.

Condition (4.4) is analogous to the semantics of formula $\Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ with the only difference that in the latter one the value of n is restricted from above by the length of ρ.

Consider a finite path ρ, such that $\rho \models_{\mathcal{M}_\ell} \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$ in the above defined sense. Then for all $i = 0..n : \rho[i] \models_{\mathcal{M}_\ell} \Phi_1 \vee \Phi_2$ and $Cyl_{\mathcal{M}_\ell}(\rho) \models_{\mathcal{M}_\ell} \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$. Equivalently $\forall i = 0..n : \rho[i] \not\models_{\mathcal{M}_\ell} \neg\Phi_1 \wedge \neg\Phi_2$. This implies that ρ is also a finite path in $\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}$ and $\rho \models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \mathtt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$.

Similarly in the other direction. If $\rho \models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \mathtt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$, then there exists $n \leqslant |\rho|$, such that $\rho[n] \models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \Phi_2$. Since all states that satisfy $\neg\Phi_1 \wedge \neg\Phi_2$ are absorbing in $\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}$, then $\forall i = 0..n : \rho[i] \not\models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \neg\Phi_1 \wedge \neg\Phi_2$ (otherwise $\rho[n] \not\models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \Phi_2$, what contradicts our first assumption). Equivalently, $\forall i = 0..n : \rho[i] \models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \Phi_1 \vee \Phi_2$ and $\rho[n] \models_{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}} \Phi_2$. This implies that ρ is also a finite path in $\mathcal{M}_\ell$ and due to assumption $\Phi_2 \Rightarrow \Phi_1$: $\forall i = 0..n : \rho[i] \models_{\mathcal{M}_\ell} \Phi_1$, what leads to $\rho \models_{\mathcal{M}_\ell} \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$.

The probabilities of the events we are interested in do generally not depend on what happens after the formula is satisfied, but only on what happens before. The latter coincides within both models and we conclude that $\mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}_\pi^{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}}(s, \mathtt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$.

$\square$

Fixpoint characterisation and optimal schedulers. Due to Lemma 4.1.1 we can restrict ourselves to formulas $\psi = \mathtt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$. Next we will show the fixpoint characterisation of value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\mathtt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$.

Lemma 4.1.2 ([GHH+14]). *The value* $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \mathtt{tt} \, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$ *is the least fixpoint of the higher-order operator* $\Omega : (S \times \mathcal{I}(\mathbb{R}_{\geqslant 0}) \to [0,1]) \to (S \times \mathcal{I}(\mathbb{R}_{\geqslant 0}) \to [0,1])$*, such that*

$$\Omega(F)(s,I) = \begin{cases} f(s,I,b) & \text{if } s \in MS \cap \mathrm{Sat}(\neg\Phi_2) \\ e^{-E(s)\cdot a} + f(s,I,a) & \text{if } s \in MS \cap \mathrm{Sat}(\Phi_2) \\ 1 & \text{if } s \in PS \cap \mathrm{Sat}(\Phi_2) \wedge 0 \in I \\ \underset{\alpha \in Act(s)}{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[s,\alpha,s'] \cdot F(s',I) & \text{otherwise,} \end{cases} \tag{4.5}$$

where for $s \in S, z \in \mathbb{R}_{\geqslant 0}, I \in \mathcal{I}(\mathbb{R}_{\geqslant 0})$*:*

$$f(s,I,z) = \int_0^z E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s' \in S} \mathbb{P}[s,s'] \cdot F(s', I \ominus \tau) \, \mathrm{d}\tau$$

It turns out that there exists an optimal scheduler for Problem 1 that has a very simple structure. Namely, it only needs to know the current state and the amount of time passed from the beginning. We start with a definition.

Definition 4.1.1. *We say that a generic measurable scheduler $\pi \in \Pi_\mu$ is* timed-memoryless *if*

– *π is not randomized, i. e. $\forall \rho \in Paths^*$, such that $\rho{\downarrow} \in PS$, $\pi(\rho) = \Delta_{Act}(\alpha)$, for some $\alpha \in Act(\rho{\downarrow})$.*

– *if $\rho_1, \rho_2 \in Paths^*$, such that $\rho_1{\downarrow} \in PS$, $\rho_1{\downarrow} = \rho_2{\downarrow}$ and $\tau_{\text{total}}(\rho_1) = \tau_{\text{total}}(\rho_2)$, then $\pi(\rho_1) = \pi(\rho_2)$.*

The set of all timed-memoryless schedulers is denoted with $\Pi_{\mathtt{TM}}$.

Notice that a timed-memoryless scheduler π can be equivalently defined as $\pi : PS \times \mathbb{R}_{\geqslant 0} \to Act$. Here the first component refers to the current state and the second one to the total amount of time passed from the beginning.

Lemma 4.1.3. *Given that $\Phi_2 \Rightarrow \Phi_1$:*

– *there exists a timed-memoryless scheduler that is optimal for $\mathrm{val}^{\mathcal{M}_\ell}_{\text{opt}}(\psi), \psi = \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2$,*

– *the function $\mathrm{val}^{\mathcal{M}_\ell}_{\text{opt}}(s, \psi \ominus t)$ is a continuous function of t on intervals $[0, a]\!]_a, [\![a, b]\!], [\![_b b, \infty)$, where $\psi \ominus t := \Phi_1 \,\mathrm{U}^{[\![a,b]\!] \ominus t} \Phi_2$, $[\![_z$ is an open bound if $z \in [\![a, b]\!]$ and is a closed bound otherwise, and analogously for $]\!]_z$.*

Proof. The proof of this lemma resembles proofs of similar results for CTMDPs (see [Neu10], Lemma 5.2 and Theorem 5.2 and [Fu14] Theorem 5).

Due to Lemma 4.1.1 for any scheduler $\pi \in \Pi_\mu$ and state $s \in S$:

$$\mathrm{val}^{\mathcal{M}_\ell}_\pi(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}^{\mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}}_\pi(s, \mathtt{tt} \,\mathrm{U}^{[\![a,b]\!]} \Phi_2)$$

Therefore w. l. o. g. we assume that $\Phi_1 = \mathtt{tt}$. Let $E(t, n) = \{\rho \in Paths^\omega \mid \mathrm{prefix}^\phi_n(\rho) \cdot \rho[n] \models \psi \ominus t\}$, (see the satisfaction relation defined for finite paths in Lemma 4.1.1, (4.4)). We define $p^n_{\text{opt}}(s, t) = \mathrm{opt}_{\pi \in \Pi_\mu} \Pr^{\mathcal{M}}_{\pi,s}[E(t, n)]$. Notice that

$$\begin{aligned}\lim_{n\to\infty} E(t, n) &= \lim_{n\to\infty} \{\rho \in Paths^\omega \mid \mathrm{prefix}^\phi_n(\rho) \cdot \rho[n] \models \psi \ominus t\} \\ &= \{\rho \in Paths^\omega \mid \rho \models \psi \ominus t\}\end{aligned}$$

Due to [ADD00], Theorem 1.2.7(a), for all $\pi \in \Pi_\mu, s \in S$:

$$\lim_{n\to\infty} \Pr^{\mathcal{M}}_{\pi,s}[E(t, n)] = \Pr^{\mathcal{M}}_{\pi,s}[\{\rho \in Paths^\omega \mid \rho \models \psi \ominus t\}] = \mathrm{val}^{\mathcal{M}_\ell}_\pi(s, \psi \ominus t)$$

Let $\pi_{\mathtt{opt}}$ be a scheduler that achieves optimum for $\mathrm{val}^{\mathcal{M}_\ell}_{\text{opt}}(\psi \ominus t)$. Since the latter equality holds for all schedulers, then the same holds for scheduler $\pi_{\mathtt{opt}}$. Consider the sequence of values $\{p^n_{\text{opt}}(s, t)\}_{n \in \mathbb{Z}_{\geqslant 0}}$. For each $n \in \mathbb{Z}_{\geqslant 0}$ the value $p^n_{\text{opt}}(s, t)$ is bounded by $\mathrm{val}^{\mathcal{M}_\ell}_{\text{opt}}(s, \psi \ominus t)$ from above if opt = sup, and from below for opt = inf. On the other hand, since the value $p^n_{\text{opt}}(s, t)$ is optimal, then $p^n_{\text{opt}}(s, t) \succcurlyeq_{\text{opt}} \Pr^{\mathcal{M}}_{\pi_{\text{opt}},s}[E(t, n)]$.

Since the sequence $\{\mathrm{Pr}^{\mathcal{M}}_{\pi_{\mathrm{opt}},s}[E(t,n)]\}$ converges to $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi \ominus t)$, then the sequence $\{p^n_{\mathrm{opt}}(s,t)\}_{n\in\mathbb{Z}_{\geqslant 0}}$ converges to $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi \ominus t)$:

$$\lim_{n\to\infty} p^n_{\mathrm{opt}}(s,t) = \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi \ominus t) \tag{4.6}$$

Since the set of actions in a Markov automaton is finite, we can define a total order over the actions. Let $\prec$ be any such order. For $t \in \mathbb{R}_{\geqslant 0}, n, m \in \mathbb{Z}_{\geqslant 0}$ we define a scheduler $\pi^{t,m}_n : PS \times \mathbb{R}_{\geqslant 0} \times \mathbb{Z}_{\geqslant 0} \to Act$, such that $\forall ps \in PS, x \in \mathbb{R}_{\geqslant 0}, k \in \mathbb{Z}_{\geqslant 0}$:

$$\pi^{t,m}_n(ps,x,k) = \begin{cases} \min\limits_{\prec}\{\alpha \in Act(ps)\} & \text{if } t + x > b \text{ or} \\ & m + k + 1 > n \\ \min\limits_{\prec}\left\{\alpha \in \arg\underset{\alpha\in Act(ps)}{\mathrm{opt}} \sum\limits_{s'\in S} \mathbb{P}[ps,\alpha,s'] \cdot p^{n-m-k-1}_{\mathrm{opt}}(s',t+x)\right\} & \text{otherwise} \end{cases}$$

Here parameter n refers to the maximal amount of transitions that can be performed. Parameters t and m denote the amount of time passed and the number of transitions performed from the beginning of the system run.

We also define a value $q^n(s,t) = \mathrm{Pr}^{\mathcal{M}}_{\pi^{t,0}_n,s}[E(t,n)]$. As the next step we will show that

$$\lim_{n\to\infty} q^n(s,t) = \lim_{n\to\infty} p^n_{\mathrm{opt}}(s,t) \tag{4.7}$$

The proof proceeds by induction over n. Let $n = 0$. If $s \in MS \cap \mathrm{Sat}(\Phi_2)$, then $p^0_{\mathrm{opt}}(s,t) = e^{-E(s)\cdot(a-t)}$ if $t \leqslant a$, $p^0_{\mathrm{opt}}(s,t) = 1$ if $t > a \wedge [\![a,b]\!] \ominus t \neq \emptyset$ and $p^0_{\mathrm{opt}}(s,t) = 0$ otherwise. If $s \in PS \cap \mathrm{Sat}(\Phi_2)$ then $p^0_{\mathrm{opt}}(s,t) = 1$ if $0 \in [\![a,b]\!] \ominus t$ and $p^0_{\mathrm{opt}}(s,t) = 0$ otherwise. If $s \in \mathrm{Sat}(\neg\Phi_2)$ then $\forall t \in \mathbb{R}_{\geqslant 0} : p^0_{\mathrm{opt}}(s,t) = 0$. In all these cases the value of $p^0_{\mathrm{opt}}(s,t)$ does not depend on any scheduler and therefore $\forall s \in S, t \in \mathbb{R}_{\geqslant 0} : p^0_{\mathrm{opt}}(s,t) = q^0(s,t)$.

Consider $n > 0, s \in MS \cap \mathrm{Sat}(\neg\Phi_2)$. Then

$$\begin{aligned} p^n_{\mathrm{opt}}(s,t) &= \Omega(p^{n-1}_{\mathrm{opt}}(s,t)) \\ &= \int_0^{b-t} E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s'] \cdot p^{n-1}_{\mathrm{opt}}(s',t+\tau)\,\mathrm{d}\tau \\ &= \int_0^{b-t} E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s'] \cdot q^{n-1}(s',t+\tau)\,\mathrm{d}\tau \\ &= \int_0^{b-t} E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s'] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi^{t+\tau,0}_{n-1},s'}[E(t+\tau,n-1)]\,\mathrm{d}\tau \\ &= \int_0^{b-t} E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s'] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi^{t+\tau,1}_{n},s'}[E(t+\tau,n-1)]\,\mathrm{d}\tau \\ &= \mathrm{Pr}^{\mathcal{M}}_{\pi^{t,0}_n,s}[E(t,n)]\,\mathrm{d}\tau = q^n(s,t) \end{aligned}$$

And analogously for $s \in MS \cap \mathrm{Sat}(\Phi_2)$. Consider $s \in PS \cap \mathrm{Sat}(\Phi_2)$. If $0 \in [\![a,b]\!] \ominus t$, then $p^n_{\mathrm{opt}}(s,t) = 1 = q^n(s,t)$. Otherwise:

$$\begin{aligned}
p^n_{\mathrm{opt}}(s,t) &= \Omega(p^{n-1}_{\mathrm{opt}}(s,t)) = \underset{\alpha \in Act(s)}{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[s,\alpha,s'] \cdot p^{n-1}_{\mathrm{opt}}(s',t) \\
&= \sum_{s' \in S} \mathbb{P}[s,\pi^{t,0}_n(s,0,0),s'] \cdot p^{n-1}_{\mathrm{opt}}(s',t) = \sum_{s' \in S} \mathbb{P}[s,\pi^{t,0}_n(s,0,0),s'] \cdot q^{n-1}(s',t) \\
&= \sum_{s' \in S} \mathbb{P}[s,\pi^{t,0}_n(s,0,0),s'] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi^{t,0}_{n-1},s'}[E(t,n-1)] \\
&= \sum_{s' \in S} \mathbb{P}[s,\pi^{t,0}_n(s,0,0),s'] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi^{t,1}_{n},s'}[E(t,n-1)] \\
&= \mathrm{Pr}^{\mathcal{M}}_{\pi^{t,0}_n,s}[E(t,n)] = q^n(s,t)
\end{aligned}$$

The induction above shows that $\forall n \in \mathbb{Z}_{\geqslant 0} : p^n_{\mathrm{opt}}(s,t) = q^n(s,t)$ and therefore

$$\lim_{n\to\infty} q^n(s,t) = \lim_{n\to\infty} p^n_{\mathrm{opt}}(s,t) = \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s, \psi \ominus t)$$

Next we will show that $\lim_{n\to\infty} q^n(s,t) = \mathrm{val}^{\mathcal{M}_\ell}_{\pi^t_*}(s, \psi \ominus t)$, where

$$\begin{aligned}
&\pi^t_*(ps,x) \\
&= \begin{cases} \min\limits_{\prec}\{\alpha \in Act(ps)\} & \text{if } t+x > b \\ \min\limits_{\prec}\left\{\alpha \in \arg \underset{\alpha \in Act(ps)}{\mathrm{opt}} \sum\limits_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s', \psi \ominus (t+x))\right\} & \text{otherwise} \end{cases}
\end{aligned}$$

We have mentioned before that $\lim_{n\to\infty} E(t,n) = \{\rho \in Paths^\omega \mid \rho \models \psi \ominus t\}$. It is left to prove that $\lim_{n\to\infty} \pi^{t,0}_n = \pi^t_*$. For $t,x \in \mathbb{R}_{\geqslant 0}$, s. t. $t+x > b$ schedulers $\pi^{t,0}_n$ and π^t_* choose the minimal action (w. r. t. $\prec$) from the set $Act(ps)$ and thus the selected actions are the same. Consider such values of t and x, that $t + x \leqslant b$, $ps \in PS, k \in \mathbb{Z}_{\geqslant 0}, k+1 \leqslant n$. Then

$$\begin{aligned}
&\lim_{n\to\infty} \underset{\alpha \in Act(ps)}{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot p^{n-k-1}_{\mathrm{opt}}(s',t+x) \\
&= \underset{\alpha \in Act(ps)}{\mathrm{opt}} \lim_{n\to\infty} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot p^{n-k-1}_{\mathrm{opt}}(s',t+x) \\
&= \underset{\alpha \in Act(ps)}{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot \lim_{n\to\infty} p^{n-k-1}_{\mathrm{opt}}(s',t+x) \\
&= \underset{\alpha \in Act(ps)}{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s', \psi \ominus (t+x))
\end{aligned}$$

Therefore the following sets are equal:

$$\begin{aligned}
&\arg \underset{\alpha \in Act(ps)}{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot p^{n-k-1}_{\mathrm{opt}}(s',t+x) \\
&=
\end{aligned}$$

$$\arg \operatorname*{opt}_{\alpha\in Act(ps)} \sum_{s'\in S} \mathbb{P}[ps,\alpha,s'] \cdot \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s', \psi \ominus (t+x))$$

Since the sets of actions coincide, so do their minimums w. r. t. order $\prec$ and therefore in this case the actions selected by the two schedulers coincide. In the limit when $n \to \infty$ condition $k+1 > n$ never happens and therefore $\lim_{n\to\infty} \pi^{t,0}_n(ps,x,k) = \pi^t_*(ps,x)$.

We have thus shown that $\forall t \in \mathbb{R}_{\geqslant 0}$:

$$\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi\ominus t) = \lim_{n\to\infty} p^n_{\mathrm{opt}}(s,t) = \lim_{n\to\infty} q^n(s,t) = \mathrm{val}^{\mathcal{M}_\ell}_{\pi^t_*}(s,\psi\ominus t)$$

Therefore the scheduler π^0_* is optimal for $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(\psi)$. Next we will show that function $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(\psi)$ is piecewise-continuous within intervals $I_0 = [0,a]\!]_a, I_1 = [\![a,b]\!], I_2 = [\![_b b,\infty)$. Lemma A.2 shows that $\forall n \in \mathbb{Z}_{\geqslant 0}, s \in S$ functions $p^n_{\mathrm{opt}}(s,t)$ are piecewise-continuous in t, and more specifically, that:

$$\forall s \in S, \delta > 0, t,t' \in I_m, m \in \{0,1,2\}, |t-t'| < \delta,$$
$$\exists C \in \mathbb{R}_{\geqslant 0} : \left|p^n_{\mathrm{opt}}(s,t) - p^n_{\mathrm{opt}}(s,t')\right| \leqslant C \cdot \delta$$

Taking the limit from both sides of the latter equation and taking into account (4.6) we conclude that the same holds for $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi\ominus t)$, what proves the second statement of the lemma.

It is left to show that the scheduler is measurable. As we have just shown, function $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi\ominus t)$ is piecewise-continuous as a function of t, which implies that the function is measurable w. r. t. the Lebesgue measure on the Borel sigma-algebra $\mathcal{B}(\mathbb{R}_{\geqslant 0})$. Therefore for any $s \in PS, \alpha \in Act(s)$ the function $f(s,t,\alpha) = \sum_{s'\in S} \mathbb{P}[s,\alpha,s'] \cdot \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s',\psi\ominus t)$ is a continuous function of t and thus measurable. Scheduler π^0_* selects such an action, that achieves an optimum over $\alpha \in Act(s)$ of values $f(s,t,\alpha)$. Since the number of enabled actions is finite and maximum/minimum of a measurable function is measurable, then the function $\mathrm{opt}_{\alpha\in Act(s)} f(s,t,\alpha)$ is measurable. The order $\prec$ ensures that a unique action is selected and therefore scheduler π^0_* is measurable and timed-memoryless.

□

In the next step we will show that one can reduce the solution of Problem 1 for formula $\mathtt{tt}\,\mathrm{U}^{[\![a,b]\!]}\Phi_2$ to solving at most two instances of Problem 1 for a formula $\mathtt{tt}\,\mathrm{U}^{=c}\Phi_2$. We start with defining the latter as a separate value.

Time-bounded Reachability for a Goal Function. First of all, we introduce the notion of a *goal function.* Reachability properties of CSL denote the probability to reach (under certain restrictions) any state from a given subset of the total state-space. The latter is usually called the *goal set* and states from this set are called *goal states.* For the problems that we consider in this chapter we need a possibility to assign to goal states arbitrary weights within $[0,1]$ range. To achieve this we introduce the notion of a *goal function*:

Definition 4.1.2. *We say that function g is a* goal function *if it is a possibly partial function with a non empty domain and such that $g(s) \in [0,1]$ for each state s in the domain of g.*

Goal functions generalise the notion of *goal states* in the following way. If g is defined for a state s, then s is a goal state, even if $g(s) = 0$. If g is undefined for a state s, then s is not a goal state.

Definition 4.1.3. *Let $\mathcal{M}$ be a Markov automaton, $s \in S$, $\pi \in \Pi_\mu$, $b \in \mathbb{R}_{\geqslant 0}$ and g be a goal function, such that $s \in dom(g) \Rightarrow s$ is Markovian. The* (time-bounded) reachability probability (of state s for the goal function g) *is defined as follows:*

$$\mathrm{val}_\pi^{\mathcal{M}}(s, b, g) := \sum_{s' \in dom(g)} \mathrm{Pr}_{\pi,s}^{\mathcal{M}} \left[\mathtt{tt} \, \mathrm{U}^{=b} ap_{s'} \right] \cdot g(s')$$

The optimal (time-bounded) reachability probability *is defined as:*

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, b, g) := \mathrm{opt}_{\pi \in \Pi_\mu} \mathrm{val}_\pi^{\mathcal{M}}(s, b, g) \tag{4.8}$$

We denote by $\mathrm{val}_\pi^{\mathcal{M}}(b, g)$ the vector of values $\mathrm{val}_\pi^{\mathcal{M}}(s, b, g)$ for all states $s \in S$ and analogously the vector of optimal values is denoted as $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b, g)$. For the above definitions we will omit the superscript $\mathcal{M}$ whenever the Markov automaton under consideration is clear from the context. The definition of (ϵ-)optimal scheduler for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b, g)$ is analogous to that of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi)$.

The fixpoint characterisation of value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b, g)$ is obtained from the fixpoint characterisation of value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\mathtt{tt} \, \mathrm{U}^{=b} \Phi_2)$ (Lemma 4.1.2) by substituting the value 1, assigned to each goal state s whenever the formula is satisfied, to value $g(s)$. Below we write it down explicitly:

Lemma 4.1.4 (Fixpoint Characterisation)**.** *Let $\mathcal{M}$ be an MA, g be a goal function, then $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, b, g)$ is a function of $b \in \mathbb{R}_{\geqslant 0}$, which is the least fixpoint of the higher-order operator $\Omega : (S \times \mathbb{R}_{\geqslant 0} \to [0,1]) \to (S \times \mathbb{R}_{\geqslant 0} \to [0,1])$, where*

$$\Omega(F)(s,t) = \begin{cases} g(s) & \text{if } t = 0 \wedge s \in dom(g) \\ e^{-E(s) \cdot t} \cdot g(s) \cdot \mathbb{1}_{dom(g)}(s) + & \text{if } t > 0 \wedge s \in MS \\ \quad \int_0^t E(s) \cdot e^{-E(s) \cdot \tau} \sum_{s' \in S} \mathbb{P}[s, s'] \cdot F(s', t - \tau) \, \mathrm{d}\tau & \\ \quad \mathrm{opt}_{\alpha \in Act(s)} \sum_{s' \in S} \mathbb{P}[s, \alpha, s'] \cdot F(s', t) & \text{if } s \in PS \wedge \\ & (t > 0 \vee s \notin dom(g)) \\ 0 & \text{in all other cases,} \end{cases} \tag{4.9}$$

Analogously to Lemma 4.1.3 it can be shown that there exist an optimal timed-memoryless scheduler that attains values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b, g)$:

Lemma 4.1.5. *There exists a timed-memoryless scheduler that is optimal for* $\mathrm{val}_{\mathrm{opt}}(b, g)$*.*

In the following we will show that computing $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\mathtt{tt}\,\mathrm{U}^{[\![a,b]\!]}\Phi_2)$ can be reduced to computing at most two instances of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s,c,g)$. The solution will depend on the exact values of a, b and the bounds of the interval. We will consider separately various combinations of these values.

We will often use the goal function that assigns value 1 to all the states that satisfy Φ_2 and is undefined otherwise. Formally g_{Φ_2} is a goal function, such that

$$g_{\Phi_2}(s) := \begin{cases} 1 & \text{if } s \in \mathrm{Sat}(\Phi_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Case $\psi = \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2, b \geqslant 0$

Lemma 4.1.6. *For any $\pi \in \Pi_\mu, s \in S, b \in \mathbb{R}_{\geqslant 0}$:*

$$\mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2) = \mathrm{val}_\pi^{\mathcal{M}^{[\Phi_2]}}(s, b, g_{\Phi_2})$$

Proof. First of all, notice that since all states in $\mathrm{Sat}(\Phi_2)$ are Markovian in $\mathcal{M}^{[\Phi_2]}$, then $\mathrm{Pr}_{\pi,s}^{\mathcal{M}^{[\Phi_2]}}\left[\mathtt{tt}\,\mathrm{U}^{=b}\Phi_2\right]$ is the transient probability of being in one of the states that satisfy Φ_2 at time b and therefore:

$$\begin{aligned}\mathrm{val}_\pi^{\mathcal{M}_\ell^{[\Phi_2]}}(s, \mathtt{tt}\,\mathrm{U}^{=b}\Phi_2) &= \mathrm{Pr}_{\pi,s}^{\mathcal{M}^{[\Phi_2]}}\left[\mathtt{tt}\,\mathrm{U}^{=b}\Phi_2\right] \\ &= \sum_{s\in\mathrm{Sat}(\Phi_2)} \mathrm{Pr}_{\pi,s}^{\mathcal{M}^{[\Phi_2]}}\left[\mathtt{tt}\,\mathrm{U}^{=b} ap_s\right] = \mathrm{val}_\pi^{\mathcal{M}^{[\Phi_2]}}(s, b, g_{\Phi_2})\end{aligned}$$

It therefore suffices to show that $\mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2) = \mathrm{val}_\pi^{\mathcal{M}_\ell^{[\Phi_2]}}(s, \mathtt{tt}\,\mathrm{U}^{=b}\Phi_2)$.

First consider the case of $[0,b]\!] = [0,b]$. In this case the proof is analogous to respective results for similar models, e.g. CTMC [BHHK03]. If a finite path reaches a state satisfying Φ_2 in $\mathcal{M}_\ell$ within time $[0,b]$, then the same path is possible in $\mathcal{M}_\ell^{[\Phi_2]}$. Since states in $\mathrm{Sat}(\Phi_2)$ are absorbing in $\mathcal{M}_\ell^{[\Phi_2]}$, then they are Markovian. Thus upon entering such a state ms at time point $t \leqslant b$ the path never leaves ms and with probability 1 the total time will reach b. Similarly in the other direction. If a finite path ρ in $\mathcal{M}_\ell^{[\Phi_2]}$ passes through a state $s \in \mathrm{Sat}(\Phi_2)$ at time b, then there exists a time point $t \leqslant b$ at which this state has been entered for the first time. Then a finite path $\rho' = \mathrm{prefix}_t^\phi(\rho) \cdot s$ is also a finite path in $\mathcal{M}_\ell$ and any of its infinite extensions from $Cyl(\rho')$ satisfy $\mathtt{tt}\,\mathrm{U}^{[0,b]}\Phi_2$. The probabilities of the two events depend only on what happened before a state in Φ_2 is encountered for the first time, and those prefixes coincide for the two considered events.

The case of $[0,b)$ is reduced to $[0,b]$ due to the fact that the probability of performing a Markovian transition within a non-empty interval $[\![c,d]\!]$ is the same as the probability to perform it within interval $[c,d]$, for any $c,d \in \mathbb{R}_{\geqslant 0}$.

□

Case $\psi = \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2, b > 0$

The main idea to solve this case is to reduce the problem to the previous one, i. e. to solving the problem for interval $[0, b]\!]$. However just changing the interval from $(0, b]\!]$ to $[0, b]\!]$ is not correct in general. For example, if there exist probabilistic states that satisfy Φ_2, then paths that reach such states at time point 0 satisfy $\mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2$, however they do not necessarily satisfy $\mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2$. Markovian states do not pose the same problem because the probability to leave a Markovian state immediately, i. e. after residing 0 time units, is 0. Thus paths that reach at time point 0 a Markovian state that satisfy Φ_2 will satisfy $\mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2$ with probability 1.

We solve the issue posed by probabilistic states that satisfy Φ_2 by creating new probabilistic states that are exact copies of the original ones - i. e. have the same outgoing transitions - however do not have any of their labels. This way if a probabilistic state ps satisfies formula Φ_2, then its copy, denoted with $cp(ps)$, does not satisfy Φ_2. If a path from $cp(ps)$ satisfies $\mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2$, then the respective path from ps satisfies $\mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2$ and vice versa.

Formally, the state-space with new copy states is defined as $S \uplus S^{cp}$, where $S^{cp} = \{ps^{cp} \mid ps \in PS_{\mathcal{M}}\}$. If $s \in PS_{\mathcal{M}}$ we will denote with $cp(s)$ its copy state s^{cp} , and if $s \in MS_{\mathcal{M}}$, then $cp(s) = s$. The Markov automaton with new copy states is defined as follows: $cp(\mathcal{M}_\ell) := (cp(\mathcal{M}), cp(AP), cp(lab))$. Here $cp(\mathcal{M}) = (S \uplus S^{cp}, Act, \dashrightarrow^{cp}, \mathbf{R})$, $\dashrightarrow^{cp} = \dashrightarrow \uplus \{(ps^{cp}, \alpha, \mu^{cp}) \mid (ps, \alpha, \mu) \in \dashrightarrow\}$, $\mu^{cp} = [cp(s) \to \mu(s) \mid s \in S]$, $cp(AP) = AP \uplus \{ap_{ps^{cp}} \mid ps^{cp} \in S^{cp}\}$, $\forall s \in S$: $cp(lab)(s) = lab(s)$ and $\forall ps^{cp} \in S^{cp} : cp(lab)(ps^{cp}) = \{ap_{ps^{cp}}\}$.

For a scheduler $\pi \in \Pi^{\mathcal{M}}_{\mathtt{TM}}$ we define a scheduler $cp(\pi) \in \Pi^{cp(\mathcal{M})}_{\mathtt{TM}}$ as follows: $\forall ps \in PS_{\mathcal{M}}, t \in \mathbb{R}_{\geqslant 0} : cp(\pi)(ps, t) = \pi(ps, t)$ and $\forall ps^{cp} \in S^{cp}, t \in \mathbb{R}_{\geqslant 0} : cp(\pi)(ps^{cp}, t) = \pi(ps, t)$.

Lemma 4.1.7. *For any timed-memoryless $\pi, s \in S$ and $b \in \mathbb{R}_{>0}$:*

$$\mathrm{val}^{\mathcal{M}_\ell}_{\pi}(s, \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2) = \mathrm{val}^{\mathcal{M}^{cp,[\Phi_2]}}_{cp(\pi)}(cp(s), b, g_{\Phi_2}),$$

where $\mathcal{M}^{cp,[\Phi_2]} = (cp(\mathcal{M}))^{[\Phi_2]}$.

Proof. It suffices to prove that $\mathrm{val}^{\mathcal{M}_\ell}_{\pi}(s, \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2) = \mathrm{val}^{cp(\mathcal{M}_\ell)}_{cp(\pi)}(cp(s), \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2)$. Given this we can apply Lemma 4.1.6 to the latter and obtain

$$\mathrm{val}^{cp(\mathcal{M}_\ell)}_{cp(\pi)}(cp(s), \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2) = \mathrm{val}^{\mathcal{M}^{cp,[\Phi_2]}}_{cp(\pi)}(cp(s), b, g_{\Phi_2})$$

This proves the statement of the lemma.

Consider a set of paths $\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=0}\Phi_2\}$. We can use the following representation:

$$\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2\} = \{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2\} \cup \{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=0}\Phi_2\} \tag{4.10}$$

For Markovian states the probability of leaving the state immediately upon entry, i. e. after residence time 0, is 0. Therefore for a state $ms \in MS$ the measure

$\Pr^{\mathcal{M}}_{\pi,ms}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=0}\Phi_2\}\right] = 0 = \Pr^{cp(\mathcal{M})}_{cp(\pi),ms}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=0}\Phi_2\}\right]$. Due to (4.10) this implies that

$$\begin{aligned}\Pr^{\mathcal{M}}_{\pi,ms}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2\}\right] &= \Pr^{\mathcal{M}}_{\pi,ms}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2\}\right]\\ &= \Pr^{cp(\mathcal{M})}_{cp(\pi),ms}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2\}\right]\end{aligned}$$

Consider a probabilistic state $ps \in PS$ and formula

$$\Phi_2^{MS} = \bigvee_{s \in MS \cap \mathrm{Sat}(\Phi_2)} ap_s$$

Due to the discussion above the following holds:

$$\Pr^{\mathcal{M}}_{\pi,ps}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2^{MS}\}\right] = \Pr^{\mathcal{M}}_{\pi,ps}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2^{MS}\}\right]$$

However if there exists a probabilistic state $s \in \mathrm{Sat}(\Phi_2)$ reachable from ps in time 0 with non-zero probability, then it does not necessarily hold that

$$\Pr^{\mathcal{M}}_{\pi,ps}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2\}\right] = \Pr^{\mathcal{M}}_{\pi,ps}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2\}\right]$$

We will show that

$$\begin{aligned}\Pr^{\mathcal{M}}_{\pi,ps}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2\}\right] &= \Pr^{cp(\mathcal{M})}_{cp(\pi),cp(ps)}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2\}\right]\\ &= \Pr^{cp(\mathcal{M})}_{cp(\pi),cp(ps)}\left[\{\rho \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2\}\right]\end{aligned} \tag{4.11}$$

Let $\rho = s_0 \xrightarrow{\nu_0,t_0} s_1 \xrightarrow{\nu_1,t_1} \cdots$ be an infinite path in $\mathcal{M}$ and $i \in \mathbb{Z}_{\geqslant 0} \uplus \{\infty\}$ is the index of the first Markovian state appearing on the path, or ∞ if no such state exists. We define a one-to-one mapping from paths in $\mathcal{M}$ to those paths in $cp(\mathcal{M})$, that originate from a copy state as follows: $cp(\rho) = cp(s_0) \xrightarrow{\nu_0,t_0} cp(s_1) \cdots cp(s_{i-1}) \xrightarrow{\nu_{i-1},t_{i-1}} s_i \cdots$, i.e. all the probabilistic states appearing on the path before index i are substituted with their copies, while all the other states are preserved, as well as all the transitions selected along the path and time delays.

We start with proving the first equality of (4.11). If $\rho \models_{\mathcal{M}} \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2$, then there exists a state $s \in S$ and $t \in (0, b]\!]$, such that $s \models_{\mathcal{M}} \Phi_2$ and $s \in \rho@t$. Since $t > 0$, then at least one Markovian state appears on the path before time t and therefore $s \in cp(\rho)@t$ and consequently $cp(\rho) \models_{cp(\mathcal{M})} \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2$. And analogously in the other direction. Notice, that the probability measure $\Pr^{cp(\mathcal{M})}_{cp(\pi),cp(ps)}[\cdot]$ assigns non-zero probability only to those sets of paths that start from the copy state $cp(ps)$. Thus paths starting from a non-copy state ps in $cp(\mathcal{M})$ may satisfy $\mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2$, however have no effect on the probability value. This proves the statement.

Next we prove the second equality of (4.11). The set $\{\rho \in \mathit{Paths}^{\omega}(cp(\mathcal{M})) \mid \rho \models \mathtt{tt}\,\mathrm{U}^{[0,b]\!]}\Phi_2, \rho[0] \in S^{cp}\}$ may contain more paths than set $\{\rho \in \mathit{Paths}^{\omega}(cp(\mathcal{M})) \mid \rho \models \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2, \rho[0] \in S^{cp}\}$. Namely, there may be paths that start from a copy state and satisfy $\mathtt{tt}\,\mathrm{U}^{=0}\Phi_2$, however they do not satisfy $\mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2$. In $cp(\mathcal{M})$ copy states do not satisfy Φ_2 and therefore a path that starts from a copy state can only satisfy formula $\mathtt{tt}\,\mathrm{U}^{=0}\Phi_2$ after it has visited at least one Markovian state. Since residence times in Markovian states can take value 0 only with probability 0, then the measure of this set of paths is 0. This proves the second equality.

□

Case $\psi = \texttt{tt}\, \mathrm{U}^{[\![a,b]\!]}\Phi_2, b \geqslant a > 0$

For this case we will first show that analysing interval $[\![a,b]\!]$ and its closure $[a,b]$ produce the same results.

Lemma 4.1.8. *Let $\mathcal{M}_\ell$ be a labelled Markov automaton. For any timed-memoryless scheduler π, state $s \in S$:*

$$\mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \texttt{tt}\, \mathrm{U}^{[\![a,b]\!]}\Phi_2) = \mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \texttt{tt}\, \mathrm{U}^{[a,b]}\Phi_2) \tag{4.12}$$

Proof. At first we separate the Markovian part of Φ_2:

$$\Phi_2^{MS} = \bigvee_{s \in MS \cap \mathrm{Sat}(\Phi_2)} ap_s$$

Let $n \in \mathbb{Z}_{\geqslant 0}$ and $\forall i = 0..n : I_i \in \mathcal{I}(\mathbb{R}_{\geqslant 0})$, $S_i \subseteq S$. We define

$$\begin{aligned} & S_0 \xrightarrow{I_0} S_1 \xrightarrow{I_1} \cdots \xrightarrow{I_{n-1}} S_n \\ & = \{\rho = s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} \cdots \xrightarrow{\alpha_{n-1}, t_{n-1}} s_n \mid \forall i = 0..n : s_i \in S_i, t_i \in I_i, \\ & \qquad \alpha_i = \pi(s_i, \tau_{\mathrm{total}}(\rho, i) \text{ if } s_i \in PS, \alpha_i = E(s_i) \text{ if } s_i \in MS\} \end{aligned}$$

Let $\psi = \texttt{tt}\, \mathrm{U}^{[\![a,b]\!]}\Phi_2$, $\psi' = \texttt{tt}\, \mathrm{U}^{[a,b]}\Phi_2$. Consider the set of paths $\{\rho \in \mathit{Paths}^\omega \mid \rho \models \psi''\}, \psi'' \in \{\psi, \psi'\}$. This set can be represented as a countable union of sets $\mathit{Cyl}(S_0 \xrightarrow{I_0} S_1 \xrightarrow{I_1} \cdots \xrightarrow{I_{n-1}} S_n)$. To achieve this, first we need to split all the paths satisfying ψ'' according to the number of transitions performed until ψ'' is satisfied. Each of the latter sets can in turn be represented as a union over all possible intervals with rational bounds, at which transitions occur. Each possible transition time will be included in at least one of such intervals. All the paths that satisfy ψ'' within 0 transitions are: $E_0 = \mathit{Cyl}(\mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a,\infty)} S)$. There are no paths starting from probabilistic states because $0 \notin [a,b]$, and therefore $0 \notin [\![a,b]\!]$. All the paths that satisfy ψ'' within 1 transition are:

$$\begin{aligned} E_1 = {} & \mathit{Cyl}(MS \xrightarrow{I_{\psi''}} \mathrm{Sat}(\Phi_2)) \cup \mathit{Cyl}(PS \xrightarrow{[0,0]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a,\infty)} S) \\ & \cup \left(\bigcup_{\substack{r_1 < r_2 \trianglelefteq a \\ r_1, r_2 \in \mathbb{Q}_{\geqslant 0}}} \mathit{Cyl}(MS \xrightarrow{[r_1, r_2]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a - r_1, \infty)} S) \right), \end{aligned}$$

where for $\psi'' = \texttt{tt}\, \mathrm{U}^{[\![a,b]\!]}\Phi_2 : I_{\psi''} = [\![a,b]\!]$, for $\psi'' = \texttt{tt}\, \mathrm{U}^{[a,b]}\Phi_2 : I_{\psi''} = [a,b]$, and $\trianglelefteq = <$ if $a \in I_{\psi''}$, $\trianglelefteq = \leqslant$ otherwise.

And similarly we can define the sets of paths E_n that satisfy ψ'' within n transitions, $n \in \mathbb{Z}_{\geqslant 0}, n \geqslant 2$. Notice that these sets are not necessarily disjoint. This representation is analogous to the similar result for CTMCs [Pan09].

Next we will consider the difference between such a representation for formula ψ and the one for ψ' on the example of the sets of paths in E_1. The same argument extends to E_n. Notice that the difference between the two representations, one for formula ψ and the one for ψ', is in the transition intervals. More specifically, only in intervals of transition times for Markovian states. Namely, some of those intervals have closed bounds for ψ' and may have open bounds for ψ. Additionally, the intervals $[r_1, r_2]$ do not include point a if $a \in I_{\psi''}$ and do include this point otherwise. We will first show that whether a is included in intervals $[r_1, r_2]$ or not does not affect the probability measure. If $a \notin I_\psi$, then:

$$\begin{aligned}
&\bigcup_{\substack{r_1<r_2\leqslant a\\ r_1,r_2\in\mathbb{Q}_{\geqslant 0}}} \mathit{Cyl}(MS \xrightarrow{[r_1,r_2]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a-r_1,\infty)} S)\\
&= \bigcup_{\substack{r_1<r_2<a\\ r_1,r_2\in\mathbb{Q}_{\geqslant 0}}} \mathit{Cyl}(MS \xrightarrow{[r_1,r_2]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a-r_1,\infty)} S)\\
&\quad \cup \mathit{Cyl}(MS \xrightarrow{[a,a]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a-r_1,\infty)} S)
\end{aligned}$$

The same sets for ψ' do not include $\mathit{Cyl}(MS \xrightarrow{[a,a]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a-r_1,\infty)} S)$. Since the probability to perform a Markovian transition at exactly time point a is 0, then the set $\mathit{Cyl}(MS \xrightarrow{[a,a]} \mathrm{Sat}(\Phi_2^{MS}) \xrightarrow{(a-r_1,\infty)} S)$ has measure 0.

It is left to prove that the probability measure does not depend on whether the bounds of transition intervals are closed or open. First, notice that the probability measure of the union of two events A and B has the following property: $\mathrm{Pr}^{\mathcal{M}}_{\pi,s}[A \cup B] = \mathrm{Pr}^{\mathcal{M}}_{\pi,s}[A] + \mathrm{Pr}^{\mathcal{M}}_{\pi,s}[B] - \mathrm{Pr}^{\mathcal{M}}_{\pi,s}[A \cap B]$, and analogously for unions of more than two sets. Intersection of multiple sets $E_i = \mathit{Cyl}(S_0^i \xrightarrow{I_0^i} S_1^i \xrightarrow{I_1^i} \cdots \xrightarrow{I_{n-1}^i} S_n^i)$ is either empty, or is once again the set that can be represented as a union of sets $\mathit{Cyl}(S_0 \xrightarrow{I_0} S_1 \xrightarrow{I_1} \cdots \xrightarrow{I_{n-1}} S_n)$. Therefore we only need to show that the probability measure of the set $\mathit{Cyl}(S_0 \xrightarrow{I_0} S_1 \xrightarrow{I_1} \cdots \xrightarrow{I_{n-1}} S_n)$ does not depend on the type of bounds of intervals I_k for Markovian states. This, however, follows from the definition of the probability measure and properties of integrals. This proves the statement of the lemma.

□

Let π be a timed-memoryless scheduler and $a \in \mathbb{R}_{\geqslant 0}$. We define a scheduler π^{+a} as follows: $\forall s \in S, t \in \mathbb{R}_{\geqslant 0} : \pi^{+a}(s,t) := \pi(s, t+a)$.

Lemma 4.1.9. *For any timed-memoryless scheduler π, state $s \in S$:*

$$\mathrm{val}^{\mathcal{M}_\ell}_{\pi}(s, \mathtt{tt}\,\mathrm{U}^{[a,b]}\Phi_2) = \sum_{ms\in MS} \mathrm{val}^{\mathcal{M}_\ell}_{\pi}(s, \mathtt{tt}\,\mathrm{U}^{=a} ap_{ms}) \cdot \mathrm{val}^{\mathcal{M}_\ell^{[\Phi_2]}}_{\pi^{+a}}(ms, \mathtt{tt}\,\mathrm{U}^{=b-a}\Phi_2) \tag{4.13}$$

Proof. If a path ρ satisfies formula $\mathtt{tt}\,\mathrm{U}^{[a,b]}\Phi_2$, then $\tau_{\mathrm{total}}(\rho) \geqslant a$ and therefore all such paths visit some states at time point a. We will split all these paths into several sets, according to states that they pass through at time point a. For a state $s \in S$ we define the following event:

$$E_{=a}(s) := \{\rho \in \mathit{Paths}^\omega \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=a} ap_s\}$$

Let $ps \in PS$ and $\rho \in E_{=a}(ps)$. It is possible that a path that satisfies $\texttt{tt}\,\mathrm{U}^{=a} ap_{ps}$ does not visit any Markovian state starting from time point a. Since we only consider non-Zeno Markov automata, the measure of the event consisting of all such paths is 0. Consider $\rho \in E_{=a}(ps)$, such that there exists a Markovian state visited along ρ after time point a. The first Markovian state entered after leaving ps will be entered at the same time as ps, since residence times in probabilistic states are 0. Thus $\exists ms \in MS : \rho \in E_{=a}(ms)$. Therefore for every probabilistic state ps each path $\rho \in E_{=a}(ps)$ either belongs to some $E_{=a}(ms), ms \in MS$, or belongs to a set of measure 0.

Taking this into account we can represent the set of paths satisfying $\texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2$ as follows:

$$\begin{aligned}&\{\rho \in \mathit{Paths}^\omega \mid \rho \models \texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\}\\ &\quad= \bigcup_{ms \in MS}\Big(\{\rho \in \mathit{Paths}^\omega \mid \rho \models \texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\} \cap E_{=a}(ms)\Big)\\ &\qquad \cup \ (\text{all the rest of measure } 0)\end{aligned}$$

Next we consider the probability of this event:

$$\begin{aligned}&\Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\Big]\\ &\quad= \Pr^{\mathcal{M}_\ell}_{\pi,s}\left[\bigcup_{ms \in MS}\Big(\{\rho \in \mathit{Paths}^\omega \mid \rho \models \texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\} \cap E_{=a}(ms)\Big)\right] + 0\\ &\quad= \sum_{ms \in MS} \Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\{\rho \in \mathit{Paths}^\omega \mid \rho \models \texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\} \cap E_{=a}(ms)\Big]\\ &\quad- \sum_{\substack{ms_1, ms_2 \in MS\\ ms_1 \neq ms_2}} \Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\{\rho \in \mathit{Paths}^\omega \mid \rho \models \texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\} \cap E_{=a}(ms_1) \cap E_{=a}(ms_2)\Big]\\ &\quad+ \sum_{\substack{ms_1, ms_2, ms_3 \in MS\\ ms_1 \neq ms_2 \neq ms_3}} \cdots\end{aligned}$$

Notice that for $ms_1 \neq ms_2 \neq \cdots \neq ms_n$:

$$\Pr^{\mathcal{M}_\ell}_{\pi,s}\left[E_{=a}(ms_1) \cap \cdots \cap E_{=a}(ms_n)\right] = 0$$

Therefore

$$\begin{aligned}\Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2\Big] &= \sum_{ms \in MS} \Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2 \cap E_{=a}(ms)\Big]\\ &= \sum_{ms \in MS} \Pr^{\mathcal{M}_\ell}_{\pi,s}\left[E_{=a}(ms)\right] \cdot \Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2 \mid E_{=a}(ms)\Big]\\ &= \sum_{ms \in MS} \Pr^{\mathcal{M}_\ell}_{\pi,s}\left[\texttt{tt}\,\mathrm{U}^{=a} ap_{ms}\right] \cdot \Pr^{\mathcal{M}_\ell}_{\pi,s}\Big[\texttt{tt}\,\mathrm{U}^{[a,b]}\Phi_2 \mid \texttt{tt}\,\mathrm{U}^{=a} ap_{ms}\Big]\\ &= \sum_{ms \in MS} \Pr^{\mathcal{M}_\ell}_{\pi,s}\left[\texttt{tt}\,\mathrm{U}^{=a} ap_{ms}\right] \cdot \Pr^{\mathcal{M}_\ell}_{\pi^{+a},ms}\Big[\texttt{tt}\,\mathrm{U}^{[0,b-a]}\Phi_2\Big]\\ &\overset{\text{(Lemma 4.1.6)}}{=} \sum_{ms \in MS} \Pr^{\mathcal{M}_\ell}_{\pi,s}\left[\texttt{tt}\,\mathrm{U}^{=a} ap_{ms}\right] \cdot \Pr^{\mathcal{M}_\ell^{[\Phi_2]}}_{\pi^{+a},ms}\Big[\texttt{tt}\,\mathrm{U}^{=b-a}\Phi_2\Big]\end{aligned}$$

□

All the results shown above lead to the following conclusion:

Lemma 4.1.10. *Let $\psi = \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2$. If $\Phi_2 \Rightarrow \Phi_1$, then*

$$\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) = \begin{cases} \mathrm{val}^{\mathcal{M}^{[\neg\Phi_1 \vee \Phi_2]}}_{\mathrm{opt}}(s, b, g_{\Phi_2}) & \textit{if } [\![a,b]\!] = [0,b]\!] \\ \mathrm{val}^{\mathcal{M}^{cp}}_{\mathrm{opt}}(cp(s), b, g_{\Phi_2}) & \textit{if } [\![a,b]\!] = (0,b]\!] \\ \mathrm{val}^{\mathcal{M}^{[\neg\Phi_1 \wedge \neg\Phi_2]}}_{\mathrm{opt}}(s, a, g) & \textit{if } b \geqslant a > 0 \end{cases}$$

where

$$\mathcal{M}^{cp} = \Big(cp(\mathcal{M}^{[\neg\Phi_1 \wedge \neg\Phi_2]})\Big)^{[\Phi_2]}$$

$$g(s) = \begin{cases} \mathrm{val}^{\mathcal{M}^{[\neg\Phi_1 \vee \Phi_2]}}_{\mathrm{opt}}(s, b-a, g_{\Phi_2}) & \textit{if } s \in MS_{\mathcal{M}} \textit{ or } s \models \neg\Phi_1 \wedge \neg\Phi_2 \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Proof. Let Ψ be a state formula. Consider MA $\mathcal{M}$ and $\mathcal{M}^{[\Psi]}$. Notice that every timed-memoryless scheduler π of $\mathcal{M}$ is also a valid timed-memoryless scheduler in $\mathcal{M}^{[\Psi]}$, since by definition, the MA $\mathcal{M}^{[\Psi]}$ has only a subset of probabilistic states present in $\mathcal{M}$. On the other hand a timed-memoryless scheduler $\pi^{[\psi]}$ in $\mathcal{M}^{[\Psi]}$ does not define a scheduler in $\mathcal{M}$. Namely, if there exists a state s in $\mathcal{M}$, such that $s \in PS$ and $s \in \mathrm{Sat}(\Psi)$, then this state becomes Markovian in $\mathcal{M}^{[\Psi]}$. Thus scheduler $\pi^{[\psi]}$ is not defined for s and cannot be applied to $\mathcal{M}$. We will extend scheduler $\pi^{[\psi]}$ so that it becomes a valid scheduler in $\mathcal{M}$ as follows. Let $\prec$ be some total order on the set Act. We define a new scheduler $\pi^{[\psi]}_{ext}$ in $\mathcal{M}$, such that $\forall ps \notin \mathrm{Sat}(\Psi), t \in \mathbb{R}_{\geqslant 0} : \pi^{[\psi]}_{ext}(ps,t) = \pi^{[\psi]}(ps,t)$ and $\forall ps \in \mathrm{Sat}(\Psi), t \in \mathbb{R}_{\geqslant 0} : \pi^{[\psi]}_{ext}(ps,t) = \min_{\prec} Act(ps)$. Thus, on all the state-time pairs on which scheduler $\pi^{[\psi]}$ is defined, scheduler $\pi^{[\psi]}_{ext}$ takes the same action. And in all other cases it takes the minimal enabled action according to order $\prec$.

Let $\mathcal{M}'_\ell = \mathcal{M}^{[\neg\Phi_1 \wedge \neg\Phi_2]}_\ell$ and $\mathcal{M}' = \mathcal{M}^{[\neg\Phi_1 \wedge \neg\Phi_2]}$. Due to Lemma 4.1.1 the following holds:

$$\forall \pi \in \Pi^{\mathcal{M}_\ell}_{\mathtt{TM}} : \mathrm{val}^{\mathcal{M}_\ell}_{\pi}(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}^{\mathcal{M}'_\ell}_{\pi}(s, \mathtt{tt} \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) \tag{4.14}$$

Consider the case of $[\![a,b]\!] = [0,b]\!]$:

$$\begin{aligned} &\forall \pi \in \Pi^{\mathcal{M}}_{\mathtt{TM}} : \\ &\mathrm{val}^{\mathcal{M}_\ell}_{\pi}(s, \Phi_1 \,\mathrm{U}^{[0,b]\!]} \Phi_2) \overset{(4.14)}{=} \mathrm{val}^{\mathcal{M}'_\ell}_{\pi}(s, \mathtt{tt} \,\mathrm{U}^{[0,b]\!]} \Phi_2) \\ &\qquad \overset{\text{Lemma 4.1.6}}{=} \mathrm{val}^{\mathcal{M}'^{[\Phi_2]}}_{\pi}(s, b, g_{\Phi_2}) = \mathrm{val}^{\mathcal{M}^{[\neg\Phi_1 \vee \Phi_2]}}_{\pi}(s, b, g_{\Phi_2}) \end{aligned} \tag{4.15}$$

Let π^* be a scheduler in $\mathcal{M}$, that is optimal for $\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(\Phi_1 \,\mathrm{U}^{[0,b]\!]} \Phi_2)$. Then

$$\begin{aligned} \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s, \Phi_1 \,\mathrm{U}^{[0,b]\!]} \Phi_2) &= \mathrm{val}^{\mathcal{M}_\ell}_{\pi^*}(s, \Phi_1 \,\mathrm{U}^{[0,b]\!]} \Phi_2) \overset{(4.15)}{=} \mathrm{val}^{\mathcal{M}^{[\neg\Phi_1 \vee \Phi_2]}}_{\pi^*}(s, b, g_{\Phi_2}) \\ &\leqslant \mathrm{val}^{\mathcal{M}^{[\neg\Phi_1 \vee \Phi_2]}}_{\mathrm{opt}}(s, b, g_{\Phi_2}) \end{aligned}$$

Let π^{**} be a scheduler in $\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}$ that is optimal for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(b, g_{\Phi_2})$ and π^{**}_{ext} is its extension on $\mathcal{M}$, obtained as described above. Then:

$$\begin{aligned}\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(s, b, g_{\Phi_2}) &= \mathrm{val}_{\pi^{**}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(s, b, g_{\Phi_2}) = \mathrm{val}_{\pi^{**}_{ext}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(s, b, g_{\Phi_2})\\ &\overset{(4.15)}{=} \mathrm{val}_{\pi^{**}_{ext}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{[0,b]\!]}\Phi_2) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{[0,b]\!]}\Phi_2)\end{aligned}$$

Thus

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]}\Phi_2) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(s, b, g_{\Phi_2})$$

Consider the case of $[\![a, b]\!] = (0, b]\!]$. Due to Lemma 4.1.7:

$$\forall \pi \in \Pi_{\mathtt{TM}}^{\mathcal{M}_\ell} : \mathrm{val}_{\pi}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2) = \mathrm{val}_{cp(\pi)}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2}),$$

Analogously to the previous case we can show that:

$$\begin{aligned}\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{(0,b]\!]}\Phi_2) &= \mathrm{val}_{\pi^*}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{(0,b]\!]}\Phi_2) = \mathrm{val}_{cp(\pi^*)}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2})\\ &\leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2})\end{aligned}$$

Next we will prove the other direction: $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{(0,b]\!]}\Phi_2) \geqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2})$. Notice, that according to the fixpoint characterisation (4.9), the equations for the probabilistic states and their copy-states are the same, and therefore their optimal values and optimal actions that attain these values coincide. Thus, there exists an optimal timed-memoryless scheduler π^{**}_{cp} for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{cp}}(b, g_{\Phi_2})$, such that $\forall ps \in PS_{\mathcal{M}^{cp}}, t \in \mathbb{R}_{\geqslant 0} : \pi^{**}_{cp}(ps, t) = \pi^{**}_{cp}(cp(ps), t)$. Notice that π^{**}_{cp} is a valid scheduler in $\mathcal{M}'$, since $PS_{\mathcal{M}'} \subseteq PS_{\mathcal{M}^{cp}}$, and also $\pi^{**}_{cp} = cp(\pi^{**}_{cp})$. Let π^{**} be an extension of π^{**}_{cp} from $\mathcal{M}'$ to $\mathcal{M}$. Then:

$$\begin{aligned}\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2}) &= \mathrm{val}_{\pi^{**}_{cp}}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2}) = \mathrm{val}_{cp(\pi^{**}_{cp})}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2})\\ &\overset{\text{Lemma 4.1.7}}{=} \mathrm{val}_{\pi^{**}_{cp}}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2) = \mathrm{val}_{\pi^{**}}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{(0,b]\!]}\Phi_2)\\ &\overset{(4.14)}{=} \mathrm{val}_{\pi^{**}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{(0,b]\!]}\Phi_2) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{(0,b]\!]}\Phi_2)\end{aligned}$$

We have thus shown that

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{(0,b]\!]}\Phi_2) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{cp}}(cp(s), b, g_{\Phi_2})$$

Consider the case of $a > 0$. Due to (4.14) and Lemmas 4.1.8 and 4.1.9:

$$\begin{aligned}&\forall \pi \in \Pi_{\mathtt{TM}}^{\mathcal{M}'_\ell} :\\ &\quad \mathrm{val}_{\pi}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{[\![a,b]\!]}\Phi_2) = \mathrm{val}_{\pi}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{[a,b]}\Phi_2)\\ &\quad = \sum_{ms \in MS_{\mathcal{M}'_\ell}} \mathrm{val}_{\pi}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{=a} ap_{ms}) \cdot \mathrm{val}_{\pi+a}^{\mathcal{M}'^{[\Phi_2]}_\ell}(ms, \mathtt{tt}\,\mathrm{U}^{=b-a}\Phi_2)\\ &\quad = \sum_{ms \in MS_{\mathcal{M}'_\ell}} \mathrm{val}_{\pi}^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s, \mathtt{tt}\,\mathrm{U}^{=a} ap_{ms}) \cdot \mathrm{val}_{\pi+a}^{\mathcal{M}_\ell^{[\neg\Phi_1\vee\Phi_2]}}(ms, b-a, g_{\Phi_2})\\ &\quad = \mathrm{val}_{\pi}^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s, a, g_{\pi}),\end{aligned}$$

where

$$g_\pi(s) = \begin{cases} \mathrm{val}_{\pi^{+a}}^{\mathcal{M}_\ell^{[\neg\Phi_1\vee\Phi_2]}}(ms, b-a, g_{\Phi_2}) & \text{if } ms \in MS_{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Thus for $a > 0$:

$$\forall \pi \in \Pi_{\mathtt{TM}}^{\mathcal{M}_\ell} : \mathrm{val}_\pi^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}_\pi^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s, a, g_\pi),$$

Since this holds for all schedulers, it also holds for a scheduler π^* that is optimal for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2)$, and therefore:

$$\begin{aligned} \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}_{\pi^*}^{\mathcal{M}_\ell}(s, \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2) &= \mathrm{val}_{\pi^*}^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s, a, g_{\pi^*}) \\ \leqslant \mathrm{val}_{\pi^*}^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s, a, g) &\leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s, a, g) \end{aligned}$$

Let π_{in} be an optimal timed-memoryless scheduler for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(b-a, g_{\Phi_2})$. Since $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(b-a, g_{\Phi_2}) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}}(\mathtt{tt}\,\mathrm{U}^{=b-a}\Phi_2)$, then it is also optimal for the latter. Let $\pi_{in,ext}$ be an extension of π_{in} from $\mathcal{M}^{[\neg\Phi_1\vee\Phi_2]}$ to $\mathcal{M}^{[\neg\Phi_1\wedge\neg\Phi_2]}$ as described in the beginning of this proof. Consider a timed-memoryless scheduler π_{out} that is optimal for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}^{[\neg\Phi_1\wedge\neg\Phi_2]}}(a, g)$. We will build a new scheduler for $\mathcal{M}^{[\neg\Phi_1\wedge\neg\Phi_2]}$ by composing the scheduler for the inner problem with the one for the outer problem. Consider a timed-memoryless scheduler π in $\mathcal{M}^{[\neg\Phi_1\wedge\neg\Phi_2]}$, such that $\forall ps \in PS_{\mathcal{M}^{[\neg\Phi_1\wedge\neg\Phi_2]}}, t \in [a,b] : \pi(ps,t) = \pi_{in,ext}(ps, t-a)$, $\forall t \in [0, b-a) : \pi(ps,t) = \pi_{out}(ps,t)$. Then:

$$\begin{aligned} \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{[\neg\Phi_1\wedge\neg\Phi_2]}}(s,a,g) &= \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}'_\ell}(s,a,g) \\ &= \sum_{ms \in MS_{\mathcal{M}'_\ell}} \mathrm{val}_{\pi_{out}}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{=a} ap_{ms}) \cdot \mathrm{val}_{\pi_{in,ext}}^{\mathcal{M}'^{[\Phi_2]}_\ell}(ms, \mathtt{tt}\,\mathrm{U}^{=b-a}\Phi_2) \\ &= \sum_{ms \in MS_{\mathcal{M}'_\ell}} \mathrm{val}_{\pi}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{=a} ap_{ms}) \cdot \mathrm{val}_{\pi^{+a}}^{\mathcal{M}'^{[\Phi_2]}_\ell}(ms, \mathtt{tt}\,\mathrm{U}^{=b-a}\Phi_2) \\ &= \mathrm{val}_\pi^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{[\![a,b]\!]}\Phi_2) = \mathrm{val}_{\pi_{ext}}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\,\mathrm{U}^{[\![a,b]\!]}\Phi_2) \\ &\overset{(4.14)}{=} \mathrm{val}_{\pi_{ext}}^{\mathcal{M}_\ell}(s, \Phi_1\,\mathrm{U}^{[\![a,b]\!]}\Phi_2) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1\,\mathrm{U}^{[\![a,b]\!]}\Phi_2) \end{aligned}$$

This concludes the proof. □

The Lemma above shows that solution to Problem 1 can be obtained if given a way to compute the values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, b, g)$. This is going to be the topic of the following sections.

4.2 Optimal Time-Bounded Reachability Problem

We have discussed above that the problem of model-checking a time-bounded reachability formula can be reduced to computing a few instances of value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b, g)$. In this section we study the computation of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b,g)$ as a separate problem, reiterate over known facts related to $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b,g)$ and introduce new concepts that will be useful in the next section, where we present a novel solution for the computation of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b,g)$.

Problem 2 (Time-Bounded Reachability Problem)**.** *Let $\mathcal{M}$ be a Markov automaton, g be a goal function and $b \in \mathbb{R}_{\geqslant 0}$. Compute the values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b,g)$ as well as an optimal scheduler for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(b,g)$.*

4.2.1 Optimal Schedulers

Notice that in terms of reusability, it is more convenient to work with a scheduler that depends on *time left until the time bound*, rather than the *time passed from the beginning*. This way given a scheduler that is optimal for time bound a we can reuse this scheduler and its reachability values when facing the problem for time bound $b > a$. Consider a scheduler $\pi : PS \times \mathbb{R}_{\geqslant 0} \to Act$ that treats the second component as the time left until the time bound, not as the time passed from the beginning. There is a simple transformation from π to a scheduler π' that satisfies Definition 4.1.1: $\forall s \in PS, t \in [0,b] : \pi(s,t) = \pi'(s, b-t)$. For time points outside of interval $[0,b]$ scheduler π' can be defined arbitrarily. Since we will only be using these schedulers to solve Problem 2, the values taken outside of interval $[0,b]$ do not affect the reachability probabilities. Using this transformation we can define the probability measure for π as the probability measure of π', and analogously for other notions. In the following when working with Problem 2 we will be using this new type of timed-memoryless schedulers.

For some instances of Problem 2 the structure of optimal schedulers turns out to be quite simple. Namely, those schedulers are *piecewise-constant*:

Definition 4.2.1. *A timed-memoryless scheduler π is called* piecewise-constant *if there exists a finite partition $\mathcal{I}(\pi)$ of the time interval $\mathbb{R}_{\geqslant 0}$ into intervals $I_0 = [0,0], I_1 = (0,t_1], \cdots, I_{k-1} = (t_{k-2}, t_{k-1}], \ldots$, such that the value of the scheduler remains constant throughout each interval of the partition, i. e. $\forall I \in \mathcal{I}(\pi), \forall t_1, t_2 \in I, \forall s \in PS : \pi(s,t_1) = \pi(s,t_2)$.*
The set of all piecewise-constant schedulers is denoted by Π_{PC}.

We will denote the value of π on an interval $I \in \mathcal{I}(\pi)$ for $s \in PS$ by $\pi(s,I)$, i. e. $\pi(s,I) = \pi(s,t)$ for any $t \in I$.

Example 4.2.1. *Let s be the only probabilistic state of a Markov automaton, and $\alpha_1, \ldots, \alpha_n \in Act(s)$. The following is an example of a piecewise-constant scheduler:*

$$\pi(s,t) = \begin{cases} \alpha_0 & \text{if } t \in [0,0] \\ \alpha_1 & \text{if } t \in (0,1] \\ \cdots & \\ \alpha_n & \text{if } t \in (n-1,n] \\ \alpha_{n+1} & \text{if } t > n \end{cases}$$

It has been shown in [RS11] that an optimal piecewise-constant scheduler for Problem 2 exists if Markov automata are *PS*-acyclic and the goal function assigns value 1 to all the goal states. The result is shown for *Markov games*, a formalism that is more general than closed Markov automata. We conjecture that taking into account Lemma 4.1.5 these results can likely be extended to the general case of Problem 2. In the following, we will define a few key notions related to piecewise-constant schedulers.

Let π be a piecewise constant scheduler and $I \in \mathcal{I}(\pi)$. For convenience we will define a stationary scheduler $\pi|_I$ as follows: $\forall ps \in PS : \pi|_I(ps) := \pi(ps, I)$. We also define a concatenation operation that combines stationary schedulers into a piecewise-constant scheduler. Let $I_0, \ldots, I_n$ be a partition of $[0, \infty)$ and $\pi_0, \ldots, \pi_n \in \Pi_{\mathtt{stat}}$. We define

$$\pi_0|_{I_0} \cdot \pi_1|_{I_1} \cdots \pi_n|_{I_n} = \pi \in \Pi_{\mathtt{PC}}, \text{ s. t. } \forall k = 0..n : \pi|_{I_k} = \pi_k$$

And analogously for a piecewise-constant scheduler π' defined on $[0, a]$ and a stationary scheduler π'' we define the concatenation as follows: $\pi'|_{[0,a]} \cdot \pi''|_{(a,b]\!]} = \pi \in \Pi_{\mathtt{PC}}$, such that $\pi|_{[0,a]} = \pi'$ and $\pi|_{(a,b]\!]} = \pi''$.

An important notion of piecewise-constant schedulers is a *switching point*, the point of time separating two intervals of constant decisions:

Definition 4.2.2. *For a piecewise-constant scheduler π and $s \in PS$ we call $\tau \in \mathbb{R}_{\geqslant 0}$ a* switching point*, iff $\exists I_1, I_2 \in \mathcal{I}(\pi)$, s. t. $\tau = \sup I_1$ and $\tau = \inf I_2$ and $\exists s \in PS : \pi(s, I_1) \neq \pi(s, I_2)$.*

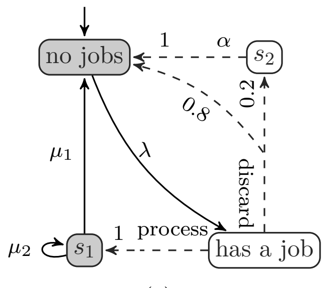
4.2.2 Hop-Unbounded Reachability Sub-problem

Very often the computation of the value $\mathrm{val}^{\mathcal{M}}_{\mathrm{opt}}(b, h)$ will stumble upon the sub-problem of computing the following values for $s \in S, \pi \in \Pi_\mu$ and a goal function g:

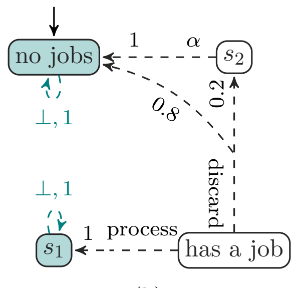
$$\begin{aligned} \mathrm{rea}^{\pi}(s, g) &:= \sum_{s' \in dom(g)} \mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[\Phi \,\mathrm{U}\, ap_{s'}\right] \cdot g(s'), \quad \text{where } \Phi = \bigvee_{s \in PS \setminus dom(g)} ap_s \\ \mathrm{rea}^{\mathrm{opt}}(s, g) &:= \operatorname*{opt}_{\pi \in \Pi_\mu} \mathrm{rea}^{\pi}(s, g) \end{aligned} \tag{4.16}$$

Here formula $\Phi \,\mathrm{U}\, ap_{s'}$ is satisfied by those infinite paths, that reach state s' via only probabilistic transitions and so that no other state $s'' \in dom(g)$ is visited before entering s'. The value $\mathrm{rea}^{\pi}(s, g)$ is the weighted sum of $g(s')$ and the probability to satisfy $\Phi \,\mathrm{U}\, ap_{s'}$ under scheduler π and when starting from state s. Value $\mathrm{rea}^{\mathrm{opt}}(s, g)$ is the optimal value $\mathrm{rea}^{\pi}(s, g)$ over all schedulers. We will denote an ε-close under-approximation of values $\mathrm{rea}^{\mathrm{opt}}(s, g)$ or $\mathrm{rea}^{\pi}(s, g)$ with $\mathrm{rea}^{\mathrm{opt}}_{\varepsilon}(s, g)$ and $\mathrm{rea}^{\pi}_{\varepsilon}(s, g)$ respectively.

The computation of values $\mathrm{rea}^{\pi}(s, g)$ and $\mathrm{rea}^{\mathrm{opt}}(s, g)$ can be reduced to solving two well-known problems on MDPs. First, we introduce an MDP for a Markov automaton that preserves these two values:



(a)



(b)

Figure 4.1: Fig. (4.1a) show an example of a Markov automaton and MDP $i\mathcal{D}(\mathcal{M}, A)$ for $A = \emptyset$ is shown in Figure (4.1b).

Definition 4.2.3. *Let* $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$ *be a Markov automaton and* $A \subseteq S$. *An* island MDP $i\mathcal{D}(\mathcal{M}, A)$ *is defined as follows* $i\mathcal{D}(\mathcal{M}, A) := (S, Act \uplus \{\bot\}, \mathbf{P})$, *where*

$$\mathbf{P}[s, \alpha, s'] = \begin{cases} \mathbb{P}[s, \alpha, s'] & \textit{if } s \in PS_{\mathcal{M}} \setminus A, \alpha \in Act(s) \\ 1 & \textit{if } s \in MS_{\mathcal{M}} \cup A, s' = s, \alpha = \bot \\ 0 & \textit{otherwise} \end{cases} \tag{4.17}$$

In the special case of $A = \emptyset$ *we will simply write* $i\mathcal{D}(\mathcal{M})$ *instead of* $i\mathcal{D}(\mathcal{M}, \emptyset)$.

The MDP $i\mathcal{D}(\mathcal{M}, A)$ characterises the behaviour of islands of probabilistic states in isolation from Markovian states, hence the name. An example of MDP $i\mathcal{D}(\mathcal{M}, A)$ for the MA shown in Figure (4.1a) and set $A = \emptyset$ is depicted in Figure (4.1b). The differences between the MA and the MDP are highlighted in green. Probabilistic states of the MA that are not in A are states of $i\mathcal{D}(\mathcal{M}, A)$ and their outgoing transitions in $\mathcal{M}$ and $i\mathcal{D}(\mathcal{M}, A)$ are the same. Markovian states of the MA and states in A are also states in the MDP however their outgoing transitions are different. Namely, they have only one self-loop transition with probability 1 via action $\bot$.

Consider formula Φ defined in (4.16). This formula ranges over only probabilistic states and additionally restricts to those that are not in the domain of g. The island MDP $i\mathcal{D}(\mathcal{M}, dom(g))$ mimics these restrictions. Here all the Markovian states as well as states in the domain of g are absorbing. We can therefore rewrite the value $\mathrm{rea}^{\pi}(s, g)$ in terms of the unbounded reachability probability in MDP $i\mathcal{D}(\mathcal{M}, A)$. Let $G = dom(g)$, then:

$$\begin{aligned} \mathrm{rea}^{\pi}(s, g) &= \sum_{s' \in dom(g)} \mathrm{Pr}^{\mathcal{M}}_{\pi,s}[\Phi \,\mathrm{U}\, ap_{s'}] \cdot g(s') = \sum_{s' \in dom(g)} \underbrace{\mathrm{Pr}^{i\mathcal{D}(\mathcal{M},G)}_{\pi,s}[\mathtt{tt} \,\mathrm{U}\, ap_{s'}]}_{\mathrm{rea}^{\pi,G}(s,s')} \cdot g(s') \\ &= \sum_{s' \in dom(g)} \mathrm{rea}^{\pi,G}(s, s') \cdot g(s') \end{aligned}$$

If the value of G is clear from the context, we will omit the superscript G in

$\mathrm{rea}^{\pi,G}(s,s')$ and simply write:

$$\mathrm{rea}^{\pi}(s,g) = \sum_{s' \in dom(g)} \mathrm{rea}^{\pi}(s,s') \cdot g(s') \tag{4.18}$$

Notice, that the term *unbounded reachability* is ambiguous when applied to Markov automata, since the word "unbounded" here may be applied to the progress of time or to the number of transitions performed. To avoid this issue, we will refer to values $\mathrm{rea}^{\pi}(g)$ and $\mathrm{rea}^{\mathrm{opt}}(g)$ as the *hop-unbounded reachability probability* and respectively *optimal hop-unbounded reachability probability.* Recall that "hop" is another way to address a probabilistic transition, that was introduced in Chapter 2 Section 2.2.

It has been shown in [Put94] that values $\mathrm{rea}^{\pi}(s,g)$ and $\mathrm{rea}^{\mathrm{opt}}(s,g)$ are the minimal solutions to the following systems of equations respectively:

$$\forall s \in S : f(s,\pi,g) = \begin{cases} g(s) & \text{if } s \in dom(g) \\ 0 & \text{else if } s \in MS \\ \sum_{s' \in S} \mathbb{P}[s,\pi(s),s'] \cdot f(s',\pi,g) & \text{otherwise} \end{cases} \tag{4.19}$$

$$\forall s \in S : f(s,g) = \begin{cases} g(s) & \text{if } s \in dom(g) \\ 0 & \text{else if } s \in MS \\ \mathrm{opt}_{\alpha \in Act(s)} \sum_{s' \in S} \mathbb{P}[s,\alpha,s'] \cdot f(s',g) & \text{otherwise} \end{cases} \tag{4.20}$$

Values $\mathrm{rea}^{\pi}(s,g)$ and $\mathrm{rea}^{\mathrm{opt}}(s,g)$ can be computed exactly by such techniques as policy iteration and linear programming [Put94], or approximated via interval value iteration [HM14] (with minor adaptations to generalise to arbitrary goal functions), [QK18, BKL+17].

We will denote with $C^{\mathrm{opt}}_{rea}(n,m,a)$ the worst-case complexity of computing values $\mathrm{rea}^{\mathrm{opt}}(g)$ in a Markov automaton with n states, m edges (edges were defined in Section 4.1.1) and $a = |Act|$ actions. And similarly, $C_{rea}(n,m,a)$ will denote the complexity of computing values $\mathrm{rea}^{\pi}(g)$. If the problem is formulated as a linear program, then the linear program has n variables and $n \cdot a$ constraints. Solving the system of equations (4.19) via Gaussian elimination is in the worst case in $C_{rea}(n,m,a) = O(n^3)$. If the MA is *PS*-acyclic, then the systems of equations (4.19-4.20) have triangular form and therefore $C^{\mathrm{opt}}_{rea}(n,m,a) = C_{rea}(n,m,a) = O(m)$.

4.3 A New Solution for Time-Bounded Reachability

In this section, we will develop a new approach for solving Problem 1. Due to Lemma 4.1.10 the problem can be reduced to solving two instances of Problem 2. In this section we will describe a new approach for approximating a solution for Problem 2 (approximation of $\mathrm{val}_{\mathrm{opt}}(b,g)$ and construction of the respective optimal scheduler).

Our approach is based on approximating an optimal scheduler by an ϵ-optimal piecewise-constant scheduler. We will first observe in Section 4.3.1 that if given a piecewise-constant scheduler, then the computation of the time-bounded reachability value induced by it is relatively straightforward. It can be reduced to a transient analysis on a time non-homogeneous CTMC and an unbounded reachability analysis on an MDP. The latter problem has been discussed in Section 4.2.2, together with efficient algorithms available for solving it. The former can be efficiently approximated by, for example, a technique called *uniformisation* [Jen53], extended to non-homogeneous CTMCs in [vMW98].

This shows that the main challenge of our approach to optimal time-bounded reachability analysis for MA is the computation of an ϵ-optimal piecewise-constant scheduler. As discussed, a piecewise-constant scheduler is characterised by its switching points and the actions that it takes in between the switching points. Computing either of the two is challenging. Results of Section 4.3.2 will provide an efficient solution to the latter, however, we still need to know the locations of the switching points on the timeline. This is taken care of in Section 4.3.3. There we will present a new characterisation of switching points in terms of function intersection. Since the switching points are not necessarily rational numbers, we need an efficient way to approximate them. This is going to be the topic of Section 4.3.4.

Finally, in Section 4.3.5 we present the resulting algorithm that computes an approximation of the solution of Problem 2. Under certain conditions it is guaranteed to provide an ϵ-close approximation for a given ϵ. And otherwise, it provides an a posteriori estimation of the approximation error.

Our approach is similar to the algorithm for CTMDPs presented in [BS11] and harvests several ideas developed there. On a high level, both algorithms follow the general outline of a theoretical (not-implementable) method presented in [Mil68]. To make it practical one needs an efficient method for (i) locating the switching points and (ii) approximating the value of the reachability probability for a piecewise-constant scheduler. The latter is solved with the help of uniformisation in case of [BS11], and a combination of uniformisation and hop-unbounded reachability analysis in case of our approach. To detect the location of switching points, the algorithm of [BS11] develops an upper- and a lower-bound on the reachability probability value and analyses the gap between the two. Our approach utilises two techniques to locate the switching points. First, we make use of the characterisation of the switching points developed in this section, based on function intersection. And second, we lift to the Markov automata setting the technique developed in [BS11] that monitors the gap between the upper- and the lower-bounds.

On a high level our solution works as follows. The computations proceed backwards in time, i. e. we first approximate the optimal reachability value given that t_i time units are left until the time bound (starting with $t_0 = 0$). When this value is computed we use it to approximate the optimal reachability value for $t_{i+1} > t_i$, and so on until the time bound b is met. Our goal is to have the intervals $[0, t_1], (t_1, t_2], \cdots, (t_{n-1}, b]$, computed by the algorithm, to resemble as closely as possible the partition $\mathcal{I}(\pi_{\mathtt{opt}})$ of an ϵ-optimal strategy $\pi_{\mathtt{opt}}$. The following are the high level steps of the algorithm:

0. a. $t = 0$

   b. Find a stationary strategy π that is optimal for $\mathrm{val}_{\mathrm{opt}}(0, g)$ (Section 4.3.2)

c. Set $\pi_{\texttt{opt}} = \pi|_{[0,0]}$

1. Find a stationary strategy π that is optimal on interval $(t, t+\delta]$, for some unknown δ (Section 4.3.2)
2. Approximate this δ (Sections 4.3.3 and 4.3.4) and set $\delta = \min\{\delta, b-t\}$
3. Set $\pi_{\texttt{opt}} = \pi_{\texttt{opt}}|_{[0,t]} \cdot \pi|_{(t,t+\delta]}$, $t = t + \delta$
4. Approximate $\mathrm{val}_{\pi_{\texttt{opt}}}(t+\delta, g)$ given an approximation of $\mathrm{val}_{\pi_{\texttt{opt}}}(t, g)$ (Section 4.3.1)
5. If $t = b$ return $\mathrm{val}_{\pi_{\texttt{opt}}}(t, g)$ and $\pi_{\texttt{opt}}$, else go to step 1.

In the following we will go into the details of each step starting with step 4.

4.3.1 Time-Bounded Reachability for a Scheduler

In this section we will show that approximating the time-bounded reachability probability for a given piecewise-constant scheduler can be done efficiently via a combination of two known techniques: Uniformisation and hop-unbounded reachability analysis. The following lemma shows that the reachability value can be characterised by a system of differential and linear equations:

Lemma 4.3.1. *Let g be a goal function, π is a piecewise-constant scheduler and $b \in \mathbb{R}_{\geqslant 0}$. Then $\forall s \in S : \mathrm{val}_\pi(s, b, g) = f_s(b)$, where $f_s(t)$ is the solution to the following system of differential and linear equations:*

$$\begin{aligned} \frac{\mathrm{d}(f_s(t))}{\mathrm{d}t} &= \sum_{s' \in S} \mathbf{R}[s,s'] \cdot f_{s'}(t) - E(s) \cdot f_s(t) \quad \textit{if } s \in MS \\ f_s(t) &= \sum_{s' \in S} \mathbb{P}[s, \pi(s,t), s'] \cdot f_{s'}(t) \qquad \textit{if } s \in PS \end{aligned} \tag{4.21}$$

$$f_s(0) = \begin{cases} g(s) & \textit{if } s \in MS, s \in dom(g) \\ 0 & \textit{if } s \in MS, s \notin dom(g) \end{cases} \tag{4.22}$$

Proof. By definition:

$$\mathrm{val}_\pi(s,t,g) = \sum_{s' \in dom(g)} \mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[\texttt{tt}\,\mathrm{U}^{=t} ap_{s'}\right] \cdot g(s')$$

Let $E(b,t,s) = \{\rho \in Paths^\omega \mid \rho \models \texttt{tt}\,\mathrm{U}^{=b-t} ap_s\}$. We define

$$h_{s,s'}(t) = \mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[\texttt{tt}\,\mathrm{U}^{=b} ap_{s'} \mid E(b,t,s)\right]$$

For a probabilistic state ps:

$$h_{ps,s'}(t) = \mathrm{Pr}^{\mathcal{M}}_{\pi,ps}\left[\texttt{tt}\,\mathrm{U}^{=t} ap_{s'} \mid E(b,t,ps)\right]$$

$$= \sum_{s'' \in S} \mathbb{P}[ps, \pi(ps,t), s''] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi,s''}\left[\mathtt{tt}\,\mathrm{U}^{=t} ap_{s'} \mid E(b,t,ps)\right]$$
$$= \sum_{s'' \in S} \mathbb{P}[ps, \pi(ps,t), s''] \cdot h_{s'',s'}(t)$$

On the other hand:

$$h_{ps,s'}(t) = \sum_{ms \in MS} \mathrm{rea}^{\pi|[t,t]}(ps, ms) \cdot h_{ms,s'}(t)$$

Consider a Markovian state ms and $t \geqslant 0$. Given a timed-memoryless scheduler the Markov automaton becomes deterministic, i. e. it has no non-determinism any more and its behaviour can be described by a stochastic process which is a non-homogeneous Markov chain. Or formally, for $ms, ms' \in MS$ the transient probability $h_{ms,ms'}(t) = \mathrm{Pr}^{\mathcal{M}}_{\pi,ms}\left[\mathtt{tt}\,\mathrm{U}^{=t} ap_{ms'} \mid E(b,t,ms)\right]$ of being in state ms' at time point b starting from ms at time $b-t$ is the same as $\mathrm{Pr}^{\mathcal{C}}_{\pi,ms}\left[\mathtt{tt}\,\mathrm{U}^{=b} ap_{ms'} \mid E(b,t,ms)\right]$[1], where $\mathcal{C} = (S', \mathbf{R}'(\tau))$ is a non-homogeneous CTMC, such that $S' = MS, \mathbf{R}'(\tau)$ is a time dependent rate matrix and $\forall ms, ms' \in S', \tau \in [0,b]$:

$$\begin{aligned}
\mathbf{R}'(\tau)[ms, ms'] &= E(ms) \cdot \sum_{s'' \in S} \frac{\mathbf{R}[ms, s'']}{E(ms)} \cdot \mathrm{rea}^{\pi|[b-\tau,b-\tau]}(s'', ms') \\
&= \sum_{s'' \in S} \mathbf{R}[ms, s''] \cdot \mathrm{rea}^{\pi|[b-\tau,b-\tau]}(s'', ms')
\end{aligned}$$

The transient probability in $\mathcal{C}$ is characterised by Kolmogorov's Backward Equations:

$$\begin{aligned}
&\forall ms, ms' \in S' : \\
&\frac{\mathrm{d}(h_{ms,ms'}(t))}{\mathrm{d}t} = \sum_{ms'' \in S'} \mathbf{R}'(b-t)[ms, ms''] \cdot h_{ms'',ms'}(t) - E(ms) \cdot h_{ms,ms'}(t) \\
&= \sum_{ms'' \in S'} \sum_{s'' \in S} \mathbf{R}[ms, s''] \cdot \mathrm{rea}^{\pi|[t,t]}(s'', ms'') \cdot h_{ms'',ms'}(t) \\
&= \sum_{s'' \in S} \mathbf{R}[ms, s''] \sum_{ms'' \in S'} \mathrm{rea}^{\pi|[t,t]}(s'', ms'') \cdot h_{ms'',ms'}(t) \\
&\quad - E(ms) \cdot h_{ms,ms'}(t) \\
&= \sum_{s'' \in PS} \mathbf{R}[ms, s''] \sum_{ms'' \in S'} \mathrm{rea}^{\pi|[t,t]}(s'', ms'') \cdot h_{ms'',ms'}(t) \\
&\quad + \sum_{s'' \in MS} \mathbf{R}[ms, s''] \cdot h_{s'',ms'}(t) - E(ms) \cdot h_{ms,ms'}(t) \\
&= \sum_{s'' \in S} \mathbf{R}[ms, s''] \cdot h_{s'',ms'}(t) - E(ms) \cdot h_{ms,ms'}(t)
\end{aligned}$$

[1]Notice the scheduler π as a subscript of $\mathrm{Pr}^{\mathcal{C}}_{\pi,ms}\left[\mathtt{tt}\,\mathrm{U}^{=t} ap_{ms'} \mid E(b,t,ms)\right]$. Here it does not play any role for the probability measure, because CTMCs contain no probabilistic states. Any scheduler indicated here induces the same probability measure. However the definition of the probability measure given in Lemma 2.2.1 requires a scheduler to be indicated, which is the reason why we cannot leave this parameter empty.

Consider a variable

$$\forall s \in S : q_s(t) = \sum_{s' \in dom(g)} h_{s,s'}(t) \cdot g(s') \tag{4.23}$$

If $s \in MS, t = 0$, then $\forall s' \neq s : h_{s,s'}(0) = 0$. Thus if $s \in dom(g)$ then $q_s(0) = g(s)$, and otherwise $q_s(0) = 0$. Therefore for $t = 0 : q_s(0)$ satisfies (4.22).

Consider $ps \in PS, t \geqslant 0$:

$$\begin{aligned}
q_s(t) &= \sum_{s' \in dom(g)} h_{s,s'}(t) \cdot g(s') \\
&= \sum_{s' \in dom(g)} \sum_{s'' \in S} \mathbb{P}[ps, \pi(ps,t), s''] \cdot h_{s'',s'}(t) \cdot g(s') \\
&= \sum_{s'' \in S} \mathbb{P}[ps, \pi(ps,t), s''] \sum_{s' \in dom(g)} h_{s'',s'}(t) \cdot g(s') \\
&= \sum_{s'' \in S} \mathbb{P}[ps, \pi(ps,t), s''] \cdot q_{s''}(t)
\end{aligned}$$

Consider $t > 0$ and $ms \in MS$:

$$\begin{aligned}
\frac{\mathrm{d}(q_{ms}(t))}{\mathrm{d}t} &= \frac{\mathrm{d}(\sum\limits_{s' \in dom(g)} h_{ms,s'}(t) \cdot g(s'))}{\mathrm{d}t} = \sum_{s' \in dom(g)} \frac{\mathrm{d}(h_{ms,s'}(t))}{\mathrm{d}t} \cdot g(s') \\
&= \sum_{s' \in dom(g)} \left(\sum_{s'' \in S} \mathbf{R}[ms, s''] \cdot h_{s'',s'}(t) - E(ms) \cdot h_{ms,s'}(t) \right) \cdot g(s') \\
&= \sum_{s'' \in S} \mathbf{R}[ms, s''] \sum_{s' \in dom(g)} h_{s'',s'}(t) \cdot g(s') - E(ms) \sum_{s' \in dom(g)} h_{ms,s'}(t) \cdot g(s') \\
&= \sum_{s'' \in S} \mathbf{R}[ms, s''] \cdot q_{s''}(t) - E(ms) \cdot q_{ms}(t)
\end{aligned}$$

Thus $q_s(t)$ satisfies system (4.21-4.22) and for $t = b$ it satisfies the definition of $\mathrm{val}_\pi(s, b, g)$.

Next we will show that the system (4.21-4.22) has a unique solution. Since we only consider non-Zeno MA, then the following condition is satisfied: $\exists m \in \mathbb{Z}_{\geqslant 0}$, such that $\forall \pi \in \Pi_{\mathtt{stat}}, ps \in PS : \Pr_{\pi,ps}[\{\rho \in Paths^\omega \mid \forall i \in 0..m : \rho[i] \notin MS\}] < 1$. To see this, choose m to be the length of the longest loop-free path from a probabilistic state to any Markovian state via only probabilistic states. Since the MA is non-Zeno, such paths exist for each probabilistic state. Under this condition the system of linear equations from (4.21-4.22) has a unique solution for a goal function v, such that $\forall ms \in MS : v(ms) = f_{ms}(t)$ and $\forall ps \in PS : v(ps)$ is undefined ([Ber05], Proposition 7.2.1).

On each interval $I \in \mathcal{I}(\pi)$ the scheduler is fixed and therefore the function $\mathbf{R}'(b-t)$ is constant in t for $t \in I$. Let $\mathcal{I}(\pi) = \{I_0 = [0], I_1 = (0, \tau_1], \cdots, I_n = (\tau_{n-1}, \tau_n], (\tau_n, \infty)\}$. For each interval I, starting from I_1, the system of differential equations from (4.21-4.22) has constant coefficients and initial conditions $\forall s \in S$:

$f_s(\inf I)$ formed by the solution from the previous interval, or values $f_s(0)$ when $I = I_1$. Under this conditions the system has a unique solution. This concludes the proof.

□

Remark 4.3.1. *Notice that due to (4.18) and (4.19) we can equivalently rewrite the equation for a probabilistic state ps as follows:*

$$f_{ps}(t) = \sum_{ms \in MS} \mathrm{rea}^{\pi|_{[t,t]}}(ps, ms) \cdot f_{ms}(t) \tag{4.24}$$

The system (4.21-4.22) can be solved iteratively over the intervals in $\mathcal{I}(\pi)$. Within each such interval $I \in \mathcal{I}(\pi)$ the scheduler is stationary, i. e. $\forall s \in S, t_1, t_2 \in I : \pi(s, t_1) = \pi(s, t_2)$. Given the solution $f_s(t)$ on an interval I, i. e. for all states $s \in S$ and $t \in I$, the solution on the neighbouring interval can be found by using $f_s(\sup I)$ as boundary conditions for the system (4.21). In the following we will show a way to approximate $f_s(t)$.

Numerical Solution. Our approximation scheme is essentially an extension of the uniformisation technique [Jen53] originally designed for CTMCs to Markov automata. It is carried over to probabilistic states via unbounded reachability analysis for MDPs. In the following, we will compute an approximation of the solution on an interval from $\mathcal{I}(\pi)$ given an approximation of the solution on the previous interval.

Just like uniformisation for CTMCs, uniformisation for Markov automata requires discrete transition probabilities within k Markovian transitions. Let h be a total goal function. For $k \in \mathbb{Z}_{\geqslant 0}, \pi' \in \Pi^{\mathcal{M}}_{\mathtt{stat}}$ we define $\mathbf{D}^k(s, \pi', h)$ as follows:

$$\mathbf{D}^k(s, \pi', h) = \begin{cases} h(s) & \text{if } s \in MS, k = 0 \\ \sum\limits_{s' \in S} \frac{\mathbf{R}[s,s']}{\mathbf{E}_{\max}} \cdot \mathbf{D}^{k-1}(s', \pi', h) + (1 - \frac{E(s)}{\mathbf{E}_{\max}}) \cdot \mathbf{D}^{k-1}(s, \pi', h) & \text{if } s \in MS, k > 0 \\ \sum\limits_{s' \in MS} \mathrm{rea}^{\pi'}(s, s') \cdot \mathbf{D}^k(s', \pi', h) & \text{if } s \in PS \end{cases} \tag{4.25}$$

The value $\mathbf{D}^k(s, \pi', h)$ is essentially the value $\sum_{s' \in S} p^k(s') \cdot h(s')$, where $p^k(s')$ is the discrete probability to reach a state s' starting from s by following a stationary strategy π' and performing exactly k Markovian transitions. This is the reason for the counter k to decrease only when a Markovian state performs a transition and not to be affected by probabilistic transitions. If for a probabilistic state ps the value $\mathbf{D}^k(ps, \pi', h)$ is under-approximated up to ε, then we will denote this with $\mathbf{D}^k_\varepsilon(s, \pi', h)$. We denote with $\mathbf{D}^k(\pi', h)$ the function over states that takes value $\mathbf{D}^k(s, \pi', h)$ for state s.

We denote with Ψ_λ the probability mass function of the Poisson distribution with parameter λ. For a $\tau \in \mathbb{R}_{\geqslant 0}$ and $\varepsilon \in (0, 1)$, we define $N(\tau, \varepsilon)$ to be some natural number satisfying $\sum_{i=0}^{N(\tau,\varepsilon)} \Psi_{\mathbf{E}_{\max} \cdot \tau}(i) \geqslant 1 - \varepsilon$, e. g. $N(\tau, \varepsilon) = \lceil \mathbf{E}_{\max} \cdot \tau \cdot e^2 - \ln(\varepsilon) \rceil$ [BHHK15].

The approximation error comes from various sources, that we will combine in a tuple:

$$\xi = (\varepsilon_\Psi, \varepsilon_{\mathrm{rea}}, \varepsilon_{\mathrm{n}}), \text{where } \varepsilon_\Psi \in (0, 1), \varepsilon_{\mathrm{rea}} \in [0, 1), \varepsilon_{\mathrm{n}} \in (0, 1) \tag{4.26}$$

Here values ε_Ψ and ε_{rea} refer to the total error accumulated over an interval I in which a piecewise constant scheduler remains constant. Component $\varepsilon_\Psi \in (0,1)$ denotes the total error allowed for the approximation over Markovian states, i. e. for uniformisation. Value $\varepsilon_{\text{rea}} \in [0,1)$ will denote the total error accumulated from the approximation of values of probabilistic states (hop-unbounded reachability probability). Value $\varepsilon_{\text{n}} \in (0,1)$ is the bound on the error that can be accumulated over probabilistic transitions between two Markovian transitions performed within interval I.

Let $I \in \mathcal{I}(\pi), \varepsilon \in (0,1)$ and $v : S \to [0,1]$ is such that $\forall s \in S : v(s) \leqslant \text{val}_\pi(\inf I, g) \leqslant v(s) + \varepsilon$. The approximation of functions $f_s(t)$ for a piecewise-constant scheduler π at time $t \in I$ will be denoted with $u^{t,v}_{\pi,\xi}(s)$, where $\xi = (\varepsilon_\Psi, \varepsilon_{\text{rea}}, \varepsilon_{\text{n}})$ satisfies (4.26), and can be obtained as follows:

$$\forall t \in I :$$
$$u^{t,v}_{\pi,\xi}(s) = \begin{cases} v(s) & \text{if } t = \inf I \\ \sum\limits_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_{\text{n}}}(s, \pi|_I, v) & \text{else if } s \in MS, \delta = t - \inf I \\ \text{rea}^{\pi|_I}_{\varepsilon_{\text{n}}}(s, u^{t,v}_{\pi,\xi}|_{MS}) & \text{else if } s \in PS \end{cases} \tag{4.27}$$

Thus one can approximate the values $f_s(t)$ by iterating over intervals in $\mathcal{I}(\pi)$. If I and I' are two neighbouring intervals, then given $u(s) = u^{t,v}_{\pi,\xi}(s)$ for $t = \sup I$, one can approximate values $f_s(t')$, for $t' \in I'$ by computing $u^{t',u}_{\pi,\xi}(s)$. The following lemma proves that the values obtained in this way indeed form an approximation to the solution of the system (4.21-4.22):

Lemma 4.3.2. *Let π be a piecewise-constant scheduler, $I \in \mathcal{I}(\pi), \varepsilon \in (0,1), \text{opt} \in \{\sup, \inf\}$. Let $v : S \to [0,1]$ be such that $\forall s \in S$:*

$$v(s) \preccurlyeq_{\text{opt}} \text{val}_\pi(s, \inf I, g) \preccurlyeq_{\text{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon$$

Then $\forall x \in I, s \in S, \xi$ that satisfies (4.26), such that $\varepsilon_n \leqslant \varepsilon_{rea} / (N(x - \inf I, \varepsilon_\Psi) + 1)$, the following holds:

$$\begin{aligned} \text{opt} = \sup &: u^{x,v}_{\pi,\xi}(s) \leqslant \text{val}_\pi(s,x,g) \leqslant u^{x,v}_{\pi,\xi}(s) + \varepsilon + \varepsilon_\Psi + \varepsilon_{rea} \\ \text{opt} = \inf &: u^{x,v}_{\pi,\xi}(s) + \varepsilon_\Psi + \varepsilon_{rea} \geqslant \text{val}_\pi(s,x,g) \geqslant u^{x,v}_{\pi,\xi}(s) - \varepsilon \end{aligned} \tag{4.28}$$

Proof. Consider $x = \inf I$. Then by definition $\forall s \in S : u^{x,v}_{\pi,\xi}(s) = v(s)$ and therefore (4.28) holds.

Consider $x > \inf I$ (this implies that $x > 0$) and $ms \in MS$. Since within interval I scheduler π is stationary, the stochastic process induced by π at time points within I is a CTMC. Or formally, for $ms, ms' \in MS$ the transient distribution $\text{Pr}^{\mathcal{M}}_{\pi,ms}[\texttt{tt}\, \text{U}^{=x} ap_{ms'}]$ of being in state ms' at time point x starting from ms in $\mathcal{M}$

is the same as $\mathrm{Pr}^{\mathcal{C}}_{\pi,ms}[\mathtt{tt}\,\mathrm{U}^{=x}ap_{ms'}]$[2], where $\mathcal{C} = (S', \mathbf{R}')$ is a CTMC, such that $S' = MS$ and $\forall ms, ms' \in S'$:

$$\mathbf{R}'[ms, ms'] = E(ms) \cdot \sum_{s'' \in S} \frac{\mathbf{R}[ms, s'']}{E(ms)} \cdot \mathrm{rea}^{\pi|_I}(s'', ms') = \sum_{s'' \in S} \mathbf{R}[ms, s''] \cdot \mathrm{rea}^{\pi|_I}(s'', ms')$$

This implies that the transient distribution satisfies the Chapman-Kolmogorov equations $\forall s, s' \in S', x \in I$:

$$\mathrm{Pr}^{\mathcal{C}}_{\pi,s}[\mathtt{tt}\,\mathrm{U}^{=x}ap_{s'}] = \sum_{s'' \in S'} \mathrm{Pr}^{\mathcal{C}}_{\pi,s}\left[\mathtt{tt}\,\mathrm{U}^{=x-\inf I}ap_{s''}\right] \cdot \mathrm{Pr}^{\mathcal{C}}_{\pi,s''}\left[\mathtt{tt}\,\mathrm{U}^{=\inf I}ap_{s'}\right] \tag{4.29}$$

And that the transient distribution between two states $s, s' \in S'$ can be computed via uniformisation [Jen53]:

$$\mathrm{Pr}^{\mathcal{C}}_{\pi,s}[\mathtt{tt}\,\mathrm{U}^{=x}ap_{s'}] = \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot x}(i) \cdot \mathbb{P}'_i[s, s'] \tag{4.30}$$

where for $i \in \mathbb{Z}_{\geqslant 0}, s, s'' \in S'$:

$$\mathbb{P}'_i[s, s''] = \begin{cases} 1 & \text{if } i = 0, s = s'' \\ \sum\limits_{s' \in S'} \frac{\mathbf{R}'[s,s']}{\mathbf{E}_{\max}} \cdot \mathbb{P}'_{i-1}[s', s''] + (1 - \frac{E(s)}{\mathbf{E}_{\max}}) \cdot \mathbb{P}'_{i-1}[s, s''] & \text{if } i > 0 \\ 0 & \text{otherwise} \end{cases}$$

By definition of $\mathrm{val}_\pi(ms, x, g)$ we can obtain the following:

$$\begin{aligned}
&\mathrm{val}_\pi(ms, x, g) \\
&= \sum_{s' \in dom(g)} \mathrm{Pr}^{\mathcal{M}}_{\pi,ms}[\mathtt{tt}\,\mathrm{U}^{=x}ap_{s'}] \cdot g(s') \\
&\overset{(4.29)}{=} \sum_{s' \in g} \sum_{s'' \in MS} \mathrm{Pr}^{\mathcal{M}}_{\pi,ms}\left[\mathtt{tt}\,\mathrm{U}^{=x-\inf I}ap_{s''}\right] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi,s''}\left[\mathtt{tt}\,\mathrm{U}^{=\inf I}ap_{s'}\right] \cdot g(s') \\
&= \sum_{s'' \in MS} \mathrm{Pr}^{\mathcal{M}}_{\pi,ms}\left[\mathtt{tt}\,\mathrm{U}^{=x-\inf I}ap_{s''}\right] \cdot \mathrm{val}_\pi(s'', \inf I, g) \\
&\overset{(4.30)}{=} \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot(x-\inf I)}(i) \underbrace{\sum_{s'' \in MS} \mathbb{P}'_i[ms, s''] \cdot \mathrm{val}_\pi(s'', \inf I, g)}_{\widehat{\mathbf{D}}^i(ms, \pi|_I, \mathrm{val}_\pi(\inf I, g))}
\end{aligned} \tag{4.31}$$

[2]Notice the scheduler π as a subscript of $\mathrm{Pr}^{\mathcal{C}}_{\pi,ms}[\mathtt{tt}\,\mathrm{U}^{=x}ap_{ms'}]$. Here it does not play any role for the probability measure, because CTMCs contain no probabilistic states. Any scheduler indicated here induces the same probability measure. However the definition of the probability measure given in Lemma 2.2.1 requires a scheduler to be indicated, which is the reason why we cannot leave out this parameter.

Consider $ps \in PS$. Since $x > \inf I$, then $x > 0$ and therefore:

$$\begin{aligned}
&\mathrm{val}_\pi(ps, x, g) \\
&\overset{\textit{Rem. } 4.3.1}{=} \sum_{ms \in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot \mathrm{val}_\pi(ms, x, g) \\
&= \sum_{ms \in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max} \cdot (x - \inf I)}(i) \cdot \widehat{\mathbf{D}}^i(ms, \pi|_I, \mathrm{val}_\pi(\inf I, g)) \\
&= \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max} \cdot (x - \inf I)}(i) \underbrace{\sum_{ms \in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot \widehat{\mathbf{D}}^i(ms, \pi|_I, \mathrm{val}_\pi(\inf I, g))}_{\widehat{\mathbf{D}}^i(ps, \pi|_I, \mathrm{val}_\pi(\inf I, g))}
\end{aligned} \tag{4.32}$$

Lemma A.3 proves that for $\pi' \in \Pi_{\mathtt{stat}}$, $h : S \to [0, 1]$ the following holds:

$$\forall s \in S, i \in \mathbb{Z}_{\geqslant 0} : \widehat{\mathbf{D}}^i(s, \pi', h) = \mathbf{D}^i(s, \pi', h) \tag{4.33}$$

Applying these results to a function h, such that $\forall s \in S : h(s) = \mathrm{val}_\pi(s, \inf I, g)$ and to scheduler $\pi|_I$, by using (4.31) we can rewrite the reachability value function for a Markovian state $ms \in MS$ as follows:

$$\mathrm{val}_\pi(ms, x, g) = \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max} \cdot (x - \inf I)}(i) \cdot \mathbf{D}^i(ms, \pi|_I, \mathrm{val}_\pi(\inf I, g)) \tag{4.34}$$

Next we show that if $w_j : S \to [0, 1], j \in \{1, 2\}$, such that $\forall s \in S : w_1(s) \preccurlyeq_{\mathrm{opt}} w_2(s) \preccurlyeq_{\mathrm{opt}} w_1(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon'$, for some $\varepsilon' \in [0, 1)$, then $\forall s \in S, i \in \mathbb{Z}_{\geqslant 0}, \pi' \in \Pi_{\mathtt{stat}}$:

$$\mathbf{D}^i(s, \pi', w_1) \preccurlyeq_{\mathrm{opt}} \mathbf{D}^i(s, \pi', w_2) \preccurlyeq_{\mathrm{opt}} \mathbf{D}^i(s, \pi', w_1) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon' \tag{4.35}$$

The statement holds for $i = 0, s \in MS$ by definition of w_j. For $i \in \mathbb{Z}_{\geqslant 0}, s \in S$ the value $\mathbf{D}^i(s, \pi', w_j)$ is a convex combination of either values $\mathbf{D}^i(\cdot, \pi', w_j)$ over Markovian states (if $s \in PS$), or values $\mathbf{D}^{i-1}(\cdot, \pi', w_j)$ (if $s \in MS$). Thus the statement holds for all states s and indices i by induction.

Let $w(s) = \mathrm{val}_\pi(s, \inf I, g)$ and $x - \inf I = \delta$. The results obtained above lead to the following. For a Markovian state ms and opt = sup:

$$\begin{aligned}
u_{\pi,\xi}^{x,v}(ms) &= \sum_{i=0}^{N(\delta, \varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max} \cdot \delta}(i) \cdot \mathbf{D}_{\varepsilon_{\mathrm{n}}}^i(ms, \pi|_I, v) \\
&\overset{\text{Lemma A.4}}{\leqslant} \sum_{i=0}^{N(\delta, \varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max} \cdot \delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, v) \\
&\overset{(4.35)}{\leqslant} \sum_{i=0}^{N(\delta, \varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max} \cdot \delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, w) \\
&\leqslant \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max} \cdot \delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, w)
\end{aligned}$$

$$
\begin{aligned}
&\overset{(4.34)}{=} \operatorname{val}_\pi(ms, x, g) \\
&\leqslant \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, v) + \varepsilon \\
&\leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, v) + \varepsilon_\Psi + \varepsilon \\
&\overset{\text{Lemma A.4}}{\leqslant} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ms, \pi|_I, v) + \varepsilon_\Psi + \varepsilon_\mathrm{n} \cdot N(\delta, \varepsilon_\Psi) + \varepsilon \\
&= u^{x,v}_{\pi,\xi}(ms) + \varepsilon_\Psi + \varepsilon_\mathrm{n} \cdot N(\delta, \varepsilon_\Psi) + \varepsilon
\end{aligned}
$$

Analogously for opt = inf:

$$
\begin{aligned}
&u^{x,v}_{\pi,\xi}(ms) + \varepsilon_\Psi + \varepsilon_\mathrm{rea} \\
&\quad\geqslant u^{x,v}_{\pi,\xi}(ms) + \varepsilon_\Psi + \varepsilon_\mathrm{n} \cdot (N(\delta, \varepsilon_\Psi) + 1) \\
&\quad\geqslant u^{x,v}_{\pi,\xi}(ms) + \varepsilon_\Psi + \varepsilon_\mathrm{n} \cdot N(\delta, \varepsilon_\Psi) \\
&\quad= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ms, \pi|_I, v) + \varepsilon_\Psi + \varepsilon_\mathrm{n} \cdot N(\delta, \varepsilon_\Psi) \\
&\quad\overset{\text{Lemma A.4}}{\geqslant} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, v) + \varepsilon_\Psi \\
&\quad\overset{(4.35)}{\geqslant} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, w) + \varepsilon_\Psi \\
&\quad\geqslant \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, w) \\
&\quad\overset{(4.34)}{=} \operatorname{val}_\pi(ms, x, g) \\
&\quad\overset{(4.35)}{\geqslant} \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, v) - \varepsilon \\
&\quad\geqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i(ms, \pi|_I, v) - \varepsilon \\
&\quad\overset{\text{Lemma A.4}}{\geqslant} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ms, \pi|_I, v) - \varepsilon \\
&\quad= u^{x,v}_{\pi,\xi}(ms) - \varepsilon
\end{aligned}
$$

For a probabilistic state $ps \in PS$, opt = sup:

$$
\begin{aligned}
u^{x,v}_{\pi,\xi}(ps) &= \operatorname{rea}^{\pi|_I}_{\varepsilon_\mathrm{n}}(ps, u^{x,v}_{\pi,\xi}|_{MS}) \\
&\leqslant \operatorname{rea}^{\pi|_I}(ps, u^{x,v}_{\pi,\xi}|_{MS}) \\
&\leqslant \operatorname{rea}^{\pi|_I}(ps, \operatorname{val}_\pi(x, g)|_{MS})
\end{aligned}
$$

$$
\begin{aligned}
&\leqslant \mathrm{rea}^{\pi|_I}(ps, u^{x,v}_{\pi,\xi}|_{MS}) + \varepsilon_\Psi + \varepsilon_{\mathrm{n}} \cdot N(\delta, \varepsilon_\Psi) + \varepsilon \\
&\leqslant \mathrm{rea}^{\pi|_I}_{\varepsilon_{\mathrm{n}}}(ps, u^{x,v}_{\pi,\xi}|_{MS}) + \varepsilon_\Psi + \varepsilon_{\mathrm{n}} \cdot (N(\delta, \varepsilon_\Psi) + 1) + \varepsilon \\
&\leqslant u^{x,v}_{\pi,\xi}(ps) + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}} + \varepsilon
\end{aligned}
$$

And for opt = inf:

$$
\begin{aligned}
u^{x,v}_{\pi,\xi}(ps) + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}} &\geqslant u^{x,v}_{\pi,\xi}(ps) + \varepsilon_\Psi + \varepsilon_{\mathrm{n}} \cdot (N(\delta, \varepsilon_\Psi) + 1) \\
&= \mathrm{rea}^{\pi|_I}_{\varepsilon_{\mathrm{n}}}(ps, u^{x,v}_{\pi,\xi}|_{MS}) + \varepsilon_\Psi + \varepsilon_{\mathrm{n}} \cdot (N(\delta, \varepsilon_\Psi) + 1) \\
&\geqslant \mathrm{rea}^{\pi|_I}(ps, u^{x,v}_{\pi,\xi}|_{MS}) + \varepsilon_\Psi + \varepsilon_{\mathrm{n}} \cdot N(\delta, \varepsilon_\Psi) \\
&\geqslant \mathrm{rea}^{\pi|_I}(ps, \mathrm{val}_\pi(x, g)|_{MS}) \\
&\geqslant \mathrm{rea}^{\pi|_I}(ps, u^{x,v}_{\pi,\xi}|_{MS}) - \varepsilon \\
&\geqslant \mathrm{rea}^{\pi|_I}_{\varepsilon_{\mathrm{n}}}(ps, u^{x,v}_{\pi,\xi}|_{MS}) - \varepsilon \\
&\geqslant u^{x,v}_{\pi,\xi}(ps) - \varepsilon
\end{aligned}
$$

□

4.3.2 Optimal Strategy

In this section, we will show how to compute an optimal piecewise-constant strategy, if it exists. Assume that for time point $t < b$ an optimal piecewise-constant scheduler $\pi_{\mathtt{opt}}$ is computed. We will show that there exists a stationary strategy π and $\delta > 0$, such that following π on time points $(t, t+\delta]$ is the optimal behaviour. Or formally, strategy $\pi'_{\mathtt{opt}} = \pi_{\mathtt{opt}}|_{[0,t]} \cdot \pi|_{(t,t+\delta]}$ is optimal for $\mathrm{val}_{\mathrm{opt}}(x, g)$, for all $x \in [0, t+\delta]$.

First of all, due to Lemma 4.3.1 and Remark 4.3.1 for $t = 0$ the problem is reduced to the computation of $\mathrm{rea}^{\mathrm{opt}}(s, g)$.

Consider $t \geqslant 0$ and let $\pi_{\mathtt{opt}}$ be an optimal piecewise-constant strategy on interval $[0, t]$. Analogously to results for CTMDPs [Mil68], we will prove that derivatives of function $\mathrm{val}_{\pi_{\mathtt{opt}}}(\tau, g)$ at time $\tau = t$ help finding the stationary strategy π that remains optimal for interval $(t, t+\delta]$, for some $\delta > 0$. This is rooted in the Taylor expansion of function $\mathrm{val}_{\pi'_{\mathtt{opt}}}(t+\delta, g)$ via the values of $\mathrm{val}_{\pi_{\mathtt{opt}}}(t, g)$, where $\pi'_{\mathtt{opt}} = \pi_{\mathtt{opt}}|_{[0,t]} \cdot \pi|_{(t,t+\delta]}$.

Let v be a goal function. We define sets

$$
\begin{aligned}
\mathcal{F}^0(v) &= \{\pi \in \Pi_{\mathtt{stat}} \mid \forall s \in PS : \pi = \arg\,\mathrm{opt}_{\,\pi' \in \Pi_{\mathtt{stat}}} \vec{d}^{(0)}_{\pi'}(s, v)\} \\
\mathcal{F}^i(v) &= \{\pi \in \mathcal{F}^{i-1}(v) \mid \forall s \in PS : \pi = \arg\,\mathrm{opt}_{\,\pi' \in \mathcal{F}^{i-1}(v)} \vec{d}^{(i)}_{\pi'}(s, v)\}, i \in \mathbb{Z}_{>0},
\end{aligned}
$$

where for $\pi \in \Pi_{\mathtt{stat}} : \vec{d}^{(0)}_\pi(s, v) = \mathrm{val}_\pi(s, 0, v)$. For $i \geqslant 1$:

$$
\begin{aligned}
\vec{d}^{(i)}_\pi(s, v) &= \begin{cases} \sum\limits_{s' \in S} \mathbf{R}[s, s'] \cdot \vec{d}^{(i-1)}(s', v) - E(s) \cdot \vec{d}^{(i-1)}(s, v) & \text{if } s \in MS \\ \sum\limits_{s' \in MS} \mathrm{rea}^\pi(s, s') \cdot \vec{d}^{(i)}(s', v) & \text{if } s \in PS, t > 0 \end{cases} \quad (4.36) \\
\vec{d}^{(i)} &= \vec{d}^{(i)}_\pi \text{ for any } \pi \in \mathcal{F}^i(v),
\end{aligned}
$$

Informally, the value $\vec{d}^{(i)}_{\pi}(s,v)$ is the ith derivative of function $\mathrm{val}_{\pi}(s,\tau,v)$ at time $\tau = 0$.

Lemma 4.3.3. *Let π be an optimal piecewise-constant scheduler, $t \geqslant 0$ and $v(s)$ be a goal function, such that $v(s) = \mathrm{val}_{\pi}(s,t,g)$ if $s \in MS$ and is undefined otherwise. If $\pi' \in \mathcal{F}^{|S|}(v)$ then $\exists \delta > 0$ such that π' is optimal for $\mathrm{val}_{\mathrm{opt}}(t+x,v)$ on all $x \in (0,\delta]$.*

Proof. Let $\pi_{st} \in \Pi_{\mathtt{stat}}$. We will first define a strategy π_{ext} that behaves as π at time points within $[0,t]$ and as π_{st} afterwards. Formally $\pi_{ext} = \pi_{st}$ if $t = 0$. If $t > 0$ and $\rho \in Paths^*$, s. t. $\tau_{\mathrm{total}}(\rho) \leqslant t$ then $\pi_{ext}(\rho) = \pi(\rho)$ and if $\tau_{\mathrm{total}}(\rho) > t$ then $\pi_{ext}(\rho) = \pi_{st}(\rho\downarrow)$. We first show that $\vec{d}^{(i)}_{\pi_{st}}(s,v)$ is the ith derivative of function $\mathrm{val}_{\pi_{ext}}(s,\tau,g)$ at time $\tau = t$.

For $i = 0 : \vec{d}^{(0)}_{\pi_{st}}(s,v) = \mathrm{val}_{\pi_{st}}(s,0,v)$. If $t = 0$, then $\mathrm{val}_{\pi_{st}}(s,0,v) = \mathrm{val}_{\pi_{ext}}(s,0,g)$. If $t > 0, ms \in MS$, then $\mathrm{val}_{\pi_{st}}(ms,0,v) = \mathrm{val}_{\pi}(ms,t,g) = \mathrm{val}_{\pi_{ext}}(ms,t,g)$. For $ps \in PS : \mathrm{val}_{\pi_{st}}(ps,0,v) = \sum_{ms\in MS} \mathrm{rea}^{\pi_{st}}(ps,ms)\cdot v(ms) = \sum_{ms\in MS} \mathrm{rea}^{\pi_{st}}(ps,ms)\cdot \mathrm{val}_{\pi}(ms,t,g) = \mathrm{val}_{\pi_{ext}}(ps,t,g)$.

Consider $i = 1, ms \in MS, t \geqslant 0$. Due to (4.21):

$$\frac{\mathrm{d}^1(\mathrm{val}_{\pi_{ext}}(ms,t,g))}{\mathrm{d}t} = \sum_{s\in S} \mathbf{R}[ms,s]\cdot \mathrm{val}_{\pi_{ext}}(s,t,g) - E(s)\cdot \mathrm{val}_{\pi_{ext}}(ms,t,g) = \vec{d}^{(1)}_{\pi_{st}}(ms,v)$$

Taking derivative from both sides, we obtain the result for any $i > 0$:

$$\begin{aligned}\frac{\mathrm{d}^i(\mathrm{val}_{\pi_{ext}}(ms,t,g))}{\mathrm{d}t} &= \sum_{s\in S} \mathbf{R}[ms,s]\cdot \frac{\mathrm{d}^{i-1}(\mathrm{val}_{\pi_{ext}}(ms,t,g))}{\mathrm{d}t} - E(s)\cdot \frac{\mathrm{d}^{i-1}(\mathrm{val}_{\pi_{ext}}(ms,t,g))}{\mathrm{d}t}\\ &= \vec{d}^{(i)}_{\pi_{st}}(ms,v)\end{aligned}$$

Consider $i = 1$, $ps \in PS, t \geqslant 0$. Due to (4.18) and the fact that $s \in dom(g) \Rightarrow s \in MS$:

$$\begin{aligned}&\frac{\mathrm{d}^1(\mathrm{val}_{\pi_{ext}}(ps,t,g))}{\mathrm{d}t}\\ &= \lim_{\tau\to 0} \frac{\mathrm{val}_{\pi_{ext}}(ps,t+\tau,g) - \mathrm{val}_{\pi_{ext}}(ps,t,g)}{\tau}\\ &= \sum_{ms\in MS} \mathrm{rea}^{\pi_{st}}(ps,ms) \lim_{\tau\to 0}\left(\frac{\mathrm{val}_{\pi_{ext}}(ms,t+\tau,g) - \mathrm{val}_{\pi_{ext}}(ms,t,g)}{\tau}\right)\\ &= \sum_{ms\in MS} \mathrm{rea}^{\pi_{st}}(ps,ms)\cdot \frac{\mathrm{d}^1(\mathrm{val}_{\pi_{ext}}(ms,t,g))}{\mathrm{d}t} = \vec{d}^{(1)}_{\pi_{st}}(ps,v)\end{aligned}$$

Taking derivatives from both sides we obtain the required equation for $i > 1$, $ps \in PS, t \geqslant 0$.

Using Taylor's theorem we can represent the value of function $\mathrm{val}_{\pi_{ext}}(s,\tau,g)$ at time point $\tau = t + \delta > t$ for $k \in \mathbb{Z}_{>0}$ as follows:

$$\mathrm{val}_{\pi_{ext}}(s,t+\delta,g) = \sum_{i=0}^{k} \vec{d}^{(i)}_{\pi_{st}}(s,v)\cdot \delta^i + h_k(t+\delta)\cdot \delta^k,$$

where $\lim_{\delta\to 0} h_k(t+\delta) = 0$. Consider $i \in \mathbb{Z}_{>0}$, a strategy $\pi' \in \mathcal{F}^i(v)$ and a strategy $\pi'' \in \Pi_{\mathtt{stat}}$, such that $\pi'' \notin \mathcal{F}^i(v)$ and either $\pi'' \notin \mathcal{F}^0(v)$, or i_0 is the maximal index, such that $\pi'' \in \mathcal{F}^{i_0}(v)$. We will denote the former case with $i_0 = -1$. Then $\forall j = 0..i_0, s \in S : \vec{d}^{(j)}_{\pi'}(s,v) = \vec{d}^{(j)}_{\pi''}(s,v), \vec{d}^{(i_0+1)}_{\pi'}(s,v) \succcurlyeq_{\mathrm{opt}} \vec{d}^{(i_0+1)}_{\pi''}(s,v)$ and $\exists s_0 \in S : \vec{d}^{(i_0+1)}_{\pi'}(s_0,v) - \vec{d}^{(i_0+1)}_{\pi''}(s_0,v) \succ_{\mathrm{opt}} 0$. If $\pi'' \notin \mathcal{F}^0(v)$, then there exists a state s_0, such that $\mathrm{val}_{\pi''}(s_0,0,v) < \mathrm{val}_{\pi'}(s,0,v)$, which implies that strategy π'' is strictly suboptimal for time point t, and therefore there exists $\delta > 0$, such that π'' remains to be strictly suboptimal within $(t, t+\delta)$. Consider the case $i_0 \geqslant 0$. Applying the Taylor expansion for $k = i_0 + 1$ we obtain the following:

$$\mathrm{val}_{\pi'}(s,t{+}\delta,g){-}\mathrm{val}_{\pi''}(s,t{+}\delta,g) = \delta^k \cdot \left(\vec{d}^{(k)}_{\pi'}(s,v) - \vec{d}^{(k)}_{\pi''}(s,v) + h'_k(t+\delta) - h''_k(t+\delta)\right),$$

where $h'_k(x)$ and $h''_k(x)$ are respective remainders. Since $h'_k(x) \in o(1)$ and $h''_k(x) \in o(1)$, then there exists $\delta > t$, such that $\forall x \in (0,\delta] : h'_k(t+x) - h''_k(t+x) \leqslant \vec{d}^{(k)}_{\pi'}(s,v) - \vec{d}^{(k)}_{\pi''}(s,v)$. Since $\exists s_0 \in S$, such that $\vec{d}^{(k)}_{\pi'}(s_0,v) - \vec{d}^{(k)}_{\pi''}(s_0,v) \succ_{\mathrm{opt}} 0$, then π'' is not an optimal strategy on $(t, t+\delta)$.

Therefore one could compute sets $\mathcal{F}^i(v)$ as long they have more than one strategy and stop whenever a singleton set is found. However, it is possible that there exist multiple optimal strategies and a singleton set will never be reached. In the following we will show that computing $|S|$ many derivatives suffices.

Let $\pi', \pi'' \in \Pi_{\mathtt{PC}}$. Analogously to the original proof of Lemma 3 in [Mil68] it can be shown that if $\forall i = 0..|S|, s \in S : \frac{\mathrm{d}^i(\mathrm{val}_{\pi'}(s,t,g))}{\mathrm{d}t} = \frac{\mathrm{d}^i(\mathrm{val}_{\pi''}(s,t,g))}{\mathrm{d}t}$, then $\forall i \in \mathbb{Z}_{\geqslant 0}, s \in S : \frac{\mathrm{d}^i(\mathrm{val}_{\pi'}(s,t,g))}{\mathrm{d}t} = \frac{\mathrm{d}^i(\mathrm{val}_{\pi''}(s,t,g))}{\mathrm{d}t}$ and $\forall \tau \geqslant t, s \in S : \mathrm{val}_{\pi'}(s,\tau,g) = \mathrm{val}_{\pi''}(s,\tau,g)$.

Thus if for $i = 0..|S|$ values $\vec{d}^{(i)}_{\pi_{st}}(s,v)$ coincide for any two stationary strategies π^1_{st} and π^2_{st}, then $\mathrm{val}_{\pi^1_{ext}}(s,\tau,g) = \mathrm{val}_{\pi^2_{ext}}(s,\tau,g)$ for $\tau \geqslant t$, where for $j \in \{1,2\}$: $\pi^j_{ext} = \pi^j_{st}$ if $t = 0$, $\pi^j_{ext} = \pi|_{[0,t)} \cdot \pi^j_{st}|_{[t,b]}$ if $t > 0$. Therefore computing the first $|S|+1$ coefficients of the Taylor's expansion suffices and any strategy that optimises those values is optimal around point t. □

Thus in order to compute a stationary strategy that is optimal starting from time t, one needs to compute for each state s at most $|S|+1$ values $\vec{d}^{(i)}_{\pi}(s,v)$, where v is defined in the lemma above. In the following we will refer to a procedure that performs this computation for a goal function v with $\mathtt{FindStrat}^{\mathrm{opt}}(v)$. It computes sets $\mathcal{F}^i(v)$ until for some $j \in 0..|S|$ there is only 1 strategy left, i. e. $|\mathcal{F}^j(v)| = 1$. If this does not happen, it outputs any strategy in $\mathcal{F}^{|S|}(v)$.

In some cases computation of an optimal strategy with procedure $\mathtt{FindStrat}^{\mathrm{opt}}$ may become expensive, or not possible at all. In the following we will discuss possible reasons for this to happen and how to deal with it.

Optimisations. While from the theoretical perspective procedure $\mathtt{FindStrat}^{\mathrm{opt}}$ is reasonably efficient, in practice it has a few issues. Below we will list the challenges that we have encountered when implementing $\mathtt{FindStrat}^{\mathrm{opt}}$ and ways in which we have addressed them. We will refer to the modified procedure with $\mathtt{FindStrat}$.

– It may happen that values $|\vec{d}^{(i)}_{\pi}(s)|$ cannot be represented by floating point numbers of the underlying computer because they are too small or too large.

To counteract this `FindStrat` defines bounds $\overline{D}^{prm}$ and $\underline{D}^{prm}$ on the absolute value of the derivatives. Values outside of the interval $[\overline{D}^{prm}, \underline{D}^{prm}]$ are truncated.

- For large state-spaces computation of $|S|$ derivatives may be too prohibitive. Procedure `FindStrat` defines an upper bound $K^{prm} \leqslant |S|$ on the number of derivatives to compute and chooses any strategy from $\mathcal{F}_{K^{prm}}$.

- Computation of sets $\mathcal{F}^i(v)$ requires computing optimal expected cumulative unbounded reward. Exact computations may be too expensive and therefore approximations of the strategies may be preferable. Procedure `FindStrat` defines ε^{prm} to be the approximation error bound. We will refer to the sets $\mathcal{F}^i(v)$ obtained in this way with $\mathcal{F}^i_{\varepsilon^{prm}}(v)$.

- It may happen that all optimal strategies switch very often, in which case the algorithm has to make many calls to procedure `FindStrat`$^{\text{opt}}$. On the other hand, there may exist a piecewise-constant strategy that switches rarely, however, induces acceptable error. Computing the latter strategy instead of an optimal one will save the runtime otherwise spent on unnecessary computations. To counteract this issue procedure `FindStrat`(v) will compute a second strategy $\widetilde{\pi}(v)$, that is ε-optimal on a short time interval (under certain assumptions), where the value of ε is known. Thus the output of `FindStrat`(v) is a pair $(\pi, \widetilde{\pi}(v))$. Details of the computation of $\widetilde{\pi}(v)$ are described below.

Due to the changes discussed above the strategy computed by `FindStrat` via sets $\mathcal{F}^i(v)$ (i. e. the first element of the output pair) is not guaranteed to be an optimal strategy, which makes `FindStrat` more of a heuristic. In our experience, however, there is a definite benefit in terms of the runtime when using `FindStrat` instead of `FindStrat`$^{\text{opt}}$. Our solution is not bound to `FindStrat` and can, in fact, use any heuristic. It will produce under certain assumptions guaranteed a priori specified error bounds, and otherwise an a posteriori estimate of the error. Needless to say, the heuristic used to choose a strategy affects its quality. A poorly selected strategy will slow down the running time of the overall algorithm.

Sound One-Step Approximation. In order to preserve correctness in presence of a heuristic that is not necessarily good, and in order to combat the last issue in the list above, we will define a value v' and a scheduler π' that are ε-close under certain assumptions to the optimal values, with an explicit expression for ε. The techniques that we use to achieve this are a simple adaptation of the discretisation approach [GHH$^+$14] to our setting. We will define procedure `FindStrat`(v) to output a tuple (π, π'), where π is the strategy that the procedure selects as described above using sets $\mathcal{F}^i(v)$, or via any other heuristic. We start by introducing the required assumption and afterwards define formally π' and v'.

Assumption 4.3.1. *If $s \in dom(g)$, then s is absorbing.*

Under Assumption 4.3.1 one can derive convenient-to-use approximations of the probability values after one Markovian transition, as we will show below.

Let π be a piecewise-constant scheduler, $t, t' \in \mathbb{R}_{\geqslant 0}, t < t', ms \in MS$. We define

$$X(ms, \pi, t, t') := (1 - e^{-E(ms)\cdot(t'-t)}) \sum_{s' \in S} \frac{\mathbf{R}[ms, s']}{E(ms)} \cdot \text{val}_\pi(s', t, g)$$

$$A(ms, \pi, t, t') := \int_0^{t'-t} E(ms) \cdot e^{-E(ms)\cdot x} \sum_{s' \in S} \frac{\mathbf{R}[ms, s']}{E(ms)} \cdot \text{val}_\pi(s', t' - x, g)\, \mathrm{d}x$$

Lemma 4.3.4. *If Assumption 4.3.1 is satisfied, then for all* $ms \in MS, \pi \in \Pi_{\mathtt{PC}}, t, t' \in \mathbb{R}_{\geqslant 0}, t < t'$*:*

$$X(ms, \pi, t, t') \leqslant A(ms, \pi, t, t') \leqslant X(ms, \pi, t, t') + \frac{(\mathbf{E}_{\max} \cdot (t' - t))^2}{2} \tag{4.37}$$

Proof. The proof is a straightforward adaptation of Lemma 6.2 [Neu10]. We only need to show that function $\text{val}_\pi(t, g)$ is monotone increasing in t for any $\pi \in \Pi_{\mathtt{PC}}$.

Let $\pi \in \Pi_{\mathtt{PC}}, ms \in MS, t, t' \in \mathbb{R}_{\geqslant 0}, t < t'$. Then the stochastic process $\{Y_\tau\}$ induced by π is a non-homogeneous CTMC over Markovian states and Chapman-Kolmogorov equations can be applied to it:

$$f_{ms,ms'}(t, t') = \sum_{ms'' \in MS} f_{ms,ms''}(t, t'') \cdot f_{ms'',ms'}(t'', t'), \tag{4.38}$$

where $f_{ms,ms'}(t, t')$ is the transient probability in stochastic process $\{Y_\tau\}$ of being in state ms' at time t' when starting from state ms at time point t, and $t \leqslant t'' \leqslant t'$.

Since Assumption 4.3.1 is satisfied, then for any goal state ms_g, any Markovian state ms' and $x \leqslant x' \in \mathbb{R}_{\geqslant 0}$: $f_{ms_g,ms'}(x, x') = 1$ iff $ms' = ms_g$ and is 0 otherwise. Additionally, $\text{val}_\pi(ms, t, g) = \sum_{ms_g \in dom(g)} f_{ms,ms_g}(0, t)$. Taking this into account we can rewrite the value $\text{val}_\pi(ms, t', g)$ as follows:

$$\begin{aligned}
&\text{val}_\pi(ms, t', g) \\
&\quad = \sum_{ms_g \in dom(g)} f_{ms,ms_g}(0, t') \\
&\quad \overset{(4.38)}{=} \sum_{ms_g \in dom(g)} \sum_{ms'' \in MS} f_{ms,ms''}(0, t) \cdot f_{ms'',ms_g}(t, t') \\
&\quad = \sum_{ms_g \in dom(g)} \Big(\sum_{ms'_g \in dom(g)} f_{ms,ms'_g}(0, t) \cdot f_{ms'_g,ms_g}(0, t) \\
&\qquad\qquad + \sum_{ms'' \notin dom(g)} f_{ms,ms''}(0, t) \cdot f_{ms'',ms_g}(t, t') \Big) \\
&\quad = \sum_{ms'_g \in dom(g)} f_{ms,ms'_g}(0, t) \sum_{ms_g \in dom(g)} f_{ms'_g,ms_g}(t, t') \\
&\qquad + \sum_{ms_g \in dom(g)} \sum_{ms'' \notin dom(g)} f_{ms,ms''}(0, t) \cdot f_{ms'',ms_g}(t, t')
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{ms'_g \in dom(g)} f_{ms,ms'_g}(0,t) + \sum_{ms_g \in dom(g)} \sum_{ms'' \notin dom(g)} f_{ms,ms''}(0,t) \cdot f_{ms'',ms_g}(t,t') \\
&= \mathrm{val}_\pi(ms,t,g) + \underbrace{\sum_{ms_g \in dom(g)} \sum_{ms'' \notin dom(g)} f_{ms,ms''}(0,t) \cdot f_{ms'',ms'}(t,t')}_{\geqslant 0}
\end{aligned}
$$

Thus $\mathrm{val}_\pi(ms,t',g) \geqslant \mathrm{val}_\pi(ms,t,g)$.

□

We are now ready to define the second scheduler π' computed by `FindStrat`(v) and value v'. If Assumption 4.3.1 holds, we will provide bounds on the error between v' and the optimal value, as well as on the error introduced by following π' on an interval of length δ. Whenever the first scheduler π selected by `FindStrat`(v) is not very good and induces large error our algorithm is forced to decrease the step size more and more. In this case we will be decreasing the step size until it reaches certain minimal value. On this interval of minimal length we will use values v' as an approximation of the optimal values and scheduler π' as an approximation of the optimal scheduler.

Let $u : S \to [0,1]$, $\delta \in \mathbb{R}_{>0}, \varepsilon \in [0,1)$. For $x \in [0,\delta]$ we define

$$
\widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(s,x,u) := \begin{cases} u(s) & \text{if } x = 0 \\ e^{-E(s)\cdot\delta} \cdot u(s) + & \text{if } x = \delta, s \in MS \\ \quad (1 - e^{-E(s)\cdot\delta}) \sum\limits_{s' \in S} \frac{\mathbf{R}[s,s']}{E(s)} \cdot u(s') & \\ \mathrm{rea}^{\mathrm{opt}}_{\varepsilon}(s, \widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\delta,u)|_{MS}) & \text{if } x = \delta, s \in PS \\ \widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(s,\delta,u) & \text{if } x \in (0,\delta) \end{cases} \tag{4.39}
$$

In the following we will denote with $\widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\delta,u)$ the function over states $\widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\cdot,\delta,u)$. We will denote with $\widetilde{\pi}_\varepsilon(u)$ a stationary strategy that satisfies the following:

$$
\begin{aligned}
&\text{For } \mathrm{opt} = \sup : \mathrm{rea}^{\mathrm{opt}}_{\varepsilon}(s, \widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\delta,u)|_{MS}) \leqslant \mathrm{rea}^{\widetilde{\pi}_\varepsilon(u)}(s, \widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\delta,u)|_{MS}) \\
&\text{For } \mathrm{opt} = \inf : \mathrm{rea}^{\mathrm{opt}}_{\varepsilon}(s, \widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\delta,u)|_{MS}) + \varepsilon \geqslant \mathrm{rea}^{\widetilde{\pi}_\varepsilon(u)}(s, \widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(\delta,u)|_{MS})
\end{aligned}
$$

Finally, we define the output of procedure `FindStrat`(v) as follows: For $v = u|_{MS}$: `FindStrat`$(v) = (\pi, \widetilde{\pi}_\varepsilon(u))$. Below we will show error bounds on value $\widetilde{\mathrm{val}}^{\varepsilon}_{\mathrm{opt}}(s,x,u)$ and scheduler $\widetilde{\pi}_\varepsilon(u)$.

Lemma 4.3.5. *Let* $t \in \mathbb{R}_{\geqslant 0}, \delta \in \mathbb{R}_{>0}, \varepsilon \in (0,1), \varepsilon' \in [0,1)$. *Let* π *be a piecewise-constant strategy and* $v : S \to [0,1]$, *such that* $\forall s \in S : v(s) \preccurlyeq_{\text{opt}} \text{val}_\pi(s,t,g)$ *and*

$$v(s) \preccurlyeq_{\text{opt}} \text{val}_{\text{opt}}(s,t,g) \preccurlyeq_{\text{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon \tag{4.40}$$

If Assumption 4.3.1 is satisfied, then $\forall s \in S$:

$$\begin{aligned}
&\widetilde{\text{val}}^{\varepsilon'}_{\sup}(s,\delta,v) \leqslant \text{val}_{\sup}(s,t+\delta,g) \leqslant \widetilde{\text{val}}^{\varepsilon'}_{\sup}(s,\delta,v) + \varepsilon + \varepsilon' + \frac{(\mathbf{E}_{\max} \cdot \delta)^2}{2} \\
&\widetilde{\text{val}}^{\varepsilon'}_{\inf}(s,\delta,v) + \varepsilon' + \frac{(\mathbf{E}_{\max} \cdot \delta)^2}{2} \geqslant \text{val}_{\inf}(s,t+\delta,g) \geqslant \widetilde{\text{val}}^{\varepsilon'}_{\inf}(s,\delta,v) - \varepsilon
\end{aligned} \tag{4.41}$$

Proof. Let $w(s) = \widetilde{\text{val}}_{\text{opt}}(s,\delta,\text{val}_{\text{opt}}(t,g))$. Then

$$\text{val}_{\text{opt}}(s,t+\delta,g) - \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(s,\delta,v) = \text{val}_{\text{opt}}(s,t+\delta,g) - w(s) + w(s) - \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(s,\delta,v)$$

According to Lemma A.6, $\forall s \in S$:

$$0 \leqslant \text{val}_{\text{opt}}(s,t+\delta,g) - w(s) \leqslant \frac{(\mathbf{E}_{\max} \cdot \delta)^2}{2}$$

Consider the difference $w(s) - \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(s,\delta,v)$. For $ms \in MS$:

$$\begin{aligned}
w(ms) &= e^{-E(ms)\cdot\delta} \cdot \text{val}_{\text{opt}}(ms,t,g) + (1 - e^{-E(ms)\cdot\delta}) \sum_{s' \in S} \frac{\mathbf{R}[ms,s']}{E(ms)} \cdot \text{val}_{\text{opt}}(s',t,g) \\
\widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(ms,\delta,v) &= e^{-E(ms)\cdot\delta} \cdot v(ms) + (1 - e^{-E(ms)\cdot\delta}) \sum_{s' \in S} \frac{\mathbf{R}[ms,s']}{E(ms)} \cdot v(s')
\end{aligned}$$

And therefore:

$$\begin{aligned}
w(ms) - \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(ms,\delta,v) = {}& e^{-E(ms)\cdot\delta} \cdot (\text{val}_{\text{opt}}(ms,t,g) - v(ms)) \\
&+ (1 - e^{-E(ms)\cdot\delta}) \sum_{s' \in S} \frac{\mathbf{R}[ms,s']}{E(ms)} \cdot \big(\text{val}_{\text{opt}}(s',t,g) - v(s')\big)
\end{aligned}$$

From (4.40) it follows that:

$$0 \preccurlyeq_{\text{opt}} w(ms) - \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(ms,\delta,v) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon \tag{4.42}$$

Consider $ps \in PS$. In this case $w(ps) = \text{rea}^{\text{opt}}(ps, w|_{MS})$. On the other hand $\widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(ps,\delta,v) = \text{rea}^{\text{opt}}_{\varepsilon'}(ps, \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(\delta,v)|_{MS})$. Therefore

$$\begin{aligned}
w(ps) - \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(ps,\delta,v) &= \text{rea}^{\text{opt}}(ps, w|_{MS}) - \text{rea}^{\text{opt}}_{\varepsilon'}(ps, \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(\delta,v)|_{MS}) \\
&= \text{rea}^{\text{opt}}(ps, w|_{MS}) - \text{rea}^{\text{opt}}(ps, \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(\delta,v)|_{MS}) \\
&\quad + \text{rea}^{\text{opt}}(ps, \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(\delta,v)|_{MS}) - \text{rea}^{\text{opt}}_{\varepsilon'}(ps, \widetilde{\text{val}}^{\varepsilon'}_{\text{opt}}(\delta,v)|_{MS})
\end{aligned}$$

By definition:

$$0 \leqslant \mathrm{rea}^{\mathrm{opt}}(ps, \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta, v)|_{MS}) - \mathrm{rea}^{\mathrm{opt}}_{\varepsilon'}(ps, \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta, v)|_{MS}) \leqslant \varepsilon'$$

According to Lemma A.5 applied to $S' = MS$, $w_1 = w|_{MS}$ and $w_2 = \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta, v)|_{MS}$:

$$0 \preccurlyeq_{\mathrm{opt}} \mathrm{rea}^{\mathrm{opt}}(ps, w|_{MS}) - \mathrm{rea}^{\mathrm{opt}}(ps, \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta, v)|_{MS}) \preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\mathrm{inf}\}}(\mathrm{opt})} \cdot \varepsilon$$

Therefore we have shown that:

$$\begin{aligned} 0 &\leqslant w(ps) - \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{sup}}(ps, \delta, v) \leqslant \varepsilon + \varepsilon' \\ \varepsilon' &\geqslant w(ps) - \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{inf}}(ps, \delta, v) \geqslant -\varepsilon \end{aligned}$$

This concludes the proof of (4.41).

□

Lemma 4.3.6. *Let $t \in \mathbb{R}_{\geqslant 0}, \delta \in \mathbb{R}_{>0}, \varepsilon \in (0,1), \varepsilon' \in [0,1)$. Let π be a piecewise-constant strategy and $v : S \to [0,1]$, such that $\forall s \in S : v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_\pi(s,t,g)$ and*

$$v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_{\mathrm{opt}}(s,t,g) \preccurlyeq_{\mathrm{opt}} v(s) + (-1)^{\mathbb{1}_{\{\mathrm{inf}\}}(\mathrm{opt})} \cdot \varepsilon \tag{4.43}$$

If Assumption 4.3.1 is satisfied, then $\pi' = \pi|_{[0,t]} \cdot \widetilde{\pi}_{\varepsilon'}(v)|_{(t,t+\delta]}$ is $\left(\varepsilon + \varepsilon' + \frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2}\right)$-optimal for $\mathrm{val}_{\mathrm{opt}}(s, t+\delta, g)$.

Proof. We will show that

$$\begin{aligned} \text{if } \mathrm{opt} = \sup &: \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{sup}}(s,\delta,v) \leqslant \mathrm{val}_{\pi'}(s, t+\delta, g) \\ \text{if } \mathrm{opt} = \inf &: \widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{inf}}(s,\delta,v) + \varepsilon' + \frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2} \geqslant \mathrm{val}_{\pi'}(s, t+\delta, g) \end{aligned}$$

Due to Lemma 4.3.5 this implies the statement of the lemma. Consider time point $t, s \in S$. Then

$$\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(s,0,v) = v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_\pi(s,t,g) = \mathrm{val}_{\pi'}(s,t,g) \tag{4.44}$$

Consider time point $t+\delta$ and $ms \in MS$. Then

$$\begin{aligned} &\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(ms,\delta,v) \\ &= e^{-E(ms)\cdot\delta} \cdot v(ms) + (1 - e^{-E(ms)\cdot\delta}) \sum_{s' \in S} \frac{\mathbf{R}[ms,s']}{E(ms)} \cdot v(s') \\ &\overset{(4.44)}{\preccurlyeq_{\mathrm{opt}}} e^{-E(ms)\cdot\delta} \cdot \mathrm{val}_{\pi'}(ms,t,g) + \underbrace{(1 - e^{-E(ms)\cdot\delta}) \sum_{s' \in S} \frac{\mathbf{R}[ms,s']}{E(ms)} \cdot \mathrm{val}_{\pi'}(s',t,g)}_{X(ms,\pi',t,t+\delta)} \end{aligned} \tag{4.45}$$

Due to Lemma 4.3.4:

$$
\begin{aligned}
X(ms,\pi',t,t+\delta) &\leqslant \underbrace{\int_0^\delta E(ms)\cdot e^{-E(ms)\cdot x}\sum_{s'\in S}\frac{\mathbf{R}[ms,s']}{E(ms)}\cdot \mathrm{val}_{\pi'}(s',t+\delta-x,g)\,\mathrm{d}x}_{A(ms,\pi',t,t+\delta)}\\
&\leqslant X(ms,\pi',t,t+\delta)+\frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2}
\end{aligned}
\tag{4.46}
$$

And therefore

$$
\begin{aligned}
\widetilde{\mathrm{val}}^{\varepsilon'}_{\sup}(ms,\delta,v) &\overset{(4.45)}{\leqslant} e^{-E(ms)\cdot\delta}\cdot\mathrm{val}_{\pi'}(ms,t,g)+X(ms,\delta)\\
&\overset{(4.46)}{\leqslant} e^{-E(ms)\cdot\delta}\cdot\mathrm{val}_{\pi'}(ms,t,g)+A(ms,\delta)\\
&=\mathrm{val}_{\pi'}(ms,t+\delta,g)\\
\widetilde{\mathrm{val}}^{\varepsilon'}_{\inf}(ms,\delta,v)+\frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2} &\overset{(4.45)}{\geqslant} e^{-E(ms)\cdot\delta}\cdot\mathrm{val}_{\pi'}(ms,t,g)+X(ms,\delta)+\frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2}\\
&\overset{(4.46)}{\geqslant} e^{-E(ms)\cdot\delta}\cdot\mathrm{val}_{\pi'}(ms,t,g)+A(ms,\delta)\\
&=\mathrm{val}_{\pi'}(ms,t+\delta,g)
\end{aligned}
\tag{4.47}
$$

Consider now a probabilistic state ps:

$$
\begin{aligned}
\mathrm{val}_{\pi'}(ps,t+\delta,g) &= \mathrm{rea}^{\pi'}(ps,\mathrm{val}_{\pi'}(t+\delta,g)|_{MS})\\
\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(ps,\delta,v) &= \mathrm{rea}^{\mathrm{opt}}_{\varepsilon'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})
\end{aligned}
$$

Therefore

$$
\begin{aligned}
&\mathrm{val}_{\pi'}(ps,t+\delta,g)-\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(ps,\delta,v)\\
&\quad=\mathrm{rea}^{\pi'}(ps,\mathrm{val}_{\pi'}(t+\delta,g)|_{MS})-\mathrm{rea}^{\pi'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})\\
&\qquad+\mathrm{rea}^{\pi'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})-\mathrm{rea}^{\mathrm{opt}}_{\varepsilon'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})
\end{aligned}
$$

If opt = inf, then:

$$
\begin{aligned}
&\mathrm{rea}^{\pi'}(ps,\mathrm{val}_{\pi'}(t+\delta,g)|_{MS})-\mathrm{rea}^{\pi'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS}) \overset{(4.47)}{\leqslant} \frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2}\\
&\mathrm{rea}^{\pi'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})-\mathrm{rea}^{\mathrm{opt}}_{\varepsilon'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})\leqslant\varepsilon'
\end{aligned}
$$

If opt = sup, then:

$$
\begin{aligned}
&\mathrm{rea}^{\pi'}(ps,\mathrm{val}_{\pi'}(t+\delta,g)|_{MS})-\mathrm{rea}^{\pi'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS}) \overset{(4.47)}{\geqslant} 0\\
&\mathrm{rea}^{\pi'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})-\mathrm{rea}^{\mathrm{opt}}_{\varepsilon'}(ps,\widetilde{\mathrm{val}}^{\varepsilon'}_{\mathrm{opt}}(\delta,v)|_{MS})\geqslant 0
\end{aligned}
$$

This concludes the proof.

□

4.3.3 Switching Points

In the previous section we have shown that given a piecewise-constant strategy $\pi_{\texttt{opt}}$ that is optimal for $\text{val}_{\text{opt}}(t, g)$ we can extend it with a stationary strategy π, such that it becomes optimal for $\text{val}_{\text{opt}}(t + x, g)$, $\forall x \in (0, \delta]$. However, we do not know the value of this δ and this is the question that we investigate in this section. The main result of the section is a sufficient condition for the nearest switching point of an optimal strategy.

Our goal is to find δ such that for all $x \in (t, t + \delta]$ scheduler $\pi'_{\texttt{opt}} = \pi_{\texttt{opt}}|_{[0,t]} \cdot \pi|_{(t,t+\delta]}$ is optimal for $\text{val}_{\text{opt}}(x, g)$, i. e.:

$$\forall s \in S, x \in (t,t + \delta], \pi \in \Pi_{\texttt{PC}} : \text{val}_{\pi'_{\text{opt}}}(s, x, g) \succcurlyeq_{\text{opt}} \text{val}_{\pi}(s, x, g)$$

Naturally, it is unrealistic to check the inequality above. It turns out, however, that given a strategy that is optimal on an interval $[0, x)$ it is cheap to make sure that the strategy remains optimal for x as well. Under this condition the value $\text{val}_{\pi'_{\text{opt}}}(ms, x, g)$ for a Markovian state ms is optimal, because it only depends on reachability values $\text{val}_{\pi'_{\text{opt}}}(s, x', g)$ for time points $x' < x$. We, therefore, need only to look at probabilistic states. Here we want to make sure that

$$\begin{gathered}\forall ps \in PS, \pi' \in \Pi_{\texttt{stat}}, \pi = \pi'_{\texttt{opt}}|_{[0,x)} \cdot \pi'|_{[x,x]} : \\ \text{val}_{\pi'_{\text{opt}}}(ps, x, g) \succcurlyeq_{\text{opt}} \text{val}_{\pi}(ps, x, g)\end{gathered} \tag{4.48}$$

Still checking this inequality for all stationary strategies means performing in the worst case exponentially many operations in the size of the Markov automaton[3]. This is not practical and in the following we discuss how this can be improved. According to Lemma 4.3.1, Remark 4.3.1 and (4.18):

$$\begin{aligned}\text{val}_{\pi'_{\text{opt}}}(ps, x, g) &= \sum_{s' \in S} \mathbb{P}[ps, \pi'_{\texttt{opt}}(ps, x), s'] \cdot \text{val}_{\pi'_{\text{opt}}}(s', x, g) \\ &= \text{rea}^{\pi'_{\texttt{opt}}|_{[x,x]}}(ps, \text{val}_{\pi'_{\text{opt}}}(x, g)|_{MS})\end{aligned}$$

The following lemma describes one important property of optimal stationary strategy for $\text{rea}^{\text{opt}}(h)$, where h is some goal function. The result is originally presented in [Put94] (Theorem 7.2.5) and is adapted here to the special case of $\text{rea}^{\text{opt}}(ps, h)$:

Lemma 4.3.7 ([Put94])**.** *Let h be a goal function. A stationary strategy π is optimal for* $\text{rea}^{\text{opt}}(h)$ *iff*

$$\forall ps \in PS : \text{rea}^{\pi}(ps, h) = \underset{\alpha \in Act(ps)}{\text{opt}} \sum_{s' \in S} \mathbb{P}[ps, \alpha, s'] \cdot \text{rea}^{\pi}(s', h) \tag{4.49}$$

[3]If each probabilistic state has at least 2 enabled actions, then there are at least $2^{|PS|}$ different stationary strategies.

Taking into account Lemma 4.3.7 and the fact that $\pi'_{\mathtt{opt}}|_{[x,x]} = \pi$ we can rewrite (4.48) as follows:

$$\begin{aligned}
&\forall ps \in PS: \\
&\quad \mathrm{rea}^{\pi}(ps, \mathrm{val}_{\pi'_{\mathrm{opt}}}(x,g)|_{MS}) = \operatorname*{opt}_{\alpha \in Act(ps)} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot \mathrm{rea}^{\pi}(s', \mathrm{val}_{\pi'_{\mathrm{opt}}}(x,g)|_{MS})
\end{aligned} \tag{4.50}$$

$$\begin{aligned}
&\iff \\
&\forall ps \in PS, \alpha \in Act(ps): \\
&\quad \mathrm{rea}^{\pi}(ps, \mathrm{val}_{\pi'_{\mathrm{opt}}}(x,g)|_{MS}) \succcurlyeq_{\mathrm{opt}} \sum_{s' \in S} \mathbb{P}[ps,\alpha,s'] \cdot \mathrm{rea}^{\pi}(s', \mathrm{val}_{\pi'_{\mathrm{opt}}}(x,g)|_{MS})
\end{aligned} \tag{4.51}$$

The latter inequality needs to be checked linearly many times, once for each probabilistic state and each enabled action of this state, as opposed to (4.48) that required exponentially many checks.

However we still need to address another problem: It is not possible to check (4.51) for all points x within interval $(t, t+\delta]$, since there are continuously many of those. We can circumvent this issue by thinking of the right- and the left-hand sides of the inequality as functions of x. Lets denote them with $r(ps,\alpha,x)$ and $l(ps,x)$ respectively. This way δ is simply the largest value of variable x, for which $l(ps,x) - r(ps,\alpha,x) \succcurlyeq_{\mathrm{opt}} 0$ for all ps and α. Functions $l(ps,x)$ and $r(ps,\alpha,x)$ are implicitly given by systems of differential equations (4.21-4.22) and one can use standard function analysis to find zeros of $l(ps,x) - r(ps,\alpha,x)$. In fact, in the next section we will apply derivative analysis to approximations of these functions to find the maximum of the difference and compare it to zero.

There is another way of interpreting (4.51). If (4.51) holds on an interval, then there exists an optimal strategy that has no switching points on this interval. And when the condition is violated, there may exist one. The lemma below summarises the discussion above and is essentially a sufficient condition on the absence of switching points on an interval. But first, we will introduce a more concise representation of the right-hand side of (4.51):

$$\begin{aligned}
&\mathrm{val}_{\pi, ps\to\alpha}(ps,x,g) \\
&\qquad := \sum_{s'' \in S} \mathbb{P}[ps,\alpha,s''] \cdot \mathrm{rea}^{\pi|_{[x,x]}}(s'', \mathrm{val}_{\pi}(x,g)|_{MS}) \\
&\qquad = \sum_{s'' \in S} \mathbb{P}[ps,\alpha,s''] \cdot \mathrm{rea}^{\pi}(s'', \mathrm{val}_{\pi}(x,g)|_{MS}) \\
&\qquad \overset{(4.18)}{=} \sum_{s'' \in S} \mathbb{P}[ps,\alpha,s''] \sum_{s' \in MS} \mathrm{rea}^{\pi}(s'',s') \cdot \mathrm{val}_{\pi}(s',x,g) \\
&\qquad = \sum_{s' \in MS} \underbrace{\sum_{s'' \in S} \mathbb{P}[ps,\alpha,s''] \cdot \mathrm{rea}^{\pi}(s'',s')}_{\mathrm{rea}^{\pi,ps\to\alpha}(ps,s')} \cdot \mathrm{val}_{\pi}(s',x,g)
\end{aligned} \tag{4.52}$$

We will denote with $\mathrm{rea}_{\varepsilon}^{\pi,ps\to\alpha}(ps,s')$ an ε-close under-approximation of value $\mathrm{rea}^{\pi,ps\to\alpha}(ps,s')$.

Lemma 4.3.8 (Sufficient Condition)**.** *Let* $t \in \mathbb{R}_{\geqslant 0}, \delta \in \mathbb{R}_{>0}, \pi_{\texttt{opt}} \in \Pi_{\texttt{PC}}$ *is optimal for* $\mathrm{val}_{\mathrm{opt}}(x, g), x \in [0, t]$*,* π *is a stationary strategy and* $\pi'_{\texttt{opt}} = \pi_{\texttt{opt}}|_{[0,t]} \cdot \pi|_{(t,t+\delta]}$*. If*

$$\forall ps \in PS, x \in (t, t+\delta], \alpha \in Act(ps) : \mathrm{val}_{\pi'_{\texttt{opt}}}(ps, x, g) \succcurlyeq_{\mathrm{opt}} \mathrm{val}_{\pi'_{\texttt{opt}}, ps \to \alpha}(ps, x, g) \tag{4.53}$$

then $\pi'_{\texttt{opt}}$ *is optimal for* $\mathrm{val}_{\mathrm{opt}}(x, g)$*, for all* $x \in (t, t+\delta]$*.*

Proof. Assume this is not the case, i. e. there exist $x \in (t, t+\delta], s \in S$, such that $\pi'_{\texttt{opt}}$ is not optimal for $\mathrm{val}_{\mathrm{opt}}(s, x, g)$. Let x_0 be the smallest of all such values x and s_0 be the respective state. Then $\pi'_{\texttt{opt}}$ is optimal for all $x' \in (t, x_0)$. This implies that s is not Markovian, since value $\mathrm{val}_{\pi'_{\texttt{opt}}}(s, x', g)$ for a Markovian state s depends only on values $\mathrm{val}_{\pi'_{\texttt{opt}}}(x'', g)$ for $x'' < x'$, and according to our assumption $\mathrm{val}_{\pi'_{\texttt{opt}}}(x'', g) = \mathrm{val}_{\mathrm{opt}}(x'', g)$.

Consider s to be probabilistic. Condition (4.53) implies that Lemma 4.3.7 holds for the stationary scheduler $\pi'_{\texttt{opt}}|_{[x_0,x_0]} = \pi$ and goal function $h = \mathrm{val}_{\mathrm{opt}}(x_0, g)|_{MS}$. Therefore $\pi'_{\texttt{opt}}|_{[x_0,x_0]}$ is optimal for $\mathrm{rea}^{\mathrm{opt}}(\mathrm{val}_{\mathrm{opt}}(x_0, g)|_{MS})$.

Thus we have shown that for a Markovian state ms value $\mathrm{val}^{ms}_{\pi'_{\texttt{opt}}}(x_0, g)$ satisfies the fixpoint characterisation 4.1.4 for x_0. For a probabilistic state ps value $\mathrm{val}^{ps}_{\pi'_{\texttt{opt}}}(x_0, g) = \mathrm{rea}^{\pi'_{\texttt{opt}}|_{[x_0,x_0]}}(ps, \mathrm{val}_{\mathrm{opt}}(x_0, g)|_{MS})$ satisfies the fixpoint characterisation 4.1.4 for x_0 due to (4.20). This implies that $\pi'_{\texttt{opt}}$ is optimal for x_0. We arrived at a contradiction that proves the claim of the lemma. □

4.3.4 Approximation of Switching Points

In the previous section we have shown that one can compute the switching points of an optimal piecewise-constant scheduler by finding intersections of several functions, given implicitly as systems of differential equations. Notice, however, that these numbers are not necessarily rational. We, therefore, start this section by showing how to approximate the switching points, rather than compute them exactly. The end of the section is devoted to bounding the error introduced by the approximations.

We will approximate switching points iteratively. In order to find the nearest switching point on the interval $(t, b]$ we define a sequence $t = t_0, t_1, \ldots, t_n = b$, and at each iteration $k = 0..n-1$ we check whether (4.53) holds for $(t_k, t_{k+1}]$. If yes, we increase k and repeat. Otherwise, we output the largest $\delta \in (t_k - t, t_{k+1} - t]$ such that (4.53) holds for $(t, t+\delta]$.

Selecting t_k. This step is essentially a heuristic. The correctness of our approach, in general, does not depend on the choices of t_k and essentially any other heuristic can as well be used. However all the results presented below, as well as all the algorithms presented in Section 4.3.5 are targeted towards the specific heuristic that we define below.

We only consider the case of $t < b$. The heuristic has two parameters $\varepsilon_h \in (0, 1)$ and $\delta_h \geqslant 0$. At every iteration $k = 0..n-1$ we choose a value t_{k+1}, such that the MA is very likely to perform at most $k + 1$ Markovian transitions within $t_{k+1} - t$ time units. “Very likely” here means with probability $1 - \varepsilon_h$. Parameter δ_h is introduced for scenarios when the difference $t_{k+1} - t$ is unacceptably small, for example, when it is lower than the smallest floating point number that a computer can store. We will bound the values $t_{k+1} - t$ from below with δ_h, unless $n = 1$ (which is the case if $b - t < \delta_h$). For $k = 1..n$ we first define $T_\Psi(k, \varepsilon_h)$ as follows:

$$T_\Psi(k, \varepsilon_h), \text{ is the largest value that satisfies } \sum_{i=0}^{k} \Psi_{\mathbf{E}_{\max}\cdot T_\Psi(k,\varepsilon_h)}(i) \geqslant 1 - \varepsilon_h$$

It may not be possible to compute the exact value of $T_\Psi(k, \varepsilon_h)$. In this case an under-approximation thereof up to a certain tolerance value suffices.

Procedure `ComputeIntervals` shown in Algorithm 1 produces a partition of interval $(t, b]$ for given t and b. Informally, the procedure considers a sequence of values $\tau_j = T_\Psi(j, \varepsilon_h)$ that generates a sequence of intervals $\{(\tau_j, \tau_{j+1}]\}$. This sequence is further refined in such a way that each of the intervals is a subset of $(i/\mathbf{E}_{\max}, (i + 1)/\mathbf{E}_{\max}]$, for some i (step 4). This is performed for efficiency reasons. Finally, the distance between each value δ_k and the given t cannot be smaller than parameter δ_h. For this reason smaller values are increased to be at least as large (step 7).

Algorithm 1 `ComputeIntervals`

```
Input: $b \in \mathbb{R}_{>0}$, $t \in [0, b)$
Output: partition of interval $(t, b]$
Parameters: $\delta_h > 0, \varepsilon_h \in (0, 1)$
 1: $t_0 = t, \delta_0 = 0$
 2: $k = 1, j = 1$
 3: do
 4:     $c_k = \min\{T_\Psi(j, \varepsilon_h), (\lfloor \delta_{k-1} \cdot \mathbf{E}_{\max} \rfloor + 1)/\mathbf{E}_{\max}\}$
 5:     if $(c_k == \delta_{k-1})$ then $c_k = T_\Psi(j, \varepsilon_h)$
 6:     if $(c_k == T_\Psi(j, \varepsilon_h))$ then $j = j + 1$
 7:     $\Delta_k = \max\{\delta_h, c_k\}$
 8:     if $(t + \Delta_k \leqslant b)$ then
 9:         $\delta_k = \Delta_k$
10:     else
11:         $\delta_k = b - t$
12:     $k = k + 1$
13: while $(t + \delta_{k-1} < b)$
14: return all the non-empty intervals from the set $\{(t + \delta_i, t + \delta_{i+1}] \mid i = 0..k - 2\}$
```

Below we prove a few properties of this heuristic that will be needed later in this Section.

Lemma 4.3.9. *Let $\delta_h \leqslant 1/\mathbf{E}_{\max}$. Consider $\delta_0, \ldots, \delta_n$ computed in* `ComputeIntervals` *(lines 9-11). Then $\forall k = 0..n-1 : \exists j \leqslant k : (\delta_k, \delta_{k+1}] \subseteq (j/\mathbf{E}_{\max}, (j+1)/\mathbf{E}_{\max}]$*

Proof. We prove the statement by induction over k. Consider $k = 0$. Then $\delta_0 = 0$ and if $n > 1$, then $\delta_1 \leqslant 1/\mathbf{E}_{\max}$. If $n = 1$, then $\delta_1 = b - t < \delta_h \leqslant 1/\mathbf{E}_{\max}$, and thus the statement holds for $j = 0$. Let $k > 0$. By induction hypothesis $\exists j$ that satisfies $0 \leqslant j-1 \leqslant k-1 : (\delta_{k-1}, \delta_k] \subseteq ((j-1)/\mathbf{E}_{\max}, j/\mathbf{E}_{\max}]$. Thus $(j-1)/\mathbf{E}_{\max} \leqslant \delta_k \leqslant j/\mathbf{E}_{\max}$.

If $\delta_k < j/\mathbf{E}_{\max}$, then $\lfloor \delta_k \cdot \mathbf{E}_{\max} \rfloor < j$ and therefore $\lfloor \delta_k \cdot \mathbf{E}_{\max} \rfloor \leqslant j-1$. This implies that $c_{k+1} \leqslant j/\mathbf{E}_{\max}$ and $\delta_{k+1} \leqslant \max\{1/\mathbf{E}_{\max}, j/\mathbf{E}_{\max}\} \leqslant j/\mathbf{E}_{\max}$, since $j \geqslant 1$. Therefore $(\delta_k, \delta_{k+1}] \subseteq ((j-1)/\mathbf{E}_{\max}, j/\mathbf{E}_{\max}]$.

If $\delta_k = j/\mathbf{E}_{\max}$, then $\lfloor \delta_k \cdot \mathbf{E}_{\max} \rfloor = j$ and therefore $c_{k+1} \leqslant (j+1)/\mathbf{E}_{\max}$. This implies that $\delta_{k+1} \leqslant \max\{1/\mathbf{E}_{\max}, (j+1)/\mathbf{E}_{\max}\} \leqslant (j+1)/\mathbf{E}_{\max}$, since $j \geqslant 1$. Therefore $(\delta_k, \delta_{k+1}] \subseteq (j/\mathbf{E}_{\max}, (j+1)/\mathbf{E}_{\max}]$. $\square$

Lemma 4.3.10. *Let $(t_0, t_1], \ldots, (t_{n-1}, t_n]$ be the partition of $(t, b]$ computed by procedure* `ComputeIntervals` *for $\varepsilon_h \in (0,1)$ and $\delta_h \leqslant 1/\mathbf{E}_{\max}$. Then for $\delta \in (t_0 - t, t_1 - t] : 0 \leqslant N(\delta, \varepsilon_h) \leqslant N(t_1 - t, \varepsilon_h)$ and $\forall k = 1..n-1, \delta \in (t_k - t, t_{k+1} - t] : N(t_k - t, \varepsilon_h) \leqslant N(\delta, \varepsilon_h) \leqslant N(t_k - t, \varepsilon_h) + 1$.*

Proof. First of all, $\forall k = 0..n-1, \delta \in (t_k - t, t_{k+1} - t] : N(t_k - t, \varepsilon_h) \leqslant N(\delta, \varepsilon_h) \leqslant N(t_{k+1} - t, \varepsilon_h)$. This proves the first claim and the first inequality of the second claim. Consider $k \geqslant 1$. For $t_k - t$ there exists $j \in \mathbb{Z}_{>0}$, such that $t_k - t$ lies within $[T_\Psi(j, \varepsilon_h), T_\Psi(j+1, \varepsilon_h))$. The value of $t_{k+1} - t$ belongs either to the same interval $[T_\Psi(j, \varepsilon_h), T_\Psi(j+1, \varepsilon_h))$, or it equals $T_\Psi(j+1, \varepsilon_h)$. By definition of $T_\Psi(j, \varepsilon_h)$: $\sum_{i=0}^{j} \Psi_{\mathbf{E}_{\max} \cdot T_\Psi(j, \varepsilon_h)}(i) \geqslant 1 - \varepsilon_h$ and therefore we can set $N(t_j - t, \varepsilon_h)$ to j. Thus the difference between $N(t_k - t, \varepsilon_h)$ and $N(t_{k+1} - t, \varepsilon_h)$ is at most 1. $\square$

Lemma 4.3.11. *Let $\varepsilon_h \in (0,1)$ and $\delta_h \leqslant 1/\mathbf{E}_{\max}$, then the number of intervals in the partition returned by* `ComputeIntervals`(b, t) *is in*

$$O((\mathbf{E}_{\max} \cdot b \cdot e^2 - \ln(\varepsilon_h)) \cdot (b-t)/\mathbf{E}_{\max})$$

Proof. Assume that variable j reaches the value $N = N(b, \varepsilon_h)$. Then $T_\Psi(N, \varepsilon_h) \geqslant b$. If $\min\{T_\Psi(N, \varepsilon_h), (\lfloor \delta_{k-1} \cdot \mathbf{E}_{\max} \rfloor + 1)/\mathbf{E}_{\max}\}$ is reached at $T_\Psi(N, \varepsilon_h)$, then the algorithm exists the while-loop. In the worst case all points $T_\Psi(i, \varepsilon_h), i = 1..N$ are part of the partition. Each interval $(T_\Psi(i-1, \varepsilon_h), T_\Psi(i, \varepsilon_h)]$ can be partitioned with steps of length at least $1/\mathbf{E}_{\max}$, except possibly the very first and the very last intervals. Therefore each interval $(T_\Psi(i-1, \varepsilon_h), T_\Psi(i, \varepsilon_h)]$ is partitioned in $O(\lceil (b-t)/\mathbf{E}_{\max} \rceil)$ of smaller intervals. Thus overall the amount of intervals in the partition produced by `ComputeIntervals` does not exceed

$$O(N(b, \varepsilon_h) \cdot (b-t)/\mathbf{E}_{\max}) = O((\mathbf{E}_{\max} \cdot b \cdot e^2 - \ln(\varepsilon_h)) \cdot (b-t)/\mathbf{E}_{\max})$$

□

Searching for switching points within $(t_k, t_{k+1}]$**.** In the following we will develop efficiently checkable sufficient conditions for testing an interval $(t_k, t_{k+1}]$ against the existence of switching points. We do this with the help of (4.53). In order to check whether $\mathrm{val}_\pi(ps, t+\delta, g) \succcurlyeq_{\mathrm{opt}} \mathrm{val}_{\pi,ps\to\alpha}(ps, t+\delta, g)$ for *all* $\delta \in (\delta_k, \delta_{k+1}]$, $\delta_i = t_i - t$, we only have to check whether the maximum of function

$$\mathrm{diff}_\pi^{\mathrm{opt}}(ps, \alpha, (t, t+\delta]) := \begin{cases} \mathrm{val}_{\pi,ps\to\alpha}(ps, t+\delta, g) - \mathrm{val}_\pi(ps, t+\delta, g) & \text{if opt} = \sup \\ (-1) \cdot \mathrm{diff}_\pi^{\sup}(ps, \alpha, (t, t+\delta]) & \text{if opt} = \inf \end{cases}$$

(as a function of δ) is at most 0 on this interval for all $ps \in PS, \alpha \in Act(ps)$. We will first build an over-approximation of $\mathrm{diff}_\pi^{\mathrm{opt}}(ps, \alpha, (t, t+\delta])$ that is a function of exponentials and polynomials. Having this representation, we apply derivative analysis to it to compute its maximum over the interval $(t_k - t, t_{k+1} - t]$. In order to find an over-approximation of $\mathrm{diff}_\pi^{\mathrm{opt}}(ps, \alpha, (t, t+\delta])$ we work on the approximation of $\mathrm{val}_{\pi,ps\to\alpha}(ps, t+\delta, g)$ and $\mathrm{val}_\pi(ps, t+\delta, g)$ derived from Lemma 4.3.2.

Lemma 4.3.12. *Let* $t \in \mathbb{R}_{\geqslant 0}, \xi = (\varepsilon_\Psi, \varepsilon_{rea}, \varepsilon_n)$ *satisfy conditions of Lemma 4.3.2,* $(t_k, t_{k+1}] \in \texttt{ComputeIntervals}(b, t, \varepsilon_h = \varepsilon_\Psi, \delta_h \in (0, 1/\mathbf{E}_{\max}])$, $\delta \in (t_k - t, t_{k+1} - t], \varepsilon \in (0,1)$, *and* $v : S \to [0,1]$ *is such that* $\forall s \in S : v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_\pi(s, t, g) \preccurlyeq_{\mathrm{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon$. *Then*

$$\mathrm{diff}_\pi^{\mathrm{opt}}(ps, \alpha, (t, t+\delta]) \leqslant \max_{N_k \leqslant j \leqslant N_{k+1}} \sum_{i=0}^{j} e^{-\mathbf{E}_{\max}\cdot\delta} \frac{(\mathbf{E}_{\max}\cdot\delta)^i}{i!} \cdot B_{\pi,\xi}^{\mathrm{opt},i}(ps, \alpha) + C, \tag{4.54}$$

where $N_j = N(t_j - t, \varepsilon_\Psi), j \in \{k, k+1\}, C := 2\cdot\varepsilon_n + \varepsilon_{rea} + \varepsilon_\Psi$,

$$\begin{aligned} &B_{\pi,\xi}^{\mathrm{opt},i}(ps, \alpha) \\ &\quad := (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \left(\mathrm{rea}_{\varepsilon_n}^{\pi|_I, ps\to\alpha}(ps, \mathbf{D}_{\varepsilon_n}^i(\pi|_I, v)|_{MS}) - \mathrm{rea}_{\varepsilon_n}^{\pi|_I}(ps, \mathbf{D}_{\varepsilon_n}^i(\pi|_I, v)|_{MS})\right) \end{aligned}$$

Proof. Let $u_{\pi,ps\to\alpha,\xi}^{t,v}(ps) = \sum_{s'\in S} \mathbb{P}[ps, \alpha, s'] \cdot u_{\pi,\xi}^{t,v}(s')$. We will first show that

$$\mathrm{diff}_\pi^{\mathrm{opt}}(ps, \alpha, (t, t+\delta]) \leqslant (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \left(u_{\pi,ps\to\alpha,\xi}^{t,v}(ps) - u_{\pi,\xi}^{t,v}(ps)\right) + \varepsilon_{\mathrm{rea}} + \varepsilon_\Psi \tag{4.55}$$

By definition

$$\mathrm{diff}_\pi^{\mathrm{opt}}(ps, \alpha, (t, t+\delta]) = (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \left(\mathrm{val}_{\pi,ps\to\alpha}(ps, t+\delta, g) - \mathrm{val}_\pi(ps, t+\delta, g)\right)$$

Consider $\mathrm{val}_\pi(ps, t+\delta, g)$:

$$\begin{aligned} \mathrm{val}_\pi(ps, t+\delta, g) &= \sum_{s'\in dom(g)} \mathrm{Pr}_{\pi,ps}^{\mathcal{M}}\left[\texttt{tt}\,\mathrm{U}^{=t+\delta} ap_{s'}\right] \cdot g(s') \\ &= \sum_{s''\in MS} \mathrm{Pr}_{\pi,ps}^{\mathcal{M}}\left[\texttt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\right] \sum_{s'\in dom(g)} \mathrm{Pr}_{\pi^{-\delta},s''}^{\mathcal{M}}\left[\texttt{tt}\,\mathrm{U}^{=t} ap_{s'}\right] \cdot g(s') \end{aligned}$$

$$= \sum_{s'' \in MS} \underbrace{\Pr^{\mathcal{M}}_{\pi,ps}\left[\texttt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\right]}_{p(ps,s'')} \cdot \mathrm{val}_{\pi^{-\delta}}(s'', t, g)$$

Notice that scheduler π is shifted with $-\delta$ since starting from Section 4.2.1 the schedulers depend on time left until the time bound, rather time passed from the beginning. Analogously derivations can be performed for $\mathrm{val}_{\pi,ps\to\alpha}(ps, t+\delta, g)$:

$$\mathrm{val}_{\pi}(ps, t+\delta, g) = \sum_{s'' \in MS} \sum_{s' \in S} \mathbb{P}[ps, \alpha, s'] \cdot p(s', s'') \cdot \mathrm{val}_{\pi^{-\delta}}(s'', t, g)$$

For opt = sup:

$$\begin{aligned}
&-\mathrm{diff}^{\mathrm{opt}}_{\pi}(ps, \alpha, (t, t+\delta])\\
&= \mathrm{val}_{\pi}(ps, t+\delta, g') - \mathrm{val}_{\pi,ps\to\alpha}(ps, t+\delta, g')\\
&= \sum_{s'' \in MS} \left(p(ps, s'') - \sum_{s' \in S} \mathbb{P}[ps, \alpha, s'] \cdot p(s', s'') \right) \cdot \mathrm{val}_{\pi^{-\delta}}(s'', t, g)\\
&\geqslant \sum_{s'' \in MS} \left(p(ps, s'') - \sum_{s' \in S} \mathbb{P}[ps, \alpha, s'] \cdot p(s', s'') \right) \cdot v(s'')\\
&\overset{\text{Lemma 4.3.2}}{\geqslant} u^{\delta,v}_{\pi,\xi}(ps) - \sum_{s' \in S} \mathbb{P}[ps, \alpha, s'] \cdot u^{\delta,v}_{\pi,\xi}(s') - \varepsilon_{\mathrm{rea}} - \varepsilon_{\Psi}\\
&= u^{\delta,v}_{\pi,\xi}(ps) - u^{t,v}_{\pi,ps\to\alpha,\xi}(ps) - \varepsilon_{\mathrm{rea}} - \varepsilon_{\Psi}
\end{aligned}$$

And analogously for opt = inf. We have thus proven (4.55).

Next we will obtain a bound on value $u^{t,v}_{\pi,ps\to\alpha,\xi}(ps) - u^{t,v}_{\pi,\xi}(ps)$:

$$\begin{aligned}
&\sum_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathrm{rea}^{\pi|_I}_{\varepsilon_{\mathrm{n}}}(ps, \mathbf{D}^{i}_{\varepsilon_{\mathrm{n}}}(\pi|_I, v)|_{MS})\\
&= \sum_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\mathrm{rea}^{\pi|_I}(ps, \mathbf{D}^{i}_{\varepsilon_{\mathrm{n}}}(\pi|_I, v)|_{MS}) - \underbrace{\varepsilon'}_{\leqslant\varepsilon_{\mathrm{n}}} \right)\\
&\geqslant \sum_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\mathrm{rea}^{\pi|_I}(ps, \mathbf{D}^{i}_{\varepsilon_{\mathrm{n}}}(\pi|_I, v)|_{MS}) \right) - \varepsilon_{\mathrm{n}}\\
&= \sum_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \sum_{ms \in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot \mathbf{D}^{i}_{\varepsilon_{\mathrm{n}}}(ms, \pi|_I, v) - \varepsilon_{\mathrm{n}}\\
&= \sum_{ms \in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot \sum_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^{i}_{\varepsilon_{\mathrm{n}}}(ms, \pi|_I, v) - \varepsilon_{\mathrm{n}}\\
&= \sum_{ms \in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot u^{t+\delta,v}_{\pi,\xi}(ms) - \varepsilon_{\mathrm{n}} = \mathrm{rea}^{\pi|_I}(ps, u^{t+\delta,v}_{\pi,\xi}|_{MS}) - \varepsilon_{\mathrm{n}}\\
&\geqslant u^{t+\delta,v}_{\pi,\xi}(ps) - \varepsilon_{\mathrm{n}}
\end{aligned}$$

And therefore

$$u^{t+\delta,v}_{\pi,\xi}(ps) \leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathrm{rea}^{\pi|_I}_{\varepsilon_\mathrm{n}}(ps, \mathbf{D}^i_{\varepsilon_\mathrm{n}}(\pi|_I, v)|_{MS}) + \varepsilon_\mathrm{n} \tag{4.56}$$

Next we obtain a lower bound:

$$\begin{aligned}
&\sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathrm{rea}^{\pi|_I}_{\varepsilon_\mathrm{n}}(ps, \mathbf{D}^i_{\varepsilon_\mathrm{n}}(\pi|_I, v)|_{MS}) \\
&\quad= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\mathrm{rea}^{\pi|_I}(ps, \mathbf{D}^i_{\varepsilon_\mathrm{n}}(\pi|_I, v)|_{MS}) - \underbrace{\varepsilon'}_{\leqslant\varepsilon_\mathrm{n}}\right) \\
&\quad\leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\mathrm{rea}^{\pi|_I}(ps, \mathbf{D}^i_{\varepsilon_\mathrm{n}}(\pi|_I, v)|_{MS})\right) \\
&\quad= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\sum_{ms\in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ms, \pi|_I, v)\right) \\
&\quad= \sum_{ms\in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ms, \pi|_I, v) \\
&\quad= \sum_{ms\in MS} \mathrm{rea}^{\pi|_I}(ps, ms) \cdot u^{t+\delta,v}_{\pi,\xi}(ms) = \mathrm{rea}^{\pi|_I}(ps, u^{t+\delta,v}_{\pi,\xi}|_{MS}) \\
&\quad\leqslant \mathrm{rea}^{\pi|_I}_{\varepsilon_\mathrm{n}}(ps, u^{t+\delta,v}_{\pi,\xi}|_{MS}) + \varepsilon_\mathrm{n} \\
&\quad= u^{t+\delta,v}_{\pi,\xi}(ps) + \varepsilon_\mathrm{n}
\end{aligned}$$

And therefore:

$$u^{t+\delta,v}_{\pi,\xi}(ps) \geqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathrm{rea}^{\pi|_I}_{\varepsilon_\mathrm{n}}(ps, \mathbf{D}^i_{\varepsilon_\mathrm{n}}(\pi|_I, v)|_{MS}) - \varepsilon_\mathrm{n} \tag{4.57}$$

Analogous results hold for $u^{t+\delta,v}_{\pi,ps\to\alpha,\xi}(ps)$. Therefore

$$u^{t+\delta,v}_{\pi,ps\to\alpha,\xi}(ps) - u^{t+\delta,v}_{\pi,\xi}(ps) \leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot B^{\sup,i}_{\pi,\xi}(ps,\alpha) + 2\cdot\varepsilon_\mathrm{n}$$

We can now rewrite the bound on value $\mathrm{diff}^{\mathrm{opt}}_\pi(ps, \alpha, (t, t+\delta])$ as follows:

$$\mathrm{diff}^{\mathrm{opt}}_\pi(ps, \alpha, (t, t+\delta]) \leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) + C$$

Notice that for $\delta \in (t_k - t, t_{k+1} - t] : N_k \leqslant N(\delta, \varepsilon_\Psi) \leqslant N_{k+1}$. Therefore

$$\mathrm{diff}^{\mathrm{opt}}_\pi(ps, \alpha, (t, t+\delta]) \leqslant \max_{N_k\leqslant j\leqslant N_{k+1}} \sum_{i=0}^{j} e^{-\mathbf{E}_{\max}\cdot\delta} \frac{(\mathbf{E}_{\max}\cdot\delta)^i}{i!} \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) + C \tag{4.58}$$

□

Notice that both $B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)$ and C are constants in terms of δ. However, the right-hand side of the bound obtained above still depends on specific δ, which means it is not possible to compute it efficiently for all $\delta \in (t_k - t, t_{k+1} - t]$.

In order to solve this problem we search for supremum of the right-hand side over all $\delta \in (t_k - t, t_{k+1} - t]$. To achieve this we find extremum of each $y_i(\delta) := B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \cdot e^{-\mathbf{E}_{\max}\cdot\delta}(\mathbf{E}_{\max}\cdot\delta)^i/i!$, $i = 0..N_{k+1}$, separately as functions of δ. Simple derivative analysis shows that extremum of these functions is achieved at $\delta = i/\mathbf{E}_{\max}$. If Algorithm 1 is used to obtain intervals $(t_k, t_{k+1}]$, then due to Lemma 4.3.10 for $k \geqslant 1 : N_{k+1} - N_k \leqslant 1$. Additionally, truncation with $(\lfloor \delta_{k-1} \cdot \mathbf{E}_{\max} \rfloor + 1)/\mathbf{E}_{\max}$ (step 4) ensures that for all $i = 0..N_{k+1}$ extremum of $y_i(\delta)$ on $(t_k - t, t_{k+1} - t]$ is attained at either $\delta_1 = t_k - t$ or $\delta_2 = t_{k+1} - t$. This allows us to use only δ_1 and δ_2 to over-approximate value $\mathrm{diff}^{\mathrm{opt}}_{\pi}(ps,\alpha,(t,t+\delta])$ for all $\delta \in (t_k - t, t_{k+1} - t]$. We thus obtain the following over-approximation:

$$\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta]) := \begin{cases} \max\limits_{0 \leqslant j \leqslant N_1} \widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta],j) & \text{if } k = 0 \\ \widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta],N_k) + u(N_k+1) & \text{if } k > 0 \end{cases}$$

where $N_m = N(t_m, \varepsilon_\Psi)$,

$$\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta],m) = \sum_{i=0}^{m} e^{-\mathbf{E}_{\max}\cdot\delta(ps,\alpha,i)} \frac{(\mathbf{E}_{\max}\cdot\delta(ps,\alpha,i))^i}{i!} \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) + C,$$

$u(m) = \Psi_{\mathbf{E}_{\max}\cdot\delta(ps,\alpha,m)}(m) \cdot B^{\mathrm{opt},m}_{\pi,\xi}(ps,\alpha)$ iff $B^{\mathrm{opt},m}_{\pi,\xi}(ps,\alpha) > 0$ and is 0 otherwise, and

$$\delta(ps,\alpha,i) = \begin{cases} t_k - t & \text{if } B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \geqslant 0 \text{ and } i/\mathbf{E}_{\max} \leqslant t_k - t \\ & \text{ or } B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \leqslant 0 \text{ and } i/\mathbf{E}_{\max} > t_k - t \\ t_{k+1} - t & \text{otherwise} \end{cases}$$

Lemma 4.3.13. *Let $(t_k, t_{k+1}]$ is an interval obtained by Algorithm 1 for $\varepsilon_h = \varepsilon_\Psi, \delta_h \in (0, 1/\mathbf{E}_{\max}]$. Then $\forall \delta \in (t_k - t, t_{k+1} - t], ps \in PS, \alpha \in Act(ps)$:*

$$\mathrm{diff}^{\mathrm{opt}}_{\pi}(ps,\alpha,(t,t+\delta]) \leqslant \widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta]) \tag{4.59}$$

Proof. Let $\delta \in (t_k - t, t_{k+1} - t]$. Consider

$$\sum_{i=0}^{j} e^{-\mathbf{E}_{\max}\cdot\delta} \frac{(\mathbf{E}_{\max}\cdot\delta)^i}{i!} \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) + C \tag{4.60}$$

and function $w(\tau,i) = \frac{(\mathbf{E}_{\max}\cdot\tau)^i}{e^{\mathbf{E}_{\max}\cdot\tau}}$ for $\tau \in [0,\infty)$. Its derivative

$$\frac{\mathrm{d}w(\tau,i)}{\mathrm{d}\tau} = \begin{cases} \frac{-\mathbf{E}_{\max}}{e^{\mathbf{E}_{\max}\cdot\tau}} & \text{if } i = 0 \\ \frac{{\mathbf{E}_{\max}}^i \cdot i\tau^{i-1}}{e^{\mathbf{E}_{\max}\cdot\tau}} - \frac{{\mathbf{E}_{\max}}^{i+1}\tau^i}{e^{\mathbf{E}_{\max}\cdot\tau}} = \frac{{\mathbf{E}_{\max}}^i\tau^{i-1}}{e^{\mathbf{E}_{\max}\cdot\tau}}(i - \mathbf{E}_{\max}\cdot\tau) & \text{if } i \geqslant 1 \end{cases}$$

Therefore

$$\frac{\mathrm{d}w(\tau,i)}{\mathrm{d}\tau} = 0 \text{ iff } (\tau = 0 \text{ and } i \geqslant 2) \text{ or } (\tau = i/\mathbf{E}_{\max} \text{ and } i \geqslant 1)$$

Consider $i \geqslant 1$:

$$\frac{\mathrm{d}w(\tau,i)}{\mathrm{d}t} = \frac{{\mathbf{E}_{\max}}^i \tau^{i-1}}{e^{\mathbf{E}_{\max}\cdot\tau}}(i - \mathbf{E}_{\max}\cdot\tau) < 0 \text{ iff } (i - \mathbf{E}_{\max}\cdot\tau) < 0 \text{ iff } \tau > i/\mathbf{E}_{\max}$$

And analogously $\frac{\mathrm{d}w(\tau,i)}{\mathrm{d}t} > 0$ iff $\tau < i/\mathbf{E}_{\max}$. Therefore the point $\tau = i/\mathbf{E}_{\max}$ is the point of maximum of $w(\tau,i)$.

For $\tau = 0, i \geqslant 2 : w(0,i) = 0$, and $\forall x > 0 : w(x,i) > 0$. Thus $\tau = 0$ is not a point of maximum of $w(\tau,i)$ on $[0,\infty)$.

And lastly for $i = 0$ the function $\frac{\mathrm{d}w(\tau,0)}{\mathrm{d}t}$ is negative and therefore $w(\tau,0)$ is decreasing. Hence its maximum is achieved at $\tau = 0 = 0/\mathbf{E}_{\max}$.

We can therefore conclude that $\forall i \in \mathbb{Z}_{\geqslant 0} : i/\mathbf{E}_{\max}$ is the only maximum of $w(\tau,i)$ on $[0,\infty)$. Consider the i^{th} summand of (4.60):

– If $B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \geqslant 0$ and $i/\mathbf{E}_{\max} \leqslant t_k - t$, then the function $\frac{(\mathbf{E}_{\max}\cdot\tau)^i}{e^{\mathbf{E}_{\max}\cdot\tau}} \cdot \frac{B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)}{i!}$ is decreasing on $(t_k - t, t_{k+1} - t]$ and therefore achieves its maximum at $\delta(ps,\alpha,i) = t_k - t$.

– If $B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \geqslant 0$ and $i/\mathbf{E}_{\max} > t_k - t$ then function $\frac{(\mathbf{E}_{\max}\cdot\tau)^i}{e^{\mathbf{E}_{\max}\cdot\tau}} \cdot \frac{B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)}{i!}$ is increasing in $(t_k - t, i/\mathbf{E}_{\max}]$. According to Lemma 4.3.9, $\exists j \leqslant k : (t_k - t, t_{k+1} - t] \subseteq (j/\mathbf{E}_{\max}, (j+1)/\mathbf{E}_{\max}]$. Therefore $i/\mathbf{E}_{\max} \geqslant t_{k+1} - t$ and the function $B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \geqslant 0$ and $i/\mathbf{E}_{\max} > t_k - t$ then the function the function $\frac{(\mathbf{E}_{\max}\cdot\tau)^i}{e^{\mathbf{E}_{\max}\cdot\tau}} \cdot \frac{B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)}{i!}$ is increasing on $(t_k - t, t_{k+1} - t]$. Thus its maximum is attained at $\delta(ps,\alpha,i) = t_{k+1} - t$.

– If $B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \leqslant 0$ and $i/\mathbf{E}_{\max} \leqslant t_k - t$, then the function $\frac{(\mathbf{E}_{\max}\cdot\tau)^i}{e^{\mathbf{E}_{\max}\cdot\tau}} \cdot \frac{B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)}{i!}$ is increasing on $(t_k - t, t_{k+1} - t]$ and therefore achieves its maximum at $\delta(ps,\alpha,i) = t_{k+1} - t$.

– If $B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \leqslant 0$ and $i/\mathbf{E}_{\max} > t_k - t$ then, analogously to the second case, the function $\frac{(\mathbf{E}_{\max}\cdot\tau)^i}{e^{\mathbf{E}_{\max}\cdot\tau}} \cdot \frac{B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)}{i!}$ is decreasing on $(t_k - t, i/\mathbf{E}_{\max}]$, $i/\mathbf{E}_{\max} \geqslant t_{k+1} - t$ and therefore the maximum is attained at $\delta(ps,\alpha,i) = t_k - t$.

We have thus shown that for $0 \leqslant i \leqslant j, \delta \in (t_k - t, t_{k+1} - t] : \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha) \leqslant \Psi_{\mathbf{E}_{\max}\cdot\delta(ps,\alpha,i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps,\alpha)$.

If $k = 0$, then $N_0 = 0$ and the proof follows from (4.58). Let $k > 0$. Then according to Lemma 4.3.10: $N_k \leqslant N(\delta,\varepsilon_\Psi) \leqslant N_k + 1$. Therefore the maximum in (4.58) ranges over $\{N_k, N_k + 1\}$. If $B^{\mathrm{opt},N_k+1}_{\pi,\xi}(ps,\alpha) > 0$ then the maximum is achieved at $N_k + 1$ $(u(N_k + 1) > 0)$ and otherwise at N_k $(u(N_k + 1) = 0)$. This finishes the proof. □

The lemma above shows that in order to check whether there may exist a switching point within $(t_k, t_{k+1}]$, one can compute the value $\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta])$ for all probabilistic states and all actions, which only depend on the boundary values t_k and t_{k+1}, and not on the δ itself. If this value is below 0, then there is no switching point, otherwise there may exist one.

Lemma 4.3.13 brings another interesting observation: for each interval $(t_k, t_{k+1}]$ there exists a subset of states, such that if the optimal strategy remains stationary for all of these states within $(t_k, t_{k+1}]$, then it remains stationary for *all* states within $(t_k, t_{k+1}]$. Thus reasoning on a subset of states provides sufficient information for all the states of the Markov automaton. This is the topic of the discussion in the following section.

Finding Maximal Transitions. In the following we call a pair $(s, \alpha) \in PS \times Act$ a *transition*. For transitions $(s, \alpha), (s', \alpha') \in PS \times Act$ we write $(s, \alpha) \preceq_k (s', \alpha')$ iff $\forall i = 0..k : B^{\mathrm{opt},i}_{\pi,\xi}(s, \alpha) \leqslant B^{\mathrm{opt},i}_{\pi,\xi}(s', \alpha')$. We say that a transition (s, α) is *maximal* if there exists no other transition (s', α') that satisfies the following: $(s, \alpha) \preceq_k (s', \alpha')$ and $\exists i = 0..k : B^{\mathrm{opt},i}_{\pi,\xi}(s, \alpha) < B^{\mathrm{opt},i}_{\pi,\xi}(s', \alpha')$. The set of all maximal transitions w. r. t. $\preceq_k$ is denoted with $\mathcal{T}_{\max}(k)$.

We prove that if inequality (4.59) holds for all transitions from $\mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$, then it holds for all transitions of the Markov automaton:

Lemma 4.3.14. *$\forall \delta \in (t_k - t, t_{k+1} - t], ps \in PS, \alpha \in Act(ps) : \exists (ps', \alpha') \in \mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$:*

$$\mathrm{diff}^{\mathrm{opt}}_{\pi}(ps, \alpha, (t, t + \delta]) \leqslant \widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps', \alpha', (t, t + \delta]) \qquad (4.61)$$

Proof. We prove that for each $ps \in PS, \alpha \in Act(ps)$ there exists a maximal transition (ps', α'), such that $\forall i = 0..N(t_{k+1} - t, \varepsilon_\Psi)$:

$$\Psi_{\mathbf{E}_{\max} \cdot \delta(ps', \alpha', i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha') \geqslant \Psi_{\mathbf{E}_{\max} \cdot \delta(ps, \alpha, i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha)$$

This implies the statement of the lemma.

If (ps, α) is maximal, then the statement holds for $(ps', \alpha') = (ps, \alpha)$. Consider $(ps, \alpha) \notin \mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$. Then there exists a transition $(ps', \alpha') \in \mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$ such that $(ps, \alpha) \preceq_{N(t_{k+1} - t, \varepsilon_\Psi)} (ps', \alpha')$ and by definition of $\preceq_{N(t_{k+1} - t, \varepsilon_\Psi)}$: $\forall i = 0..N(t_{k+1} - t, \varepsilon_\Psi) : B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha) \leqslant B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha')$. If for all $i \in 0..N(t_{k+1} - t, \varepsilon_\Psi) : B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha)$ and $B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha')$ are either both $\geqslant 0$ or $\leqslant 0$, then $\delta(ps, \alpha, i) = \delta(ps', \alpha', i)$ and therefore

$$\begin{aligned}\Psi_{\mathbf{E}_{\max} \cdot \delta(ps, \alpha, i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha) &\leqslant \Psi_{\mathbf{E}_{\max} \cdot \delta(ps, \alpha, i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha') \\ &= \Psi_{\mathbf{E}_{\max} \cdot \delta(ps', \alpha', i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha')\end{aligned}$$

If for some $i \in 0..N(t_{k+1} - t, \varepsilon_\Psi)$ the signs of $B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha)$ and $B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha')$ are different, the only possibility is that $B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha) < 0$ and $B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha') \geqslant 0$. Then

$$\begin{aligned}\Psi_{\mathbf{E}_{\max} \cdot \delta(ps, \alpha, i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha) &= -\Psi_{\mathbf{E}_{\max} \cdot \delta(ps, \alpha, i)}(i) \cdot \left| B^{\mathrm{opt},i}_{\pi,\xi}(ps, \alpha) \right| \\ &\leqslant 0 \leqslant \Psi_{\mathbf{E}_{\max} \cdot \delta(ps', \alpha', i)}(i) \cdot B^{\mathrm{opt},i}_{\pi,\xi}(ps', \alpha')\end{aligned}$$

□

Thus only transitions from $\mathcal{T}_{\max}(N(t_{k+1}-t,\varepsilon_\Psi))$ have to be checked in order to establish existence of switching points. In the worst case the set $\mathcal{T}_{\max}(N(t_{k+1}-t,\varepsilon_\Psi))$ can be almost as large as $PS \times Act$. The complexity of the computation of $\mathcal{T}_{\max}(N(t_{k+1}-t,\varepsilon_\Psi))$ is in $O(|PS \times Act|^2 \cdot N(t_{k+1}-t,\varepsilon_\Psi))$.

Switching Points of ϵ-optimal Strategy. So far our approach to detect switching points was based on testing $\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta])$ against 0. There are a few issues related to this. First of all, when performing computations with floating points, the results may be imprecise: Values that are 0 in theory might be represented by non-zero float-point numbers. Thus it may look like the value of $\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta])$ is above 0, while in reality it is below 0 but the final result is affected by floating point errors.

Secondly, it may happen that the optimal strategy switches very often on a time interval, while the effect of these frequent switches is negligible. The difference may be so small that an ϵ-optimal strategy actually stays stationary on this interval. Testing $\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta])$ against 0 does not take this into account.

And lastly, we have not discuss yet what to do in case our sufficient conditions fail already for intervals of length smaller than δ_h. Our approach so far forbids to perform steps smaller then δ_h, however there may exist switching points within such intervals.

To counteract these issues we introduce two methods to estimate error introduced by a potentially sub-optimal policy, i. e. policy for which $\widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta])$ is above 0.

Let $\mathcal{M}$ be PS-acyclic, $\xi = (\varepsilon_\Psi,\varepsilon_{\mathrm{rea}},\varepsilon_{\mathrm{n}})$ satisfies (4.26), $t \in [0,b), (t_k,t_{k+1}]$ is an interval outputted by `ComputeIntervals` for $b, t, \varepsilon_h = \varepsilon_\Psi, \delta_h \in (0, 1/\mathbf{E}_{\max}]$. Let $\delta \in (t_k - t, t_{k+1} - t]$, $A \subseteq PS \times Act$, $c_2 = (\mathbf{E}_{\max} \cdot \delta)^2/2$. If $A \neq \emptyset$ we define $\varepsilon_{\max} = \max_{\tau=(ps,\alpha)\in A}\{0, \widetilde{\mathrm{diff}}^{\mathrm{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta])\}$, $c_1 = d_{\max} \cdot \varepsilon_{\max}$, and if $c_1 > 0$, then $n = \left\lceil \sqrt{\frac{c_2}{c_1}} \right\rceil$. We define

$$\mathrm{err}^1_{\pi,\xi}((t,t+\delta],A) := \begin{cases} 0 & \text{if } c_1 = 0 \\ c_1 \cdot n + c_2/n & \text{otherwise} \end{cases} \tag{4.62}$$

Lemma 4.3.15. *Let $\mathcal{M}$ be a PS-acyclic MA, $A = PS \times Act$ or $A = \mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$, $\xi = (\varepsilon_\Psi, \varepsilon_{rea}, \varepsilon_n)$ satisfies (4.26), $\varepsilon_n \leqslant \varepsilon_{rea}/\left(N(\delta, \varepsilon_\Psi) + 1\right)$, and Assumption 4.3.1 is satisfied.*
Let $\pi \in \Pi_{\mathtt{PC}}$, such that π is stationary on $(t, t+\delta]$. Let $v : S \to [0,1]$ be such that $\forall s \in S : v(s) \preccurlyeq_{\text{opt}} \text{val}_\pi(s,t,g)$ and

$$v(s) \preccurlyeq_{\text{opt}} \text{val}_{\text{opt}}(s,t,g) \preccurlyeq_{\text{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon \tag{4.63}$$

Then π is at least $(\varepsilon + \text{err}^1_{\pi,\xi}((t,t+\delta], A))$-optimal for $\text{val}_{\text{opt}}(s, t+\delta, g)$ and $\forall s \in S$:

$$\begin{aligned} u^{t+\delta,v}_{\pi,\xi}(s) \leqslant \text{val}_{\sup}(s,t+\delta,g) \leqslant u^{t+\delta,v}_{\pi,\xi}(s) + \varepsilon + \text{err}^1_{\pi,\xi}((t,t+\delta],A) + \varepsilon_\Psi + \varepsilon_{rea} \\ u^{t+\delta,v}_{\pi,\xi}(s) + \varepsilon_\Psi + \varepsilon_{rea} \geqslant \text{val}_{\inf}(s,t+\delta,g) \geqslant u^{t+\delta,v}_{\pi,\xi}(s) - \varepsilon - \text{err}^1_{\pi,\xi}((t,t+\delta],A) \end{aligned} \tag{4.64}$$

Proof. First of all, it holds that

$$v(s) \preccurlyeq_{\text{opt}} \text{val}_\pi(s,t,g) \preccurlyeq_{\text{opt}} \text{val}_{\text{opt}}(s,t,g) \preccurlyeq_{\text{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon$$

Let $\varepsilon = \varepsilon_1 + \varepsilon_2$, where $\varepsilon_1, \varepsilon_2 \in [0,1]$ and

$$\text{val}_\pi(s,t,g) - v(s) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon_1 \tag{4.65}$$
$$\text{val}_{\text{opt}}(s,t,g) - \text{val}_\pi(s,t,g) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})} \cdot \varepsilon_2 \tag{4.66}$$

Due to Lemma 4.3.2, $\forall t' \in (t, t+\delta], s \in S$:

$$\begin{aligned} \text{opt} = \sup : u^{t',v}_{\pi,\xi}(s) \leqslant \text{val}_\pi(s,t',g) \leqslant u^{t',v}_{\pi,\xi}(s) + \varepsilon_1 + \varepsilon_{\text{rea}} + \varepsilon_\Psi \\ \text{opt} = \inf : u^{t',v}_{\pi,\xi}(s) + \varepsilon_{\text{rea}} + \varepsilon_\Psi \geqslant \text{val}_\pi(s,t',g) \geqslant u^{t',v}_{\pi,\xi}(s) - \varepsilon_1 \end{aligned}$$

Consider the case of $c_1 = 0$. This is possible if $d_{\max} = 0$ or $\varepsilon_{\max} = 0$. The former can only happen if $PS = \emptyset$. In this case any strategy is optimal, i. e.:

$$\forall \pi' \in \Pi_{\mathtt{PC}}, t' \in (t,t+\delta], s \in S : \text{val}_{\pi'}(s,t',g) = \text{val}_{\text{opt}}(s,t',g)$$

Thus (4.64) follows from Lemma 4.3.2.

If $\varepsilon_{\max} = 0$, then

$$\forall ps \in PS, \alpha \in Act(ps) : \widetilde{\text{diff}}^{\text{opt}}_{\pi,\xi}(ps, \alpha, (t,t+\delta]) \leqslant 0$$

Due to Lemma 4.3.13 this implies that $\forall ps \in PS, \alpha \in Act$:

$$\text{diff}^{\text{opt}}_\pi(ps,\alpha,(t,t+\delta]) \leqslant \widetilde{\text{diff}}^{\text{opt}}_{\pi,\xi}(ps,\alpha,(t,t+\delta]) \leqslant 0$$

Thus the sufficient condition 4.3.8 holds and therefore π is optimal on $(t, t+\delta]$, i. e.:

$$\forall t' \in (t,t+\delta], s \in S : \text{val}_\pi(s,t',g) = \text{val}_{\text{opt}}(s,t',g)$$

We conclude that in this case (4.64) holds due to 4.3.2.

Consider now the case $c_1 > 0$. At first we will show that π is $(\varepsilon_2 + \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A))$-optimal on $(t, t+\delta]$, i.e. that $\forall s \in S, t' \in (t, t+\delta]$:

$$0 \preccurlyeq_{\mathrm{opt}} \mathrm{val}_{\mathrm{opt}}(s, t', g) - \mathrm{val}_\pi(s, t', g) \preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \left(\varepsilon_2 + \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A)\right) \tag{4.67}$$

The leftmost inequality follows directly from the optimality of $\mathrm{val}_{\mathrm{opt}}(s, t', g)$. In order to prove the rightmost inequality, we will first introduce a new variable. Let $n \in \mathbb{Z}_{>0}, \delta_{st} = (t' - t)/n$. Let $\tau_i = t + i \cdot \delta_{st}$ for $i = 0..n$. Consider $\widetilde{\mathrm{val}}_{\mathrm{opt}}(s, x, g)$ defined as follows:

$$\widetilde{\mathrm{val}}_{\mathrm{opt}}(s, x, g) = \begin{cases} \mathrm{val}_{\mathrm{opt}}(s, t, g) & \text{if } x = \tau_0 \\ e^{-E(s)\cdot\delta_{st}} \cdot \widetilde{\mathrm{val}}_{\mathrm{opt}}(s, \tau_{i-1}, g) + & \text{if } x = \tau_i, i > 0, s \in MS \\ \quad (1 - e^{-E(s)\cdot\delta_{st}}) \sum\limits_{s' \in S} \frac{\mathbf{R}[s,s']}{E(s)} \cdot \widetilde{\mathrm{val}}_{\mathrm{opt}}(s', \tau_{i-1}, g) & \\ \mathrm{rea}^{\mathrm{opt}}(s, \widetilde{\mathrm{val}}_{\mathrm{opt}}(\tau_i, g)|_{MS}) & \text{if } x = \tau_i, i > 0, s \in PS \\ \widetilde{\mathrm{val}}_{\mathrm{opt}}(s, \tau_i, g) & \text{if } x \in (\tau_{i-1}, \tau_i), n \geqslant i > 0 \end{cases}$$

We will prove the rightmost inequality of (4.67) by splitting the value into two:

$$\begin{aligned} &\mathrm{val}_{\mathrm{opt}}(s, t', g) - \mathrm{val}_\pi(s, t', g) \\ &\quad = \mathrm{val}_{\mathrm{opt}}(s, t', g) - \widetilde{\mathrm{val}}_{\mathrm{opt}}(s, t', g) + \widetilde{\mathrm{val}}_{\mathrm{opt}}(s, t', g) - \mathrm{val}_\pi(s, t', g) \end{aligned}$$

Lemma A.6 proves that $\forall s \in S, i = 0..n$:

$$0 \leqslant \mathrm{val}_{\mathrm{opt}}(s, \tau_i, g) - \widetilde{\mathrm{val}}_{\mathrm{opt}}(s, \tau_i, g) \leqslant i \cdot \frac{(\mathbf{E}_{\max} \cdot \delta_{st})^2}{2} \tag{4.68}$$

We will show by induction that $\forall s \in S, i = 0..n$:

$$\widetilde{\mathrm{val}}_{\sup}(s, \tau_i, g) - \mathrm{val}_\pi(s, \tau_i, g) \leqslant \varepsilon_2 + i \cdot d_{\max} \cdot \varepsilon_{\max} \tag{4.69}$$

$$\widetilde{\mathrm{val}}_{\inf}(s, \tau_i, g) - \mathrm{val}_\pi(s, \tau_i, g) \geqslant -\left(\varepsilon_2 + i \cdot \left(d_{\max} \cdot \varepsilon_{\max} + \frac{(\mathbf{E}_{\max} \cdot \delta_{st})^2}{2}\right)\right) \tag{4.70}$$

For $i = 0, s \in S$:

$$\begin{aligned} \widetilde{\mathrm{val}}_{\mathrm{opt}}(s, \tau_0, g) - \mathrm{val}_\pi(s, \tau_0, g) &= \mathrm{val}_{\mathrm{opt}}(s, \tau_0, g) - \mathrm{val}_\pi(s, \tau_0, g) \\ &\overset{(4.66)}{\preccurlyeq_{\mathrm{opt}}} (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon_2 \end{aligned}$$

Consider $0 < i \leqslant n$ and $ms \in MS$. According to Lemma 4.3.4, the value $\mathrm{val}_\pi(ms, \tau_i, g)$ can be bounded as follows:

$$Y(ms, \delta_{st}) \leqslant \mathrm{val}_\pi(ms, \tau_i, g) \leqslant Y(ms, \delta_{st}) + \frac{(\mathbf{E}_{\max} \cdot \delta_{st})^2}{2},$$

where

$$\begin{aligned} Y(ms, \delta_{st}) &:= e^{-E(ms)\cdot\delta_{st}} \cdot \mathrm{val}_\pi(ms, \tau_{i-1}, g) \\ &\quad + (1 - e^{-E(ms)\cdot\delta_{st}}) \sum_{s' \in S} \frac{\mathbf{R}[ms, s']}{E(ms)} \cdot \mathrm{val}_\pi(s', \tau_{i-1}, g) \end{aligned}$$

Thus

$$\widetilde{\mathrm{val}}_{\sup}(ms,\tau_i,g) - \mathrm{val}_\pi(ms,\tau_i,g) \leqslant \varepsilon_2 + (i-1)\cdot d_{\max}\cdot\varepsilon_{\max} \tag{4.71}$$

$$\widetilde{\mathrm{val}}_{\inf}(ms,\tau_i,g) - \mathrm{val}_\pi(ms,\tau_i,g) \geqslant -\left(\varepsilon_2 + (i-1)\cdot d_{\max}\cdot\varepsilon_{\max} + i\cdot\frac{(\mathbf{E}_{\max}\cdot\delta_{st})^2}{2}\right) \tag{4.72}$$

Consider a probabilistic state $ps \in PS$. Let

$$\pi_i = \arg\operatorname*{opt}_{\pi'\in\Pi_{\mathrm{stat}}} \mathrm{rea}^{\pi'}(ps, \widetilde{\mathrm{val}}_{\mathrm{opt}}(\tau_i,g)|_{MS})$$

Then

$$\begin{aligned}
&\widetilde{\mathrm{val}}_{\mathrm{opt}}(ps,\tau_i,g) - \mathrm{val}_\pi(ps,\tau_i,g)\\
&\quad= \sum_{ms\in MS}\Big(\mathrm{rea}^{\pi_i}(ps,ms)\cdot\widetilde{\mathrm{val}}_{\mathrm{opt}}(ms,\tau_i,g) - \mathrm{rea}^{\pi}(ps,ms)\cdot\mathrm{val}_\pi(ms,\tau_i,g)\Big)\\
&\quad= \underbrace{\sum_{ms\in MS}\mathrm{val}_\pi(ms,\tau_i,g)\cdot(\mathrm{rea}^{\pi_i}(ps,ms) - \mathrm{rea}^{\pi}(ps,ms))}_{\preccurlyeq_{\mathrm{opt}}(-1)^{\mathbb{1}\{\inf\}(\mathrm{opt})}d_{\max}\cdot\varepsilon_{\max}(\text{ Lemma A.7})}\\
&\qquad+ \underbrace{\sum_{ms\in MS}\mathrm{rea}^{\pi_i}(ps,ms)\cdot\Big(\widetilde{\mathrm{val}}_{\mathrm{opt}}(ms,\tau_i,g) - \mathrm{val}_\pi(ms,\tau_i,g)\Big)}_{\text{bounded by (4.71-4.72)}}
\end{aligned}$$

For opt = sup:

$$\begin{aligned}
\widetilde{\mathrm{val}}_{\sup}(ps,\tau_i,g) - \mathrm{val}_\pi(ps,\tau_i,g) &\leqslant d_{\max}\cdot\varepsilon_{\max} + \varepsilon_2 + (i-1)\cdot d_{\max}\cdot\varepsilon_{\max}\\
&= \varepsilon_2 + i\cdot d_{\max}\cdot\varepsilon_{\max}
\end{aligned}$$

For opt = inf:

$$\begin{aligned}
&\widetilde{\mathrm{val}}_{\inf}(ps,\tau_i,g) - \mathrm{val}_\pi(ps,\tau_i,g)\\
&\qquad\geqslant -\left(\varepsilon_2 + d_{\max}\cdot\varepsilon_{\max} + (i-1)\cdot d_{\max}\cdot\varepsilon_{\max} + i\cdot\frac{(\mathbf{E}_{\max}\cdot\delta_{st})^2}{2}\right)\\
&\qquad= -\left(\varepsilon_2 + i\cdot\left(d_{\max}\cdot\varepsilon_{\max} + \frac{(\mathbf{E}_{\max}\cdot\delta_{st})^2}{2}\right)\right)
\end{aligned}$$

This concludes the proof of (4.69-4.70). Consider $\tau_n = t', s \in S$:

$$\begin{aligned}
\mathrm{val}_{\sup}(s,t',g) - \widetilde{\mathrm{val}}_{\sup}(s,t',g) &\overset{(4.68)}{\leqslant} \frac{(\mathbf{E}_{\max}\cdot(t'-t))^2}{2\cdot n} \leqslant \frac{(\mathbf{E}_{\max}\cdot\delta)^2}{2\cdot n}\\
\widetilde{\mathrm{val}}_{\sup}(s,t',g) - \mathrm{val}_\pi(s,t',g) &\overset{(4.69)}{\leqslant} \varepsilon_2 + n\cdot d_{\max}\cdot\varepsilon_{\max}\\
\mathrm{val}_{\inf}(s,t',g) - \widetilde{\mathrm{val}}_{\inf}(s,t',g) &\overset{(4.68)}{\geqslant} 0\\
\widetilde{\mathrm{val}}_{\inf}(s,t',g) - \mathrm{val}_\pi(s,t',g) &\overset{(4.70)}{\geqslant} -\left(\varepsilon_2 + n\cdot\left(d_{\max}\cdot\varepsilon_{\max} + \frac{(\mathbf{E}_{\max}\cdot\delta_{st})^2}{2}\right)\right)
\end{aligned}$$

By summing up the inequalities above we can prove (4.67) for an arbitrary n. In order to obtain the best error bound it is better to choose such n that minimises the expression. Derivative analysis w. r. t. n shows that the minimum is attained at $n = \sqrt{\frac{c_2}{c_1}}$. For this reason we choose $n = \left\lceil \sqrt{\frac{c_2}{c_1}} \right\rceil$.

We have just shown that π is $(\varepsilon_2 + \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A))$-optimal on $(t, t+\delta]$. Thus for all $t' \in (t, t+\delta], s \in S$:

$$\begin{aligned}
u^{t',v}_{\pi,\xi}(s) &\overset{\text{Lem. 4.3.2}}{\leqslant} \mathrm{val}_\pi(s, t', g)\\
&\overset{(4.67)}{\leqslant} \mathrm{val}_{\sup}(s, t', g)\\
&\overset{(4.67)}{\leqslant} \mathrm{val}_\pi(s, t', g) + \varepsilon_2 + \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A)\\
&\overset{\text{Lem. 4.3.2}}{\leqslant} u^{t',v}_{\pi,\xi}(s) + \varepsilon_2 + \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A) + \varepsilon_1 + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}}\\
&= u^{t',v}_{\pi,\xi}(s) + \varepsilon + \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A) + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}}
\end{aligned}$$

And analogously for infinum:

$$\begin{aligned}
u^{t',v}_{\pi,\xi}(s) + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}} &\overset{\text{Lem. 4.3.2}}{\geqslant} \mathrm{val}_\pi(s, t', g)\\
&\overset{(4.67)}{\geqslant} \mathrm{val}_{\inf}(s, t', g)\\
&\overset{(4.67)}{\geqslant} \mathrm{val}_\pi(s, t', g) - \varepsilon_2 - \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A)\\
&\overset{\text{Lem. 4.3.2}}{\geqslant} u^{t',v}_{\pi,\xi}(s) - \varepsilon_1 - \varepsilon_2 - \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A)\\
&= u^{t',v}_{\pi,\xi}(s) - \varepsilon - \mathrm{err}^1_{\pi,\xi}((t, t+\delta], A)
\end{aligned}$$

This concludes the proof.

□

The error bound obtained above is likely over-pessimistic. As long as switches of an ϵ-optimal strategy bring significant improvement for the reachability value even a rough estimate like this suffices. However if switching from one stationary strategy to another changes slightly the reachability probability, Lemma 4.3.15 may fail to detect that ϵ-optimal strategy may remain stationary over an interval. Our experimental evaluation shows that among the published case studies there are many models and properties that have this feature. In the following we will present another test on the presence of switching points of an ϵ-optimal strategy that performs well in this case. It is originally developed in [BS11] for late CTMDPs and implemented in `IMCA` toolset [GHKN12]. It was later partially extended to Markov automata in [Gro18].

Let h be a total goal function, $k \in \mathbb{Z}_{\geqslant 0}, s \in S$. We define

$$\overline{\mathbf{D}}^k(s, h) := \begin{cases} h(s) & \text{if } s \in MS, k = 0\\ \sum\limits_{s' \in S} \frac{\mathbf{R}[s,s']}{\mathbf{E}_{\max}} \cdot \overline{\mathbf{D}}^{k-1}(s', h) + (1 - \frac{E(s)}{\mathbf{E}_{\max}}) \cdot \overline{\mathbf{D}}^{k-1}(s, h) & \text{if } s \in MS, k > 0\\ \mathrm{rea}^{\mathrm{opt}}(s, \overline{\mathbf{D}}^k(h)|_{MS}) & \text{if } s \in PS \end{cases}$$

If the equations above use an approximation $\mathrm{rea}_{\varepsilon}^{\mathrm{opt}}(\cdot,\cdot)$ instead of $\mathrm{rea}^{\mathrm{opt}}(\cdot,\cdot)$, for some $\varepsilon \in [0,1)$, then we denote thus obtained values with $\overline{\mathbf{D}}_{\varepsilon}^{k}(s,h)$.

Let $\varepsilon \in (0,1), \pi \in \Pi_{\mathtt{PC}}, I \in \mathcal{I}(\pi),\ t \in I, \delta = t - \inf I,\ s \in S,\ \xi = (\varepsilon_{\Psi}, \varepsilon_{\mathrm{rea}}, \varepsilon_{\mathrm{n}})$ satisfies (4.26) and $v : S \to [0,1]$. We define

$$\overline{u}_{\mathrm{opt},\xi}^{t,v}(s) := \begin{cases} v & \text{if } t = \inf I \\ \sum\limits_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \overline{\mathbf{D}}_{\varepsilon_{\mathrm{n}}}^{i}(s,v) & \text{else if } s \in \mathit{MS} \\ \mathrm{rea}_{\varepsilon_{\mathrm{n}}}^{\mathrm{opt}}(s, \overline{u}_{\mathrm{opt},\xi}^{t,v}|_{\mathit{MS}}) & \text{else if } s \in \mathit{PS} \end{cases}$$

And

$$\begin{aligned} \mathrm{err}_{\pi,\xi}^{2}(I,\delta) &:= \sum_{i=0}^{N(\delta,\varepsilon_{\Psi})} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot D_{\mathrm{opt},\xi}^{i} + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot 2 \cdot \varepsilon_{\mathrm{n}} \\ D_{\mathrm{opt},\xi}^{i} &:= \operatorname*{opt}_{s\in S} \left(\overline{\mathbf{D}}_{\varepsilon_{\mathrm{n}}}^{i}(s,v) - \mathbf{D}_{\varepsilon_{\mathrm{n}}}^{i}(s,\pi|_{I},v) \right) \end{aligned}$$

Lemma 4.3.16. *Let $\mathcal{M}$ be an MA, $\pi \in \Pi_{\mathtt{PC}}$, such that π is stationary on $I \in \mathcal{I}(\pi)$ and $(t, t+\delta] \subseteq I, \delta > 0,\ \xi = (\varepsilon_{\Psi}, \varepsilon_{rea}, \varepsilon_{n})$ satisfies (4.26) and $\varepsilon_{n} \leqslant \varepsilon_{rea}/\left(N(\delta,\varepsilon_{\Psi})+1\right)$.*
Let $v : S \to [0,1]$ be such that $\forall s \in S : v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_{\pi}(s,t,g)$ and

$$v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_{\mathrm{opt}}(s,t,g) \preccurlyeq_{\mathrm{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon$$

Then π is at least $(\varepsilon + |\mathrm{err}_{\pi,\xi}^{2}(I,\delta)| + \varepsilon_{\Psi} + \varepsilon_{rea})$-optimal for $\mathrm{val}_{\mathrm{opt}}(s, t+\delta, g)$ and $\forall s \in S$:

$$\begin{aligned} u_{\pi,\xi}^{t+\delta,v}(s) \leqslant \mathrm{val}_{\sup}(s,t+\delta,g) \leqslant u_{\pi,\xi}^{t+\delta,v}(s) + \varepsilon + \mathrm{err}_{\pi,\xi}^{2}(I,\delta) + \varepsilon_{\Psi} + \varepsilon_{rea} \\ u_{\pi,\xi}^{t+\delta,v}(s) + \varepsilon_{\Psi} + \varepsilon_{rea} \geqslant \mathrm{val}_{\inf}(s,t+\delta,g) \geqslant u_{\pi,\xi}^{t+\delta,v}(s) - \varepsilon + \mathrm{err}_{\pi,\xi}^{2}(I,\delta) \end{aligned} \tag{4.73}$$

Proof. First of all, it holds that

$$v(s) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_{\pi}(s,t,g) \preccurlyeq_{\mathrm{opt}} \mathrm{val}_{\mathrm{opt}}(s,t,g) \preccurlyeq_{\mathrm{opt}} v(s) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon$$

Due to Lemma 4.3.2, $\forall t' \in (t, t+\delta], s \in S$:

$$\begin{aligned} \mathrm{opt} = \sup : u_{\pi,\xi}^{t',v}(s) \leqslant \mathrm{val}_{\pi}(s,t',g) \leqslant u_{\pi,\xi}^{t',v}(s) + \varepsilon + \varepsilon_{\mathrm{rea}} + \varepsilon_{\Psi} \\ \mathrm{opt} = \inf : u_{\pi,\xi}^{t',v}(s) + \varepsilon_{\mathrm{rea}} + \varepsilon_{\Psi} \geqslant \mathrm{val}_{\pi}(s,t',g) \geqslant u_{\pi,\xi}^{t',v}(s) - \varepsilon \end{aligned}$$

Uniformisation of a Markov automaton is the same as uniformisation of CTMCs [Jen53]: for each Markovian state ms with exit rate strictly below $\mathbf{E}_{\max}$ a new self-loop transition is added with rate $\mathbf{E}_{\max} - E(ms)$. We will denote the uniformised version of $\mathcal{M}$ with $\mathcal{M}_{u}$. Notice that uniformising a Markov automaton does not affect the value $\mathrm{val}_{\mathrm{opt}}(x,g)$, i. e. $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(x,g) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_{u}}(x,g)$. Therefore for $s \in S$ the following holds:

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, t+\delta, g)$$

$$
\begin{aligned}
&= \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_u}(s, t+\delta, g)\\
&\overset{\text{Lemma 4.1.5}}{=} \underset{\pi' \in \Pi_{\mathtt{TM}}}{\mathrm{opt}} \sum_{s' \in dom(g)} \mathrm{Pr}_{\pi',s}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=t+\delta} ap_{s'}\right] \cdot g(s')\\
&= \underset{\pi' \in \Pi_{\mathtt{TM}}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi',s}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\right] \sum_{s' \in dom(g)} \mathrm{Pr}_{\pi'-\delta,s''}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=t} ap_{s'}\right] \cdot g(s')\\
&= \underset{\pi' \in \Pi_{\mathtt{TM}}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi',s}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\right] \cdot \mathrm{val}_{\pi'-\delta}^{\mathcal{M}_u}(s'', t, g)
\end{aligned}
$$

The event $\{\rho \in \mathit{Paths}^\omega \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\}$ can be partitioned according to the number of Markovian transitions performed before $\mathtt{tt}\,\mathrm{U}^{=\delta} ap_{s''}$ is satisfied. Let $\#_{\delta,i}$ be the set of infinite paths that perform i Markovian transitions until time δ. Then

$$
\begin{aligned}
\{\rho \in \mathit{Paths}^\omega \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\} &= \biguplus_{i=0}^{\infty} \{\rho \in \mathit{Paths}^\omega \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=\delta} ap_{s''}\} \cap \#_{\delta,i}\\
&= \biguplus_{i=0}^{\infty} \{\rho \in \mathit{Paths}^\omega \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\} \cap \#_{\delta,i},
\end{aligned}
$$

where $\{\rho \in \mathit{Paths}^\omega \mid \rho \models \mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\}$ is the event of reaching s'' by performing i Markovian transitions. Thus

$$
\begin{aligned}
\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, x, g) &= \underset{\pi' \in \Pi_{\mathtt{TM}}}{\mathrm{opt}} \sum_{s'' \in MS} \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathrm{Pr}_{\pi',s}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\right] \cdot \mathrm{val}_{\pi'-\delta}^{\mathcal{M}_u}(s'', t, g)\\
&\preccurlyeq_{\mathrm{opt}} \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \underset{\pi' \in \Pi_{\mu}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi',s}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\right] \cdot \mathrm{val}_{\pi'-\delta}^{\mathcal{M}_u}(s'', t, g)
\end{aligned}
\tag{4.74}
$$

Next we will show by induction that for $h : MS \to [0,1]$, $\forall s \in S, i \in \mathbb{Z}_{\geqslant 0}$:

$$
\overline{\mathbf{D}}^i(s, h) = \underset{\pi' \in \Pi_{\mu}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi',s}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\right] \cdot h(s'')
$$

For $i = 0, ms \in MS$:

$$
\overline{\mathbf{D}}^0(ms, h) = h(ms) = \underset{\pi' \in \Pi_{\mu}}{\mathrm{opt}}\ h(ms) = \underset{\pi' \in \Pi_{\mathtt{PC}}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi',ms}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=0} ap_{s''}\right] \cdot h(s'')
$$

For $ps \in PS, i = 0$:

$$
\begin{aligned}
\overline{\mathbf{D}}^i(ps, h) &= \mathrm{rea}^{\mathrm{opt}}(ps, \overline{\mathbf{D}}^i(h)|_{MS})\\
&= \underset{\pi' \in \Pi_{\mu}}{\mathrm{opt}} \sum_{ms \in MS} \mathrm{rea}^{\pi'}(ps, ms) \cdot \overline{\mathbf{D}}^i(ms, h)\\
&= \underset{\pi' \in \Pi_{\mu}}{\mathrm{opt}} \sum_{ms \in MS} \mathrm{rea}^{\pi'}(ps, ms) \cdot \underset{\pi'' \in \Pi_{\mu}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi'',ms}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\right] \cdot h(s'')\\
&= \underset{\pi' \in \Pi_{\mu}}{\mathrm{opt}} \sum_{s'' \in MS} \mathrm{Pr}_{\pi',ps}^{\mathcal{M}_u}\left[\mathtt{tt}\,\mathrm{U}^{=i} ap_{s''}\right] \cdot h(s'')
\end{aligned}
$$

For $ps \in PS, i > 0$ the proof is identical. Consider $ms \in MS, i > 0$:

$$\begin{aligned}\overline{\mathbf{D}}^i(ms, h) &= \sum_{s' \in S} \frac{\mathbf{R}[ms, s']}{\mathbf{E}_{\max}} \cdot \overline{\mathbf{D}}^{i-1}(s', h) + (1 - \frac{E(ms)}{\mathbf{E}_{\max}}) \cdot \overline{\mathbf{D}}^{i-1}(ms, h) \\ &\overset{IH}{=} \operatorname*{opt}_{\pi' \in \Pi_\mu} \sum_{s' \in S} \Pr^{\mathcal{M}_u}_{\pi', ms}\left[\texttt{tt}\, \mathrm{U}^{=i} ap_{s'}\right] \cdot h(s')\end{aligned}$$

Therefore we can rewrite (4.74) as follows:

$$\mathrm{val}^{\mathcal{M}}_{\mathrm{opt}}(s, t+\delta, g) \preccurlyeq_{\mathrm{opt}} \begin{cases} \sum_{i=0}^{\infty} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \overline{\mathbf{D}}^i(s, \mathrm{val}^{\mathcal{M}_u}_{\mathrm{opt}}(t, g)) & \text{if } s \in MS \\ \mathrm{rea}^{\mathrm{opt}}(\mathrm{val}^{\mathcal{M}_u}_{\mathrm{opt}}(t+\delta, g)|_{MS}) & \text{if } s \in PS \end{cases}$$

We will denote the right-hand side of the inequality above with $\overline{\mathrm{val}}_{\mathrm{opt}}(s, t+\delta, g)$. The value $\overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(s)$ approximates $\overline{\mathrm{val}}_{\mathrm{opt}}(s, t+\delta, g)$ on interval $(t, t+\delta]$ just like $u^{t+\delta,v}_{\pi,\xi}(s)$ approximates $\mathrm{val}_\pi(t+\delta, g)$. Therefore the following holds:

$$\begin{aligned}\mathrm{val}_{\sup}(s, t+\delta, g) \leqslant \overline{\mathrm{val}}_{\sup}(s, t+\delta, g) &\leqslant \overline{u}^{t+\delta,v}_{\sup,\xi}(s) + \varepsilon + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}} \\ \mathrm{val}_{\inf}(s, t+\delta, g) \geqslant \overline{\mathrm{val}}_{\inf}(s, t+\delta, g) &\geqslant \overline{u}^{t+\delta,v}_{\inf,\xi}(s) - \varepsilon\end{aligned}$$

And

$$\begin{aligned}u^{t+\delta,v}_{\pi,\xi}(s) \leqslant \mathrm{val}_\pi(s, t+\delta, g) &\leqslant \mathrm{val}_{\sup}(s, t+\delta, g) \\ &\leqslant \overline{\mathrm{val}}_{\sup}(s, t+\delta, g) \leqslant \overline{u}^{t+\delta,v}_{\sup,\xi}(s) + \varepsilon + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}} \\ u^{t+\delta,v}_{\pi,\xi}(s) + \varepsilon_\Psi + \varepsilon_{\mathrm{rea}} \geqslant \mathrm{val}_\pi(s, t+\delta, g) &\geqslant \mathrm{val}_{\inf}(s, t+\delta, g) \\ &\geqslant \overline{\mathrm{val}}_{\inf}(s, t+\delta, g) \geqslant \overline{u}^{t+\delta,v}_{\inf,\xi}(s) - \varepsilon\end{aligned}$$

Next we will bound the difference $\overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(ms) - u^{t+\delta,v}_{\pi,\xi}(ms)$ from above for $\mathrm{opt} = \sup$ and from below for $\mathrm{opt} = \inf$. We will show that

$$\forall s \in S : \overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(s) - u^{t+\delta,v}_{\pi,\xi}(s) \preccurlyeq_{\mathrm{opt}} \mathrm{err}^2_{\pi,\xi}(I, \delta) \tag{4.75}$$

Consider $ms \in MS$:

$$\begin{aligned}&\overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(ms) - u^{t+\delta,v}_{\pi,\xi}(ms) \\ &= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\overline{\mathbf{D}}^i_{\varepsilon_{\mathrm{n}}}(ms, v) - \mathbf{D}^i_{\varepsilon_{\mathrm{n}}}(ms, \pi|_I, v)\right) \\ &\preccurlyeq_{\mathrm{opt}} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot D^i_{\mathrm{opt},\xi} \preccurlyeq_{\mathrm{opt}} \mathrm{err}^2_{\pi,\xi}(I, \delta)\end{aligned}$$

Consider $ps \in PS$. From the proof of Lemma 4.3.12 (see (4.56, 4.57)) we obtain the following:

$$\sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_{\mathrm{n}}}(ps, \pi|_I, v) - \varepsilon_{\mathrm{n}} \leqslant u^{t+\delta,v}_{\pi,\xi}(ps)$$

$$\leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ps,\pi|_I,v) + \varepsilon_\mathrm{n}$$

For $\overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(ps)$:

$$\begin{aligned}
&\sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(ps,v)\\
&\quad= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \mathrm{rea}^{\mathrm{opt}}_{\varepsilon_\mathrm{n}}(ps,\overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(v)|_{MS})\\
&\quad= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\mathrm{rea}^{\mathrm{opt}}(ps,\overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(v)|_{MS}) - \underbrace{\varepsilon'}_{\leqslant\varepsilon_\mathrm{n}}\right)\\
&\quad\succcurlyeq_{\mathrm{opt}} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \cdot \left(\mathrm{rea}^{\mathrm{opt}}(ps,\overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(v)|_{MS})\right) - \mathbb{1}_{\{\sup\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n}\\
&\quad= \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i) \operatorname*{opt}_{\pi'\in\Pi_{\mathrm{stat}}} \sum_{ms\in MS} \mathrm{rea}^{\pi'}(ps,ms)\cdot \overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(ms,v) - \mathbb{1}_{\{\sup\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n}\\
&\quad\succcurlyeq_{\mathrm{opt}} \operatorname*{opt}_{\pi'\in\Pi_{\mathrm{stat}}} \sum_{ms\in MS} \mathrm{rea}^{\pi'}(ps,ms)\cdot \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i)\cdot \overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(ms,v) - \mathbb{1}_{\{\sup\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n}\\
&\quad= \operatorname*{opt}_{\pi'\in\Pi_{\mathrm{stat}}} \sum_{ms\in MS} \mathrm{rea}^{\pi'}(ps,ms)\cdot \overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(ms) - \mathbb{1}_{\{\sup\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n}\\
&\quad= \mathrm{rea}^{\mathrm{opt}}(ps,\overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}|_{MS}) - \mathbb{1}_{\{\sup\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n}\\
&\quad\succcurlyeq_{\mathrm{opt}} \overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(ps) - \mathbb{1}_{\{\sup\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n} + \mathbb{1}_{\{\inf\}}(\mathrm{opt})\cdot\varepsilon_\mathrm{n}
\end{aligned}$$

And therefore

$$\begin{aligned}
\overline{u}^{t+\delta,v}_{\sup,\xi}(ps) &\leqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i)\cdot\overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(ps,v) + \varepsilon_\mathrm{n}\\
\overline{u}^{t+\delta,v}_{\inf,\xi}(ps) &\geqslant \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i)\cdot\overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(ps,v) - \varepsilon_\mathrm{n}
\end{aligned}$$

We can now bound the difference as follows:

$$\begin{aligned}
&\overline{u}^{t+\delta,v}_{\mathrm{opt},\xi}(ps) - u^{t+\delta,v}_{\pi,\xi}(ps)\\
&\quad\preccurlyeq_{\mathrm{opt}} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i)\cdot\left(\overline{\mathbf{D}}^i_{\varepsilon_\mathrm{n}}(ps,v|_{MS}) - \mathbf{D}^i_{\varepsilon_\mathrm{n}}(ps,\pi|_I,v)\right) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})}\cdot 2\cdot\varepsilon_\mathrm{n}\\
&\quad\preccurlyeq_{\mathrm{opt}} \sum_{i=0}^{N(\delta,\varepsilon_\Psi)} \Psi_{\mathbf{E}_{\max}\cdot\delta}(i)\cdot D^i_{\mathrm{opt},\xi} + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})}\cdot 2\cdot\varepsilon_\mathrm{n}\\
&\quad= \mathrm{err}^2_{\pi,\xi}(I,\delta)
\end{aligned}$$

This concludes the proof. □

Algorithm 2 `SwitchStep`

Input: MA $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$, goal function g, time-bound $b \in \mathbb{R}_{\geqslant 0}$, precision $\epsilon > 0$

Output: error estimate $\epsilon' \in (0, 1]$, $\vec{v} \in [0, 1]^{|S|}$, s. t. $||\vec{v} - \mathrm{val}_{\mathrm{opt}}(b, g)||_\infty < \epsilon'$, and an ϵ'-optimal scheduler $\pi_{\mathtt{opt}}$

Parameters: $\delta^{prm} \in (0, 1/\mathbf{E}_{\max}]$, $\overline{D}^{prm}, \underline{D}^{prm} \in \mathbb{Q}_{\geqslant 0}, K^{prm} = 0..|S|, \varepsilon^{prm} \in [0, 1)$, if $b = 0 : \varepsilon_{\mathrm{i}}^{prm} \in [0, \epsilon]$, else $\varepsilon_{\mathrm{i}}^{prm} \in [0, \epsilon)$

Default Values: $\varepsilon_{\mathrm{i}}^{prm} = 0, \delta^{prm} = \min\{10^{-12}, 1/\mathbf{E}_{\max}\}, \overline{D}^{prm} = 10^{12}, \underline{D}^{prm} = 10^{-20}, K^{prm} = 10, \varepsilon^{prm} = 0$

```
1: Initialise $\underline{\delta}, \varepsilon_{\mathrm{rea}}, \varepsilon_\Psi$ as described in Section 4.3.5, Variable Initialisation.
2: $\varepsilon_{\mathrm{n}} = \varepsilon_{\mathrm{rea}} / \left(N(b, \varepsilon_\Psi) + 1\right)$
3: **if** (Assumption 4.3.1 is satisfied) **then** $\delta_{\min} = \max\{\delta^{prm}, \underline{\delta}\}$ **else** $\delta_{\min} = \delta^{prm}$
4: $t = 0$, $\varepsilon_{\mathrm{acc}}^{t} = \varepsilon_{\mathrm{i}}^{prm}$, $\xi = (\varepsilon_\Psi, \varepsilon_{\mathrm{rea}}, \varepsilon_{\mathrm{n}})$
5: $v^0, \pi \leftarrow \varepsilon_{\mathrm{i}}^{prm}$-approximation of $\mathrm{rea}^{\mathrm{opt}}(g)$ and respective $\varepsilon_{\mathrm{i}}^{prm}$-optimal scheduler
6: $\pi_{\mathtt{opt}} = \pi|_{[0,0]}$
7: **while** $t < b$ **do**
8:     $\pi, \widetilde{\pi}_{\varepsilon_{\mathrm{rea}}}(v^t) = \mathtt{FindStrat}(v^t|_{MS})$        ▷ see Section 4.3.2
9:     $t + \delta, \varepsilon_{\mathrm{acc}}^{t+\delta} = \mathtt{FindStep}((t, b], \pi_{\mathtt{opt}}|_{[0,t]} \cdot \pi|_{(t,b]})$        ▷ Algorithm 3
10:    **if** (Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}$ and $\delta \leqslant \delta_{\min}$) **then**
11:        $v^{t+\delta} = \widetilde{\mathrm{val}}_{\mathrm{opt}}^{\varepsilon_{\mathrm{rea}}}(\delta, v^t)$        ▷ see (4.39)
12:        **if** (opt = inf) **then** $v^{t+\delta} = v^{t+\delta} + \varepsilon_{\mathrm{rea}} + \frac{(\mathbf{E}_{\max} \cdot \delta)^2}{2}$
13:        $\pi_{\mathtt{opt}} = \pi_{\mathtt{opt}}|_{[0,t]} \cdot \widetilde{\pi}_{\varepsilon_{\mathrm{rea}}}(v^t)|_{(t,t+\delta]}$
14:    **else**
15:        $v^{t+\delta} = u_{\pi,\xi}^{t+\delta, v^t}$        ▷ see (4.27)
16:        **if** (opt = inf) **then** $v^{t+\delta} = v^{t+\delta} + \varepsilon_{\mathrm{rea}} + \varepsilon_\Psi$
17:        $\pi_{\mathtt{opt}} = \pi_{\mathtt{opt}}|_{[0,t]} \cdot \pi|_{(t,t+\delta]}$
18:    $t = t + \delta$
19: **return** $\varepsilon_{\mathrm{acc}}^{b}, v^b, \pi_{\mathtt{opt}}$
```

4.3.5 Algorithm

Based on all the results presented in the previous sections we developed Algorithm 2 that approximates solution to Problem 2 for a Markov automaton $\mathcal{M}$, goal function g, time-bound $b \in \mathbb{R}_{\geqslant 0}$. In the following we will also refer to this algorithm with `SwitchStep`. The algorithm has the following parameters: $\varepsilon_{\mathrm{i}}^{prm}$ denotes the upper bound on the error introduced by approximation of the reachability value for time point 0, δ^{prm} is the lower bound on the length of time step that can be performed at each iteration, parameters $\overline{D}^{prm}, \underline{D}^{prm}, K^{prm}, \varepsilon^{prm}$ were discussed in Section 4.3.2, they affect the quality of the stationary strategy selected at the current iteration.

Notice that not all combinations of ϵ and δ^{prm} are possible. For example, parameter δ^{prm} may be set so high, that there exists no ϵ-optimal scheduler for $\mathrm{val}_{\mathrm{opt}}(b, g)$.

Algorithm 3 `FindStep`

```
Input: interval $I = (a_1, a_2]$, $\pi \in \Pi_{\texttt{PC}}$,
Output: time point $a \in (a_1, a_2]$ and upper bound on accumulated error $\varepsilon \geqslant 0$
 1: $\{(t_i, t_{i+1}] \mid i = 0..n-1\} = \texttt{ComputeIntervals}(a_1, a_2, \varepsilon_h = \varepsilon_\Psi, \delta_h = \delta_{\min})$
 2: $k = 0$
    set boolean function $f : \mathbb{R}_{\geqslant 0} \to \{0,1\}$ as follows:
 3: if (Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}$) then
 4:     $f(t) = ((t - a_1 \geqslant \delta_{\min})$ and $((t - a_1) \mod \delta_{\min} = 0))$
 5: else
 6:     $f(t) = (t - a_1 \geqslant \delta_{\min})$
 7: do
 8:     set $\mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$ or $A = PS \times Act$
 9:     $\mathrm{err}_{t_{k+1}} = \varepsilon_{\mathrm{i}}^{prm} + (\epsilon - \varepsilon_{\mathrm{i}}^{prm}) \cdot t_{k+1}/b$
10:     $toswitch, \varepsilon = \texttt{CheckInterval}((a_1, t_{k+1}], \mathrm{err}_{t_{k+1}}, \pi, A)$
11:     $k = k + 1$
12: while ((not $toswitch$) and $k < n$)

13: if ($toswitch = true$ and $a_2 - a_1 \geqslant \delta_{\min}$) then
14:     compute the largest $t' \in (t_{k-1}, t_k]$, s. t. $f(t') = true$ and for $\mathrm{err}_{t'} = \varepsilon_{\mathrm{i}}^{prm} +$
    $(\epsilon - \varepsilon_{\mathrm{i}}^{prm}) \cdot t'/b$ the following holds:
            $\texttt{CheckInterval}((a_1, t'], \mathrm{err}_{t'}, \pi, A) = \text{false}, \varepsilon'$
15:     if (such a value $t'$ exists) then
16:         return $t', \varepsilon'$
17:     else
18:         $t' = a_1 + \delta_{\min}$
19:         $b, \varepsilon' = \texttt{CheckInterval}((a_1, t'], \mathrm{err}_{t'}, \pi, A)$
20:         return $t', \varepsilon'$
21: else
22:     return $a_2, \varepsilon$
```

`SwitchStep` treats parameter δ^{prm} as a hard constraint and input value ϵ as a soft constraint. Namely, the algorithm only performs time steps of length at least δ^{prm}, even if this means that the total error accumulated throughout computations exceeds ϵ. An estimate of the accumulated error is one of the output values of the algorithm.

`SwitchStep` returns values $\epsilon', \vec{v} \in [0,1]^{|S|}$ and $\pi \in \Pi_{\texttt{PC}}$, such that $\vec{v}$ is an ϵ'-close approximation of $\mathrm{val}_{\mathrm{opt}}(b, g)$ and π is an ϵ'-optimal scheduler. Given that Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}$, `SwitchStep` is guaranteed to output $\epsilon' \leqslant \epsilon$.

We assume that all the procedures called from Algorithm 2 have access to the input values $\mathcal{M}$, b, g and ϵ, parameters as well as variables defined within Algorithm 2, even if they are not listed explicitly as input values.

The algorithm iterates over intervals $\mathcal{I}(\pi_{\texttt{opt}})$ of an ϵ-optimal strategy $\pi_{\texttt{opt}}$. At each iteration it computes: (i) a scheduler π that is believed to be close to optimal on the current interval (line 8) (via procedure `FindStrat`), (ii) length δ of the interval, on which π introduces acceptable error, together with the estimate of this error (line 9) (procedure `FindStep`) and (iii) the reachability values for time $t + \delta$ (lines 11, 15).

Procedure `FindStep` is presented in Algorithm 3. It takes as input an interval $(a_1, a_2]$ and a piecewise-constant strategy, that is stationary on interval $(a_1, a_2]$, and outputs $t \in (a_1, a_2]$, such that either the error introduced by π for time t is within acceptable range, or the smallest allowed t, which is $a_1 + \delta_{\min}$. The procedure approximates switching points iteratively. It uses partition of $(a_1, a_2] = (t_0, t_1], \ldots, (t_{n-1}, t_n]$ computed by procedure `ComputeIntervals` and at each iteration k checks whether the error accumulated over interval $[0, t_{k+1}]$ is small enough. The latter is performed by procedure `CheckInterval`. When `FindStep` finds the value t_{k+1} that induces larger than acceptable error, it refines $(t_k, t_{k+1}]$ until it finds the largest still acceptable subinterval $(t_k, t'] \subset (t_k, t_{k+1}]$, or it becomes clear that all such intervals are shorter than the lower bound on the interval length, specified by $\delta_{\min}$. Procedure `FindStep` also computes the set of transitions that will be used by `CheckInterval` in line 8. The algorithm is correct irrespective of whether $\mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$ or $PS \times Act$ is used. For efficiency reasons in our implementation we only compute $\mathcal{T}_{\max}(N(t_{k+1} - t, \varepsilon_\Psi))$ before the call to `CheckInterval` at line 14, and use the set $A = PS \times Act$ within the while-loop.

Procedure `CheckInterval`$((a_1, a_2], err, \pi, A)$ tests whether the error accumulated within interval $[0, a_2]$ is below the allowed threshold *err*. It achieves this by first computing

$$e_1 = \begin{cases} \mathrm{err}^1_{\pi,\xi}(A, (a_1, a_2]) & \text{if Assumption 4.3.1 is satisfied and } \mathcal{M} \text{ is } PS-\text{acyclic} \\ 1 & \text{otherwise} \end{cases}$$

$$e_2 = |\mathrm{err}^2_{\pi,\xi}((a_1, a_2], a_2 - a_1)|$$

$$e_m = \begin{cases} (\mathbf{E}_{\max} \cdot (a_2 - a_1))^2/2 - \varepsilon_\Psi & \text{if Assumption 4.3.1 is satisfied,} \\ & \delta^{prm} \leqslant \underline{\delta} \text{ and } a_2 - a_1 \leqslant \underline{\delta} \\ \min\{e_1, e_2\} & \text{otherwise} \end{cases}$$

And then checking whether $\varepsilon_{\mathrm{acc}}^{a_1+\delta} = \varepsilon_{\mathrm{acc}}^{a_1} + \varepsilon_{\mathrm{rea}} + \varepsilon_\Psi + e_m \leqslant err$. If yes, it outputs false (no switching point) and $\varepsilon_{\mathrm{acc}}^{a_1+\delta}$, otherwise it outputs true (there may be a switching point) and $\varepsilon_{\mathrm{acc}}^{a_1+\delta}$. Here values $\mathrm{err}^1_{\pi,\xi}(A, (a_1, a_2])$ and $\mathrm{err}^2_{\pi,\xi}((a_1, a_2], a_2 - a_1)$ are computed based on $v^{a_1}(s)$.

Variable Initialisation. Here we will discuss how to initialise variables $\varepsilon_{\mathrm{rea}}, \varepsilon_\Psi$ and $\underline{\delta}$ depending on the values of parameters and inputs of `SwitchStep`.

If $b = 0$ then variables $\varepsilon_{\mathrm{rea}}, \varepsilon_\Psi$ and $\underline{\delta}$ have no effect on the output of `SwitchStep` and can therefore be initialised to any value.

If $b > 0$ then choose the values that satisfy the following requirements:

$$
\begin{aligned}
&\begin{cases} \varepsilon' = 0 & \text{or} \\ 0 < \varepsilon' \leqslant \frac{(\epsilon-\varepsilon_{\mathrm{i}}^{prm})^2}{2\cdot(\mathbf{E}_{\max}\cdot b)^2} \cdot \left(1 - \frac{(\epsilon-\varepsilon_{\mathrm{i}}^{prm})^2}{4\cdot(\mathbf{E}_{\max}\cdot b)^4}\right) & \text{if } \varepsilon_{\mathrm{i}}^{prm} \in [\epsilon - 2\cdot(\mathbf{E}_{\max}\cdot b)^2, \epsilon) \cap [0,1] \end{cases} \\
&D = (\epsilon - \varepsilon_{\mathrm{i}}^{prm})^2 - 2\cdot\varepsilon'\cdot(\mathbf{E}_{\max}\cdot b)^2 \\
&N = \begin{cases} \left\lceil \max\{\frac{(\mathbf{E}_{\max}\cdot b)^2}{2\cdot(\epsilon-\varepsilon_{\mathrm{i}}^{prm})}, b\cdot\mathbf{E}_{\max}\} \right\rceil & \text{if } \varepsilon' = 0 \\ \\ N', \text{ s.t. } N' \in \left[\left\lceil \frac{\epsilon-\varepsilon_{\mathrm{i}}^{prm}-\sqrt{D}}{2\cdot\varepsilon'} \right\rceil, \left\lfloor \frac{\epsilon-\varepsilon_{\mathrm{i}}^{prm}+\sqrt{D}}{2\cdot\varepsilon'} \right\rfloor \right] \text{ and } b/N' \leqslant 1/\mathbf{E}_{\max} & \text{otherwise} \end{cases} \\
&\underline{\delta}' = b/N \\
&\varepsilon'' \text{ satisfies: } 0 < \varepsilon'' \leqslant \frac{\epsilon - \varepsilon_{\mathrm{i}}^{prm}}{N} - \varepsilon'
\end{aligned}
\tag{4.76}
$$

If Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}'$, set $\varepsilon_{\mathrm{rea}} = \varepsilon', \varepsilon_{\Psi} = \varepsilon'', \underline{\delta} = \underline{\delta}'$. Otherwise set $\varepsilon_{\mathrm{rea}} \in [0,1), \varepsilon_{\Psi} \in (0,1)$ to such values that:

$$\varepsilon_{\mathrm{rea}} + \varepsilon_{\Psi} \leqslant (\epsilon - \varepsilon_{\mathrm{i}}^{prm}) \cdot \min\{\delta^{prm}, b\}/b \tag{4.77}$$

In our experimental results we observed that better results were achieved when $\varepsilon_{\mathrm{rea}} = 0$. This is likely due to the case that all known published case studies are *PS*-acyclic and thus exact computation of hop-unbounded reachability probability is quite efficient - linear in the size of the Markov automaton. And regarding the value of ε_{Ψ}, we used $\varepsilon_{\Psi} = \min\{10^{-20}, (\epsilon - \varepsilon_{\mathrm{i}}^{prm})/N - \varepsilon_{\mathrm{rea}}\}$ for the case when Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}'$, otherwise we set $\varepsilon_{\Psi} = \min\{10^{-20}, (\epsilon - \varepsilon_{\mathrm{i}}^{prm}) \cdot \min\{\delta^{prm}, b\}/b\}$. This is explained by the fact that $N(\tau, \varepsilon_{\Psi})$ grows slow with decrease of ε_{Ψ}, on the other hand smaller values of ε_{Ψ} leave more room for error induced by following sub-optimal strategy as opposed to the optimal one (e_m), which tends to be the major source of error.

Complexity. Let $|S| = n$ be the amount of states, $a = |Act|$ and m is the amount of *edges* in $\mathcal{M}$ (see Section 4.1.1). Recall that $C_{rea}^{\mathrm{opt}}(n,m,a)$ and $C_{rea}(n,m,a)$, defined in Section 4.2.2, denote the complexity of computing $\mathrm{rea}^{\mathrm{opt}}(g)$ and $\mathrm{rea}^{\pi}(g)$ respectively.

Performance of `SwitchStep` depends strongly on the number of iterations it performs. The algorithm adapts the number of iterations to a specific instance of the problem and its runtime benefits from this. The worst-case should not happen on real-life case studies unless the strategy selected by `FindStrat` is far from being optimal (for example if the parameters used for `FindStrat` are not good enough, or if a different heuristic is used instead of `FindStrat` to guess a strategy). In the worst case $k = b/\delta_{\min}$. If Assumption 4.3.1 is satisfied, then $k \leqslant N$ (defined above in Section Variable Initialisation.) Otherwise $\delta_{\min}$ equals the value of user-specified parameter δ^{prm} and therefore $k = b/\delta^{prm}$.

At each iteration the algorithm calls procedures `FindStrat`, `FindStep` and approximates the reachability values according to (4.27). The complexity of the latter operation is in the worst case $O(N(b, \varepsilon_{\Psi}) \cdot (C_{rea}(n,m,a) + m))$

Procedure `FindStrat` performs at most $n+1$ iterations. At each iteration, it solves the hop-unbounded reachability problem on probabilistic states of the MA ($C^{\text{opt}}_{rea}(n,m,a)$) and computes derivatives, which takes $O(m+n)$ time for Markovian states and $C_{rea}(n,m,a)$ time for probabilistic states. Thus the complexity of `FindStrat` is in $O((n+1)\cdot(C^{\text{opt}}_{rea}(n,m,a)+m+n))$.

The complexity of procedure `FindStep` is determined by complexities of procedures `CheckInterval` and `ComputeIntervals`. Due to Lemma 4.3.11 the complexity of the latter is in the worst case in $O(N(b,\varepsilon_\Psi)\cdot b\cdot \log(b)/\mathbf{E}_{\max})$.

Complexity of procedure `CheckInterval` comes from the computation of values $\text{err}^1_{\pi,\xi}(A,(a_1,a_2])$ and $|\text{err}^2_{\pi,\xi}((a_1,a_2],a_2-a_1)|$. Computing $\text{err}^1_{\pi,\xi}(A,(a_1,a_2])$ requires in the worst case $O(|A|\cdot N(b,\varepsilon_\Psi)\cdot(C_{rea}(n,m,a)+m))$, where $A = PS\times Act$, or is the set of maximal transitions. Computation of $|\text{err}^2_{\pi,\xi}((a_1,a_2],a_2-a_1)|$ is in $O(N(b,\varepsilon_\Psi)\cdot n\cdot(C^{\text{opt}}_{rea}(n,m,a)+m))$. Thus the complexity of `CheckInterval` in the worst case is in $O(|A|\cdot N(b,\varepsilon_\Psi)\cdot n\cdot(C^{\text{opt}}_{rea}(n,m,a)+m))$.

Due to Lemma 4.3.11, procedure `FindStep` checks in the worst case $O(N(b,\varepsilon_\Psi)\cdot b/\mathbf{E}_{\max})$ intervals computed by procedure `ComputeIntervals`. At each iteration it calls procedure `CheckInterval` and possibly computes maximal transitions ($O(m^2\cdot N(b,\varepsilon_\Psi))$). Afterwards it performs at most $\log(\min\{1/\mathbf{E}_{\max},b\})$ (due to Lemma 4.3.9) iterations and calls `CheckInterval` at each of them. Thus overall the complexity of `FindStep` is determined by worst-case $O(N(b,\varepsilon_\Psi)\cdot b/\mathbf{E}_{\max})$ iterations each of complexity of `CheckInterval` and one call to `ComputeIntervals`.

The general worst-case complexity of Algorithm 2 depends on many parameters and we will instead discuss the most common scenario. If Assumption 4.3.1 is satisfied, the MA is *PS*-acyclic, $\varepsilon_{\text{rea}}=0, \varepsilon^{prm}_{\text{i}}=0$, $b>\mathbf{E}_{\max}$, then the overall worst-case complexity is $O(k\cdot(b\cdot\mathbf{E}_{\max}+|\log(\varepsilon_\Psi)|)^2\cdot(n^2\cdot a\cdot m+\frac{b\cdot\log(b)}{\mathbf{E}_{\max}})\cdot b/\mathbf{E}_{\max})$, where k is the number of iterations of the main loop.

Lemma 4.3.17. *Let $a_1,a_2\in\mathbb{R}_{\geqslant 0}, a_1<a_2$, $\delta_{\min},\xi,v^x$ are the variables computed by Algorithm 2, $\varepsilon^{a_1}_{acc}$ is the respective variable that Algorithm 2 takes when $t=a_1$ and $\pi,\widetilde{\pi}_{\varepsilon_{rea}}(v^{a_1}) = \texttt{FindStrat}(v^{a_1}|_{MS})$.*
Let $\pi_{\texttt{opt}}\in\Pi_{\texttt{PC}}$ be such a scheduler that $\forall s\in S: v^{a_1}(s)\preccurlyeq_{\text{opt}}\text{val}_{\pi_{\texttt{opt}}}(s,a_1,g)$ and

$$v^{a_1}(s)\preccurlyeq_{\text{opt}}\text{val}_{\text{opt}}(s,a_1,g)\preccurlyeq_{\text{opt}} v^{a_1}(s)+(-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})}\cdot\varepsilon^{a_1}_{acc}$$

Let $\pi\in\{\pi,\widetilde{\pi}_{\varepsilon_{rea}}(v^{a_1})\}$ be the stationary scheduler selected by Algorithm 2 at the current iteration and $\pi'_{\texttt{opt}}=\pi_{\texttt{opt}}|_{[0,a_1]}\cdot\pi|_{(a_1,a_2]}$.
If $\texttt{FindStep}((a_1,a_2],\pi'_{\texttt{opt}})=\delta,\varepsilon^{a_1+\delta}_{acc}$, then $\pi'_{\texttt{opt}}$ is $\varepsilon^{a_1+\delta}_{acc}$-optimal for $\text{val}_{\text{opt}}(a_1+\delta,g)$ and

$$v^{a_1+\delta}\preccurlyeq_{\text{opt}}\text{val}_{\text{opt}}(s,a_1+\delta,g)\preccurlyeq_{\text{opt}} v^{a_1+\delta}(s)+(-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})}\cdot\varepsilon^{a_1+\delta}_{acc}$$

Proof. Procedure `FindStep` outputs as error $\varepsilon^{a_1+\delta}_{\text{acc}}$ the value computed by procedure `CheckInterval`. By construction, the latter computes it as follows $\varepsilon^{a_1+\delta}_{\text{acc}}=\varepsilon^{a_1}_{\text{acc}}+\varepsilon_{\text{rea}}+\varepsilon_\Psi+e_m$.

Consider the case when Assumption 4.3.1 is satisfied, $\delta^{prm}\leqslant\underline{\delta}$. In this case $\delta_{\min}=\underline{\delta}$ and the step size does not go below $\delta_{\min}$, unless $a_2-a_1<\delta_{\min}$. We at first consider the case when $\delta \leqslant \delta_{\min}$. Here `CheckInterval` sets $e_m = (\mathbf{E}_{\max} \cdot \delta)^2/2 - \varepsilon_\Psi$. If opt $=$ sup:

$$\begin{aligned}
v^{a_1+\delta} &= \widetilde{\mathrm{val}}^{\varepsilon_{\mathrm{rea}}}_{\sup}(\delta, v^{a_1}) \\
&\overset{\text{Lemma 4.3.5}}{\leqslant} \mathrm{val}_{\sup}(s, a_1+\delta, g) \\
&\overset{\text{Lemma 4.3.5}}{\leqslant} \widetilde{\mathrm{val}}^{\varepsilon_{\mathrm{rea}}}_{\sup}(\delta, v^{a_1}) + \varepsilon^{a_1}_{\mathrm{acc}} + \varepsilon_{\mathrm{rea}} + (\mathbf{E}_{\max} \cdot \delta)^2/2 \\
&= \widetilde{\mathrm{val}}^{\varepsilon_{\mathrm{rea}}}_{\sup}(\delta, v^{a_1}) + \varepsilon^{a_1+\delta}_{\mathrm{acc}} \\
&= v^{a_1+\delta} + \varepsilon^{a_1+\delta}_{\mathrm{acc}}
\end{aligned}$$

And for opt $=$ inf:

$$\begin{aligned}
v^{a_1+\delta} &= \widetilde{\mathrm{val}}^{\varepsilon_{\mathrm{rea}}}_{\inf}(\delta, v^{a_1}) + \varepsilon_{\mathrm{rea}} + (\mathbf{E}_{\max} \cdot \delta)^2/2 \\
&\overset{\text{Lemma 4.3.5}}{\geqslant} \mathrm{val}_{\inf}(s, a_1+\delta, g) \\
&\overset{\text{Lemma 4.3.5}}{\geqslant} \widetilde{\mathrm{val}}^{\varepsilon_{\mathrm{rea}}}_{\inf}(\delta, v^{a_1}) - \varepsilon^{a_1}_{\mathrm{acc}} \\
&= v^{a_1+\delta} - \varepsilon^{a_1+\delta}_{\mathrm{acc}}
\end{aligned}$$

The scheduler in this case is defined as $\pi'_{\mathtt{opt}} = \pi_{\mathtt{opt}}|_{[0,a_1]} \cdot \widetilde{\pi}_{\varepsilon_{\mathrm{rea}}}(v^{a_1})|_{(a_1,a_1+\delta]}$. Due to Lemma 4.3.6, $\pi'_{\mathtt{opt}}$ is $\varepsilon^{a_1+\delta}_{\mathrm{acc}}$-optimal for $\mathrm{val}_{\mathrm{opt}}(a_1+\delta, g)$.

In all other cases the procedure is the same. Therefore

$$\begin{aligned}
v^{a_1+\delta} &= u^{a_1+\delta, v^{a_1}}_{\pi_{\mathtt{opt}},\xi} \\
&\overset{\text{Lemmas 4.3.15, 4.3.16}}{\leqslant} \mathrm{val}_{\sup}(s, a_1+\delta, g) \\
&\overset{\text{Lemmas 4.3.15, 4.3.16}}{\leqslant} u^{a_1+\delta, v^{a_1}}_{\pi_{\mathtt{opt}},\xi} + \varepsilon^{a_1}_{\mathrm{acc}} + \varepsilon_{\mathrm{rea}} + \varepsilon_\Psi + e_m \\
&= v^{a_1+\delta} + \varepsilon^{a_1+\delta}_{\mathrm{acc}}
\end{aligned}$$

And for opt $=$ inf:

$$\begin{aligned}
v^{a_1+\delta} &= u^{a_1+\delta, v^{a_1}}_{\pi_{\mathtt{opt}},\xi} + \varepsilon_{\mathrm{rea}} + \varepsilon_\Psi \\
&\overset{\text{Lemmas 4.3.15, 4.3.16}}{\geqslant} \mathrm{val}_{\inf}(s, a_1+\delta, g) \\
&\overset{\text{Lemmas 4.3.15, 4.3.16}}{\geqslant} u^{a_1+\delta, v^{a_1}}_{\pi_{\mathtt{opt}},\xi} - \varepsilon^{a_1}_{\mathrm{acc}} - e_m \\
&= v^{a_1+\delta} - \varepsilon^{a_1}_{\mathrm{acc}} - \varepsilon_{\mathrm{rea}} - \varepsilon_\Psi - e_m \\
&= v^{a_1+\delta} - \varepsilon^{a_1+\delta}_{\mathrm{acc}}
\end{aligned}$$

The scheduler in this case is defined as $\pi'_{\mathtt{opt}} = \pi_{\mathtt{opt}}|_{[0,a_1]} \cdot \pi|_{(a_1,a_1+\delta]}$. Due to Lemmas 4.3.15, 4.3.16, $\pi'_{\mathtt{opt}}$ is $\varepsilon^{a_1+\delta}_{\mathrm{acc}}$-optimal for $\mathrm{val}_{\mathrm{opt}}(a_1+\delta, g)$.

□

Theorem 4.3.18. *Let $\mathcal{M}$ be a Markov automaton, g is a goal function, $b \in \mathbb{R}_{\geqslant 0}$ and $\epsilon \in (0,1)$. Let $\epsilon', \vec{v} \in [0,1]^{|S|}, \pi_{\mathtt{opt}}$ be the output of Algorithm 2 for $\mathcal{M}, g, b, \epsilon$. Then $\forall s \in S$:*

$$\vec{v}(s) \preccurlyeq_{\text{opt}} \text{val}_{\text{opt}}^{\mathcal{M}}(s,b,g) \preccurlyeq_{\text{opt}} \vec{v}(s) + (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \cdot \epsilon' \tag{4.78}$$

and $\pi_{\mathtt{opt}}$ is ϵ'-optimal for $\text{val}_{\text{opt}}^{\mathcal{M}}(b,g)$. If Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}$, then $\epsilon' \leqslant \epsilon$, otherwise ϵ' may be larger than ϵ.

Proof. Consider the values $t_i, i \in \mathbb{Z}_{\geqslant 0}$, that variable t takes in Algorithm 2 at iteration i. Here $t_0 = 0$ and for $i \geqslant 1$ the value t_i is computed by procedure `FindStep` based on t_{i-1}. Procedure `FindStep` iterates over intervals computed by `ComputeIntervals` and outputs a value t_i, such that either $t_i - t_{i-1} \geqslant \delta_{\min} > 0$ (procedure `FindStep`, lines 18 - 20), or $0 < t_i - t_{i-1} < \delta_{\min}$ and $t_i = b$. Thus variable t in Algorithm 2 takes finitely many values $0 = t_0, \ldots, t_n = b$.

Let $\varepsilon_0 = \varepsilon_{\text{i}}^{prm}$ and $\forall i = 1..n : \varepsilon_i$ is the error returned by `FindStep` at iteration i. The output value $\vec{v}$ of Algorithm 2 equals the value that variable v^x takes for $x = b$. We will first prove that (4.78) holds and that $\pi_{\mathtt{opt}}$ is ϵ'-optimal for $\text{val}_{\text{opt}}^{\mathcal{M}}(b,g)$. We achieve this by showing by induction that

$$\forall k = 0..n, s \in S : v^{t_k}(s) \preccurlyeq_{\text{opt}} \text{val}_{\text{opt}}(s, t_k, g) \preccurlyeq_{\text{opt}} v^{t_k}(s) + (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \varepsilon_k \tag{4.79}$$

and $\pi_{\mathtt{opt}}|_{[0,t_k]}$ is ε_k-optimal for $\text{val}_{\text{opt}}(t_k, g)$. Notice that $\epsilon' = \varepsilon_n$.

We start with $t_0 = 0$. Here the strategy $\pi_{\mathtt{opt}}|_{[0,0]}$ is computed in step 6 of Algorithm 2 as an $\varepsilon_{\text{i}}^{prm}$-optimal strategy for $\text{rea}^{\text{opt}}(g)$ and thus

$$v^0(s) \preccurlyeq_{\text{opt}} \text{rea}^{\text{opt}}(g) \preccurlyeq_{\text{opt}} v^0(s) + (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \varepsilon_{\text{i}}^{prm} = v^0(s) + (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \varepsilon_0 \tag{4.80}$$

Thus the claim follows due to Lemma 4.1.4, (4.20).

For $t_k, k = 1..n$, (4.79) follows from Lemma 4.3.17 and the induction hypothesis. This concludes the proof of the first part. Condition (4.77) ensures that:

$$\begin{aligned}
\varepsilon_k &= \varepsilon_{k-1} + \varepsilon_{\text{rea}} + \varepsilon_{\Psi} + e_m \\
&\leqslant \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_{k-1}/b + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot \min\{\delta^{prm}, b\}/b + e_m \\
&\leqslant \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_k/b + e_m
\end{aligned}$$

If $e_m \leqslant (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot (t_k - t_{k-1} - \min\{\delta^{prm}, b\})/b$, then $\epsilon' \leqslant \epsilon$. This however cannot be fulfilled for arbitrary values of δ^{prm}.

Next, we show that if Assumption 4.3.1 is satisfied and $\delta^{prm} \leqslant \underline{\delta}$, then $\epsilon' \leqslant \epsilon$. Since $\delta^{prm} \leqslant \underline{\delta}$, then $\delta_{\min} = \underline{\delta}$. Notice that in this case $\forall i = 1..n : (t_i - t_{i-1}) \bmod \delta_{\min} = 0$ and $b = \delta_{\min} \cdot N$. We prove the statement by showing the following:

$$\forall k = 0..n : \varepsilon_k \leqslant \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_k/b \tag{4.81}$$

Given this, it follows that $\epsilon' = \varepsilon_n \leqslant \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_n/b = \epsilon$.

For $k = 0$ (4.81) holds by definition of ε_0. Consider interval $(t_{k-1}, t_k], k = 1..n$. The value of ε_k is an output of one of the calls to `CheckInterval` in `FindStep`.

If `CheckInterval` returns *false* for interval $(t_{k-1}, t_k]$, then, by construction $\varepsilon_k = \varepsilon_{k-1} + \varepsilon_{\text{rea}} + \varepsilon_\Psi + e_m \leqslant \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_k/b$. Conditions imposed by (4.76) on ε_Ψ ensure that: $\varepsilon_{\text{rea}} + \varepsilon_\Psi \leqslant (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot \delta_{\text{min}}/b$. This way there is some room left for e_m, to make sure that $t_k - t_{k-1} > \delta_{\text{min}}$ can actually happen.

Procedure `CheckInterval` can only return *true* for interval $(t_{k-1}, t_k]$ if $t_k - t_{k-1} = \delta_{\text{min}}$. Then `CheckInterval` outputs $\varepsilon_k = \varepsilon_{k-1} + \varepsilon_{\text{rea}} + (\mathbf{E}_{\text{max}} \cdot \delta_{\text{min}})^2/2$. We will show that

$$\varepsilon_{\text{rea}} + (\mathbf{E}_{\text{max}} \cdot \delta_{\text{min}})^2/2 \leqslant (\epsilon - \varepsilon_{\text{i}}^{prm})/N \tag{4.82}$$

Taking into account the induction hypothesis, the definition of $\underline{\delta}$ given in (4.76) and that $\delta_{\text{min}} = \underline{\delta}$ this suffices, since

$$\varepsilon_k \leqslant \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_{k-1}/b + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot \delta_{\text{min}}/b = \varepsilon_{\text{i}}^{prm} + (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot t_k/b$$

We will first rewrite the inequality as follows:

$$\varepsilon_{\text{rea}} + \frac{(\mathbf{E}_{\text{max}} \cdot \delta_{\text{min}})^2}{2} - \frac{\epsilon - \varepsilon_{\text{i}}^{prm}}{N} = \varepsilon_{\text{rea}} + \frac{(\mathbf{E}_{\text{max}} \cdot b)^2}{2 \cdot N^2} - \frac{\epsilon - \varepsilon_{\text{i}}^{prm}}{N} \leqslant 0 \Leftrightarrow \tag{4.83}$$

$$\varepsilon_{\text{rea}} \cdot N^2 - (\epsilon - \varepsilon_{\text{i}}^{prm}) \cdot N + \frac{(\mathbf{E}_{\text{max}} \cdot b)^2}{2} \leqslant 0 \tag{4.84}$$

If $\varepsilon_{\text{rea}} = 0$, then N must satisfy $N \geqslant (\mathbf{E}_{\text{max}} \cdot b)^2/2/(\epsilon - \varepsilon_{\text{i}}^{prm})$ and $b/N \leqslant 1/\mathbf{E}_{\text{max}}$. The value assigned to N by (4.76) satisfies this requirement.

If $\varepsilon_{\text{rea}} > 0$ then the value D defined in (4.76) is exactly the discriminant of inequality (4.84). Requirements imposed by (4.76) on ε_{rea} ensure that $D > 0$ and thus the inequality has two real zeros:

$$N_1 = \frac{\epsilon - \varepsilon_{\text{i}}^{prm} - \sqrt{D}}{2 \cdot \varepsilon_{\text{rea}}} > 0, \quad N_2 = \frac{\epsilon - \varepsilon_{\text{i}}^{prm} + \sqrt{D}}{2 \cdot \varepsilon_{\text{rea}}} > 0$$

It can be rewritten as follows:

$$\varepsilon_{\text{rea}} \cdot (N - N_1) \cdot (N - N_2) \leqslant 0$$

The values that satisfy the inequality are those within $[N_1, N_2]$. In the following we will show that there exist at least one integer value within interval $[N_1, N_2]$.

$$N_2 - N_1 = \frac{\sqrt{D}}{\varepsilon_{\text{rea}}} \geqslant 1 \Leftrightarrow \sqrt{D} - \varepsilon_{\text{rea}} \geqslant 0$$

Therefore it is enough to prove the latter.

$$\sqrt{D} - \varepsilon_{\text{rea}} \overset{(4.76)}{\geqslant} \sqrt{D} - \frac{(\epsilon - \varepsilon_{\text{i}}^{prm})^2}{2 \cdot (\mathbf{E}_{\text{max}} \cdot b)^2}$$

Let $a = (\epsilon - \varepsilon_{\text{i}}^{prm})^2$ and $c = 2 \cdot (\mathbf{E}_{\text{max}} \cdot b)^2$. Then

$$\begin{aligned} &\varepsilon_{\text{rea}} \leqslant \frac{a}{c} \cdot \left(1 - \frac{a}{c^2}\right) \Rightarrow \\ &\sqrt{D} - \varepsilon_{\text{rea}} \geqslant \sqrt{a - c \cdot \varepsilon_{\text{rea}}} - \frac{a}{c} \geqslant 0 \end{aligned}$$

Inequality $\varepsilon_{\text{rea}} \leqslant \frac{a}{c} \cdot \left(1 - \frac{a}{c^2}\right)$ is exactly what (4.76) restricts for the value of ε_{rea}. In the following we will establish under which conditions $1 - \frac{a}{c^2} > 0$:

$$1 - \frac{a}{c^2} = (1 - \frac{\sqrt{a}}{c}) \cdot (1 + \frac{\sqrt{a}}{c}) = (1 - \frac{\epsilon - \varepsilon_{\text{i}}^{prm}}{2 \cdot (\mathbf{E}_{\max} \cdot b)^2}) \cdot (1 + \frac{\epsilon - \varepsilon_{\text{i}}^{prm}}{2 \cdot (\mathbf{E}_{\max} \cdot b)^2})$$

Therefore

$$\begin{aligned} 1 - \frac{a}{c^2} > 0 &\Leftrightarrow (\varepsilon_{\text{i}}^{prm} - (\epsilon - 2 \cdot (\mathbf{E}_{\max} \cdot b)^2)) \cdot (\varepsilon_{\text{i}}^{prm} - (\epsilon + 2 \cdot (\mathbf{E}_{\max} \cdot b)^2)) < 0 \\ &\Leftrightarrow \varepsilon_{\text{i}}^{prm} \in (\epsilon - 2 \cdot (\mathbf{E}_{\max} \cdot b)^2, \epsilon + 2 \cdot (\mathbf{E}_{\max} \cdot b)^2) \end{aligned}$$

Taking into account that $\varepsilon_{\text{i}}^{prm}$ cannot exceed ϵ, we obtain condition (4.76) on the values of ε_{rea} and $\varepsilon_{\text{i}}^{prm}$. When $\varepsilon_{\text{rea}} \to 0$, $N_2 \to \infty$ and therefore by lowering the value of ε_{rea} it is possible to ensure that such a value N, that $b/N \leqslant 1/\mathbf{E}_{\max}$ will be found. We have thus proven (4.82).

□

4.4 Time-Bounded Reachability on Partial State-Space

So far we have discussed an approach to compute $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(\Phi_1 \, \text{U}^{[\![a,b]\!]} \Phi_2)$ in an MA $\mathcal{M}_\ell$ by performing computations on the whole state-space of $\mathcal{M}_\ell$. This requires the full state-space to be loaded into memory at some point during the computations. The state-space of MA models however usually grows very large very quickly making this exhaustive approach to model-checking very time- and memory consuming. Consider the system C that comprises of n separate sub-components $C_1, \ldots, C_n$. To build the MA model $\mathcal{M}_C$ of such a system each of the sub-components C_i needs to be modelled as a Markov automaton $\mathcal{M}_{C_i}$. The MA model of the whole system $\mathcal{M}_C$ is obtained by composing $\mathcal{M}_{C_1}, \ldots, \mathcal{M}_{C_n}$ (more detailed discussion on modelling with MA can be found in Chapter 2 Section 2.2). The state-space of $\mathcal{M}_C$ is the reachable fragment of the Cartesian product of state-spaces of $\mathcal{M}_{C_i}$ and thus is in the worst case exponential in the number of sub-components.

In the worst case, there is likely not much we can do and in fact, all the states of the state-space have to be loaded into the memory at some point in time and processed. However, we may not always face the worst-case scenario. Consider, for example, the queuing system shown in Figure 4.2. Here two queues of finite size K store requests arriving with an exponentially distributed delay. Requests are processed by corresponding servers, one per queue. Server 1 may non-deterministically insert a request that it has already processed into the queue of server 2. The state-space of the MA $\mathcal{M}$ modelling this queueing system is a tuple (q_1, q_2, s_1, s_2), where $q_i, i \in \{1, 2\}$ is the amount of requests in queue i and $s_i, i \in \{1, 2\}$ is a state of server i (e. g. *processing a request, awaiting a request*, etc.).

An example of a property that one could be interested in is: *What is the maximum probability of both queues becoming full within some time bound, starting from both queues being empty*? This corresponds to target states of the form (K, K, s_1, s_2), where s_1, s_2 are any states of servers 1 and 2. All the paths reaching states $(K, K, \cdot, \cdot)$ from states $(k_1, k_2, \cdot, \cdot)$ have to go through all the intermediate states $(j_1, j_2, \cdot, \cdot)$,

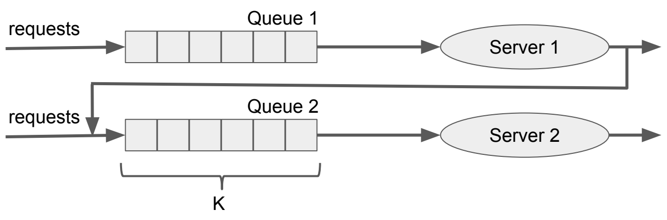


FIGURE 4.2: Queuing system.

where $j_i \in \{k_i, \ldots, K\}$ for $i \in \{1, 2\}$. Thus in order to compute the required probability most of the states of the model have to be processed.

Consider another property: *What is the maximum probability of the second queue becoming full within some time bound, starting from both queues being empty*? Here target states are of the form $(\cdot, K, \cdot, \cdot)$. The most effective strategy to reach a goal state is for the first server to always insert tasks it has taken from its queue into the second queue upon service termination. If the first server follows this strategy, then the second queue will fill up faster, compared to a strategy when the first server does not insert the tasks it has processed into the second queue. Consider the situation in which the rate with which requests arrive into the first queue is lower than the rate with which the first server processes these requests. And on the other hand the rate with which requests arrive into the second queue is higher than the processing rate of the second server. In this scenario the states that are most likely to be visited on the way to a goal state are those with a small number of requests in the first queue. Assuming that the amount of requests in the first queue rarely exceeds 2, all the states $(k_1, \cdot, \cdot, \cdot)$, where $k_1 = 3..K$ may not affect the reachability probability too much.

This example shows that the property under consideration affects the part of state-space that is relevant for the reachability probability. Classical model-checking algorithms, as well as the one discussed in the previous section, do not utilise any information about the property when performing model-checking and perform the computations on the full state-space. In such examples like the one discussed above, when only a subset of states is actually contributing to the reachability probability value, these algorithms may perform many unnecessary computations.

In this section, we target the problem of model-checking Markov automata with large state-spaces against such properties, in which only a small part of the state-space is actually relevant. More precisely, we want to approximate up to an arbitrary value $\epsilon \in (0, 1)$ the solution to Problem 1 of computing the value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s_0, \psi)$ for a Markov automaton $\mathcal{M}_\ell = (S, Act, \dashrightarrow, \mathbf{R}, AP, lab)$, a selected state $s_0 \in S$ and formula $\psi = \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2$. In this section we will refer to states in $\mathrm{Sat}(\Phi_2)$ as *goal* or *target* states. The main idea of our approach is quite simple and is outlined below:

0. $S_{\mathrm{rel}} = \{s_0\}$ // *Initial approximation of the relevant subset*
1. Heuristically add states to the relevant subset S_{rel}.
2. Compute partial MA $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ based on state-space S_{rel}.

3. Approximate $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s_0, \psi)$ by calculating $\underline{v}$ and $\overline{v}$, such that $\underline{v} \leqslant \text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s_0, \psi) \leqslant \overline{v}$ using $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$.
4. If the two approximations are sufficiently close, i. e. $\overline{v} - \underline{v} \leqslant \epsilon$, return $[\underline{v}, \overline{v}]$. Otherwise, repeat from step 1.

We iteratively refine the subset of relevant states S_{rel} using a heuristic. At each iteration, we construct a Markov automaton $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$ which has mainly only these states as a state-space. From this smaller Markov automaton we derive an under- and an over-approximation of value $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s_0, \psi)$ of the given Markov automaton $\mathcal{M}_\ell$. If the approximations are good enough, the computations stop and otherwise the relevant subset is refined and the process repeats. If it is possible for a given Markov automaton and property to discard significant parts of the full state-space without contributing a large error, then using this approach can save both memory and time of the computations. In the subsequent sections, we discuss in detail each step of the algorithm.

4.4.1 Relevant Subset

Extracting the subset of states that carries the most of the probability mass (up to ϵ) is the main challenge of this approach. For this, we use a heuristic. Naturally, the set of relevant states depends on the model and the property under consideration. A heuristic that provides the relevant subset should take into account all the information available to it regarding how the specific property can be satisfied in a specific Markov automaton. If, for example, it is known that transitions with higher rates/probabilities are more likely to lead to satisfying the property, then an example of a good heuristic is the one that chooses states that are reachable from the initial state via such transitions. Of course, the same heuristic will perform poorly given a Markov automaton with the opposite property: transitions with higher rates/probabilities lead to states that are less likely to satisfy the property (rare-event case). Thus, information about the model and the property is very important when choosing the relevant subset of states.

Here we propose a heuristic that is well suited for such Markov automata and properties for which transitions with high rates/probabilities are good indicators that the property will be satisfied. Our algorithm is not restricted to this specific heuristic. The correctness of our approach does not depend on the heuristic used, however, its termination requires the heuristic to satisfy certain requirements (see Remark 4.4.2).

At the core of our heuristic is a sampling of random paths of the model based on probabilities of its transitions, similar to what various *simulation* approaches do. A path with higher transition rates/probabilities is more likely to be sampled. Every simulation usually requires very little memory, only what is needed to store the currently simulated path, making it a good candidate to guess the relevant subset of states.

A path is sampled in the following way. Upon entering a Markovian state ms the residence time in this state is sampled from the exponential distribution with parameter $E(ms)$ and then a successor state is sampled randomly from the distribution $[s' \to \mathbf{R}[ms, s']/E(ms) \mid s' \in S]$. For probabilistic states, however, we need

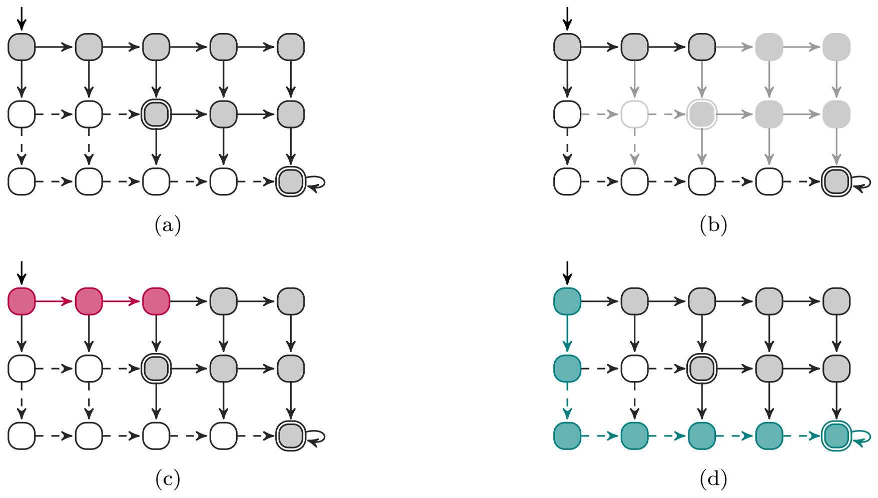


FIGURE 4.3: An example MA is presented in Fig. (4.3a), Fig. (4.3c) and (4.3d) show two sampled paths and Fig. (4.3b) highlights the subset of states constituting the relevant subset.

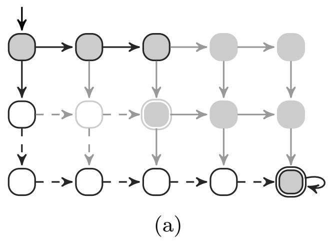
a scheduler due to the presence of non-determinism. We discuss the choice of the scheduler to be used for simulations later in Section 4.4.4. Given this scheduler π, when entering a probabilistic state ps we sample the action to be taken from the distribution $\pi(\rho)$, where ρ is the path that has been observed so far. Next, a successor state is sampled from the distribution $\mathbb{P}[ps, \alpha, \cdot]$, where α is the action picked by the scheduler. The simulation proceeds to the sampled successor state, setting the residence time in ps to 0. The process is repeated from the successor until the property is satisfied or it becomes clear that the path will not be able to satisfy it (for example if the overall time over the path exceeds the largest of the time-bounds b and the property is not satisfied).

Our approach uses a parameter $n_{\text{sim}} \in \mathbb{N}$ which defines the number of paths that is to be sampled. As a relevant subset, we choose those states that appear on at least one of these paths.

Example 4.4.1. *Consider the MA shown in Fig. (4.3a). We omitted rates, probabilities and action labels. Goal states are denoted with double circles. Figures (4.3c) and (4.3d) show two possible sampled paths: The path in Fig. (4.3d) satisfies the property while the path in Fig. (4.3c) is assumed to have timed out before it reaches the target state. The relevant subset of states includes all the states visited during the two simulations. States selected for the relevant subset are highlighted in Fig. (4.3b).*

4.4.2 Partial Markov Automaton

We will now discuss a way to construct a Markov automaton with partial state-space based on the relevant subset S_{rel} computed at step 1. We do this by transforming all the states outside the relevant subset into absorbing. We demonstrate the transformation first on an example.



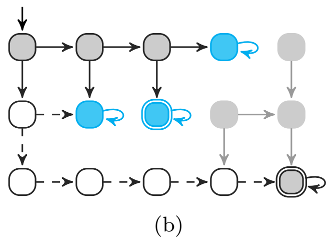


Figure 4.4: Fig. (4.4a) shows the relevant subset S_{rel}. Fig. (4.4b) shows the partial MA $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$. Greyed out states are not part of the state-space.

Example 4.4.2. *Consider Figure 4.4. Here the states of S_{rel} are highlighted in Fig. (4.4a) (this figure is a copy of (4.3b), we repeat it here for convenience). Once again we depict the goal states with double circle. States that are reachable from S_{rel} in one transition, however are not part of S_{rel}, are shown in blue in Fig. (4.4b). These states are made absorbing. The transformation makes some states unreachable, those are the states that are greyed out. Those states do not affect the reachability probability and can therefore be removed from the state-space of the partial MA.*

Formally, for $S_{\text{rel}} \subseteq S$ we construct an MA $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}}) := (\mathcal{M}', AP, lab)$, where $\mathcal{M}' = (S', Act, \dashrightarrow', \mathbf{R}')$, $S' := S_{\text{rel}} \cup post(S_{\text{rel}})$, $S_{\text{fr}} := S' \setminus S_{\text{rel}}$ and $\forall s', s'' \in S'$:

$$\mathbf{R}'[s', s''] := \begin{cases} \mathbf{R}[s', s''] & \text{if } s' \in S_{\text{rel}} \cap MS \\ \mathbf{E}_{\max} & \text{if } s' \in S_{\text{fr}}, s'' = s' \\ 0 & \text{otherwise} \end{cases} \tag{4.85}$$

$$(s', \alpha, \mu) \in \dashrightarrow' \quad \text{iff} \quad s' \in S_{\text{rel}} \text{ and } (s', \alpha, \mu) \in \dashrightarrow$$

The state-space of this MA consists of states that belong to the set S_{rel} and successors of those states. Transitions of states from S_{rel} are preserved. Set S_{fr} contains successors of states from S_{rel}, which are not in S_{rel} themselves (fringe states). These states are made absorbing and thus are Markovian. Notice that some of those states may have been probabilistic before. Since many states became absorbing, some states of the original state-space may have become unreachable and they therefore do not affect the reachability probability value.

4.4.3 Under- and Over-Approximations

We proceed to discussing a way to obtain safe under- and over-approximations of the reachability probability of the original Markov automaton $\mathcal{M}_\ell$ from the partial Markov automaton $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$.

Consider the reachability probability in $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$ with respect to formula $\psi = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$. Let $s \in S_{\text{fr}}$. If $s \not\models \Phi_2$, then there are no paths starting from s that satisfy ψ in $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$, while in $\mathcal{M}_\ell$ such paths may exist. This makes value $\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \psi)$ an under-approximation of $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi)$. However if $s \models \Phi_2$ the situation is different. In this case s is made absorbing and therefore all the paths starting from it will satisfy ψ. To combat this issue, instead of the original formula

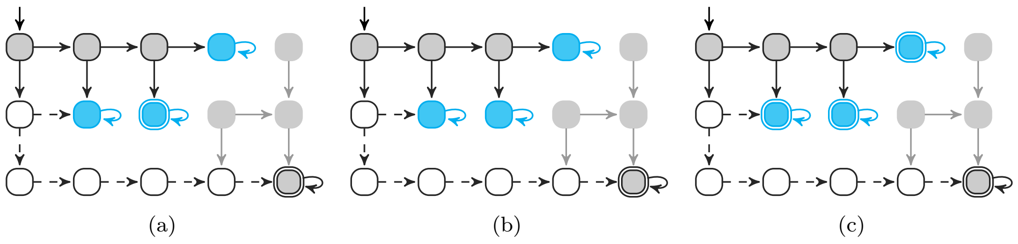


FIGURE 4.5: Updated target states for $\underline{\psi}$ and $\overline{\psi}$ (Fig. (4.5b) and (4.5c) resp.).

ψ we will consider

$$\underline{\psi} = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \, (\Phi_2 \wedge_{s \in S_{\mathrm{fr}}} \neg ap_s)$$

Value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \underline{\psi})$ does not suffer from the issue mentioned above and can therefore be used as an under-approximation of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi)$. We will show below that this value serves as an under-approximation not only for states in S_{fr}, but also for all other states in the state-space of $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$.

We will perform similar alteration of the formula to obtain an over-approximation of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\psi)$. Consider a state $s \in S_{\mathrm{fr}}$, such that $s \models \Phi_2$. In $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ this state is absorbing and therefore satisfies ψ. However problems arise with $s \in S_{\mathrm{fr}}$ in case $s \not\models \Phi_2$. While in the original model $\mathcal{M}$ there may exist paths starting from s that satisfy ψ, in $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ no such paths exist. This means that we cannot use value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \psi)$ as an over-approximation of value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi)$. However if instead of ψ we consider formula

$$\overline{\psi} = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]}(\Phi_2 \vee_{s \in S_{\mathrm{fr}}} ap_s)$$

then all paths starting from s would satisfy $\overline{\psi}$. In the lemma below we will show that the same holds not only for states in S_{fr}, but also for all other states of the state-space of $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$.

Example 4.4.3. *As an example consider Fig. 4.5. Here Fig. (4.5a) is the copy of Fig. (4.4b) repeated here for convenience. Fig. (4.5b) shows target states of the partial MA $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ w. r. t. $\underline{\psi}$ and Fig. (4.5c) shows target states of $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ against $\overline{\psi}$.*

Remark 4.4.1. *Notice that one could further reduce the state-space $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ without altering values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \underline{\psi})$ and $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \overline{\psi})$. Since all states in S_{fr} are essentially sink state w. r. t. $\underline{\psi}$, i. e. there are no paths starting from those states that satisfy $\underline{\psi}$, one could keep only one of those states and discard the rest. Transitions to discarded states need to be redirected to this one sink state. In the same way one can reduce the number of states in S_{fr} w. r. t. $\overline{\psi}$ by keeping only one of them.*

Lemma 4.4.1. *Let $\mathcal{M}$ be a Markov automaton, $s_0 \in S, \psi = \Phi_1 \mathrm{U}^{[\![a,b]\!]} \Phi_2$, $\Phi_2 \Rightarrow \Phi_1$. If $s_0 \in S_{\mathrm{rel}} \cup S_{\mathrm{fr}}$, then*

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s_0, \underline{\psi}) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s_0, \psi) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s_0, \overline{\psi})$$

Proof. Due to Lemma 4.1.1: $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \Phi_1 \mathrm{U}^{[\![a,b]\!]} \Phi_2) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}'_\ell}(s, \mathtt{tt}\, \mathrm{U}^{[\![a,b]\!]} \Phi_2)$, where $\mathcal{M}'_\ell = \mathcal{M}_\ell^{[\neg\Phi_1 \wedge \neg\Phi_2]}$. Therefore w. l. o. g. we assume that $\Phi_1 = \mathtt{tt}$.

Let $I = [\![a, b]\!]$ and $\psi \ominus x = \mathtt{tt}\, \mathrm{U}^{I \ominus x} \Phi_2$. We refine the fixpoint characterisation of value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi)$ given in Lemma 4.1.4 to account for the number of Markovian transitions. We define value $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi, n)$ for $n \in \mathbb{Z}_{\geqslant 0}$ as follows:

For $s \in PS$:

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi, n) = \begin{cases} 1 & \text{if } s \models \Phi_2, 0 \in I \\ \operatorname*{opt}_{\alpha \in Act(s)} \sum_{s' \in S} \mathbb{P}[s, \alpha, s'] \cdot \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi, n) & \text{otherwise} \end{cases}$$

For $s \in MS$:

$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi, n) =$$

$$\begin{cases} \int_0^b E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s' \in S} \mathbb{P}[s, s'] \cdot \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus \tau, n-1)\, \mathrm{d}\tau & \text{if } s \not\models \Phi_2, n > 0 \\ e^{-E(s)\cdot a} + & \text{if } s \models \Phi_2, n \geqslant 0 \\ \quad \mathbb{1}_{\mathbb{Z}_{>0}}(n) \cdot \int_0^a E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s' \in S} \mathbb{P}[s, s'] \cdot \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus \tau, n-1)\, \mathrm{d}\tau & \\ 0 & \text{if } s \not\models \Phi_2, n = 0 \end{cases}$$

It holds that $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi) = \lim_{n\to\infty} \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi, n)$. And analogously for values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \underline{\psi}, n)$ and $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \overline{\psi}, n)$. We will prove the following by induction over n:

$$\forall s \in S_{\mathrm{rel}} \cup S_{\mathrm{fr}}, \forall n \in \mathbb{Z}_{\geqslant 0}, t \in \mathbb{R}_{\geqslant 0} :$$
$$\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s_0, \underline{\psi} \ominus t, n) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s_0, \psi \ominus t, n) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s_0, \overline{\psi} \ominus t, n)$$

First of all, if $I \ominus t = \emptyset$, then $\forall s \in S_{\mathrm{rel}} \cup S_{\mathrm{fr}}, n \in \mathbb{Z}_{\geqslant 0} : \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \underline{\psi} \ominus t, n) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \overline{\psi} \ominus t, n) = 0$.

Consider such values t, that $I \ominus t \neq \emptyset$. Consider $s \in S_{\mathrm{fr}}$. This state is absorbing and non-target in $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ for $\underline{\psi}$. Therefore $\forall n \in \mathbb{Z}_{\geqslant 0}, t \in \mathbb{R}_{\geqslant 0}$: $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \underline{\psi} \ominus t, n) = 0 \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n)$. When analysed against $\overline{\psi}$, state s is absorbing and a target state. Therefore $\forall n \in \mathbb{Z}_{\geqslant 0}, t \in \mathbb{R}_{\geqslant 0} : \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}(s, \overline{\psi} \ominus t, n) = 1 \geqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n)$

Consider $s \in S_{\mathrm{rel}}$. Notice that for such states $s \models \Phi_2 \Leftrightarrow s \models \underline{\Phi_2} \Leftrightarrow s \models \overline{\Phi_2}$, where $\underline{\Phi_2} = (\Phi_2 \wedge_{s \in S_{\mathrm{fr}}} \neg ap_s)$ and $\overline{\Phi_2} = (\Phi_2 \vee_{s \in S_{\mathrm{fr}}} ap_s)$.

Let $s \in S_{\text{rel}} \cap MS$. If $s \not\models \Phi_2, n = 0$, then $\forall t \in \mathbb{R}_{\geqslant 0} : \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi} \ominus t, n) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n) = 0$. If $s \models \Phi_2, n = 0$, then $\forall t \in \mathbb{R}_{\geqslant 0} : \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi} \ominus t, n) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n) = e^{-E(s)\cdot c}$, where c is the lower bound of the interval in formula $\psi \ominus t$.

Consider $s \in PS \cap S_{\text{rel}}, n = 0$. If $s \models \Phi_2, 0 \in I \ominus t$, then $\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi} \ominus t, n) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n) = 1$.

Otherwise the value $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n)$ is the hop-unbounded reachability probability $\text{rea}^{\text{opt}}(s, h)$ where the goal function h is defined over states that satisfy Φ_2 or $s \in MS$ and the value that function h takes for state s' is $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s', \psi \ominus t, n)$. If $s \not\models \Phi_2$ and $0 \in I \ominus t$ then h is defined over all Markovian states and those that satisfy Φ_2. If $s \not\models \Phi_2$ and $0 \notin I \ominus t$ then h is defined only over Markovian states. If $s \models \Phi_2$ and $0 \notin I \ominus t$ then h is defined only over Markovian states. And analogously for $\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n), \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi} \ominus t, n)$ with the respective goal functions $\overline{h}$ and $\underline{h}$. Let s' be a state in the domain of $\underline{h}$. Then we have shown above that

$$\underline{h}(s') = \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s', \underline{\psi} \ominus t, n) \leqslant \text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s', \psi \ominus t, n) \leqslant \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s', \overline{\psi} \ominus t, n) = \overline{h}$$

Let $\underline{\pi}_{\text{opt}}$ be a stationary scheduler that achieves optimum for $\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi} \ominus t, n)$, $\overline{\pi}_{\text{opt}}$ and $\pi_{\texttt{opt}}$ are the same for $\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n)$ and $\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n)$ resp. Then for opt = sup:

$$\begin{aligned}
\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi} \ominus t, n) &= \text{rea}^{\underline{\pi}_{\text{opt}}}(s, \underline{h}) \overset{(4.18)}{=} \sum_{s' \in \underline{h}} \text{rea}^{\underline{\pi}_{\text{opt}}}(s, s') \cdot \underline{h}(s') \\
&= \sum_{s' \in \underline{h}, s' \in h} \text{rea}^{\underline{\pi}_{\text{opt}}}(s, s') \cdot \underline{h}(s') + \sum_{s' \in \underline{h}, s' \notin h} \text{rea}^{\underline{\pi}_{\text{opt}}}(s, s') \cdot \underbrace{\underline{h}(s')}_{=0} \\
&\leqslant \sum_{s' \in \underline{h}, s' \in h} \text{rea}^{\underline{\pi}_{\text{opt}}}(s, s') \cdot h(s') + \sum_{s' \in \underline{h}, s' \notin h} \text{rea}^{\underline{\pi}_{\text{opt}}}(s, s') \cdot \text{val}_{\underline{\pi}_{\text{opt}}}^{\mathcal{M}_\ell}(s', \psi \ominus t, n) \\
&= \text{rea}^{\underline{\pi}_{\text{opt}}}(s, h) \leqslant \text{rea}^{\text{opt}}(s, h) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n)
\end{aligned}$$

Similarly for h and $\overline{h}$:

$$\begin{aligned}
\text{val}_{\text{opt}}^{\mathcal{M}_\ell}(s, \psi \ominus t, n) &= \text{rea}^{\pi_{\texttt{opt}}}(s, h) \overset{(4.18)}{=} \sum_{s' \in h} \text{rea}^{\pi_{\texttt{opt}}}(s, s') \cdot h(s') \\
&= \sum_{s' \in h, s' \in \overline{h}} \text{rea}^{\pi_{\texttt{opt}}}(s, s') \cdot h(s') + \sum_{s' \notin h, s' \in \overline{h}} \text{rea}^{\pi_{\texttt{opt}}}(s, s') \cdot \text{val}_{\pi_{\texttt{opt}}}^{\mathcal{M}_\ell}(s', \psi \ominus t, n) \\
&\leqslant \sum_{s' \in h, s' \in \overline{h}} \text{rea}^{\pi_{\texttt{opt}}}(s, s') \cdot \overline{h}(s') + \sum_{s' \notin h, s' \in \overline{h}} \text{rea}^{\pi_{\texttt{opt}}}(s, s') \cdot \underbrace{\overline{h}(s')}_{=1} \\
&= \text{rea}^{\pi_{\texttt{opt}}}(s, \overline{h}) \leqslant \text{rea}^{\text{opt}}(s, \overline{h}) = \text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n)
\end{aligned}$$

Next we prove the same for opt = inf:

$$\text{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi} \ominus t, n) = \text{rea}^{\overline{\pi}_{\text{opt}}}(s, \overline{h}) \overset{(4.18)}{=} \sum_{s' \in \overline{h}} \text{rea}^{\overline{\pi}_{\text{opt}}}(s, s') \cdot \overline{h}(s')$$

$$
\begin{aligned}
&= \sum_{s'\in h, s'\in \overline{h}} \mathrm{rea}^{\overline{\pi}_{\mathrm{opt}}}(s,s')\cdot \overline{h}(s') + \sum_{s'\notin h, s'\in \overline{h}} \mathrm{rea}^{\overline{\pi}_{\mathrm{opt}}}(s,s')\cdot \underbrace{\overline{h}(s')}_{=1} \\
&\geqslant \sum_{s'\in h, s'\in \overline{h}} \mathrm{rea}^{\overline{\pi}_{\mathrm{opt}}}(s,s')\cdot h(s') + \sum_{s'\notin h, s'\in \overline{h}} \mathrm{rea}^{\overline{\pi}_{\mathrm{opt}}}(s,s')\cdot \mathrm{val}^{\mathcal{M}_\ell}_{\overline{\pi}_{\mathrm{opt}}}(s',\psi\ominus t,n) \\
&= \mathrm{rea}^{\overline{\pi}_{\mathrm{opt}}}(s,h) \geqslant \mathrm{rea}^{\mathrm{opt}}(s,h) = \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi\ominus t,n)
\end{aligned}
$$

And similarly in the other direction:

$$
\begin{aligned}
&\mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s,\psi\ominus t,n) = \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,h) \overset{(4.18)}{=} \sum_{s'\in h} \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,s')\cdot h(s') \\
&= \sum_{s'\in h, s'\in \underline{h}} \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,s')\cdot h(s') + \sum_{s'\notin h, s'\in \underline{h}} \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,s')\cdot \mathrm{val}^{\mathcal{M}_\ell}_{\pi_{\mathrm{opt}}}(s',\psi\ominus t,n) \\
&\geqslant \sum_{s'\in h, s'\in \underline{h}} \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,s')\cdot \underline{h}(s') + \sum_{s'\notin h, s'\in \overline{h}} \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,s')\cdot \underbrace{\underline{h}(s')}_{=0} \\
&= \mathrm{rea}^{\pi_{\mathrm{opt}}}(s,\underline{h}) \geqslant \mathrm{rea}^{\mathrm{opt}}(s,\underline{h}) = \mathrm{val}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}_{\mathrm{opt}}(s,\underline{\psi}\ominus t,n)
\end{aligned}
$$

For $n > 0$ and $s \in PS \cap S_{\mathrm{rel}}$ derivations above still apply, taking into account the induction hypothesis. It is left to prove the claim for $n > 0, s \in MS$, that in turn follows from the induction hypothesis and monotonicity of the integral.

□

In order to compute, or approximate, the values $\mathrm{val}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}_{\mathrm{opt}}(s_0,\underline{\psi})$ and $\mathrm{val}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}_{\mathrm{opt}}(s_0,\overline{\psi})$ any algorithm for reachability analysis on Markov automata can be used. For example, the one that has been presented in Section 4.3 of this chapter, or those discussed in Chapter 4, at the end of Section 4.2.

4.4.4 The Choice of Simulating Scheduler

Here we discuss how to chose a scheduler to perform sampling of paths in an MA. For a measurable scheduler π on $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$ we define its extension π^x on $\mathcal{M}_\ell$ as follows:

$$
\pi^x(\rho) := \begin{cases} \pi(\rho) & \text{if } \rho \in \mathit{Paths}^*(\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})) \text{ and } \rho{\downarrow} \in S_{\mathrm{rel}} \\ \text{any distribution over } Act(\rho{\downarrow}) & \text{otherwise} \end{cases} \tag{4.86}
$$

We additionally require that this extension is also measurable. This scheduler takes the same decisions as π on paths of $\mathcal{M}_\ell$ that are possible in $\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})$, and otherwise samples an action from some distribution.

We propose two options for the simulating scheduler: (i) the uniform scheduler π_{uni}, which picks one of the enabled actions uniformly at random, and (ii) scheduler $\overline{\pi}_{\mathrm{opt}}$ that optimises $\mathrm{val}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}_{\mathrm{opt}}(\overline{\psi})$ for opt = sup or, conversely, scheduler $\underline{\pi}_{\mathrm{opt}}$ that optimises $\mathrm{val}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_{\mathrm{rel}})}_{\mathrm{opt}}(\underline{\psi})$ for opt = inf. For the latter option we use the respective extensions of $\underline{\pi}_{\mathrm{opt}}$ and $\overline{\pi}_{\mathrm{opt}}$ to $\mathcal{M}_\ell$, i. e. $\underline{\pi}^x_{\mathrm{opt}}$ and $\overline{\pi}^x_{\mathrm{opt}}$.

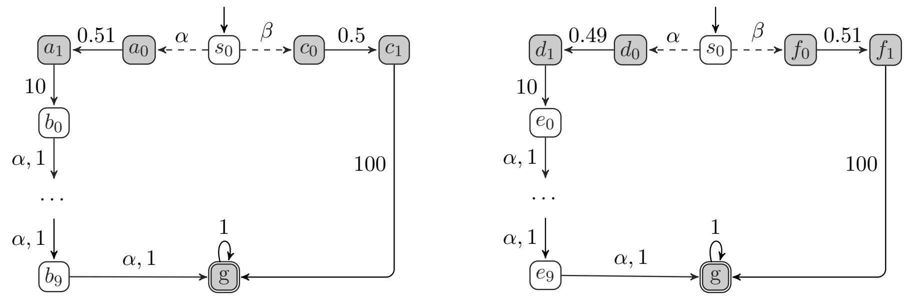


Figure 4.6

The uniform scheduler has the advantage of being simple to implement and lightweight in terms of memory requirements. Moreover, in some cases this may be the only option, if, for example, the underlying algorithm for time-bounded reachability analysis does not output an optimal scheduler. There are situations in which the uniform scheduler is a good choice. For example, if different actions lead to Markovian states with exit rates that differ significantly, e. g. by orders of magnitude. In this case, whenever the simulation reaches a state with low exit rate, the residence time in such a state will likely be high. As a result, fewer states will be sampled within the same amount of time, compared to a path that visits Markovian states with high exit rates. If the formula is likely to be satisfied over paths with higher exit rates of Markovian states, then the price for choosing a wrong action (in terms of excess state-space) will not be high.

On the other hand, the uniform scheduler defines a positive probability of choosing each action, even the one that is clearly suboptimal. Given enough time, the algorithm that uses this scheduler for obtaining the relevant subset will eventually explore the entire MA. This is not the case for $\overline{\pi}^x_{\text{opt}}$ and $\underline{\pi}^x_{\text{opt}}$. These schedulers are optimal on the partial model in a best-case scenario arising when all states outside of S_{rel} are absorbing target or sink states. This means that the schedulers will most likely navigate towards the borders of the partial model as long as states at the border look promising. If it becomes clear that some of those states, say reachable via action α of a probabilistic state ps, have a worse probability of satisfying the property than states reachable via action β of ps, than action α will not be chosen by $\overline{\pi}^x_{\text{opt}}/\underline{\pi}^x_{\text{opt}}$. Thus states that are only reachable via suboptimal actions will not be part of the relevant subset under these schedulers.

The following example illustrates that neither (i) nor (ii) is a strictly better option for simulations.

Example 4.4.4. *Consider the MA on the left of Figure 4.6 and the time interval* $[0, 1]$. *We will show that there may be scenarios when sampling with the uniform scheduler produces a set* S_{rel} *of a smaller size than that induced by scheduler* $\overline{\pi}^x_{\text{opt}}$.

If $S_{\text{rel}} = \{s_0, a_0, c_0\}$, *then action* α *delivers higher values* $\text{val}^{\mathcal{M}^{\text{sub}(S_{\text{rel}})}_{\ell}}_{\text{opt}}(\overline{\psi})$ *than action* β *and therefore* $\overline{\pi}^x_{\text{opt}}(s_0, 0) = \alpha$. *If at the next iteration during one of the simulations the system leaves state* a_1, *then all the states* $b_0 - b_9$ *will be visited at that same time and added to* S_{rel}. *At this moment action* β *becomes optimal. The algorithm converges when at least one of the simulations reaches the goal state via* *action β, at which moment $S_{\text{rel}} = \{s_0, a_0, a_1, b_0, \ldots, b_9, c_0, c_1, g\}$.*

If the uniform scheduler is used instead of $\overline{\pi}^x_{\text{opt}}$, then with probability 0.5 it will select action β and with probability 0.5 it selects action α in state s_0. If simulations never leave state a_1, however do leave state c_1, then $S_{\text{rel}} = \{s_0, a_0, a_1, c_0, c_1, g\}$ and action β becomes optimal for both lower- and upper-bound formulas. At this moment the algorithm has converged.

The MA on the right of Figure 4.6 illustrates that there are scenarios where scheduler $\overline{\pi}^x_{\text{opt}}$ performs better than the uniform scheduler. Here when $\{s_0, d_0, f_0\} \subseteq S_{\text{rel}}$ it becomes clear that action α is sub-optimal. From this moment on only action β is chosen for simulations, which means that states of the left chain that have not been visited yet, will never be visited. However, the uniform scheduler may keep choosing action α and adding unnecessary states from the left chain to S_{rel}. At the moment when $\{s_0, d_0, f_0, f_1, g\} \subseteq S_{\text{rel}}$: $\text{val}^{\mathcal{M}^{\text{sub}}_\ell(S_{\text{rel}})}_{\text{opt}}(\overline{\psi}) = \text{val}^{\mathcal{M}^{\text{sub}}_\ell(S_{\text{rel}})}_{\text{opt}}(\underline{\psi})$ and the algorithm converges.

4.4.5 The Algorithm

An algorithm that approximates value $\text{val}^{\mathcal{M}_\ell}_{\text{opt}}(s_0, \psi)$ for an arbitrary MA $\mathcal{M}_\ell$, $s_0 \in S, \psi = \Phi_1 \operatorname{U}^{[\![a,b]\!]} \Phi_2$ up to $\epsilon \in (0,1)$ is shown in Algorithm 4. It follows closely the steps outlined in Section 4.4. Parameter w affects precision used for the reachability analysis on the partial MA and parameter *sch* defines the scheduler used for obtaining the relevant subset. Notice that even if *sch* = opt, no optimal scheduler is available for the very first iteration and therefore only the uniform scheduler can be used in this case.

Here `ComputeReachability`$(\mathcal{M}'_\ell, s'_0, \psi', \text{opt}', \epsilon')$ denotes a procedure that for $\text{opt}' = \sup$ under-approximates and for $\text{opt}' = \inf$ over-approximates the value $\text{val}^{\mathcal{M}'_\ell}_{\text{opt}'}(\psi')$ up to ϵ'. Its output is an ϵ'-close approximation of value $\text{val}^{\mathcal{M}'_\ell}_{\text{opt}'}(s'_0, \psi')$ and an ϵ'-optimal scheduler π for $\text{val}^{\mathcal{M}'_\ell}_{\text{opt}'}(\psi')$.

Procedure `GetRelevantSubset`$(\mathcal{M}, s_0, \psi, \pi_{\text{sim}})$ shown in Algorithm 5 samples n_{sim} random paths of $\mathcal{M}$ starting from s_0 using scheduler π_{sim}. Simulation of a path stops at the moment when the time runs out ($t > b$) or it becomes clear that any infinite extension of the currently simulated path ρ will satisfy formula ψ. Below we show the conditions under which the latter is fulfilled:

$$\begin{aligned}
&\exists t \in [\![a,b]\!], n \leqslant |\rho|, s = \rho[n], s \in \rho@t, s \models_{\mathcal{M}_\ell} \Phi_2, \\
&\forall \tau \in [0,t), s' \in \rho@\tau : s' \models_{\mathcal{M}_\ell} \Phi_1 \text{ and} \\
&\forall k = 0..n-1 : \rho[k] \models \Phi_1
\end{aligned} \tag{4.87}$$

Condition (4.87) is essentially the semantics of the until formula presented in Chapter 3, Section 3.1, tuned for finite paths. The only difference with respect to infinite paths is that we require that $n \leqslant |\rho|$. If (4.87) is satisfied, then any infinite extension of ρ satisfies ψ, or formally: $\mathit{Cyl}(\rho) \models \psi$.

Algorithm 4 `PartialTBR`

Input: MA $\mathcal{M}_\ell = (\mathcal{M}, AP, lab)$, where $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$, initial state $s_0 \in S$, path formula $\psi = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$, opt $\in \{\sup, \inf\}$, precision $\epsilon \in (0,1)$
Output: $\underline{v}, \overline{v} \in [0,1]$, such that $\underline{v} \leqslant \mathrm{val}^{\mathcal{M}_\ell}_{\mathrm{opt}}(s_0, \psi) \leqslant \overline{v}$ and $\overline{v} - \underline{v} \leqslant \epsilon$ and a scheduler $\pi_{\mathtt{opt}}$ on $\mathcal{M}_\ell$ such that $\underline{v} \leqslant \mathrm{val}^{\mathcal{M}_\ell}_{\pi_{\mathtt{opt}}}(s_0, \psi) \leqslant \overline{v}$
Parameters: $w \in (0, 1/3), sch \in \{\mathrm{uni}, \mathrm{opt}\}$, by default $w = 0.1, sch = \mathrm{opt}$

```
 1: $\underline{v} = 0, \overline{v} = 1$
 2: $\pi_{\mathrm{sim}} = \pi_{\mathrm{uni}}$
 3: $S_{\mathrm{rel}} = \{s_0\}$
 4: while $\overline{v} - \underline{v} > \epsilon$ do
 5:     $S_{\mathrm{rel}} = S_{\mathrm{rel}} \cup$ GetRelevantSubset$(\mathcal{M}_\ell, s_0, \psi, \pi_{\mathrm{sim}})$        ▷ see Algorithm 5
 6:     $S_{\mathrm{fr}} = (S_{\mathrm{rel}} \cup post(S_{\mathrm{rel}})) \setminus S_{\mathrm{rel}}$
 7:     Build $\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}})$ as discussed in Section 4.4.2
 8:     Set $\overline{\psi} = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]}(\Phi_2 \vee_{s \in S_{\mathrm{fr}}} ap_s)$
        Set $\underline{\psi} = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} (\Phi_2 \wedge_{s \in S_{\mathrm{fr}}} \neg ap_s)$
 9:     $\overline{v}, \overline{\pi}_{\mathrm{opt}}$ = ComputeReachability$(\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}}), s_0, \overline{\psi}, \mathrm{opt}, w \cdot \epsilon)$,
        $\underline{v}, \underline{\pi}_{\mathrm{opt}}$ = ComputeReachability$(\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}}), s_0, \underline{\psi}, \mathrm{opt}, w \cdot \epsilon)$
10:     if (opt = sup) then $\overline{v} = \overline{v} + w \cdot \epsilon$ else $\underline{v} = \underline{v} - w \cdot \epsilon$
11:     if ($sch$ == uni) then
12:         $\pi_{\mathrm{sim}} = \pi_{\mathrm{uni}}$
13:     else
14:         if (opt = sup) then $\pi_{\mathrm{sim}} = \overline{\pi}^x_{\mathrm{opt}}$ else $\pi_{\mathrm{sim}} = \underline{\pi}^x_{\mathrm{opt}}$
15: if (opt = sup) then $\pi_{\mathtt{opt}} = \underline{\pi}^x_{\mathrm{opt}}$ else $\pi_{\mathtt{opt}} = \overline{\pi}^x_{\mathrm{opt}}$
16: return $\underline{v}, \overline{v}, \pi_{\mathtt{opt}}$
```

Lemma 4.4.2. *Let $\mathcal{M}_\ell = (\mathcal{M}, AP, lab)$, $s_0 \in S$, $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$, $\psi = \Phi_1 \, \mathrm{U}^{[\![a,b]\!]} \Phi_2$, $\Phi_2 \Rightarrow \Phi_1$ and $\epsilon \in (0,1)$. If procedure* `PartialTBR` $(\mathcal{M}_\ell, s_0, psi, \epsilon)$ *terminates and returns values $\underline{v}, \overline{v}, \pi_{\mathtt{opt}}$, then they satisfy the following:*

$$\overline{v} - \underline{v} \leqslant \epsilon \tag{4.88}$$

$$\underline{v} \leqslant \mathrm{val}^{\mathcal{M}}_{\mathrm{opt}}(s_0, \psi) \leqslant \overline{v} \tag{4.89}$$

$$\underline{v} \leqslant \mathrm{val}^{\mathcal{M}_\ell}_{\pi_{\mathtt{opt}}}(s_0, \psi) \leqslant \overline{v} \tag{4.90}$$

Proof. The procedure returns a value only at line 16, which is reachable only if $\overline{v} - \underline{v} \leqslant \epsilon$, what proves (4.88). Let $\overline{u} = \mathtt{ComputeReachability}(\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}}), s_0, \overline{\psi}, \mathrm{opt}, w \cdot \epsilon)$ and $\underline{u} = \mathtt{ComputeReachability}(\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}}), s_0, \underline{\psi}, \mathrm{opt}, w \cdot \epsilon)$, where $S_{\mathrm{rel}}, \underline{\psi}$ and $\overline{\psi}$ are values of the respective variables of the algorithm computed at the very last iteration of the while-loop. By definition values $\underline{v}$ and $\overline{v}$ satisfy the following:

$$\begin{aligned} \text{If opt} = \sup: \overline{v} = \overline{u} + w \cdot \epsilon \geqslant \mathrm{val}^{\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}})}_{\mathrm{opt}}(s_0, \overline{\psi}) \geqslant \overline{u} \\ \underline{v} = \underline{u} \leqslant \mathrm{val}^{\mathcal{M}^{\mathrm{sub}}_\ell(S_{\mathrm{rel}})}_{\mathrm{opt}}(s_0, \underline{\psi}) \leqslant \underline{u} + w \cdot \epsilon \end{aligned} \tag{4.91}$$

Algorithm 5 GetRelevantSubset

Input: MA $\mathcal{M}_\ell = (\mathcal{M}, AP, lab)$, where $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$, initial state $s_0 \in S$, path formula $\psi = \Phi_1 \operatorname{U}^{[\![a,b]\!]} \Phi_2$, a scheduler π_{sim}
Output: $S' \subseteq S$
Parameters: $n_{\text{sim}} \in \mathbb{N}$

```
1:  $S' = \emptyset$
2:  for ($i = 0$; $i < n_{\text{sim}}$; $i = i + 1$) do
3:      $\rho = s_0$, $t = 0$
4:      while ($t \leqslant b$ and (4.87) is not satisfied) do
5:          $s = \rho{\downarrow}$
6:          if $s \in MS_{\mathcal{M}_\ell}$ then
7:              Sample $t'$ from exponential distribution with parameter $E(s)$
8:              Set $\nu = E(s), \mu = [s' \to \mathbf{R}[s, s']/E(s) \mid s' \in S]$
9:          else
10:             Set $t' = 0$
11:             Sample action $\alpha$ from distribution $\pi_{\text{sim}}(\rho)$
12:             Set $\nu = \alpha, \mu = \mathbb{P}[s, \alpha, \cdot]$
13:         Sample a successor $s'$ of $s$ from distribution $\mu$
14:         $\rho = \rho \xrightarrow{\nu, t'} s'$
15:         $t = t + t'$
16:     add all states of $\rho$ to $S'$
```

$$\begin{aligned} \text{If opt} = \inf : & \overline{u} - w \cdot \epsilon \leqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \overline{\psi}) \leqslant \overline{u} = \overline{v} \\ & \underline{u} \geqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \underline{\psi}) \geqslant \underline{u} - w \cdot \epsilon = \underline{v} \end{aligned} \tag{4.92}$$

The proof of (4.89) follows from Lemma 4.4.1 and (4.91-4.92):

$$\underline{v} \leqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \underline{\psi}) \leqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell}(s_0, \psi) \leqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \overline{\psi}) \leqslant \overline{v} \tag{4.93}$$

Next we prove (4.90). We achieve this by showing that

$$\operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \underline{\psi}) \leqslant \operatorname{val}_{\pi_{\text{opt}}}^{\mathcal{M}_\ell}(s_0, \psi) \leqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \overline{\psi})$$

One of the inequalities follows from Lemma 4.4.1:

$$\begin{aligned} \text{If opt} = \sup : \operatorname{val}_{\pi_{\text{opt}}}^{\mathcal{M}_\ell}(s_0, \psi) \leqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell}(s_0, \psi) &\overset{\text{Lemma 4.4.1}}{\leqslant} \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \overline{\psi}) \\ \text{If opt} = \inf : \operatorname{val}_{\pi_{\text{opt}}}^{\mathcal{M}_\ell}(s_0, \psi) \geqslant \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell}(s_0, \psi) &\overset{\text{Lemma 4.4.1}}{\geqslant} \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \underline{\psi}) \end{aligned}$$

We will now prove the other inequality:

$$\begin{aligned} \forall s \in S : \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \underline{\psi}) \leqslant \operatorname{val}_{\pi_{\text{opt}}}^{\mathcal{M}_\ell}(s, \psi) \quad & \text{if opt} = \sup \\ \operatorname{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s, \overline{\psi}) \geqslant \operatorname{val}_{\pi_{\text{opt}}}^{\mathcal{M}_\ell}(s, \psi) \quad & \text{if opt} = \inf \end{aligned}$$

We consider the case of opt = sup. The case opt = inf is analogous. For opt = sup scheduler $\pi_{\texttt{opt}}$ is an extension of $\underline{\pi}_{\text{opt}}$ to $\mathcal{M}_\ell$. Consider a finite path $\rho \in \mathit{Paths}^*(\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}}))$, such that it satisfies (4.87) for $\underline{\psi}$ and $\rho\!\downarrow\, \models (\Phi_2 \wedge_{s \in S_{\text{fr}}} \neg ap_s)$. Let A^* be the set of all such paths. Since states from the set S_{fr} are not target states w. r. t. $\underline{\psi}$, then $\{\rho \in \mathit{Paths}^\omega(\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}}) \mid \rho \models \underline{\psi})\} = \mathit{Cyl}_{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(A^*)$

Notice that every path $\rho \in A^*$ is also a finite path in $\mathcal{M}_\ell$, because finite paths that are not shared between $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$ and $\mathcal{M}_\ell$ are only those that end in a fringe state S_{fr}. Moreover $\mathit{Cyl}_{\mathcal{M}_\ell}(A^*) \models \psi$. However there may exist paths in $\mathcal{M}_\ell$ that satisfy ψ, but do not satisfy $\underline{\psi}$ in $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$, or are not valid paths in $\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})$ at all. Let B^* be the set of finite paths ρ, such that any of its infinite extensions in $\mathcal{M}_\ell$ satisfies formula ψ : $\mathit{Cyl}_{\mathcal{M}_\ell}(\rho) \models \psi$ and $\rho\!\downarrow\, \models \Phi_2$. Then $A^* \subseteq B^*$, or $B^* = A^* \uplus C$, where C is possibly empty. By definition scheduler $\pi_{\texttt{opt}}$ takes the same decisions along paths in set A^* as $\underline{\pi}_{\text{opt}}$. Therefore

$$\begin{aligned}\mathrm{val}_{\pi_{\texttt{opt}}}^{\mathcal{M}_\ell}(s_0, \psi) &= \mathrm{Pr}_{\pi_{\texttt{opt}}, s_0}^{\mathcal{M}_\ell}\left[\mathit{Cyl}_{\mathcal{M}_\ell}(A^*)\right] + \mathrm{Pr}_{\pi_{\texttt{opt}}, s_0}^{\mathcal{M}_\ell}\left[C\right] \\ &\geqslant \mathrm{Pr}_{\pi_{\texttt{opt}}, s_0}^{\mathcal{M}_\ell}\left[\mathit{Cyl}_{\mathcal{M}_\ell}(A^*)\right] = \mathrm{Pr}_{\underline{\pi}_{\text{opt}}, s_0}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}\left[\mathit{Cyl}_{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(A^*)\right] \\ &= \mathrm{val}_{\text{opt}}^{\mathcal{M}_\ell^{\text{sub}}(S_{\text{rel}})}(s_0, \underline{\psi})\end{aligned}$$

□

Theorem 4.4.3. *If $\Phi_2 \Rightarrow \Phi_1$, then Algorithm 4 terminates almost surely.*

Proof. First we will show that procedure `GetRelevantSubset` terminates almost surely. For this we only need to prove that its while-loop terminates almost surely. Consider infinite paths produced by this while-loop that never satisfy its exit condition, i. e.

$$\begin{aligned}A = \{\rho \in \mathit{Paths}^\omega \mid\ &\rho \text{ is generated within lines 3-15 of Algorithm 5,} \\ &\tau_{\text{total}}(\rho) \leqslant b \text{ and (4.87) is not satisfied}\}\end{aligned}$$

Notice that path sampling in Algorithm 5 follows the probability measure for MA. Or formally, let X be a random variable that takes as value the infinite path produced by Algorithm 5, when its while-loop is set to run to infinity (i. e. its exit condition is ignored). Then

$$\mathrm{Pr}[X \in A] = \mathrm{Pr}_{\pi_{\text{sim}}, s_0}^{\mathcal{M}}[A]$$

Due to assumptions introduced in the very beginning of this chapter, $\mathcal{M}$ is non-Zeno and therefore

$$\begin{aligned}&\forall \pi \in \Pi_\mu^{\mathcal{M}}, s \in S : \\ &\quad \mathrm{Pr}_{\pi, s}^{\mathcal{M}}\left[\{\rho \in \mathit{Paths}^\omega(\mathcal{M}) \mid \exists t \in \mathbb{R}_{\geqslant 0} : \forall i \in \mathbb{Z}_{\geqslant 0} : \tau_{\text{total}}(\rho, i) \leqslant t\}\right] = 0\end{aligned}$$

Since $A \subseteq \{\rho \in \mathit{Paths}^\omega(\mathcal{M}) \mid \exists t \in \mathbb{R}_{\geqslant 0} : \forall i \in \mathbb{Z}_{\geqslant 0} : \tau_{\text{total}}(\rho, i) \leqslant t\}$ it follows that $\mathrm{Pr}[X \in A] = 0$. Thus with probability 1 the while-loop of Algorithm 5 samples a path that satisfies the exit condition of the while loop, what proves that Algorithm 5 terminates almost surely.

Consider the sequence $w = \{S_n\}_{n \in \mathbb{Z}_{\geqslant 0}}$ of sets, such that S_i is the value that variable S_{rel} of Algorithm 4 takes at the end of iteration $n \in \mathbb{Z}_{>0}$. For $n = 0 : S_0 = \{s_0\}$. We ignore for the moment the termination condition of the while-loop thus letting it run to infinity. This way sequence $\{S_n\}_{n \in \mathbb{Z}_{\geqslant 0}}$ is infinite. It holds that $\forall n \in \mathbb{Z}_{\geqslant 0} : S_n \subseteq S$ and the sequence is non-decreasing, i. e. $\forall n_1 < n_2 : S_{n_1} \subseteq S_{n_2}$. Since the state-space is finite, this implies that every run of the algorithm generates a sequence $\{S_n\}_{n \in \mathbb{Z}_{\geqslant 0}}$ that is converging to some set S_∞ and $\exists k \in \mathbb{Z}_{\geqslant 0}$, s. t. $\forall n \geqslant k : S_n = S_\infty$. Let W be the set of sequences $w = \{S_n\}_{n \in \mathbb{Z}_{\geqslant 0}}$ We define $w{\downarrow} = S_\infty$ to be the limit of sequence $w \in W$.

We will first prove that

$$\begin{aligned}
&\mathrm{opt} = \sup : \\
&\Pr\Big[w \in W \mid \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\underline{\psi}) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\overline{\psi}) \leqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\underline{\psi}) + w \cdot \epsilon\Big] = 1 \\
&\mathrm{opt} = \inf : \\
&\Pr\Big[w \in W \mid \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\overline{\psi}) \geqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\underline{\psi}) \geqslant \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\overline{\psi}) - w \cdot \epsilon\Big] = 1
\end{aligned} \tag{4.94}$$

Let S' be the set of states reachable from s_0. If for $w \in W : w{\downarrow} = S'$, then $S_{\mathrm{fr}} = \emptyset$ and therefore $\overline{\psi} = \underline{\psi} = \psi$, Markovian and probabilistic transitions over S' in $\mathcal{M}$ and $\mathcal{M}_\ell^{\mathrm{sub}}(w{\downarrow})$ coincide and therefore $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(s_0, \overline{\psi}) = \mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(s_0, \underline{\psi})$.

If there are no probabilistic states, then the probability that the Algorithm generates an w, such that $w{\downarrow} \neq S'$ is 0. This is due to the fact that the probability of never sampling a state that is reachable with non-zero probability is 0. The same holds if scheduler π_{sim} selects each action of every probabilistic state with non-zero probability (for example $\pi_{\mathrm{sim}} = \pi_{\mathrm{uni}}$).

Due to the discussion in the previous paragraph it is left to prove that (4.94) holds for *sch* = opt. A run of Algorithm 4 for *sch* = opt produces a sequence of schedulers π_{sim}^n and state-spaces $w = \{S_n\}_{n \in \mathbb{Z}_{\geqslant 0}}$. Here $\pi_{\mathrm{sim}}^0 = \pi_{\mathrm{uni}}$ and $\forall n > 0 : \pi_{\mathrm{sim}}^n$ is an extension in the sense of (4.86) to $\mathcal{M}_\ell$ of an $(w \cdot \epsilon)$-optimal scheduler for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(S_n)}(\psi')$, where $\psi' \in \{\overline{\psi}, \underline{\psi}\}$. State-space S_n is obtained by using π_{sim}^{n-1} for simulations. As discussed before, the sequence of S_n is converging to some $w{\downarrow} = S_\infty$. Therefore the sequence of schedulers π_{sim}^n is also converging to a scheduler $\pi_{\mathrm{sim}}^\infty$, such that it is $(w \cdot \epsilon)$-optimal scheduler for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\psi')$[4]. It also follows that with probability 1 the set $w{\downarrow}$ comprises of all the states that are reachable by $\pi_{\mathrm{sim}}^\infty$ with non-zero probability. Let S_{fr}^w be fringe states relative to $w{\downarrow}$, i. e. $S_{\mathrm{fr}}^w = (w{\downarrow} \cup \mathit{post}(w{\downarrow})) \setminus w{\downarrow}$. For any scheduler π value $\mathrm{val}_\pi^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\underline{\psi})$ can only differ from $\mathrm{val}_\pi^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\overline{\psi})$ if a state in S_{fr}^w is reachable with non-zero probability by π. For $\pi_{\mathrm{sim}}^\infty$ states in S_{fr}^w are only reachable with probability 0. Therefore the reachability values

[4]There may be multiple $(w \cdot \epsilon)$-optimal schedulers for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell^{\mathrm{sub}}(w\downarrow)}(\psi')$ and choosing a different one at each iteration may lead to the sequence π_{sim}^n being non-converging. We however assume w. l. o. g. that only one of those schedulers is always selected by procedure `ComputeReachability`.

of states in S^w_{fr} do not affect values $\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi^{\infty}_{\text{sim}}}(\underline{\psi})$ and $\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi^{\infty}_{\text{sim}}}(\overline{\psi})$. Therefore

$$\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi_{\texttt{opt}}}(\underline{\psi}) = \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi_{\texttt{opt}}}(\overline{\psi}) \tag{4.95}$$

Since $\pi_{\texttt{opt}}$ is $(w \cdot \epsilon)$-optimal, for opt $=$ sup:

$$\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi_{\texttt{opt}}}(\overline{\psi}) \leqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\overline{\psi}) \leqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi_{\texttt{opt}}}(\overline{\psi}) + w \cdot \epsilon$$

Since $\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\pi_{\texttt{opt}}}(\underline{\psi}) \leqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\underline{\psi})$ and due to Lemma 4.4.1 $\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\underline{\psi}) \leqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\overline{\psi})$, then

$$\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\underline{\psi}) \leqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\overline{\psi}) \leqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\underline{\psi}) + w \cdot \epsilon \tag{4.96}$$

And analogously for opt $=$ inf:

$$\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\overline{\psi}) \geqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\underline{\psi}) \geqslant \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(w\downarrow)}_{\text{opt}}(\overline{\psi}) - w \cdot \epsilon \tag{4.97}$$

Next we will show that given (4.94) the claim of the theorem follows. Let w be a sequence generated by the algorithm, such that for opt $=$ sup (4.96) holds and for opt $=$ inf (4.97) holds. Let $\underline{v}_n, \overline{v}_n$ be the values that variables $\underline{v}, \overline{v}$ of Algorithm 4 take at the end of iteration n respectively. We can therefore use the following representation $\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(S_n)}_{\text{opt}}(s_0, \overline{\psi}) = \overline{v}_n - \epsilon'_n$, $\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(S_n)}_{\text{opt}}(s_0, \underline{\psi}) = \underline{v}_n + \epsilon''_n$, where $0 \leqslant \varepsilon'_n, \varepsilon''_n \leqslant w \cdot \epsilon$. Then:

$$\begin{aligned}
\lim_{n\to\infty}(\overline{v}_n - \underline{v}_n) &= \lim_{n\to\infty}(\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(S_n)}_{\text{opt}}(s_0, \overline{\psi}) + \varepsilon'_n - \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(S_n)}_{\text{opt}}(s_0, \underline{\psi}) + \varepsilon''_n) \\
&\leqslant \lim_{n\to\infty}(\text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(S_n)}_{\text{opt}}(s_0, \overline{\psi}) - \text{val}^{\mathcal{M}^{\text{sub}}_{\ell}(S_n)}_{\text{opt}}(s_0, \underline{\psi})) + 2 \cdot w \cdot \epsilon \\
&\overset{(4.94)}{\leqslant} 3 \cdot w \cdot \epsilon \quad \text{(with probability 1)}
\end{aligned}$$

Let $\lim_{n\to\infty}(\overline{v}_n - \underline{v}_n) = a$. Then $\forall \delta > 0$ there exists $k(\delta)$, such that $\forall n \geqslant k(\delta)$: $\overline{v}_n - \underline{v}_n - a < \delta$, or equivalently $\overline{v}_n - \underline{v}_n < a + \delta \leqslant 3 \cdot w \cdot \epsilon + \delta$. Since it holds for all values of $\delta > 0$, it also holds for $0 < \delta_0 \leqslant \epsilon \cdot (1 - 3 \cdot w)$ and we obtain that there exists an iteration $k(\delta_0)$ at which $\overline{v}_{k(\delta_0)} - \underline{v}_{k(\delta_0)} < 3 \cdot w \cdot \epsilon + \epsilon \cdot (1 - 3 \cdot w) = \epsilon$, which means that the exit condition of the while-loop of Algorithm 4 is fulfilled and therefore the algorithm terminates.

□

Remark 4.4.2. *As follows from the proof of the theorem above, if a scheduler satisfies (4.94), then Algorithm 4 is guaranteed to terminate with probability 1. Since correctness of the algorithm does not depend on the choice of* π_{sim}*, this means that Algorithm 4 is not restricted to only two options (*π_{uni} *or* $\overline{\pi}^x_{\text{opt}}/\underline{\pi}^x_{\text{opt}}$*) when it comes to choosing scheduler* π_{sim}*. One trivial way to satisfy (4.94) is to ensure that eventually the whole state-space is sampled under scheduler* π_{sim} *with probability* 1*. This is exactly the reason why scheduler* π_{uni} *can be used.*

4.5 Experimental Evaluation

In this section we will present an empirical comparison of (i) various exhaustive approaches to quantify the time-bounded reachability probability (Section 4.5.1) and (ii) instantiations of `PartialTBR` with their exhaustive counterparts (Section 4.5.2). The experiments were conducted on Intel Core i7-4790 with 16 GB of RAM running 64-bit Ubuntu Linux 18.04. Details on how to obtain all the data points used in the plots of this section can be found in Appendix B.1.1.

All the scatter plots shown below have log-log axes. A point $\langle x, y\rangle$ states that the runtime of the algorithm noted on the x-axis on one instance was x seconds while the runtime of the algorithm noted on the y-axis was y seconds. Thus points above the diagonal indicate instances where the x-tool was the fastest. We set the timeout to 20 minutes; a timeout is denoted in plots by an "`x`".

4.5.1 Comparison of Exhaustive Approaches

In this section, we will compare `SwitchStep` (implemented in the `Modest Toolset` [HH14]), `Unif+` (available in the `Modest Toolset`) and `FixStep` (available in `Storm` [DJKV17]). In plots and tables we will refer to `SwitchStep` with `Sw`, `Unif+` with `U+` and `FixStep` with `Fx`. We keep the default parameters for all the algorithms. When we request a certain precision for results, we request absolute, not relative, precision. Notice that by default `Unif+` guarantees the requested precision only for the selected initial states of the model, rather than for the whole state-space. In contrast, `SwitchStep` provides guarantees for values of all the states.

In this section, the runtime of an algorithm on a certain problem includes only the time it takes to approximate the time-bounded reachability values and does not include the time that it takes to load the respective model into memory. The experiments are performed on benchmarks from the Quantitative Verification Benchmark Set (QVBS [HKP$^+$19]). Some of the models that we used are different from those in the QVBS because we use the same models for experimental evaluation of this chapter and the following one. The latter required us to enrich the models with rewards and add new properties. The modified models can be found in Appendix B.2.1.

Figure 4.7 shows the results of the experiments on a combined set of benchmarks containing one or two instances of almost all the benchmarks from the Quantitative Verification Benchmark Set (QVBS [HKP$^+$19]). Excluded are those benchmarks that have spurious non-determinism, those that are way too large for being loaded into memory, or are analysed very quickly by most of the algorithms. The state-space of benchmarks in this set ranges from 10^2 to 10^7. We run experiments for precision 10^{-3} and 10^{-6}.

`SwitchStep` vs `FixStep`. The left plot in Figure 4.7 represents the general trend observed in many experiments: The algorithm `FixStep` does not scale well with the size of the problem (model parameters, precision, time bound). For larger benchmarks, it usually requires more than 20 minutes. This is likely due to the fact that the discretisation step used by `FixStep` is very small, which means that the algorithm performs many iterations. Table 4.1 reports on the size of the discreti-

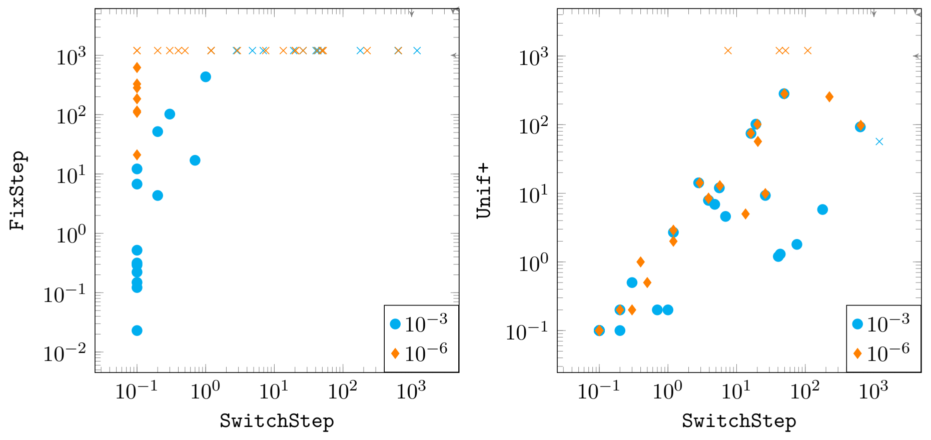


FIGURE 4.7: Runtime of `SwitchStep`, `Unif+` and `FixStep` on a combined set of benchmarks.

sation step for `FixStep` and the steps made by `SwitchStep` on a few benchmarks. Here column δ(`Fx`) shows the length of the discretisation step of `FixStep`. Columns min δ(`Sw`), avg δ(`Sw`) and max δ(`Sw`) show minimal, average and maximal steps computed by `SwitchStep` respectively. The average step used by `SwitchStep` is several orders of magnitude larger than that of `FixStep`. Therefore `SwitchStep` performs much fewer iterations. Even though each iteration takes longer, the overall significant decrease in the number of iterations leads to much smaller total runtime.

`SwitchStep` vs `Unif+`. Situation between `SwitchStep` and `Unif+` is not so clear. In general, the plot on the right in Figure 4.7 suggests that `Unif+` performs better than `SwitchStep` when precision set for the experiments is relatively small. To investigate this further we conduct a series of experiments in which we vary one of the parameters of the problem (model parameters, time bound, precision) and keep the other parameters fixed. We perform this evaluation on four of the benchmarks from the Quantitative Verification Benchmark Set (QVBS, [HKP+19]): *Dynamic Power Management* (`dpm`, version 2), *Fault Tolerant Workstation Cluster* (`ftwc`, version 3), *Polling System* (`ps`, version 3) and *Reentrant Queuing System* (`qs`, version 3). All models have open parameters, such as buffer capacity (that affects state-space size), the number of various request types (that related to the maximum number of enabled actions), etc. Varying these parameters allows us to scale the models up from small to large state-spaces. Table 4.2 reports on minimum and maximum values of

Table 4.1: The discretisation step used in some of the experiments from Fig. 4.7 for precision 0.001.

Model	δ(`Fx`)	min δ(`Sw`)	avg δ(`Sw`)	max δ(`Sw`)
`dpm`	$1.6 \cdot 10^{-6}$	$1.6 \cdot 10^{-6}$	0.013	1.212
`qs`	$1.075 \cdot 10^{-5}$	$1.075 \cdot 10^{-5}$	0.007	3.252
`ps`	$5.16 \cdot 10^{-6}$	5	5	5
`ftwc`	$7.86 \cdot 10^{-5}$	0.757	2.5	4.242

Table 4.2: Parameters of models used for the experiments.

Model	$\|S\|$	$\mathbf{E}_{\max}$	$\max_{s\in S} \|Act(s)\|$
`dpm`	225-111,329	2.60-5.00	4-7
`ftwc`	35,995-8,008,795	2.06-3.02	4-4
`ps`	70-6,545,410	2.80-128.80	2-7
`qs`	204-332,107	4.50-8.50	2-12

the state-space sizes, $\mathbf{E}_{\max}$ and highest number of enabled actions $\max_{s\in S} |Act(s)|$ across the four models used for the evaluation presented in this section.

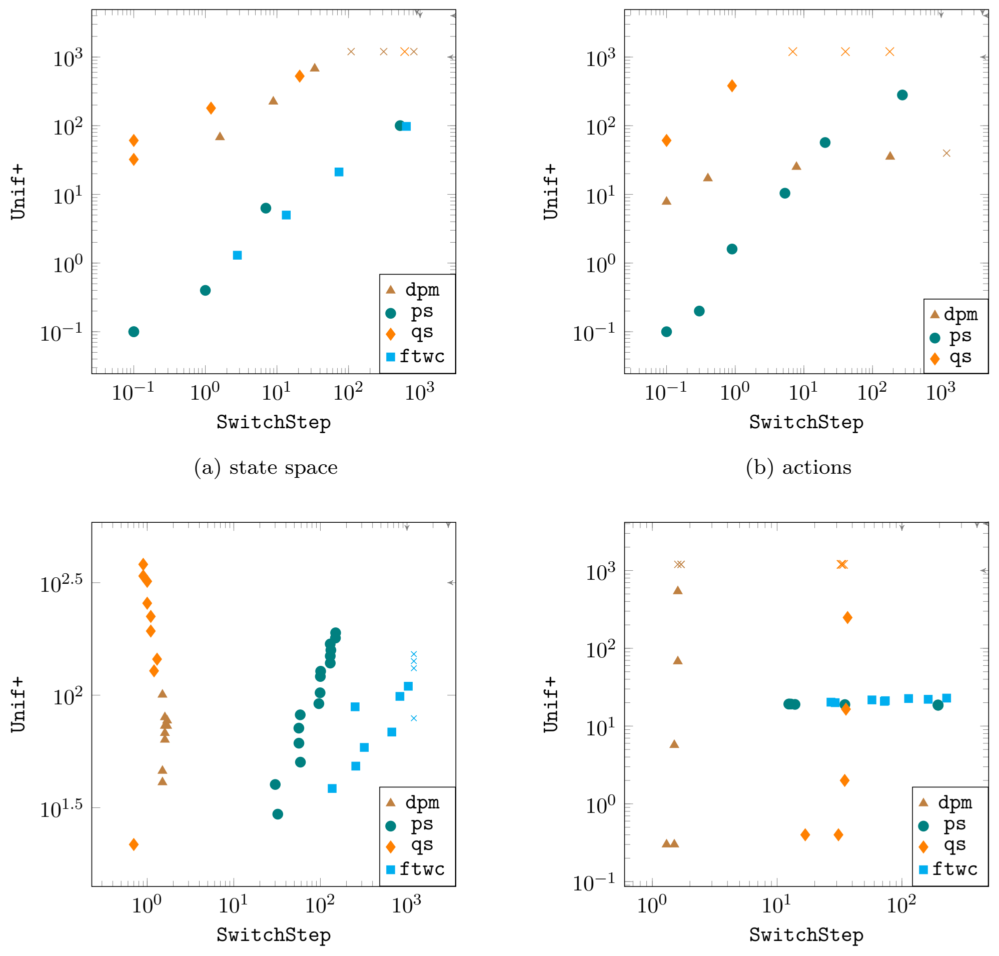
Figure 4.8 summarises the results of the experiments. Here we fix the precision to be 10^{-6} for all the experiments except the ones in the plot on the bottom right.

Model size. For the top left plot we vary only those parameters of models that affect the state-space size and for the top right plot we vary parameters that affect the number of enabled actions. For the latter plot we do not compare the algorithms on `ftwc` model because the model does not provide any interface to vary the number of enabled actions. In general, as expected, the running time of both algorithms grows with the growth of the model size. On two models, `ftwc` and `ps`, `Unif+` and `SwitchStep` perform similarly. There are case studies on which `SwitchStep` performs better than `Unif+`, those are `dpm` and `qs`. However, the data for model `dpm` from the top right plot suggests that `Unif+` scales better than `SwitchStep` with the increase of the number of enabled actions, which is an interesting question to investigate as future work.

Precision and time bounds. Bottom plots show the results of the evaluation when changing the time bounds (left) and precision (right). When it comes to varying the time bounds, `SwitchStep` seems to be more robust than `Unif+` on most of the considered benchmarks. When varying precision, there are cases when `SwitchStep` scales better than `Unif+` (`dpm` and `qs`), and there are cases of the contrary (`ps`, `ftwc`).

4.5.2 Comparison of Partial Approaches

In this section, we will present the experimental evaluation of various instantiations of `PartialTBR` (Algorithm 4), implemented in the `Modest Toolset`, and their comparison to exhaustive approaches. The evaluation will be performed on the following benchmarks: `ftwc`, `qs`, `dpm` considered in the previous section, a variation of the `ps` benchmark with five queues instead of two and only one task type, and three more benchmarks from QVBS: *Vehicle Guidance System* (`vgs`, version 1), *Video Streaming Client* (`stream`, version 1) and *Hypothetical Example Computer System* (`hecs`, version 1). The models and properties used for the experiments can be found in Appendix B.2.2. Their parameters are reported in Table 4.3. In this section, the runtime of an algorithm on a certain problem includes the time it takes to load the respective model into memory. We set the limit to the amount of RAM available to processes to 10Gb. A timeout is denoted in tables by "`TO`" and a memout by "`MO`".



(a) state space

(b) actions

(c) time

(d) precision

FIGURE 4.8: Runtime comparison of `SwitchStep` and `Unif+` varying those problem parameters that affect state-space size (4.8a), max number of actions (4.8b), time (4.8c) and precision (4.8d)

Recall that `PartialTBR` has parameter $sch \in \{\text{uni}, \text{opt}\}$. We will use sch as a subscript to specify which exactly parameter is used. As an underlying solver for time-bounded reachability, we will use `Unif+` or `SwitchStep`. Thus $\texttt{Unif+}_{\text{uni}}$ will refer to an instantiation of `PartialTBR` with sch = uni and where `Unif+` is used for time-bounded reachability. Here we do not consider the combination of `Unif+` and sch = opt because `Unif+` does not output a scheduler in a classical way, as it is defined in Definition 2.2.5.

We compare the performance of the instantiated algorithms with their originals. The precision for all the experiments is set to 0.01. The average runtime after 3 runs is presented in Table 4.4 and the average size of the explored state-space after 3 runs is reported in Table 4.5. Here the values for the column labelled with "uni" are the best values among the results of $\texttt{Unif+}_{\text{uni}}$ and $\texttt{SwitchStep}_{\text{uni}}$.

Uniform simulating scheduler performs well on many instances: `vgs`, `stream`,

Table 4.3: Parameters of models used for the experiments on `PartialTBR`.

Model	$\lvert S\rvert$
`vgs`	$> 1,640,000$
`stream`	200,070,001
`hecs`	$> 33,500,000$
`ftwc`	31,877,211
`ps`	$> 60,000,000$
`dpm`	$> 100,000,000$
`qs`	332,107

`hecs` and `ftwc`. On all these models the runtime of algorithms $\texttt{Unif+}_{\text{uni}}$ and $\texttt{SwitchStep}_{\text{uni}}$ is a few seconds (except for $\texttt{SwitchStep}_{\text{uni}}$ on `ftwc`) and the explored state-space constitutes less than 1% of the total state-space. The size of the explored state-space in these experiments is the reason for the instantiations of `PartialTBR` to outperform the original algorithms `Unif+` and `SwitchStep`.

Optimal Simulating Scheduler. There are cases in which the uniform scheduler performs poorly, those are the experiments on `ps` and `dpm` models. The `ps` model, for example, has five queues that store tasks. The tasks are to be processed by a server. The system starts when all the queues are full and the considered property requires one of the queues to be empty. Schedulers in this model need to decide from which queue to take the task for processing. Naturally, when a task is selected uniformly at random from different queues, then more states are explored and the overall runtime is higher. However due to setting $sch = \text{opt}$ the most promising scheduler is the one that selects tasks from the correct queue, which leads to more targeted exploration. Here $\texttt{SwitchStep}_{\text{opt}}$ performs even better than the original algorithm `SwitchStep`, finishing within less than 4 minutes (compared to more than 20 minutes for the original) and exploring less than 1% of the total state-space.

Hard instances. There are cases in which there is no small sub-MA that preserves the properties of interest with the required precision. Consider the experiment on the `qs` model. The model has two queues storing tasks and the initial state is when all the queues are empty. The property under consideration is the

Model	`U+`	$\texttt{U+}_{\text{uni}}$	`Sw`	$\texttt{Sw}_{\text{uni}}$	$\texttt{Sw}_{\text{opt}}$
`vgs`	`TO`	4.4	`TO`	4.4	4.4
`stream`	`MO`	1.01	`MO`	1.01	1.11
`hecs`	`TO`	2.93	`TO`	2.92	2.95
`ftwc`	`MO`	2.16	`MO`	550.2	261.1
`ps`	`MO`	`TO`	`MO`	`TO`	216.3
`dpm`	`MO`	870.3	`MO`	`TO`	276.2
`qs`	12.6	330.9	34.5	`TO`	`TO`

Table 4.4: Runtime in seconds for various instantiations of `PartialTBR`.

Table 4.5: Explored state-space size for *sch* = uni and *sch* = opt. Here "-" is used whenever the respective value is not available due to a timeout or a memout.

Model	Explored states		% of the total state space	
	uni	opt	uni	opt
`vgs`	1552	1568	< 0.09	< 0.09
`stream`	269	270	$1.4 \cdot 10^{-4}$	$1.4 \cdot 10^{-4}$
`hecs`	392	392	< 0.0012	< 0.0012
`ftwc`	7981	48,447	0.026	0.152
`ps`	-	43,976	-	< 0.074
`dpm`	738,029	439,769	< 0.73	< 0.44
`qs`	332,085	-	99.9	-

probability of all the queues to be full within a given time bound. Naturally, a queue cannot be full without going through states when the queue has n tasks, for n ranging from 1 to the maximal queue capacity. This means that most of the states of the model have to be explored in order to reach a goal state. As expected, on this problem the instantiations of `PartialTBR` run significantly longer than the exhaustive algorithms and almost all the states are explored.

`Unif+` *vs* `SwitchStep`. We observe that `Unif+` tends to perform better on those instances of the problem in which the probabilities are close to 0 or close to 1. It, therefore, seems to be a better choice for instances with rare events, such as those coming from the fault tree domain. However, at the moment `Unif+` cannot be used with *sch* = opt, in which case `SwitchStep` should be used. The optimal scheduler obtained from running `SwitchStep` on the upper bound model seems to have many more switching points compared to that of the lower bound model. As a future work one could consider optimising `SwitchStep` for these special cases.

Parameter n_{sim}. Performance of `PartialTBR` depends significantly on the value of parameter n_{sim}. Solving the underlying time-bounded reachability problem is expensive and therefore should be performed only a few times. This means that it is better to set n_{sim} to larger values so that more states are added to the model each time the sub-problem is solved. In our experiments, we choose the value of n_{sim} so that at each iteration the state-space is at least 1.5 times larger than the state-space at the previous iteration.

4.5.3 Conclusions

Exhaustive Approaches. We conclude that `SwitchStep` does not replace all existing algorithms for time-bounded reachability, however it does improve the state of the art in many cases and thus occupies its own niche among available solutions. We conjecture from the experimental data that the running time of `SwitchStep` tends to be better in those cases when switching points of the optimal strategy have to be computed accurately. Such problems are harder for `Unif+`, since the algorithm can only approximate time. When the error introduced by not switching the policy at the right moment is low, then `Unif+` runs quite fast, often performing only a few iterations. However slightly changing the time bound for the problem may lead to `Unif+` taking at least 20 minutes to compute the values.

Partial Approaches. In general, partial approaches are a good pick whenever huge models have a small sub-MA that preserves the value with adequate error, or when one needs to know a rough approximation of the time-bounded reachability. On such instances `PartialTBR` successfully finds a small subset of states and significantly reduces either the running time or the memory consumption required to solve the problem (as demonstrated in our experimental evaluation). Simply loading a large model into memory takes a lot of time and resources, which is exactly what the partial approaches avoid. As expected, the performance of `PartialTBR` heavily depends on the structure of the model and the property under consideration. For example, the same problems but with larger time bounds may not have a small sub-MA preserving the value because the optimal scheduler on these new time points has to visit the states that could be otherwise discarded.

4.6 Conclusions and Discussion

In this chapter we looked closely at the problem of computing optimal reachability probabilities $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\psi)$ for an arbitrary formula $\psi = \Phi_1 \,\mathrm{U}^{[\![a,b]\!]} \Phi_2$ and labelled MA $\mathcal{M}_\ell = (\mathcal{M}, AP, \mathit{lab})$. We reiterate over major steps below and highlight the main contributions with $*$:

– We have identified that computing $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\psi)$ can be reduced to computing at most two values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}'}(s, b-a, g')$ and $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}''}(s, a, g'')$, where $\mathcal{M}'$ and $\mathcal{M}''$ are slightly modified versions of $\mathcal{M}$ and g', g'' are goal functions that depend on ψ (Section 4.1.2).

$*$ We have presented a convenient characterisation of the next switching point of an optimal piecewise-constant scheduler for $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, c, g)$ as the earliest intersection of finitely many functions (linear in the number of state-action pairs of $\mathcal{M}$), given implicitly by a system of linear differential equations (Section 4.3.3).

– Using this characterisation we have developed a scheme to approximate the switching points (Section 4.3.4). We achieve this by approximating the solution to the system of differential equations via uniformisation.

$*$ All the results discussed above enabled us to design a novel algorithm for approximating values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, c, g)$ (`SwitchStep`, Algorithm 2). Under the assumption that all the goal states are absorbing (Assumption 4.3.1) the error induced by the approximations is guaranteed to be below a given threshold ϵ.

$*$ We explore an alternative way to optimise the analysis of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\psi)$ on large Markov automata by performing computations on partial state-spaces, what resulted in algorithm `PartialTBR` (Algorithm 4). We use simulations to identify the subset of states that are most relevant to values $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\psi)$, i. e. removing one of those states may induce an error that is larger than the requested error bound. Instead of solving $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}_\ell}(\psi)$ on $\mathcal{M}_\ell$ we solve two simpler problems $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}'_\ell}(\psi')$ and $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}'_\ell}(\psi'')$, where the state-space of $\mathcal{M}'_\ell$ is previously identified subset of relevant states. The algorithm terminates almost surely and the error induced by the approximations is guaranteed to be below a given error bound ϵ.

– We conclude the chapter with the empirical evaluation of both the algorithms. It shows that algorithm `SwitchStep` outperforms `FixStep` on most of the benchmarks, with the exception of some small models. The algorithm is competitive with `Unif+`, however, does not strictly outperform it. There are many MA models that are extremely large and cannot be analysed by exhaustive algorithms, such as `SwitchStep` or `Unif+`, within 20 minutes, but can be analysed by `PartialTBR` within a few seconds.

Discussion.

– *The study of switching points of optimal piecewise constant schedulers for* $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(c, g)$. If an optimal piecewise constant scheduler exists and its switching points are known, then computation of $\mathrm{val}_{\mathrm{opt}}^{\mathcal{M}}(s, c, g)$ can be performed efficiently, for example as shown in Section 4.3.1. And the same holds for CTMDPs, see [Mil68] (or the same in [BS11]). For Markov automata efficient computation of switching points is made possible due to the decoupling of timed delays from non-deterministic actions. We conjecture that this may be the key in developing a good characterisation of switching points. It is not straightforward whether the theory that we developed for switching points in this chapter can be transferred to CTMDPs, where this decoupling is not present. As mentioned before, one advantage of Markov automata over CTMDPs is their compositionality. For us, another significant advantage is the possibility to efficiently compute switching points of an optimal strategy.

– *Decidability of CSL model-checking.* As observed by Daniel Stan, our characterisation of switching points suggests that the switching points are not algebraic numbers due to the Lindemann-Weierstrass theorem. This could lead to a negative result on decidability of model-checking formula $\mathcal{P}_{\trianglelefteq p}^{\mathrm{opt}}(\Phi_1 \,\mathrm{U}^{[\![a,b]\!]}\, \Phi_2)$.

Future Directions.

– *Generalisations.* Formula $\mathcal{P}_{\trianglelefteq p}^{\mathrm{opt}}(\Phi_1 \,\mathrm{U}^{[\![a,b]\!]}\, \Phi_2)$ is the simplest time-bounded property among those presented in Chapter 3 Section 3.3. An obvious question is whether it is possible to generalise our results to other problems, such as: (i) the cumulative time-bounded reward property (see Chapter 3 Section 3.3); (ii) an even more general problem of cost-bounded rewards considered in [Hat17]; (iii) all these properties over hybrid systems with non-determinism; (iv) integrate our solution into multi-objective model-checking [QJK17].

– *Optimise Algorithm 2.* Efficiency of this algorithm depends on Lemmas 4.3.15 and 4.3.16 that give a bound on the error introduced by following a possibly sub-optimal strategy. If these bounds are too pessimistic, the scheduler approximated by the algorithm switches more often then is needed, what leads to higher running time. The algorithm would definitely benefit from better bounds that are easier to compute.

– *Optimise Algorithm 4.* The main weakness of this algorithm is the fact that the relevant state-space can only grow from one iteration to another. This means that at each iteration the underlying exhaustive analysis on the partial state-space becomes harder and harder. The algorithm would definitely benefit from a more lightweight version of this step that only works on a subset of states of the relevant set and only for some of the time points, just like the classical BRTDP.

– *Patterns in optimal schedulers.* Most of the known MA case studies consist of parametrised sub-components running in parallel. New models can be obtained by changing the number of these components or their parameters. As an example, many case studies use a buffer of finite length K. Large buffer size leads to a larger state-space. It is interesting from a theoretical and a practical perspective to see whether an optimal scheduler for a problem with $K = k_1$ and an optimal scheduler for a problem with $K = k_2 > k_1$ have anything in common. It could be possible that there is a pattern in the decisions of the optimal scheduler. If this is the case, one could compute optimal schedulers on models with small values of K and from there obtain a guess of an optimal scheduler for models with larger K. The guessed scheduler will most likely not be optimal, but it may not be too far from an optimal one. Notice that similar approaches that guess optimal schedulers for other models, such as CTMDPs, usually do not provide any guarantees on how far the guessed scheduler is from an optimal one. In order to establish this one needs to compute the value induced by the scheduler, the optimal value and compare the two. The results of this chapter, in particular, Lemmas 4.3.15 and 4.3.16, allow us to obtain the bound on the error without computing the optimal value. This can be achieved by first setting scheduler π to the respective stationary part of the guessed scheduler in step 8. Second, one needs to skip the computation of the step size δ and instead set it to the step size of the guessed scheduler. This can be achieved by substituting the call to `FindStep` at line 9 with the call to `CheckInterval` with respective arguments. No modifications of Lemmas 4.3.15 and 4.3.16 are needed. The error bound returned by the algorithm is the error induced by the scheduler.

  Assuming that the guessed scheduler is not far from the optimal one, one could also use the guessed scheduler as a basis to speed up the computation of the optimal scheduler. This could be achieved by optimising procedure `FindStep` to check for minor derivations of the step size.

Optimal Long-Run Rewards 5

In this chapter, we move our attention to the problem of computing optimal long-run average rewards for Markov automata. Recall that this property describes the average reward that is expected to be accumulated per time unit in the best or worst case. The main result of this chapter is an iterative algorithm for approximating optimal long-run average rewards for Markov automata.

We start in Section 5.1 with laying the foundations for introducing the algorithm and describing a sub-problem of optimal cumulative unbounded reward for MDPs. In Section 5.2 we describe an algorithm developed in [GHH$^+$14, GTH$^+$14] to solve the problem for the general case. The algorithm reduces the problem to computing optimal long-run average rewards for a subclass of *unichain* Markov automata. Informally, this subclass consists of Markov automata that have a strongly connected underlying graph. The authors of [GHH$^+$14, GTH$^+$14] solve this latter problem via a reduction to linear programming.

In Section 5.3 we present a value iteration algorithm for approximating long-run average rewards in a unichain Markov automaton. To arrive there, we show that a Markov automaton can be considered as a compact representation of a CTMDP with – in the worst case – exponentially more transitions. From this new perspective, the analysis of Markov automata does not require designing new techniques but lets us adopt those used for CTMDPs. However, a trivial adaptation of CTMDP algorithms to an exponentially larger model obtained from a Markov automaton would obviously induce exponential runtime. We deal with exponentiality with the help of dynamic programming. We isolate a part of the CTMDP algorithm that runs in exponential time and show that by exploiting the structure of the original Markov automaton we can solve this sub-problem efficiently. Next, we show how to integrate the approximate solution for a unichain Markov automaton into the general case solution with guaranteed error bounds. We conclude the section with an empirical comparison of the linear-programming based approach and this new iterative algorithm.

5.1 Preliminaries and Problem Statement

Throughout the chapter we work with a Markov reward automaton $\mathcal{M}_\varrho = (\mathcal{M}, \varrho)$, where ϱ is a reward structure over the MA $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$.

In this chapter we will use the same relation operators as in the previous one (see Chapter 4, Section 4.1), in order to present the results for both cases opt = sup and opt = inf. Below we reiterate over the definitions for convenience:

$$\preccurlyeq_{\text{opt}} = \begin{cases} \leqslant & \text{if opt} = \sup \\ \geqslant & \text{if opt} = \inf \end{cases} \qquad \succcurlyeq_{\text{opt}} = \begin{cases} \geqslant & \text{if opt} = \sup \\ \leqslant & \text{if opt} = \inf \end{cases} \tag{5.1}$$

$$\prec_{\text{opt}} = \begin{cases} < & \text{if opt} = \sup \\ > & \text{if opt} = \inf \end{cases} \qquad \succ_{\text{opt}} = \begin{cases} > & \text{if opt} = \sup \\ < & \text{if opt} = \inf \end{cases} \tag{5.2}$$

Definition 5.1.1. *Let $\pi \in \Pi_\mu$ and $s \in S$. The* (expected) long-run average reward value *and* optimal (expected) long-run average reward value *are defined as follows:*

$$\begin{aligned} \text{lval}_\pi^{\mathcal{M}_\varrho}(s) &:= \mathbb{E}\left[L_{\mathcal{M}_\varrho,s,\pi}\right] = \int_{Paths^\omega} L_{\mathcal{M}_\varrho,s,\pi}(\rho) \cdot \Pr_{\pi,s}\left[\mathrm{d}\rho\right] \\ \text{lval}_{\text{opt}}^{\mathcal{M}_\varrho}(s) &:= \underset{\pi\in\Pi_\mu}{\text{opt}} \ \text{lval}_\pi^{\mathcal{M}_\varrho}(s) \end{aligned} \tag{5.3}$$

We denote by $\overline{\text{lval}}_\pi^{\mathcal{M}_\varrho}$ the vector of values $\text{lval}_\pi^{\mathcal{M}_\varrho}(s)$ for all states $s \in S$ and analogously the vector of optimal values is denoted as $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}$. For the above definitions we will omit the superscript $\mathcal{M}_\varrho$ whenever the Markov reward automaton under consideration is clear from the context.

We will call a scheduler π *optimal*, if $\text{lval}_{\text{opt}}^{\mathcal{M}_\varrho}(s) = \text{lval}_\pi^{\mathcal{M}_\varrho}(s)$. A scheduler π is called *ϵ-optimal* if $\forall s \in S : \text{lval}_{\text{opt}}^{\mathcal{M}_\varrho}(s) - \text{lval}_\pi^{\mathcal{M}_\varrho}(s) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\text{opt})}\epsilon$.

We are now ready to define the problem that is going to be the topic of this chapter:

Problem 3. *Compute the values $\text{lval}_{\text{opt}}^{\mathcal{M}_\varrho}(s)$ for all $s \in S$, as well as an optimal scheduler π.*

In order to solve this problem one needs to be able to solve a sub-problem of computing optimal expected cumulative unbounded reward in special instances of Markov automata - MDPs. This sub-problem plays an important role in the solution to Problem 3 and is going to be the topic of the following section.

5.1.1 Cumulative Unbounded Reward for MDPs

In this section we will work on the definition of MDPs presented in Chapter 2, Section 2.2.5. We will tailor to this definition the notion of optimal expected cumulative unbounded reward, that has been already defined for Markov reward automata in Chapter 3, Section 3.3.

First, we need to define reward structures for MDPs. Recall that when thinking of an MDP as of a special instance of Markov automata, states of the MDP are probabilistic states. Due to Remark 3.3.1, only transition rewards of probabilistic states affect various reward functions for Markov automata. In order to obtain a reward structure for an MDP we can, therefore, simplify the definition of the reward structure for Markov automata by omitting state rewards:

Definition 5.1.2. *A* reward structure *for an MDP* $\mathcal{D} = (S, Act, \mathbf{P})$ *is a function* $\varrho_{\text{trn}} : S \times Act \to \mathbb{R}$*.*

Given an MDP $\mathcal{D} = (S, Act, \mathbf{P})$ with a reward structure $\varrho_{\text{trn}}^{\mathcal{D}}$ we can obtain a Markov reward automaton representation of this MDP as follows: $\mathcal{M}_\varrho = (\mathcal{M}, \varrho)$, where $\mathcal{M} = (S, Act, \dashrightarrow, \mathbf{R})$, such that $\forall s, s' \in S : \mathbf{R}[s, s'] = 0$, $\varrho = (\varrho_{\text{st}}, \varrho_{\text{trn}}), \forall s \in S : \varrho_{\text{st}}(s) = 0, \forall \tau = (s, \nu, \mu) \in \rightsquigarrow : \varrho_{\text{trn}}(\tau) = \varrho_{\text{trn}}^{\mathcal{D}}(s, \nu)$.

Now we can define the (optimal) expected cumulative unbounded reward for MDPs as the (optimal) expected cumulative unbounded reward of its MRA representation (Chapter 3, Section 3.3). In the following we will write it down explicitly.

Let $\mathcal{D} = (S, Act, \mathbf{P})$ be an MDP and ϱ_{trn} is its reward structure, $s \in S, \pi \in \Pi_\mu$. We will denote with $\mathcal{D}_\varrho$ a pair $\mathcal{D}_\varrho = (\mathcal{D}, \varrho_{\text{trn}})$. The *cumulative unbounded reward* is the random variable $C^{unb}_{\mathcal{D}_\varrho, s, \pi} : Paths^\omega \to \mathbb{R}^\infty_{\geqslant 0}$ on the probability space $(Paths^\omega, \mathcal{P}^\omega, \Pr_{\pi,s}[\cdot])$, such that for a path $\rho = s_0 \xrightarrow{\nu_0, t_0} \cdots s_n \xrightarrow{\nu_n, t_n} \cdots \in Paths^\omega$:

$$C^{unb}_{\mathcal{D}_\varrho, s, \pi}(\rho) := \lim_{k \to \infty} \sum_{i=0}^{k} \varrho_{\text{trn}}(s_i, \nu_i)$$

Definition 5.1.3. *The* optimal expected unbounded reward for an MDP $\mathcal{D}_\varrho, s \in S$ *is defined as:*

$$\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s) := \underset{\pi \in \Pi_\mu}{\text{opt}}\ \mathbb{E}\left[C^{unb}_{\mathcal{D}_\varrho, s, \pi}\right] = \underset{\pi \in \Pi_\mu}{\text{opt}} \int_{Paths^\omega} C^{unb}_{\mathcal{D}_\varrho, s, \pi}(\rho) \cdot \Pr^{\mathcal{D}_\varrho}_{\pi, s}[\mathrm{d}\rho], \qquad (5.4)$$

and for a given stationary scheduler π*:*

$$\text{urew}^{\mathcal{D}_\varrho}_{\pi}(s) := \mathbb{E}\left[C^{unb}_{\mathcal{D}_\varrho, s, \pi}\right],$$

if the set of paths for which $C^{unb}_{\mathcal{D}_\varrho, s, \pi}(\rho)$ *is undefined has measure* 0*.*

We will denote with $\overline{\text{urew}}^{\mathcal{D}_\varrho}_{\text{opt}}$ and $\overline{\text{urew}}^{\mathcal{D}_\varrho}_{\pi}$ vectors of values $\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s)$ and $\text{urew}^{\mathcal{D}_\varrho}_{\pi}(s)$ for each $s \in S$ respectively.

The problem of computing value $\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s)$ for MDPs is well studied in, e. g. [Put94]. Values $\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s)$ can be computed exactly by such techniques as policy iteration and linear programming [Put94], or approximated via interval value iteration [QK18, BKL$^+$17]. Below we will discuss an important special case of MDPs that will appear often in the context of Markov automata and for which there exists an efficient iterative solution for computing values $\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s)$ exactly.

A state of an MDP is called a *terminal state* if it has only one outgoing transition, which is a self-loop with probability 1 and the reward assigned to this transition is 0. We call an MDP *acyclic* if from each state of the MDP the probability to reach at least one terminal state equals 1 and there are no cycles in the MDP except for the self-loops of terminal states. We can assume w. l. o. g. that an acyclic MDP has only one terminal state, that we will denote with t.

We say that a non-terminal state s *has maximal depth* i, or $d(s) = i$, if the longest finite path ρ from s until a terminal state via non-terminal states has length $|\rho| = i$. We define $d(t) = 0$. Since value $\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(s)$ for each state s of an acyclic MDP depends only on values $\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(s')$, where $d(s') < d(s)$, one can compute the values $\overline{\mathrm{urew}}_{\mathrm{opt}}^{\mathcal{D}_\varrho}$ backwards starting from states with lowest value $d(s)$ and proceeding as follows:

$$\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(s) = \begin{cases} 0 & d(s) = 0 \\ \underset{\alpha \in Act(s)}{\mathrm{opt}} \left\{ \varrho_{\mathrm{trn}}(s, \alpha) + \sum\limits_{s' \in S} \mathbb{P}[s, \alpha, s'] \cdot \mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(s') \right\} & d(s) > 0 \end{cases}$$

Having computed values $\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(s)$ for all states with the same $d(s) = d$ we can proceed to computing $\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(s)$ for states with $d(s) = d + 1$.

5.2 General Case Solution

To date, there exists only one solution for Problem 3 presented in [GTH+14], which is going to be the topic of this section. All the definitions, examples, figures and results presented in this section originate either from [GTH+14] or from [GHH+14].

The approach is based on splitting the problem into three larger steps:

1. Find all *maximal end components* of $\mathcal{M}_\varrho$.
2. Compute $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{M}'_\varrho}$ for each maximal end component $\mathcal{M}'_\varrho$.
3. Compute $\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}}(\varrho_{\mathrm{trn}})$ for the *collapsed MDP* $\mathcal{D}$.

In the following, we will go into detail of each of these steps.

Step 1. Maximal End Components. A *maximal end component* of an MRA can be seen as a maximal sub-MRA whose underlying graph is strongly connected:

Definition 5.2.1. *A* sub-MRA $\mathcal{M}'_\varrho$ *of* $\mathcal{M}_\varrho$ *is an MRA* $\mathcal{M}'_\varrho = (\mathcal{M}', \varrho)$*, where* $\mathcal{M}' = (S', Act, \dashrightarrow', \mathbf{R}')$*,* $S' \subseteq S$ *and transition relation* $\rightsquigarrow'$ *induced by* $\dashrightarrow'$ *and* $\mathbf{R}'$ *satisfies the following :*

- $\rightsquigarrow' \subseteq \rightsquigarrow$*;*
- *each state* $s \in S'$ *has at least one outgoing transition* $\tau = (s, \nu, \mu) \in \rightsquigarrow'$*;*
- *all successors of a state in* S' *via transitions in* $\rightsquigarrow'$ *are in* S' *as well: if* $s \in S$ *and* $\tau = (s, \nu, \mu) \in \rightsquigarrow'$*, then* $post(s, \tau) \subseteq S'$*.*

Since state-space and transition relations are the only things that differ between an MRA and its sub-MRA, in the following for simplicity we will denote a sub-MRA as a tuple $\mathcal{M}'_\varrho = (S', \rightsquigarrow')$.

Definition 5.2.2. *An* end component $\mathcal{M}'_\varrho = (S', \rightsquigarrow')$ *is a sub-MRA with a strongly connected underlying graph. An end component is called* maximal (MEC) *if there exists no other end component* $\mathcal{M}''_\varrho = (S'', \rightsquigarrow'')$*, such that* $S' \subseteq S''$ *and* $\rightsquigarrow' \subseteq \rightsquigarrow''$*.*

The problem of finding all MECs of an MRA is equivalent to decomposing a graph into strongly connected components. This problem can be solved efficiently via graph-based algorithms, e. g. by the solution proposed in [CH11] in time $O(m \cdot \min\{\sqrt{m}, n^{2/3}\})$, where $n = |S|$ and m is the number of edges of the induced graph (see Chapter 4, Section 4.1.1).

Step 2. Computation of $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}'_\varrho}$ for each MEC $\mathcal{M}'_\varrho$ is possible due to the specific structure of MECs. It has been shown in [GHH+14, GTH+14] that for each MEC $\mathcal{M}'_\varrho$ there exists an optimal stationary scheduler for $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}'_\varrho}$ that induces a *unichain stochastic process*:

Definition 5.2.3. *A continuous-time stochastic process is a* unichain *if there exists a single recurrent subset of states of the process and possibly some transient states. A subset of states* S' *is called* recurrent *if for each two states* $s, s' \in S'$ *the probability of the set of paths starting from* s *and visiting* s' *is non-zero, and the probability of the set of paths starting from* s *and revisiting* s *after leaving it is* 1*. If the latter condition does not hold for a state* s*, then it is called* transient*.*

Therefore in order to compute the value $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}'_\varrho}$ it suffices to consider only those stationary schedulers for MEC $\mathcal{M}'_\varrho$ that induce a unichain stochastic process.

Definition 5.2.4. *An MRA* $\mathcal{M}'_\varrho$ *is called* unichain *if every stationary scheduler on* $\mathcal{M}'_\varrho$ *induces a unichain stochastic process.*

In a MEC or a unichain MRA $\mathcal{M}'_\varrho = (S', \rightsquigarrow')$ the long-run average reward value is the same for all states, i. e. $\forall s', s'' \in S' : \text{lval}_{\text{opt}}^{\mathcal{M}'_\varrho}(s') = \text{lval}_{\text{opt}}^{\mathcal{M}'_\varrho}(s'')$. In this case, we will denote the value simply with $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}'_\varrho}$.

To conclude, the computation of $\overline{\text{lval}}_{\text{opt}}$ for MECs is strongly related to the computation of $\overline{\text{lval}}_{\text{opt}}$ for unichain MRA. The solution provided in [GTH+14] is based on a reduction to a linear program with $|S|$ variables.

Step 3. Optimal Expected Cumulative Reward for a Collapsed MDP. Let $\mathcal{M}^j_\varrho = (S^j, \rightsquigarrow^j), j = 1..k$ be all the MECs of $\mathcal{M}_\varrho$. Given the optimal values $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}^j_\varrho}$

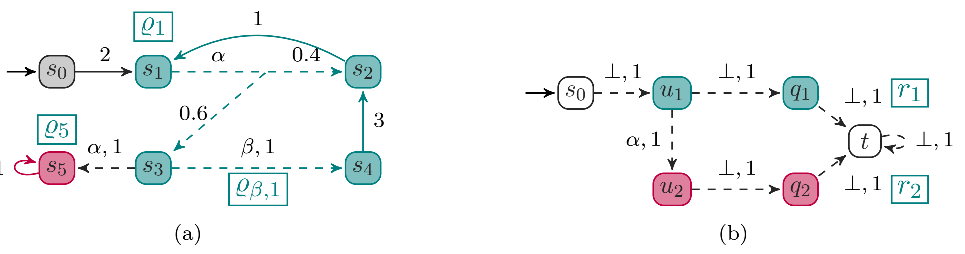


Figure 5.1: Fig. (5.1a) shows an example of a Markov reward automaton and Fig. (5.1b) depicts its collapsed MDP.

for each $\mathcal{M}_\varrho^j$ one can derive the value of $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}$ as follows:

$$\forall s \in S : \text{lval}_{\text{opt}}^{\mathcal{M}_\varrho}(s) = \underset{\pi \in \Pi_\mu}{\text{opt}} \sum_{j=1}^{k} \Pr_{\pi,s}^{\mathcal{M}} [\Diamond\Box S_j] \cdot \overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho^j}, \tag{5.5}$$

where

$$\Diamond\Box S_j = \{\rho \in \mathit{Paths}^\omega(\mathcal{M}) \mid \exists n \in \mathbb{Z}_{\geqslant 0} : \forall k \geqslant n : \rho[k] \in S^j\}$$

is the set of paths that starting from state s eventually reach and then forever stay in the MEC $\mathcal{M}_\varrho^j$.

Computation of the right-hand side of (5.5) can be reduced to the computation of $\overline{\text{urew}}_{\text{opt}}^{\mathcal{D}_\varrho}$ for the *collapsed MDP* $\mathcal{D}_\varrho$. We first explain how to obtain this MDP on an example.

Example 5.2.1. *Consider the MRA depicted in Fig. 5.1. Red and blue colours denote states and transitions that belong to a certain MEC. For the collapsed MDP $\mathcal{D}_\varrho$ each MEC $\mathcal{M}_\varrho^j = (S^j, \rightsquigarrow^j)$ is substituted with two states q_j and u_j. State q_j denotes the MEC itself while u_j represents a possibility to decide whether to stay forever in $\mathcal{M}_\varrho^j$ (by transitioning to q_j) or to go to some other MEC. It, therefore, has a transition leading to q_j with probability* 1 *and all the transitions of states from S^j, that are not part of $\mathcal{M}_\varrho^j$. In case of the MRA from Fig. (5.1a) it is only one transition from state s_3 via action α. Markovian states that are not part of any MEC become regular MDP states with only one enabled action $\perp$ and probability distribution $\mathbb{P}[s,\cdot]$ over successor states. Probabilistic states that are not part of any MEC are moved to $\mathcal{D}_\varrho$ without any changes. The reward of each state q_j is set to be equal to $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho^j}$. Since reward from a MEC needs to be collected only once, each state q_j has a transition leading to a new state t with probability* 1. *The latter has reward* 0, *what ensures that an infinite path entering this state at some point will have a finite reward. All other states of the MDP also have reward* 0.

Formally, the collapsed MDP $\mathcal{D}_\varrho = (\mathcal{D}, \varrho_\mathcal{D})$, where $\mathcal{D} = (S_\mathcal{D}, \mathit{Act}_\mathcal{D}, \mathbf{P})$ and the reward structure $\varrho_\mathcal{D}$ are defined as follows. Let $\mathcal{M}_\varrho^j = (S^j, \rightsquigarrow^j), j = 1..k$ be the MECs of $\mathcal{M}_\varrho$. Then $S_\mathcal{D} = \left(S \setminus \left(\cup_{j=1}^k S^j\right)\right) \uplus U \uplus Q \uplus \{t\}$, where $U = \{u_1, \ldots, u_k\}, Q = \{q_1, \ldots, q_k\}$. One new action is added to the set of actions: $\mathit{Act}_\mathcal{D} = \mathit{Act} \uplus \{\perp\}$. All states from the set Q have only one transition leading to state t: $\forall q \in Q : \mathbf{P}[q, \alpha, q'] = 1$ iff $\alpha = \perp, q' = t$ and is 0 otherwise. All states from set U have a transition leading to the respective q state and to other MECs: $\forall u_j \in U : \mathbf{P}[u_j, \perp, q_j] = 1$ and $\forall \tau = (s, \nu, \mu) \in \rightsquigarrow \setminus \rightsquigarrow^j, s \in S^j : \mathbf{P}[u_j, \nu, u_i] = \mu(s')$, where $s' \in S^i$. The reward structure assigns non-zero rewards only to states from set Q: $\forall q \in Q : \varrho_{\mathcal{D}}(q, \perp) = \overline{\text{lval}}^{\mathcal{M}^j_\varrho}_{\text{opt}}$ and in all other cases $\varrho_{\mathcal{D}}(s, \alpha) = 0$.

The connection between the long-run average values in $\mathcal{M}_\varrho$ and total reward values in $\mathcal{D}_\varrho$ is established as follows:

$$\forall s \in S : \text{lval}^{\mathcal{M}_\varrho}_{\text{opt}}(s) = \text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(f(s)), \text{ where } f(s) = \begin{cases} u_j & \text{if } s \in S^j \\ s & \text{otherwise} \end{cases}$$

One can see that steps 1 and 3 outlined above admit efficient solutions, while the algorithm for step 2 is based on linear programming. In the context of Markov decision processes algorithms for linear programming often do not scale well with the size of the problem, compared to iterative algorithms, such as value or policy iteration. So far, however, no iterative algorithm has been developed for long-run average rewards for Markov reward automata. In the next section, we fill this gap by presenting an iterative algorithm for the computation of long-run average rewards for MRA.

5.3 Iterative Approach

In this section, we present a new approach for quantifying the optimal long-run average reward for Markov reward automata. Recall that for steps 1 and 3 of the general case solution, shown in the previous section, there exist efficient algorithms. At step 2 one needs to compute the long-run average reward for each MEC of the MRA. This problem can be reduced to the computation of optimal long-run average reward for unichain MRA and so far only a solution based on linear programming is known for this problem. In this section we present an iterative approach for quantifying optimal long-run average rewards in unichain MRA and show how to integrate it into the general case solution to obtain an iterative solution for the general problem. When solving the former we will only consider those unichain MRA that have strongly connected underlying graph, since we only need a solution for MECs of a given MRA.

The core of our approach lies in the following observation: a Markov reward automaton can be considered as a compact representation of a possibly exponentially larger CTMDP. This observation enables us to use efficient algorithms available for CTMDPs [Put94] to compute optimal long-run average rewards. But since that CTMDP, in the worst case, has exponentially more transitions, this naïve approach does not seem promising. We circumvent this problem by means of classical dynamic programming and thereby arrive at an efficient solution that avoids the construction of the large CTMDP.

We start by formally defining the long-run average rewards in CTMDPs in Section 5.3.1. Next in Section 5.3.2 we show how to obtain a CTMDP that has the same long-run average reward value as a given unichain MRA. We describe ways to avoid construction of this potentially exponentially large CTMDP in Section 5.3.3.

All these results allow us to obtain an iterative algorithm that approximates long-run average rewards value, which we discuss in Section 5.3.4 and evaluate on a set of benchmarks in Section 5.3.5.

5.3.1 Long-run Rewards for CTMDPs

In this section we formally define optimal expected long-run average reward for CTMDPs.

Definition 5.3.1. *A* continuous-time Markov decision process (CTMDP) with rewards *is a tuple* $\mathcal{C}_\varrho = (S_\mathcal{C}, Act_\mathcal{C}, \mathbf{R}_\mathcal{C}, \varrho_{\text{st}}^\mathcal{C}, \varrho_{\text{trn}}^\mathcal{C})$*, where* $(S_\mathcal{C}, Act_\mathcal{C}, \mathbf{R}_\mathcal{C})$ *is a CTMDP,* $\varrho_{\text{st}}^\mathcal{C} : S_\mathcal{C} \to \mathbb{R}$ *is a* state reward function *and* $\varrho_{\text{trn}}^\mathcal{C} : S_\mathcal{C} \times Act_\mathcal{C} \to \mathbb{R}$ *is a* transition reward function*.*

Example 5.3.1. *Figure 5.2 shows an example of a CTMDP with rewards. Rewards are denoted in the same way as for MRA, i. e. next to the respective state or transition in a green frame. Here state* (high) *has a state reward* 0.4 *and state* (low) *has reward* 0.1*. Transition labelled with* high *has transition reward* 10*, the one labelled with* low *has transition reward* 2*, and transition* α *has reward* -1*.*

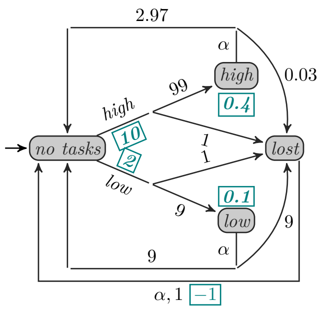


Figure 5.2: An example of a CTMDP with rewards.

For a path fragment $\phi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots s_n \xrightarrow{\alpha_n,t_n}$ the *total reward over* ϕ is defined as follows:

$$\text{rew}_{\mathcal{C}_\varrho}(\phi) := \sum_{i=0}^{n} \varrho_{\text{st}}^\mathcal{C}(s_i) \cdot t_i + \varrho_{\text{trn}}^\mathcal{C}(s_i, \alpha_i)$$

For $t \in \mathbb{R}_{\geqslant 0}$ the *prefix* of an infinite path $\rho = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} s_2 \cdots$ until time t is a path fragment or an infinite path, defined as follows:

$$\text{prefix}_t^\phi(\rho) := \begin{cases} \theta & \text{if } t_0 > t \\ s_0 \xrightarrow{\alpha_0,t_0} \cdots s_{n-1} \xrightarrow{\alpha_{n-1},t_{n-1}} & \text{if } \sum_{i=0}^{n-1} t_i \leqslant t, \sum_{i=0}^{n} t_i > t \\ \rho & \text{otherwise} \end{cases}$$

Let $s \in S_\mathcal{C}, \pi \in \Pi_\mu$. The *long-run average reward* is the random variable $L_{\mathcal{C}_\varrho,s,\pi} : Paths^\omega \to \mathbb{R}_{\geqslant 0}^\infty$ on the probability space $(Paths^\omega, \mathcal{P}^\omega, \Pr_{\pi,s}^\mathcal{C}[\cdot])$ defined as follows:

$$L_{\mathcal{C}_\varrho,s,\pi}(\rho) := \lim_{t \to \infty} \frac{1}{t} \text{rew}_{\mathcal{C}_\varrho}(\text{prefix}_t^\phi(\rho))$$

Definition 5.3.2. *Let* $\pi \in \Pi_\mu$ *and* $s \in S_\mathcal{C}$. *The* (expected) long-run average reward *and* optimal (expected) long-run average reward *are defined as follows:*

$$\begin{aligned} \mathrm{lval}_\pi^{\mathcal{C}_\varrho}(s) &:= \mathbb{E}\left[L_{\mathcal{C}_\varrho,s,\pi}\right] = \int_{Paths^\omega} L_{\mathcal{C}_\varrho,s,\pi}(\rho) \cdot \mathrm{Pr}_{\pi,s}^{\mathcal{C}}\left[\mathrm{d}\rho\right] \\ \mathrm{lval}_{\mathrm{opt}}^{\mathcal{C}_\varrho}(s) &:= \underset{\pi\in\Pi_\mu}{\mathrm{opt}}\ \mathrm{lval}_\pi^{\mathcal{C}_\varrho}(s) \end{aligned} \tag{5.6}$$

We denote with $\overline{\mathrm{lval}}_\pi^{\mathcal{C}_\varrho}$ the vector of values $\mathrm{lval}_\pi^{\mathcal{C}_\varrho}(s)$ for all states $s \in S_\mathcal{C}$ and analogously the vector of optimal values is denoted as $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{C}_\varrho}$. As shown in [Put94], for a unichain[1] CTMDP $\mathcal{C}$ we have $\forall s, s' \in S_\mathcal{C} : \mathrm{lval}_{\mathrm{opt}}^{\mathcal{C}_\varrho}(s) = \mathrm{lval}_{\mathrm{opt}}^{\mathcal{C}_\varrho}(s')$. In the following we will refer to this value as $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{C}_\varrho}$.

Computation of value $\mathrm{lval}_{\mathrm{opt}}^{\mathcal{C}_\varrho}(s)$ in unichain CTMDPs is a well-studied problem that is solved via a reduction to optimal long-run average rewards in discrete-time MDPs [Put94]. The latter can be solved exactly via linear programming or policy iteration or approximated with value iteration.

5.3.2 Value Preserving CTMDP

Our first step towards the iterative solution is showing that for any unichain Markov reward automaton there exists a unichain CTMDP with rewards that has the same optimal long-run average reward value. We first demonstrate the transformation on an example.

Example 5.3.2. *Consider the MRA $\mathcal{M}_\varrho$ depicted on the left of Fig. 5.3. The state-space of the value preserving CTMDP $\mathcal{C}_\varrho$, shown on the right of Fig. 5.3, is the set of Markovian states of $\mathcal{M}_\varrho$. If a Markovian state s of $\mathcal{M}_\varrho$ has outgoing transitions leading only to Markovian states, e. g. states s_1 and s_2, then $\mathcal{C}_\varrho$ preserves the transition rates, i. e. $\forall s' \in MS : \mathbf{R}_\mathcal{C}[s, \perp, s'] = \mathbf{R}[s, s']$. States that have an outgoing transition leading to a probabilistic state are treated differently. Consider state s_0. Let PS_{s_0} be all probabilistic states between s_0 and some other Markovian state. In our case $PS_{s_0} = \{p_0, p_1, p_2, p_3\}$. In $\mathcal{C}_\varrho$ state s_0 has as many actions as there are different mappings from state $p_i \in PS_{s_0}$ to one of its enabled actions. In other words, each enabled action of s_0 in $\mathcal{C}_\varrho$ corresponds to a stationary strategy over states from PS_{s_0}. For example, A_0 could be the following mapping: $A_0(p_0) = \alpha_0, A_0(p_1) = \alpha_1, A_0(p_2) = \gamma_2, A_0(p_3) = \gamma_3$. In this example state s_0 has $2^4 = 16$ actions in $\mathcal{C}_\varrho$.*

Next we discuss transition rates in $\mathcal{C}_\varrho$. The probability of moving from s to s' via action A in $\mathcal{C}_\varrho$ is the same as probability to reach s' when starting from s and following strategy A in $\mathcal{M}_\varrho$. The rate of the respective transition is obtained by multiplying this number with the exit rate of state s. Assume that all the distributions of outgoing transitions of states p_i are uniform. In this case $\mathbf{R}_\mathcal{C}[s_0, A_0, s_1] = (\lambda_1 + \lambda_2) \cdot \left[\frac{\lambda_1}{\lambda_1+\lambda_2}(0.5 \cdot 1 + 0.5 \cdot 0) + \frac{\lambda_2}{\lambda_1+\lambda_2}(1 \cdot 1)\right] = 0.5 \cdot \lambda_1 + \lambda_2$.

[1] The definition of a unichain CTMDP is analogous to that of a unichain Markov automaton.

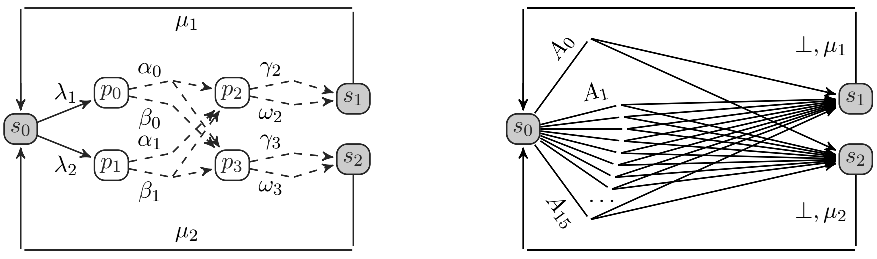


Figure 5.3: An example of an MRA (left) and its value preserving CTMDP (right). In this picture we omitted the probabilities of the probabilistic transitions of the MRA and the rewards of the MRA and the CTMDP.

Finally, state rewards of Markovian states are the same in $\mathcal{M}_\varrho$ and $\mathcal{C}_\varrho$. Since probabilistic states are not present in $\mathcal{C}_\varrho$, we need to store rewards of those transitions somewhere, and we will store them in Markovian states. Namely, transition reward for a state s and an action A in $\mathcal{C}_\varrho$ will be the expected transition reward gathered when starting from s and until reaching any other Markovian state in $\mathcal{M}_\varrho$ while following strategy A. For example, if the only non-zero rewards are reward of transition $p_0 \overset{\alpha_0}{\dashrightarrow} \mu$, which is 2, reward of transition $p_3 \overset{\gamma_3}{\dashrightarrow} \mu'$, which is 3, and reward of the Markovian transition $s_0 \xrightarrow{E(s)} \mu''$, which is 1, then reward of s_0 via action A_0 in $\mathcal{C}_\varrho$ is $\frac{\lambda_1}{\lambda_1+\lambda_2} \cdot \frac{1}{2} \cdot 1 \cdot (1+2) + \frac{\lambda_1}{\lambda_1+\lambda_2} \cdot \frac{1}{2} \cdot 1 \cdot (1+2+3) + \frac{\lambda_2}{\lambda_1+\lambda_2} \cdot 1 \cdot 1 \cdot (1)$.

We will next describe this transformation formally. Let $\mathcal{M}_\varrho$ be a unichain MRA. The value preserving CTMDP $\mathcal{C}_\varrho(\mathcal{M}_\varrho) = (S_\mathcal{C}, Act_\mathcal{C}, \mathbf{R}_\mathcal{C}, \varrho^\mathcal{C}_{\mathrm{st}}, \varrho^\mathcal{C}_{\mathrm{trn}})$ is obtained as follows:

State-Space: $S_\mathcal{C} := MS_\mathcal{M}$.

Action Space: Informally, an action for a CTMDP state $s \in S_\mathcal{C}$ will be a stationary strategy back in the MRA, that defines an action for each of the probabilistic states reachable from s and located between s and some other state from $S_\mathcal{C}$. Formally, we denote with PS_s the set of all probabilistic states $s' \in PS_\mathcal{M}$ that are reachable from s via the transition relation $\dashrightarrow$, i. e. $\exists s'' \in S : s'' \in post(s)$ and $s'' \dashrightarrow^* s'$, where $\dashrightarrow^*$ is the transitive closure of relation $\dashrightarrow$. We create functions A_s that will serve as an action for state s in the CTMDP $\mathcal{C}$. The function maps each state s' reachable from s to one of its enabled actions, i. e. if $PS_s \neq \emptyset$, then $A_s : PS_s \to Act$, s. t. $\forall s' \in PS_s : A_s(s') \in Act(s')$. If $PS_s = \emptyset$, then there are no probabilistic states reachable from s and we just create a new action $\perp$. Each function A_s can be thought of as a macro-action, defining an action for each state in PS_s. The set of all such functions for state s denotes all possible mappings of probabilistic states reachable from s to one of their enabled actions. Then the set of all enabled actions $Act_\mathcal{C}(s)$ for state s in $\mathcal{C}$ is the set of all possible functions A_s or action $\perp$ if $PS_s = \emptyset$, and $Act_\mathcal{C} = \bigcup_{s \in S_\mathcal{C}} Act_\mathcal{C}(s)$.

Transition matrix $\mathbf{R}_{\mathcal{C}}$. If state s has no probabilistic successors, then state s has only one enabled action $\perp$ and matrix $\mathbf{R}_{\mathcal{C}}$ just repeats matrix $\mathbf{R}$: If $Act_{\mathcal{C}}(s) = \{\perp\}$, then $\mathbf{R}_{\mathcal{C}}[s, \perp, s'] = \mathbf{R}[s, s']$. If $Act_{\mathcal{C}}(s) \neq \{\perp\}$, then

$$\forall A_s \in Act_{\mathcal{C}}(s), s' \in S_{\mathcal{C}} : \mathbf{R}_{\mathcal{C}}[s, A_s, s'] := E(s) \cdot \Pr^{\mathcal{M}}_{A_s,s}\left[Cyl_{\mathcal{M}}(E(s, s'))\right],$$

where

$$E(s, s') = \{\rho \in Paths^*(\mathcal{M}) \mid |\rho| \geqslant 1, \rho[0] = s, \rho{\downarrow} = s' \text{ and}$$
$$\text{if } |\rho| > 1 \text{ then } \forall 1 \leqslant i \leqslant |\rho| - 1 : \rho[i] \in PS_{\mathcal{M}}\}$$

The value $\Pr^{\mathcal{M}}_{A_s,s}[E(s, s')]$ denotes the probability when using scheduler A_s to reach state s' from state s via one Markovian transition from s and only probabilistic transitions afterwards.

State Rewards remain unaffected $\varrho^{\mathcal{C}}_{\text{st}}(s) := \varrho_{\text{st}}(s)$.

Transition Rewards are slightly more involved. Essentially, transition reward for action A of state s in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ combines transition reward of the outgoing Markovian transition of s and transition rewards of all the probabilistic states that are in the MRA, but are not present in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. Formally, let $E^{UT}(s, s')$ be the set of *untimed paths* from $E(s, s')$, i. e. $\rho^{UT} = s \xrightarrow{\perp} s_1 \xrightarrow{A_s(s_1)} \cdots s_n \xrightarrow{A_s(s_n)} s' \in E^{UT}(s, s')$ iff $\rho = s \xrightarrow{\perp, 0} s_1 \xrightarrow{A_s(s_1), t_1} \cdots s_n \xrightarrow{A_s(s_n), t_n} s' \in E(s, s')$. Then $\forall s \in S_{\mathcal{C}}, A \in Act_{\mathcal{C}}(s)$:

$$\varrho^{\mathcal{C}}_{\text{trn}}(s, A) := \sum_{s' \in S_{\mathcal{C}}} \sum_{\rho^{UT} \in E^{UT}(s, s')} \mathbb{P}_A[\rho^{UT}] \cdot r_A[\rho^{UT}],$$

where for $\rho^{UT} = s \xrightarrow{\perp} s_1 \xrightarrow{A(s_1)} \cdots s_n \xrightarrow{A(s_n)} s'$:

$$\mathbb{P}_A[\rho^{UT}] = \mathbb{P}[s, s_1] \cdot \mathbb{P}[s_1, A(s_1), s_2] \cdots \mathbb{P}[s_n, A(s_n), s']$$
$$r_A[\rho^{UT}] = \varrho_{\text{trn}}((s, E(s), \mu_0)) + \varrho_{\text{trn}}((s_1, A(s_1), \mu_1)) + \cdots + \varrho_{\text{trn}}((s_n, A(s_n), \mu_n))$$

The action reward for a state s and an action A in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ is therefore the transition reward expected to be accumulated over all untimed paths from s to some $s' \in S_{\mathcal{C}}$.

Lemma 5.3.1. *Let $\mathcal{M}_\varrho$ be a Markov reward automaton and $\mathcal{C}_\varrho(\mathcal{M}_\varrho) = (S_{\mathcal{C}}, Act_{\mathcal{C}}, \mathbf{R}_{\mathcal{C}}, \varrho^{\mathcal{C}}_{\text{st}}, \varrho^{\mathcal{C}}_{\text{trn}})$ is the CTMDP obtained as described above. Then*

$$\overline{\text{lval}}^{\mathcal{M}_\varrho}_{\text{opt}} = \overline{\text{lval}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}_{\text{opt}}$$

Proof. First of all we will define a mapping between schedulers in $\mathcal{M}_\varrho$ and $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. Let Π^{MS} be a set of schedulers in $\mathcal{M}_\varrho$ that depend only on the last state on the path and on the last Markovian state on this path, if any. Such a scheduler $\pi \in \Pi^{MS}$ can be defined as $\pi : MS \uplus \{\perp\} \times PS \to Act$, where the first component equals $\perp$ if there is no Markovian state on the path.

Let $\pi \in \Pi^{MS}$ be a strategy in $\mathcal{M}_\varrho$. We will denote with $f[\pi]$ a stationary strategy in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ obtained as follows:

$$\forall s \in S_\mathcal{C} :$$
$$f[\pi](s) = \begin{cases} \bot & \text{if } PS_s = \emptyset \\ A, \text{ s.t.} \forall ps \in PS_s : A(ps) = \pi(s, ps) & \text{otherwise} \end{cases}$$

The inverse of mapping f denotes a mapping from a stationary strategy in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ to a set of strategies in $\mathcal{M}_\varrho$ from the set Π^{MS}. Let $\pi_\mathcal{C}$ be a stationary strategy in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$, then $f^{-1}[\pi_\mathcal{C}]$ is a set of strategies $\pi \in \Pi^{MS}$ in $\mathcal{M}_\varrho$ that satisfy the following:

$$\forall ms \in MS, ps \in PS_{ms}, \pi \in f^{-1}[\pi_\mathcal{C}] : \pi(ms, ps) = A(ps), \text{ where } A = \pi_\mathcal{C}(ms)$$

Mapping $f^{-1}[\pi_\mathcal{C}]$ is a set because a stationary strategy of $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ does not define any behaviour on the paths of $\mathcal{M}_\varrho$ that did not reach any Markovian state yet, i. e. schedulers $\pi \in f^{-1}[\pi_\mathcal{C}]$ may differ in decisions for the situations when the first component is $\bot$.

Notice that both mappings $f[\cdot]$ and $f^{-1}[\cdot]$ are surjective, i, e. each scheduler $\pi \in \Pi^{MS}$ is mapped to at least one stationary scheduler $\pi_\mathcal{C} \in \Pi_{\mathtt{stat}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}$ and for each stationary scheduler $\pi_\mathcal{C} \in \Pi_{\mathtt{stat}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}$ there exists a scheduler $\pi \in \Pi^{MS}$ that is mapped to it. Taking into account that optimal schedulers in both $\mathcal{M}_\varrho$ and $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ are stationary and $\Pi_{\mathtt{stat}}^{\mathcal{M}_\varrho} \subseteq \Pi^{MS}$, it suffices to prove either of the following statements:

$$\forall \pi \in \Pi^{MS}, s \in MS : \mathrm{lval}_\pi^{\mathcal{M}_\varrho}(s) = \mathrm{lval}_{f[\pi]}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}(s) \tag{5.7}$$

Or in the other direction:

$$\forall \pi_\mathcal{C} \in \Pi_{\mathtt{stat}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}, \pi \in f^{-1}[\pi_\mathcal{C}], s \in S_\mathcal{C} : \mathrm{lval}_{\pi_\mathcal{C}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}(s) = \mathrm{lval}_\pi^{\mathcal{M}_\varrho}(s)$$

We will prove the former and start by showing that the stochastic process induced by a scheduler $\pi \in \Pi^{MS}$ on Markovian states of $\mathcal{M}_\varrho$ coincides with the one induced by $f[\pi]$ on states $S_\mathcal{C}$ of $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$.

First of all, the exit rate of state $s \in MS$ coincides with the exit rate of s and any other action $A \in Act_\mathcal{C}(s)$ in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. Consider $s \in MS$. If $A = \bot$, then s has no probabilistic successors and by definition:

$$E(s, \bot) = \sum_{s' \in S_\mathcal{C}} \mathbf{R}_\mathcal{C}[s, \bot, s'] = \sum_{s' \in S_\mathcal{C}} \mathbf{R}[s, s'] = \sum_{s' \in S} \mathbf{R}[s, s'] = E(s)$$

If $A \neq \bot$, then:

$$\begin{aligned} E(s, A) &= \sum_{s' \in S_\mathcal{C}} \mathbf{R}_\mathcal{C}[s, A, s'] = \sum_{s' \in S_\mathcal{C}} E(s) \cdot \mathrm{Pr}_{A,s}^{\mathcal{M}} \left[Cyl_\mathcal{M}(E(s, s')) \right] \\ &= E(s) \sum_{s' \in S_\mathcal{C}} \mathrm{Pr}_{A,s}^{\mathcal{M}} \left[Cyl_\mathcal{M}(E(s, s')) \right] \end{aligned}$$

Since $\mathcal{M}_\varrho$ is non-Zeno, then all paths starting from a Markovian state reach another Markovian state with probability 1, therefore $\sum_{s' \in S_\mathcal{C}} \mathrm{Pr}_{A,s}^{\mathcal{M}} \left[Cyl_\mathcal{M}(E(s, s')) \right] = 1$. Thus also in this case $E(s, A) = E(s)$.

Next we show that the probability induced by $\pi \in \Pi^{MS}$ in $\mathcal{M}_\varrho$ to transition from s to a Markovian state s' is the same as the distribution $\mathbb{P}[s, f[\pi](s), s']$ in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. Let $s \in MS$. If s has no probabilistic successors then $\mathbf{R}_\mathcal{C}[s, \perp, s'] = \mathbf{R}[s, s']$ and therefore the probability distribution over successor states (which are only Markovian) is the same in $\mathcal{M}_\varrho$ and $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. If s has probabilistic successors, then

$$\begin{aligned}\Pr^{\mathcal{M}}_{\pi,s}\left[Cyl_{\mathcal{M}}(E(s,s'))\right] &= \frac{E(s)\cdot \Pr^{\mathcal{M}}_{\pi,s}\left[Cyl_{\mathcal{M}}(E(s,s'))\right]}{E(s, f[\pi](s))} \\ &= \frac{\mathbf{R}_\mathcal{C}[s, f[\pi](s), s']}{E(s, f[\pi](s))} = \mathbb{P}[s, f[\pi](s), s']\end{aligned}$$

Thus the probability to move to state $s' \in MS$ starting from s via only probabilistic intermediate states and when following strategy π in $\mathcal{M}_\varrho$ is the same as the probability to transition to s' starting from s and following strategy $f[\pi]$ in $\mathcal{C}_\varrho(\pi)$.

We have thus shown that the stochastic process induced by $\pi \in \Pi^{MS}$ over Markovian states of $\mathcal{M}_\varrho$ is the same as the stochastic process induced by $f[\pi]$ on states of $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. Next we will show that the rewards accumulated in both processes also coincide. We consider state rewards and transition rewards separately.

We start with expected state rewards. Expected state reward accumulated from a Markovian state s in $\mathcal{M}_\varrho$ over all paths is $\varrho_{\text{st}}(s)/E(s)$. Since exit rate of state s in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ is the same for each action and equals $E(s)$, then expected state reward of state s in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ is $\varrho^{\mathcal{C}}_{\text{st}}(s)/E(s) = \varrho_{\text{st}}(s)/E(s)$. Residence time in probabilistic states is 0 and therefore expected state reward accumulated from a probabilistic state s is $\varrho_{\text{st}}(s)\cdot 0 = 0$. Thus expected state reward accumulated in both processes coincide.

We move to expected transition rewards. In the following we denote a path $\rho \in E(s,s')$ as follows $\rho = ((s_0, \nu_0, \mu_0), t_0) \cdots ((s_n, \nu_n, \mu_n), t_n) \cdot s_{n+1}$, where $s_0 = s, s_{n+1} = s', n \in \mathbb{Z}_{\geqslant 0}$. Similarly, $\rho^{UT} = ((s_0, \nu_0, \mu_0)) \cdots ((s_n, \nu_n, \mu_n)) \cdot s_{n+1}$, where $s_0 = s, s_{n+1} = s'$. Consider expected transition reward collected when following strategy $\pi \in \Pi^{MS}$ starting from a Markovian state s and until the first visit to some other Markovian state (i. e. over finite paths in $\cup_{s' \in MS} E(s,s')$) in $\mathcal{M}_\varrho$:

$$\begin{aligned}&\sum_{s'\in MS}\int_{\rho\in E(s,s')}\sum_{i=0}^{|\rho|-1}\varrho_{\text{trn}}\left(s_i \overset{\nu_i}{\rightsquigarrow} \mu_i\right)\cdot \Pr^{\mathcal{M}}_{\pi,s}[\mathrm{d}\rho] \\ &= \sum_{s'\in MS}\sum_{\rho^{UT}\in E^{UT}(s,s')}\sum_{i=0}^{|\rho^{UT}|-1}\varrho_{\text{trn}}\left(s_i \overset{\nu_i}{\rightsquigarrow} \mu_i\right)\cdot \mathbb{P}[s_0,s_1]\cdot \prod_{j=1}^{|\rho^{UT}|-1}\mathbb{P}[s_j, \pi(s,s_j), s_{j+1}] \\ &= \sum_{s'\in MS}\sum_{\rho^{UT}\in E^{UT}(s,s')} r_{f[\pi](s)}(\rho^{UT})\cdot \mathbb{P}_{f[\pi](s)}[\rho^{UT}] \\ &= \varrho^{\mathcal{C}}_{\text{trn}}(s, f[\pi](s))\end{aligned}$$

Thus the expected reward accumulated over transitions in $\mathcal{M}_\varrho$ after leaving a Markovian state s until encountering some Markovian state s' while following strategy π is the same as the expected transition reward accumulated in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ when leaving state s via action dictated by $f[\pi]$.

This concludes the proof of (5.7).

□

Remark 5.3.1. *Notice that a stationary scheduler that is optimal for* $\overline{\text{lval}}_{\text{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}$ *may not be stationary w. r. t.* $\mathcal{M}_\varrho$*. For example, consider the situation in which two Markovian states* ms_1 *and* ms_2 *can reach the same probabilistic state ps. Both Markovian states become states of CTMDP* $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$*. If state ps has at least two enabled actions in* $\mathcal{M}_\varrho$*, then there exists an enabled action* $A \in Act_{\mathcal{C}}(ms_1)$ *and* $B \in Act_{\mathcal{C}}(ms_2)$*, such that* $A(ps) \neq B(ps)$*. If an optimal strategy* $\pi_{\mathcal{C}}$ *for* $\overline{\text{lval}}_{\text{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}$ *chooses action A for* ms_1 *and action B for* ms_2*, then there exists no stationary strategy* π *in* $\mathcal{M}_\varrho$ *that takes the same decisions as* $\pi_{\mathcal{C}}$*. However [GTH+ 14] shows that there exists an optimal strategy for* $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}$ *that is stationary. Therefore there exists an optimal strategy* $\pi_{\mathcal{C}}$ *for* $\overline{\text{lval}}_{\text{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}$ *that is stationary and such that in situations when the same probabilistic state ps is reachable by multiple Markovian states, this strategy takes the same decision for ps, i. e. for the example above* $A(ps) = B(ps)$*.*

It is easy to see that even for toy models the number of transitions of the CTMDP $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ can grow extremely fast. If a Markovian state s in $\mathcal{M}_\varrho$ can reach n probabilistic states (i, e. $|PS_s| = n$), and each of those states has at least two enabled actions, then the set of enabled actions $Act_{\mathcal{C}}(s)$ of s in $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ is 2^n. This growth is therefore exponential in the worst case. This means that there may exist such MRA, for which the value preserving CTMDP is sub-exponential (e. g. polynomial) in the size of the MRA. And if this is the case, one can simply apply existing CTMDP algorithms to compute value $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho} = \overline{\text{lval}}_{\text{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}$ (see Section 5.3.1 of this chapter) and skip the rest of this chapter. For those cases when the CTMDP model preserving the long-run average value of an MRA is indeed way too large to be efficiently analysed, we designed efficient techniques that utilise the structure of the MRA and dynamic programming. This will be described in the next sections.

5.3.3 Dealing with Exponentiality

In this section, we will develop a simple yet efficient solution to cope with exponentiality, harvesting the Bellman equation for CTMDPs [Put94] together with the structure of $\mathcal{M}_\varrho$. The Bellman equation for CTMDPs looks as follows:

Lemma 5.3.2 ([Put94]). *Let $\mathcal{C} = (S_{\mathcal{C}}, Act_{\mathcal{C}}, \mathbf{R}_{\mathcal{C}}, \varrho_{\mathrm{st}}^{\mathcal{C}}, \varrho_{\mathrm{trn}}^{\mathcal{C}})$ be a unichain CT-MDP with rewards. Let $\eta > \mathbf{E}_{\max}$, then there exists a function $h : S_{\mathcal{C}} \to \mathbb{R}$ and value $a \in \mathbb{R}_{\geqslant 0}$ that are a solution to the Bellman equation:*

$\forall s \in S_{\mathcal{C}}$:

$$h(s) + \frac{a}{\eta} = \underset{\alpha \in Act_{\mathcal{C}}(s)}{\mathrm{opt}} \left\{ \frac{\varrho_{\mathrm{trn}}^{\mathcal{C}}(s,\alpha) \cdot E(s,\alpha) + \varrho_{\mathrm{st}}^{\mathcal{C}}(s)}{\eta} + \sum_{s' \in S_{\mathcal{C}}} \mathbf{P}_{\mathcal{C}}[s,\alpha,s'] \cdot h(s') \right\} \tag{5.8}$$

where

$$\mathbf{P}_{\mathcal{C}}[s,\alpha,s'] := \begin{cases} \frac{\mathbf{R}_{\mathcal{C}}[s,\alpha,s']}{\eta} & \textit{if } s' \neq s \\ 1 - \frac{(E(s,\alpha) - \mathbf{R}_{\mathcal{C}}[s,\alpha,s])}{\eta} & \textit{if } s' = s \end{cases} \tag{5.9}$$

Moreover $a = \overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{C}_{\varrho}}$.

A naïve direct application of this Bellman equation to CTMDP $\mathcal{C}_{\varrho}(\mathcal{M}_{\varrho})$ is problematic in the right-hand side of (5.8). As we established at the end of the previous section, the optimisation step required for computing the right-hand side may have to range over exponentially many actions. Left untreated, this operation, in essence, is a brute force check of optimality of each stationary strategy in $\mathcal{M}_{\varrho}$ (each of those strategies induces an action in the CTMDP). In the following, we will show how to avoid this problem by working with $\mathcal{M}_{\varrho}$ itself and not with $\mathcal{C}_{\varrho}(\mathcal{M}_{\varrho})$. Informally, we will show that the value on the right-hand side of (5.8) is nothing more than the optimal expected unbounded reward for a certain MDP. We start with defining this MDP.

Terminal MDP and its Reward Structure. Let $\eta > \mathbf{E}_{\max}$. The two-step procedure below constructs an MDP that will be used for an efficient solution of the optimisation problem on the right-hand side of (5.8).

1. At first we obtain the MDP $\mathcal{D}_{\eta} = (S, Act \uplus \{\perp\}, \mathbf{P}_{\eta})$. This MDP keeps all probabilistic states of $\mathcal{M}_{\varrho}$ and their actions the way they are in $\mathcal{M}_{\varrho}$. For each Markovian state $s \in MS$ the MDP contains state s that has only action $\perp$ enabled. The probability distribution for s and $\perp$ is the distribution over successors of s after it has been *uniformised with rate* η (i. e. a self-loop transition with rate $\eta - E(s)$ is added to the Markovian transition relation):

$$\mathbf{P}_{\eta}[s,\alpha,s'] := \begin{cases} \mathbb{P}[s,\alpha,s'] & \text{if } s \in PS, \alpha \in Act(s) \\ \mathbf{R}[s,s']/\eta & \text{if } s \in MS, \alpha = \perp, s' \neq s \\ 1 - \frac{E(s) - \mathbf{R}[s,s]}{\eta} & \text{if } s \in MS, \alpha = \perp, s' = s \\ 0 & \text{in all other cases} \end{cases}$$

An example of the MDP $\mathcal{D}_{\eta}$ for the MRA depicted in Fig. (5.4a) is shown in Fig. (5.4b).

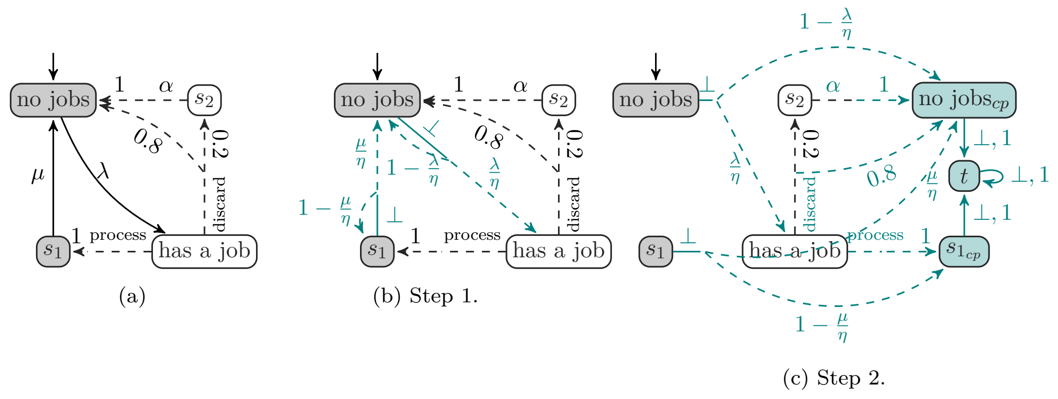


Figure 5.4: Construction of the terminal MDP with uniformisation rate η. Figure (5.4a) depicts an example MRA. The result of the first step of the transformation is shown in figure (5.4b), and the second step is depicted in (5.4c).

2. Next, for each Markovian state we introduce a copy state and redirect all the transitions leading to Markovian states to the respective new copy states. Additionally, we introduce a terminal state t, that has only one self-loop transition. Let $\mathcal{D}_\lambda = (S, Act \uplus \{\perp\}, \mathbf{P}_\eta)$ be the MDP obtained in the previous step, then $t\mathcal{D}(\eta) := (S', Act \uplus \{\perp\}, \mathbf{P}')$, where $S' = S \uplus S_{cp} \uplus \{t\}$, $S_{cp} = \{s_{cp} \mid s \in MS\}$ and

$$\mathbf{P}'[s,\alpha,s'] = \begin{cases} \mathbf{P}_\eta[s,\alpha,p] & \text{if } s' = p_{cp} \in S_{cp}, \alpha \in Act \uplus \{\perp\} \\ \mathbf{P}_\eta[s,\alpha,s'] & \text{if } s' \in PS, \alpha \in Act \uplus \{\perp\} \\ 1 & \text{if } s \in S_{cp}, s' = t, \alpha = \perp \\ 1 & \text{if } s, s' = t, \alpha = \perp \\ 0 & \text{in all other cases} \end{cases}$$

We will call MDP $t\mathcal{D}(\eta)$ *terminal*, because all the paths of this MDP reach the terminal state with probability 1. Fig. (5.4c) shows an example of MDP $t\mathcal{D}(\eta)$ for the MDP $\mathcal{D}_\eta$ depicted in Fig. (5.4b).

Terminal MDP $t\mathcal{D}(\eta) = (S', Act \uplus \{\perp\}, \mathbf{P}')$ will be used along with the reward structure $\varrho^h_{t\mathcal{D}(\eta)}$ defined for $h : MS \to \mathbb{R}$ as follows:

$$\varrho^h_{t\mathcal{D}(\eta)}(s,\alpha) := \begin{cases} \varrho_{\text{trn}}(\tau) & \text{if } s \in PS, \alpha \in Act(s), \tau = (s,\alpha,\mu) \in \dashrightarrow \\ (\varrho_{\text{st}}(s) + E(s) \cdot \varrho_{\text{trn}}(\tau))/\eta & \text{if } s \in MS, \alpha = \perp, \tau = (s, E(s), \mu) \in \longrightarrow \\ h(ms) & \text{if } s = ms_{cp} \in S_{cp}, \alpha = \perp \\ 0 & \text{in all other cases} \end{cases}$$

A Better Bellman Equation. The following lemma shows that the right-hand side of the Bellman equation in (5.8) is the optimal expected unbounded reward in MDP $t\mathcal{D}(\eta)$ for reward structure $\varrho^h_{t\mathcal{D}(\eta)}$:

Lemma 5.3.3. *Let $\mathcal{C}_\varrho(\mathcal{M}_\varrho) = (S_\mathcal{C}, Act_\mathcal{C}, \mathbf{R}_\mathcal{C}, \varrho^\mathcal{C}_{\text{st}}, \varrho^\mathcal{C}_{\text{trn}})$, $\eta > \mathbf{E}_{\max}$, $h : S_\mathcal{C} \to \mathbb{R}$ and $\mathcal{D}_\varrho = (t\mathcal{D}(\eta), \varrho^h_{t\mathcal{D}(\eta)})$, where $t\mathcal{D}(\eta) = (S_\mathcal{D}, Act_\mathcal{D}, \mathbf{P}_\mathcal{D})$. Then*

$\forall s \in S_\mathcal{C}$:

$$\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s) = \frac{\varrho^\mathcal{C}_{\text{st}}(s)}{\eta} + \underset{A \in Act_\mathcal{C}(s)}{\text{opt}} \left\{ \frac{\varrho^\mathcal{C}_{\text{trn}}(s, A) \cdot E(s, A)}{\eta} + \sum_{s' \in S_\mathcal{C}} \mathbf{P}_\mathcal{C}[s, A, s'] \cdot h(s') \right\} \tag{5.10}$$

Proof. For a state $s, s' \in MS$ we denote with $Paths^\omega_{s,s'}(\mathcal{D}_\varrho)$ the set of infinite paths ρ in $\mathcal{D}_\varrho$ that start from state s and eventually reach state t by passing through state s', i. e. $\rho[0] = s$ and $\exists n \in \mathbb{Z}_{>0} : \rho[n] = t, \rho[n-1] = s'_{cp}$. Since $\mathcal{M}_\varrho$ is non-Zeno, then in $\mathcal{D}_\varrho$ the measure of set $\cup_{s' \in MS} Paths^\omega_{s,s'}(\mathcal{D}_\varrho)$ is 1. Additionally, optimal schedulers for $\overline{\text{urew}}^{\mathcal{D}_\varrho}_{\text{opt}}$ are stationary. Therefore for $s \in MS$ we can rewrite value $\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s)$ as follows:

$$\begin{aligned}
&\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(s) \\
&= \underset{\pi \in \Pi^{\mathcal{D}_\varrho}_\mu}{\text{opt}} \int_{Paths^\omega(\mathcal{D}_\varrho)} C^{unb}_{\mathcal{D}_\varrho, s, \pi}(\rho) \cdot \Pr^{\mathcal{D}_\varrho}_{\pi, s}[\mathrm{d}\rho] \\
&= \underset{\pi \in \Pi^{\mathcal{D}_\varrho}_{\text{stat}}}{\text{opt}} \sum_{s' \in MS} \sum_{\rho \in Paths^\omega_{s,s'}(\mathcal{D}_\varrho)} \prod_{j=0}^{k(\rho)-1} \mathbf{P}_\mathcal{D}[s_j, \pi(s_j), s_{j+1}] \sum_{i=0}^{k(\rho)-1} \varrho^h_{t\mathcal{D}(\eta)}(s_i, \pi(s_i)) \\
&= \varrho^h_{t\mathcal{D}(\eta)}(s, \bot) + \\
&\quad \underset{\pi \in \Pi^{\mathcal{D}_\varrho}_{\text{stat}}}{\text{opt}} \sum_{s' \in MS} \sum_{\rho \in Paths^\omega_{s,s'}(\mathcal{D}_\varrho)} \underbrace{\prod_{j=0}^{k(\rho)-1} \mathbf{P}_\mathcal{D}[s_j, \pi(s_j), s_{j+1}]}_{\mathbb{P}^\mathcal{D}_\pi[\rho]} \underbrace{\sum_{i=1}^{k(\rho)-1} \varrho^h_{t\mathcal{D}(\eta)}(s_i, \pi(s_i))}_{r^\mathcal{D}_\pi[\rho]} \\
&= \frac{\varrho^\mathcal{C}_{\text{st}}(s) + E(s) \cdot \varrho_{\text{trn}}(\tau)}{\eta} + \underset{\pi \in \Pi^{\mathcal{D}_\varrho}_{\text{stat}}}{\text{opt}} \sum_{s' \in MS} \sum_{\rho \in Paths^\omega_{s,s'}(\mathcal{D}_\varrho)} \mathbb{P}^\mathcal{D}_\pi[\rho] \cdot r^\mathcal{D}_\pi[\rho]
\end{aligned}$$

where $\forall j \in \mathbb{Z}_{\geqslant 0} : s_j = \rho[j]$, $k(\rho) \geqslant 2$ is the minimal index i, such that $\rho[i] = t$, and $\tau = s \xrightarrow{E(s)} \mu$.

Next we will rewrite the right-hand side of (5.10). By definition

$$\varrho^\mathcal{C}_{\text{trn}}(s, A) := \sum_{s' \in MS} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}] \cdot r_A[\rho^{UT}]$$

If $s \neq s'$:

$$\mathbf{P}_\mathcal{C}[s, A, s'] = \frac{E(s)}{\eta} \cdot \Pr^\mathcal{M}_{A,s}\left[Cyl_\mathcal{M}(E(s, s'))\right] = \frac{E(s)}{\eta} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}]$$

If $s = s'$:

$$\begin{aligned}\mathbf{P}_{\mathcal{C}}[s,A,s] &= 1 - \frac{(E(s) - \mathbf{R}_{\mathcal{C}}[s,A,s])}{\eta} = 1 - \frac{E(s)}{\eta} \cdot \left(1 - \Pr^{\mathcal{M}}_{A,s}\left[Cyl_{\mathcal{M}}(E(s,s))\right]\right) \\ &= 1 - \frac{E(s)}{\eta} \cdot \left(1 - \sum_{\rho^{UT} \in E^{UT}(s,s)} \mathbb{P}_A[\rho^{UT}]\right)\end{aligned}$$

As established in the proof of Lemma 5.3.1: $\forall A \in Act_{\mathcal{C}}(s) : E(s,A) = E(s)$. Therefore

$$\begin{aligned}&\frac{\varrho^{\mathcal{C}}_{\mathrm{trn}}(s,A) \cdot E(s,A)}{\eta} + \sum_{s' \in S_{\mathcal{C}}} \mathbf{P}_{\mathcal{C}}[s,A,s'] \cdot h(s') \\ &= \sum_{\substack{s' \in MS \\ s' \neq s}} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}] \cdot \frac{E(s)}{\eta} \cdot \left(r_A[\rho^{UT}] + h(s')\right) \\ &\quad + \frac{E(s)}{\eta} \cdot \sum_{\rho^{UT} \in E^{UT}(s,s)} \mathbb{P}_A[\rho^{UT}] \cdot r_A[\rho^{UT}] + \mathbf{P}_{\mathcal{C}}[s,A,s] \cdot h(s) \\ &= \sum_{\substack{s' \in MS \\ s' \neq s}} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}] \cdot \frac{E(s)}{\eta} \cdot \left(r_A[\rho^{UT}] + h(s')\right) \\ &\quad + \left(1 - \frac{E(s)}{\eta}\right) \cdot h(s) + \frac{E(s)}{\eta} \cdot \sum_{\rho^{UT} \in E^{UT}(s,s)} \mathbb{P}_A[\rho^{UT}] \cdot \left(r_A[\rho^{UT}] + h(s)\right) \\ &= \sum_{s' \in MS} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}] \cdot \frac{E(s)}{\eta} \cdot \left(r_A[\rho^{UT}] + h(s')\right) + \left(1 - \frac{E(s)}{\eta}\right) \cdot h(s) \\ &= \sum_{s' \in MS} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}] \cdot \frac{E(s)}{\eta} \cdot \left(r'_A[\rho^{UT}] + h(s')\right) + \left(1 - \frac{E(s)}{\eta}\right) \cdot h(s) \\ &\quad + \frac{E(s) \cdot \varrho_{\mathrm{trn}}(\tau)}{\eta},\end{aligned}$$

where $r'_A[\rho^{UT}] = r_A[\rho^{UT}] - \varrho_{\mathrm{trn}}(\tau)$.

Taking into account that for each $s \in MS$ a stationary scheduler in $\Pi^{\mathcal{D}_\varrho}_{\mathtt{stat}}$ can be considered as an action from set $Act_{\mathcal{C}}(s)$ and vice versa an action $A \in Act_{\mathcal{C}}(s)$ defines a stationary scheduler for paths from $Paths^{\omega}_{s}(\mathcal{D}_\varrho)$, it suffices to prove that $\forall A \in Act_{\mathcal{C}}(s)$:

$$\begin{aligned}&\sum_{s' \in MS} \sum_{\rho \in Paths^{\omega}_{s,s'}(\mathcal{D}_\varrho)} \mathbb{P}^{\mathcal{D}}_A[\rho] \cdot r^{\mathcal{D}}_A[\rho] = \\ &\sum_{s' \in MS} \sum_{\rho^{UT} \in E^{UT}(s,s')} \mathbb{P}_A[\rho^{UT}] \cdot \frac{E(s)}{\eta} \cdot \left(r'_A[\rho^{UT}] + h(s')\right) + \left(1 - \frac{E(s)}{\eta}\right) \cdot h(s)\end{aligned} \tag{5.11}$$

For $\rho \in Paths^{\omega}_{s,s'}(\mathcal{D}_\varrho)$, where $s' \neq s$:

$$\mathbb{P}^{\mathcal{D}}_A[\rho] = \frac{\mathbf{R}[s,s_1]}{\eta} \cdot \mathbb{P}[s_1, A(s_1), s_2] \cdots \mathbb{P}[s_n, A(s_n), s']$$

$$= \frac{E(s)}{\eta} \cdot \frac{\mathbf{R}[s, s_1]}{E(s)} \cdot \mathbb{P}[s_1, A(s_1), s_2] \cdots \mathbb{P}[s_n, A(s_n), s']$$
$$= \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}],$$

where ρ^{UT} is the respective path in $E^{UT}(s, s')$. Notice that for $s \neq s'$ each path $\rho \in \mathit{Paths}^{\omega}_{s,s'}(\mathcal{D}_{\varrho})$ corresponds to exactly one path $\rho^{UT} \in E^{UT}(s, s')$ and vice versa. If $\rho = s \xrightarrow{\nu_0} s_1 \cdots s_n \xrightarrow{\nu_n} s'_{cp} \xrightarrow{\perp} t \cdots \xrightarrow{\perp} t \cdots$, then $\rho^{UT} = s \xrightarrow{\nu_0} s_1 \cdots s_n \xrightarrow{\nu_n} s'$ and analogously in the other direction.

Next we consider transition rewards. For $\rho \in \mathit{Paths}^{\omega}_{s,s'}(\mathcal{D}_{\varrho})$ and respective $\rho^{UT} \in E^{UT}(s, s')$:

$$r^{\mathcal{D}}_A[\rho] = h(s') + \underbrace{\sum_{i=1}^{k(\rho)-2} \varrho_{\text{trn}}\left((s_i, A(s_i), \mu_i)\right)}_{=0 \text{ if } k(\rho)=2} = h(s') + r'_A[\rho^{UT}],$$

where μ_i is the distribution of the respective transition. Therefore

$$\sum_{\substack{s \in MS \\ s \neq s'}} \sum_{\rho \in \mathit{Paths}^{\omega}_{s,s'}(\mathcal{D}_{\varrho})} \mathbb{P}^{\mathcal{D}}_A[\rho] \cdot r^{\mathcal{D}}_A[\rho]$$
$$= \sum_{\substack{s \in MS \\ s \neq s'}} \sum_{\rho^{UT} \in E^{UT}(s,s')} \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}] \cdot (r'_A[\rho^{UT}] + h(s'))$$

Next we consider the case of $s' = s$. For paths $\rho \in \mathit{Paths}^{\omega}_{s,s}(\mathcal{D}_{\varrho})$ that pass through at least one probabilistic state, everything above still applies:

$$\sum_{\substack{\rho \in \mathit{Paths}^{\omega}_{s,s}(\mathcal{D}_{\varrho}) \\ k(\rho) > 2}} \mathbb{P}^{\mathcal{D}}_A[\rho] \cdot r^{\mathcal{D}}_A[\rho]$$
$$= \sum_{\substack{\rho^{UT} \in E^{UT}(s,s) \\ |\rho^{UT}| > 1}} \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}] \cdot (r'_A[\rho^{UT}] + h(s'))$$

What is left is the path $\rho = s \xrightarrow{\perp} s_{cp} \xrightarrow{\perp} t \xrightarrow{\perp} \cdots \xrightarrow{\perp} t \in \mathit{Paths}^{\omega}_{s,s}(\mathcal{D}_{\varrho})$:

$$\mathbb{P}^{\mathcal{D}}_A[\rho] \cdot r^{\mathcal{D}}_A[\rho] = (1 - \frac{E(s) - \mathbf{R}[s, s]}{\eta}) \cdot h(s)$$

If $\mathbf{R}[s, s] = 0$ then there is no respective path in $E^{UT}(s, s)$ and therefore:

$$\sum_{\rho \in \mathit{Paths}^{\omega}_{s,s}(\mathcal{D}_{\varrho})} \mathbb{P}^{\mathcal{D}}_A[\rho] \cdot r^{\mathcal{D}}_A[\rho]$$
$$= \sum_{\rho^{UT} \in E^{UT}(s,s)} \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}] \cdot (r'_A[\rho^{UT}] + h(s')) + (1 - \frac{E(s)}{\eta}) \cdot h(s)$$

If $\mathbf{R}[s,s] > 0$ then there exists a path $\rho^{UT} = s \xrightarrow{\perp} s \in E^{UT}(s,s)$, and therefore:

$$\begin{aligned}
&\sum_{\rho \in Paths^{\omega}_{s,s}(\mathcal{D}_\varrho)} \mathbb{P}^{\mathcal{D}}_A[\rho] \cdot r^{\mathcal{D}}_A[\rho] \\
&= \sum_{\substack{\rho^{UT} \in E^{UT}(s,s) \\ |\rho^{UT}|>1}} \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}] \cdot (r'_A[\rho^{UT}] + h(s')) + (1 - \frac{E(s) - \mathbf{R}[s,s]}{\eta}) \cdot h(s) \\
&= \sum_{\substack{\rho^{UT} \in E^{UT}(s,s) \\ |\rho^{UT}|>1}} \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}] \cdot (r'_A[\rho^{UT}] + h(s')) + \frac{E(s)}{\eta} \cdot \frac{\mathbf{R}[s,s]}{E(s)} \cdot h(s) \\
&\quad + (1 - \frac{E(s)}{\eta}) \cdot h(s) \\
&= \sum_{\rho^{UT} \in E^{UT}(s,s)} \frac{E(s)}{\eta} \cdot \mathbb{P}_A[\rho^{UT}] \cdot (r'_A[\rho^{UT}] + h(s')) + (1 - \frac{E(s)}{\eta}) \cdot h(s)
\end{aligned}$$

This concludes the proof. □

Due to Lemmas 5.3.2 and 5.3.3 we can obtain a better Bellman equation for an MRA as follows:

Lemma 5.3.4. *Let $\mathcal{M}_\varrho$ be a unichain MRA and $\eta > \mathbf{E}_{\max}$ then there exists a function $h : MS \to \mathbb{R}$ and value $a \in \mathbb{R}_{\geqslant 0}$ that are a solution to the Bellman equation:*

$$\forall s \in MS : h(s) + \frac{a}{\eta} = \mathrm{urew}^{\mathcal{D}_\varrho}_{\mathrm{opt}}(s), \tag{5.12}$$

where $\mathcal{D}_\varrho = (t\mathcal{D}(\eta), \varrho^h_{t\mathcal{D}(\eta)})$. Moreover $a = \overline{\mathrm{lval}}^{\mathcal{M}_\varrho}_{\mathrm{opt}}$.

Equation (5.12) is better than (5.8) because it makes clear that the right-hand side of (5.8) has some structure and therefore can be evaluated better than by naively brute-forcing all the possibly exponentially many actions of the CTMDP $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$. Instead one can compute the values $\overline{\mathrm{urew}}^{\mathcal{D}_\varrho}_{\mathrm{opt}}$. Previously in Section 5.1.1 of this chapter we have established that computation of $\overline{\mathrm{urew}}^{\mathcal{D}_\varrho}_{\mathrm{opt}}$ is a well-studied problem that admits efficient solutions. In the next section, we will make use of those techniques to develop an algorithm for solving equation (5.12).

5.3.4 Algorithmic Solution

The goal of this section is to develop an iterative algorithm that computes value $\overline{\mathrm{lval}}^{\mathcal{M}_\varrho}_{\mathrm{opt}}$. We will achieve this by solving iteratively the Bellman equation (5.12).

For the case of CTMDPs, solution of the Bellman equation (5.8) can be found with value or policy iteration [Put94]. Since equation (5.12) is a different way of writing (5.8), we can apply these algorithms to (5.12). The algorithms would evaluate the right-hand side of (5.12) in a brute-force manner, i. e. the optimal value $\mathrm{urew}^{\mathcal{D}_\varrho}_{\mathrm{opt}}(s)$ would be computed by first computing values $\mathrm{urew}^{\mathcal{D}_\varrho}_{\pi}(s)$ for each stationary scheduler π and then choosing the optimal one. Since the right-hand side

Algorithm 6 `RVI`

```
Input: Unichain MRA $\mathcal{M}_\varrho$, opt $\in \{\sup, \inf\}$, approximation error $\epsilon > 0$
Output: $u$ such that $|u - \overline{\mathrm{lval}}^{\mathcal{M}_\varrho}_{\mathrm{opt}}| < \epsilon/2$, and $\epsilon$-optimal scheduler $\pi_{\mathtt{opt}}$
 1: Choose $\eta > \mathbf{E}_{\max}$
 2: $t\mathcal{D}(\eta) \longleftarrow$ terminal MDP (see Section 5.3.3)
 3: $s^* \longleftarrow$ any Markovian state of $\mathcal{M}$
 4: $\forall s \in MS : u_0(s) = 0, u_1(s) = 1$
 5: $\forall s \in MS : w_0(s) = 0$
 6: for $(n = 0;\ sp(u_{n+1} - u_n) < \frac{\epsilon}{\eta};\ n++)$ do
 7:     $\varrho^{w_n}_{t\mathcal{D}(\eta)} \longleftarrow$ rewards structure for the terminal MDP (see Section 5.3.3)
 8:     $\mathcal{D}_\varrho = (t\mathcal{D}(\eta), \varrho^{w_n}_{t\mathcal{D}(\eta)})$
 9:     $v, \pi = \mathtt{TotalExpectedReward}(\mathcal{D}_\varrho, \mathrm{opt})$
10:     $\forall s \in MS : u_{n+1}(s) = v(s)$
11:     $\forall s \in PS : \pi_{\mathtt{opt}}(ps) = \pi(ps)$
12:     $\forall s \in MS : w_{n+1}(s) = u_{n+1}(s) - u_{n+1}(s^*)$
13: $g' = \frac{1}{2} \cdot (\max_{s \in MS}(u_{n+1}(s) - u_n(s)) + \min_{s \in MS}(u_{n+1}(s) - u_n(s))) + u_n(s^*)$
14: return $g' \cdot \eta, \pi_{\mathtt{opt}}$;
```

of the Bellman equation (5.12) is in fact a separate optimisation problem in itself, as shown by Lemma 5.3.4, instead we will apply dedicated algorithms that optimise values $\mathrm{urew}^{\mathcal{D}_\varrho}_{\mathrm{opt}}(s)$.

Our solution, shown in Algorithm 6, is a relative value iteration algorithm[2] for approximating value $\overline{\mathrm{lval}}^{\mathcal{M}_\varrho}_{\mathrm{opt}}$. Here $sp(v) := \left|\max_{s \in MS}\{v(s)\} - \min_{s \in MS}\{v(s)\}\right|$. We denote with $\mathtt{TotalExpectedReward}(\mathcal{D}_\varrho, \mathrm{opt})$ a procedure that computes values $\overline{\mathrm{urew}}^{\mathcal{D}_\varrho}_{\mathrm{opt}}$ for an MDP with rewards $\mathcal{D}_\varrho = (\mathcal{D}, \varrho_{\mathrm{trn}})$. If $\mathcal{D} = (S_\mathcal{D}, Act_\mathcal{D}, \mathbf{P}_\mathcal{D})$, then its output is a pair (v, π), where $\pi \in \Pi^{\mathcal{D}}_{\mathtt{stat}}$ and $v : S_\mathcal{D} \to \mathbb{R}$, such that $\forall s \in S_\mathcal{D} : v(s) = \mathrm{urew}^{\mathcal{D}_\varrho}_{\mathrm{opt}}(s)$. Algorithm 6 has two levels of computations: the standard relative value iteration for MDPs as an outer loop and during each of these iterations we make a call to procedure `TotalExpectedReward` to compute values $\mathrm{urew}^{\mathcal{D}_\varrho}_{\mathrm{opt}}(s)$.

Theorem 5.3.5. *Let $u, \pi_{\mathtt{opt}}$ be the output of Algorithm 6 for an MRA $\mathcal{M}_\varrho$, $\epsilon \in (0,1)$, $\mathrm{opt} \in \{\sup, \inf\}$. Then $\pi_{\mathtt{opt}}$ is ϵ-optimal for $\overline{\mathrm{lval}}^{\mathcal{M}_\varrho}_{\mathrm{opt}}$ and $|u - \overline{\mathrm{lval}}^{\mathcal{M}_\varrho}_{\mathrm{opt}}| < \epsilon/2$.*

Proof. Due to Lemmas 5.3.2 and 5.3.3 one can approximate the solution to (5.12) with relative value iteration algorithm from [Put94]. Consider the case of $\varepsilon = 0$. In this case Algorithm 6 is exactly the relative value iteration algorithm from [Put94] applied to (5.8) for $\epsilon' = \epsilon/\eta$.

Let v_n be a sequence of values produced by the classical (i. e. non relative) value iteration algorithm, applied to (5.8). It can be shown via simple induction, that

[2]Classical value iteration is also possible, but may be numerically unstable, just like in the CTMDP case [Put94].

if $\forall s \in MS : v_0(s) = u_0(s)$, then $\forall n \in \mathbb{Z}_{>0}, s \in S : u_n(s) = v_n(s) - \sum_{i=0}^{n-1} u_i(s^*)$. Consider value

$$\begin{aligned} g' &= \frac{1}{2} \cdot \left(\max_{s \in MS}(v_{n+1}(s) - v_n(s)) + \min_{s \in MS}(v_{n+1}(s) - v_n(s)) \right) \\ &= \frac{1}{2} \cdot \left(\max_{s \in MS}(u_{n+1}(s) - u_n(s)) + \min_{s \in MS}(u_{n+1}(s) - u_n(s)) \right) + u_n(s^*) \\ &= u/\lambda \end{aligned}$$

It has been shown in Theorem 8.5.6 of [Put94] that $|g' - \overline{\text{lval}}_{\pi_{\texttt{opt}}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}/\eta| < \epsilon'/2$ and $|g' - \overline{\text{lval}}_{\text{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}/\eta| < \epsilon'/2$. Or equivalently $|g' \cdot \eta - \overline{\text{lval}}_{\pi_{\texttt{opt}}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}| < \epsilon/2$ and $|g' \cdot \eta - \overline{\text{lval}}_{\text{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}| < \epsilon/2$. Due to Lemma 5.3.1 this implies that $0 \preccurlyeq_{\text{opt}} \overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho} - \overline{\text{lval}}_{\pi_{\texttt{opt}}}^{\mathcal{M}_\varrho} \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \cdot \epsilon$ and therefore $\pi_{\texttt{opt}}$ is ϵ-optimal for $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}$. Additionally, $|u - \overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}| < \epsilon/2$.

□

Remark 5.3.2. *Notice that Algorithm 6 relies on values $\overline{\text{urew}}_{\text{opt}}^{\mathcal{D}_\varrho}$ to be computed exactly by procedure* `TotalExpectedReward`$(\mathcal{D}_\varrho, \text{opt})$. *As mentioned in Section 5.1.1 possible candidates to solve the general case of this problem is an algorithm based on a reduction to linear-programming and policy iteration. Value iteration based approaches provide only an ε-approximation of values $\overline{\text{urew}}_{\text{opt}}^{\mathcal{D}_\varrho}$ and an ε-optimal scheduler, for a given ε, and we cannot guarantee soundness of Algorithm 6 in this case. This could seem as an obstacle towards a fully iterative solution for approximating values $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}$, and in fact it is from the theoretical perspective. However we argue that this is not a barrier in practice. The only Markov automata models known to us to date (that can be found in, e. g. [HKP+19]) have a very specific structure. Namely, their terminal MDP is acyclic. Acyclic MDPs admit a value iteration based solution for computing values $\overline{\text{urew}}_{\text{opt}}^{\mathcal{D}_\varrho}$ exactly, as discussed in Section 5.1.1. We therefore argue that Algorithm 6 suffices as an iterative approximation algorithm for $\overline{\text{lval}}_{\text{opt}}^{\mathcal{M}_\varrho}$ for most practical needs.*

Complexity. The algorithm iterates until convergence. The complexity of each iteration is determined by the complexity of procedure `TotalExpectedReward`$(\mathcal{D}_\varrho, \text{opt})$. Let $|S| = n$ be the amount of states, $a = |Act|$ and m is the amount of *edges* in $\mathcal{M}$ (see Section 4.1.1). Let $C_{urew}^{\text{opt}}(n, m, a)$ denote the complexity of computing values $\overline{\text{urew}}_{\text{opt}}^{\mathcal{D}_\varrho}$. If these values are computed via a reduction to linear programming, then the linear program has n variables and $n \cdot a$ constraints. If the MA is *PS*-acyclic, then these values can be computed in time $O(m)$.

Integration with the General Case Solution. In the following, we will (i) discuss how Algorithm 6 can be used to approximate the optimal long-run average reward value for a MEC, and (ii) discuss how to integrate thus obtained approximate values into the general case solution presented in Section 5.2.

In order to apply to MECs the theory developed in this chapter for unichain MRA, we need to ensure that Lemmas 5.3.1, 5.3.3 and Theorem 5.3.5 hold for MECs.

This is enabled by two core observations, both established in [GTH+14]. First, there exists an optimal strategy for MECs that is stationary. Second, the value of the optimal long-run average reward is constant for all states of a MEC. These properties ensure that Lemma 5.3.1, and consequently Lemma 5.3.3, hold for MECs. Next, it has been shown in Chapters 9.1.1 and 11.5.3 of [Put94] how the optimality equation for MEC CTMDPs corresponds to that of unichain CTMDPs. Lastly, the correctness of Theorem 5.3.5 for MECs follows from the correctness of the stopping criterion $sp(u_{n+1} - u_n)$ in the "for-loop" of Algorithm 6 for MECs. The latter has been shown in Chapter 9.5.3 of [Put94]. Therefore, in order to approximate value $\overline{\text{lval}}^{\mathcal{M}_\varrho}_{\text{opt}}$ for a MEC $\mathcal{M}_\varrho$, one can follow the same pipeline "MRA $\rightarrow$ CTMDP $\overset{\text{[Put94]}}{\rightarrow}$ MDP" that we have established for unichain MRA. This means that one simply needs to run Algorithm 6 on $\mathcal{M}_\varrho$.

Consider an arbitrary (not necessarily unichain) MRA $\mathcal{M}_\varrho$ and let $\mathcal{M}^j_\varrho = (S^j, \rightsquigarrow^j), j = 1..k$ be the MECs of $\mathcal{M}_\varrho$. General case solution presented in Section 5.2 is only applicable if values $\overline{\text{lval}}^{\mathcal{M}^j_\varrho}_{\text{opt}}$ are computed exactly. Since Algorithm 6 approximates these values, we cannot use the algorithm to solve the general problem yet. We will show that given an ε-approximation of values $\overline{\text{lval}}^{\mathcal{M}^j_\varrho}_{\text{opt}}$ one can construct a collapsed MDP $\mathcal{D}_\varrho(\varepsilon)$ that uses these approximations instead of exact values and compute an ε-approximation of value $\overline{\text{urew}}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt}}$. This way the total error introduced by all the approximations does not exceed $2 \cdot \varepsilon$.

Let $\varepsilon \in [0, 1)$ and $\mathcal{D}_\varrho = (\mathcal{D}, \varrho_\mathcal{D})$ be the collapsed MDP of $\mathcal{M}_\varrho$. We will denote with $\overline{\text{lval}}^{\mathcal{M}'_\varrho}_{\text{opt},\varepsilon}$ and $\overline{\text{urew}}^{\mathcal{D}'_\varrho}_{\text{opt},\varepsilon}$ ε-approximations of $\overline{\text{lval}}^{\mathcal{M}'_\varrho}_{\text{opt}}$ and $\overline{\text{urew}}^{\mathcal{D}'_\varrho}_{\text{opt}}$ for an MRA $\mathcal{M}'_\varrho$ and MDP $\mathcal{D}'_\varrho$ respectively. Let $\mathcal{D}_\varrho(\varepsilon)$ be an MDP obtained in exactly the same way as $\mathcal{D}_\varrho$ with the only difference in the reward structure. Instead of assigning reward $\overline{\text{lval}}^{\mathcal{M}^j_\varrho}_{\text{opt}}$ to MEC $\mathcal{M}^j_\varrho$ it assigns reward $\overline{\text{lval}}^{\mathcal{M}^j_\varrho}_{\text{opt},\varepsilon}$. Formally, $\mathcal{D}_\varrho(\varepsilon) = (\mathcal{D}, \varrho_{\mathcal{D},\varepsilon})$, where $\forall q_j \in Q : \varrho_{\mathcal{D},\varepsilon}(q_j, \perp) = \overline{\text{lval}}^{\mathcal{M}^j_\varrho}_{\text{opt},\varepsilon}$ and in all other cases $\varrho_{\mathcal{D},\varepsilon}(s, \alpha) = \varrho_\mathcal{D}(s, \alpha)$.

Lemma 5.3.6. $\forall s \in S : \left|\text{lval}^{\mathcal{M}_\varrho}_{\text{opt}}(s) - \text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt},\varepsilon}(f(s))\right| \leqslant 2 \cdot \varepsilon$, *where*

$$f(s) = \begin{cases} u_j & \textit{if } s \in S^j \\ s & \textit{otherwise} \end{cases}$$

Proof. For $\varepsilon = 0$ the statement follows from the general case solution described in Section 5.2, since $\text{lval}^{\mathcal{M}_\varrho}_{\text{opt}}(s) = \text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(f(s))$. Consider $\varepsilon > 0$. Then:

$$\begin{aligned}
&\left|\text{lval}^{\mathcal{M}_\varrho}_{\text{opt}}(s) - \text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt},\varepsilon}(f(s))\right| \\
&\quad = \left|\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(f(s)) - \text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt},\varepsilon}(f(s))\right| \\
&\quad \leqslant \left|\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(f(s)) - \text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt}}(f(s))\right| + \left|\text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt}}(f(s)) - \text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt},\varepsilon}(f(s))\right| \\
&\quad \leqslant \left|\text{urew}^{\mathcal{D}_\varrho}_{\text{opt}}(f(s)) - \text{urew}^{\mathcal{D}_\varrho(\varepsilon)}_{\text{opt}}(f(s))\right| + \varepsilon
\end{aligned}$$

Table 5.1: Parameters of models used for the experiments.

Model	$\lvert S\rvert$	MECs	$\mathbf{E}_{\max}$	$\max_{s\in S}\lvert Act(s)\rvert$
`dpm`	112-5,666,749	1	2.00-6.00	3-8
`ps`	113-9,565,937	1	2.80-256.80	2-8
`qs`	577-7,065,927	1	4.50-8.50	2-12
`ftwc`	147,681-8,749,281	1	2.13-3.02	5-5

Analogously to Lemma A.5 we can show that $\left|\mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho}(f(s)) - \mathrm{urew}_{\mathrm{opt}}^{\mathcal{D}_\varrho(\varepsilon)}(f(s))\right| \leqslant \varepsilon$. This concludes the proof.

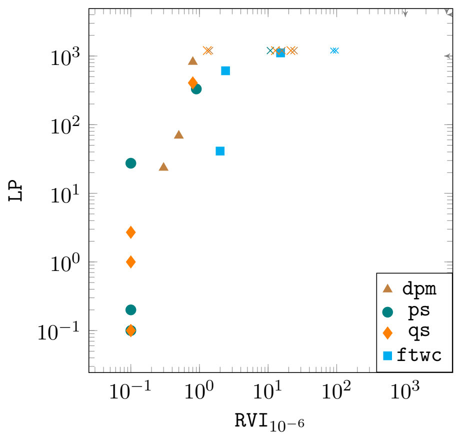
□

5.3.5 Experimental Evaluation

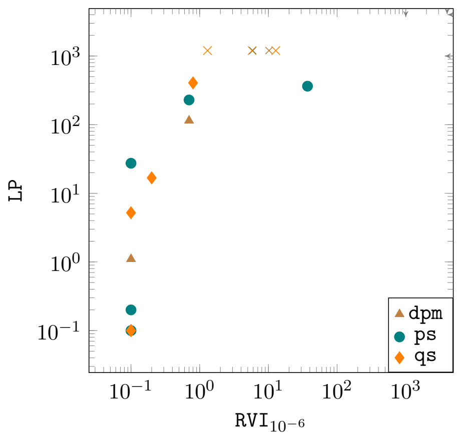
In this section we present results of an empirical comparison of Algorithm 6 (denoted in the following with `RVI`) and the linear-programming based solution from [GTH+14] (denoted in the following with `LP`), discussed in Section 5.2. Both algorithms are implemented in the `Modest Toolset` [HH14]. All experiments were conducted on Intel Core i7-4790 with 16 GB of RAM running 64-bit Ubuntu Linux 18.04. Details on how to obtain all the data points used in the plots of this section can be found in Appendix B.1.2.

Whenever we request a certain accuracy for results, we request absolute, not relative, accuracy. If algorithm `RVI` is used to approximate long-run rewards up to error ϵ, then we will denote this with $\texttt{RVI}_\epsilon$. The `Modest Toolset` uses `Google OR-tools` [PF] as an underlying linear programming solver and does not provide any means to adjust the parameters of the solver. We, therefore, use the default ones for the `LP`. We set the timeout for experiments to 20 minutes; a timeout is denoted in plots by an "`x`". In this section, the runtime of an algorithm on a certain problem includes only the time it takes to compute the long-run average values and does not include the time that it takes to load the respective model into memory.

Models. We evaluate the algorithms on four MA benchmarks from the Quantitative Verification Benchmark Set (QVBS, [HKP+19]): *Dynamic Power Management* (`dpm`, version 2), *Fault Tolerant Workstation Cluster* (`ftwc`, version 3), *Polling System* (`ps`, version 3) and *Reentrant Queuing System* (`qs`, version 3). All models have open parameters, such as buffer capacity (that affects state-space size), the number of various request types (that is related to the maximum number of enabled actions), etc. Varying these parameters allows us to scale the models up from small to large state-spaces to compare the performance and scalability of the algorithms. Due to the absence of long-run average reward properties in most MRA models of the benchmark set, we added sensible long-run average properties to most of the `Modest` models in order to be able to do a performance comparison. Those are mainly steady-state probabilities (i.e. the special case of a rate reward of 1 in some states and of 0 in all others), or properties describing long-run average costs of running the system. All the models and properties used for the experiments can



(a) state space



(b) actions

FIGURE 5.5: Runtime comparison of `RVI` and `LP` varying those problem parameters that affect state-space size (5.5a) and max number of actions (5.5b).

be found in Appendix B.2.1. Table 5.1 reports on minimum and maximum values of the state-space sizes, $\mathbf{E}_{\max}$, highest number of enabled actions $\max_{s\in S}|Act(s)|$ and number of MECs across all the models used for the evaluation presented in this section. The number of maximal end components was at most 1 across all models.

Discussion. We evaluate the algorithms on three sets of experiments: (i) Varying those parameters of models that affect the state-space size, (ii) the number of enabled actions and (iii) varying accuracy of the approximations.

Model Size. The results for experiments (i) and (ii) are presented in Figure (5.5a) and Figure (5.5b). We show the results as scatter plots with log-log axes. A point $\langle x, y\rangle$ states that the runtime of the algorithm noted on the x-axis on one instance was x seconds while the runtime of the algorithm noted on the y-axis was y seconds. Thus points above the diagonal indicate instances where the x-tool was faster than the y-tool. For the plot in Figure (5.5b) we do not compare the algorithms on `ftwc` model because the model does not provide any interface to vary the number of available actions. The general observation from these experiments is that `RVI` scales much better with the increase of model size than `LP`, often having an advantage of several orders of magnitude. The `LP` algorithm can be comparable on smaller models, but on larger ones `RVI` takes over.

Accuracy. The comparison with respect to changing the accuracy of the approximations is shown in Figure 5.6. Here the plots have log-log axes and we reversed the x-axis since approximation with higher accuracy (lower values of the x-axis) is a harder problem than approximation with lower accuracy (higher values of the x-axis). We evaluate both algorithms on four models and accuracy values ranging from 10^{-3} to 10^{-10}. The value of the y-axis is the running time of the algorithms in seconds. Recall that we cannot vary the accuracy parameter for the `LP` algorithm. Therefore for each experiment in this plot, the `LP` algorithm provides only one value.

We depict this value with a solid line parallel to the x-axis. The symbol at the beginning of the line together with the colour of the line refers to the specific model that was evaluated. The RVI algorithm provides approximations to the true value up to the given accuracy and therefore for each of the accuracy values we obtain a different point on the plot. These points are denoted with respective symbols from the legend of the plot. Once again we observe that RVI significantly outperforms LP even when the requested accuracy is very high.

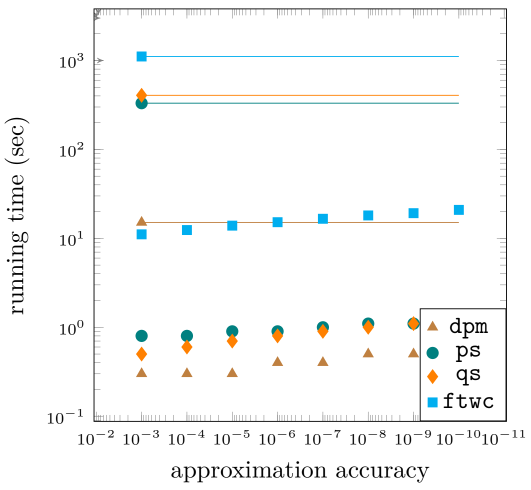


Figure 5.6: Runtime comparison of RVI and LP when varying accuracy.

5.4 Conclusions and Discussion

In this chapter, we revisited the analysis of long-run average reward value $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{M}_\varrho}$ for an arbitrary Markov reward automaton $\mathcal{M}_\varrho$. Below is a brief overview of the chapter:

- We recalled that computation of $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{M}_\varrho}$ for an arbitrary Markov reward automaton can be reduced to three separate problems. Only one of those problems appears to be challenging, which is the computation of optimal long-run average rewards for a *unichain* Markov reward automaton.

- We show that in terms of long-run rewards Markov automata can be thought of as an efficient encoding of an exponentially large CTMDP. Namely, for every unichain Markov automaton $\mathcal{M}_\varrho$ there exists a (possibly exponentially larger) unichain CTMDP $\mathcal{C}_\varrho(\mathcal{M}_\varrho)$ that has the same optimal long-run average reward value.

- We derive a Bellman equation for value $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{M}_\varrho}$ of a unichain Markov automaton from the Bellman equation for value $\mathrm{lval}_{\mathrm{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}(s)$ of its respective CTMDP. The right-hand side of the Bellman equation for $\mathrm{lval}_{\mathrm{opt}}^{\mathcal{C}_\varrho(\mathcal{M}_\varrho)}(s)$ requires exponentially many computations. We bypass this problem by showing that the right-hand side is a known problem of cumulative unbounded reward in MDPs that can be solved efficiently.

- We obtain a sound iterative algorithm (Algorithm 6) for approximating $\overline{\mathrm{lval}}_{\mathrm{opt}}^{\mathcal{M}_\varrho}$ of a unichain MRA $\mathcal{M}_\varrho$ up to an arbitrary given error bound ϵ and show how to integrate this approximate solution into the general case solution for an arbitrary MRA.

- An empirical evaluation of the iterative algorithm, that is presented in Section 5.3.5, shows that the iterative algorithm is better than the linear programming based approach on many case studies. The running time of the iterative algorithm often is several orders of magnitude better than that of the linear programming based solution.

Future Directions.

- *Approximate solution for the inner loop.* We have shown correctness of Algorithm 6 only under the assumption that the sub-problem of computing optimal expected cumulative rewards in MDPs (step 9) is solved exactly. In general case, this problem can be solved with the help of policy iteration or linear programming. Both tend to scale worse than approximate algorithms, such as value iteration. Value iteration, however, can provide an exact solution only for a subclass of problems, e. g. *PS*-acyclic MA. An interesting problem to investigate is whether correctness of Algorithm 6 can be ensured while using an approximation of the solution of the sub-problem at step 9. And if yes, which error should be used for the approximations of this sub-problem, to ensure that the overall error remains below a given threshold.
- *Apply Bounded Real-Time Dynamic Programming (BRTDP) techniques* to further improve the running time of the analysis of long-run average objectives for large Markov automata. This has been done for MDPs [BCC$^+$14], however, has not been considered yet for Markov automata.
- *Multi-objective model checking.* The only existing algorithm for model checking multiple objective for Markov automata is presented in [QJK17] and so far ignores long-run objectives. It is interesting to study whether it is possible to integrate any of the algorithms for long-run rewards into the multi-objective setting.

Conclusions 6

In this thesis, we studied several analysis problems for Markov automata that remained challenging to date. The main contribution of this work is in new algorithms to solve these problems. We believe that all the results that we have shown here do advance our understanding of the problems and make a step forward in terms of their analysis. There are, however, ways to improve at least some of our algorithms. Below we give a brief overview of the achieved results:

Analysis of Time-Bounded Reachability Based on Switching Points. We have studied switching points of optimal piecewise-constant schedulers and discovered a convenient way to characterise them. Due to these results, we were able to develop a new algorithm that approximates the time-bounded reachability probabilities up to a given error bound. An empirical comparison of this algorithm with existing ones shows that the algorithm is competitive, however, does not strictly outperform all existing approaches. A more detailed overview of the results can be found in Chapter 4, Section 4.6.

Time-Bounded Reachability on Partial State-space. We have designed an algorithm that approximates time-bounded reachability probabilities with guaranteed error bounds by performing computations only on a part of the total state-space. The algorithm is randomised and is guaranteed to converge almost surely. The empirical evaluation shows that the algorithm can analyse very large models within a few seconds, as opposed to classical approaches that take more than 20 minutes. A more detailed discussion of this algorithm can be found in Chapter 4, Section 4.6.

Analysis of Long-Run Average Rewards. We have designed a new algorithm based on value iteration that approximates the value of long-run average rewards for a unichain Markov automaton up to a given error bound. Empirical evaluation of this algorithm shows that it outperforms the existing approach based on linear programming. For a more technical overview of the results we refer the reader to Chapter 5, Section 5.4.

Overall, we consider the analysis of the long-run average rewards for Markov automata mostly a closed topic. The value iteration algorithm that we developed in this work seems to be a natural Markov automata extension of the same algorithm for CTMDPs, which is considered to be one of the most efficient algorithms for practical cases. The situation is not the same with the time-bounded reachability probability. The empirical evaluation shows that there are still many models on which our algorithm could be improved. The main direction for improvement is to lower the bound on the error introduced by following a possibly sub-optimal strategy. A more technical discussion of these questions can be found in Chapter 4, Section 4.6.

Bibliography 6

[ACD^{+}17] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value iteration for long-run average reward in Markov decision processes. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 201–221. Springer, 2017. URL: `https://doi.org/10.1007/978-3-319-63387-9_10`, `doi:10.1007/978-3-319-63387-9\_10`.

[ADD00] Robert B. Ash and Catherine A. Doleans-Dade. *Probability and Measure Theory*. Elsevier Science, 2000. URL: `https://books.google.co.uz/books?id=GkqQoRpCO2QC`.

[BCC^{+}14] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014. URL: `https://doi.org/10.1007/978-3-319-11936-6_8`, `doi:10.1007/978-3-319-11936-6\_8`.

[Ber05] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Number v. 1 in Athena Scientific optimization and computation series. Athena Scientific, 2005. URL: `https://books.google.co.uz/books?id=-E5qQgAACAAJ`.

[BH99] Henrik C. Bohnenkamp and Boudewijn R. Haverkort. Semi-numerical solution of stochastic process algebra models. In *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, volume 1601 of *Lecture Notes in Computer Science*, pages 228–243.

Springer, 1999. URL: `https://doi.org/10.1007/3-540-48778-6_14`, `doi:10.1007/3-540-48778-6\_14`.

[BHHK00] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the logical characterisation of performability properties. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792. Springer, 2000. URL: `https://doi.org/10.1007/3-540-45022-X_65`, `doi:10.1007/3-540-45022-X\_65`.

[BHHK03] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003. URL: `https://doi.org/10.1109/TSE.2003.1205180`, `doi:10.1109/TSE.2003.1205180`.

[BHHK15] Yuliya Butkova, Hassan Hatefi, Holger Hermanns, and Jan Krcál. Optimal continuous time Markov decisions. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, volume 9364 of *Lecture Notes in Computer Science*, pages 166–182. Springer, 2015. URL: `https://doi.org/10.1007/978-3-319-24953-7_12`, `doi:10.1007/978-3-319-24953-7\_12`.

[BHHZ11] Peter Buchholz, Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Model checking algorithms for CTMDPs. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2011. URL: `https://doi.org/10.1007/978-3-642-22110-1_19`, `doi:10.1007/978-3-642-22110-1\_19`.

[BKL$^+$17] 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 *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 160–180. Springer, 2017. URL: `https://doi.org/10.1007/978-3-319-63387-9_8`, `doi:10.1007/978-3-319-63387-9\_8`.

[BS11] Peter Buchholz and Ingo Schulz. Numerical analysis of continuous time Markov decision processes over finite horizons. *Computers & OR*, 38(3):651–659, 2011. URL: `https://doi.org/10.1016/j.cor.2010.08.011`, `doi:10.1016/j.cor.2010.08.011`.

[BWH18] Yuliya Butkova, Ralf Wimmer, and Holger Hermanns. Markov automata on discount! In *Measurement, Modelling and Evaluation of*

Computing Systems - 19th International GI/ITG Conference, MMB 2018, Erlangen, Germany, February 26-28, 2018, Proceedings, volume 10740 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2018. URL: `https://doi.org/10.1007/978-3-319-74947-1_2`, `doi:10.1007/978-3-319-74947-1\_2`.

[CH11] Krishnendu Chatterjee and Monika Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1318–1336. SIAM, 2011. URL: `https://doi.org/10.1137/1.9781611973082.101`, `doi:10.1137/1.9781611973082.101`.

[DBB90] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Fault trees and sequence dependencies. In *Annual Proceedings on Reliability and Maintainability Symposium*, pages 286–293, Jan 1990. `doi:10.1109/ARMS.1990.67971`.

[DH11] Yuxin Deng and Matthew Hennessy. On the semantics of Markov automata. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2011. URL: `https://doi.org/10.1007/978-3-642-22012-8_24`, `doi:10.1007/978-3-642-22012-8\_24`.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer, 2017. URL: `https://doi.org/10.1007/978-3-319-63390-9_31`, `doi:10.1007/978-3-319-63390-9\_31`.

[EHZ10] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On probabilistic automata in continuous time. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 342–351. IEEE Computer Society, 2010. URL: `https://doi.org/10.1109/LICS.2010.41`, `doi:10.1109/LICS.2010.41`.

[FRSZ11] John Fearnley, Markus N. Rabe, Sven Schewe, and Lijun Zhang. Efficient approximation of optimal control for continuous-time Markov games. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 399–410. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. URL: `https://doi.org/10.4230/LIPIcs.FSTTCS.2011.399`, `doi:10.4230/LIPIcs.FSTTCS.2011.399`.

[Fu14] Hongfei Fu. Maximal cost-bounded reachability probability on continuous-time Markov decision processes. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2014. URL: `https://doi.org/10.1007/978-3-642-54830-7_5`, `doi:10.1007/978-3-642-54830-7\_5`.

[GHH$^+$14] Dennis Guck, Hassan Hatefi, Holger Hermanns, Joost-Pieter Katoen, and Mark Timmer. Analysis of timed and long-run objectives for Markov automata. *Logical Methods in Computer Science*, 10(3), 2014.

[GHKN12] Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer. Quantitative timed analysis of interactive Markov chains. In *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 8–23. Springer, 2012.

[Gro18] Timo P. Gros. Markov automata taken by Storm. Master's thesis, Saarland University, Germany, 2018.

[GTH$^+$14] Dennis Guck, Mark Timmer, Hassan Hatefi, Enno Ruijters, and Mariëlle Stoelinga. Modelling and analysis of Markov reward automata. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 168–184. Springer, 2014. URL: `https://doi.org/10.1007/978-3-319-11936-6_13`, `doi:10.1007/978-3-319-11936-6\_13`.

[Hat17] Hassan Hatefi-Ardakani. *Finite horizon analysis of Markov automata.* PhD thesis, Saarland University, Germany, 2017. URL: `http://scidok.sulb.uni-saarland.de/volltexte/2017/6743/`.

[Her02] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science.* Springer, 2002.

[HH12] Hassan Hatefi and Holger Hermanns. Model checking algorithms for Markov automata. *ECEASST*, 53, 2012. URL: `http://journal.ub.tu-berlin.de/eceasst/article/view/783`.

[HH13] Hassan Hatefi and Holger Hermanns. Improving time bounded reachability computations in interactive Markov chains. In *Fundamentals of Software Engineering - 5th International Conference, FSEN 2013, Tehran, Iran, April 24-26, 2013, Revised Selected Papers*, volume 8161 of *Lecture Notes in Computer Science*, pages 250–266. Springer, 2013. URL: `https://doi.org/10.1007/978-3-642-40213-5_16`, `doi:10.1007/978-3-642-40213-5\_16`.

[HH14] Arnd Hartmanns and Holger Hermanns. The Modest Toolset: An integrated environment for quantitative modelling and verification. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 593–598. Springer, 2014. URL: `https://doi.org/10.1007/978-3-642-54862-8_51`, `doi:10.1007/978-3-642-54862-8\_51`.

[HHHK13] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013. `doi:10.1007/s10703-012-0167-z`.

[HKP+19] Arnd Hartmanns, Michaela Klauck, David Parker, Tim Quatmann, and Enno Ruijters. The quantitative verification benchmark set. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 344–350. Springer, 2019. URL: `https://doi.org/10.1007/978-3-030-17462-0_20`, `doi:10.1007/978-3-030-17462-0\_20`.

[HM14] Serge Haddad and Benjamin Monmege. Reachability in MDPs: Refining convergence of value iteration. In *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 125–137. Springer, 2014. URL: `https://doi.org/10.1007/978-3-319-11439-2_10`, `doi:10.1007/978-3-319-11439-2\_10`.

[HMN13] Serge Haddad, Jean Mairesse, and Hoang-Thach Nguyen. Synthesis and analysis of product-form petri nets. *Fundam. Inform.*, 122(1-2):147–172, 2013. URL: `https://doi.org/10.3233/FI-2013-786`, `doi:10.3233/FI-2013-786`.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978. URL: `https://doi.org/10.1145/359576.359585`, `doi:10.1145/359576.359585`.

[Jen53] Arne Jensen. Markoff chains as an aid in the study of Markoff processes. *Scandinavian Actuarial Journal*, 1953(sup1):87–91, 1953. URL: `https://doi.org/10.1080/03461238.1953.10419459`, `arXiv:https://doi.org/10.1080/03461238.1953.10419459`, `doi:10.1080/03461238.1953.10419459`.

[Mar95] M.A. Marsan. *Modelling with generalized stochastic Petri nets.* Wiley series in parallel computing. Wiley, 1995. URL: `https://books.google.com/books?id=ZN5QAAAAMAAJ`.

[Mil68] B. Miller. Finite state continuous time Markov decision processes with a finite planning horizon. *SIAM Journal on Control*, 6(2):266–280, 1968. URL: `https://doi.org/10.1137/0306020`, `arXiv:https://doi.org/10.1137/0306020`, `doi:10.1137/0306020`.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. URL: `https://doi.org/10.1007/3-540-10235-3`, `doi:10.1007/3-540-10235-3`.

[MLG05] H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 569–576. ACM, 2005. URL: `https://doi.org/10.1145/1102351.1102423`, `doi:10.1145/1102351.1102423`.

[Neu10] Martin R. Neuhäußer. *Model checking nondeterministic and randomly timed systems*. PhD thesis, RWTH Aachen University, 2010.

[NS91] Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In *Computer Aided Verification, 3rd International Workshop, CAV '91, Aalborg, Denmark, July, 1-4, 1991, Proceedings*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1991. URL: `https://doi.org/10.1007/3-540-55179-4_36`, `doi:10.1007/3-540-55179-4\_36`.

[NZ10] Martin R. Neuhäußer and Lijun Zhang. Time-bounded reachability probabilities in continuous-time Markov decision processes. In *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010*, pages 209–218. IEEE Computer Society, 2010. URL: `https://doi.org/10.1109/QEST.2010.47`, `doi:10.1109/QEST.2010.47`.

[Pan09] P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009. URL: `https://books.google.co.uz/books?id=N28mUNHxeYQC`.

[PF] Laurent Perron and Vincent Furnon. *OR-Tools*. Google. URL: `https://developers.google.com/optimization/`.

[Pnu85] Amir Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, pages 123–144, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[QJK17] Tim Quatmann, Sebastian Junges, and Joost-Pieter Katoen. Markov automata with multiple objectives. In *Computer Aided Verification - 29th*

International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I, volume 10426 of *Lecture Notes in Computer Science*, pages 140–159. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-63387-9_7, doi:10.1007/978-3-319-63387-9_7.

[QK18] Tim Quatmann and Joost-Pieter Katoen. Sound value iteration. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 643–661. Springer, 2018. URL: https://doi.org/10.1007/978-3-319-96145-3_37, doi:10.1007/978-3-319-96145-3_37.

[Ros06] J.S. Rosenthal. *A First Look at Rigorous Probability Theory.* World Scientific, 2006. URL: https://books.google.com/books?id=PYNAgzDffDMC.

[RS10] Markus N. Rabe and Sven Schewe. Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *CoRR*, abs/1004.4005, version 2, June 4, 2010. URL: http://arxiv.org/abs/1004.4005, arXiv:1004.4005.

[RS11] Markus N. Rabe and Sven Schewe. Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Informatica*, 48(5):291, July 2011. URL: https://doi.org/10.1007/s00236-011-0140-0, doi:10.1007/s00236-011-0140-0.

[SL94] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR '94, Concurrency Theory, 5th International Conference, Uppsala, Sweden, August 22-25, 1994, Proceedings*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1994. URL: https://doi.org/10.1007/978-3-540-48654-1_35, doi:10.1007/978-3-540-48654-1_35.

[vMW98] Aad P. A. van Moorsel and Katinka Wolter. Numerical solution of non-homogeneous Markov processes through uniformization. In *12^{th} European Simulation Multiconference - Simulation - Past, Present and Future, June 16-19, 1998, Machester, United Kingdom*, pages 710–717. SCS Europe, 1998.

[Yi91] Wang Yi. CCS + time = an interleaving model for real time systems. In *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, volume 510 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 1991. URL: https://doi.org/10.1007/3-540-54233-7_136, doi:10.1007/3-540-54233-7_136.

Proofs A

A.1 Auxiliary Lemmas for Section 4.1.2

Lemma A.1. *Let π be a generic measurable scheduler, $ps \in PS, b \in \mathbb{R}_{>0}$, then*

$$\forall s \in S : \mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[\texttt{tt}\,\mathrm{U}^{=b} ap_{ps}\right] = 0 \tag{A.1}$$

Proof. For $s \in S, k \in \mathbb{Z}_{\geqslant 0}$ we define $E(s,k,b) \subseteq \{\rho \mid \rho \models \texttt{tt}\,\mathrm{U}^{=b} ap_{s'}\}$ to be the set of all infinite paths ρ, such that exactly k transitions happen on ρ before reaching ps at time b. Then:

$$\{\rho \mid \rho \models \texttt{tt}\,\mathrm{U}^{=b} ap_{s'}\} = \biguplus_{k \in \mathbb{Z}_{\geqslant 0}} E(s,k,b)$$

We will show by induction that

$$\forall s \in S, k \in \mathbb{Z}_{\geqslant 0}, x \in \mathbb{R}_{>0}, \pi \in \Pi_\mu : \mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[E(s,k,x)\right] = 0$$

This implies the statement of the lemma.

Consider $k = 0$. In this case $E(s,0,x) = \emptyset$ and therefore $E(s,0,x) = \emptyset$ and $\mathrm{Pr}^{\mathcal{M}}_{\pi',s}\left[E(s,0,x)\right] = 0$.

For a scheduler $\pi' \in \Pi_\mu$ and a path fragment ϕ in the following we will define a scheduler $\pi'_{+\phi}$, such that $\rho \in \mathit{Paths}^* : \pi'_{+\phi}(\rho) = \pi'(\phi \circ \rho)$.

Assume the statement is proven for all $i \leqslant k$. Consider $i = k+1, s \in PS$:

$$\mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[E(s,k+1,x)\right] = \sum_{\alpha \in Act(s)} \pi(s)(\alpha) \sum_{s' \in S} \mathbb{P}[s,\alpha,s'] \cdot \mathrm{Pr}^{\mathcal{M}}_{\pi_{+\phi(\alpha)},s'}\left[E(s',k,x)\right]$$

where $\phi(\alpha) = s \xrightarrow{\alpha,0}$. By induction hypothesis $\forall s' \in S, \pi' \in \Pi_\mu : \mathrm{Pr}^{\mathcal{M}}_{\pi',s'}\left[E(s',k,x)\right] = 0$ and therefore $\mathrm{Pr}^{\mathcal{M}}_{\pi,s}\left[E(s,k+1,x)\right] = 0$.

Consider $i = k+1, s \in MS$.

$$\Pr^{\mathcal{M}}_{\pi,s}\left[E(s,k+1,x)\right] = \int\limits_0^x E(s)\cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \Pr^{\mathcal{M}}_{\pi_{+\phi'},s'}\left[E(s',k,x-\tau)\right] \,\mathrm{d}\tau$$

where $\phi' = s \xrightarrow{E(s),\tau}$. According to the induction hypothesis $\forall s' \in S, \pi' \in \Pi_\mu$: $\Pr^{\mathcal{M}}_{\pi',s'}\left[E(s',k,x-\tau)\right]$ is 0 for all $\tau < x$, and may be non-zero for $\tau = x$. However, due to properties of integrals, the value of the integral over interval $[0,x)$ is the same as over interval $[0,x]$. Therefore, due to the induction hypothesis $\Pr^{\mathcal{M}}_{\pi,s}\left[E(s,k+1,x)\right] = 0$. This concludes the proof. □

Consider sets $E(t,n)$ and functions $p^n_{\text{opt}}(s,t)$ defined in the proof of Lemma 4.1.3.

Lemma A.2.

$$\begin{aligned}\forall s \in S, 1 > \delta > 0, t, t' \in I_m, m \in \{0,1,2\}, |t-t'| < \delta,\\ \exists C \in \mathbb{R}_{\geqslant 0} : \left|p^n_{\text{opt}}(s,t) - p^n_{\text{opt}}(s,t')\right| \leqslant C\cdot\delta\end{aligned} \tag{A.2}$$

Proof. Let $I_0 = [0, a]\!]_a, I_1 = [\![a,b]\!], I_2 =]\!]_b b, \infty)$. The proof proceeds by induction over n. Let $n = 0$. For $s \in MS \cap \text{Sat}(\Phi_2)$ and $t \in I_0 : p^0_{\text{opt}}(s,t) = e^{-E(s)\cdot(a-t)}$. For $t \in I_1 : p^0_{\text{opt}}(s,t) = 1$ and for $t \in I_2 : p^0_{\text{opt}}(s,t) = 0$. Thus for $s \in MS \cap \text{Sat}(\Phi_2)$ (A.2) holds. For $s \in PS \cap \text{Sat}(\Phi_2), t \in I_0 \cup I_2 : p^0_{\text{opt}}(s,t) = 0$. For $t \in I_1 : p^0_{\text{opt}}(s,t) = 1$. Thus for $s \in PS \cap \text{Sat}(\Phi_2)$ (A.2) holds. For $s \in \text{Sat}(\neg\Phi_2)$ function $p^0_{\text{opt}}(s,t)$ is a constant 0 and therefore (A.2) holds.

Consider $n > 0, s \in MS \cap \text{Sat}(\neg\Phi_2)$.

$$\begin{aligned}p^n_{\text{opt}}(s,t) &= \Omega(p^{n-1}_{\text{opt}}(s,t))\\ &= \int\limits_0^{b-t} E(s)\cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s']\cdot p^{n-1}_{\text{opt}}(s',t+\tau)\,\mathrm{d}\tau\\ &= \int\limits_0^{b} E(s)\cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s']\cdot \mathbb{1}_{[0,b-t]}(\tau)\cdot p^{n-1}_{\text{opt}}(s',t+\tau)\,\mathrm{d}\tau\end{aligned}$$

For $t, t' \in I_2 : p^n_{\text{opt}}(s,t') = p^n_{\text{opt}}(s,t) = 0$. Consider $s \in MS \cap \text{Sat}(\neg\Phi_2)$, $t, t' \in I_0$ or $t, t' \in I_1$. w. l. o. g. we assume that $t \leqslant t'$. Then:

$$\begin{aligned}&\left|p^n_{\text{opt}}(s,t') - p^n_{\text{opt}}(s,t)\right|\\ &= \int\limits_0^{b} E(s)\cdot e^{-E(s)\cdot\tau} \sum_{s'\in S} \mathbb{P}[s,s']\cdot \Big(\mathbb{1}_{(b-t',b-t]}(\tau)\cdot p^{n-1}_{\text{opt}}(s',t+\tau)\\ &\qquad\qquad + \mathbb{1}_{[0,b-t']}(\tau)\cdot\left|p^{n-1}_{\text{opt}}(s',t+\tau) - p^{n-1}_{\text{opt}}(s',t'+\tau)\right|\Big)\,\mathrm{d}\tau\end{aligned}$$

Consider the first integral obtained by splitting the sum. It equals the following:

$$\int_{b-t'}^{b-t} E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s' \in S} \mathbb{P}[s,s'] \cdot p_{\text{opt}}^{n-1}(s', t+\tau)\,\mathrm{d}\tau \leqslant C \cdot \big(t'-t\big)$$

Consider the second sum. We will split the integral according to whether $t+\tau$ and $t'+\tau$ are both in the same set I_j, or in different ones:

$$\int_0^b E(s) \cdot e^{-E(s)\cdot\tau} \sum_{s' \in S} \mathbb{P}[s,s'] \cdot \mathbb{1}_{[0,b-t']}(\tau) \cdot \left|p_{\text{opt}}^{n-1}(s', t+\tau) - p_{\text{opt}}^{n-1}(s', t'+\tau)\right| \mathrm{d}\tau$$

$$= \int\limits_{(t+\tau,t'+\tau)\in I_0\times I_0} \cdots \quad + \int\limits_{(t+\tau,t'+\tau)\in I_1\times I_1} \cdots \quad + \int\limits_{(t+\tau,t'+\tau)\in I_2\times I_2} \cdots \quad + \int\limits_{\text{all the rest}} \cdots$$

If $t+\tau$ and $t'+\tau$ are in the same set I_j, then according to the induction hypothesis the respective integral is bounded by $C \cdot (t'-t)$. Otherwise, the length of the integration interval is not greater than $t'-t$ and therefore the value of the integral is bounded by $C' \cdot (t'-t)$.

The case $s \in MS \cap \mathrm{Sat}(\Phi_2)$ is treated analogously. Consider $s \in PS$. If $t, t' \in I_1$ and $s \in \mathrm{Sat}_{\Phi_2}()$, then $p_{\text{opt}}^n(s,t) = p_{\text{opt}}^n(s,t') = 1$. If $t,t' \in I_2$, then $p_{\text{opt}}^n(s,t) = p_{\text{opt}}^n(s,t') = 0$. In all other cases for $x \in \{t,t'\}$:

$$p_{\text{opt}}^n(s,x) = \Omega(p_{\text{opt}}^{n-1}(s,x)) = \underset{\alpha\in Act(s)}{\text{opt}} \sum_{s'\in S} \mathbb{P}[s,\alpha,s'] \cdot p_{\text{opt}}^{n-1}(s',x)$$

And therefore

$$\begin{aligned}\left|p_{\text{opt}}^n(s,t) - p_{\text{opt}}^n(s,t')\right| &= \left|\sum_{s'\in S} \mathbb{P}[s,\alpha,s'] \cdot p_{\text{opt}}^{n-1}(s',t) - \mathbb{P}[s,\beta,s'] \cdot p_{\text{opt}}^{n-1}(s',t')\right| \\ &\leqslant \sum_{s'\in S} \left|\mathbb{P}[s,\alpha,s'] \cdot p_{\text{opt}}^{n-1}(s',t) - \mathbb{P}[s,\beta,s'] \cdot p_{\text{opt}}^{n-1}(s',t')\right|,\end{aligned}$$

where α and β are respective optimal actions. For each of s' we define $\gamma_{s'} = \arg\max\{\mathbb{P}[s,\alpha,s'], \mathbb{P}[s,\beta,s']\}$. Then

$$\left|p_{\text{opt}}^n(s,t) - p_{\text{opt}}^n(s,t')\right| \leqslant \sum_{s'\in S} \mathbb{P}[s,\gamma_{s'},s'] \cdot \left|p_{\text{opt}}^{n-1}(s',t) - p_{\text{opt}}^{n-1}(s',t')\right| \leqslant C \cdot (t'-t)$$

This concludes the proof.

□

A.2 Auxiliary Lemmas For Lemma 4.3.2

Lemma A.3. *Consider the values $\widehat{\mathbf{D}}^i(s,\pi',h)$, defined in the proof of Lemma 4.3.2 for $\pi' \in \Pi_{\mathtt{stat}}$, $h : S \to [0,1]$. Then*

$$\forall s \in S, i \in \mathbb{Z}_{\geqslant 0} : \widehat{\mathbf{D}}^i(s,\pi',h) = \mathbf{D}^i(s,\pi',h) \tag{A.3}$$

Proof. We will prove the statement by induction. It is enough to prove it for $s \in \mathit{MS}$. Given this, the case of $ps \in \mathit{PS}$ follows from the definitions of $\widehat{\mathbf{D}}^i(s, \pi', h)$ and $\mathbf{D}^i(s, \pi', h)$.

For $i = 0, ms \in \mathit{MS}$:

$$\widehat{\mathbf{D}}^0(ms, \pi', h) = \sum_{s'' \in \mathit{MS}} \mathbb{P}'_0[ms, s''] \cdot h(s'') = h(ms) = \mathbf{D}^0(ms, \pi', h)$$

Consider $i > 0, ms \in \mathit{MS}$:

$$\begin{aligned}
&\widehat{\mathbf{D}}^i(ms, \pi', h) \\
&\quad = \sum_{s'' \in \mathit{MS}} \mathbb{P}'_i[ms, s''] \cdot h(s'') \\
&\quad = \sum_{s'' \in \mathit{MS}} \left(\sum_{s' \in \mathit{MS}} \frac{\mathbf{R}'[ms, s']}{\mathbf{E}_{\max}} \cdot \mathbb{P}'_{i-1}[s', s''] + (1 - \frac{E(ms)}{\mathbf{E}_{\max}}) \cdot \mathbb{P}'_{i-1}[ms, s''] \right) \cdot h(s'') \\
&\quad \overset{IH}{=} \sum_{s' \in \mathit{MS}} \frac{\mathbf{R}'[ms, s']}{\mathbf{E}_{\max}} \cdot \widehat{\mathbf{D}}^{i-1}(s', \pi', h) + (1 - \frac{E(ms)}{\mathbf{E}_{\max}}) \cdot \widehat{\mathbf{D}}^{i-1}(ms, \pi', h)
\end{aligned}$$

We will now rewrite the first summand according to the definition of $\mathbf{R}'$:

$$\begin{aligned}
&\sum_{s' \in \mathit{MS}} \frac{\mathbf{R}'[ms, s']}{\mathbf{E}_{\max}} \cdot \widehat{\mathbf{D}}^{i-1}(s', \pi', h) \\
&= \sum_{s' \in \mathit{MS}} \sum_{s'' \in S} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \cdot \mathrm{rea}^{\pi}(s'', s') \cdot \widehat{\mathbf{D}}^{i-1}(s', \pi', h) \\
&= \sum_{s'' \in S} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \sum_{s' \in \mathit{MS}} \mathrm{rea}^{\pi}(s'', s') \cdot \widehat{\mathbf{D}}^{i-1}(s', \pi', h) \\
&= \sum_{s'' \in \mathit{PS}} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \cdot \widehat{\mathbf{D}}^{i-1}(s'', \pi', h) \\
&\qquad + \sum_{s'' \in \mathit{MS}} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \sum_{s' \in \mathit{MS}} \mathrm{rea}^{\pi}(s'', s') \cdot \widehat{\mathbf{D}}^{i-1}(s', \pi', h) \\
&= \sum_{s'' \in \mathit{PS}} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \cdot \widehat{\mathbf{D}}^{i-1}(s'', \pi', h) \\
&\qquad + \sum_{s'' \in \mathit{MS}} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \cdot \widehat{\mathbf{D}}^{i-1}(s'', \pi', h) \\
&= \sum_{s'' \in S} \frac{\mathbf{R}[ms, s'']}{\mathbf{E}_{\max}} \cdot \widehat{\mathbf{D}}^{i-1}(s'', \pi', h)
\end{aligned}$$

□

Lemma A.4. *Let $\pi \in \Pi_{\mathtt{stat}}, v : S \to [0,1], \varepsilon \in [0,1)$. Then $\forall i \in \mathbb{Z}_{\geqslant 0}, \forall ms \in \mathit{MS}$ the following holds:*

$$\mathbf{D}^i_{\varepsilon}(ms, \pi, v) \leqslant \mathbf{D}^i(ms, \pi, v) \leqslant \mathbf{D}^i_{\varepsilon}(ms, \pi, v) + i \cdot \varepsilon \qquad \text{(A.4)}$$

Proof. The leftmost part of inequality in (A.4) can be proven via a simple inductive argument taking into account that value $\mathbf{D}^i_\varepsilon(ps, \pi, v)$ for a probabilistic state ps is an ε under-approximation of value $\mathbf{D}^i(ps, \pi, v)$.

In the following we prove the rightmost inequality in (A.4). In order to prove it we will additionally show that for a probabilistic state $ps, i \in \mathbb{Z}_{\geqslant 0}$ the following holds:

$$\mathbf{D}^i(ps, \pi, v) \leqslant \mathbf{D}^i_\varepsilon(ps, \pi, v) + (i+1) \cdot \varepsilon$$

Consider $i = 0, ms \in MS$. Then by definition $\mathbf{D}^0(ms, \pi, v) - \mathbf{D}^0_\varepsilon(ms, \pi, v) = v(ms) - v(ms) = 0$. For a probabilistic state $ps \in PS$ the value $\mathbf{D}^0_\varepsilon(ps, \pi, v)$ is an ε under-approximation of $\mathbf{D}^0(ps, \pi, v)$ and therefore $\mathbf{D}^0(ps, \pi, v) - \mathbf{D}^0_\varepsilon(ps, \pi, v) \leqslant \varepsilon$.

Let $i > 0, ms \in MS$.

$$\begin{aligned}
\mathbf{D}^i(ms, \pi, v) - \mathbf{D}^i_\varepsilon(ms, \pi, v) &= \sum_{s' \in S} \frac{\mathbf{R}[ms, s']}{\mathbf{E}_{\max}} \cdot \big(\mathbf{D}^{i-1}(s', \pi, v) - \mathbf{D}^{i-1}_\varepsilon(s', \pi, v)\big) \\
&\quad + (1 - \frac{E(ms)}{\mathbf{E}_{\max}}) \cdot \big(\mathbf{D}^{i-1}(ms, \pi, v) - \mathbf{D}^{i-1}_\varepsilon(ms, \pi, v)\big) \\
&\overset{IH}{\leqslant} i \cdot \varepsilon
\end{aligned}$$

Consider $ps \in PS$:

$$\begin{aligned}
&\mathbf{D}^i(ps, \pi, v) - \mathbf{D}^i_\varepsilon(ps, \pi, v) \\
&\quad = \mathrm{rea}^\pi(ps, \mathbf{D}^i(\pi, v)|_{MS}) - \mathrm{rea}^\pi_\varepsilon(ps, \mathbf{D}^i_\varepsilon(\pi, v)|_{MS}) \\
&\quad = \underbrace{\mathrm{rea}^\pi(ps, \mathbf{D}^i(\pi, v)|_{MS}) - \mathrm{rea}^\pi(ps, \mathbf{D}^i_\varepsilon(\pi, v)|_{MS})}_{\textit{IH: } \leqslant i \cdot \varepsilon} \\
&\qquad + \underbrace{\mathrm{rea}^\pi(ps, \mathbf{D}^i_\varepsilon(\pi, v)|_{MS}) - \mathrm{rea}^\pi_\varepsilon(ps, \mathbf{D}^i_\varepsilon(\pi, v)|_{MS})}_{\leqslant \varepsilon} \\
&\quad \leqslant (i+1) \cdot \varepsilon
\end{aligned}$$

This concludes the proof. □

A.3 Auxiliary Lemmas for Lemma 4.3.6

Lemma A.5. *Let $\mathcal{M}$ be an MA, $\varepsilon \in [0, 1]$, $\mathrm{opt} \in \{\sup, \inf\}$, $S' \subseteq S$ and $w_i : S \to [0, 1], i \in \{1, 2\}$ are goal functions defined only on S' that satisfy:*

$$\forall s \in S' : 0 \preccurlyeq_{\mathrm{opt}} w_1(s) - w_2(s) \preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon$$

Then

$$\forall ps \in PS : 0 \preccurlyeq_{\mathrm{opt}} \mathrm{rea}^{\mathrm{opt}}(ps, w_1) - \mathrm{rea}^{\mathrm{opt}}(ps, w_2) \preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon \quad \text{(A.5)}$$

Proof. According to [Put94], for any goal function g, the optimal reachability value for probabilistic states satisfies the following equation:

$$\text{rea}^{\text{opt}}(ps, g) = \lim_{n\to\infty} \text{rea}^{\text{opt}}(ps, n, g) \tag{A.6}$$

$$\text{rea}^{\text{opt}}(s, n, g) = \begin{cases} g(s) & \text{if } s \in g \\ \operatorname*{opt}_{\alpha\in Act(s)} \sum_{s'\in S} \mathbb{P}[s, \alpha, s'] \cdot \text{rea}^{\text{opt}}(s', n-1, g) & \text{else if } n > 0, s \in PS \\ 0 & \text{otherwise} \end{cases} \tag{A.7}$$

We will show by induction that $\forall n \in \mathbb{Z}_{\geqslant 0}, s \in S$:

$$0 \preccurlyeq_{\text{opt}} \text{rea}^{\text{opt}}(s, n, w_1) - \text{rea}^{\text{opt}}(s, n, w_2) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \cdot \varepsilon \tag{A.8}$$

This implies that $\forall s \in S$:

$$0 \preccurlyeq_{\text{opt}} \text{rea}^{\text{opt}}(ps, w_1) - \text{rea}^{\text{opt}}(ps, w_2) \preccurlyeq_{\text{opt}} (-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \cdot \varepsilon \tag{A.9}$$

First of all, for $n = 0$ or $s \in MS$ (A.8) holds due to the restrictions on w_i. Consider $n > 0, s \in PS$. Let π_1 be a strategy that achieves optimum in (A.7) for $g = w_1$ and π_2 is the same for $g = w_2$. Two cases are possible: $\pi_1(s) = \pi_2(s)$ and $\pi_1(s) \neq \pi_2(s)$. In case of the former:

$$\begin{aligned} &\text{rea}^{\text{opt}}(s, n, w_1) - \text{rea}^{\text{opt}}(s, n, w_2) \\ &\quad= \sum_{s'\in S} \mathbb{P}[s, \pi_1(s), s'] \cdot \big(\text{rea}^{\text{opt}}(s', n-1, w_1) - \text{rea}^{\text{opt}}(s', n-1, w_2)\big) \end{aligned}$$

The equation above is a convex combination and therefore (A.8) holds due to the induction hypothesis. Consider the case of $\pi_1(s) \neq \pi_2(s)$. By definition of π_1 and π_2 the following holds:

$$\begin{aligned} \text{rea}^{\text{opt}}(s, n, w_1) &= \sum_{s'\in S} \mathbb{P}[s, \pi_1(s), s'] \cdot \text{rea}^{\text{opt}}(s', n-1, w_1) \\ &\succcurlyeq_{\text{opt}} \sum_{s'\in S} \mathbb{P}[s, \pi_2(s), s'] \cdot \text{rea}^{\text{opt}}(s', n-1, w_1) \\ &\overset{IH}{\succcurlyeq_{\text{opt}}} \sum_{s'\in S} \mathbb{P}[s, \pi_2(s), s'] \cdot \text{rea}^{\text{opt}}(s', n-1, w_2) \\ &= \text{rea}^{\text{opt}}(s, n, w_2) \\ &\succcurlyeq_{\text{opt}} \sum_{s'\in S} \mathbb{P}[s, \pi_1(s), s'] \cdot \text{rea}^{\text{opt}}(s', n-1, w_2) \end{aligned}$$

On the other hand, the induction hypothesis also implies that

$$\begin{aligned} 0 \preccurlyeq_{\text{opt}} &\sum_{s'\in S} \mathbb{P}[s, \pi_1(s), s'] \cdot \text{rea}^{\text{opt}}(s', n-1, w_1) \\ &- \sum_{s'\in S} \mathbb{P}[s, \pi_1(s), s'] \cdot \text{rea}^{\text{opt}}(s', n-1, w_2) \\ \preccurlyeq_{\text{opt}} &(-1)^{\mathbb{1}_{\{\text{inf}\}}(\text{opt})} \cdot \varepsilon \end{aligned}$$

Thus (A.8) holds also in this case. □

Let $b \in \mathbb{R}_{>0}, t', t \in [0, b), t' > t, k \in \mathbb{Z}_{\geqslant 0}$, $\delta = (t' - t)/(k + 1)$, $\pi \in \Pi_{\mathtt{PC}}$. Let $\tau_i = t + i \cdot \delta$ for $i = 0..k + 1$. We define

$$\widetilde{\mathrm{val}}_\pi(s, x, g) := \begin{cases} \mathrm{val}_\pi(s, t, g) & \text{if } x = \tau_0 \\ e^{-E(s)\cdot\delta} \cdot \widetilde{\mathrm{val}}_\pi(s, \tau_{i-1}, g) + & \text{if } x = \tau_i, i > 0, s \in MS \\ \quad (1 - e^{-E(s)\cdot\delta}) \sum\limits_{s' \in S} \frac{\mathbf{R}[s,s']}{E(s)} \cdot \widetilde{\mathrm{val}}_\pi(s', \tau_{i-1}, g) & \\ \quad \underset{\pi' \in \Pi_{\mathtt{stat}}}{\mathrm{opt}} \sum\limits_{ms \in MS} \mathrm{rea}^{\pi'}(s, ms) \cdot \widetilde{\mathrm{val}}_\pi(ms, \tau_i, g) & \text{if } x = \tau_i, i > 0, s \in PS \\ \widetilde{\mathrm{val}}_\pi(s, \tau_i, g) & \text{if } x \in (\tau_{i-1}, \tau_i), i > 0 \end{cases}$$

Lemma A.6. *Let $\pi_{\mathtt{opt}}$ be an optimal strategy for $\mathrm{val}_{\mathrm{opt}}(\tau_i, g), i = 0..k + 1$. If Assumption 4.3.1 is satisfied, then $\forall i = 0..k + 1$:*

$$\widetilde{\mathrm{val}}_{\pi_{\mathtt{opt}}}(s, \tau_i, g) \leqslant \mathrm{val}_{\pi_{\mathtt{opt}}}(s, \tau_i, g) \leqslant \widetilde{\mathrm{val}}_{\pi_{\mathtt{opt}}}(s, \tau_i, g) + i \cdot \frac{(\mathbf{E}_{\max} \cdot \delta)^2}{2} \tag{A.10}$$

Proof. First of all, for τ_0 the statement holds by definition of $\widetilde{\mathrm{val}}_{\pi_{\mathtt{opt}}}(s, x, g)$. By fixpoint characterisation, $\forall i = 0..k, x \in (\tau_i, \tau_{i+1}], ms \in MS$:

$$\begin{aligned} \mathrm{val}_{\pi_{\mathtt{opt}}}(ms, x, g) = e^{-E(ms)\cdot(x-\tau_i)} \cdot \mathrm{val}_{\pi_{\mathtt{opt}}}(ms, \tau_i, g) \\ + \int_0^{(x-\tau_i)} E(ms) \cdot e^{-E(ms)\cdot\tau} \sum_{s \in S} \frac{\mathbf{R}[ms, s]}{E(ms)} \cdot \mathrm{val}_{\pi_{\mathtt{opt}}}(s, x - \tau, g) \, \mathrm{d}\tau \end{aligned}$$

Due to Lemma 4.3.4:

$$X(ms, \pi_{\mathtt{opt}}, x, \tau_i) \leqslant A(ms, \pi_{\mathtt{opt}}, x, \tau_i) \leqslant X(ms, \pi_{\mathtt{opt}}, x, \tau_i) + (\mathbf{E}_{\max} \cdot \delta)^2/2,$$

Thus for τ_{i+1} :

$$\begin{aligned} \mathrm{val}_{\pi_{\mathtt{opt}}}(ms, \tau_{i+1}, g) \leqslant e^{-E(ms)\cdot\delta} \cdot \mathrm{val}_{\pi_{\mathtt{opt}}}(ms, \tau_i, g) \\ + (1 - e^{-E(ms)\cdot\delta}) \sum_{s \in S} \frac{\mathbf{R}[ms, s]}{E(ms)} \cdot \mathrm{val}_{\pi_{\mathtt{opt}}}(s, \tau_i, g) + (\mathbf{E}_{\max} \cdot \delta)^2/2 \end{aligned} \tag{A.11}$$

And analogously for the lower bound:

$$\begin{aligned} \mathrm{val}_{\pi_{\mathtt{opt}}}(ms, \tau_{i+1}, g) \geqslant e^{-E(ms)\cdot\delta} \cdot \mathrm{val}_{\pi_{\mathtt{opt}}}(ms, \tau_i, g) \\ + (1 - e^{-E(ms)\cdot\delta}) \sum_{s \in S} \frac{\mathbf{R}[ms, s]}{E(ms)} \cdot \mathrm{val}_{\pi_{\mathtt{opt}}}(s, \tau_i, g) \end{aligned}$$

Therefore:

$$\begin{aligned}
&\mathrm{val}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g)\\
&\quad\leqslant e^{-E(ms)\cdot\delta}\cdot\Big(\mathrm{val}_{\pi_{\mathsf{opt}}}(ms,\tau_i,g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ms,\tau_i,g)\Big)\\
&\quad+(1-e^{-E(ms)\cdot\delta})\sum_{s\in S}\frac{\mathbf{R}[ms,s]}{E(ms)}\cdot\Big(\mathrm{val}_{\pi_{\mathsf{opt}}}(s,\tau_i,g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(s,\tau_i,g)\Big) + (\mathbf{E}_{\max}\cdot\delta)^2/2\\
&\quad\leqslant (i+1)\cdot(\mathbf{E}_{\max}\cdot\delta)^2/2
\end{aligned} \tag{A.12}$$

And analogously in the other direction:

$$\begin{aligned}
&\mathrm{val}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g)\\
&\quad\geqslant e^{-E(ms)\cdot\delta}\cdot\Big(\mathrm{val}_{\pi_{\mathsf{opt}}}(ms,\tau_i,g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ms,\tau_i,g)\Big)\\
&\quad+(1-e^{-E(ms)\cdot\delta})\sum_{s\in S}\frac{\mathbf{R}[ms,s]}{E(ms)}\cdot\Big(\mathrm{val}_{\pi_{\mathsf{opt}}}(s,\tau_i,g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(s,\tau_i,g)\Big)\\
&\quad\geqslant 0
\end{aligned} \tag{A.13}$$

Consider now a probabilistic state $ps \in PS$. Here by definition:

$$\begin{aligned}
\widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ps,\tau_{i+1},g) &= \operatorname*{opt}_{\pi'\in\Pi_{\mathsf{stat}}}\sum_{ms\in MS}\mathrm{rea}^{\pi'}(ps,ms)\cdot\widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g)\\
\mathrm{val}_{\pi_{\mathsf{opt}}}(ps,\tau_{i+1},g) &= \operatorname*{opt}_{\pi'\in\Pi_{\mathsf{stat}}}\sum_{ms\in MS}\mathrm{rea}^{\pi'}(ps,ms)\cdot\mathrm{val}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g)
\end{aligned} \tag{A.14}$$

Let $\pi_{i+1} \in \arg\mathrm{opt}_{\pi'\in\Pi_{\mathsf{stat}}}\sum_{ms\in MS}\mathrm{rea}^{\pi'}(ps,ms)\cdot\widetilde{\mathrm{val}}_{\pi_{\mathsf{opt}}}(ms,\tau_{i+1},g)$. Then for opt $=$ sup:

$$\begin{aligned}
&\mathrm{val}_{\pi_{\mathsf{sup}}}(ps,\tau_{i+1},g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{sup}}}(ps,\tau_{i+1},g)\\
&\quad\overset{(A.14)}{=}\sum_{ms\in MS}\Big(\mathrm{rea}^{\pi_{\mathsf{sup}}}(ps,ms)\cdot\mathrm{val}_{\pi_{\mathsf{sup}}}(ms,\tau_{i+1},g)\\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps,ms)\cdot\widetilde{\mathrm{val}}_{\pi_{\mathsf{sup}}}(ms,\tau_{i+1},g)\Big)\\
&\quad\overset{(A.12)}{\leqslant}\sum_{ms\in MS}\Big(\mathrm{rea}^{\pi_{\mathsf{sup}}}(ps,ms)\cdot(\widetilde{\mathrm{val}}_{\pi_{\mathsf{sup}}}(ms,\tau_{i+1},g) + (i+1)\cdot(\mathbf{E}_{\max}\cdot\delta)^2/2))\\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps,ms)\cdot\widetilde{\mathrm{val}}_{\pi_{\mathsf{sup}}}(ms,\tau_{i+1},g)\Big)\\
&\quad\leqslant\underbrace{\sum_{ms\in MS}\widetilde{\mathrm{val}}_{\pi_{\mathsf{sup}}}(ms,\tau_{i+1},g)\cdot(\mathrm{rea}^{\pi_{\mathsf{sup}}}(ps,ms) - \mathrm{rea}^{\pi_{i+1}}(ps,ms))}_{\leqslant 0}\\
&\qquad\qquad + (i+1)\cdot(\mathbf{E}_{\max}\cdot\delta)^2/2))\\
&\quad\leqslant (i+1)\cdot(\mathbf{E}_{\max}\cdot\delta)^2/2))
\end{aligned}$$

And similarly in the other direction:

$$\mathrm{val}_{\pi_{\mathsf{sup}}}(ps,\tau_{i+1},g) - \widetilde{\mathrm{val}}_{\pi_{\mathsf{sup}}}(ps,\tau_{i+1},g)$$

$$
\begin{aligned}
&= \sum_{ms \in MS} \Big(\mathrm{rea}^{\pi_{\mathtt{sup}}}(ps, ms) \cdot \mathrm{val}_{\pi_{\mathtt{sup}}}(ms, \tau_{i+1}, g) \\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \widetilde{\mathrm{val}}_{\pi_{\mathtt{sup}}}(ms, \tau_{i+1}, g) \Big) \\
&\geqslant \sum_{ms \in MS} \Big(\mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \mathrm{val}_{\pi_{\mathtt{sup}}}(ms, \tau_{i+1}, g) \\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \widetilde{\mathrm{val}}_{\pi_{\mathtt{sup}}}(ms, \tau_{i+1}, g) \Big) \\
&= \sum_{ms \in MS} \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \Big(\mathrm{val}_{\pi_{\mathtt{sup}}}(ms, \tau_{i+1}, g) - \widetilde{\mathrm{val}}_{\pi_{\mathtt{sup}}}(ms, \tau_{i+1}, g) \Big) \\
&\overset{(A.13)}{\geqslant} 0
\end{aligned}
$$

For opt = inf :

$$
\begin{aligned}
&\mathrm{val}_{\pi_{\mathtt{inf}}}(ps, \tau_{i+1}, g) - \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ps, \tau_{i+1}, g) \\
&\quad= \sum_{ms \in MS} \Big(\mathrm{rea}^{\pi_{\mathtt{inf}}}(ps, ms) \cdot \mathrm{val}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \Big) \\
&\quad\leqslant \sum_{ms \in MS} \Big(\mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \mathrm{val}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \Big) \\
&\quad= \sum_{ms \in MS} \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \Big(\mathrm{val}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) - \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \Big) \\
&\quad\overset{(A.12)}{\leqslant} (i+1) \cdot (\mathbf{E}_{\max} \cdot \delta)^2/2))
\end{aligned}
$$

And in the other direction:

$$
\begin{aligned}
&\mathrm{val}_{\pi_{\mathtt{inf}}}(ps, \tau_{i+1}, g) - \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ps, \tau_{i+1}, g) \\
&\quad= \sum_{ms \in MS} \Big(\mathrm{rea}^{\pi_{\mathtt{inf}}}(ps, ms) \cdot \mathrm{val}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \\
&\qquad\qquad - \mathrm{rea}^{\pi_{i+1}}(ps, ms) \cdot \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \Big) \\
&\quad\geqslant \sum_{ms \in MS} \Big(\mathrm{rea}^{\pi_{\mathtt{inf}}}(ps, ms) \cdot \mathrm{val}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \\
&\qquad\qquad - \mathrm{rea}^{\pi_{\mathtt{inf}}}(ps, ms) \cdot \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \Big) \\
&\quad= \sum_{ms \in MS} \mathrm{rea}^{\pi_{\mathtt{inf}}}(ps, ms) \cdot \Big(\mathrm{val}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) - \widetilde{\mathrm{val}}_{\pi_{\mathtt{inf}}}(ms, \tau_{i+1}, g) \Big) \\
&\quad\overset{(A.13)}{\geqslant} 0
\end{aligned}
$$

□

A.4 Auxiliary Lemma for Lemma 4.3.15

Lemma A.7. *Let* opt $\in \{\sup, \inf\}, \pi \in \Pi_{\texttt{stat}}$*,* $t \in [0,b), \delta \in (0, b-t]$*. If* $\mathcal{M}$ *is PS-acyclic and* $\exists \varepsilon_{\max} \geqslant 0$*, such that* $\forall ps \in PS, \alpha \in Act(ps)$*:*

$$\mathrm{diff}_{\pi}^{\mathrm{opt}}(ps, \alpha, (t, t+\delta]) \leqslant \varepsilon_{\max}$$

then $\forall ps \in PS, \forall \pi' \in \Pi_{\texttt{stat}}$*:*

$$(-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \sum_{ms \in MS} \mathrm{val}_{\pi}(ms, t, g) \cdot \Big(\mathrm{rea}^{\pi'}(ps, ms) - \mathrm{rea}^{\pi}(ps, ms)\Big) \leqslant d(ps) \cdot \varepsilon_{\max} \tag{A.15}$$

Proof. By definition

$$\begin{aligned}
&\mathrm{diff}_{\pi}^{\mathrm{opt}}(ps, \alpha, (t, t+\delta]) \\
&\quad = (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \sum_{ms \in MS} (\mathrm{rea}^{\pi, ps \to \alpha}(ps, ms) - \mathrm{rea}^{\pi}(ps, ms)) \cdot \mathrm{val}_{\pi}(ms, t+\delta, g)
\end{aligned}$$

We prove (A.15) by induction over depth of probabilistic states. Let $\pi' \in \Pi_{\texttt{stat}}$ and $ps \in PS$ has depth 1. Then

$$\begin{aligned}
&(-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \sum_{ms \in MS} \mathrm{val}_{\pi}(ms, t, g) \cdot \Big(\mathrm{rea}^{\pi'}(ps, ms) - \mathrm{rea}^{\pi}(ps, ms)\Big) \\
&= (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \sum_{ms \in MS} \mathrm{val}_{\pi}(ms, t, g) \cdot \Big(\mathbb{P}[ps, \pi'(ps), ms] - \mathbb{P}[ps, \pi(ps), ms]\Big) \\
&= \mathrm{diff}_{\pi}^{\mathrm{opt}}(ps, \pi'(ps), (t, t+\delta]) \leqslant \varepsilon_{\max} = d(ps) \cdot \varepsilon_{\max}
\end{aligned}$$

Assume that the statement holds for all states with depth $j \geqslant 1$, we prove it for a state ps of depth $j+1$.

$$\begin{aligned}
&\sum_{ms \in MS} \mathrm{val}_{\pi}(ms, t, g) \cdot \Big(\mathrm{rea}^{\pi'}(ps, ms) - \mathrm{rea}^{\pi}(ps, ms)\Big) \\
&= \sum_{s \in S} \Big(\mathbb{P}[ps, \pi'(ps), s] \sum_{ms \in MS} \mathrm{rea}^{\pi'}(s, ms) \cdot \mathrm{val}_{\pi}(ms, t, g) \\
&\qquad\qquad - \mathbb{P}[ps, \pi(ps), s] \sum_{ms \in MS} \mathrm{rea}^{\pi}(s, ms) \cdot \mathrm{val}_{\pi}(ms, t, g)\Big) \\
&= \sum_{s \in S} \mathbb{P}[ps, \pi'(ps), s] \times \\
&\qquad\qquad \sum_{ms \in MS} (\mathrm{rea}^{\pi'}(s, ms) - \mathrm{rea}^{\pi}(s, ms)) \cdot \mathrm{val}_{\pi}(ms, t, g) \\
&+ \sum_{s \in S} \big(\mathbb{P}[ps, \pi'(ps), s] - \mathbb{P}[ps, \pi(ps), s]\big) \times \\
&\qquad\qquad \sum_{ms \in MS} \mathrm{rea}^{\pi}(s, ms) \cdot \mathrm{val}_{\pi}(ms, t, g)
\end{aligned}$$

Thus

$$\begin{aligned}
&(-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \sum_{ms \in MS} \mathrm{val}_\pi(ms,t,g) \cdot \Big(\mathrm{rea}^{\pi'}(ps,ms) - \mathrm{rea}^{\pi}(ps,ms)\Big) \\
&\leqslant (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \left(\sum_{s \in S} \mathbb{P}[ps,\pi'(ps),s] \cdot d(s) \cdot \varepsilon_{\max} + \mathrm{diff}^{\sup}_{\pi}(ps,\pi'(ps),(t,t+\delta]) \right) \\
&= (j+1) \cdot \varepsilon_{\max} = d(ps) \cdot \varepsilon_{\max}
\end{aligned}$$

□

A.5 Auxiliary Lemma for Lemma 4.3.16

Lemma A.8. *Let $\pi \in \Pi_{\mathtt{stat}}, v : S \to [0,1], \varepsilon \in [0,1)$. Then $\forall i \in \mathbb{Z}_{\geqslant 0}, \forall ms \in MS$ the following holds:*

$$\overline{\mathbf{D}}^i_\varepsilon(ms,v) \preccurlyeq_{\mathrm{opt}} \overline{\mathbf{D}}^i(ms,v) \preccurlyeq_{\mathrm{opt}} \overline{\mathbf{D}}^i_\varepsilon(ps,v) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot i \cdot \varepsilon \tag{A.16}$$

Proof. The leftmost part of inequality in (A.4) can be proven via a simple inductive argument taking into account that value $\overline{\mathbf{D}}^i_\varepsilon(ps,v)$ for a probabilistic state ps is an ε under-approximation of value $\overline{\mathbf{D}}^i(ps,v)$ for opt = sup and over-approximation for opt = inf.

In the following we prove the rightmost inequality in (A.16). In order to prove it we will additionally show that for a probabilistic state $ps, i \in \mathbb{Z}_{\geqslant 0}$ the following holds:

$$\overline{\mathbf{D}}^i(ps,v) \preccurlyeq_{\mathrm{opt}} \overline{\mathbf{D}}^i_\varepsilon(ps,v) + (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot (i+1) \cdot \varepsilon$$

Consider $i = 0, ms \in MS$. Then by definition $\overline{\mathbf{D}}^0(ms,v) - \overline{\mathbf{D}}^0_\varepsilon(ms,v) = v(ms) - v(ms) = 0$. For a probabilistic state $ps \in PS$ the value $\overline{\mathbf{D}}^0_\varepsilon(ps,v)$ is an ε under- or over-approximation of $\overline{\mathbf{D}}^0(ps,v)$ and therefore $\overline{\mathbf{D}}^0(ps,v) - \overline{\mathbf{D}}^0_\varepsilon(ps,v) \preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot \varepsilon$.

Let $i > 0, ms \in MS$.

$$\begin{aligned}
\overline{\mathbf{D}}^i(ms,v) - \overline{\mathbf{D}}^i_\varepsilon(ms,v) &= \sum_{s' \in S} \frac{\mathbf{R}[ms,s']}{\mathbf{E}_{\max}} \cdot \Big(\overline{\mathbf{D}}^{i-1}(s',v) - \overline{\mathbf{D}}^{i-1}_\varepsilon(s',v)\Big) \\
&\quad + (1 - \frac{E(ms)}{\mathbf{E}_{\max}}) \cdot \Big(\overline{\mathbf{D}}^{i-1}(ms,v) - \overline{\mathbf{D}}^{i-1}_\varepsilon(ms,v)\Big) \\
&\overset{IH}{\preccurlyeq_{\mathrm{opt}}} (-1)^{\mathbb{1}_{\{\inf\}}(\mathrm{opt})} \cdot i \cdot \varepsilon
\end{aligned}$$

Consider $ps \in PS$:

$$\begin{aligned}
&\overline{\mathbf{D}}^i(ps,v) - \overline{\mathbf{D}}^i_\varepsilon(ps,v) \\
&\qquad = \mathrm{rea}^{\mathrm{opt}}(ps, \overline{\mathbf{D}}^i(v)|_{MS}) - \mathrm{rea}^{\mathrm{opt}}_\varepsilon(ps, \overline{\mathbf{D}}^i_\varepsilon(v)|_{MS})
\end{aligned}$$

$$
\begin{aligned}
&= \underbrace{\mathrm{rea}^{\mathrm{opt}}(ps, \overline{\mathbf{D}}^{i}(v)|_{MS}) - \mathrm{rea}^{\mathrm{opt}}(ps, \overline{\mathbf{D}}^{i}_{\varepsilon}(v)|_{MS})}_{\textit{IH: } \preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\mathrm{inf}\}}(\mathrm{opt})} \cdot i \cdot \varepsilon} \\
&\qquad + \underbrace{\mathrm{rea}^{\mathrm{opt}}(ps, \overline{\mathbf{D}}^{i}_{\varepsilon}(v)|_{MS}) - \mathrm{rea}^{\mathrm{opt}}_{\varepsilon}(ps, \overline{\mathbf{D}}^{i}_{\varepsilon}(v)|_{MS})}_{\preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\mathrm{inf}\}}(\mathrm{opt})} \cdot \varepsilon} \\
&\preccurlyeq_{\mathrm{opt}} (-1)^{\mathbb{1}_{\{\mathrm{inf}\}}(\mathrm{opt})} \cdot (i+1) \cdot \varepsilon
\end{aligned}
$$

This concludes the proof.

□

Experiments B

Below we provide additional data for the experimental evaluation carried out in Chapter 4, Section 4.5 and Chapter, 5 Section 5.3.5.

B.1 Command Line Arguments

B.1.1 Time-Bounded Reachability

Below we list the command line arguments used to obtain data points for all figures and tables in Section 4.5:

Figure 4.7 left:

1. modest mcsta bitcoin-attack.v1.modest -E MALICIOUS=40,CD=50 --props P_MWinMax --width 0.0005 --time-bounded-alg SwitchStep --store-mode M emory --unsafe

2. modest mcsta bitcoin-attack.v1.modest -E MALICIOUS=40,CD=50 --props P_MWinMax --width 5e-07 --time-bounded-alg SwitchStep --store-mode Me mory --unsafe

3. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

4. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=2,C=2,TIME_BOUND=2.0 --props Pm axQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-m ode Memory --unsafe

6. modest mcsta dpm.v2.modest -E N=2,C=2,TIME_BOUND=2.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
7. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=3.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
8. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=3.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
9. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=4.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
10. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=4.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
11. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
12. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
13. modest mcsta erlang.v2.modest -E K=10,R=1,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
14. modest mcsta erlang.v2.modest -E K=10,R=1,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
15. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
16. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
17. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=100 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
18. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=100 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

19. modest mcsta flexible-manufacturing.21.v1.jani -E T=1.0 --props M3Fail_Pb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

20. modest mcsta flexible-manufacturing.21.v1.jani -E T=1.0 --props M3Fail_Pb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

21. modest mcsta flexible-manufacturing.21.v1.jani -E T=10.0 --props M3Fail_Pb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

22. modest mcsta flexible-manufacturing.21.v1.jani -E T=10.0 --props M3Fail_Pb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

23. modest mcsta ftwc.v3.modest -E N=16,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

24. modest mcsta ftwc.v3.modest -E N=16,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

25. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

26. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

27. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

28. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

29. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_BOUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

30. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_BOUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

31. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

32. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

33. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

34. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

35. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

36. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

37. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

38. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

39. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

40. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

41. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

42. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

43. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

44. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

45. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

46. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

47. modest mcsta readers-writers.20.v1.jani --props prtb_many_requests --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

48. modest mcsta readers-writers.20.v1.jani --props prtb_many_requests --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

49. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=2,C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

50. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=2,C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

51. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

52. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

53. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

54. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

55. modest mcsta stream.v1.jani -E N=1000 --props pr_underrun_tb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

56. modest mcsta stream.v1.jani -E N=1000 --props pr_underrun_tb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

57. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M2Fail_Pb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

58. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M2Fail_Pb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

59. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M3Fail_Pb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

60. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M3Fail_Pb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

61. modest mcsta jobs.15-3.v1.jani --props prhalfdone --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

62. modest mcsta jobs.15-3.v1.jani --props prhalfdone --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

63. storm --jani bitcoin-attack.v1___MALICIOUS=40,CD=50.jani --janiproperty P_MWinMax --absolute --precision 0.001 --general:sound --minmax:mamethod imca

64. storm --jani bitcoin-attack.v1___MALICIOUS=40,CD=50.jani --janiproperty P_MWinMax --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

65. storm --jani dpm.v2___N=3,C=10,TIME_BOUND=30.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

66. storm --jani dpm.v2___N=3,C=10,TIME_BOUND=30.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

67. storm --jani dpm.v2___N=2,C=2,TIME_BOUND=2.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

68. storm --jani dpm.v2___N=2,C=2,TIME_BOUND=2.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

69. storm --jani dpm.v2___N=3,C=3,TIME_BOUND=3.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

70. storm --jani dpm.v2___N=3,C=3,TIME_BOUND=3.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

71. storm --jani dpm.v2___N=3,C=4,TIME_BOUND=4.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

72. storm --jani dpm.v2___N=3,C=4,TIME_BOUND=4.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

73. storm --jani dpm.v2___N=5,C=3,TIME_BOUND=50.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

74. storm --jani dpm.v2___N=5,C=3,TIME_BOUND=50.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
75. storm --jani erlang.v2___K=10,R=1,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
76. storm --jani erlang.v2___K=10,R=1,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
77. storm --jani erlang.v2___K=50000,R=100,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
78. storm --jani erlang.v2___K=50000,R=100,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
79. storm --jani erlang.v2___K=50000,R=100,TIME_BOUND=100.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
80. storm --jani erlang.v2___K=50000,R=100,TIME_BOUND=100.jani --janiproperty PmaxReachBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
81. storm --jani flexible-manufacturing.21.v1.jani --constants T=1.0 --janiproperty M3Fail_Pb --absolute --precision 0.001 --general:sound --minmax:mamethod imca
82. storm --jani flexible-manufacturing.21.v1.jani --constants T=1.0 --janiproperty M3Fail_Pb --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
83. storm --jani flexible-manufacturing.21.v1.jani --constants T=10.0 --janiproperty M3Fail_Pb --absolute --precision 0.001 --general:sound --minmax:mamethod imca
84. storm --jani flexible-manufacturing.21.v1.jani --constants T=10.0 --janiproperty M3Fail_Pb --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
85. storm --jani ftwc.v3___N=16,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
86. storm --jani ftwc.v3___N=16,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

87. storm --jani ftwc.v3___N=64,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
88. storm --jani ftwc.v3___N=64,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
89. storm --jani ftwc.v3___N=256,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
90. storm --jani ftwc.v3___N=256,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
91. storm --jani polling-system.v3___JOB_TYPES=1,C=3,TIME_BOUND=3.jani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
92. storm --jani polling-system.v3___JOB_TYPES=1,C=3,TIME_BOUND=3.jani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
93. storm --jani polling-system.v3___JOB_TYPES=2,C=3,TIME_BOUND=3.jani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
94. storm --jani polling-system.v3___JOB_TYPES=2,C=3,TIME_BOUND=3.jani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
95. storm --jani polling-system.v3___JOB_TYPES=1,C=3,TIME_BOUND=4.jani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
96. storm --jani polling-system.v3___JOB_TYPES=1,C=3,TIME_BOUND=4.jani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
97. storm --jani polling-system.v3___JOB_TYPES=2,C=3,TIME_BOUND=4.jani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
98. storm --jani polling-system.v3___JOB_TYPES=2,C=3,TIME_BOUND=4.jani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca
99. storm --jani polling-system.v3___JOB_TYPES=1,C=4,TIME_BOUND=3.jani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

100. storm --jani polling-system.v3___JOB_TYPES=1,C=4,TIME_BOUND=3.j ani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general: sound --minmax:mamethod imca

101. storm --jani polling-system.v3___JOB_TYPES=1,C=4,TIME_BOUND=4.j ani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general: sound --minmax:mamethod imca

102. storm --jani polling-system.v3___JOB_TYPES=1,C=4,TIME_BOUND=4.j ani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general: sound --minmax:mamethod imca

103. storm --jani polling-system.v3___JOB_TYPES=2,C=4,TIME_BOUND=4.j ani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general: sound --minmax:mamethod imca

104. storm --jani polling-system.v3___JOB_TYPES=2,C=4,TIME_BOUND=4.j ani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general: sound --minmax:mamethod imca

105. storm --jani polling-system.v3___JOB_TYPES=3,C=3,TIME_BOUND=5.j ani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general: sound --minmax:mamethod imca

106. storm --jani polling-system.v3___JOB_TYPES=3,C=3,TIME_BOUND=5.j ani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general: sound --minmax:mamethod imca

107. storm --jani polling-system.v3___JOB_TYPES=6,C=3,TIME_BOUND=5.j ani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general: sound --minmax:mamethod imca

108. storm --jani polling-system.v3___JOB_TYPES=6,C=3,TIME_BOUND=5.j ani --janiproperty PmaxBothFullBound --absolute --precision 1e-06 --general: sound --minmax:mamethod imca

109. storm --jani readers-writers.20.v1.jani --janiproperty prtb__many__requests -- absolute --precision 0.001 --general:sound --minmax:mamethod imca

110. storm --jani readers-writers.20.v1.jani --janiproperty prtb__many__requests -- absolute --precision 1e-06 --general:sound --minmax:mamethod imca

111. storm --jani reentrant-queues.v3___JOB_TYPES=1,C_LEFT=2,C_RIGH T=2,TIME_BOUND=1.jani --janiproperty PmaxBothQueuesFullBound --a bsolute --precision 0.001 --general:sound --minmax:mamethod imca

112. storm --jani reentrant-queues.v3___JOB_TYPES=1,C_LEFT=2,C_RIGH T=2,TIME_BOUND=1.jani --janiproperty PmaxBothQueuesFullBound --a bsolute --precision 1e-06 --general:sound --minmax:mamethod imca

113. storm --jani reentrant-queues.v3___JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5.jani --janiproperty PmaxBothQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

114. storm --jani reentrant-queues.v3___JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5.jani --janiproperty PmaxBothQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

115. storm --jani reentrant-queues.v3___JOB_TYPES=5,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5.jani --janiproperty PmaxBothQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

116. storm --jani reentrant-queues.v3___JOB_TYPES=5,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5.jani --janiproperty PmaxBothQueuesFullBound --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

117. storm --jani stream.v1.jani --constants N=1000 --janiproperty pr_underrun_tb --absolute --precision 0.001 --general:sound --minmax:mamethod imca

118. storm --jani stream.v1.jani --constants N=1000 --janiproperty pr_underrun_tb --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

119. storm --jani flexible-manufacturing.21.v1.jani --constants T=1 --janiproperty M2Fail_Pb --absolute --precision 0.001 --general:sound --minmax:mamethod imca

120. storm --jani flexible-manufacturing.21.v1.jani --constants T=1 --janiproperty M2Fail_Pb --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

121. storm --jani flexible-manufacturing.21.v1.jani --constants T=1 --janiproperty M3Fail_Pb --absolute --precision 0.001 --general:sound --minmax:mamethod imca

122. storm --jani flexible-manufacturing.21.v1.jani --constants T=1 --janiproperty M3Fail_Pb --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

123. storm --jani jobs.15-3.v1.jani --janiproperty prhalfdone --absolute --precision 0.001 --general:sound --minmax:mamethod imca

124. storm --jani jobs.15-3.v1.jani --janiproperty prhalfdone --absolute --precision 1e-06 --general:sound --minmax:mamethod imca

Figure 4.7 right:

1. modest mcsta bitcoin-attack.v1.modest -E MALICIOUS=40,CD=50 --props P_MWinMax --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

2. modest mcsta bitcoin-attack.v1.modest -E MALICIOUS=40,CD=50 --props P_MWinMax --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
3. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
4. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
5. modest mcsta dpm.v2.modest -E N=2,C=2,TIME_BOUND=2.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
6. modest mcsta dpm.v2.modest -E N=2,C=2,TIME_BOUND=2.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
7. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=3.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
8. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=3.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
9. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=4.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
10. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=4.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
11. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
12. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
13. modest mcsta erlang.v2.modest -E K=10,R=1,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
14. modest mcsta erlang.v2.modest -E K=10,R=1,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

15. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
16. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
17. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=100 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
18. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=100 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
19. modest mcsta flexible-manufacturing.21.v1.jani -E T=1.0 --props M3Fail_Pb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
20. modest mcsta flexible-manufacturing.21.v1.jani -E T=1.0 --props M3Fail_Pb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
21. modest mcsta flexible-manufacturing.21.v1.jani -E T=10.0 --props M3Fail_Pb --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
22. modest mcsta flexible-manufacturing.21.v1.jani -E T=10.0 --props M3Fail_Pb --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
23. modest mcsta ftwc.v3.modest -E N=16,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
24. modest mcsta ftwc.v3.modest -E N=16,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
25. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
26. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
27. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

28. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

29. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

30. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe

31. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

32. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe

33. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

34. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe

35. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

36. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe

37. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_B OUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

38. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_B OUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe

39. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_B OUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

40. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_B OUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe

41. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
42. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
43. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
44. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
45. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
46. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
47. modest mcsta readers-writers.20.v1.jani --props prtb_many_requests --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
48. modest mcsta readers-writers.20.v1.jani --props prtb_many_requests --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
49. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=2,C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
50. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=2,C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
51. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
52. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
53. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

54. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2, C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

55. modest mcsta stream.v1.jani -E N=1000 --props pr_underrun_tb --width 0. 0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

56. modest mcsta stream.v1.jani -E N=1000 --props pr_underrun_tb --width 5e- 07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

57. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M2Fail_Pb - -width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

58. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M2Fail_Pb - -width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

59. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M3Fail_Pb - -width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

60. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M3Fail_Pb - -width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

61. modest mcsta jobs.15-3.v1.jani --props prhalfdone --width 0.0005 --time-bou nded-alg SwitchStep --store-mode Memory --unsafe

62. modest mcsta jobs.15-3.v1.jani --props prhalfdone --width 5e-07 --time-boun ded-alg SwitchStep --store-mode Memory --unsafe

63. modest mcsta bitcoin-attack.v1.modest -E MALICIOUS=40,CD=50 --props P_MWinMax --width 0.0005 --time-bounded-alg UnifPlus --store-mode Mem ory --unsafe

64. modest mcsta bitcoin-attack.v1.modest -E MALICIOUS=40,CD=50 --props P_MWinMax --width 5e-07 --time-bounded-alg UnifPlus --store-mode Mem ory --unsafe

65. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-m ode Memory --unsafe

66. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

67. modest mcsta dpm.v2.modest -E N=2,C=2,TIME_BOUND=2.0 --props Pm axQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mod e Memory --unsafe

68. modest mcsta dpm.v2.modest -E N=2,C=2,TIME_BOUND=2.0 --props Pm axQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

69. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=3.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

70. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=3.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

71. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=4.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

72. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=4.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

73. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

74. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

75. modest mcsta erlang.v2.modest -E K=10,R=1,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

76. modest mcsta erlang.v2.modest -E K=10,R=1,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

77. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

78. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

79. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=100 --props PmaxReachBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

80. modest mcsta erlang.v2.modest -E K=50000,R=100,TIME_BOUND=100 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

81. modest mcsta flexible-manufacturing.21.v1.jani -E T=1.0 --props M3Fail_Pb --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

82. modest mcsta flexible-manufacturing.21.v1.jani -E T=1.0 --props M3Fail_P b --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

83. modest mcsta flexible-manufacturing.21.v1.jani -E T=10.0 --props M3Fail_P b --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

84. modest mcsta flexible-manufacturing.21.v1.jani -E T=10.0 --props M3Fail_ Pb --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

85. modest mcsta ftwc.v3.modest -E N=16,TIME_BOUND=5 --props PmaxRea chBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory -- unsafe

86. modest mcsta ftwc.v3.modest -E N=16,TIME_BOUND=5 --props PmaxRea chBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory -- unsafe

87. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxRea chBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory -- unsafe

88. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxRea chBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory -- unsafe

89. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxR eachBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memor y --unsafe

90. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxR eachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

91. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg U nifPlus --store-mode Memory --unsafe

92. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

93. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg U nifPlus --store-mode Memory --unsafe

94. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

95. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg U nifPlus --store-mode Memory --unsafe

96. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
97. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
98. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
99. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=3 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
100. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=3 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
101. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
102. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
103. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
104. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=4,TIME_BOUND=4 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
105. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
106. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
107. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
108. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

109. modest mcsta readers-writers.20.v1.jani --props prtb_many_requests --widt h 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

110. modest mcsta readers-writers.20.v1.jani --props prtb_many_requests --widt h 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

111. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=2, C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --wid th 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

112. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=2, C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

113. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --wid th 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

114. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

115. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2, C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --wid th 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

116. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2, C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

117. modest mcsta stream.v1.jani -E N=1000 --props pr_underrun_tb --width 0. 0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

118. modest mcsta stream.v1.jani -E N=1000 --props pr_underrun_tb --width 5e- 07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

119. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M2Fail_Pb - -width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

120. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M2Fail_Pb - -width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

121. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M3Fail_Pb - -width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

122. modest mcsta flexible-manufacturing.21.v1.jani -E T=1 --props M3Fail_Pb - -width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

123. modest mcsta jobs.15-3.v1.jani --props prhalfdone --width 0.0005 --time-bou nded-alg UnifPlus --store-mode Memory --unsafe

124. modest mcsta jobs.15-3.v1.jani --props prhalfdone --width 5e-07 --time-boun ded-alg UnifPlus --store-mode Memory --unsafe

Table 4.1:

1. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=50.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
2. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
3. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
4. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
5. storm --jani dpm.v2___N=5,C=3,TIME_BOUND=50.0.jani --janiproperty PmaxQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
6. storm --jani reentrant-queues.v3___JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=5.jani --janiproperty PmaxBothQueuesFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
7. storm --jani polling-system.v3___JOB_TYPES=3,C=3,TIME_BOUND=5.jani --janiproperty PmaxBothFullBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca
8. storm --jani ftwc.v3___N=64,TIME_BOUND=5.jani --janiproperty PmaxReachBound --absolute --precision 0.001 --general:sound --minmax:mamethod imca

Figure (4.8a):

1. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
2. modest mcsta dpm.v2.modest -E N=3,C=6,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
3. modest mcsta dpm.v2.modest -E N=3,C=8,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
4. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=3,C=12,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
6. modest mcsta dpm.v2.modest -E N=3,C=14,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
7. modest mcsta ftwc.v3.modest -E N=32,TIME_BOUND=5 --props PmaxRea chBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
8. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxRea chBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
9. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe
10. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe
11. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=1,TIME_B OUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe
12. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe
13. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=5,TIME_B OUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe
14. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=7,TIME_B OUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe
15. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=9,TIME_B OUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Sw itchStep --store-mode Memory --unsafe
16. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=1, C_RIGHT=1,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
17. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

18. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=3, C_RIGHT=3,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

19. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=4, C_RIGHT=4,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

20. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5, C_RIGHT=5,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

21. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

22. modest mcsta dpm.v2.modest -E N=3,C=6,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

23. modest mcsta dpm.v2.modest -E N=3,C=8,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

24. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

25. modest mcsta dpm.v2.modest -E N=3,C=12,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

26. modest mcsta dpm.v2.modest -E N=3,C=14,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mo de Memory --unsafe

27. modest mcsta ftwc.v3.modest -E N=32,TIME_BOUND=5 --props PmaxRea chBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory -- unsafe

28. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=5 --props PmaxRea chBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory -- unsafe

29. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxR eachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

30. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=5 --props PmaxR eachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

31. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=1,TIME_BOUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

32. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

33. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=5,TIME_BOUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

34. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=7,TIME_BOUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

35. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=9,TIME_BOUND=8 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

36. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=1,C_RIGHT=1,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

37. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

38. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=3,C_RIGHT=3,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

39. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=4,C_RIGHT=4,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

40. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5,C_RIGHT=5,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

Figure (4.8b): List of command line arguments to obtain data points for Fig.(4.8b):

1. modest mcsta dpm.v2.modest -E N=2,C=3,TIME_BOUND=5.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

2. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=5.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

3. modest mcsta dpm.v2.modest -E N=4,C=3,TIME_BOUND=5.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
4. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=5.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
5. modest mcsta dpm.v2.modest -E N=6,C=3,TIME_BOUND=5.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
6. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
7. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
8. modest mcsta polling-system.v3.modest -E JOB_TYPES=4,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
9. modest mcsta polling-system.v3.modest -E JOB_TYPES=5,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
10. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
11. modest mcsta polling-system.v3.modest -E JOB_TYPES=7,C=3,TIME_BOUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
12. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
13. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
14. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=4,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
15. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

16. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=6,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

17. modest mcsta dpm.v2.modest -E N=2,C=3,TIME_BOUND=5.0 --props Pm axQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

18. modest mcsta dpm.v2.modest -E N=3,C=3,TIME_BOUND=5.0 --props Pm axQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

19. modest mcsta dpm.v2.modest -E N=4,C=3,TIME_BOUND=5.0 --props Pm axQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

20. modest mcsta dpm.v2.modest -E N=5,C=3,TIME_BOUND=5.0 --props Pm axQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

21. modest mcsta dpm.v2.modest -E N=6,C=3,TIME_BOUND=5.0 --props Pm axQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

22. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_B OUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

23. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_B OUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

24. modest mcsta polling-system.v3.modest -E JOB_TYPES=4,C=3,TIME_B OUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

25. modest mcsta polling-system.v3.modest -E JOB_TYPES=5,C=3,TIME_B OUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

26. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_B OUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

27. modest mcsta polling-system.v3.modest -E JOB_TYPES=7,C=3,TIME_B OUND=5 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg Un ifPlus --store-mode Memory --unsafe

28. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

29. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

30. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=4,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

31. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

32. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=6,C_LEFT=2, C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

Figure (4.8c):

1. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=18.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

2. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=20.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

3. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=22.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

4. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=24.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=26.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

6. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=28.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

7. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

8. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=32.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

9. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=34.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

10. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=10 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

11. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=13 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

12. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=16 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

13. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=19 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

14. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=22 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

15. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=25 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

16. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=28 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

17. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=31 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

18. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=34 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

19. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=37 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

20. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=40 --props PmaxR eachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

21. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_B OUND=10 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg S witchStep --store-mode Memory --unsafe

22. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=14 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
23. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=18 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
24. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=22 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
25. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=26 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
26. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=30 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
27. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=34 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
28. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=38 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
29. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=44 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
30. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=48 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
31. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=52 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
32. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=56 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
33. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=60 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
34. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=64 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

35. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=68 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

36. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=72 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

37. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

38. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=2 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

39. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

40. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=4 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

41. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

42. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=6 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

43. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=7 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

44. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=8 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

45. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=9 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

46. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=18.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

47. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=20.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

48. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=22.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
49. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=24.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
50. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=26.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
51. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=28.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
52. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
53. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=32.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
54. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=34.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
55. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=10 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
56. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=13 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
57. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=16 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
58. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=19 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
59. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=22 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
60. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=25 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

61. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=28 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

62. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=31 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

63. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=34 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

64. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=37 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

65. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=40 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

66. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=10 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

67. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=14 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

68. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=18 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

69. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=22 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

70. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=26 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

71. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=30 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

72. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=34 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

73. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=38 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

74. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=44 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

75. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=48 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

76. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=52 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

77. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=56 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

78. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=60 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

79. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=64 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

80. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=68 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

81. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=72 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

82. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

83. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=2 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

84. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=3 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

85. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=4 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

86. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=5 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

87. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2, C_RIGHT=2,TIME_BOUND=6 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

88. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2, C_RIGHT=2,TIME_BOUND=7 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

89. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2, C_RIGHT=2,TIME_BOUND=8 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

90. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2, C_RIGHT=2,TIME_BOUND=9 --props PmaxBothQueuesFullBound --wid th 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

Figure (4.8d):

1. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

2. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-05 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

3. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-06 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

4. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-08 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

6. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-09 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

7. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-10 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

8. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props P maxQueuesFullBound --width 5e-11 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

9. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
10. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-05 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
11. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-06 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
12. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
13. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-08 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
14. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-09 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
15. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-10 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
16. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-11 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
17. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
18. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-05 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
19. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-06 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
20. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe
21. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-08 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

22. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-09 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

23. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-10 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

24. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-11 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

25. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

26. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-05 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

27. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-06 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

28. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

29. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-08 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

30. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-09 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

31. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-10 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

32. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-11 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

33. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

34. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-05 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

35. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-06 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
36. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
37. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-08 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
38. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-09 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
39. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-10 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
40. modest mcsta dpm.v2.modest -E N=3,C=4,TIME_BOUND=30.0 --props PmaxQueuesFullBound --width 5e-11 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
41. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
42. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-05 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
43. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-06 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
44. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
45. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-08 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
46. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-09 --time-bounded-alg UnifPlus --store-mode Memory --unsafe
47. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-10 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

48. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=5 --props PmaxReachBound --width 5e-11 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

49. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

50. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-05 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

51. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-06 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

52. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

53. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-08 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

54. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-09 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

55. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-10 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

56. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=7 --props PmaxBothFullBound --width 5e-11 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

57. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 0.0005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

58. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-05 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

59. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-06 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

60. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --width 5e-07 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

61. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --wid th 5e-08 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

62. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --wid th 5e-09 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

63. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --wid th 5e-10 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

64. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=1 --props PmaxBothQueuesFullBound --wid th 5e-11 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

Tables 4.4 and 4.5:

1. modest mcsta polling-system-k.v3.modest -E JOB_TYPES=1,C=20,TIME_ BOUND=4 --props PmaxOneFullBound --width 0.005 --partial Simulation -- partial-max-run-length 0 --partial-sim Simple --time-bounded-alg UnifPlus -- store-mode Memory --unsafe

2. modest mcsta dpm-full.v2.modest -E N=20,C=100,TIME_BOUND=6 --pro ps PmaxOneQueueThreeBound --width 0.005 --partial Simulation --partial- max-run-length 0 --partial-sim Simple --time-bounded-alg UnifPlus --store-m ode Memory --unsafe

3. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5, C_RIGHT=5,TIME_BOUND=10 --props PmaxBothQueuesFullBound --wi dth 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simpl e --time-bounded-alg UnifPlus --store-mode Memory --unsafe

4. modest mcsta vgs.5.v1.jani -E TIME_BOUND=10000.0 --props MaxPrReac hFailedTB --width 0.005 --partial Simulation --partial-max-run-length 0 --pa rtial-sim Simple --time-bounded-alg UnifPlus --store-mode Memory --unsafe

5. modest mcsta stream.v1.jani -E N=20000 --props pr_underrun_tb --width 0. 005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --tim e-bounded-alg UnifPlus --store-mode Memory --unsafe

6. modest mcsta ftwc.v3.modest -E N=512,TIME_BOUND=10 --props PmaxR eachBound --width 0.005 --partial Simulation --partial-max-run-length 0 --pa rtial-sim Simple --time-bounded-alg UnifPlus --store-mode Memory --unsafe

7. modest mcsta hecs.false-4-3.v1.jani --props Unreliability --width 0.005 --parti al Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded -alg UnifPlus --store-mode Memory --unsafe

8. modest mcsta polling-system-k.v3.modest -E JOB_TYPES=1,C=20,TIME_BOUND=4 --props PmaxOneFullBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
9. modest mcsta dpm-full.v2.modest -E N=20,C=100,TIME_BOUND=6 --props PmaxOneQueueThreeBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
10. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5,C_RIGHT=5,TIME_BOUND=10 --props PmaxBothQueuesFullBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
11. modest mcsta vgs.5.v1.jani -E TIME_BOUND=10000.0 --props MaxPrReachFailedTB --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
12. modest mcsta stream.v1.jani -E N=20000 --props pr_underrun_tb --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
13. modest mcsta ftwc.v3.modest -E N=512,TIME_BOUND=10 --props PmaxReachBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
14. modest mcsta hecs.false-4-3.v1.jani --props Unreliability --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Uniform --store-mode Memory --unsafe
15. modest mcsta polling-system-k.v3.modest -E JOB_TYPES=1,C=20,TIME_BOUND=4 --props PmaxOneFullBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Optimal --store-mode Memory --unsafe
16. modest mcsta dpm-full.v2.modest -E N=20,C=100,TIME_BOUND=6 --props PmaxOneQueueThreeBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Optimal --store-mode Memory --unsafe
17. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5,C_RIGHT=5,TIME_BOUND=10 --props PmaxBothQueuesFullBound --width 0.005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Optimal --store-mode Memory --unsafe

18. modest mcsta vgs.5.v1.jani -E TIME_BOUND=10000.0 --props MaxPrReac hFailedTB --width 0.005 --partial Simulation --partial-max-run-length 0 --pa rtial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Opti mal --store-mode Memory --unsafe

19. modest mcsta stream.v1.jani -E N=20000 --props pr_underrun_tb --width 0. 005 --partial Simulation --partial-max-run-length 0 --partial-sim Simple --tim e-bounded-alg SwitchStep --partial-sim-scheduler Optimal --store-mode Mem ory --unsafe

20. modest mcsta ftwc.v3.modest -E N=512,TIME_BOUND=10 --props Pmax ReachBound --width 0.005 --partial Simulation --partial-max-run-length 0 -- partial-sim Simple --time-bounded-alg SwitchStep --partial-sim-scheduler Op timal --store-mode Memory --unsafe

21. modest mcsta hecs.false-4-3.v1.jani --props Unreliability --width 0.005 --parti al Simulation --partial-max-run-length 0 --partial-sim Simple --time-bounded -alg SwitchStep --partial-sim-scheduler Optimal --store-mode Memory --unsa fe

22. modest mcsta polling-system-k.v3.modest -E JOB_TYPES=1,C=20,TIME_ BOUND=4 --props PmaxOneFullBound --width 0.005 --time-bounded-alg S witchStep --store-mode Memory --unsafe

23. modest mcsta dpm-full.v2.modest -E N=20,C=100,TIME_BOUND=6 --pro ps PmaxOneQueueThreeBound --width 0.005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

24. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5, C_RIGHT=5,TIME_BOUND=10 --props PmaxBothQueuesFullBound --wi dth 0.005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

25. modest mcsta vgs.5.v1.jani -E TIME_BOUND=10000.0 --props MaxPrReac hFailedTB --width 0.005 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

26. modest mcsta stream.v1.jani -E N=20000 --props pr_underrun_tb --width 0. 005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

27. modest mcsta ftwc.v3.modest -E N=512,TIME_BOUND=10 --props PmaxR eachBound --width 0.005 --time-bounded-alg SwitchStep --store-mode Memo ry --unsafe

28. modest mcsta hecs.false-4-3.v1.jani --props Unreliability --width 0.005 --time-bounded-alg SwitchStep --store-mode Memory --unsafe

29. modest mcsta polling-system-k.v3.modest -E JOB_TYPES=1,C=20,TIME_ BOUND=4 --props PmaxOneFullBound --width 0.005 --time-bounded-alg U nifPlus --store-mode Memory --unsafe

30. modest mcsta dpm-full.v2.modest -E N=20,C=100,TIME_BOUND=6 --pro ps PmaxOneQueueThreeBound --width 0.005 --time-bounded-alg UnifPlus -- store-mode Memory --unsafe

31. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=5, C_RIGHT=5,TIME_BOUND=10 --props PmaxBothQueuesFullBound --wi dth 0.005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

32. modest mcsta vgs.5.v1.jani -E TIME_BOUND=10000.0 --props MaxPrReac hFailedTB --width 0.005 --time-bounded-alg UnifPlus --store-mode Memory - -unsafe

33. modest mcsta stream.v1.jani -E N=20000 --props pr_underrun_tb --width 0. 005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

34. modest mcsta ftwc.v3.modest -E N=512,TIME_BOUND=10 --props PmaxR eachBound --width 0.005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

35. modest mcsta hecs.false-4-3.v1.jani --props Unreliability --width 0.005 --time-bounded-alg UnifPlus --store-mode Memory --unsafe

B.1.2 Long-Run Average Rewards

Below we list the command line arguments used to obtain data points for all figures and tables in Section 4.5:

Figure (5.5a):

1. modest mcsta dpm.v2.modest -E N=3,C=8,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

2. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=0.0 --props M inAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mod e Memory --unsafe

3. modest mcsta dpm.v2.modest -E N=3,C=12,TIME_BOUND=0.0 --props M inAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mod e Memory --unsafe

4. modest mcsta dpm.v2.modest -E N=3,C=14,TIME_BOUND=0.0 --props M inAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mod e Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=3,C=30,TIME_BOUND=0.0 --props M inAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mod e Memory --unsafe

6. modest mcsta dpm.v2.modest -E N=3,C=35,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
7. modest mcsta dpm.v2.modest -E N=3,C=40,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
8. modest mcsta dpm.v2.modest -E N=3,C=45,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
9. modest mcsta dpm.v2.modest -E N=3,C=50,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
10. modest mcsta dpm.v2.modest -E N=3,C=55,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
11. modest mcsta ftwc.v3.modest -E N=32,TIME_BOUND=0.0 --props SmaxReach --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
12. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props SmaxReach --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
13. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=0.0 --props SmaxReach --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
14. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=0.0 --props SmaxReach --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
15. modest mcsta ftwc.v3.modest -E N=32,TIME_BOUND=0.0 --props MinAvgRepairCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
16. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAvgRepairCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
17. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=0.0 --props MinAvgRepairCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
18. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=0.0 --props MinAvgRepairCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

19. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=1,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

20. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=2,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

21. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

22. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

23. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

24. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=6,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

25. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=7,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

26. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=1,C_RIGHT=1,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

27. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

28. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

29. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=4,C_RIGHT=4,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

30. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=5,C_RIGHT=5,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

31. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=6,C_RIGHT=6,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

32. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=1, C_RIGHT=1,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
33. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2, C_RIGHT=2,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
34. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
35. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=4, C_RIGHT=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
36. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=5, C_RIGHT=5,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
37. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=6, C_RIGHT=6,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
38. modest mcsta dpm.v2.modest -E N=3,C=8,TIME_BOUND=0.0 --props Mi nAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe
39. modest mcsta dpm.v2.modest -E N=3,C=10,TIME_BOUND=0.0 --props M inAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe
40. modest mcsta dpm.v2.modest -E N=3,C=12,TIME_BOUND=0.0 --props M inAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe
41. modest mcsta dpm.v2.modest -E N=3,C=14,TIME_BOUND=0.0 --props M inAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe
42. modest mcsta dpm.v2.modest -E N=3,C=30,TIME_BOUND=0.0 --props M inAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe
43. modest mcsta dpm.v2.modest -E N=3,C=35,TIME_BOUND=0.0 --props M inAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe
44. modest mcsta dpm.v2.modest -E N=3,C=40,TIME_BOUND=0.0 --props M inAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe

45. modest mcsta dpm.v2.modest -E N=3,C=45,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
46. modest mcsta dpm.v2.modest -E N=3,C=50,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
47. modest mcsta dpm.v2.modest -E N=3,C=55,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
48. modest mcsta ftwc.v3.modest -E N=32,TIME_BOUND=0.0 --props SmaxReach --long-run-alg LinearProgramming --store-mode Memory --unsafe
49. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props SmaxReach --long-run-alg LinearProgramming --store-mode Memory --unsafe
50. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=0.0 --props SmaxReach --long-run-alg LinearProgramming --store-mode Memory --unsafe
51. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=0.0 --props SmaxReach --long-run-alg LinearProgramming --store-mode Memory --unsafe
52. modest mcsta ftwc.v3.modest -E N=32,TIME_BOUND=0.0 --props MinAvgRepairCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
53. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAvgRepairCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
54. modest mcsta ftwc.v3.modest -E N=128,TIME_BOUND=0.0 --props MinAvgRepairCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
55. modest mcsta ftwc.v3.modest -E N=256,TIME_BOUND=0.0 --props MinAvgRepairCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
56. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=1,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
57. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=2,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
58. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

59. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
60. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=5,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
61. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=6,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
62. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=7,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
63. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=1,C_RIGHT=1,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe
64. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe
65. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe
66. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=4,C_RIGHT=4,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe
67. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=5,C_RIGHT=5,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe
68. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=6,C_RIGHT=6,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe
69. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=1,C_RIGHT=1,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
70. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=2,C_RIGHT=2,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
71. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

72. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=4, C_RIGHT=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --lo ng-run-alg LinearProgramming --store-mode Memory --unsafe

73. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=5, C_RIGHT=5,TIME_BOUND=0.0 --props MinAverageOperationCosts --lo ng-run-alg LinearProgramming --store-mode Memory --unsafe

74. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=6, C_RIGHT=6,TIME_BOUND=0.0 --props MinAverageOperationCosts --lo ng-run-alg LinearProgramming --store-mode Memory --unsafe

Figure (5.5b):

1. modest mcsta dpm.v2.modest -E N=1,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

2. modest mcsta dpm.v2.modest -E N=2,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

3. modest mcsta dpm.v2.modest -E N=3,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

4. modest mcsta dpm.v2.modest -E N=4,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=5,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

6. modest mcsta dpm.v2.modest -E N=6,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

7. modest mcsta dpm.v2.modest -E N=7,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

8. modest mcsta dpm.v2.modest -E N=8,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

9. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

10. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
11. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
12. modest mcsta polling-system.v3.modest -E JOB_TYPES=4,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
13. modest mcsta polling-system.v3.modest -E JOB_TYPES=5,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
14. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
15. modest mcsta polling-system.v3.modest -E JOB_TYPES=7,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
16. modest mcsta polling-system.v3.modest -E JOB_TYPES=8,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
17. modest mcsta polling-system.v3.modest -E JOB_TYPES=9,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
18. modest mcsta polling-system.v3.modest -E JOB_TYPES=10,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
19. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
20. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
21. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
22. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=4,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

23. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

24. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=6,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

25. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=7,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --wi dth 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

26. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

27. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

28. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

29. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=4,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

30. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

31. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=6,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

32. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=7,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e -07 --long-run-alg ValueIteration --store-mode Memory --unsafe

33. modest mcsta dpm.v2.modest -E N=1,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe

34. modest mcsta dpm.v2.modest -E N=2,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe

35. modest mcsta dpm.v2.modest -E N=3,C=5,TIME_BOUND=0.0 --props Mi nAvgOperationCost --long-run-alg LinearProgramming --store-mode Memor y --unsafe

36. modest mcsta dpm.v2.modest -E N=4,C=5,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
37. modest mcsta dpm.v2.modest -E N=5,C=5,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
38. modest mcsta dpm.v2.modest -E N=6,C=5,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
39. modest mcsta dpm.v2.modest -E N=7,C=5,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
40. modest mcsta dpm.v2.modest -E N=8,C=5,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
41. modest mcsta polling-system.v3.modest -E JOB_TYPES=1,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
42. modest mcsta polling-system.v3.modest -E JOB_TYPES=2,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
43. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
44. modest mcsta polling-system.v3.modest -E JOB_TYPES=4,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
45. modest mcsta polling-system.v3.modest -E JOB_TYPES=5,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
46. modest mcsta polling-system.v3.modest -E JOB_TYPES=6,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
47. modest mcsta polling-system.v3.modest -E JOB_TYPES=7,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe
48. modest mcsta polling-system.v3.modest -E JOB_TYPES=8,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

49. modest mcsta polling-system.v3.modest -E JOB_TYPES=9,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

50. modest mcsta polling-system.v3.modest -E JOB_TYPES=10,C=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

51. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

52. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

53. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

54. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=4,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

55. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

56. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=6,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

57. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=7,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

58. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=1,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

59. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=2,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

60. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

61. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=4,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

62. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=5,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

63. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=6,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

64. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=7,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

Figure (5.6):

1. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 0.0005 --long-run-alg ValueIteration --store-mode Memory --unsafe

2. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-05 --long-run-alg ValueIteration --store-mode Memory --unsafe

3. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-06 --long-run-alg ValueIteration --store-mode Memory --unsafe

4. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

5. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-08 --long-run-alg ValueIteration --store-mode Memory --unsafe

6. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-09 --long-run-alg ValueIteration --store-mode Memory --unsafe

7. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-10 --long-run-alg ValueIteration --store-mode Memory --unsafe

8. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --width 5e-11 --long-run-alg ValueIteration --store-mode Memory --unsafe

9. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAvgRepairCost --width 0.0005 --long-run-alg ValueIteration --store-mode Memory --unsafe

10. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-05 --long-run-alg ValueIteration --store-mode Memor y --unsafe

11. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-06 --long-run-alg ValueIteration --store-mode Memor y --unsafe

12. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-07 --long-run-alg ValueIteration --store-mode Memor y --unsafe

13. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-08 --long-run-alg ValueIteration --store-mode Memor y --unsafe

14. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-09 --long-run-alg ValueIteration --store-mode Memor y --unsafe

15. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-10 --long-run-alg ValueIteration --store-mode Memor y --unsafe

16. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAv gRepairCost --width 5e-11 --long-run-alg ValueIteration --store-mode Memor y --unsafe

17. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 0.0005 --long-run-al g ValueIteration --store-mode Memory --unsafe

18. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 5e-05 --long-run-alg ValueIteration --store-mode Memory --unsafe

19. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 5e-06 --long-run-alg ValueIteration --store-mode Memory --unsafe

20. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe

21. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 5e-08 --long-run-alg ValueIteration --store-mode Memory --unsafe

22. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_B OUND=0.0 --props MinAverageOperationCosts --width 5e-09 --long-run-alg ValueIteration --store-mode Memory --unsafe

23. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-10 --long-run-alg ValueIteration --store-mode Memory --unsafe
24. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --width 5e-11 --long-run-alg ValueIteration --store-mode Memory --unsafe
25. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 0.0005 --long-run-alg ValueIteration --store-mode Memory --unsafe
26. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-05 --long-run-alg ValueIteration --store-mode Memory --unsafe
27. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-06 --long-run-alg ValueIteration --store-mode Memory --unsafe
28. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-07 --long-run-alg ValueIteration --store-mode Memory --unsafe
29. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-08 --long-run-alg ValueIteration --store-mode Memory --unsafe
30. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-09 --long-run-alg ValueIteration --store-mode Memory --unsafe
31. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-10 --long-run-alg ValueIteration --store-mode Memory --unsafe
32. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3,C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --width 5e-11 --long-run-alg ValueIteration --store-mode Memory --unsafe
33. modest mcsta dpm.v2.modest -E N=4,C=4,TIME_BOUND=0.0 --props MinAvgOperationCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
34. modest mcsta ftwc.v3.modest -E N=64,TIME_BOUND=0.0 --props MinAvgRepairCost --long-run-alg LinearProgramming --store-mode Memory --unsafe
35. modest mcsta polling-system.v3.modest -E JOB_TYPES=3,C=4,TIME_BOUND=0.0 --props MinAverageOperationCosts --long-run-alg LinearProgramming --store-mode Memory --unsafe

36. modest mcsta reentrant-queues.v3.modest -E JOB_TYPES=3,C_LEFT=3, C_RIGHT=3,TIME_BOUND=0.0 --props SmaxBothQueuesFull --long-run-alg LinearProgramming --store-mode Memory --unsafe

B.2 Case Studies

Below we provide the models used for the experimental evaluation. We only attach here those models that differ from the respective model in the benchmark set [HKP+19].

B.2.1 Time-Bounded Reachability (exhaustive) and Long-Run Average Rewards

Figure B.1: dpm.v2.modest. Part 1.

```
const int N; // number of different types of tasks, maximum 10
const int C; // queue size
const real TIME_BOUND;

int(0..C) items1 = 0; int(0..C) items2 = 0; int(0..C) items3 = 0; int
    (0..C) items4 = 0; int(0..C) items5 = 0; int(0..C) items6 = 0; int
    (0..C) items7 = 0; int(0..C) items8 = 0; int(0..C) items9 = 0; int
    (0..C) items10 = 0;

binary action sleep, standby, idle;

property PminQueuesFull = Pmin(<> ((N < 1 || items1 == C) && (N < 2 ||
    items2 == C) && (N < 3 || items3 == C) && (N < 4 || items4 == C) &&
    (N < 5 || items5 == C) && (N < 6 || items6 == C) && (N < 7 || items7
     == C) && (N < 8 || items8 == C) && (N < 9 || items9 == C) && (N <
    10 || items10 == C)));
property PmaxQueuesFull = Pmax(<> ((N < 1 || items1 == C) && (N < 2 ||
    items2 == C) && (N < 3 || items3 == C) && (N < 4 || items4 == C) &&
    (N < 5 || items5 == C) && (N < 6 || items6 == C) && (N < 7 || items7
     == C) && (N < 8 || items8 == C) && (N < 9 || items9 == C) && (N <
    10 || items10 == C)));

property PminQueue1Full = Pmin(<> (items1 == C));
property PmaxQueue1Full = Pmax(<> (items1 == C));

property TminQueuesFull = Xmin(T, (N < 1 || items1 == C) && (N < 2 ||
    items2 == C) && (N < 3 || items3 == C) && (N < 4 || items4 == C) &&
    (N < 5 || items5 == C) && (N < 6 || items6 == C) && (N < 7 || items7
     == C) && (N < 8 || items8 == C) && (N < 9 || items9 == C) && (N <
    10 || items10 == C));

property PmaxQueuesFullBound = Pmax(<>[T<=TIME_BOUND] ((N < 1 || items1
     == C) && (N < 2 || items2 == C) && (N < 3 || items3 == C) && (N < 4
     || items4 == C) && (N < 5 || items5 == C) && (N < 6 || items6 == C)
     && (N < 7 || items7 == C) && (N < 8 || items8 == C) && (N < 9 ||
    items9 == C) && (N < 10 || items10 == C)));
```

Figure B.2: dpm.v2.modest. Part 2.

```
property SmaxQueuesFullT = Smax(T((N < 1 || items1 == C)?1:0));

property MinAvgOperationCost = Smin(T((pmode == SP_SLEEP)? 0.1 : ((
    pmode == SP_STANDBY) ? 0.5 : ((pmode == SP_IDLE) ? 1.0 : 5))));

const int SP_IDLE = 1; const int SP_SLEEP = 2; const int SP_STANDBY =
    3; const int SP_WORK = 4; int(1..4) pmode = SP_SLEEP;


process ServiceProvider()
{
   int(1..N) t;
   alt {
   :: when(pmode == SP_IDLE)
      alt {
      :: sleep?; rate(0.5) tau {= pmode = SP_SLEEP =}; ServiceProvider
          ()
      :: standby?; rate(1) tau {= pmode = SP_STANDBY =};
          ServiceProvider()
      :: alt {
         :: when(N >= 1 && items1 > 0) tau {= t = 1, items1-- =}
         :: when(N >= 2 && items2 > 0) tau {= t = 2, items2-- =}
         :: when(N >= 3 && items3 > 0) tau {= t = 3, items3-- =}
         :: when(N >= 4 && items4 > 0) tau {= t = 4, items4-- =}
         :: when(N >= 5 && items5 > 0) tau {= t = 5, items5-- =}
         :: when(N >= 6 && items6 > 0) tau {= t = 6, items6-- =}
         :: when(N >= 7 && items7 > 0) tau {= t = 7, items7-- =}
         :: when(N >= 8 && items8 > 0) tau {= t = 8, items8-- =}
         :: when(N >= 9 && items9 > 0) tau {= t = 9, items9-- =}
         :: when(N >= 10 && items10 > 0) tau {= t = 10, items10-- =}
         }; tau{= pmode = SP_WORK =};
         ServiceProvider()
      }
   :: when(pmode == SP_SLEEP) alt {
      :: idle?; rate(0.166) tau {= pmode = SP_IDLE =}; ServiceProvider
          ()
      :: standby?; rate(1.5) tau {= pmode = SP_STANDBY =};
          ServiceProvider()
      }
   :: when(pmode == SP_STANDBY) alt {
      :: idle?; rate(0.454) tau {= pmode = SP_IDLE =}; ServiceProvider
          ()
      :: sleep?; rate(1.5) tau {= pmode = SP_SLEEP =}; ServiceProvider
          ()
      }
   :: when(pmode == SP_WORK) rate(0.2 * t) tau {= pmode = SP_IDLE =};
       ServiceProvider()
   }
}
```

Figure B.3: dpm.v2.modest. Part 3.

```
process PowerManager()
{
   alt {
   :: standby!
   :: sleep!
   :: idle!
   };
   PowerManager()
}

process ServiceRequester()
{
   alt {
   :: when(N >= 1) rate(0.1 * 1 + 0.4) tau; when(items1 < C) {= items1
       ++ =}
   :: when(N >= 2) rate(0.1 * 2 + 0.4) tau; when(items2 < C) {= items2
       ++ =}
   :: when(N >= 3) rate(0.1 * 3 + 0.4) tau; when(items3 < C) {= items3
       ++ =}
   :: when(N >= 4) rate(0.1 * 4 + 0.4) tau; when(items4 < C) {= items4
       ++ =}
   :: when(N >= 5) rate(0.1 * 5 + 0.4) tau; when(items5 < C) {= items5
       ++ =}
   :: when(N >= 6) rate(0.1 * 6 + 0.4) tau; when(items6 < C) {= items6
       ++ =}
   :: when(N >= 7) rate(0.1 * 7 + 0.4) tau; when(items7 < C) {= items7
       ++ =}
   :: when(N >= 8) rate(0.1 * 8 + 0.4) tau; when(items8 < C) {= items8
       ++ =}
   :: when(N >= 9) rate(0.1 * 9 + 0.4) tau; when(items9 < C) {= items9
       ++ =}
   :: when(N >= 10) rate(0.1 * 10 + 0.4) tau; when(items10 < C) {=
       items10++ =}
   };
   ServiceRequester()
}

restrict { sleep!, sleep?, standby!, standby?, idle!, idle? }
{
   par {
   :: ServiceRequester()
   :: ServiceProvider()
   :: PowerManager()
   }
}
```

FIGURE B.4: polling-system.v3.modest.

```
const int JOB_TYPES;
const int C;
const real TIME_BOUND;
binary action deliver;
transient int(0..JOB_TYPES) item;
bool working;
int(0..C)[] size = [0, 0];

property PminBothFullIsOne = Pmin(<>(size[0]==C && size[1]==C))==1;
property TminBothFull = Xmin(T, size[0] == C && size[1] == C);
property TmaxBothFull = Xmax(T, size[0] == C && size[1] == C);
property PmaxBothFullBound = Pmax(<>[T<=TIME_BOUND] (size[0] == C && size[1] == C));
property SmaxBothFull = Smax(T((size[0]==C && size[1]==C) ? 1.0 : 0));
property MinAverageOperationCosts = Smin(T((working && size[0] < C && size[1] < C) ? 2.0 : ((working && (size[0] == C || size[1] == C)) ? 10.0 : 0)));

process Station(int id, int(0..JOB_TYPES)[] q) {
   alt {
   :: when(size[id - 1] < C) rate(0.2 * id + 0.1) tau;
      {= q[size[id - 1]] = (int)any(i, 1 <= i && i <= JOB_TYPES), size[id - 1]++ =}
   :: when(size[id - 1] > 0) deliver! {= item = q[0] =};
      alt {
      :: {= q = array(i, C, i < size[id - 1] - 1 ? q[i + 1] : 0), size[id - 1]-- =}
      :: {==}
      }
   };
   Station(id, q)
}

process Server() {
   int(1..JOB_TYPES) j;

   deliver? {= j = item, working = true =};
   rate(pow(2, j)) tau {= working = false =};
   Server()
}

restrict { deliver!, deliver? } {
   par {
   :: Station(1, array(i, C, 0))
   :: Station(2, array(i, C, 0))
   :: Server()
   }
}
```

Figure B.5: reentrant-queues.v3.modest. Part 1.

```
const int JOB_TYPES; // number of different job types
const int LAMBDA = 2; // job arrival rate
const real MU_LEFT = 1.5; // service rate of left desk
const int MU_RIGHT = 1; // service rate of right desk
const int C_LEFT; // capacity of left queue
const int C_RIGHT; // capacity of right queue
const real TIME_BOUND;

binary action service_left, service_right, reenter;

transient int(0..JOB_TYPES) j;

int sizeLeft = 0;
int sizeRight = 0;

bool processingLeft = false, processingRight = false;

property PminBothQueuesFullIsOne = Pmin(<> (sizeLeft == C_LEFT &&
    sizeRight == C_RIGHT)) == 1;
property TminBothQueuesFull = Xmin(T, sizeLeft == C_LEFT && sizeRight
    == C_RIGHT);
property TmaxBothQueuesFull = Xmax(T, sizeLeft == C_LEFT && sizeRight
    == C_RIGHT);
property PmaxBothQueuesFullBound = Pmax(<>[T<=TIME_BOUND] (sizeLeft ==
    C_LEFT && sizeRight == C_RIGHT));
property SmaxBothQueuesFull = Smax(T((sizeLeft == C_LEFT && sizeRight
    == C_RIGHT) ? 1.0 : 0));
property MinAverageOperationCosts = Smin(T((processingLeft && sizeLeft
    < C_LEFT && sizeRight < C_RIGHT) ? 0.5 * lj : ((processingRight &&
    sizeLeft < C_LEFT && sizeRight < C_RIGHT) ? 0.8 * rj  : (((
    processingLeft || processingRight) && (sizeLeft == C_LEFT ||
    sizeRight == C_RIGHT)) ? 10.0 * (lj + rj) : 0) )));
int(0..JOB_TYPES) lj = 0;

process LeftDesk()
{
   service_left? {= lj = j, processingLeft = true =};
   rate(MU_LEFT + 0.5 * (lj - 1)) tau {= processingLeft = false, lj = 0
       =};
   alt {
   :: LeftDesk()
   :: reenter! {= j = lj =}; LeftDesk()
   }
}
```

Figure B.6: reentrant-queues.v3.modest. Part 2.

```
int(0..JOB_TYPES) rj = 0;
process RightDesk()
{
   service_right? {= rj = j, processingRight = true =};
   rate(MU_RIGHT + 0.3 * (rj - 1)) tau {= processingRight = false, rj =
       0 =};
   RightDesk()
}

process Arrival()
{
   int(0..JOB_TYPES)[] ql = array(i, C_LEFT, 0);
   int(0..JOB_TYPES)[] qr = array(i, C_RIGHT, 0);

   do {
   :: when(sizeLeft < C_LEFT || sizeRight < C_RIGHT) rate(LAMBDA) tau;
       alt {
      :: when(sizeLeft < C_LEFT) {= ql[sizeLeft] = (int)any(i, 1 <= i
          && i <= JOB_TYPES), sizeLeft++ =}
      :: when(sizeRight < C_RIGHT) {= qr[sizeRight] = (int)any(i, 1 <=
          i && i <= JOB_TYPES), sizeRight++ =}
      }
   :: when(sizeRight < C_RIGHT) reenter? {= qr[sizeRight] = j,
       sizeRight++ =}
   :: when(sizeLeft > 0) service_left! {= j = ql[0], ql = array(i,
       C_LEFT, i < sizeLeft - 1 ? ql[i + 1] : 0), sizeLeft-- =}
   :: when(sizeRight > 0) service_right! {= j = qr[0], qr = array(i,
       C_RIGHT, i < sizeRight - 1 ? qr[i + 1] : 0), sizeRight-- =}
   }
}

restrict { service_left!, service_left?, service_right!, service_right
   ?, reenter!, reenter? }
{
   par {
   :: Arrival()
   :: LeftDesk()
   :: RightDesk()
   }
}
```

Figure B.7: ftwc.v3.modest. Part 1.

```
const int N; // number of workstations

const int LEFT = 0;
const int RIGHT = 1;
const real TIME_BOUND;

bool backboneDown = false;
int(0..N)[] workstations_up = [N,N]; // workstations_up[0] ~ left, workstations_up[1] ~ right
bool[] switches_down = [false,false]; // switches_down[0] ~ left, switches_down[1] ~ right

transient real repCost;

binary action startRepairBackbone, finishRepairBackbone;
binary action startRepairWorkstation, finishRepairWorkstation, startRepairLeftWorkstation, finishRepairLeftWorkstation, startRepairRightWorkstation, finishRepairRightWorkstation;
binary action startRepairSwitch, finishRepairSwitch, startRepairLeftSwitch, finishRepairLeftSwitch, startRepairRightSwitch, finishRepairRightSwitch;

property ReachMinIsOne = Pmin(<> ((workstations_up[LEFT] == 0 || switches_down[LEFT]) && (workstations_up[RIGHT] == 0 || switches_down[RIGHT]))) == 1;
property TimeMax = Xmax(T, (workstations_up[LEFT] == 0 || switches_down[LEFT]) && (workstations_up[RIGHT] == 0 || switches_down[RIGHT]));
property TimeMin = Xmin(T, (workstations_up[LEFT] == 0 || switches_down[LEFT]) && (workstations_up[RIGHT] == 0 || switches_down[RIGHT]));
property PmaxReachBound = Pmax(<>[T<=TIME_BOUND] ((workstations_up[LEFT] == 0 || switches_down[LEFT]) && (workstations_up[RIGHT] == 0 || switches_down[RIGHT])));
property SmaxReach = Smax(T((workstations_up[LEFT] == 0 || switches_down[LEFT]) && (workstations_up[RIGHT] == 0 || switches_down[RIGHT]) ? 1.0 : 0));
property MinAvgRepairCost = Smin(S(repCost));
```

FIGURE B.8: ftwc.v3.modest. Part 2.

```
process Backbone()
{
   alt {
   :: when(!backboneDown) rate(0.0002) {= backboneDown = true =}
   :: when(backboneDown) startRepairBackbone?;
      finishRepairBackbone? {= backboneDown = false =}
   };
   Backbone()
}

process Switch(int(0..1) id)
{
   alt {
   :: when(!switches_down[id]) rate(0.00025) tau {= switches_down[id] =
        true =}
   :: when(switches_down[id]) startRepairSwitch?;
      finishRepairSwitch? {= switches_down[id] = false =}
   };
   Switch(id)
}

process Workstation(int(0..1) id)
{
   alt {
   :: when(workstations_up[id] > 0) rate(workstations_up[id] / 500.0)
       tau {= workstations_up[id]-- =}; Workstation(id)
   :: when(workstations_up[id] < N) startRepairWorkstation?;
       Workstation(id)
   :: when(workstations_up[id] < N) finishRepairWorkstation? {=
       workstations_up[id]++ =}; Workstation(id)
   }
}

process RepairUnit()
{
   alt {
   :: startRepairBackbone! {= repCost = 5.5 =}; rate(0.125) tau;
       finishRepairBackbone!
   :: startRepairLeftWorkstation! {= repCost = 0.5 =}; rate(2.0) tau;
       finishRepairLeftWorkstation!
   :: startRepairRightWorkstation! {= repCost = 0.5 =}; rate(2.0) tau;
       finishRepairRightWorkstation!
   :: startRepairRightSwitch! {= repCost = 1.1 =}; rate(0.25) tau;
       finishRepairRightSwitch!
   :: startRepairLeftSwitch! {= repCost = 1.1 =}; rate(0.25) tau;
       finishRepairLeftSwitch!
   };
   RepairUnit()
}
```

Figure B.9: ftwc.v3.modest. Part 3.

```
restrict {
   startRepairLeftWorkstation?, startRepairLeftWorkstation!,
   startRepairRightWorkstation?, startRepairRightWorkstation!,
   finishRepairLeftWorkstation?, finishRepairLeftWorkstation!,
   finishRepairRightWorkstation?, finishRepairRightWorkstation!,
   startRepairLeftSwitch?, startRepairLeftSwitch!,
   startRepairRightSwitch?, startRepairRightSwitch!,
   finishRepairLeftSwitch?, finishRepairLeftSwitch!,
   finishRepairRightSwitch?, finishRepairRightSwitch!,
   startRepairBackbone?, startRepairBackbone!,
   finishRepairBackbone?, finishRepairBackbone!
} {
   par {
   :: Backbone()
   :: relabel { startRepairSwitch, finishRepairSwitch} by {
       startRepairLeftSwitch, finishRepairLeftSwitch }
      Switch(LEFT)
   :: relabel { startRepairSwitch, finishRepairSwitch} by {
       startRepairRightSwitch, finishRepairRightSwitch }
      Switch(RIGHT)
   :: relabel { startRepairWorkstation, finishRepairWorkstation} by {
       startRepairLeftWorkstation, finishRepairLeftWorkstation }
      Workstation(LEFT)
   :: relabel { startRepairWorkstation, finishRepairWorkstation} by {
       startRepairRightWorkstation, finishRepairRightWorkstation }
      Workstation(RIGHT)
   :: RepairUnit()
   }
}
```

B.2.2 Time-Bounded Reachability (partial)

FIGURE B.10: dpm-full.v2.modest. Part 1.

```
const int N; // number of different types of tasks, maximum 10
const int C; // queue size
const real TIME_BOUND;

int(0..C) items1 = C;
int(0..C) items2 = C;
int(0..C) items3 = C;
int(0..C) items4 = C;
int(0..C) items5 = C;
int(0..C) items6 = C;
int(0..C) items7 = C;
int(0..C) items8 = C;
int(0..C) items9 = C;
int(0..C) items10 = C;
int(0..C) items11 = C;
int(0..C) items12 = C;
int(0..C) items13 = C;
int(0..C) items14 = C;
int(0..C) items15 = C;
int(0..C) items16 = C;
int(0..C) items17 = C;
int(0..C) items18 = C;
int(0..C) items19 = C;
int(0..C) items20 = C;

binary action sleep, standby, idle;

property PmaxOneQueueThreeBound=Pmax(<>[T<=TIME_BOUND](ntasksh==3));

const int SP_IDLE = 1;
const int SP_SLEEP = 2;
const int SP_STANDBY = 3;
const int SP_WORK = 4;
int(1..4) pmode = SP_SLEEP;

int ntasksh = 0;
```

Figure B.11: dpm-full.v2.modest. Part 2.

```
process ServiceProvider() {
int(1..N) t;
alt {
:: when(pmode == SP_IDLE)
 alt {
 :: sleep?; rate(0.5) tau {= pmode = SP_SLEEP =}; ServiceProvider()
 :: standby?; rate(1) tau {= pmode = SP_STANDBY =}; ServiceProvider()
 :: alt {
  :: when(N >= 1 && items1 > 0) tau {= t = 1, items1-- =}
  :: when(N >= 2 && items2 > 0) tau {= t = 2, items2-- =}
  :: when(N >= 3 && items3 > 0) tau {= t = 3, items3-- =}
  :: when(N >= 4 && items4 > 0) tau {= t = 4, items4-- =}
  :: when(N >= 5 && items5 > 0) tau {= t = 5, items5-- =}
  :: when(N >= 6 && items6 > 0) tau {= t = 6, items6-- =}
  :: when(N >= 7 && items7 > 0) tau {= t = 7, items7-- =}
  :: when(N >= 8 && items8 > 0) tau {= t = 8, items8-- =}
  :: when(N >= 9 && items9 > 0) tau {= t = 9, items9-- =}
  :: when(N >= 10 && items10 > 0) tau {= t = 10, items10-- =}
  :: when(N >= 11 && items11 > 0) tau {= t = 11, items11-- =}
  :: when(N >= 12 && items12 > 0) tau {= t = 12, items12-- =}
  :: when(N >= 13 && items13 > 0) tau {= t = 13, items13-- =}
  :: when(N >= 14 && items14 > 0) tau {= t = 14, items14-- =}
  :: when(N >= 15 && items15 > 0) tau {= t = 15, items15-- =}
  :: when(N >= 16 && items16 > 0) tau {= t = 16, items16-- =}
  :: when(N >= 17 && items17 > 0) tau {= t = 17, items17-- =}
  :: when(N >= 18 && items18 > 0) tau {= t = 18, items18-- =}
  :: when(N >= 19 && items19 > 0) tau {= t = 19, items19-- =}
  :: when(N >= 20 && items20 > 0) tau {= t = 20, items20--, ntasksh++
     =}
  }; tau{= pmode = SP_WORK =}; ServiceProvider()
 }
 :: when(pmode == SP_SLEEP) alt {
  :: idle?; rate(0.166) tau {= pmode = SP_IDLE =}; ServiceProvider()
  :: standby?; rate(1.5) tau {= pmode = SP_STANDBY =}; ServiceProvider
     ()
  }
 :: when(pmode == SP_STANDBY) alt {
  :: idle?; rate(0.454) tau {= pmode = SP_IDLE =}; ServiceProvider()
  :: sleep?; rate(1.5) tau {= pmode = SP_SLEEP =}; ServiceProvider()
  }
 :: when(pmode == SP_WORK) rate(0.2 * t) tau {= pmode = SP_IDLE =};
    ServiceProvider()
}
}
```

Figure B.12: dpm-full.v2.modest. Part 3.

```
process PowerManager()
{
   alt {
   :: standby!
   :: sleep!
   :: idle!
   };
   PowerManager()
}

process ServiceRequester()
{
alt {
::when(N >= 1) rate(0.1*1+0.4) tau; when(items1 < C) {= items1++ =}
::when(N >= 2) rate(0.1*2+0.4) tau; when(items2 < C) {= items2++ =}
::when(N >= 3) rate(0.1*3+0.4) tau; when(items3 < C) {= items3++ =}
::when(N >= 4) rate(0.1*4+0.4) tau; when(items4 < C) {= items4++ =}
::when(N >= 5) rate(0.1*5+0.4) tau; when(items5 < C) {= items5++ =}
::when(N >= 6) rate(0.1*6+0.4) tau; when(items6 < C) {= items6++ =}
::when(N >= 7) rate(0.1*7+0.4) tau; when(items7 < C) {= items7++ =}
::when(N >= 8) rate(0.1*8+0.4) tau; when(items8 < C) {= items8++ =}
::when(N >= 9) rate(0.1*9+0.4) tau; when(items9 < C) {= items9++ =}
::when(N >= 10) rate(0.1*10+0.4) tau; when(items10 < C) {= items10++ =}
::when(N >= 11) rate(0.1*11+0.4) tau; when(items11 < C) {= items11++ =}
::when(N >= 12) rate(0.1*12+0.4) tau; when(items12 < C) {= items12++ =}
::when(N >= 13) rate(0.1*13+0.4) tau; when(items13 < C) {= items13++ =}
::when(N >= 14) rate(0.1*14+0.4) tau; when(items14 < C) {= items14++ =}
::when(N >= 15) rate(0.1*15+0.4) tau; when(items15 < C) {= items15++ =}
::when(N >= 16) rate(0.1*16+0.4) tau; when(items16 < C) {= items16++ =}
::when(N >= 17) rate(0.1*17+0.4) tau; when(items17 < C) {= items17++ =}
::when(N >= 18) rate(0.1*18+0.4) tau; when(items18 < C) {= items18++ =}
::when(N >= 19) rate(0.1*19+0.4) tau; when(items19 < C) {= items19++ =}
::when(N >= 20) rate(0.1*20+0.4) tau; when(items20 < C) {= items20++ =}
}; ServiceRequester()
}

restrict { sleep!, sleep?, standby!, standby?, idle!, idle? }
{
   par {
   :: ServiceRequester()
   :: ServiceProvider()
   :: PowerManager()
   }
}
```

Figure B.13: polling-system-k.v3.modest.

```
const int JOB_TYPES;
const int C;
const real TIME_BOUND;

binary action deliver;
bool working;
int(1..5) stationid;
int(0..C)[] size = [C, C, C, C, C];
real[] rates = [0.1, 0.1, 0.1, 0.1, 5];

property PmaxBothFullBound = Pmax(<>[T<=TIME_BOUND] (size[0] == C &&
    size[1] == C));
property PmaxOneFullBound = Pmax(<>[T<=TIME_BOUND] (size[4] == 0));

process Station(int id)
{
   alt {
   :: when(size[id - 1] < C) rate(0.2 * id + 0.1) tau; {= size[id -
       1]++ =}
   :: when(size[id - 1] > 0) deliver! {= size[id - 1]--, stationid = id
        =}
   };
   Station(id)
}

process Server()
{
   deliver? {= working = true =};
   rate(rates[stationid - 1]) tau {= working = false =};
   Server()
}

restrict { deliver!, deliver? }
{
   par {
   :: Station(1)
   :: Station(2)
   :: Station(3)
   :: Station(4)
   :: Station(5)
   :: Server()
   }
}
```